der Bundeswehr
Universität München

Towards a Network-based Approach for Smartphone Security

Ingo Bente, M.Sc.

Vollständiger Abdruck der von der Fakultät für Informatik der Universität der Bundeswehr München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Promotionsausschuss:

Vorsitzender:	Prof. Dr. Peter Hertling
1. Berichterstatter:	Prof. Dr. Gabi Dreo Rodosek
2. Berichterstatter:	Prof. Dr. Udo Helmbrecht
1. Prüfer:	Prof. Dr.-Ing. Mark Minas
2. Prüfer:	Prof. Dr.-Ing. Helmut Mayer

Tag der Prüfung: 25.07.2013

Neubiberg, August 2013

Abstract

Smartphones have become an important utility that affects many aspects of our daily life. Due to their large dissemination and the tasks that are performed with them, they have also become a valuable target for criminals. Their specific capabilities and the way they are used introduce new threats in terms of information security.

The research field of smartphone security has gained a lot of momentum in the past eight years. Approaches that have been presented so far focus on investigating design flaws of smartphone operating systems as well as their potential misuse by an adversary. Countermeasures are often realized based upon extensions made to the operating system itself, following a host-based design approach. However, there is a lack of network-based mechanisms that allow a secure integration of smartphones into existing IT infrastructures. This topic is especially relevant for companies whose employees use smartphones for business tasks.

This thesis presents a novel, network-based approach for smartphone security called **CADS: Context-related Signature and Anomaly Detection for Smartphones**. It allows to determine the security status of smartphones by analyzing three aspects: (1) their current configuration in terms of installed software and available hardware, (2) their behavior and (3) the context they are currently used in. Depending on the determined security status, enforcement actions can be defined in order to allow or to deny access to services provided by the respective IT infrastructure. The approach is based upon the distributed collection and central analysis of data about smartphones. In contrast to other approaches, it explicitly supports to leverage existing security services both for analysis and enforcement purposes.

A proof of concept is implemented based upon the IF-MAP protocol for network security and the Google Android platform. An evaluation verifies (1) that the CADS approach is able to detect so-called sensor sniffing attacks and (2) that reactions can be triggered based on detection results to counter ongoing attacks. Furthermore, it is demonstrated that the functionality of an existing, host-based approach that relies on modifications of the Android smartphone platform can be mimicked by the CADS approach. The advantage of CADS is that it does not need any modifications of the Android platform itself.

Zusammenfassung

Smartphones sind zu einem wichtigen Werkzeug geworden, welches viele Aspekte in unserem täglichen Leben betrifft. Da Smartphones auch für sensitive Aufgaben verwendet werden, sind sie ebenfalls zu einem begehrten Ziel für Kriminelle geworden. Durch ihre Funktionsvielfalt entstehen neue Bedrohungen im Bereich der Informationssicherheit.

In den letzten acht Jahren wurden daher zahlreiche Forschungsarbeiten veröffentlicht, die sich mit dem Thema Smartphone-Sicherheit befassen. Viele der aktuellen Arbeiten basieren darauf, Schwächen in der auf Smartphones eingesetzten Software zu finden, um diese wiederum für Angriffe auszunutzen. Gegenmaßnahmen werden oft durch Host-basierte Erweiterungen realisiert, in der Regel durch Modifikationen an dem Betriebssystem des Smartphones selbst. Im Gegensatz dazu existieren nur wenige Netzwerk-basierte Ansätze, die sich mit der sicheren Anbindung von Smartphones an vorhandene IT-Infrastrukturen befassen. Dabei ist das Thema insbesondere dann relevant, wenn Smartphones innerhalb eines Unternehmens eingesetzt werden.

In dieser Arbeit wird ein neuer, Netzwerk-basierter Ansatz zur sicheren Integration von Smartphones in vorhandene IT-Infrastrukturen vorgestellt: **CADS: Context-related Signature and Anomaly Detection for Smartphones**. CADS basiert auf einer verteilten Sammlung und zentralen Auswertung von Daten, mit denen sich der Sicherheitsstatus eines Smartphones ermitteln lässt. Zur Ermittlung dieses Sicherheitsstatus werden drei Eigenschaften von Smartphones betrachtet: (1) ihre aktuelle Konfiguration im Sinne von installierter Software und verwendeter Hardware, (2) ihr Verhalten und (3) der Kontext, in dem Smartphones verwendet werden. Abhängig von dem ermittelten Sicherheitsstatus können Reaktionen definiert werden, um den Zugriff auf bestimmte Bereiche der IT-Infrastruktur, in dem das Smartphone verwendet wird, zuzulassen oder zu unterbinden. Im Gegensatz zu existierenden Ansätzen unterstützt CADS explizit die Integration vorhandener Sicherheitslösungen.

Im Rahmen der Arbeit wird ein Prototyp des CADS Ansatzes vorgestellt, der basierend auf dem IF-MAP Protokoll und der Google Android Smartphone-Plattform entwickelt worden ist. Eine Evaluation zeigt, (1) dass mit dem CADS Ansatz so genannte Sensor Sniffing Angriffe erfolgreich erkannt werden können und (2) dass Reaktionen ausgelöst werden können, um stattfindende Angriffe zu unterbinden. Außerdem wird demonstriert, wie mit CADS die Funktionalität eines existierenden, Host-basierten Ansatzes für Smartphone-Sicherheit nachgebildet werden kann. Der Vorteil bei der Verwendung von CADS ist, dass auf Modifikationen der Smartphone-Plattform selbst verzichtet werden kann.

Acknowledgments

First of all, I want to thank the members of the examination board from the Universität der Bundeswehr in Munich. My warmest thanks go to my first examiner Prof. Dr. Gabi Dreo Rodosek. Her comments during, frankly speaking, endless Skype calls helped me to write down my thesis. Another big thank you goes to Prof. Dr. Udo Helmbrecht for being willing to jump in as second examiner on short notice.

I owe my deepest gratitude to Prof. Dr. Josef von Helden from the University of Applied Sciences and Arts in Hannover. He was responsible for introducing me to my supervisors from the Universität der Bundeswehr in Munich. To work with him has been a truly satisfying experience.

I would also like to thank my colleague Jörg Vieweg. He started the adventure to pursue a Ph.D. at the same time I did. The numerous insightful, stressful and sometimes ridiculously funny discussions kept me going on. Furthermore, thanks to all the other members of the Trust@FHH research group.

Finally, I would also like to thank my family. During the time of writing this thesis, they helped me to stay focused. From the members of my family, my partner Gunda probably had the most hardest time. Thank you for being patient and helping me, no matter what. I also thank my sister Sonja who is a tremendous role model for an excellent academic career. Thank you Alex for the entertaining conversations that had luckily nothing to do with my thesis (most of the time). And of course I want to thank my parents for always believing in the things that I have started, aborted and finished.

Contents

1 Introduction

"The beginning is the most important part of the work."

(Plato)

Contents

1.1 Motivation

Today, the importance of secure and reliable information technology (IT) infrastructures is obvious. Almost any domain of our everyday life depends on IT services. Social networking, online banking and e-commerce are some examples. The Internet is the crucial backbone that enables the necessary communication between the participating computer systems. Nowadays, news about attacks on IT services and infrastructures are common. Some recent targets of cyber attacks include the Sony PlayStation Network (PSN) [1] and newspapers like the New York Times [2]. The relevance of information security has become even more severe, as coordinated online attacks between different countries can be considered as an act of war [3].

Over the past years, the dissemination of a new type of computing device has changed the IT landscape: the so-called smartphones. Besides ordinary telephony services known from mobile phones, one key aspect of these devices is the support of Internet based communication. There is a trend already observed in 2008 that in developed countries mobile handsets are going to outnumber the country's population [4]. According to the

International Data Corporation (IDC), more than 494 million smartphone devices have been shipped alone in 2011 [5]. A recent study conducted by Google in partnership with Ipsos OTX MediaCT proves the significance of smartphones, as 89% of the surveyed people state that they use their phone for their daily life activities, such as shopping or performing searches for local information [6].

Today's smartphones provide a wide range of features such as

1. mobile access to the Internet,

2. the support of sensors to capture data about the physical environment the smartphone is used in (such as video, audio and position data) and

3. the support of third-party applications (apps). Those apps allow users to customize their smartphones according to their personal needs.

Companies can use smartphones for business tasks [7]. Since smartphones support mobile access to the Internet, employees can react faster on emails and the scheduling of appointments. Although the usefulness of smartphones in corporate environments is obvious, their impact in terms of information security is an open question. Smartphones have specific characteristics that change the attack surface of the environment they are used in. For example, they provide a rich set of built-in sensors in order to obtain the current location, record audio via the microphone or capture pictures via the camera. Furthermore, the way smartphones are extended by third-party apps is distinct from other computing devices. For example, on a laptop running a standard operating system such as Linux, Windows or OS X, usually a single web browser is used in order to view arbitrary websites. On smartphones, this situation is different. A general purpose web browser is also available. However, it is common that websites provide third-party apps in order to deliver their content in a way that is more suitable for smartphones, both in terms of usability and functionality. Examples include social networks like Facebook[1] or Google+[2] as well as news portals. That is, instead of using just a single, general purpose app for surfing the Internet, smartphones usually have additional apps installed in order to access specific websites or services.

[1] http://www.facebook.com
[2] http://plus.google.com

Companies have to deal with smartphones as new type of devices whether they choose to use them for business tasks or not. There are basically three options for a company to address and regulate the use of smartphones:

1. Allow smartphones for business tasks and provide corporate owned devices to the employees.

2. Allow smartphones for business tasks but allow employees to use their privately owned devices. This is commonly referred to as Bring Your Own Device (BYOD) policy.

3. Forbid the use of smartphones for business tasks. In environments where highly sensitive information is available (such as a company's research and development department), this might lead to policies that forbid employees to even bring their privately owned devices with them.

If smartphones are allowed to be used, the challenge is to integrate them into the company's existing IT infrastructure in a secure way. What "secure" actually means in this context is discussed in the remainder of this thesis.

No matter whether smartphones are used for business tasks or not, sensitive data is processed by them. Common examples include credit card information that needs to be entered by the user to buy goods such as new apps. In addition, a user has to enter his credentials to access online services like his email account. As a consequence, smartphones have become a valuable target for attackers. Third-party apps turn out to be one major threat for modern smartphones. Malware as it is known from other computing platforms has become a real threat for smartphones as well. The research community has proposed several proof of concept malware prototypes for various platforms [8, 9, 10, 4, 11]. Furthermore, malware has also hit the "real world", spreading itself by leveraging the respective platform's online market stores [12, 13]. The malicious functionality is diverse, ranging from simple denial of service attacks (for example by draining the smartphone's battery) to more sophisticated attacks that aim to steal sensitive data by leveraging the smartphone's built-in sensors.

Approaches that aim to improve the security of smartphone platforms have been proposed as well (such as [14, 15, 16, 17, 18]). Most of them suggest host-based security extensions that are able to prevent certain types of attacks or to detect the presence of

malicious apps. They tackle the field of smartphone security from the perspective of the smartphones themselves, focusing on improving the security of the smartphone platforms that are used. However, there are only few network-based approaches that engage the field of smartphone security from the perspective of the IT infrastructures the smartphones are used in. One reason for favoring host-based approaches is that they allow to add security mechanisms at several layers of the respective smartphone platform. However, it also means that unmodified versions of the smartphone platforms cannot benefit from this type of host-based extensions. For example, approaches like BizzTrust[3] modify both the Linux kernel and the middleware of the Android smartphone platform. Although this allows to easily add security related functionality like better isolation of apps, it comes at the cost of rendering the use of unmodified Android versions infeasible. Especially when a company allows to use privately owned devices, those approaches are not suitable. Instead, a more lightweight approach that at most requires to install additional third-party apps but omits the need for modifying the smartphone platform itself is often desired.

The thesis proposes a novel, network-based approach for the secure integration of smartphones into existing IT infrastructures. It allows to determine the security status of smartphones by analyzing three aspects: (1) their current configuration in terms of installed software and available hardware, (2) their behavior and (3) the context they are currently used in. Depending on the determined security status, enforcement actions can be defined in order to allow or to deny access to services provided by the IT infrastructure. The approach is based upon the distributed collection and central analysis of data about smartphones. In contrast to other approaches, it explicitly supports to leverage existing security services both for analysis and enforcement purposes.

1.2 Research Questions

The following research questions are addressed in the remainder of this thesis:

1. What approach is appropriate to enable a secure integration of smartphones into existing IT infrastructures?

   The field of smartphone security is dominated by host-based approaches that add specific security extensions to current smartphone platforms. However, an approach

[3] http://www.bizztrust.de/

that addresses the problem of securely integrating smartphones into existing IT infrastructures from the infrastructure's perspective is still missing. By answering this question a set of requirements is derived that must be fulfilled to enable a secure integration of smartphones into existing IT infrastructures.

2. What data should be collected and how should the collected data be analyzed in order to determine the security status of smartphones?

   Smartphones can be customized by users via third-party apps, have various built-in sensors and support Internet-based communication. As a consequence, smartphones can be in a status that is considered to be insecure from the perspective of the IT infrastructure they are used in. An open research question is what data is appropriate for determining the security status of smartphones. Furthermore, to know what data is needed is not sufficient. It is also necessary to use appropriate methods in order to analyze the data. Both aspects are addressed by answering this question.

3. How can the context of smartphones be obtained and used in order to contribute to their secure integration into existing IT infrastructures?

   Smartphones accompany their users throughout the day. As a consequence, they are used in a number of different contexts. A context is defined as *"the situation in which something happens and that helps you to understand it"* [19]. One goal of this thesis is to investigate how the context, a smartphone is currently used in, can be captured. Furthermore, it is investigated to what extent information about a smartphone's context can contribute to its secure integration into existing IT infrastructures.

The research questions stated above are reflected in Chapter 7, with respect to the results that were achieved within this thesis.

1.3 Outline of the Thesis

The remainder of the thesis is organized as follows (cf. Figure 1.1). Chapter 2 presents a set of scenarios that form the motivating background for the remainder of this thesis. They are formulated with respect to the reference IT infrastructure of a company. Based on

the identified scenarios, requirements are derived that must be fulfilled by the developed approach. These requirements especially define the necessary functionalities for a secure integration of smartphones into existing IT infrastructures. Thus, Chapter 2 addresses research question 1.

An analysis of the state of the art in the field of smartphone security is conducted in Chapter 3. The analysis covers both standard security mechanisms that are supported by today's smartphone platforms as well as approaches that have been proposed by the research community. Furthermore, the IF-MAP protocol [20] for network security is introduced. It is used in the remainder of this thesis to implement a prototype of the developed approach. Based on the literature review, an assessment is done with respect to the requirements that were derived in Chapter 2. The assessment reveals that existing approaches are not able to fulfill the stated requirements in a sufficient manner.

A novel, network-based approach for smartphone security is developed and presented in Chapter 4. It is referred to as **CADS: Context-related Signature and Anomaly Detection for Smartphones** and represents the main contribution of this thesis. The approach is composed of four parts: (1) a conceptual model that defines its main building blocks and the relationships between them, (2) an architecture that defines logical roles that must be fulfilled by components in order to support the distributed collection and central analysis of data about smartphones, (3) a correlation model that defines how the collected data can be analyzed and (4) a process to create so-called domain-specific instances. Thus, this chapter addresses the research questions 2 and 3.

A prototype of the CADS approach is presented in Chapter 5. It is implemented based on the previously mentioned IF-MAP protocol. The chapter discusses how the CADS approach can be realized by leveraging IF-MAP and presents the software components that have been developed for that purpose.

An evaluation of the CADS approach is presented in Chapter 6. It investigates to what extent the CADS approach is able to ensure the secure integration of smartphones into existing IT infrastructures.

Finally, conclusions are drawn in Chapter 7. The results of the thesis are discussed regarding the research questions that were stated in Section 1.2. Furthermore, directions for future work are given.

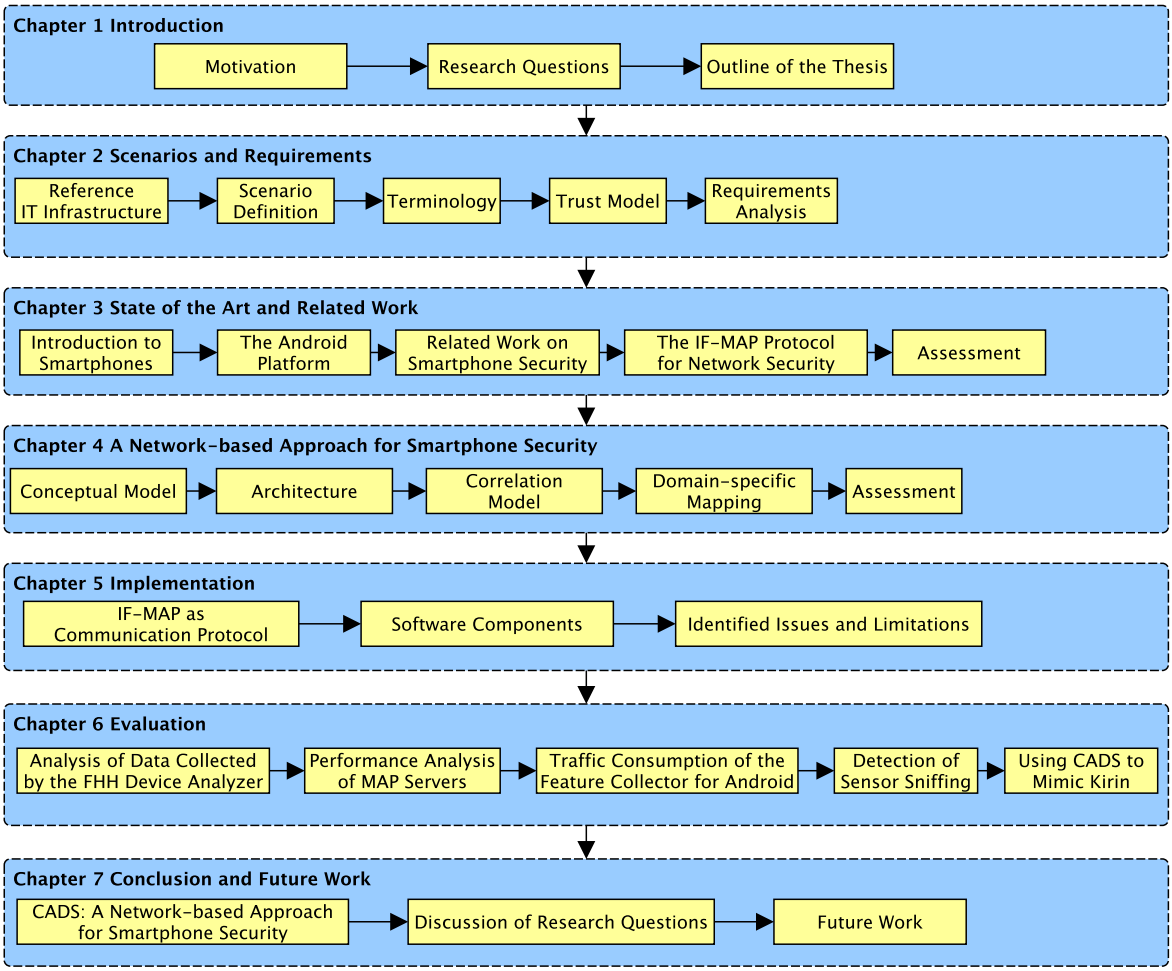

Figure 1.1: Outline of the thesis. Blue boxes represent chapters, yellow boxes represent sections. Arrows indicate the sequence of chapters, respectively the sequence of sections within a single chapter.

2 Scenarios and Requirements

"Do, or do not. There is no 'try'."

(Yoda)

Contents

This chapter introduces the scenarios that form the motivating background for the remainder of this thesis. In the first place, a reference IT infrastructure is defined. After that, four scenarios are presented. These scenarios were developed as part of the analysis phase within the ESUKOM research project [21]. They depict use cases that the

participating companies agreed upon to be relevant in terms of the secure integration of smartphones into their IT infrastructures. Based on the scenarios, a terminology that is used within the remainder of this thesis is defined. Furthermore, assumptions regarding the trustworthiness of some components are described within a trust model.

Given the scenarios, the terminology and the trust model, a list of requirements is derived that define the functionality which is needed for a secure integration of smartphones into existing IT infrastructures. The requirements will be used in the remainder of this thesis to assess related work that has been conducted in the field of smartphone security and to assess the novel, network-based approach for smartphone security that is developed within this thesis.

2.1 Reference IT Infrastructure

Today, virtually any company uses some sort of IT infrastructure to support their business activities. The details and the complexity of such infrastructures vary greatly, depending on factors like the company size and its main business model. For example, a large company with departments distributed across Europe is likely to have a more complex infrastructure compared to a small to medium sized company with just one local department. Furthermore, a company that provides general consulting services will have different IT systems in use compared to a company that provides hosting services for their customers, or even offer their own developed services to the public like social networks or cloud service providers.

Although there is a certain degree of diversity in terms of current IT infrastructures, there are also lots of common aspects that are shared across them. Therefore standards like the IT-Grundschutz[1] methodology [23, 24, 25, 26] of the Federal Office for Information Security (BSI) can be used effectively.

In order to limit the scope of the following scenario definitions and to provide a set of common terms, a reference IT infrastructure is defined. It models a generic IT infrastructure that explicitly addresses the integration of smartphones. It basically defines a network topology, mentions smartphones as special type of endpoints, and highlights a set of services that are common in most IT infrastructures. The description aims to be

[1] IT-Grundschutz is also referred to as IT Baseline Protection in some BSI publications [22].

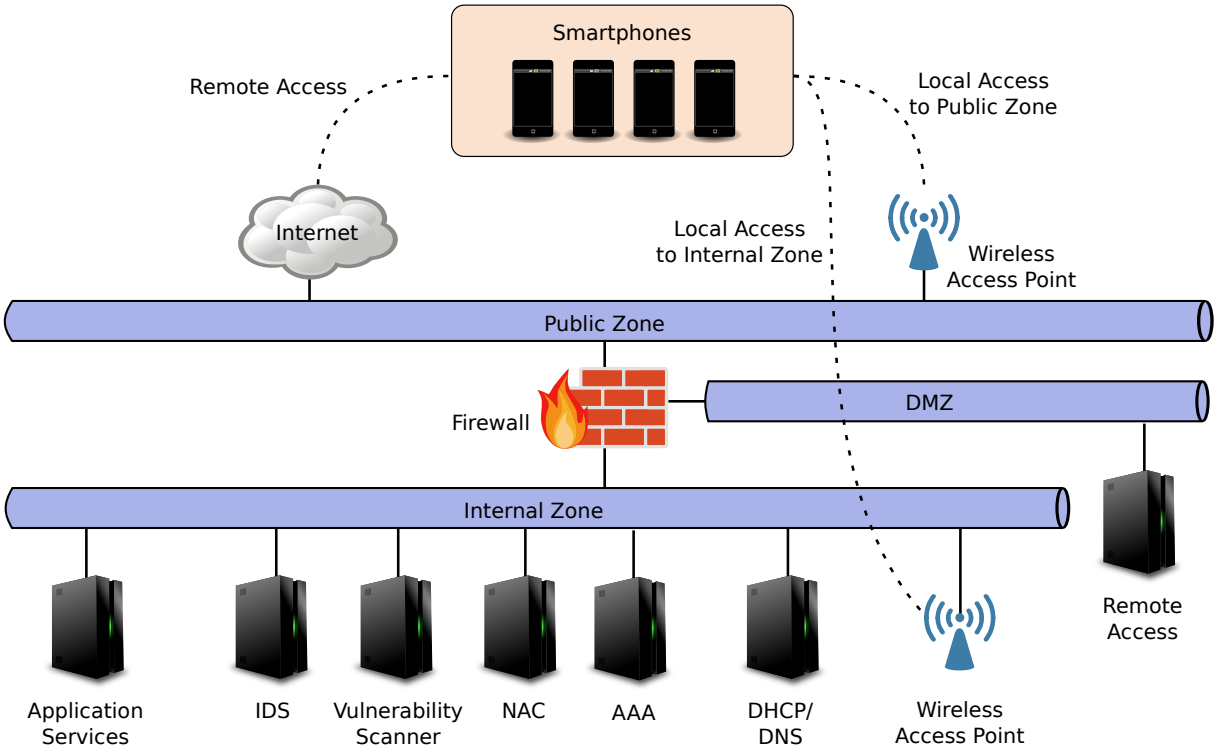

Figure 2.1: Reference IT Infrastructure. Icons taken from Openclipart [27].

independent of concrete technologies and rather focuses on the purpose that the deployed services fulfill. The reference IT infrastructure is depicted in Figure 2.1. The basic network topology, the notion of endpoints and the set of available services are detailed in the following.

2.1.1 Network Topology

The reference IT infrastructure defines a very basic network topology. It follows the well-known approach of establishing a Demilitarized Zone (DMZ) with firewalls in order to separate the internal network from the Internet. The approach is widely adopted and can be seen as baseline in order to manage and secure access from the Internet to services in the internal network and vice versa.

In real IT infrastructures, the network topology is likely to be more complex. Especially the internal network is often partitioned into further subnetworks, either for security purposes (like isolation of productive from testing environments) or to provide quality of

service aspects (for example in order to handle voice over IP with higher priority). Isolation between these subnetworks can be enforced by different technical means, including simple routing, dedicated firewall systems or by establishing Virtual Local Area Networks (VLANs) according to IEEE 802.1Q [28].

2.1.2 Endpoints

Endpoints are computing devices that are used by employees within the IT infrastructure of a company. They make use of available services, but do not provide services on their own. Examples include classical desktop computers, laptops, tablets and smartphones. The latter ones are of special interest for the remainder of this thesis. Although some concepts can be adopted for other types of endpoints, the approach focuses on those. Thus, only smartphones are depicted in Figure 2.1.

Basically, smartphones can access the IT infrastructure in two different ways. The first one is to establish access locally via a Wireless Local Area Network (WLAN). The WLAN is provided by dedicated infrastructure components referred to as Wireless Access Points (WAPs). Depending on the configuration of the WAP and its physical location, the smartphone has either access to the public zone or to the internal zone. The wireless communication is often based on the IEEE 802.11 set of standards. The second way is to access the IT infrastructure remotely over the Internet. This typically involves a remote access technology like a Virtual Private Network (VPN).

It should be noted that the use of VPNs is not limited to enabling remote access. When smartphones access the IT infrastructure locally via the WLAN, the use of additional VPNs can be reasonable as well. A common use case for the combination of both access technologies is to establish isolated environments within a company network via dedicated VPNs.

2.1.3 Services

An IT infrastructure is composed of a set of services. The purpose and the functionality of such services varies. For example, common services that are deployed in virtually any IT infrastructure are the dynamic assignment of IP addresses to endpoints via the Dynamic Host Configuration Protocol (DHCP) or the mapping between domain names and IP addresses via the Domain Name System (DNS).

Services that are relevant in terms of information security are of special interest for the remainder of this thesis. They are referred to as security services. Most companies that plan to integrate smartphones into their IT infrastructure in a secure way will already have existing security services deployed. However, those services will have little or no specific functionality to address smartphone specific threats. The approach that is developed within this thesis shall allow to leverage these existing security services in order to achieve a secure integration of smartphones. In the following, a list of relevant security services is given (cf. Figure 2.1):

AAA AAA refers to authentication, authorization and accounting. Services that provide AAA functionality are used to verify the user's identity (authentication), grant access to other services based on the user's identity (authorization) and also track the consumption of services by the user (accounting). Today, two protocols are commonly used in order to implement AAA services: Remote Authentication Dial In User Service (RADIUS) [29] and its successor Diameter [30].

Flow Controller The term flow controller refers to any service that can actively modify or block the network traffic. The notion of a flow controller is derived from the Trusted Network Connect (TNC) specifications published by the Trusted Computing Group (TCG). Examples include packet filters that operate on the network and transport layer according to the Open Systems Interconnection (OSI) model [31]. In terms of the reference IT infrastructure, there are two types of flow controllers. First, a firewall is used in order to establish the DMZ. Note that additional firewalls can be added to protect services that are of special interest or that operate on sensitive data. The second type of flow controllers are WAPs that support IEEE 802.1X [32].

IDS An Intrusion Detection System (IDS) monitors the behavior of a computer system or the traffic within a network in order to detect malicious activities. According to the National Institute of Standards and Technology (NIST), an IDS *"... is software that automates the intrusion detection process."* [33]. One popular example for a network-based open source IDS is Snort[2].

Remote Access Companies often require that employees get remote access to their IT infrastructure, either completely or to a subset of the provided services. If the ser-

[2] http://www.snort.org/

vices themselves should not be exposed to the Internet, a VPN is usually deployed. Various protocols exist in order to implement a VPN. Among others the Internet Protocol Security (IPsec) protocol suite is popular to realize a VPN at the OSI layer 3. The systems that provide services for remote access are typically deployed within the DMZ.

NAC The term Network Access Control (NAC) is not precisely defined. The term is commonly used in order to refer to protocols, components and software that grant access to the local network based on the users identity and the software configuration of the endpoint. Different vendors provide products that implement NAC functionality, including Microsoft[3] [34] and Cisco[4] [35]. Furthermore, the TCG has published the TNC standards [36] in order to provide an open, interoperable and vendor-neutral framework for implementing NAC services.

Vulnerability Scanner Vulnerability scanners are used in order to search computer systems, networks and applications for known vulnerabilities. The notion of a vulnerability in terms of information security is not globally defined. RFC 4949 defines it as follows: *"A flaw or weakness in a system's design, implementation, or operation and management that could be exploited to violate the system's security policy."* [37]. Examples for network vulnerability scanners are Nessus[5] and OpenVAS[6].

The set of described security services aims to form a baseline for the following definition of scenarios. Furthermore, the network-based approach for smartphone security that is developed in the remainder of this thesis will leverage some of these services for the secure integration of smartphones into existing IT infrastructures.

2.2 Scenario Definition

Four scenarios are described in the following. They form the motivating background for the remainder of this thesis. As already mentioned, these scenarios were developed as part of the analysis phase within the ESUKOM research project [21]. In terms of smartphone

[3] Instead of NAC, the term Network Access Protection (NAP) is used by Microsoft.
[4] Cisco refers to the term NAC as Network Admission Control.
[5] http://www.tenable.com/
[6] http://www.openvas.org/

security, the first three scenarios focus on detection tasks, whereas the fourth scenario addresses the need for appropriate reaction capabilities.

2.2.1 Scenario I: Smartphone Visibility

From the perspective of a companies' IT infrastructure, smartphones are quite similar to other endpoints. They can connect to WAPs, log in to the corporate VPN and access virtually any service that supports the IP communication protocol. Assuming that a smartphone is connected to a wireless network based on the IEEE 802.11 set of standards, they send the same datagrams composed of source and destination Media Access Control (MAC) addresses, source and destination IP addresses and optionally Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) port numbers plus an arbitrary payload as other endpoints connected to the same wireless network. That is, from a network point of view, smartphones cannot directly by differentiated from other endpoints.

However, to know if a certain request originates from a smartphone or not can be beneficial. For example, it allows smartphone specific provisioning of services. A service that presents its result in a graphical way might adjust its layout depending on the type of device that sent the request. This is already done frequently by websites that provide special mobile versions for smartphones (for example by using a subdomain like `http://m.example.com` for the mobile version of `http://www.example.com`). Furthermore, a company might want to deny access to a service from a smartphone due to security concerns.

Thus, the objective of this scenario is to enable application services within an infrastructure to determine whether a certain request was issued by a smartphone or not. This will primarily require to express the fact that a certain device is a smartphone (binary yes/no) in an efficient way and to propagate this knowledge within the infrastructure of the company.

Example Use Case A company provides employees with smartphones in order to process business emails and to manage their time schedule. However, it is not allowed to access certain application services that process more sensitive data with them (like project specific deliverables, version control systems or servers that process personal data). In this

case, the developed solution needs to be able to block access to those critical services and the sensitive data when the corresponding requests originate from a smartphone. The definition whether a service is too critical to be processed by a smartphone or not is domain-specific.

2.2.2 Scenario II: Context-related Service Provisioning

This scenario addresses the need to provision services with respect to the context of the endpoint that accesses them. Following the first scenario, in addition to know whether a certain request was issued by a smartphone or not, further information regarding the context of the device is needed to allow or deny a certain access request.

A context in this scenario is defined by a set of variables and their associated values. Examples for such variables are a geographical location (for example Global Positioning System (GPS) coordinates), a time interval (like from 8am to 16pm) or the presence of other devices (like nearby WAPs). The notion of a context is derived from CRePE [38]. However, this scenario uses a context for a different purpose. In contrast to CRePE, which basically extends the Android framework to allow additional security checks depending on policies that are pushed to the smartphone (more precisely additional checks for the use of Android permissions), this scenario focuses on the provision of services based on the smartphone's context.

This scenario covers mechanisms to identify the current set of contexts a smartphone is used in. Similar to the first scenario, it must be possible to distribute this knowledge within the infrastructure of a company in an efficient way.

Example Use Case Again, a company provides their employees with smartphones. Those devices can be used both for business as well as for private tasks. The company wants to enforce a policy that ensures that business data is only exposed to a phone if and only if the following contexts are fulfilled: the employee uses the smartphone on-site at the company and the access takes place during the normal working hours. The approach developed in this thesis must provide means to capture the smartphone's context and to propagate this context within the IT infrastructure. Furthermore, a service that is about to be consumed must be able to check the context of the requesting device and allow or deny its consumption according to a policy defined by the respective company.

2.2.3 Scenario III: Detection of Malicious and Unwanted Apps

One crucial aspect of modern smartphones is their support for third-party software. Users can customize their smartphones by installing apps. While the majority of available apps provides benign functionalities, malicious apps that aim to harm the user exist too. Their malicious functionality ranges from simple defacement capabilities over blackmailing the user to more sophisticated approaches that aim to sniff for sensitive data by leveraging the built-in sensors of smartphones. The latter category of malicious apps is also referred to as sensory malware [10].

The presence of third-party apps can impose risks both for the smartphone itself and the IT infrastructure it is used in. It is therefore necessary to detect both the presence and the activity of apps on smartphones. For the depicted scenario, it is not sufficient to use existing, host-based security tools like anti virus scanners that specifically search for malicious apps. Instead, it depends on the policy of the respective company to specify the characteristics of an unwanted or malicious app. That is, a genuine app that is popular, provides correct functionality and that is used by a lot of different users might nevertheless be unwanted by a company under certain circumstances and must thus be detectable by the developed approach. It is an open question which data and approaches are best suited in order to find malicious and unwanted apps. The approach will investigate methods in order to leverage the capabilities of existing security services, in conjunction with new developed software components, in order to find adequate answers.

Example Use Case A company provides their employees with smartphones. Smartphones are primarily used for business tasks. However, the employees are also allowed to use the smartphones for private purposes. This directly implies that third-party apps are installed by the employees. Thus, it is also possible that malicious and unwanted apps are installed. In this scenario, the term *malicious* refers especially to apps that try to snoop for sensitive data by leveraging a smartphone's built-in sensors (like audio, video or position data), and then try to sent the gathered data to a remote destination under the control of an adversary for further processing. The developed solution should detect such malicious apps by analyzing data that describe the current status and activities of the smartphone.

2.2.4 Scenario IV: Policy-based Enforcement

The first three scenarios focused on detection capabilities that must be supported in order to ensure a secure integration of smartphones into existing IT infrastructures. In contrast to that, the fourth scenario explicitly addresses the question how the endpoints and services within an IT infrastructure can be protected once a policy violation is detected. Regarding the first scenarios, such a policy violation can manifest itself in different ways:

1. A smartphone tries to access a service that is not allowed to be accessed by this type of endpoints.
2. A smartphone tries to access a service while being within a context that disallows the service consumption.
3. A smartphone has malicious or unwanted apps installed while being connected to the IT infrastructure of the respective company.

In order to react on such policy violations, the approach must support to distribute the detection results throughout the corporate IT infrastructure. Thus, services that are capable of mitigating the detected policy violation can respond accordingly. Referring to the previously defined reference IT infrastructure, especially flow controllers are expected to be used for this purpose.

Example Use Case Flow controllers are part of the reference IT infrastructure (cf. Figure 2.1). They are primarily firewall systems and layer 3 switches. However, also NAC solutions and VPN gateways can provide some sort of flow controlling functionality. For this use case, again a smartphone is used by an employee both for private and business tasks. Thus, the employee is allowed to customize the smartphone by installing third-party apps. Since he did not limit himself to the official app stores of his respective smartphone platform but also installed apps from unofficial web sites, the device was compromised with a sensory malware app as mentioned in Section 2.2.3. The presence of this malicious app is detected based on the approach developed within this thesis. This detection result must be propagated transparently to other parties in the network. Flow controllers are expected to consume these results in order to limit the potential damage that can be caused by the smartphone and the respective, malicious app. For example, a packet filter can be configured to block any outgoing traffic that originates from the smartphone. A

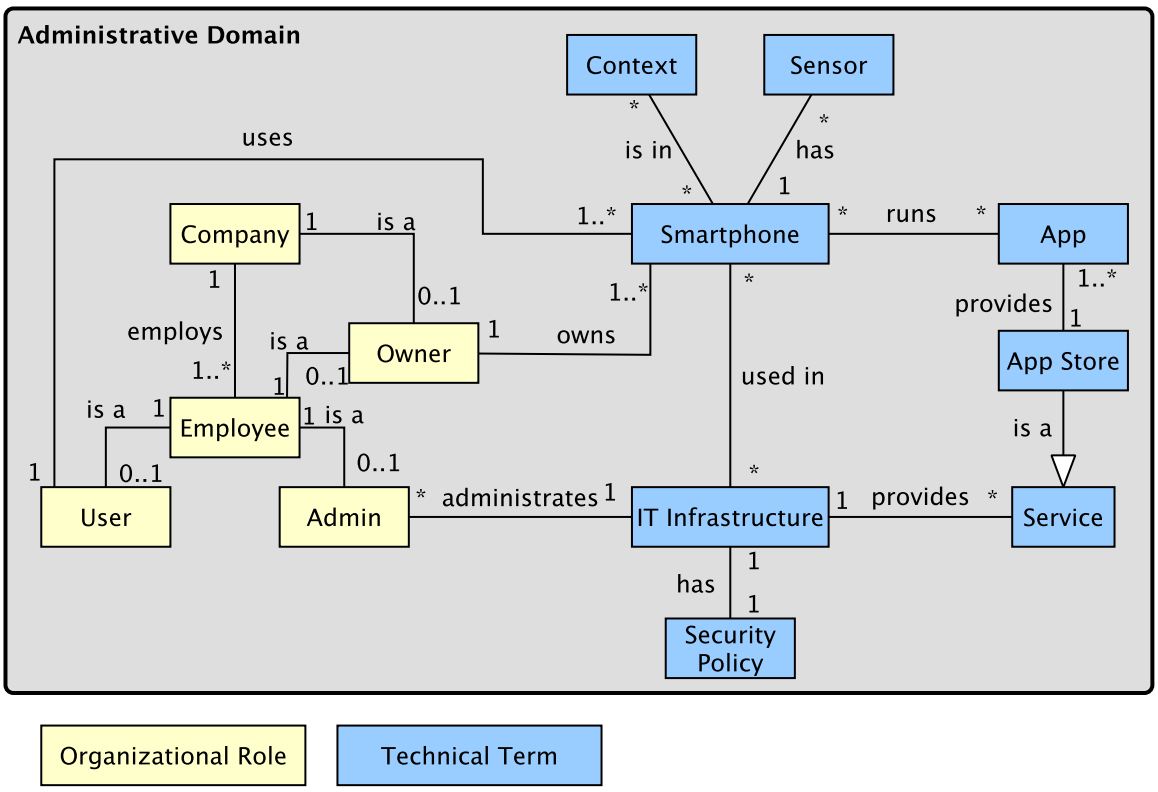

Figure 2.2: Organizational roles, technical terms and their relationships depicted as UML class diagram.

WAP that supports IEEE 802.1X [32] can be used to isolate the connected smartphone to a specific VLAN. If the policy violation is critical, the respective smartphone might also be disabled remotely.

2.3 Terminology

In the following, a set of organizational roles, technical terms and their relationships are defined in order to provide a consistent vocabulary for the remainder of this thesis. These terms allow to express the previously defined scenarios in a more generic way. Furthermore, the notion of administrative domains is introduced. They are used to clearly specify the scope of this thesis.

2.3.1 Organizational Roles

The following organizational roles are defined for the remainder of this thesis as depicted in Figure 2.2. Their notion was derived based on the online Oxford Advanced Learner's Dictionary[7].

Company A company represents an organization that conducts business in order to make profit. Regarding the relevant scenarios, a company wants to use smartphones in order to support their business tasks.

Owner This is the actual owner of the physical smartphone device. That is, the owner has paid money in order to buy the smartphone. Thus, it is his property. A smartphone is either owned by a company or by an employee.

Employee An employee works for a company in order to earn money. There are two types of relevant employees that are distinguished: users and administrators. Employees that are not appropriately represented by those two types are not of interest.

User A user is an employee that uses a smartphone for various tasks. The smartphone in use can be his own (which means that the user is also the owner) or can be provided by a company (which is then considered to be the owner).

Administrator An administrator is responsible for managing the IT infrastructure of the company he is working for.

It is important to note that an employee can be a user, an administrator and an owner at the same time.

2.3.2 Technical Terms

IT Infrastructure For the remainder of this thesis, an IT infrastructure is primarily composed of smartphones and services. Smartphones are used within the IT infrastructure, services are provided by the IT infrastructure. An IT infrastructure has a security policy. Furthermore, an IT infrastructure is administered by a number of admins.

[7] http://oxfordlearnersdictionaries.com/

Service A service provides a certain functionality within an IT infrastructure. As already stated in Section 2.1, security services are of special interest for this thesis. However, there might also be further services available that provide arbitrary functionality. Those services might be considered as valuable assets, and thus need appropriate protection from threats as well.

Security Policy A security policy defines how information security shall be established within an organization. This is usually done on an abstract level, omitting technical details. The security policy must be kept up to date. Changes to the security policy are necessary if the addressed IT infrastructure changes [24]. The introduction of a new type of devices like smartphones is an event that demands to update the corresponding security policy.

Smartphone The term smartphone refers to the physical device. It is owned by an owner and used by an user. Due to its mobility, a smartphone can be used in numerous IT infrastructures.

App Apps are software components specifically developed for a certain smartphone platform. They are installed on and executed by a smartphone. Apps can either be shipped with the smartphone platform itself or can be installed later.

App Store An app store represents a service that is normally provided by the maintainer of a concrete smartphone platform. In this case, it is referred to as official app store. However, there are also unofficial app stores that are maintained by other parties. This type of service enables the user of a smartphone to install third-party apps on demand. App stores are usually web-based services and are thus accessed via the Internet.

Sensor Smartphones come with a set of built-in sensors. A sensor is able to capture a certain aspect of the smartphone's physical environment (such as audio, video or position data). The exact number and functionality of sensors vary depending on the smartphone model and platform.

Context Generally, the term context is defined as *"the situation in which something happens and that helps you to understand it"* [19]. Within this thesis, a context encapsulates information about the physical environment of a smartphone. Examples

for concrete contexts are given in Section 2.2.2. A smartphone is either within a certain context or not. That is, there is no way that a smartphone can be partially within a concrete context. However, it can be within multiple contexts at the same time.

Further details on the technical terms related to smartphones are given in Section 3.1 and 3.2 while discussing the state of the art in smartphone security.

2.3.3 The Notion of Administrative Domains

The last important building block is the notion of administrative domains. The term is used in order to address all organizational roles and technical terms that belong to the same organizational entity. Referring to a company, all components within the IT infrastructure and all employees belong to the same administrative domain (the one of the respective company). The company is responsible for making decisions that affect the way the infrastructure is maintained and used. In contrast to that, an employee or a smartphone from another company or a customer is considered to belong to another administrative domain.

Administrative domains are important in order to clearly specify the scope of this thesis. Any further discussion focuses on issues that are relevant within a single administrative domain. Questions that introduce or require interaction across multiple administrative domains are not completely ignored. However, it is not the focus of this work. For example, app stores are usually web-based services that are maintained by the manufacturer of the smartphone platform. Thus, those services usually do not belong to the administrative domain of a company (as long as they do not host their own app store). However, as app stores are a crucial part of modern smartphone's ecosystem, they are also considered within the remainder of this thesis.

On the other hand, the secure integration of smartphones that belong to another administrative domain is not specifically addressed. Consider two companies A and B. Employees of company B might bring their smartphones to a meeting that is held within the buildings of company A. The secure integration of smartphones from company B within the IT infrastructure of company A is not directly addressed within this thesis. This use case is more related to the problem of integrating guest devices into an IT infrastructure in a secure way.

With the set of organizational roles and technical terms defined above, all of the concrete scenarios can be expressed in a more abstract way. Furthermore, the roles and terms help to prevent ambiguity during the remainder of this thesis. In each of the concrete scenarios, employees use smartphones within an IT infrastructure provided by their company. The smartphone can either be their own or can be provided by the company. In each case, the smartphone will be used for both business and private tasks. This situation covers the current challenge that companies have to face due to the ubiquitousness and high usage of smartphones.

Administrative domains are used to define the scope of this thesis. As stated above, the focus of this work is to investigate what impact smartphones have with respect to security while considering one single administrative domain. This is sufficient to cover the scenarios stated above where employees use smartphones within the IT infrastructure of their company. Investigations that cover multiple administrative domains are not addressed within this thesis. However, the achieved results may form the basis for future work that addresses multi domain scenarios. If smartphones can be securely integrated into existing IT infrastructures of a single administrative domain, further work can be conducted to aim for a secure integration among multiple administrative domains.

That is, use cases where an employee uses his smartphone outside of his company's administrative domain (for example by using a public hotspot within a coffee shop) are considered to be out of scope. This is analogues to a user that tries to access a company's IT infrastructure with his smartphone although he is not an employee. In order to protect an IT infrastructure from unknown devices, several well-known techniques can be employed. Authentication of users and devices and strong encryption of wireless networks are two of them. To protect a company's smartphone in foreign administrative domains is not that trivial. However, this research question belongs primarily to the field of system security and will most likely lead to solutions that harden the smartphone platform in use. Although these use cases are not the focus of this work, the approach that is presented in the following might also provide some benefits for them.

2.4 Trust Model

Some assumptions are made regarding the trustworthiness of networks, smartphones and the type of attacks that are considered within this thesis. They are described in the following:

Untrusted Networks Any network is untrusted. An attacker can modify or eavesdrop any traffic that is carried over the network. This holds both for known (that is a company's) network as well as for unknown networks (such as the Internet). Thus, measures in order to ensure confidentiality and integrity of data that is transmitted over networks are mandatory.

Trusted Smartphone Platforms The smartphone platform (that is the hardware, the firmware, the operating system and the smartphone platform's middleware software stack) form the Trusted Computing Base (TCB) as defined by Lampson et al. [39] for the remainder of this thesis. Hardware-based attacks and attacks that modify the smartphone platform itself are not the focus of this work. This thesis primarily addresses threats that are caused by third-party apps that either are benign but unwanted from the perspective of a company or apps that implement malicious functionality.

No Insider Attacks Insider attacks are out of scope. It is expected that employees behave in accordance to policies defined by their company. However, they can be fooled with social engineering techniques to perform actions that violate the security policy of their company. The detection and prevention of insider attacks is a separate field of current research [40].

2.5 Requirements Analysis

In the following, a requirements analysis based on the presented scenarios is performed. The analysis highlights the most important requirements that have been identified and omits those that are considered trivial in a sense that they are virtually relevant for any approach in any domain (for example the requirement to have an adequate documentation). Each of the stated requirements is equally important (that is, there is no weighting of individual requirements).

R-01 Detection of unwanted and malicious configurations of smartphones The developed approach must support to detect unwanted and/or malicious configurations of smartphones that are connected to the IT infrastructure. This requirement is directly derived from the scenarios I and III (Sections 2.2.1 and 2.2.3). The term configuration refers to the status of the smartphone's hardware and software. In general, the presence of malicious apps leads to a malicious configuration. The same holds if a smartphone's built-in sensors are activated although it is not permitted. The approach must support to capture the current configuration of a smartphone and to reason about it.

R-02 Detection of abnormal smartphone behavior The approach must detect if a smartphone behaves abnormal. Such a behavioral change can be caused by malicious apps, which impose great risks for an IT infrastructure and the smartphone itself. However, malicious apps are just one possible factor that can cause abnormal behavior. Others include physical loss of the device (which is then potentially used by an unauthorized user) or the effects of apps that are generally benign but are considered to cause abnormal behavior under certain circumstances (like streaming video data from a sensitive environment within an IT infrastructure). Thus, the requirement is derived from scenario III (Section 2.2.3).

R-03 Consideration of context information for detection This requirement is primarily derived from scenario II (Section 2.2.2). The approach must provide mechanisms to easily capture the context of each smartphone. This information will also be used to support the fulfillment of the first two requirements. That is, based on the context of a smartphone, it is decided whether a certain configuration is malicious respectively unwanted or if the observed behavior is considered as being abnormal.

R-04 Policy-based reaction on detection results This requirement is derived from scenario IV (Section 2.2.4). It is necessary that the approach allows to react on detection results in a flexible way based on defined policies. Simple notifications without initiating countermeasures to mitigate identified threats are not sufficient.

R-05 Dynamic analysis at runtime The approach must support dynamic analysis at runtime. That is, any analysis, detection and enforcement capabilities must be employed while smartphones are actually used within an IT infrastructure. Techniques

that are limited to analyze data offline (like inspecting the code of installed apps) can be used in addition. In this case, they have to be integrated in such a way that their results can be used at runtime without any delays, for example by precomputing them.

R-06 Extensibility of processed data and used methods The approach must be extensible. This refers both to the data that is processed in order to detect unwanted configurations and abnormal behavior as well as to the methods that are used for processing. The same requirement applies for the data that is used in order to determine a smartphone's context.

R-07 Ability to integrate the approach in existing environments Another requirement is the capability to integrate the developed approach into existing environments. As companies will have a wide range of IT systems and security services already in use, the goal is to find ways to leverage the functionality of available components for the described scenarios. For example, existing services can be used to provide data about a smartphone. This data can then be used to detect malicious apps and abnormal smartphone behavior. Furthermore, existing services should be used to react on detection results. Another aspect is the fact that strategies like BYOD should be supported as well. Thus, approaches that need modifications of the smartphone platform itself are generally not well suited.

Table 2.1 summarizes the list of requirements. They will be used in the remainder of this thesis in order to review related work and existing approaches. Furthermore, the requirements will be used in order to assess the novel, network-based approach for smartphone security that is developed within this thesis.

Table 2.1: Requirements that must be fulfilled in order to enable a secure integration of smartphones into existing IT infrastructures.

ID	**Requirement**
R-01	Detection of unwanted and malicious configurations of smartphones
R-02	Detection of abnormal smartphone behavior
R-03	Consideration of context information for detection
R-04	Policy-based reaction on detection results
R-05	Dynamic analysis at runtime
R-06	Extensibility of processed data and used methods
R-07	Ability to integrate the approach in existing environments

3 State of the Art and Related Work

"Motivation is what gets you started. Habit is what keeps you going."

(Jim Rohn)

Contents

This chapter discusses the state of the art and related work in the field of smartphone security. Based on a literature review, an assessment of existing approaches is done with respect to the requirements that were identified in Chapter 2. It reveals that current approaches fail to meet all of the requirements for a secure integration of smartphones into existing IT infrastructures.

3.1 Introduction to Smartphones

3.1.1 Definition

The field of smartphone security has gained a lot of momentum during the past six years. Both researchers and companies have started to work on related topics in parallel. This development and the rapid change of the capabilities of devices led to a certain amount of ambiguity when it comes to actually name relevant concepts. This caused some confusion, especially regarding the distinction between mobile devices, mobile phones, feature phones and smartphones. The terminology that is defined in the following is based on the work of Becher [41] and Zheng et al. [42] while using some of the terms defined in Section 2.1.

Mobile Device A mobile device is any endpoint that is powered by a battery. This includes laptops, PDAs, tablets and mobile phones.

Mobile Phone Mobile phones are mobile devices that provide a limited but essential set of features. They are primarily used to make phone calls and to send Short Message Service (SMS) messages. As they have only low processing power and small displays, they provide a long battery life. The terms cell phone and mobile phone are used interchangeably. Any mobile phone contains a so-called subscriber identity module (SIM) card that is controlled by a mobile network operator (MNO).

Feature Phone In contrast to mobile phones, feature phones provide larger displays, larger processing power and are able to browse the Internet. This comes at the

cost of reduced battery life. Feature phones are based on closed operating systems and although generally support the concept of apps that can be provided by their manufacturer, they cannot be extended by adding third-party apps.

Smartphone The main aspect that differentiates smartphones from feature phones is their support for third-party apps. Smartphones have operating systems that provide a rich application programming interfaces (APIs) to allow those third-party apps tight integration with the rest of the platform. As already mentioned in Section 2.3, those apps are obtained from web-based services called app stores.

Since 2011, there is a legal dispute whether the term "app store" (and variations thereof like "appstore") is a official trademark of the company Apple Inc. or not [43]. Furthermore, app stores recently also provide other assets besides apps like music, videos and books. Thus, other terms like mobile markets, mobile marketplaces or just app markets have evolved. However, since this thesis is primarily concerned about smartphones and their extensibility by third-party apps, the generic term app store will be used. When explicitly addressing the store maintained by Apple, the phrase Apple App Store will be used.

In addition to the extensibility, there are further aspects that differentiate a smartphone from a feature phone. Smartphones provide even more processing power than feature phones and achieve better connectivity via various interfaces such as Bluetooth and near field communication (NFC). They support to access the Internet both via WLANs that are based on the IEEE 802.11 set of standards as well as directly via cellular phone networks. Additionally, they incorporate various sensors in order to obtain data from their physical environment. For example, the phone's location can be obtained via GPS sensors, audio data via the microphone and video data via built-in cameras. As also smartphones evolve, more sophisticated sensors like accelerometers, gyroscopes and barometers are now also common in new devices like the Google Nexus 4. Due to their versatility, smartphones are used for a wide range of tasks (besides telephony and web-based services). Especially the support of NFC has yielded new use cases such as NFC-based access tokens [44] and identity verification techniques [45].

3.1.2 Overview of Current Smartphone Platforms

As smartphones have evolved over the past years by means of additional processing power, larger displays and more sophisticated built-in sensors, so have their operating systems. Thus, there has been a wide range of both commercial and open source platforms for smartphones. Some projects even aimed to specify both the hardware and the software of smartphones, like the Openmoko project[1].

Today, there are two major smartphone platforms of importance: Google Android and Apple iOS. According to the "Worldwide Quarterly Mobile Phone Tracker" maintained by IDC, those two platforms had a market share of 75% and 14.9% respectively in the third quarter 2012 in terms of shipped units [46]. Further platforms that are still available and that were considered by the study include BlackBerry from Research in Motion (RIM) (4.3%), Symbian from Symbian Ltd. (2.3%) and Windows Phone 7 from Microsoft (2.0%).

These numbers proof that the presence of smartphones based on Android or iOS is predominant. However, they are also somewhat misleading, implying that the importance of Apple's iOS devices is almost irrelevant compared to Android. Two side notes should be taken into account before reasoning about the given numbers:

- The new iPhone 5 was launched late in the third quarter (September 12th 2012). Given the fact that the demand for the new model was very high [47], the market share will likely change accordingly.

- Although the market share of Android is predominant, the platform's app store generates less revenue compared to the official Apple App Store. In 2011, the Apple App Store for iPhone generated nearly four times more revenue [48]. This even excludes apps that are only available for the Apple iPad.

Furthermore, there might be a shift of market shares from Android and iOS to smartphones that use Microsoft's Windows Phone 8, once these devices are widely available in the fourth quarter of 2012. Nevertheless, Google Android has become the market leader for smartphone platforms. Some of its details will be presented in Section 3.2 with a focus on its security mechanisms. However, a lot of the concepts used in Google Android are also available for other smartphone platforms.

[1] http://www.openmoko.org/

3.2 The Android Platform

The first version of the Android platform was announced in 2007. Its development is mainly driven by Google and other members of the Open Handset Alliance. The source code is released under the Apache License [49]. The progress of the open source development is tracked within the Android Open Source Project (AOSP)[2], which is also led by Google. The first Android smartphone was the HTC Dream (also known as T-Mobile G1), which was released in October 2008. Since then, the Android platform has encountered several extensions and improvements. This also includes specific security mechanisms like the support for address space layout randomization (ASLR). Since version 1.5, each major Android version is named after a dessert. The latest version at the time of this writing was 4.2 codename "Jelly Bean" released in November 2012. The complete version history of Android is summarized in Table A.1 in the appendix.

Since its initial release in 2008, there have been 32 updates released for Android. Most of them included bug fixes. There have been eight larger updates, each one introducing a new Android version with an associated codename. Those major updates normally introduced a set of new features, including security features like updated kernel versions, the support for VPNs, a full implementation of ASLR or the support of SELinux[3]. The version history proofs that the Android platform is actively maintained, and still under a rapid development. In addition to the official Android version, there are also projects that provide customized Android versions, for example CyanogenMod[4].

However, the plethora of Android versions also introduces some issues in terms of security, commonly referred to as Android Update Problem [50]. Smartphone manufacturers like HTC and Samsung provide customized versions of Android for their phones, primarily to ensure their own look and feel and to adapt Android to their specific hardware platforms. Due to the rapid frequency of updates provided by Google, manufacturers fall behind to actually adapt the updates for their needs and to deploy them on their devices. As a result, most of the Android devices run outdated versions of the Android platform, thus lacking the latest bug fixes and security updates. Google provides data regarding the distribution of Android versions on a regular basis [51]. Data that was collected during a 14-day period ending on November 1st 2012 is depicted in Table 3.1. Only 2,7% of

[2] http://source.android.com/
[3] http://selinuxproject.org/
[4] http://www.cyanogenmod.org/

Table 3.1: Distribution of Android versions. Data obtained within a 14-day period ending on November 1st, 2012 [51].

Version	Codename	API Level	Distribution
1.5	Cupcake	3	0.1%
1.6	Donut	4	0.3%
2.1	Éclair	7	3.1%
2.2	Froyo	8	12%
2.3 - 2.3.2	Gingerbread	9	0.3%
2.3.3 - 2.3.7		10	53.9%
3.1	Honeycomb	12	0.4%
3.2		13	1.4%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	25.8%
4.1	Jelly Bean	16	2.7%

devices were running the latest Android version "Jelly Bean" that was released in July 2012. In contrast, more than 50% of the devices were still running Android "Gingerbread" which was initially released in 2010. This problem is unique to the Google platform. For example, when Apple releases a new version of their iOS platform, all supported devices are updated by default (as long as the user does not choose to prevent the update). At the Google I/O conference 2012, Google announced that it will release a special Platform Development Kit (PDK) for hardware developers in order to mitigate this problem. The PDK will be made available two to three months before each major Android update.

The following sections detail certain aspects of the Android platform. An overview of its architecture is given in Section 3.2.1. Google Play, the app store for the Android platform, is introduced in Section 3.2.2. The fundamentals of Android apps are described in Section 3.2.3. The main part is a discussion of Android's built-in security mechanisms in Section 3.2.4.

3.2.1 Architecture

The architecture of the Android platform consists of five main building blocks. They are briefly described in the following.

Linux Kernel The fundamental basis for the Android platform is a customized Linux kernel. Enhancements that are added for Android primarily address the power manage-

ment and the support of inter-process communication (IPC) via the Binder driver. The first Android releases including Honeycomb were based upon a Linux kernel in version 2.6.x. Since Android Ice Cream Sandwich, a Linux kernel in version 3.x is used.

Libraries Android includes a set of native libraries for various purposes, including SSL/TLS for secure connections, SQLite databases and WebKit for rendering HTML and JavaScript. As standard C library, Android uses the Bionic libc instead of the GNU C library.

Android Runtime The Android runtime environment consists of the Dalvik Virtual Machine (Dalvik VM) and a set of Java core libraries. Apps for Android are normally written in Java and executed by the Dalvik VM. Java `.class` files are converted to `.dex` (Dalvik Executable) files before they are executed by the Dalvik VM on a device. The Java core libraries provide developers with a familiar environment, comparable to ordinary Java Development Kits (JDKs).

Application Framework The application framework includes a set of services that are essential for the Android platform in order to handle telephony, resource management and location based tasks. Apps can use these services via a simple Java API.

Applications The applications building block includes any app that is installed and executed on the device. This includes so-called system apps that are directly shipped with the device as well as third-party apps that are installed afterwards. Apps are written in Java and make normally use of services provided by the application framework. However, it is also possible to use native code within an app by leveraging the Java Native Interface (JNI).

These components form the basic architecture of the Android platform. A more detailed introduction to Android and its architecture is given by Reto Meier [52].

3.2.2 Google Play

Google Play[5] is the official app store for the Android platform. It was formerly known as the Android Market. It was renamed in March 2012 as Google added further digi-

[5] http://play.google.com/

tal content besides apps, including music, movies and books. The store can be accessed directly from smartphones with the corresponding app named Google Play Store. Furthermore, Google Play can also be accessed with an ordinary web browser. Users can browse through available apps based on aspects like their category or whether they are free or cost money. For each app, further information like the developer, the average user rating and the number of downloads are provided as well. Currently, there are 34 different categories supported. Those categories are itself divided into categories for "applications" and categories for "games". Within this thesis, both applications and games are referred to as apps.

In October 2012, Google announced that there were more than 700,000 apps available in their app store [53], which is a similar number as the main competitor Apple features in its own App Store for iOS devices. Although the total numbers of available apps are comparable, the distribution model of Google and Apple varies greatly. Whereas Apple employs a closed approach, restricting its users solely to its official app store, Android supports a more open model that explicitly allows unofficial app stores (such as AndroidPIT[6]).

3.2.3 App Fundamentals

The extensibility through third-party apps is one major success factor of modern smartphones. Thus, the fundamentals of Android's app development framework will be introduced in the following. Android apps are written in Java and bundled, together with all further resources like images or sound files, into an Android Package archive with a (`.apk`) suffix. Multiple apps are executed in isolated environments, each belonging to its own security sandbox. Section 3.2.4 gives further details on the sandboxing mechanisms of Android. According to the Google API Guides [54], each app is composed of one or more of the following components:

Activities An Activity is the main component for building user interfaces on Android. Each Activity represents a single screen that is shown to the user on the smartphone's display. A single app is usually composed of multiple Activities. For example, a camera app can have one Activity in order to display the current picture and another Activity that allows the user to specify configuration settings.

[6] http://www.androidpit.de/de/android-market

Services Services do not provide a user interface screen. Instead, they are used for performing long running tasks in the background. For example, in order to continuously play a music file, a Service component should be implemented. The user interface however would be realized by implementing an appropriate Activity.

Content Providers Content Providers are responsible for providing and controlling access to data. The data can be private for one app or can be shared among multiple apps. For example, each Android smartphone ships with a standard Content Provider that manages access to the users address book.

Broadcast Receivers Android heavily relies on broadcast messages that are sent both by third-party apps and system services. For example, when the screen is turned off or an SMS message is received, appropriate broadcast messages are sent. Broadcast Receivers are components that are responsible for receiving and reacting on such broadcast messages.

Android supports inter-component communication (ICC). Generally, any app can start not only its own components, but also the components of another app. For example, if a third-party app wants to take a picture, it will likely start the corresponding activity of the standard camera app provided as part of the Android platform. Communication between components, whether within a single app or across multiple apps, is primarily realized by asynchronous messages referred to as Intents. Activities, Services and Broadcast Receivers make use of Intents. An Intent declares a recipient and optionally contains further data as payload. The recipient can be named explicitly, ensuring that the Intent is transmitted to a specific, known component, or implicitly by specifying a so-called action string. Receiving components can define Intent Filters based on these action strings in order to get started when an Intent occurs whose action string is covered by their Intent Filter. After the called component receives the Intent, it can make use of its payload. Content Providers are not accessed by means of Intents. Instead, they receive requests from so-called Content Resolvers. This introduces another layer of abstraction, primarily for security reasons [54].

In order to protect an app's components, they can be declared public or private. Furthermore, permissions can be named that the calling app must have requested in order to access the respective component. If not explicitly declared as public or private, the Android platform infers a default setting based on other parameters provided as part of

the app's manifest file [55] which is detailed below. However, recent work has shown that these default inference rules cause many developers to implement app's that unintentionally provide access to components that should remain private [56].

The last crucial building block of Android's app framework that is presented here is the so-called manifest file. The manifest is a Extensible Markup Language (XML) configuration file that is contained within each `.apk` package. Among other parameters, it declares all of the app's components. Furthermore, the manifest contains the list of permissions that the corresponding app requests to use and the list of Intents that it wants to receive. Both of them are expressed by specific XML elements referred to as `<uses-permission>` and `<intent-filter>` respectively.

3.2.4 Security Mechanisms

The Android platform includes several security mechanisms. Most of them aim to protect the user and its data from malicious third-party apps or in case the device is lost. Although their concrete implementation is specific for the Android platform, the general security mechanisms are also common on other platforms like Apple iOS. In order to describe the security mechanisms of Android, terminologies that have been proposed in the past [57, 58, 59] will be used. The description covers all Android versions that were available at the time of writing, including Android 4.2.

Kernel Security

At the kernel level, Android provides basic security mechanisms that are known from classical Linux-based computing platforms such as discretionary access control (DAC) for files based on user identities and the isolation of concurrently running processes. Furthermore, the kernel provides protection from runtime exploits that work based on corrupting the stack or heap memory. This includes a full ASLR implementation since Android version 4.1 and hardware-based No eXecute (NX) to prevent code execution on the stack and heap since Android version 2.3. Furthermore, support of SELinux was added in Android version 4.2.

Device Access Control

Device access control mechanisms enable to control which users are allowed to use a smartphone. The purpose is to prevent unauthorized access to the smartphone, especially when the device is lost. Thus, in contrast to other security mechanisms, this one does not primarily target the threats introduced by malicious apps. Protected smartphones are "locked" when they are not actively used. In order to "un-lock" a device, Android supports various techniques, including passwords, patterns and personal identification numbers (PINs). Furthermore, since Android version 4.0 it is also possible to un-lock a device by taking a picture of the user (referred to as Face Unlock). However, it is also possible to completely disable the device access control mechanism or to configure weak techniques that simply rely on the physical presence of any user (finger swipe). Apple iOS provides similar device access control mechanisms as Android.

Filesystem Encryption

Filesystem encryption seeks to protect data that is stored on the smartphone in case it is stolen or lost. Since Android version 3.0, full filesystem encryption is supported. Thus, any smartphones that use prior Android versions are not able to protect their data this way. Given the distribution of Android versions presented in Table A.1, more than half of the devices are unable to appropriately protect data at rest. On Android, encryption is done in the Linux kernel by using the dmcrypt[7] implementation of the Advanced Encryption Standard (AES) [60, 61]. Apple iOS supports a similar level of filesystem encryption.

Sandboxing

A crucial security concept of Android (and many other smartphone platforms) is the isolation of third-party apps by means of so-called application sandboxes. Apps are both isolated from accessing each other as well as from accessing the smartphones resources in an uncontrolled way. Application sandboxing on Android is realized based on the features provided by the Linux kernel. Each Android app is assigned a unique user id (UID), runs in its own process and has its own directory. The permissions of the directory are set in such a way that the app's UID is the owner and only owner filesystem permissions are

[7] http://code.google.com/p/cryptsetup/wiki/DMCrypt

set. Furthermore, each app runs in its own instance of the Dalvik Virtual Machine. This sandboxing mechanism also applies for native code used by the app. Since the interaction of apps both with other apps and with resources provided by the smartphone is essential, Android supports mechanisms to "exit" an app's sandbox in a defined way. These mechanisms are controlled by a permission-based access control model that is detailed below. Apple's iOS platform supports similar sandboxing mechanisms.

Permission-based Access Control

In order to enable apps to access resources and data that is not contained within their sandbox, modern smartphone platforms implement permission-based access control models. Basically, apps are allowed to access a smartphone's resources such as location sensors and components of other apps if they have the necessary permissions to do so. The permission model is enforced by the smartphone platform at runtime and can generally not be circumvented. Android provides a sophisticated permission framework in order to realize mandatory access control for ICC between different apps and for the access to the smartphone's resources [62]. It is based upon more than 100 predefined permissions [63] that enforce control to the smartphones resources. For example, permissions that are often requested by apps include `INTERNET` for accessing the Internet and `ACCESS_COARSE_LOCATION` respectively `ACCESS_FINE_LOCATION` in order to obtain the current location of the smartphone. Actually, each of the standard Android permissions also has the prefix `android.permission`. However, it is omitted here for readability. Permissions are primarily enforced by a reference monitor in the Android middleware. However, some permissions like `INTERNET` are enforced by the Linux kernel.

In addition to the predefined set of permissions, developers can also define their own permissions. These self defined permissions are normally used in order to control access to components of an app. For example, a developer that has published many apps can define its own set of permissions in order to limit access to exported components within its own set of apps.

Permissions are classified according to protection levels. The purpose of a protection level is twofold. First, it characterizes the general risk that is implied by the respective permission. Second, it determines how the Android platform handles the process when

apps request the respective permission. Permissions on Android are categorized into four protection levels [64]:

1. Normal permissions impose negligible risks to the smartphone and its user. If requested by an app, they are always granted by default without the user's confirmation. An example for such a permission is `VIBRATE`, which allows access to the vibrator of the smartphone.

2. Dangerous permissions are potentially harmful. Granting this type of permission gives apps access to the user's private data and to actions that can cost money. Thus, user confirmation at install time is necessary in order to grant these permissions to apps. An example for such a permission is `SEND_SMS`, which allows an app to sent a SMS message.

3. Signature permissions are only granted to apps that have been cryptographically signed with the same certificate as the app that initially declared the permission. User confirmation is not necessary in this case. An example for such a permission is `CLEAR_APP_USER_DATA`. It basically allows the requesting app to delete all of the user's data. Note that although this might seem like a severe threat, it is not. The permission is declared by an app that was signed by a certificate owned by Google. Thus, no third-party app can successfully request this permission.

4. SignatureOrSystem permissions are granted only to apps that are part of the default system image or are signed with the same certificate as these apps are. According to the official developer documentation, this protection level should not be used at all by developers: *"Please avoid using this option, as the signature protection level should be sufficient for most needs and works regardless of exactly where applications are installed."* [65]. Permissions with this protection level are primarily used by the Android platform in order to protect access to background services that should not be used by ordinary third-party apps. An example is the permission `DELETE_PACKAGES` that allows an app to delete installed apps.

On Android, the granting of permissions to apps is done at install time. It is an all-or-nothing decision made by the user. That is, an app is either granted all permissions that it requests, or it is not installed. It is not possible to just grant a subset of the requested

permissions. Although Android's permission model has been effective in limiting the access of apps to resources that are within the scope of their requested permissions, it has also some drawbacks that have been recently identified by researchers[66]:

- For users, it is difficult to interpret the meaning of some permissions.
- It is not transparent whether a certain combination of permissions might be dangerous.
- Overdeclaration is an issue, that is many apps request more permissions than they would actually need to function properly.

In contrast to the model described above, the permission model that is supported by Apple's iOS platform is rather limited. Basically, all apps share the same permissions. There is no user confirmation required at install time. Instead, the system requests the confirmation of the user when an app aims to access sensitive data in the device at runtime. This includes data such as the users address book, the calendar and the smartphones location.

Application Provenance

The term "application provenance" basically refers to two concepts that are employed in order to mitigate the threats introduced by third-party apps. First, application provenance includes mechanisms that aim to ensure the integrity of an app and to authenticate its author. This way, users can decide whether they want to install a certain app based on the identity of the author and they can be assured that the app has not been tampered. Furthermore, software developers can be made accountable for apps that behave maliciously. Second, additional security checks are performed by the platform's app store, either before or after an app has been made publicly available for download.

Android supports both concepts by using digital signatures [67]. In order to fulfill the first concept, any Android app must be digitally signed by the developer before it can be installed on a device. This shall ensure both the integrity of the app an the authenticity of the app developer. However, the Android app signing model is pretty open. That is, apps can be signed by virtually any digital certificate, including self signed ones. There is no need for a certificate authority that is trusted by Google. Consequently, the certificate chain is not checked. The only prerequisite that must be fulfilled is that a

corresponding developer profile is created via the Google Play Android developer console[8]. That is, any developer that is willing to pay 25USD is able to publish apps with self-signed certificates to Google Play. Regarding the used certificate, Google Play only enforces that the expiration date is in the distant future[9]. Other contents of the certificate are ignored. Furthermore, the contents of the certificate used for signing are not transparent for the smartphone user that install the apps. As a consequence, malicious software developers can easily create developer accounts for Google Play as needed.

An implementation of the second concept was recently introduced for Android, referred to as Google Bouncer. It ensures that each app which is uploaded to Google Play is executed in a virtual environment in order to detect malicious behavior. Furthermore, new apps are compared against known malicious apps. If an app is flagged as potentially malicious, it is further investigated manually and if necessary removed from Google Play. The existence of Google Bouncer was only mentioned in a Blogpost [68]. According to Google, Bouncer decreased the number of malicious downloads in the first half of 2011 by 40%. Technical details are not provided by Google. At BlackHat USA 2012, researchers demonstrated that it is easy to circumvent Google Bouncer by making malicious apps context-aware, thus hiding their malicious behavior when they are analyzed [69]. Thus, although Bouncer certainly is a step forward compared to having no vetting process at Google Play at all, there is room for future improvements. However, since there are at least basic security checks, the security level of apps installed from Google Play is higher compared to those obtained from unofficial app stores.

With Android version 4.2, Google added a so-called "application verification service". It allows to perform security scans of third-party apps that have been obtained form unofficial sources. Thus, the general idea of Google Bouncer is extended beyond the scope of the Google Play app store. Although this additional security service is certainly reasonable, first studies reveal that its detection rate is rather low (approximately 15%) compared to those of other anti virus services [70].

In contrast to the open approach followed by Google Play for Android, the Apple App Store is far more restrictive. First, Apple does not support to install apps from unofficial app stores. User that want to install apps from unofficial stores need to gain root privileges by willingly performing a privilege escalation attack. This process is also referred to as

[8] https://play.google.com/apps/publish/signup
[9] After October 22nd 2033.

"jailbreaking". Second, developers must officially register at Apple and pay an annual license fee. In response, they are issued with a digital certificate that must be used to sign their apps. Third, each app must undergo a review conducted by Apple before it is released in the official App Store. Details of the review process are not publicly available.

3.2.5 Summary

The previous sections gave a brief overview of the Android platform. The focus was set on the security mechanisms that are supported by the platform. Android supports features that are well known from other computing platforms such as kernel security features, sandboxing techniques and filesystem encryption. Furthermore, it supports a sophisticated and complex access control model based on so-called permissions. Among other aspects, the permissions that are requested by installed apps will be used in the remainder of this thesis to determine the security status of smartphones. Finally, the discussion of the Android application provenance model revealed that it differs from the provenance model of other smartphone platforms. Android follows an open approach. Reactive measures such as Google Bouncer are used to counter malicious apps.

For more detailed information, especially concerning the development of apps and the implementation of Android's security mechanisms, the reader is referred to the official developer documentation [54] and related work conducted by Enck et al. [55] and Shabtai et al. [58].

3.3 Related Work on Smartphone Security

In the past years, the field of smartphone security, also referred to as mobile phone security, has gained a lot of momentum and was extensively studied by the research community. Starting from the first papers that began to examine the threat of malware for mobile devices [71, 72], the number of publications has grown steadily. The topics and challenges that have been addressed by other researchers are diverse. In order to provide a complete overview of relevant related approaches, the presentation of related work is basically structured by means of three categories:

1. The first category covers all research papers whose main contribution is to provide a basic analysis of certain smartphone security issues. Furthermore, survey articles

that aim to provide a scientific view on the special aspects of smartphones and the ecosystem they are used in are presented as well.

2. The second category of presented research approaches covers work that addresses the specific threats of smartphones in detail. Those articles usually include approaches for exemplary attacks in order to abuse the discovered threats. This kind of work often builds on or refers to results that were accomplished within research papers of the first category.

3. The third category focuses on approaches that introduce countermeasures in order to mitigate various threats that were identified for smartphones. Some approaches also include a simple proof-of-concept attack in order to emphasize their motivation. This third category is especially important in order to compare existing approaches against the requirements defined in Chapter 2.

A mind map that visualizes the related work is depicted in Figure 3.1. It will be shown that the existing approaches cover a wide range of topics in the field of smartphone security. However, no approach does sufficiently address all requirements for the scenarios defined in Chapter 2. The analysis covers approaches that have been published from 2000 to October 2012 at conferences that employed a peer-review of the submitted papers. Primarily, conferences held by ACM[10] and IEEE[11] with a good ranking were considered. Furthermore, workshops that were held in conjunction with well-known conferences were considered as well. Technical reports of universities and research institutions were considered if they match the scope of this thesis and provide a major contribution. Some of the presented research papers address aspects that belong to more than just one of the mentioned categories. For example, a research paper can discuss a new type of attack (thus matching category 2) and provide appropriate countermeasures (matching category 3). In these cases, the categorization is made based on the focus of the contributions.

[10] www.acm.org
[11] www.ieee.org

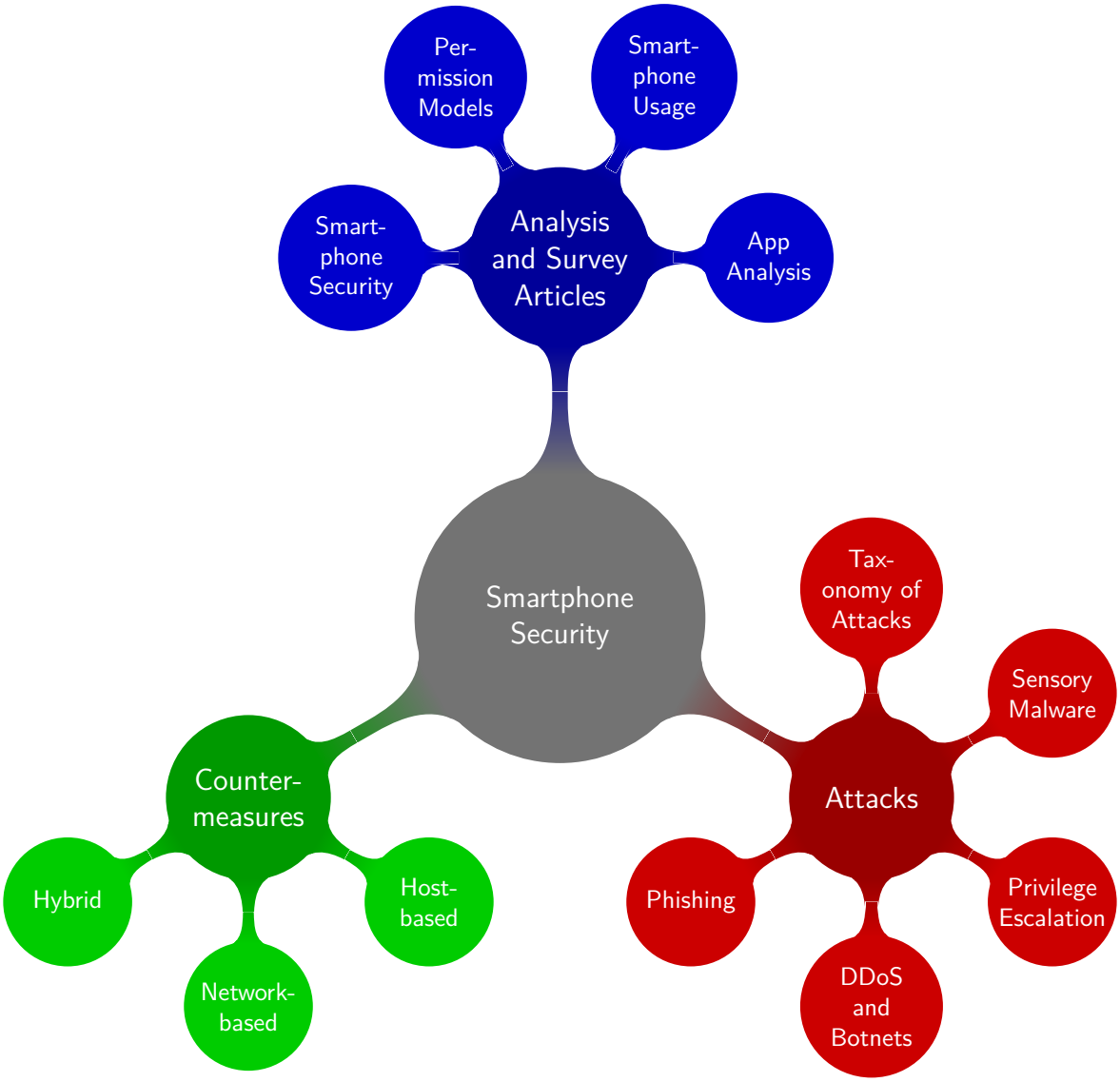

Figure 3.1: Mind map of related work in the field of smartphone security.

3.3.1 Analysis and Survey Articles

General Analysis of Smartphone Security

The first articles that address the field of smartphone security focus on the threat of mobile malware [71, 72, 73, 74, 8, 75, 76]. At that time (from 2000 to 2007), the term smartphone was not even widely established. Instead, the devices were also called mobile phones, cell phones or feature phones. It was claimed that malicious software for mobile phones will be a crucial future threat. Given the latest news on mobile malware that have been seen, the assumption made years ago was true. Jamaluddin et al. [74] provide a proof of concept malware that supports SMS spamming. They emphasize that such mobile malware has a great potential for actually causing a financial loss to the user. This is different from malware for classical platforms. Dagon et al. [8] provide a taxonomy for new smartphone threats. One of the new threats that was not addressed before is called battery exhaustion. Corresponding attacks aim to deplete the battery of a smartphone in a short time. Thus, they can be seen as kind of a Denial of Service (DoS) attack specific for smartphones.

An state of the art survey that explicitly addresses the differences between mobile security and classical fields of information security is contributed by Becher et al. [77]. They argue that the specifics of mobile devices is crucial for research in the field of smartphone security. Three examples for such specifics are (1) the limited resources of the device, (2) the ability of an attacker to easily create financial damage to the user (referred to as creation of costs) and (3) users that are generally unaware of any security issues. Furthermore, they define a classification of attack vectors for smartphones, distinguishing between hardware-centric, device-independent, software-centric and user layer attacks. They especially elaborate the category of software-centric attacks employed by using mobile malware. In terms of the specifics of mobile devices, they conclude that some of them will remain relevant in the long term (such as the creation of costs) whereas others might change (such as the device's resource limitations).

Shabtai et al. [58] provide a detailed security analysis of the Android platform that identifies risks and suggest appropriate countermeasures on a conceptual level. Since they do not provide any details regarding the highlighted countermeasures, their work is categorized as analysis and survey article. They group the security mechanisms that are incorporated in Android into three categories: (1) Linux mechanisms that are provided

by the Linux kernel (such as app specific user IDs), (2) environmental features such as SIM card based user authentication and (3) Android-specific mechanisms like the permission framework. Based on their own taxonomy for Android that covers 18 potential threats, a risk analysis is performed that maps the identified threats to a risk matrix according to their likelihood of occurrence and their potential impact. Based on this analysis, they derive five high-risk threat clusters and suggest appropriate countermeasures. Two of the most important recommendations are (1) to harden the Linux kernel by leveraging access control mechanisms such as SELinux and (2) to extend the permissions framework in order to prevent misuse of granted permissions. Their first recommendation has been meanwhile addressed with the release of Android version 4.2. Their second recommendation however still remains open. A similar study about the security mechanisms of Android was also performed by Enck et al. [55].

Oberheide et al. [78] examine the general challenges of securing mobile environments, especially when approaches from non-mobile domains are adopted. In order to compare existing mobile platforms in terms of security, they introduce a taxonomy for mobile security models that consists of three components: (1) app delivery refers to the ability to verify the authenticity of an app that is deployed on a smartphone, (2) trust levels describe the ability of a platform to assign privileges to apps to access the phone's resources or to perform specific tasks and (3) system isolation refers to the capability of a platform to isolate apps from each other. Five smartphone platforms are compared based on the three mentioned categories, including Google Android and Apple iOS. Android gets the best overall rating. In the end, the authors suggest five general commandments for future smartphone security research like to take forward lessons and to consider multiple platforms when developing new security concepts.

Dixon et al. [79] claim that there is a strong correlation between the location of a smartphone and its power consumption. Depending on the location, smartphones are used differently by their users, leading to specific, location based power consumption profiles. The authors have gathered reference data from 20 users over a period of three months in order to proof their hypothesis. They further claim that malicious code is likely to increase the power consumption of smartphones (primarily due to the use of peripherals like Bluetooth, the radio or built-in sensors). Thus, they propose to implement a malware detection tool based on the location based power consumption of smartphones. Although their approach sounds reasonable, it has two obvious drawbacks: (1) Other features aside

from location and power consumption are not considered. (2) The approach fails at times where the smartphone's battery is charged.

Studies that address the question if smartphone users are concerned about security and privacy have been performed as well. A study that investigates if users are willing to pay premiums when installing apps, given that the additional costs will limit their personal information exposure, was conducted by Egelman et al. [80] . Their results show that users are concerned about their privacy and would pay extra money for apps that do not request access to personal information. A more general study that investigates the perceptions of users in terms of smartphone security was recently conducted by Chin et al. [81].

Permission Models

Permission-based access control models are one main security mechanism of modern smartphone platforms and web browsers. Thus, this field has also gained a lot of attention from the research community. The approaches to analyze permission-based access control models vary. Shin et al. [82, 83, 84] propose a formal approach for analyzing the Android permission model. They identify several drawbacks. Namely that (1) the user is responsible for making informed decisions on permissions that are requested by an app, (2) there are no naming conventions for permissions and (3) once granted, permissions cannot be revoked without uninstalling the respective app. Another formal approach was recently conducted by Fragkaki et al. [85]. Based on their analysis, they propose SORBET, an extended permission system for Android. Besides other features, SORBET prevents confused deputy attacks [86].

Barrera et al. [87] have developed a methodology for the empirical analysis of permission-based security models. Their approach is based on Self-Organizing Maps (SOM) [88]. To proof the feasibility of their approach, they have analyzed 1,100 Android apps in order to learn how permissions are used and to identify the strengths and weaknesses of this implemented security model. Their findings show that permissions are used diversely: there are predominant permissions that are used very frequently (like the `INTERNET` permission used by more than 60% of the apps) whereas others are requested far less frequently (like the `RECEIVE_BOOT_COMPLETED` permission requested by 5% of the apps). Furthermore, their findings show that apps of a certain category tend to request similar permissions. In addition, there are permission pairs that provide similar functionality (like `READ_SMS`

and `WRITE_SMS`). The authors consider the `INTERNET` permission to provide insufficient semantics in terms of how an app actually uses the Internet and suggest to split it up in more fine-grained permissions. Another identified drawback is the large number of available permissions (more than 100), leading to developers that tend to "over-request" permissions just to be sure that their app works properly. Thus, it is even more difficult for users to understand the meaning of a large set of requested permissions at install time. In order to circumvent these drawbacks, the authors suggest to introduce a hierarchical permission structure in order to achieve a logical grouping according to the semantics of permissions.

The problem of over-requesting permissions identified by Barrera et al. [87] has also been addressed as the problem of permission overdeclaration. Developers often declare more permissions than their app actually uses due to (1) the complex and partially ambiguous permission system and (2) due to the lack of assistance in determining the right permissions for specific function calls. Overdeclaration violates the principle of least privilege [89]. The problem was recently tackled by Felt et al. [90] and Vidas et al. [91]. Both have developed tools that help developers to infer the minimal set of necessary permissions by performing a static analysis of the respective apps. The Permission Check Tool by Vidas et al. is realized as an Eclipse IDE plugin that parses the source code and thus infers the set of necessary permissions. The mapping between the Android API calls and the actually needed permissions was derived by analyzing the available Android SDK documentation. In contrast, Felt et al. propose a tool that performs static analysis of compiled Android apps called Stowaway. The tool maps API calls to permission checks. The mapping was derived by leveraging automated testing techniques on the Android API. Their experiment with 940 apps shows that about one-third are overprivileged. Their findings show that developers try to follow least privilege but fail due to poor API documentation and the overall complexity of the permission model.

A general analysis of smartphone permission models is conducted by Au et al. [66]. They provide a taxonomy for the most popular smartphone platforms according to (1) the amount of control the user has, (2) the amount of information that is conveyed to the user which forms the basis for his decision making and (3) the level of interactivity that is required from the user. Their results show that Android has the most complex permission model due to the number of available standard permissions (more than 100) compared to Blackberry OS (24 permissions), Windows Phone 7 (15 permissions) and Apple iOS

(1 permission). Furthermore, younger smartphone platforms like Android (first release in 2008) and Windows Phone 7 (first release in 2010) have very similar permissions models, giving the user some information based on (more or less) fine-grained permissions but limiting the level of control and interactivity to an all-or-nothing decision at install time. The authors claim that this trend is the cause for the problem of permission overdeclaration. Similar to Felt et al. [90], they propose to use static-analysis of an app's source code in order to automatically derive the set of permissions that are actually used, and thus need to be requested by the app. Their system is a work-in-progress.

The effectiveness of app permissions for the Google Chrome Extension system and the Android platform was investigated by Felt et al. [64] as well. They notice that Android apps request an average of less than four dangerous permissions. The most popular permission is the `INTERNET` permission. Apps frequently access permissions that allow them to obtain personal information (such as the location) in conjunction with the permission to access the Internet. A security measure that solely focuses on this type of permissions (for example that denies the installation of such apps) will thus likely fail. However, there are also permissions that are request far less frequently, such as those of the category `COST_MONEY`. The presence of such permissions can be leveraged in order to provide security warnings to the user or implement more sophisticated security measures. The authors further suggest that coarse permissions like `INTERNET` on Android would benefit from a more fine-grained approach as it is implemented in the Google Chrome Extensions system (where Internet access can be restricted to a set of domains).

A study that examines whether the Android permission system is effective in warning the user about risks associated with the installation of third-party apps is conducted by Felt et al. [92]. They performed structured interviews with participants, both on-site and online via the Internet. The authors state that only a minority of the participants were able to fully understand the implications of permissions that are requested by apps. Guidelines for improving permission granting mechanisms are presented in a separate publication [93].

Extensive analysis of the Android permission system has revealed its vulnerability to confused deputy attacks [86]. That this is not a problem of poorly implemented, third-party apps alone was recently proven by Grace et al. [94]. With their analysis tool Woodpecker, they found several apps in the stock images of Android phones that unsafely expose permissions to other apps.

Smartphone Usage Studies

Recently, some studies have been conducted in order to learn how users actually use their smartphones. To learn the normal behavior of smartphone users can be beneficial for a wide range of use cases. Within the scope of this thesis, this is especially true for any kind of anomaly detection. Anomaly detection is a common technique that has been widely adopted in the field of information security, especially for intrusion detection systems (IDS) [95, 96, 97]. The concept is to built a model that expresses the normal behavior of a system (in our case smartphones). Then, based on observations of that system, one aims to find patterns that do not conform to the expected behavior. Observations that deviate from the expected or normal behavior are treated as anomalies. This approach is complementary to misuse detection which is also commonly employed by IDS. Misuse detection aims to model bad or malicious behavior based on signatures of known attacks. Observations of the modeled system are then compared to the list of known signatures.

Xu et al. [98] perform the first large scale study in this field, covering smartphone users across the whole United States of America. They provide various insightful findings that are not directly related to the field of smartphone security. For example, they state that certain apps tend to be used in pairs. That is, if a user has installed one app of a pair, he will likely have installed the second app as well. Furthermore, they conclude that the context of a user has impact on the type of apps he uses. For example, social networking apps are used more frequently when the user is moving around. Furthermore, the location of the user has great impact on the set of apps he uses.

Another study that is still in progress is conducted by the University of Cambridge[12]. Their Device Analyzer app for Android collects various data on the phone, including apps in use, when the user makes phone calls and the coarse, network-based location. Although there have been more than 10,000 contributors, there is still no publication available. Furthermore, even preliminary results are not published yet.

Although there have been some studies that tackle the question of smartphone usage, the effort that is put into this research field is limited compared to other areas.

[12] http://deviceanalyzer.cl.cam.ac.uk/

App Analysis

As already stated, third-party apps impose a major threat for smartphones. Thus, a lot of work that explicitly studies smartphone apps and the corresponding app store platforms has been done.

A major contribution in this field was conducted by Enck et al. [99]. They provide an in-depth study of 1,100 popular, free Android apps. That is, they did not limit their study on data which can be easily obtained from the Android app store (such as app rating, category or the use of permissions). Instead, they developed the `ded` decompiler to recover the source code from binary `.apk` files. The recovered code is analyzed by automated tests and manual inspection. They contribute four main findings: (1) apps misuse sensitive information such as phone identifiers or geographic locations, (2) background recording of audio or video data was not observed, (3) advertisement libraries are common for free apps and (4) developers fail to use the Android APIs in a secure way. The PiOS study conducted by Egele et al. [100] yields similar results in terms of misuse of sensitive information and the use of advertisement libraries for iOS apps by using static analysis techniques as well.

The first survey discussing mobile malware was conducted by Felt at al. [101]. They focus on mobile malware for Android, iOS and Symbian that was seen in the wild from January 2009 to June 2011. Their work basically provides three key findings: First, malicious apps can be classified according to their behavior. The most common malicious activities include the collection of user information and the sending of premium-rate SMS messages. Future malware is expected to exploit new features of smartphones, such as NFC. Second, the detection of malicious apps based upon their requested permissions is in general feasible. For example, 73% of the malicious Android apps requested the permission to send SMS messages, which also allows them to send such messages silently without user notification. Benign apps normally do not request that permission (96%). Even if they need to send SMS messages, they would use the built-in SMS messenger, allowing them to send SMS messages without the need to request any special permission. In this case, the user will be notified about each SMS message that is sent. Other permissions are less obvious in terms of their potential for malicious abuse (like the permission to access the Internet). Hence, future work is needed in order to derive meaningful permission sets that reliably indicate malicious behavior. Third, root exploits are used both

by authors of malicious apps and benign users in order to gain privileged access to a smartphone. Their analysis shows for at least 74% of a smartphone's lifetime, a working root exploit is available. Thus, approaches that tackle the field of smartphone security, especially in terms of integration them into existing IT infrastructures, should also aim to leverage network-based measures.

Chin et al. [56] investigate vulnerabilities of Android apps with a focus on inter app communication. They present ComDroid, a tool that performs static analysis of disassembled Android apps. Their findings show that many apps include components that are unintentionally accessible by other apps, enabling Broadcast injection and Activity hijacking attacks among others. This attack vector can be abused by malicious apps in order to employ attacks on the device against benign apps.

Fahl et al. [102] analyzed the 13,500 most popular, free Android apps. Their analysis focuses on how apps make use of the SSL/TLS protocols in order to protect data in transit. With their static code analyzer MalloDroid, they find severe programming flaws in the SSL/TLS code in 8% of the apps. A manual inspection of a subset of those apps showed that 41 of 100 inspected apps were vulnerable to Man-in-the-Middle (MITM) attacks. Thus, the authors were able to capture various sensitive data, including credit card numbers and login credentials. Their study shows that there is need for better education of developers and for better tools to support secure development of Android apps. Furthermore, it again proofs that companies who want to securely integrate smartphones within their IT infrastructure must be concerned about the threats imposed by third-party apps.

The recent work of Sanz et al. [103] aims at classifying apps into categories (like game, travel, etc.) by using machine learning techniques. They considered features that were directly obtained from the app's package (contained strings and permissions) as well as features obtained from the Google Play app store (rating, number of ratings and size of the app). They provide an empirical evaluation of several machine learning classifiers, including Bayesian Networks [104], Decision Trees [105], K-Nearest Neighbour [106] and Support Vector Machines [107], where Bayesian Networks perform best.

Zhou et al. [108] perform an evaluation of apps obtained from non-official, third-party app stores. They explicitly aim to detect malicious apps that have been created by repackaging benign apps. For this purpose, they have developed a system called DroidMOSS that can measure the similarity of Android apps. Their findings show that up to 13% of apps hosted in unofficial app stores fall into this category. Although the main focus

of such repackaged apps is to change the in-app advertisements in order to re-route advertisement revenues, the technique can also be employed for other malicious tasks like stealing sensitive information from the smartphone. Thus, the authors claim that there is a need to improve the vetting process for third-party app stores. When integrating smartphones into existing IT infrastructures, the knowledge whether a certain smartphone uses apps from unofficial app stores might be beneficial in order to reason about the devices security status.

The latest, large scale analysis of Android malware was conducted by Zhou et al. [109] as well. They were able to collect 1,200 malware samples between August 2010 and October 2011. Based on these collected samples, they provide an characterization based on aspects like installation methods, activation techniques, the concrete malicious payload and the permission use. Their findings show that malicious apps tend to request more permissions than benign apps (11 versus 4 in average). Furthermore, the majority of malware families (68%) were obtained from unofficial app stores. Repackaging is the predominant installation method for malicious apps (86%). Furthermore, more than one third (36,7%) try to employ privilege escalation attacks once installed. Based on their dataset, they evaluated several mobile anti-virus tools. Detection rates varied from 79,6% to only 20,2%. The authors conclude that there is a need for better anti-mobile-malware solutions. In addition, they made their dataset publicly available[13].

In order to examine the overall "health" of both official and unofficial Android app stores, Zhou et al. propose DroidRanger [110]. The term health refers to the ratio of benign to malicious apps that are provided within the respective app store. DroidRanger allows to classify apps as benign or malicious based on two schemes: (1) permission-based behavioral footprinting and (2) heuristics-based filtering. The first one aims to detect known malware by analyzing the app's requested permissions retrieved from the manifest file. Furthermore, they employ a static code analysis in order to detect suspicious API call patterns (like Broadcast Receivers whose Intent Filter allows to be notified about received SMS messages and that subsequently hinder the respective Intent from further dissemination by calling an appropriate API function (`abortBroadcast`). The second scheme aims to detect unknown malware (that is malware without any sample apps). This is done by using heuristics during the static code analysis of apps. The heuristics are based on the assumption that the dynamic loading and execution of code is a strong

[13] http://www.malgenomeproject.org/

indicator for a potentially malicious app. Apps that match the heuristics are further analyzed by dynamic execution and function call monitoring. Their system uncovered 211 malicious apps and two zero-day malware samples from 204,040 apps collected from May to June 2011. The infection rates of the analyzed app stores are generally low (ranging from 0,02% to 0,47%). However, this study proofs that unofficial app stores impose a higher threat level due to the fact that their infection rates are more than a magnitude higher (0,2% to 0,47%) compared to the official Android app store (0,02%).

A system that works very similar but specifically aims to find Zero-day malware for Android is RiskRanker [111]. Among the 118,318 apps collected from multiple Android app stores from September to October 2011, their system found 718 malicious apps, including 322 being zero-day.

Research in the field of app analysis strongly focuses on the Android platform. Probably because of its more open approach compared to other platforms. However, Egele et al. [100] conducted a study of iOS apps. They developed PiOS, a tool that aims to detect potential privacy leaks in Mach-0 binaries that were compiled from Objective-C code. They found that the majority of apps leak the smartphone's unique device identifier. However, they were not able to find instances of apps that secretly leak sensitive information that can directly be attributed to a person.

3.3.2 Attacks

Besides the analysis of smartphone platforms and their security mechanisms, a lot of effort has been put into performing actual attacks.

Taxonomy for Attacks

Vidas et al. [112] provide a survey on current attacks for the Android platform. They contribute a taxonomy for mobile attack classes, give concrete examples of attacks and present guidelines for mitigations where possible. They claim that one of Android's vulnerabilities is the combination of an open, less restrictive app store to purchase apps with the long patch cycle durations (as already introduced as Android Update Problem in Section 3.2). They classify attacks according to five classes, basically differentiating between unprivileged attacks carried out by fooling the user to install malicious apps, remote exploitation attacks that aim to get privileged access and attacks that need physical access

to the device. Concerning mitigations, among others a reduction of the patch cycle length and the usage of a Trusted Platform Module (TPM) [113, 114, 115] are proposed.

Sensory Malware

The first sensor sniffing attack was performed by Xu et al. [11]. They implemented SVC, the Stealthy Video Capturer for Windows Mobile. SVC captures video data which in turn can be sent to a remote party via email. While doing so, SVC behaves stealthy, limiting costly operations to a minimum. At that time, none of the established anti virus tools was able to detect SVC. This early work proofs that the built-in sensors of smartphones can be abused to compromise the user's privacy.

Schlegel et al. [10] have presented Soundcomber, a stealthy sensory Trojan for Android. Soundcomber can retrieve sensitive data such as credit card numbers and PINs when the user is interacting with a so-called interactive voice response (IVR) system and can send the gathered data to a remote system. It is composed of two colluding apps, one responsible for data collection and processing, the second responsible for data transmission to the attacker. Since both apps communicate on the device via covert channels, they have only a limited set of unsuspicious permissions on their own. Speech recognition and exfiltration of sensitive data is done on the device, thus limiting the amount of data that is transmitted to the attacker over the Internet. The authors discuss multiple defense mechanisms, including a more fine-grained permission model. The authors claim that monitoring the network traffic for anomalies will likely fail to detect Soundcomber, given the fact that it only transmits very few data over any communication channel.

The first malware for Android was developed by Schmidt et al. [116]. They abuse functions of the Android API that were undocumented in its early versions in order to execute native code on retail devices. Their attacks are based on Android's support for executing native code from the Java environment by using the JNI. This way, it was possible to bypass some parts of the Android permission system (for example the permission `BATTERY_STATS`). Although the authors state that their first findings are not critical, it proofs that the support of using native code from any third-party app is a potential attack vector.

Bickford et al. [117] investigate rootkits for smartphones. They perform exemplary attacks that allow an attacker to (1) snoop for sensitive data (conversations and geographic

location) and (2) to perform a DoS attack by depleting the phone's battery. Their proof-of-concept rootkits are targeting the Openmoko platform.

Distributed Denial of Service Attacks and Botnets

Liu et al. [118] were the first who discussed the potential of Distributed Denial of Service (DDoS) attacks performed by smartphones in detail. They describe how smartphones that are infected by malicious apps can be used in order to attack the public 911 emergency service. A similar study was done by Traynor et al. [119], focusing on the threats of mobile botnets. Furthermore, Singh et al. [120] investigate the feasibility to use Bluetooth as communication channel for botnet command and control on mobile phones.

Privilege Escalation Attacks

Davi et al. [9] have successfully performed privilege escalation attacks on Android. In general, this type of attacks circumvent Android's mandatory permission system by enabling unprivileged apps to use resources of the smartphone although they do not have sufficient permissions (like sending SMS messages). The proposed attacks are based upon the runtime compromise concept of return-oriented programming (ROP) without returns [121].

Egners et al. [122] recently presented a sequence of attacks that abuse vulnerabilities of the Android permission model. They allow them to establish a bidirectional communication channel to the Internet, without using the `INTERNET` permission. It is another example for privilege escalation attacks on Android.

Luo et al. [123] have identified that one critical attack vector for modern smartphones is their support of a technology referred to as WebView. It enables third-party apps to easily render and interact with content from web servers by using the HTTP protocol. The integration of JavaScript is supported as well. As already stated in Section 3.1, one important aspect of smartphones is that instead of using a general purpose browser for browsing the Internet, dedicated apps to interact with single web pages are common. The authors describe numerous attacks, either employed by malicious web sites or by malicious apps. Their work proofs that smartphones have a different attack surface compared to other computing platforms.

Orthacker et al. [124] provide an analysis of the Android permission framework. They especially describe the problem of permission spreading. The problem refers to the fact

that collaborating, malicious apps are able to disguise the total set of their granted permissions. In this case, the user can still reason about the subset of permissions that are requested by each app. However, it is not transparent for him whether installed apps collaborate with each other in such a way that functionality which is protected by a certain permission is exposed via custom interfaces or covert channels. In this case, an app could access a phone's resources although it does not have the required permissions. Permission spreading is especially useful to separate suspicious permissions from each other: For example, a third-party app that can both access the Internet and is able to read the user's private contacts would be considered to possibly leak private data. If the permissions are spread across two collaborating apps, this observation cannot be made anymore by the user. The authors provide a demo implementation of a permission spreading attack where two collaborating apps leak the phone's GPS position to a Twitter account. Their results show that apps can access phone resources although they do not have the necessary permissions.

Phishing

The threat of phishing attacks on mobile devices was first investigated by Felt and Wagner [125]. Their findings show that phishing attacks on Google Android and Apple iOS are feasible due to the lack of so-called application identity indicators. A user has little means to verify that the current screen belongs to an app that he considers to be trustworthy for a specific task (such as logging into his social network account). Since users are used to enter credentials as part of their normal workflow (for example when sharing information on social networks, when purchasing music or when updating already installed apps), it is easy to fool them to enter their sensitive information to a phishing app that fakes a benign app.

3.3.3 Countermeasures

Several countermeasures have been proposed in order to mitigate threats that smartphones have to face. Most approaches address mobile malware. In the following, countermeasures are presented and grouped according to their architecture as being either host-based, network-based or hybrid. Host-based approaches focus on extending existing smartphone platforms with additional security mechanisms. Network-based approaches aim to limit

the necessary changes that need to be made to an existing smartphone platform, while introducing new components whose functionality is accessible via the network. Hybrid approaches combine both host- and network-based ideas.

Host-based

Some major contributions have been made by members of the Systems and Internet Infrastructure Security Laboratory (SIIS)[14] in the department of Computer Science and Engineering (CSE) at Pennsylvania State University. Their work primarily aims at mitigating the threats that are introduced by third-party apps.

One of their first approaches is called the Policy Reduced Integrity Measurement Architecture (PRIMA) for the Symbian platform and was introduced by Muthukumaran et al. [126]. PRIMA enables to protect the phone's integrity by isolating trusted code (like an app for online banking) from untrusted code (like a game). This is achieved by enforcing a mandatory access control policy which is based on SELinux that encapsulates the allowed information flows between the phone's components (both trusted and untrusted). The term *reduced* refers to the fact that PRIMA uses a coarse-grained policy with only three types describing the levels of integrity (namely kernel, trusted and untrusted), whereas default SELinux policies leverage hundreds of types in order to achieve the least privilege principle. PRIMA implements the CW-Lite integrity model [127] which is less restrictive than the classical Biba model [128]. More precisely, CW-Lite requires filtering methods for processes that allow them to drop or upgrade their integrity level based upon the data they have read. For PRIMA this is necessary in order to support the installation of both trusted and untrusted apps while maintaining the desired isolation. The installer component is generally trusted. If it installs an untrusted app (like one that has not been digitally signed), the installer drops its integrity level to untrusted, with all the implications in terms of access to the phone's resources. PRIMA also supports to attest the enforced policy to a remote party.

Their first work that explicitly addresses the Android platform is the Kirin security service [129, 14]. Kirin aims to mitigate malware at install time by checking the respective app's security configuration against a predefined policy. The policy is like a blacklist that encapsulates undesired properties of third-party apps. If an app that is about to

[14] http://siis.cse.psu.edu

be installed matches one of these properties, the user is informed about the potential threat and the installation of the app is denied. The policies are written in the Kirin Security Language (KSL). The authors provide nine example policy rules that render undesired example properties. Each rule encapsulates a list of Android permission labels (like `SEND_SMS`) and Intent Filter action strings (like `CALL`) that are combined by logical `and` operators. According to their evaluation based on the example policy, five out of 311 apps from the official Google Play app store implement potentially dangerous functionality. This work proves that it is worth to take the security configuration of Android apps into account in order to determine the security status of a smartphone, especially in terms of mitigating malicious apps.

Ongtang et al. introduce the Secure Application INTeraction (Saint) framework for Android [15]. It addresses the fact that the Android platform does only provide very limited means to regulate the interaction between apps that are running on a smartphone based upon permission labels. However, application interaction is a core principle for the Android platform in order to reuse existing functionality: in order to make a phone call, an app developer would leverage the existing phone app by sending an appropriate `Intent` message rather than implementing the functionality on its own. Saint generally allows to specify two types of policies: install-time and runtime policies. The install-time policy allows to define under which conditions a permission label P defined by an app A is granted to another app B at install time. Once this permission label is granted, the respective app can make use of it at any time (for example in order to access the phone's microphone), without referring to the corresponding policy again. Run-time policies on the other hand allow to regulate the IPC that takes place at runtime. They allow to specify *access* rules for the caller and *expose* rules for the callee. The IPC is only allowed if all specified rules match. The concept is comparable to a stateful packet filter: the source and the destination of each rule are Android apps (or components thereof), combined with further conditions (like a minimum version of the destination component). A remarkable feature of these runtime policies is that it is possible to include conditions based on context information like location, time or the status of communication interfaces like Bluetooth. Furthermore, conditions can cover requirements related to the app developer's signature key.

The TaintDroid system is another approach to detect malicious third-party apps that was introduced by Enck et al. [16]. They adapted the idea of information flow tracking [130]

for the Android platform. TaintDroid extends the Android core components in such a way that allows tracking of privacy sensitive data (like location, audio, video and sensor data) on a smartphone through third-party apps. TaintDroid notices when sensitive data leaves the phone via untrusted apps and informs the user about it. Their results proof that leaking privacy sensitive data through such third-party apps is a real issue: 20 out of 30 popular apps available on the Google Play app store leaked data in an undesired way.

Besides the SIIS laboratory, other research groups have also suggested host-based approaches in order to improve the security of smartphones.

The problem that many third-party apps tend to leak private data of the user to a remote party [16] is also addressed by Hornyack et al. [131]. To counter this threat, they propose an extension to the Android platform called AppFence. It basically adds two benefits. First, when apps request access to sensitive data on the device, users can choose to provide faked, so-called shadow data instead. This may break the correct functionality of some apps that require valid sensitive data (such as the users contact list). Thus, AppFence also supports to grant apps access to sensitive data, but prevents the data from leaving the device over any communication channel. This functionality is referred to as exfiltration blocking. In order to implement the additional security checks, the Android platform must be extended. However, there is no need to modify third-party apps to work on a device that uses AppFence.

Nauman et al. suggest Apex, a new policy enforcement framework for Android [132]. It addresses one major drawback of Android: the "all-or-nothing" decision that a user must make when a third-party app is installed. As already stated, the term "all-or-nothing" refers to the fact that a user must either grant all permissions that the app requests, or to decide to deny all of them, and thus abort the installation completely. Apex enables the user to make more fine-grained decisions under which circumstances permissions shall be granted to apps. Apex basically improves the Android framework in three ways: (1) at install time, the user can choose to grant only a subset of the permissions requested by the app, (2) at runtime, the usage of resources can be restricted based upon the phone's context (such as its location), (3) the access to resources protected by permissions can also be restricted based on the app's behavior and its current state (enabling policies that cover aspects like a maximum amount of SMS messages an app is allowed to send). The authors present a formal model for Apex policies and furthermore mention a prototype implementation of their approach.

Conti et al. [38] propose CRePE, a system that enables context-related enforcement of fine-grained policies for Android. A context is defined by a set of variables (like location or time), the presence of other devices, specific user interactions or a combination thereof. A policy consists of rules that allow or deny access to resources. Resources are either apps or system services. Policies can be defined both locally by the user and remotely by a trusted third party and are always associated with a specific context. If the context is active (such as the phone is within a specific location), the corresponding policy is enforced. CRePE enables use cases such as a company that enforces a restricted set of available apps for their employees when they are working. Their prototype implementation imposes negligible time overhead (at most 0.6 ms for additional permission checks) but a considerable overhead in terms of energy consumption (50% to 100% for permission checks and context observation).

Ongtang et al. [133] introduce Porscha, a Digital Rights Management (DRM) extension for Android. They address the drawback that today's smartphones, including Android based phones, provide almost no means to enforce DRM policies on content that is delivered to the phone. Porscha supports to enforce policies within two separate phases: (1) when the content is in transit, which means when it is delivered to the respective phone and (2) when the content is located on the platform. Porscha allows to bind content both to a particular phone as well as to a set of endorsed apps on the phone. Furthermore, the use of delivered content can be constrained (for example allowing to play a video within 48h of the purchase date). The authors focus on SMS, Multimedia Messaging Service (MMS) and Email as content providing communication channels. Content in transit is protected by Identity-based encryption [134], an asymmetric encryption scheme. Content on the phone is protected by adding a Porscha mediator to the Android middleware that acts as reference monitor which enforces the content policies. Porscha's DRM mechanism could be leveraged in an enterprise environment in order to protect sensitive data.

Felt et al. [17] discuss the impact of permission re-delegation attacks for modern smartphone platforms. Permission re-delegation occurs when an unprivileged app accesses protected resources without the necessary permissions by abusing another, privileged app's vulnerable interface. It is a special case of a confused deputy attack [86]. Their survey of popular Android apps shows that more than one third of the considered apps are vulnerable to such attacks, including even core system apps that are directly shipped with the smartphone. The authors also propose a possible defense mechanism called IPC Inspec-

tion. When a communication is initiated between a caller and a callee, IPC Inspection reduces the permissions of the callee to the intersection of the caller's and the callee's permission set. Thus, permission re-delegation attacks can be prevented. However, this approach places a burden on genuine apps to correctly request the permissions that they use indirectly via deputies.

Quire [135] is another security extension for Android. Similar to IPC inspection introduced before, Quire aims to prevent confused deputy attacks. It keeps track of the IPC call chain, allowing an app to drop its privileges to those requested by the calling app. Furthermore, Quire enables the called app to reason about the complete call chain of preceding apps, instead of only seeing the last calling app. One major drawback of the approach compared to others like IPC inspection is the fact that apps need to be recompiled in order to support Quire.

Xie et al. [136] propose pBMDS, a behavior-based malware detection system. Their approach aims to correlate user input characteristics (such as touchscreen usage) with system calls at the kernel level in order to detect malicious behavior. pBMDS is a host-based, probabilistic approach that leverages Hidden Markov Models in order to learn normal user and system call behavior. It was primarily designed to detect malware that propagates through MMS/SMS and Bluetooth. Their prototype implementation is based on the Openmoko platform. Modern platforms such as Google Android are not specifically addressed. The authors claim that their approach was one of the first that introduces artificial intelligence (AI) for smartphone security.

Zhou et al. [137] address the threat of apps that leak sensitive, personal information of the user to a third party. They argue that modern smartphones need a privacy mode that enables the user to enforce fine-grained policies regarding which information can be used by which apps. They present TISSA, a prototype of their system for the Android platform. TISSA supports to specify the availability of sensitive information such as location, contacts and the phone's identity (IMEI) on a per app basis. For each type of sensitive information, TISSA can be configured to either allow the access, return anonymous or fake results or simply return nothing. The last option however will likely cause many installed apps to not work properly. Although the approach is sound, it is questionable if users really have the passion to maintain such policies on a per app basis.

Shabtai et al. [18] propose Andromaly, a lightweight, host-based IDS for Android-based devices. In order to detect malicious apps, their framework is able to collect 88 features

with different semantics (such as CPU Usage or Incoming SMS). Furthermore, it supports different algorithms for both the selection of features (i.e. those that are used for further analysis) and the detection of malicious apps (such as k-means and Bayesian Networks). Their empirical findings show that machine learning techniques are a viable approach for the detection of malicious software on Android powered devices. They conclude that a reasonable enhancement would be to combine anomaly detection algorithms and misuse-based detectors (such as rule-based approaches).

Bugiel et al. [138] propose XManDroid, a security framework for Android that is capable of detecting and preventing application-level privilege escalation attacks at runtime. The authors particularly address the threat of sophisticated malware apps that use advanced techniques such as covert channels in order to hide their malicious functionality (like Soundcomber [10]) as well as confused deputy attacks where vulnerable interfaces of genuine apps are exploited [86, 135]. XManDroid extends the Android reference monitor to enable runtime monitoring of communication between apps. Based on a system policy which is consulted when a particular interaction between two apps is requested, these communication links can either be allowed or denied. The system policy encapsulates rules that are based on previous work done by Enck et al. [129, 14]. However, XManDroid's policies are not solely restricted to permissions. Instead, they also incorporate the concepts of ICC content inspection and user confirmation. This allows more fine-grained policy rules such as to allow the sending of text messages depending on the confirmation of the user. XManDroid is an extension of the Android middleware and thus can only detect malicious behavior that actually uses Android's ICC mechanisms. This drawback is addressed in a subsequent paper, where a kernel-level module is added to the XManDroid approach [139]. Although they provide seven example rules, the engineering of reasonable security policies remains a challenging task for future work.

Bugiel et al. [62] also propose TrustDroid, a security framework for Android that enables to isolate installed apps based on their trustworthiness. Their motivating example is to separate private apps from those that are used for business tasks, preventing unauthorized communication and data access between them. TrustDroid adds security checks at several layers of the Android stack: (1) the middleware layer, (2) the kernel layer and (3) the network layer. Context-related policies are supported as well, enabling use cases such as to prevent untrusted apps from using the Internet while an employee's smartphone is connected to the company's IT infrastructure. Their trust model is similar to the one

defined in Section 2.4. They provide a prototype implementation and proof that their approach imposes negligible overhead in terms of battery consumption. Thus, TrustDroid is well suited for isolating apps based on their trustworthiness, limiting the potential damage that malicious apps can cause. Especially within a corporate scenario, TrustDroid could be beneficial. However, the approach does not include any detection capabilities. Furthermore, it requires extensive changes to the Android platform.

Pearce et al. [140] have conducted a study of the Google Play Store in order to investigate how advertising libraries (also referred to as ad libraries) impact the permission usage of third-party apps. Advertising is a crucial part of the Android ecosystem. As many apps are available for free, ad libraries are a convenient way for developers to gain revenue by displaying in-app advertisements. They just have to bundle an appropriate ad library together with their app. A side effect is that many apps tend to request privacy sensitive permissions only because of the bundled ad library (for example in order to obtain the location of the smartphone). This is a clear instance of the over declaration problem introduced before. In order to address this problem, the authors propose AdDroid, a new advertisement framework for Android. It introduces a new API and a set of new permissions, enabling third-party developers to display advertisement within their apps without requesting otherwise unnecessary and privacy sensitive permissions. Another study dealing with the impact of advertisement libraries that yield similar results was also conducted by Grace et al. [141].

Network-based

Cheng et al. [142] present SmartSiren, a collaborative virus detection system. They address the threat of mobile viruses that infect smartphones and try to spread themselves by abusing the phone's communication capabilities (for example SMS/MMS, Bluetooth and IP-based communication). Their system includes an agent on the smartphone that monitors the device's communication activities, creates appropriate reports and sends them to a remote server. Monitoring and detection capabilities are limited to SMS/MMS messages, emails or messages sent via Bluetooth. The remote server tries to detect the presence of viruses by processing the reports received from the agents. SmartSiren generally supports three detection strategies: (1) performing statistical calculations on the overall amount of traffic (moving average), (2) counting the number of messages for each

destination in order to detect highly frequented destinations and (3) the use of fake contacts added to the smartphone's address book. The last detection method is used in order to detect viruses that try to disseminate themselves by a brute force strategy. Upon detection, alerts are sent both to the infected smartphone as well as to smartphones that are logically connected (via contact lists) or physically connected (that is via proximity) to the infected device. This early work proofs that network-based approaches for mobile malware detection are feasible.

The early work of Kim et al. [143] addresses mobile malware that targets the depletion of battery energy. They propose a framework consisting of two components (1) a power monitor that measures the power consumption on a mobile device and (2) a data analyzer that generates power signatures based on the taken measurements. These signatures are compared against a priori defined energy consumption profiles that express normal behavior. The approach of Buennemeyer et al. [144] works similar. In contrast to the approach that is developed within this thesis, they solely rely on the power consumption and do not consider any further aspects of smartphones. Furthermore, these early approaches can easily be circumvented if (1) mobile malware does not use a lot of resources or (2) mobile malware is context-aware, thus performing costly tasks only at times when the smartphone's battery is currently charged.

Oberheide et al. [145] were one of the first that introduce the concept of providing cloud security services for smartphones in order to circumvent their resource constraints. They leverage their CloudAV malware detection engine and extended it with a mobile-specific behavioral detection engine. This engine performs dynamic analysis of apps based on system calls. Furthermore, they envision that their approach can offer more sophisticated security services like SMS spam filtering that go beyond the classical anti virus detection. Their prototype is realized based on Nokia smartphones.

The Paranoid Android system is proposed by Portokalidis et al. [7]. Their idea is to host virtual replicas of the smartphones on remote servers. Based on these virtual replicas, various security checks are performed. In order to establish security measures that are independent of the smartphones resource constraints, the authors envision to provide security in terms of attack detection as a cloud service. A monitoring component on the smartphone, called tracer, gathers execution traces which are then send to a component located on the remote server, called replayer. The execution trace covers all necessary information to replay the execution that has taken place on the smartphone within its

virtual replica (such as system calls that pass data from kernel to user space or operating system signals). The authors propose four categories of security methods that can be employed on the remote server: (1) dynamic runtime analysis, (2) anti virus scanning, (3) memory scanners and (4) system call anomaly detection. A prototype for the Android platform is also presented which exemplary implements two security methods: Dynamic Taint Analysis [146] (which falls in category 1) and a ClamAV anti virus scanner (belonging to category 2). The necessary execution trace is recorded by leveraging the Linux ptrace system call. An evaluation of their prototype shows that their approach is feasible in terms of introduced processing and data transmission overhead. The authors claim that their approach is also capable of detection zero-day-exploits. However, they do not provide concrete examples for detected attacks.

An approach to monitor Symbian OS smartphones for remote anomaly detection was introduced by Schmidt et al. [147]. Their architecture consists of a monitoring component located on a phone and a remote server that hosts the anomaly detection components. The monitoring component sends the feature vectors to the remote server, which in turn can leverage various methods from the field of artificial intelligence like Self-Organizing Maps (SOM) [88] in order to detect anomalies. The proposed features that are extracted include system properties like the amount of free RAM, usage properties like whether the user is currently inactive and smartphone properties like the amount of sent SMS messages. In order to capture the normal behavior of a phone, the authors specified 40 use cases (such as playing a specific game on the smartphone) with corresponding testing protocols. The resulting feature vectors where aggregated and considered as being the normal behavior. This way, they were able to proof quiet obvious facts, like that the CPU usage is significantly higher when a game is played compared to when the user writes a SMS message. In order to learn abnormal behavior, they monitored the features when executing a known malicious app developed by Jamaluddin et al. [74]. This malware sends an SMS every time the key “2” is pressed, which caused an abnormal increase of the SMS_SENT_COUNT feature.

Another approach to detect malware on Android is the Android Application Sandbox System (AASandbox) proposed by Bläsing et al. [148]. Besides static analysis features, it also supports dynamic analysis of apps at runtime. Static analysis is done prior to the installation of an app by decompressing and disassembling it. Dynamic analysis is performed by leveraging the Android emulator in an isolated environment. The system

is intended to be provided as a cloud service. AASandbox processes Android `.apk` files and performs a two step detection approach: first, a simple static pre-check is performed which basically scans the decompiled Java bytecode for suspicious patterns (like usage of Java Native Interface to load native libraries or the presence of Java code that is capable to spawn native child processes). Afterward, the dynamic analysis component monitors the behavior of the respective app within an Android emulator. The component basically logs all system calls that take place to a separated log file. The authors evaluate their approach based on a sample malicious app which they implemented on their own. The malware basically launches a Denial of Service attack targeting the smartphone itself by continuously creating new child processes. The static analysis component detects the suspicious Java instructions that spawn the child processes. The dynamic analysis component in turn detects that the behavior of the malicious app is suspicious compared to the normal behavior of third-party apps. However, the authors do not provide suggestions for concrete machine learning techniques in order to process the dynamically observed data set.

Burguera et al. [149] propose the Crowdroid system for dynamic, behavior-based malware detection on Android powered smartphones. They specifically aim to detect trojan horses that infect benign apps in order to spread themselves (like DroidDream in March 2011). Thus, they are focussing on finding apps that have the same name and version, but behave differently due to the added malicious code. The Crowdroid client monitors system calls on an Android device and sends the collected traces to a remote server. The remote server then creates system call vectors based on the received traces. Afterwards, partitional clustering by leveraging the k-means algorithm is applied on the system call vectors in order to distinguish between benign and malicious samples. Their detection rate of self-written malware was 100% and 85% to 100% against malware that was seen in the wild. Aside from solely focusing on system calls as only features, the major limitation is that Crowdroid can only detect malware samples that have a matching *goodware* sample (that is a benign app).

Hybrid

One of the first papers on intrusion detection for smartphones that follow a hybrid approach was conducted by Miettinen et al. [95]. They propose a framework that leverages

both host-based and network-based detection methods. An IDS module on the smartphone obtains relevant features and raises alerts if intrusions are detected. These alerts are forwarded to a network-based IDS module, which in turn correlates them with other alerts that have been observed due to monitoring the network traffic. Besides the architecture of the framework, only little detail is given on the smartphone features that should be used for intrusion detection. The authors provide three general categories of features (operating system events, measurements and application level events) and give few examples of features that have been used in the past on commodity PC platforms (such as system calls and CPU usage). However, in contrast to this thesis, they do not provide a detailed analysis of features that should be obtained on smartphones. Furthermore, no approach is presented in order to realize the network-based correlation engine. Their work does not address any particular smartphone platform at all.

Schmidt et al. [150] introduce the concept to use static analysis of executable for collaborative malware detection on Android. Their system provides three main services: (1) on device-analysis (2) collaboration and (3) remote analysis. The on-device analysis component is responsible for extracting system and library calls from binaries by leveraging the readelf command and can perform first analysis methods. When on-device analysis is not feasible, the data can also be sent to a remote server which then performs the analysis. Furthermore, the collaboration function enables to even share analysis results directly between smartphones. In order to create reference data for a benign smartphone, the function calls of all Executable and Linking Format (ELF) compliant Linux system commands available on Android were extracted. Accordingly, to create malicious reference data, a set of known malware for Linux was processed in the same way. Although not specifically targeted for Android, the authors argue that the malware examples would only show minor differences when cross-compiled to Android's ARM architecture, and thus form a reasonable basis for malicious reference data. Based on the readelf output, attribute sets were defined that basically encapsulate the names of the system functions that were called. The data was then processed with three different classifiers: PART [151], Prism [152] and a modified version of the Nearest Neighbor Algorithm. Their findings show that classifying malware based on function calls is feasible. The classifiers in use performed well, with high detection and low false positive rates.

Nauman et al. [153] introduce an approach to enable the concept of remote attestation as defined by the TCG for the Android platform. They leverage the Integrity Measurement

Architecture (IMA) for Linux in order to bootstrap a chain of trust from the kernel level. Based on this chain of trust, they support two attestation schemes. The first one supports to attest the integrity of complete apps. The second one enables the more fine-grained attestation of individual class files. The approach can be used by administrators of an IT infrastructure in order to obtain the software configuration of connected smartphones, secured by leveraging the capabilities of a TPM. However, it requires extensive modifications to the Android platform itself. Furthermore, the authors do not address the problem how informed decisions can be made based on the attested software configuration, which is a fundamental challenge for all attestation approaches.

The number and variety of presented approaches proofs that smartphone security has become and still is a hot topic in the research community. The focus of the presented publications ranges from survey and analysis papers over exemplary attacks to countermeasures that aim to mitigate identified threats. An assessment of the discussed countermeasures is provided in Section 3.5.

3.4 The IF-MAP Protocol for Network Security

In the following, the IF-MAP protocol for network security is introduced. Its major strength is the ability to integrate existing security and management systems, enabling them to share data about the network in order to employ collaborative efforts for mitigating potential threats. Although IF-MAP is not related to the field of smartphone security, it will be used in the remainder of this thesis to implement the CADS approach which is presented in Chapter 4. Details why IF-MAP is suitable for implementing the CADS approach are discussed in Section 5.1.

The term IF-MAP refers to a set of specifications that were published by the Trusted Computing Group (TCG) as part of the Trusted Network Connect (TNC) framework. TNC defines and open set of standards and protocols for building interoperable Network Access Control solutions. IF-MAP defines a network protocol for exchanging so-called metadata among an arbitrary number of MAP clients (MAPCs) via a central MAP server (MAPS) in real-time. The main motivating scenario for IF-MAP is to distribute security related metadata between components within a network in a standard and thus interoperable way. The term metadata with respect to IF-MAP refers to data that describes the overall status of the network, including attached devices and their users. Since the speci-

fications include a flexible extension mechanism, IF-MAP can be customized to virtually any scenario even beyond the classical network security domain.

IF-MAP in its latest version 2.1 is specified by two types of documents: One document defines the core data model, the basic operations that MAPCs and the MAPS must support and their encapsulation within SOAP [20]. The second type of documents specifies metadata for specific domains. For example, there is currently one dedicated specification that addresses metadata for the domain of network security [154] and another one that addresses security in the domain of industrial control systems (ICS) [155]. Thus, it is easy to integrate metadata for further domains, without the need to change the core protocol. A specification that addresses the domain of smartphone security does not exist yet.

3.4.1 TNC Architecture

The TNC architecture is depicted in Figure 3.2. The architecture is organized in five columns, each one representing a logical role:

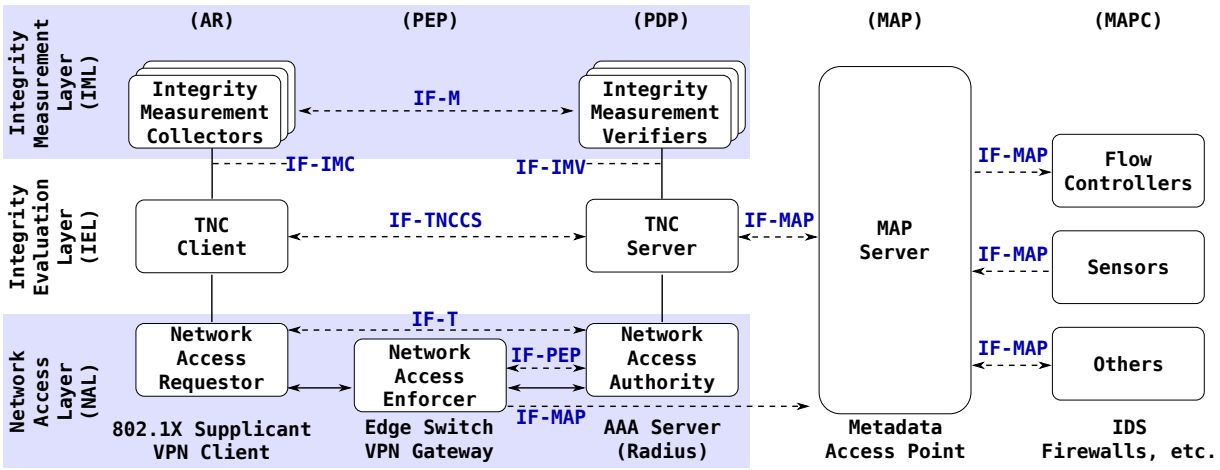

Figure 3.2: TNC Architecture [36].

1. An *Access Requestor (AR)* represents an endpoint that wants to get access to a TNC protected network.

2. A *Policy Decision Point (PDP)* is located within the protected network and is responsible for authenticating endpoints that try to access the network. This usually

includes an authentication of the user as well as a check of the endpoint's current integrity state (that is its software configuration).

3. A *Policy Enforcement Point (PEP)* is located at the edge of a protected network. The AR tries to access the protected network via the PEP. The PEP is responsible for enforcing the access decision of the PDP.

4. A *Metadata Access Point (MAP)* is responsible for storing and providing state information about ARs (such as current integrity state, IP Address, authenticated user) and other components of the TNC protected network. This state information is generally referred to as metadata. It can be used for further policy decision making and enforcement. The specification defines the role simply as MAP. However, in order to emphasize that a MAP provides server functions to store and retrieve arbitrary metadata, it is also referred to as Metadata Access Point Server (MAP Server, or simply MAPS).

5. A *MAP Client (MAPC)* is able to publish metadata to and receive metadata from a MAP Server. Examples for MAPCs include sensors like Intrusion Detection Systems (IDS) that publish metadata that describes security alerts.

The PDP and the PEP can act as MAPCs as well. For example, a PDP that supports IF-MAP will publish metadata to the MAPS for each user and each device that is authenticated by means of TNC. The logical roles are further subdivided in several components that perform a specific task within the TNC framework. Three layers group components that provide similar functionality. Dashed lines depict the interfaces between those components that are specified within the TNC framework.

3.4.2 Data Model

The data model of IF-MAP is represented by an undirected graph where cycles and loops are allowed. It is composed by three types of components:

1. *Identifiers* are represented by the nodes of the graph. Each identifier belongs to a certain type as specified by an XML schema document. The type limits the potential values of an identifier. There are five basic types of identifiers:

a) `ip-address` to represent an IP address that is currently used within a network.

b) `mac-address` to represent a MAC address that is currently used within a network.

c) `identity` primarily used to represent users who have been authenticated within a network.

d) `device` to represent devices (servers and endpoints) that are connected to a network.

e) `access-request` to represent a request for access to a network that was issued by an endpoint.

2. *Links* are represented by the edges of the graph. A link establishes an undirected, bi-directional relationship between two identifiers.

3. *Metadata* is specified by XML schema documents. Metadata can be attached both to single identifiers or to links that connect two identifiers.

Depending on its actual type, metadata should or must be attached to a specific type of identifier or on a link between two identifiers of a specific type respectively. For example, one standard type of metadata is `ip-mac`. It can be attached on a link between an `mac-address` and an `ip-address` identifier. This would express the fact that one device is using the associated MAC and IP addresses (for example as provisioned by a DHCP server). An example of an IF-MAP graph is depicted in Figure 3.3.

3.4.3 Communication Model

IF-MAP is a content-based publish-subscribe network protocol. In essence, a MAPC and a MAPS exchange XML documents encapsulated within SOAP over HTTPS [156]. Thus, the protocol is secured by means of TLS [157]. MAP clients and MAP servers must mutually authenticate themselves. MAP clients must verify a MAP server's certificate and determine whether it is trustworthy or not. MAP servers in turn must authenticate MAP clients either by (1) verifying a client's certificate as part of the TLS handshake or (2) by employing password based basic authentication as described in RFC 2617 [158] after the TLS handshake was finished.

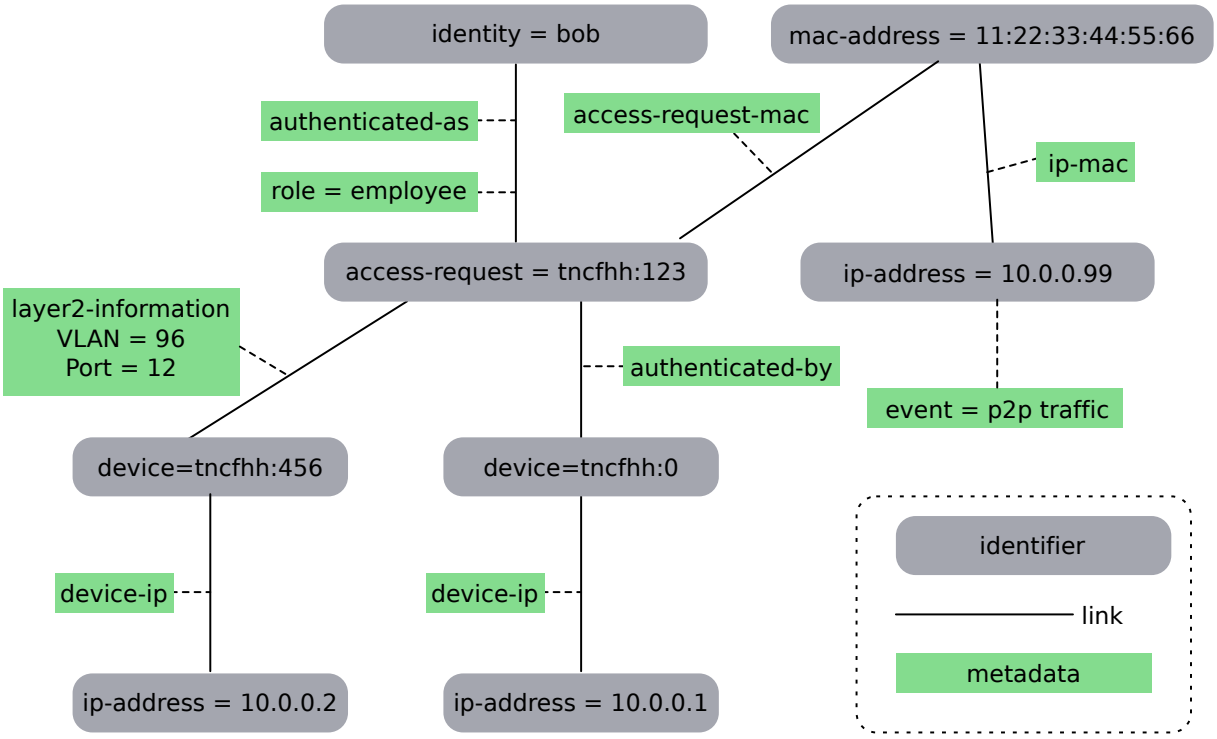

Figure 3.3: Example of an IF-MAP graph.

IF-MAP follows the request-response paradigm. A MAPC initiates the communication by sending a request that expresses an IF-MAP operation. The MAPS performs the requested operation and answers with an appropriate IF-MAP response, either immediately or with a certain delay. The following operations are supported:

- The *publish* operation is used by a MAPC to create, change or delete metadata stored in a MAPS.

- The *search* operation is used in order to search the current IF-MAP graph that is stored in the MAPS. A single identifier must be specified that represents the root of the search, that is where the search starts. The search algorithm can be implemented either by following a depth-first or a breadth-first strategy. A couple of parameters can be specified to customize the way the IF-MAP graph is traversed during the search. This especially includes the maximum depth to which the graph is traversed (relative to the root identifier). Furthermore, it can be specified that certain links

in the graph are only traversed if they have a certain type of metadata attached to them (also referred to as `match-links` filter).

- The *subscribe* operation is similar to the search operation. In fact, the syntax of the operation is basically the same. It enables a MAPC to get notified immediately when changes to the IF-MAP graph in the MAPS are made that match one of his subscriptions. A MAPC can hold numerous subscriptions to one MAPS at the same time. Basically, those subscriptions work like stored search operations.
- The *poll* operation is used by a MAP client to get notified as soon as changes to the IF-MAP graph are made that match one of his previously issued subscriptions. That is, the *poll* operation is a blocking operation. It returns in the event that a part of the IF-MAP graph that is covered by the MAP client's subscriptions is changed. The *subscribe/poll* operations allow a MAP client to specify which metadata it is interested in and to get notified immediately after the respective metadata has been changed. Once a *poll* operation returns, the MAP client can process the received metadata. Afterwards, it can issue another request that contains a *poll* operation in order to get notified when the IF-MAP graph changes again.

IF-MAP distinguishes between two types of channels: a *Synchronous Send and Receive Channel (SSRC)* and an *Asynchronous Receive Channel (ARC)*. The latter one is only used for the blocking poll operation. Any other IF-MAP operation is issued over the SSRC. Further details regarding the IF-MAP protocol and its operations are provided by the respective specification [20].

3.5 Assessment

In the following, the approaches in the field of smartphone security that have been presented in Section 3.3 are assessed regarding the requirements that have been identified in Section 2.5. Approaches that solely focus on exemplary attacks or analysis of smartphone-specific threats are omitted. The purpose of this assessment is twofold:

- First, general findings are discussed based on the presented approaches. The purpose is to clarify the key aspects that have been addressed by the research community in the past.

- Second, the presented approaches are compared to the requirements defined in Section 2.5. The goal is to analyze to what extend existing approaches are feasible to fulfill the list of requirements.

The approaches that have been presented so far are summarized in Table 3.2. It is subdivided into two parts:

- The left part summarizes general information of the respective approach. It includes the name (or the name of the author), the focus of the approach, the architecture it is based on and the platform it aims at. Architecture-wise, the approaches have been classified as being host-based (H), network-based (N) or hybrid (X) or offline/out-of-band (-). The last category encapsulates approaches that aim to enhance components of a smartphone's ecosystem like the platform's app store. Relevant platforms include Android (A), iOS (i), Openmoko (O), Symbian (S), Windows CE/Mobile (W) and Maemo (M).
- The right part depicts to what extent each of the presented approaches fulfill the requirements defined in Section 2.5. The fulfillment grade is classified into three categories: complete (+), partial (o) and not fulfilled/addressed (-).

Table 3.2: Comparison of previous work in the field of smartphone security regarding the requirements defined in Section 2.5. The focus is on related research approaches.

Name/Author	Focus of Approach	Architecture	Platform	Unwanted Configurations	Abnormal Behavior	Context-based Detection	Policy-based Reaction	Dynamic Analysis	Extensibility	Integration Capabilities
Stowaway [90]	Detect overdeclaration of apps	-	A	o	-	-	-	-	o	o
Woodpecker [94]	Detect permission leaks of apps	-	A	o	-	-	-	-	o	o
PiOS [100]	Detect information leaks of apps	-	i	o	-	-	-	-	o	o
ComDroid [56]	Detect permission leaks of apps	-	A	o	-	-	-	-	o	o
MalloDroid [102]	Detect SSL/TLS vulnerabilities in apps	-	A	o	-	-	-	-	o	o
Sanz et al. [103]	General classification of apps	-	A	o	-	-	-	-	o	o
DroidMOSS [108]	Detect repackaged apps	-	A	o	-	-	-	-	o	o
DroidRanger [110]	Examine overall health of app stores	-	A	o	-	-	-	-	o	o

Name/Author	Focus of Approach	Architecture	Platform	Unwanted Configurations	Abnormal Behavior	Context-based Detection	Policy-based Reaction	Dynamic Analysis	Extensibility	Integration Capabilities
RiskRanker [111]	Detect zero-day malware	-	A	o	-	-	-	-	o	o
SORBET [85]	Prevent confused deputy attacks	H	A	-	-	-	-	+	-	-
PRIMA [126]	Isolate trusted and untrusted code with SELinux based mandatory access control	H	S	o	-	-	o	+	-	-
Kirin [129, 14]	Detect suspicious apps at install time based on static properties	H	A	o	-	-	o	+	-	-
Saint [15]	Enforce inter-app communication security policies	H	A	o	-	o	o	+	-	-
TaintDroid [16]	Detect information leaks of apps	H	A	-	o	-	-	+	-	-
Apex [132]	Fine-grained, context-aware runtime permissions	H	A	o	-	o	o	+	-	-
CRePE [38]	Context-related policy enforcement	H	A	o	-	+	o	+	-	-
Porscha [133]	Digital Rights Management	H	A	o	-	+	-	+	-	-
IPC Inspection [17]	Prevent confused deputy attacks	H	A	o	-	-	-	+	-	-
Quire [135]	Prevent confused deputy attacks	H	A	o	-	-	-	+	-	-
pBMDS [136]	Behavior-based malware detection	H	O	-	o	-	-	+	-	-
TISSA [137]	Prevent information leaks of apps	H	A	o	-	-	-	+	-	-
Andromaly [18]	Behavior-based malware detection	H	A	-	+	o	-	+	-	-
XManDroid [138, 139]	Prevent application-level privilege escalation attacks	H	A	o	-	-	o	+	-	-
TrustDroid [62]	Isolate apps based on their trustworthiness	H	A	o	-	o	o	+	-	-
AppFence [131]	Prevent information leaks of apps	H	A	o	-	-	-	+	-	-
AdDroid [140]	Prevent information leaks of advertisement libraries	H	A	o	-	-	-	+	-	-
Dixon et al. [79]	Location-based detection of power anomalies	N	A	-	o	o	-	+	o	o
Kim et al. [143]	Detection of power anomalies	N	S	-	o	-	-	+	o	o
Buennemeyer et al. [144]	Detection of power anomalies	N	W	-	o	-	-	+	o	o
SmartSiren et al. [142]	Behavior-based malware detection	N	W	-	o	-	-	+	o	o

Name/Author	Focus of Approach	Architecture	Platform	Unwanted Configurations	Abnormal Behavior	Context-based Detection	Policy-based Reaction	Dynamic Analysis	Extensibility	Integration Capabilities
CloudAV [145]	Cloud-based anti virus detection	N	M	o	o	-	-	+	o	o
Paranoid Android [7]	Behavior-based malware detection	N	A	o	o	-	-	+	o	-
Schmidt et al. [147]	Behavior-based malware detection	N	S/W/A	-	o	-	-	+	o	o
AASandbox [148]	Static and dynamic analysis to detect malware	N	A	o	o	-	-	+	o	o
Miettinen et al. [95]	Position paper IDS for smartphones	X	-	-	-	-	-	-	-	-
Crowdroid [149]	Behavior-based malware detection	X	A	-	o	-	-	+	o	o
Schmidt et al. [150]	Collaborative malware detection via static analysis of apps	X	A	o	-	-	-	+	o	-
Nauman et al. [153]	Remote Attestation	X	A	+	-	-	-	+	o	-

3.5.1 General Findings

Based on the related work that has been discussed so far, the following general findings can be made.

Host-based security extensions are predominant Most of the approaches that have been proposed recently focus on host-based extensions for certain smartphone platforms. Out of the 38 approaches listed in Table 3.2, 17 implement a host-based architecture. Only 8 implement a network-based architecture and 4 follow a hybrid approach combining host- and network-based mechanisms. Since the majority of security extensions aim to prevent some type of attack, it is reasonable to follow a host-based approach that extends a concrete platform with the appropriate additional security mechanisms. The distribution also proofs that the question how IT infrastructures can be protected from threats introduced by smartphones has not gained a lot of attention. Recently, several approaches have been proposed that do not match the classification as being host-based, network-based or hybrid (9 out of 38). Instead, these approaches aim to improve aspects

of the ecosystem of smartphones, namely the platform's app stores. The primary goal of these approaches is to analyze third-party apps in order to detect malware.

Android related research is predominant Early approaches have focused on the Symbian platform. However, since the first release of Android in 2008, research focuses on Android as smartphone platform. iOS is only addressed by one of the presented approaches. The same is true for the outdated Openmoko and Maemo platforms. Although most of the developed concepts are platform independent, researchers tend to built their prototypes on Android. One main reason is the accessibility and openness of the platform, enabling researchers to add new security features based on the officially available source code. Furthermore, the Google Play app store for Android is more opened compared to its competitors, easing the deployment of apps that are subject of research.

Countermeasures focus on privilege escalation attacks, privacy leaks and malware The countermeasures that have been developed thus far focus on three security problems of modern smartphones.

1. To prevent privilege escalation attacks. Especially the Android platform is prone to this kind of attacks due to the design of its permission model.

2. To prevent leakage of sensitive data (private contacts, messages, the users location) to remote third parties. This problem is especially important since it is not restricted to malicious apps. Also benign apps tend to leak sensitive data, either on purpose or by accident.

3. To detect and mitigate malware. Third-party apps have been identified as the major threat for modern smartphones. Thus, many approaches focus on the problem to effectively detect malicious apps. Some of them even focus on the detection of malware that exploits zero-day vulnerabilities.

Commonly known security mechanisms are applied to smartphones Although smartphones have special characteristics that need to be taken into account such as limited battery power, mobile usage, and app-based architectures, many approaches are based on known techniques that are adapted for smartphones. This especially include work that

aims to detect malicious apps. These approaches are based on the static analysis of binaries or the dynamic behavior of apps. The same techniques have also been used in order to detect malicious software on other platforms. However, the main challenge is to determine relevant features that are suitable for the specific problem. For example, which features should be considered in order to distinguish malicious from benign apps.

Smartphone usage patterns are not well understood The question how users preferably use their smartphone, in terms of what tasks they perform under which circumstances, is only addressed by very few approaches. However, this knowledge can be beneficial for various use cases, including smartphone security. Regarding the given scenarios defined in Chapter 2, it is a precondition to know the normal behavior of a smartphone in a certain IT infrastructure before any anomalies can be detected. If it is known how users preferably use their smartphones in a given IT infrastructure, this knowledge can be used to actually develop models that express the normal behavior of smartphones.

3.5.2 Fulfillment of Requirements

In the following, the discussed approaches will be compared to the requirements identified in Section 2.5.

R-01 Detection of unwanted and malicious configurations of smartphones The majority of the presented approaches fulfill this requirement partially. This is because their focus lies on the detection of malicious apps and on the prevention of certain types of attacks (primarily privilege escalation). The fact that benign apps can be unwanted within an IT infrastructure under certain circumstances is not properly addressed by these approaches. Furthermore, they do not cover the fact that the status of smartphones itself can violate a company's security policy (like the activation of the camera within a sensitive environment). Only one approach fulfills this requirement completely due to the adoption of remote attestation for smartphones. However, this comes at the cost of the inherent drawbacks of remote attestation, namely certain hardware requirements (a TPM is needed), scalability and privacy issues.

R-02 Detection of abnormal smartphone behavior Various anomaly detection techniques have been applied to the field of smartphone security. Early approaches focused

on detecting abnormal battery usage of smartphones as an indicator for malicious apps. Other approaches rely on monitoring system calls instead. The requirement is partially met by 11 of the presented approaches. They only meet this requirement partially because they are either limited in the type of data that can be considered for anomaly detection or the concrete anomaly detection techniques that can be used. Furthermore, the approaches focus on detecting anomalies that are caused by malicious apps. They do not explicitly address cases where anomalies are caused by benign apps. Only the Andromaly approach completely fulfills this requirement. It supports to capture various types of data on a smartphone which in turn can be processed by numerous anomaly detection methods. However, it fails to meet almost all of the remaining requirements, thus renders itself as not being applicable to the scenarios described before.

R-03 Consideration of context information for detection The context of a smartphone is only taken into account by few of the presented approaches (7 out of 38). Five of the approaches partially meet this requirement as they leverage basic context information for the detection of malicious apps. However, the use of context information is generally restricted to basic timestamps that enable to order sequences of observed events for further processing. Only the approach proposed by Dixon et al. [79] aims to use sophisticated context information for detection purposes. However, the authors lack of presenting concrete results. The two approaches that completely meet this requirement use context information in order to enforce policies on the smartphone itself rather than for analysis or detection purposes. Furthermore, they do not consider to restrict access to services within the IT infrastructure but rather focus on granting access to resources on the smartphone depending on its context. That is, although context information has been taken into account, existing approaches either limit themselves to time-based context information or use the context for different purposes as demanded by this requirement.

R-04 Policy-based reaction on detection results Similar to the previous requirement, a policy-based reaction is only supported by few of the presented approaches. This requirement is partially met by 7 out of 38 approaches. They provide the user with basic means to react on a detection result. For example, if a potential privacy leak is detected, the user can choose to supply no data at all, to supply anonymized data or to supply his real data. However, the majority of the approaches focuses on the detection task and does

not deal with the dissemination of the detection result. Thus, mechanisms that enable an IT infrastructure to react on identified threats are generally not discussed at all. Since existing IT infrastructures commonly provide a wide range of security services, it is an open question how these can be effectively used to mitigate threats that are introduced by smartphones.

R-05 Dynamic analysis at runtime The majority of approaches works at runtime (28 out of 38). Thus, they can employ their analysis or detection capabilities when the smartphone is actually used within an IT infrastructure, completely fulfilling this requirement. The approaches generally impose negligible overhead in terms of resource consumption. However, it must be noted that the runtime capabilities are restricted to the detection of either malicious configurations or anomalies. The dissemination of detection results at runtime within an IT infrastructure is generally not covered. Approaches that cannot be used at runtime as required by the defined scenarios primarily aim to improve the security of a smartphone platform's app store. Thus, they are complementary to the approach developed in this thesis.

R-06 Extensibility of processed data and used methods None of the presented approaches explicitly addresses the requirement to ensure extensibility in terms of data that is processed and the methods that are employed for processing. However, approaches that implement a network-based, hybrid or out-of-band architecture are generally extensible, thus partially fulfilling the respective requirement. These approaches perform most of the data processing remotely. This generally allows to add further detection methods without changing the components that are deployed on the smartphones themselves. However, if new data can be easily added for processing depends on the concrete implementation of the approach. Still, there is no general framework that allows both to define what data should be captured and what detection methods should be used in order to process them. The host-based approaches fail to meet this requirement. In addition to the lack of a sound framework, they also require extensive modifications to the respective smartphone platform.

R-07 Ability to integrate the approach in existing environments The question how new approaches for smartphone security can be integrated into existing IT infrastructures

is generally not discussed at all. Host-based approaches are usually based upon low-level modifications to the smartphone platform. Thus, they do not meet this requirement. Out-of-band approaches can generally be added to existing IT infrastructures. However, there is no defined way to make their detection results available to existing security services. Thus, they only partially fulfill the requirement. The same generally also holds for network-based and hybrid approaches. However, depending on the implementation of the components that gather the data for processing, some of them also fail to meet this requirement. Since some rely on agents that capture low level data on smartphones which makes modifications to the smartphone platform itself necessary, those approaches fail to fulfill this requirement.

3.6 Summary

This chapter provided an extensive presentation of related work in the field of smartphone security. The assessment in Section 3.5 proofs that there is no suitable approach that adequately addresses all requirements derived from the scenarios defined in Chapter 2. Besides the drawbacks of existing approaches in fulfilling the stated requirements, the analysis revealed that most of the related work focuses on detecting malicious apps or on extending smartphone platforms with additional security mechanisms. However, the question how smartphones can be securely integrated in existing IT infrastructures is not addressed at all. In terms of detecting unwanted configurations and anomalies, there is no extensible framework available that allows to define both the data that shall be collected and the methods that should be used to process them in a flexible way. The context of smartphones is often ignored by current approaches. Furthermore, there is a lack of understanding how smartphones are actually used by their users due to the absence of representative studies. Thus, the remainder of this thesis tackles these questions and aims to answer them with a novel, network-based approach for smartphone security. During the evaluation that is presented in Chapter 6, it will also be demonstrated that the novel approach is capable of mimicking the functionality of the host-based Kirin security service that has been presented above, while omitting the need for any modifications to the Android platform.

4 A Network-based Approach for Smartphone Security

"You can't defend. You can't prevent. The only thing you can do is detect and respond."

(Bruce Schneier)

Contents

This chapter introduces a novel, network-based approach for smartphone security. The approach addresses the requirements stated in Section 2.5 that have been identified based on the scenarios defined in Section 2.2. Hence, the approach aims

- to detect unwanted configurations and abnormal behavior of smartphones,
- to use the context of smartphones for detection purposes and
- to enable immediate reactions on detected threats

in order to securely integrate smartphones into existing IT infrastructures. The approach is referred to as **CADS: Context-related Signature and Anomaly Detection for Smartphones**. It is composed of four main parts:

1. A generic, conceptual model that defines the main building blocks and the relationships between them, especially the notion of signatures and anomalies.
2. A distributed architecture that identifies the components that need to be deployed in the target IT infrastructure and their responsibilities.
3. A correlation model that defines how collected data is processed.
4. A process model that defines how the first three parts of the CADS approach can be mapped to a specific problem domain. The domain-specific mapping presented in this thesis targets the problem domain of securely integrating smartphones into existing IT infrastructures as discussed in Chapter 2.

4.1 Conceptual Model

In the following, the components of the conceptual model and their relationships are defined (cf. Figure 4.1). The components are defined in order to meet the requirements that were specified in Section 2.5. There are five different types of components:

1. Core Components. They are used to describe data about smartphones in an abstract manner. The Core Components ensure that CADS is not limited to a certain type of data. Therefore, they primarily address requirement "R-06 Extensibility of processed data and used methods".

2. Context-related Components. They are used to express the context of smartphones. Therefore, they primarily address requirement "R-03 Consideration of context information for detection".

3. Signature Components. They provide the functionality to express the configuration of smartphones. Signature Components are primarily used to address requirement "R-01 Detection of unwanted and malicious configurations of smartphones".

4. Anomaly Detection Components. These components allow to detect abnormal behavior of smartphones. They are mainly used to address requirement "R-02 Detection of abnormal smartphone behavior".

5. Policy Components. These components are used to express policies based on the other types of components. They allow to define under which circumstances certain reactions should be employed. Thus, they are primarily used to address requirement "R-04 Policy-based reaction on detection results".

It should be noted that two requirements ("R-05 Dynamic analysis at runtime" and "R-07 Ability to integrate the approach in existing environments") are not directly addressed by the Conceptual Model. Instead, they are covered by the CADS architecture (Section 4.2) and the Correlation Model (Section 4.3).

4.1.1 Core Components

The Core Components form the most basic building blocks of the model. They are used in order to describe the data that is collected and processed in a structured manner. One of

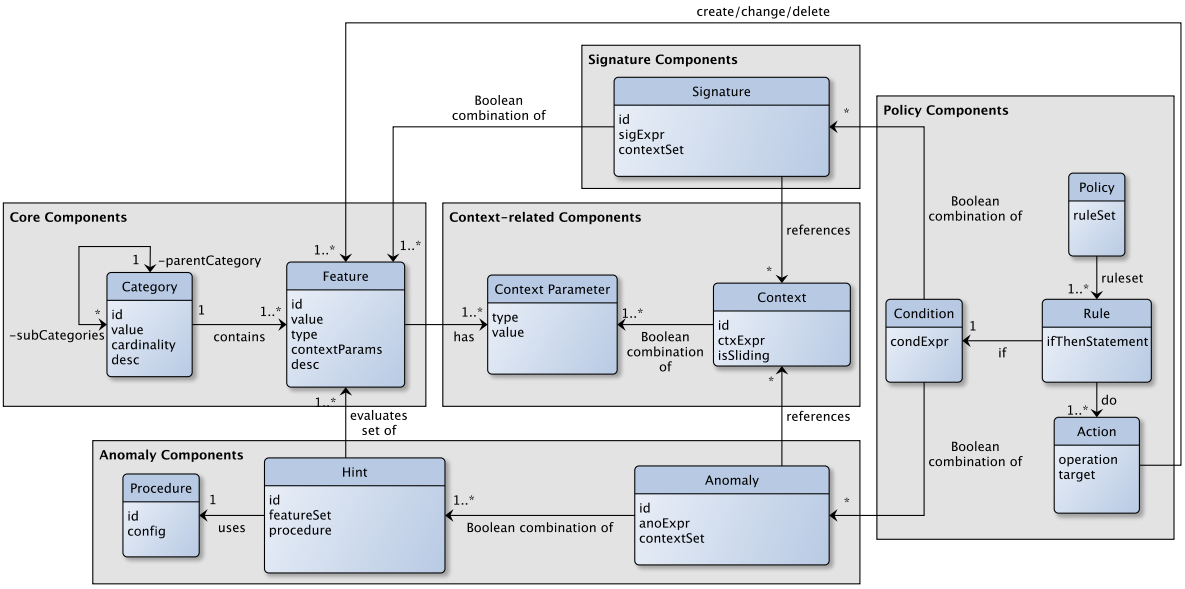

Figure 4.1: CADS Conceptual Model.

the main challenges is the fact that the question what data should be collected and processed in order to detect unwanted configurations and abnormal behavior of smartphones cannot be generally answered. As discussed in Chapter 3, existing approaches have used different types of data for the detection of malicious apps and abnormal behavior. Types of data that have been used include system calls, data that describes static properties of apps and data that represents low level aspects of the smartphone device itself (such as the depletion rate of its battery). However, in order to fulfill the requirements defined in Section 2.5, especially requirement R-06 "Extensibility of processed data and used methods", an extensible mechanism that allows to consider arbitrary data is necessary. In this respect, the term "arbitrary data" means that the approach must not be limited to a certain type of data (battery level, permissions of apps) for employing its detection tasks as existing approaches are. Instead, it must provide a mechanism that allows to define on demand what type of data should be considered for the detection tasks. This is achieved by introducing the Core Components Feature and Category.

Feature A Feature describes the most basic building block of the model and thus forms the fundamental basis for any further components. In essence, a Feature represents an

issue of interest within a certain problem domain. The notion of the Feature component is derived from the field of artificial intelligence, specifically the field of anomaly detection. There, the term "feature" is used to describe data instances that are used as input for concrete anomaly detection techniques [96]. A Feature is composed of the following elements:

- A globally unique identifier. This enables to unambiguously identify a certain Feature within the set of all potentially defined Features.
- A value that contains data to represent the issue of interest.
- A type that provides information regarding the data that is contained in the value element. There are three different types of values that are distinguished: (1) qualified (2) quantitative and (3) arbitrary.
  - The content of qualified values stems from a limited list of possible values (such as low, medium or high).
  - Quantitative values contain numerical, ordinal data that can have a measurement unit associated (such as percent).
  - Regarding the content of arbitrary values, no restrictions are made. They are not limited by an enumeration and have no associated measurement units. This type of value is necessary to express arbitrary Features like the name of an app.

  For convenience, the terms qualified, quantitative or arbitrary Feature refer to a Feature whose value has the corresponding type (either qualified, quantitative or arbitrary).
- A set of Context Parameters. They encapsulate data that describe the contextual situation at the moment when the Feature's value is set.
- A human readable description that provides the semantics of the Feature.

Thus, a Feature combines data with semantics. Regarding the conceptual model, a Feature represents an atomic piece of information. At runtime, instances of Features are created, transmitted, updated and deleted. The logical roles that are involved in these operations are presented in Section 4.2. The elements of a Feature can be classified as being either static or dynamic. The static part is set once when an instance of a Feature

is created and never changes afterwards. The dynamic part of a Feature instance can be changed during its lifetime. It is composed of the Features value and the referenced Context Parameters. As there has been a lot of ambiguity regarding the terms *data* and *information*, a definition that is used to distinguish the terms throughout the rest of this thesis is provided now.

Information is defined as data attached with semantics. For example, a qualified Feature has the value "false". The value alone is just data representing the string "false". However, considering the Feature's identifier and the description, semantic background is given that enables to derive information based on the provided data. That is, information which is derived from the data varies depending on the definition of the Feature. For example, the value "false" can provide the information that the user is not present or that a certain sensor of a smartphone is not active. A more involved discussion regarding the differences between data and information that also covers the term *knowledge* is given by Boisot and Canals [159].

Category Categories are used to structure the set of defined Features according to their semantics. Thus, a Category *contains* Features that have similar semantics. Furthermore, Categories themselves can be hierarchically structured as well. That is, a Category can have multiple sub Categories in addition to the Features it contains. However, each Category can only have a single parent Category. A Category is composed of the following elements:

- A globally unique identifier. This enables to unambiguously identify a certain Category within the infinite set of all potentially defined Categories.
- A value. As Categories primarily work as containers for Features and other Categories, a single value does not make much sense. Instead, the value of a Category is the set of identifiers of all contained Features and Categories.
- A cardinality. It defines the cardinality of the contained Features and is set to either 1 or N. For example, a Feature that represents a smartphone's current battery level will be contained by a Category with a cardinality of 1 (as there is always one single valid value for such a Feature at any given time). In contrast, a Feature that represents the name of a permission requested by an app on a smartphone will be

contained within a Category with the cardinality N (since apps likely request more than a single permission). Thus, there will be multiple instances of this Feature at the same time. As already stated, the components and the process that details the instantiation of Features are presented in Section 4.2.

- A human readable description that provides the semantics of the Category.

Note that Categories do no have any Context Parameters. This is due to the fact that their primary purpose is to structure the infinite set of Features that can be defined.

Identifier Generation for Core Components Both Features and Categories need a globally unique identifier. Although those could be chosen totally random, the following production rules should be followed when defining Features and Categories for a specific problem domain.

The naming of identifiers should follow a similar, hierarchical approach as the naming conventions of Java packages and classes do [160]. According to the naming conventions, Java packages are named by lower case American Standard Code for Information Interchange (ASCII) letters. Each layer of the hierarchy is separated by a dot (.). Classes are named by ASCII letters as well. However, at least the first letter is capitalized. Long class names that constitute of more than one word may also follow the CamelCase notation, that is the first letter of each word is capitalized. This approach is adopted for the definition of identifiers for Features and Categories. Categories are named like Java packages (lower case ASCII letters). Features are named like Java classes (ASCII letters with CamelCase). The hierarchy of Categories is expressed by concatenating their respective identifiers by dot notation.

Assuming there is a Category c with a sub Category s and assuming that this sub Category includes a feature F, the fully qualified identifiers are as follows:

- The fully qualified identifier for Category c would simply be c.
- The fully qualified identifier for Category s would be $c.s$.
- The fully qualified identifier for Feature F would be $c.s.F$.

Thus, the fully qualified identifiers always represent the complete hierarchy of Categories and the contained Feature. If there is no ambiguity, only the single identifiers of Categories

and Features will be used in the remainder of this thesis in order to address them, omitting their hierarchical relation ship. Regarding the example above, the phrase "the Feature F is used" refers to the Feature whose fully qualified identifier is $c.s.F$.

However, a problem still exists if single instances of Features that are contained by Categories with a cardinality of N should be referenced by their identifier. Given the production rule above, this can lead to collisions as there might be more than one instance of the same Feature F at the same moment in time. Given the production rule above, those instances cannot be distinguished if necessary. Thus, in order to solve this problem, a slight addition to the production of identifiers for Categories is made, referred to as instance dispatch. If a Category is of cardinality N and single instances should be identified, its identifier is attached with an instance counter. Referring to the example above, assume that Category s is of cardinality N. If multiple instances of the Category (and its contained sub-categories and Features) must be distinguished, they can be identified as follows:

- The identifier for the first instance is $c.s_0.F$.
- The identifier for the second instance is $c.s_1.F$.
- The identifier for the n^{th} instance is $c.s_n.F$.

As discussed later in this thesis, there are circumstances where both the instance aware and instance unaware identifiers are used. Again, the short form of identifiers will be used throughout the remainder of this thesis as long as it unambiguously identifies the respective Category and Feature (that is F instead of $c.s.F$ or $c.s_n.F$).

With these two Core Component types it is possible to render issues that are of interest for a specific problem domain in a structured manner, encapsulating both the semantics and the actual data for later processing. An example for a Feature that represents the name of an Android app could be defined as follows: $smartphone.android.app.Name$. In this case, the Category $smartphone.android.app$ would be the only one that is of cardinality N, whereas $smartphone$ and $smartphone.android$ would be of cardinality 1. Thus, the identifier $smartphone.android.app.Name$ would match all instances of the respective Feature, whereas $smartphone.android.app_5.Name$ would only match the 6th instance of the Feature.

4.1.2 Context-related Components

This type of components is used to express the context of a Feature. More precisely, they describe the context at the moment when the value of the respective Feature is set. There are two components of this type: Context Parameter and the Context.

Context Parameter Context Parameter encapsulate data that describe the context of a Feature at that moment when its value is set. Each Feature references a set of Context Parameters. Examples for Context Parameter include timestamps and location data. Their internal structure is similar to that of a Feature. However, there are two main differences: (1) Context Parameters cannot be hierarchically structured by means of Categories and (2) their only purpose is to form the basis for the definition of Context components as described below. A Context Parameter is composed of the following elements:

- A type that provides the semantic background for the Context Parameter.
- A value that contains data to represent the contextual information.

For example, in order to express the geographical location at which the value of a Feature was set, one can define Context Parameters that encapsulate the coordinates obtained via GPS.

Context A Context is basically a Boolean expression that is formulated based on the previously introduced Context Parameters. By evaluating the Boolean expression, it is determined whether the Context is fulfilled (the Boolean expression evaluates to *true*) or not (the Boolean expression evaluates to *false*). A Context is composed of the following elements:

- A globally unique identifier. As for the Core Components, this is used to unambiguously identify a certain Context. However, since there is no hierarchical structuring of Context components, its generation is more simple. The identifier could be chosen randomly. However, it is again advised to choose meaningful names composed of ASCII letters, where the first letter is capitalized and an appropriate prefix is used. That is, in order to identify a Context that expresses the time interval of a company's working hours, the identifier can be set as $context.WorkingHours$.

- A Boolean expression which is formulated based on Context Parameters. The expression only uses basic binary Boolean operations, that is "and" ($\wedge$) and "or" ($\vee$). The Boolean expression itself is composed of numerous relational expressions which are combined by the previously mentioned Boolean operators. Each relational expression is composed of a Context Parameter, a relational operator and a value. The following relational operators can be used: $<, >, \leq, \geq, =, \neq$. In order to evaluate a relational expression, the value of the respective Context Parameter is compared against the value that is provided as part of the relational expression. The same Context Parameter can be used in multiple relational expressions (thus comparing its value against multiple other values).

- A flag that indicates whether the Context is a so-called *sliding* Context. This flag is relevant during the evaluation of the Context's Boolean expression and used for Context definitions that are based on temporal Context Parameters. For example, a sliding Context based on a temporal Context Parameter could define that only Features whose values have been set within the last five minutes fulfill the Context (thus ensuring their freshness). The different handling of sliding and normal Context components is further detailed in Section 4.3.

That is, the general structure of the Boolean expression of a Context $ctxExpr$ can be formalized as follows: $\diamond$ denotes any binary Boolean operator, $\asymp$ denotes any relational operator. Let N be the number of Context Parameters that are used by a Context. A Context Parameter is denoted as $ctxP_i$ with $0 \leq i < N$. Each Context Parameter can be used in multiple relational expressions. Let M_i be the number of relational expressions the Context Parameter $ctxP_i$ is used in. Thus, the value of $ctxP_i$ is compared against M_i other values denoted as $val_{i,j}$ with $0 \leq j < M_i$. Then, the formal definition of a Context's Boolean expression is as follows:

$$
\begin{aligned}
&ctxExpr := \\
&((ctxP_0 \asymp val_{0,0}) \diamond (ctxP_0 \asymp val_{0,1}) \diamond \ldots \diamond (ctxP_0 \asymp val_{0,M_0}) \diamond \\
&(ctxP_1 \asymp val_{1,0}) \diamond (ctxP_1 \asymp val_{1,1}) \diamond \ldots \diamond (ctxP_1 \asymp val_{1,M_1}) \diamond \\
&\vdots \\
&\overbrace{(\underbrace{ctxP_{N-1} \asymp val_{N-1,0}}_{\textit{relational expression}}) \diamond (\underbrace{ctxP_{N-1} \asymp val_{n1}}_{\textit{relational expression}}) \diamond \ldots \diamond (\underbrace{ctxP_{N-1} \asymp val_{N-1,M_{N-1}}}_{\textit{relational expression}})}^{\textit{Boolean expression}})
\end{aligned}
$$

Given the scenarios defined in Chapter 2, the number of Context Parameters that are actually used by a Context will likely be small (not more than three). Furthermore, the number of relational expressions used per Context Parameter is likely to be at most two (in order to test a certain interval, for example $(ctxP_0 > 5) \wedge (ctxP_0 < 10)$. If a sliding Context is used, there will likely be only one relational expression. For example, $(ctxP_0 < 00:03:00)$ in order to select Features whose values have been set within the last three minutes.

The purpose of a Context is primarily to easily choose instances of Features that are relevant for further processing. That is, it allows to easily select instances of Features whose values have been set at a given time or at a given location (depending on which Context Parameters are used). If the Boolean expression of a Context is *true* for a specific Feature instance, it *fulfills* the respective Context. This is essential for the detection capabilities of the introduced approach. For example, the use of a certain app might be prohibited by the company's security policy for smartphones. However, this only applies if the smartphone is currently used on site at the company during working hours. The question whether the smartphone is on site or not and whether the respective app was used during working hours or not can be answered by using appropriate Contexts.

4.1.3 Signature Components

Signature components are used to express patterns based on Feature and Context components. One essential requirement stated in Section 2.5 is the ability to detect unwanted

configurations of smartphones. Signature components are the main components that enable to fulfill this requirement. There is one component of this type.

Signature The purpose of a Signature is to express a pattern based on a set of previously defined Features. Furthermore, a Signature can reference a set of Contexts that have been defined. Similar to a Context, the notion of a Signature is binary, that is the pattern which is defined by a respective Signature actually matches or not. A Signature is composed of the following elements:

- A globally unique identifier. As for the Core Components, this is used to unambiguously identify a certain Signature. Similar to the Context components described above, there is no hierarchical structuring of Signatures. Thus, their identifier should be composed following the rules defined for the Context component, with a different prefix (*sig*).
- A Boolean expression which is formulated based on Features. The structure of the expression is the same as the one used by a Context. The only difference is that Signatures use Features instead of Context Parameters. That is the Boolean expression for a Signature is composed of relational expressions that compare Features (more precisely the values of Features) against values that are specified as part of the Signature definition. The same Feature can be used in multiple relational expressions (thus comparing its value against multiple other values). During the evaluation of the Boolean expression, all instances that are available for a referenced Feature are considered. For example, if a Signature references a Feature that represents a requested permission of an app, it could match multiple times depending on how many apps have requested the respective permission. Further details on the evaluation of Signatures are given in Section 4.3 as part of the Correlation Model.
- A list of referenced Context components. These are used in order to limit the set of Feature instances that are considered during the evaluation of the Signature. That is, only instances of Features that fulfill all of the referenced Context components will be considered during the evaluation of the Boolean expression.

Similar to the Context component, the general structure of the Boolean expression of a Signature $sigExpr$ can be formalized. Let N be the number of Features that are used

by a Signature. A Feature is denoted as ζ_i with $0 \leq i < N$, identified by means of its global identifier. Each Feature can be used in multiple relational expressions. Let M_i be the number of relational expressions for a Feature ζ_i. The value of each instance of the Feature ζ_i is compared against M_i other values denoted as $val_{i,j}$ with $0 \leq j < M_i$. Thus, the complete definition is as follows:

$$
\begin{aligned}
&sigExpr := \\
&((\zeta_0 \asymp val_{0,0}) \diamond (\zeta_0 \asymp val_{0,1}) \diamond \ldots \diamond (\zeta_0 \asymp val_{0,M_0}) \diamond \\
&(\zeta_1 \asymp val_{1,0}) \diamond (\zeta_1 \asymp val_{1,1}) \diamond \ldots \diamond (\zeta_1 \asymp val_{1,M_1}) \diamond \\
&\vdots \\
&\overbrace{(\underbrace{\zeta_{N-1} \asymp val_{N-1,0}}_{\textit{relational expression}}) \diamond (\underbrace{\zeta_{N-1} \asymp val_{n1}}_{\textit{relational expression}}) \diamond \ldots \diamond (\underbrace{\zeta_{N-1} \asymp val_{N-1,M_{N-1}}}_{\textit{relational expression}})}^{\textit{Boolean expression}}
\end{aligned}
$$

To summarize, a Signature is used to define patterns based on Features. In order to evaluate a Signature, its Boolean expression is evaluated based on a set of Feature instances. Only those Feature instances are relevant for evaluation that (1) have an identifier that is used within one of the Signatures relational expressions and (2) that fulfill the referenced Context components.

One example for a Signature is to define a pattern to detect a suspicious app. A company could define a Signature that states an app as being suspicious when it has (1) a certain combination of dangerous permissions and (2) was obtained from an unofficial app store. Furthermore, it could leverage Context components in order to specify that the presence of such apps should only be detected within a certain time interval each day and at a specific location (for example the company's research facilities).

4.1.4 Anomaly Detection Components

As already stated in Chapter 3, anomaly detection techniques have been widely used in the field of information security. According to Chandola et al. [96], an anomaly is defined as follows:

> *"At an abstract level, an anomaly is defined as a pattern that does not conform to expected normal behavior."*

The goal of this approach is to model normal behavior based on the previously defined Core Components (Section 4.1.1), primarily the Features, while leveraging the benefits introduced by the Context-related Components. This is achieved by defining appropriate Anomaly Detection Components in the following. One major challenge is that at this time, the concrete technique that is used for detecting anomalies should not be specified. That is, the model must be flexible in expressing which Features should be analyzed by which methods in order to detect anomalies. This is achieved by introducing three components, referred to as (1) Anomaly, (2) Hint and (3) Procedure. Anomaly components are defined based on Hints, and Hints make use of Procedures. The details of the components are described in the following.

Anomaly The Anomaly component is used to express that an abnormal behavior was detected. This is not done by referencing Features directly (like the Signature does). Instead, there are two intermediary components used in order to provide the necessary flexibility to process arbitrary Features with arbitrary anomaly detection techniques. An Anomaly is composed of the following elements:

- A globally unique identifier which should be generated similar to the identifier of a Signature. However, instead of the prefix *sig* the prefix *ano* should be used.
- A Boolean expression which is formulated based on Hints. The structure of the expression is comparable to the one used by a Signature, but less complex. This time the relational expressions are using Hints instead of Features (more precisely the scoring of each Hint that was returned by their respective Procedure as detailed below). In contrast to a Signature, there can only be one relational expression for each Hint. This limitation is possible since the relational expressions are only used in order to verify to what extent a certain Hint is fulfilled (based on the scoring result of its Procedure).
- A list of referenced Context components. These are used in order to limit the set of Feature instances that are considered during the evaluation of an Anomaly. Note that Anomaly components do not reference Features directly. Instead, this is done by

the Hint components as detailed below. Thus, the referenced Context components determine the set of Feature instances for all Hints that are used by an Anomaly.

A formal definition of an Anomaly's Boolean expression *anoExpr* is given in the following. Let N be the number of Hints that are used by an Anomaly. Each Hint is denoted as h_i with $0 \leq i < N$. The definition is less complex since there is only exactly one relational expression for each Hint. Within each of the relational expressions, the score that is returned by the Hint's Procedure is compared against the given value denoted as val_i with $0 \leq i < N$. Thus, the complete definition is as follows:

$$anoExpr := \overbrace{(\underbrace{h_0 \asymp val_0}_{relational\ expression}) \diamond (\underbrace{h_1 \asymp val_1}_{relational\ expression}) \diamond \ldots \diamond (\underbrace{h_{N-1} \asymp val_{N-1}}_{relational\ expression})}^{Boolean\ expression}$$

If the Boolean expression of an Anomaly evaluates to *true*, an abnormal behavior was detected. The use of Hints allows to specify which Features should be processed by which anomaly detection techniques in a flexible way as detailed below. It is important to note that many anomaly detection techniques have constraints, especially concerning the data that they can process [96]. Since the conceptual model does not limit the nature of Features and their semantics nor the anomaly detection methods that can be used, the flexibility provided by Hints is essential.

Hint A Hint is a part of an Anomaly. A Hint generally expresses whether the value of a set of Features differs from the expected values in such a way that it is considered as being abnormal. The actual anomaly detection technique that is used in order to analyze the set of Features is referred to as a so-called Procedure. Each Hint references exactly one such Procedure. The same Hint can be referenced by multiple Anomaly components. Each Hint is composed of the following elements:

- A globally unique identifier which should be generated similar to the identifier of an Anomaly. However, instead of the prefix *ano* the prefix *hint* should be used.
- A list of Features that are referenced by their respective identifiers. The set of Feature instances that are considered during the evaluation of a Hint is limited by the Context components that are referenced by the respective Anomaly component.

- A referenced Procedure. The Procedure is responsible for processing all Feature instances (that is all Feature instances that have been referenced by the Hint and that fulfill all Context components as defined by the respective Anomaly) in order to detect abnormal behavior.

Thus, the main purpose of a Hint is to map a certain set of Feature instances to a certain Procedure. The Procedure employs a concrete anomaly detection method. The idea is that one Anomaly (that is a smartphone's abnormal behavior) can manifest itself in such a way that different Features must be processed by different anomaly detection techniques. This is supported by the model by means of Hint components that make use of Procedures.

Procedure A procedure is referenced by a Hint in order to process a set of Feature instances to detect abnormal behavior. Within the conceptual model, a Procedure is a generic component and does not rely on or propose a specific anomaly detection technique. Instead, it models a generic interface with only minor assumptions that must be fulfilled. These are

- the Procedure must be able to process a set of Feature instances as input data and
- the Procedure must support to map its detection result to an anomaly score in the range of $[-1, 1]$ indicating the amount of abnormal behavior that was detected. -1 indicates that there was no abnormal behavior, 1 indicates that the Procedure is certain to have observed abnormal behavior. Decimal values in between are also possible (such as 0.75). Anomaly scores are a common approach to render the output of anomaly detection techniques [96]. The score that is returned by a Procedure thus indicates to what extent the respective Hint is fulfilled.

  The range of $[-1, 1]$ was chosen arbitrarily. Other ranges would work as well, as long as it is defined which end of the range indicates abnormal behavior and which range indicates normal behavior.

This allows to use virtually any anomaly detection method that might give reasonable results for a specific set of Features. Examples include simple statistical analysis or more sophisticated machine learning techniques. Thus, regarding the conceptual model, a Procedure basically consists of only two elements:

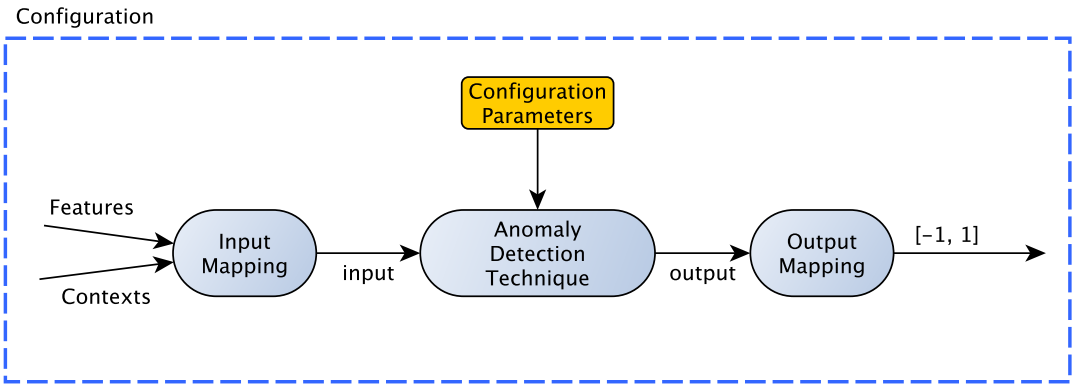

Figure 4.2: Configuration of a Procedure.

- A globally unique identifier which should be generated similar to the identifier of an Anomaly. However, instead of the prefix *ano* the prefix *proc* should be used.

- A configuration that ensures that the assumptions stated above are met. The configuration basically defines three aspects as depicted in Figure 4.2:

  1. It defines an input mapping. This mapping handles how the set of Features, their values and their Context Parameters are mapped to the concrete anomaly detection technique (such as mapping the Features to input signals of a neural network).

  2. Similar to that, an output mapping is necessary as well. This mapping defines how the result of the concrete anomaly detection technique is mapped to the range $[-1, 1]$.

  3. Further configuration parameters that are specific for the concrete anomaly detection technique. For a neural network, this would include parameters such as the number of nodes and their associated weights.

The introduced components for anomaly detection enable to model the expected, normal behavior based on Features while providing the flexibility to use virtually any anomaly detection technique that follows the stated assumptions.

4.1.5 Policy Components

This type of components provides means in order to formulate policies based on the previously introduced components of the conceptual model. This ensures that the functionality provided by the approach can be used in various companies with different requirements in terms of the integration of smartphones. There are four components of this type. It is important to note that the presented approach does not aim to make any major contribution in the research fields related to policy design and policy languages but just applies existing approaches.

Policy A Policy P within the conceptual model is composed of a set of Rules. An expert is necessary in order to formulate reasonable Policies for a specific domain. Regarding the scenarios described in Section 2.2, there will be one Policy for each administrative domain. The purpose of a Policy is twofold: (1) it defines which Signatures and Anomalies should be detected and (2) what Actions should be employed in order to react on detected Signatures and Anomalies.

Rule A Rule R in the conceptual model is a simple statement of the form *if Condition do Actions*. If the Condition is fulfilled, the associated Actions are performed.

Condition A Condition C is a Boolean expression that is used to trigger Actions. Conditions in the conceptual model are formulated based on the previously introduced Signature and Anomaly components. This way, a domain expert can specify which reactions are necessary upon the occurrence of a Signature respectively an Anomaly. Both Signature and Anomaly instances can be formulated based on Contexts, thus allowing Context-related policies.

Action An action *Act* can change, create or delete a set of Feature instances once it is triggered by its corresponding Rule. The idea is to model the consequences that are associated to a rule that has fired by using the Core Components again. For example, in order to render a notification about an intrusion that was detected, appropriate alert Features can be created. Besides the general flexibility, this back coupling approach provides a major benefit: it allows to reuse Features that are created, changed or deleted by Actions as basis for further Signature and Anomaly definitions.

4.2 Architecture

The previous sections introduced the conceptual model of the CADS approach. Its main purpose is to provide a domain independent model that defines the basic structure and notion of Signatures, Anomalies and Contexts. The Feature was introduced as fundamental basis for any further components. However, it has not yet been defined which components are necessary in order to make use of the conceptual model.

This is done now by introducing the CADS architecture. It defines logical roles and their responsibilities. These roles must be fulfilled by components that are deployed within an IT infrastructure in order to benefit from the CADS approach. The idea to define an architecture based on logical roles was adapted from the TNC Architecture for Interoperability specification that was defined by the TCG [36]. The use of logical roles emphasizes that the architecture is mostly independent from physical aspects. For example, the functionality that is expected by a single logical role can be provided by numerous software or hardware components.

In order to fulfill the defined logical roles, it is not necessarily required to deploy new components or services to an existing IT infrastructure. Instead, it is expected that existing components and services (like those discussed as part of the reference IT infrastructure in Section 2.1) are extended with additional functionality. Those extended components and services can then fulfill some of the logical roles as defined by the CADS architecture.

4.2.1 Logical Roles

There are four logical roles defined by the CADS architecture:

1. Feature Collector,
2. Feature Provider,
3. Correlation Engine and
4. Feature Consumer.

They are detailed in the following.

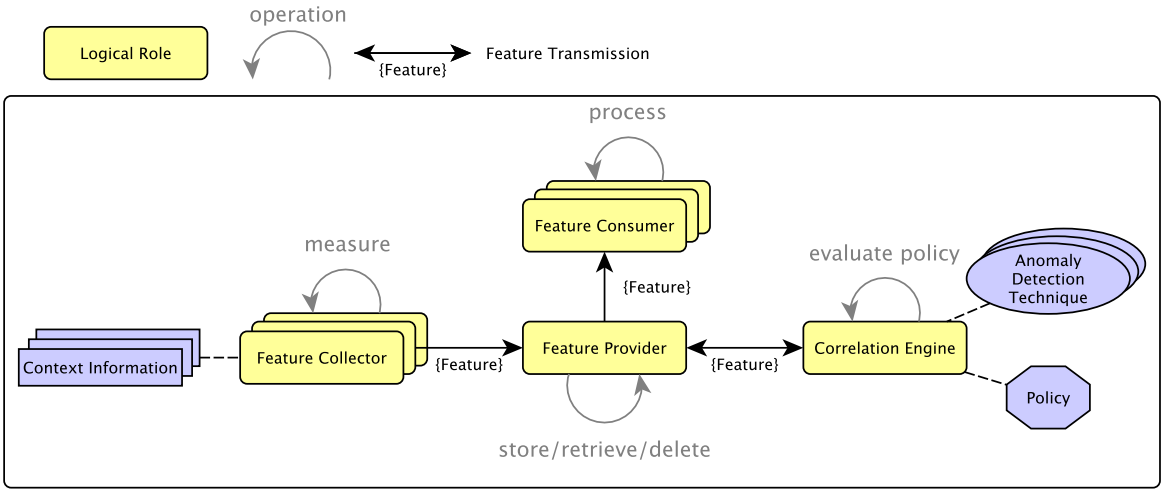

Figure 4.3: Architecture of the CADS approach.

Feature Collector The main responsibility of Feature Collectors is to collect Features that are expected to be useful in order to detect unwanted configurations and abnormal behavior of smartphones at runtime. Compared to known approaches in the field of intrusion detection, Feature Collectors act as sensors in the CADS architecture. Thus, Feature Collectors can reside at arbitrary components and services within the respective corporate IT infrastructure. This also includes the smartphones themselves. The process of collecting a Feature constitutes of two sub steps:

1. **Feature Measurement**. In order to be used, Features need to be instantiated at runtime. This generally involves to observe a certain issue of interest (for example like monitoring the outgoing traffic of a smartphone), create an instance of a Feature that reflects the issue of interest and to set its value. Furthermore, a Feature Collector is required to tag each created Feature with appropriate information about its context by setting the Feature's Context Parameters accordingly (like the current time or location of the Feature Collector). The same Feature Collector can measure multiple Features, that is Features that have different identifiers. Furthermore, a single Feature Collector can generally measure Features for multiple smartphones, depending on where it is deployed in the target IT infrastructure. Depending on the

cardinality of the Categories a Feature is contained in, the measurement step may require to create multiple instances of the same Feature.

2. **Feature Transmission**. This involves to marshal the measured Feature instances and to send the resulting message to the Feature Provider. The communication protocol that is used for Feature Transmission is not specified by the CADS architecture. However, general requirements that must be met by a protocol that shall be used for implementing the CADS approach are given in Section 4.2.2.

The collection of Features can be triggered by different means. Collecting Features at predefined time intervals (like every 15 minutes) or based on certain events (like having moved a specific distance) are two examples. There will be several components and services acting as Feature Collectors in an IT infrastructure that implements the CADS approach. In addition to measuring and transmitting Features, a Feature Collector can also choose to delete Feature instances that have been collect before. In this case, the Feature Collector is only required to transmit a list of identifiers corresponding to the Feature instances it wishes to delete.

Feature Provider The Feature Provider acts as single point of storage for all Features that have been collected. Communication among the logical roles always involves the participation of the Feature Provider. It provides functionality to store, retrieve and delete Feature instances based on their global identifier. Since Features describe issues of interests of smartphones, and since there will be numerous smartphones present within the same IT infrastructure, the Feature Provider must provide means to distinguish the received Features according to the smartphone they belong to. This is required for all three operations supported by the Feature Provider (store, retrieve, delete).

1. **Feature Storage**. This is composed of three sub-steps. A set of Features must be received (either from a Feature Collector or the Correlation Engine), unmarshalled and then stored for later use. If the Feature Provider has received the same instance of a Feature before, it simply updates the respective instance by setting its value and Context Parameters accordingly. For each Feature instance, the Feature Provider keeps track of all changes that have been applied until the respective instance is deleted.

2. **Feature Retrieval**. A Feature Provider includes functionality to retrieve Features it has stored before. The retrieval is triggered by requests sent from either the Correlation Engine or a Feature Consumer. The requests specify which Features should be retrieved, thus acting like a search query. After receiving such a request, the Feature Provider looks up the matching Feature instances it has stored, marshals them and transmits the resulting message to the logical role that has issued the request. Within CADS, this is either the Correlation Engine or a Feature Consumer.

3. **Feature Deletion**. A Feature Provider also supports to delete Feature instances it has stored before. This involves to receive a set of Feature identifiers that specify which instances should be deleted, unmarshal them and perform the necessary delete operations within the storage of the Feature Provider. That is, the previously stored Feature instances are flagged as being deleted. Deleted Features are also covered by the retrieval functionality. This is necessary in order to let Feature Consumers and the Correlation Engine know when a certain Feature was deleted. For the deletion of Features, it is sufficient to sent the appropriate identifiers to the Feature Provider. Again, similar to the storage functionality, the Feature Provider must be able to distinguish between the numerous smartphones that are present in an IT infrastructure. This functionality is used either by a Feature Collector or the Correlation Engine.

Again, the concrete protocols that are used in order to access the Feature Provider's storage and retrieval functionalities are not specified at this point. However, referring to the classical client-server model of distributed systems [161], the Feature Provider acts as server (hosting Features), whereas the other logical roles act as clients that aim to access the functionality provided by the Feature Provider.

Correlation Engine Within the CADS architecture, there is exactly one Correlation Engine that is responsible for processing the set of collected Features. In order to do so, it evaluates a Policy that has been defined according to the conceptual model described in Section 4.1. In order to obtain the set of Features that are needed for processing, the Correlation Engine leverages the retrieval functionality of the Feature Provider. The evaluation of the Policy is done by checking the Condition of each Rule, thus looking for matching Signatures and abnormal behavior. As for the anomaly detection, any technique

can be used, as long as it meets the assumptions regarding the definition of a Procedure in Section 4.1.4. For each Rule that fires, the associated Actions are performed. This can trigger the creation of new Feature instances or the update / deletion of existing Feature instances. Thus, the storage and deletion capabilities of the Feature Provider are leveraged as well. Within the CADS architecture, the Correlation Engine is the only logical role that both receives Features from and transmits Features to the Feature Provider. The algorithms that are employed in order to evaluate the Policy based on the set of Feature instances are detailed in Section 4.3.

Feature Consumer The last logical role within the CADS architecture is the Feature Consumer. Similar to the Correlation Engine, it requests Features from the Feature Provider and processes them. However, the Consumer is not responsible for evaluating the defined Policy. Instead, it is expected to react on Features that have been created, updated or deleted by the Correlation Engine. This allows any component within the IT infrastructure that acts as Feature Consumer to react on the detection results of the Correlation Engine. The way a Feature Consumer receives new Feature from the Feature Provider depends on the concrete communication protocol that is used in order to implement the CADS approach. For example, depending on the communication protocol and its capabilities, the Feature Consumer is notified about new Features or needs to actively poll the Feature Provider for any updates at a regular interval.

4.2.2 Communication Protocol

So far, the main logical roles of the CADS architecture have been defined. Components and services that fulfill these roles must be deployed within an IT infrastructure in order to benefit from the CADS approach. Whereas the Correlation Engine and the Feature Provider will likely be implemented by new components and services which are added to the respective IT infrastructure, Feature Collectors and Feature Consumers can be realized by extending existing components and services. Communication between these logical roles must be possible in order to exchange Features. As the logical roles will be fulfilled by components that are distributed in the target IT infrastructure, this calls for a network communication protocol.

A concrete protocol is not chosen at this point. This would limit the CADS approach to a specific technology. However, some requirements can be defined that a communication protocol must fulfill in order to be used for implementing the CADS approach:

1. **Request-Response Interaction**. The communication protocol must allow a point-to-point interaction between the Feature Provider (acting as server) and each of the other logical roles (acting as clients). This interaction is based on requests that are sent to the Feature Provider which in turn answers each request with an appropriate response.

2. **Proper Transmission of Features**. The protocol must allow to transmit a set of Features between the logical roles of the CADS architecture. That is, it must allow proper marshalling and unmarshalling of Features and their transmission as payload. Furthermore, the protocol must allow to distinguish which measured Features belong to which smartphone.

3. **Support for Remote Procedure Calls**. The protocol must allow to encapsulate remote procedure calls. This is necessary in order to specify which functionality of the Feature Provider is desired by the requesting logical role on a per request basis. That is, each request that is sent to the provider must be marked according to the operation that should be triggered (either store, retrieve or delete). The arguments of the remote procedure calls are Features that are encapsulated as payload.

4. **Secure**. The communication protocol must ensure a secure transmission of Features. Secure means that the integrity and confidentiality of the transported Features must be ensured. As stated in Section 2.4, it is assumed that an attacker can eavesdrop and modify any traffic that is transported over the network.

5. **Efficient and Scalable**. The amount of Features that are exchanged between the logical roles can be high. Depending on the concrete IT infrastructure the CADS approach is deployed in, this amount might range from some tens of Features to thousands of Features per second. Thus, the communication protocol must be efficient and scalable in terms of Feature encapsulation and transmission.

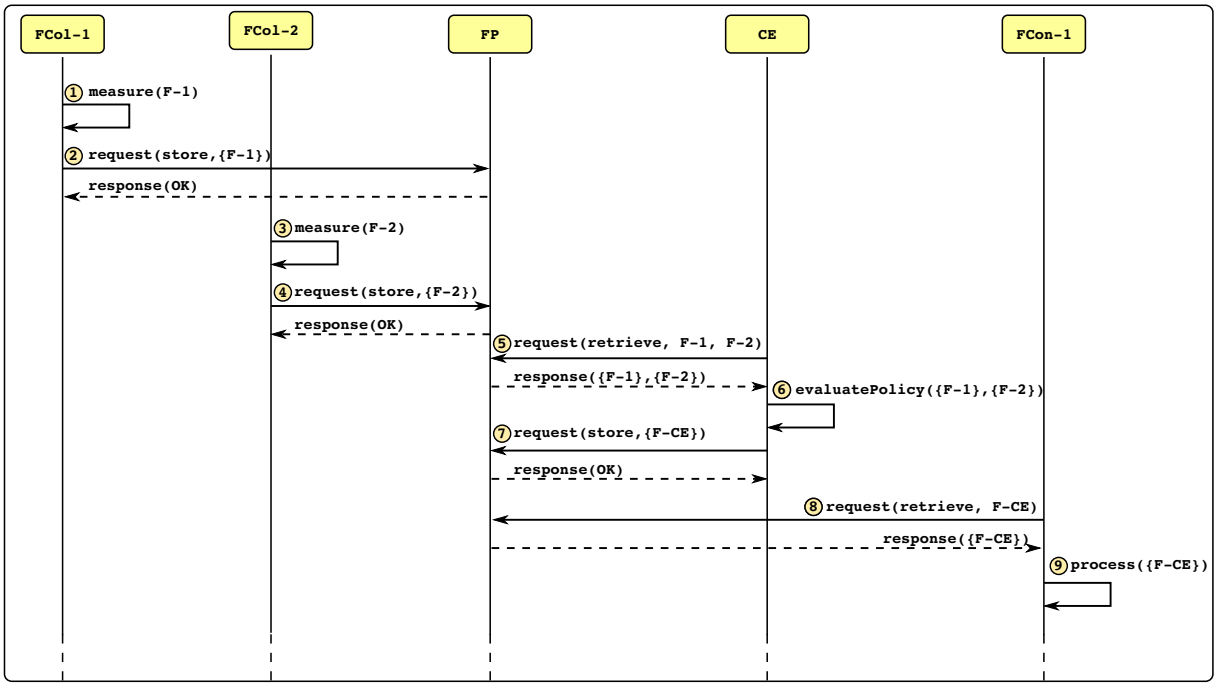

Figure 4.4: CADS Communication Flow Example.

As shown later in Chapter 5, the IF-MAP protocol for network security presented in Section 3.4 completely fulfills all of the stated requirements and is thus well suited for implementing the CADS approach.

With this basic definition of requirements regarding the communication protocol that can be used for implementing the CADS approach, the generic interaction between components and services that fulfill the logical roles can be specified. The sequence diagram depicted in Figure 4.4 visualizes a generic example of such an interaction. The example assumes that there are two Feature Collectors (*FCol-1, FCol-2*), the Feature Provider (*FP*), the Correlation Engine (*CE*) and one Feature Consumer (*FCon-1*). Each one of the Feature Collectors only collects one Feature (denoted as *F-1* and *F-2* respectively). It is assumed that the Policy that is used by the Correlation Engine includes Signature and Anomaly definitions that make use of these Features. Furthermore, it is assumed that at least one of the defined Rules fires (as their Condition is fulfilled), which causes the creation of a new Feature by the Correlation Engine (denoted as *F-CE*). This Feature in turn is consumed by the Feature Consumer. The interaction among the logical roles is detailed in the following:

1. As the first step, the Feature Collector *FCol-1* measures the Feature *F-1*. Depending on the definition of the Feature, that is its encapsulation in Categories of cardinality N and depending on the actual issue of interest that the Feature Collector can observe, this leads to the creation of a set of appropriate Feature instances (denoted as *{F-1}*). For example, a Feature Collector deployed on a smartphone that measures a Feature which encapsulates the name of an app will create as many instances as there are apps installed on the smartphone.

2. In the second step, the Feature Collector transmits all measured Features to the Feature Provider. It does so by marshalling a message that encapsulates all Feature instances obtained in step 1 and the type of operation it requests from the Feature Provider (*store* in this case). The resulting request message is sent to the Feature Provider (likely via a network connection). The Feature Provider receives the request, performs the operation (stores all contained Feature instances), and acknowledges the request with a response. Throughout the example, it is assumed that all operations are successful. Thus the Feature Provider simply acknowledges the operation (OK).

3. Similar to the first step, the second Feature Collector now measures *F-2*, resulting in M instances of the Feature (denoted as *{F-2}*).

4. Similar to the second step, Feature Collector *FCol-2* sends a request to store the instances to the Feature Provider, which is again successfully acknowledged (OK).

5. In the fifth step, the Correlation Engine sends a request to retrieve the Features *F-1* and *F-2* to the Feature Provider. Note that only Feature identifiers are mentioned in the request, no Feature instances. The Feature Provider processes the request and responds with the set of Feature instances that have been measured before.

6. Now the Correlation Engine can evaluate the Policy. Based on the set of Feature instances that it has received in the previous step, it checks each Rule of the Policy. That is, the Condition of each Rule is evaluated. If the Condition is *true*, the corresponding Rule *fires*. In this example, at least one Rule fires, which leads to the creation of a set of Features (denoted as *{F-CE}*).

7. Afterwards, the Correlation Engine sends a request to store the created Features to the Feature Provider. Again, this is successful and acknowledged by a response.

8. In step eight, the Feature Consumer requests to retrieve the Feature *F-CE* from the Feature Provider. The response contains all instances of the Feature that were previously stored on behalf of the Correlation Engine.

9. In the last step, the Feature Consumer processes the received Features. The details of this last step are out-of-scope for the CADS architecture. However, a common example is to extend a flow controller with functionality to fulfill the role of a Feature Consumer. Thus, the flow controller can adjust its configuration based on the Features that were created by the Correlation Engine.

Note that this is an example for a communication flow. Aspects that vary depending on the concrete implementation of CADS within an IT infrastructure include

- the number of Feature Collectors,
- the number of Features that are collected by each Feature Collector,
- the number of Feature Consumers,
- the number of Features that each Feature Consumer retrieves from the Feature Provider,
- the number of Features that are actually used within the Policy of the Correlation Engine,
- the number of Rules that fire and the Actions that they imply (create, update, delete Features),
- the flow of communication may happen in a different temporal order. This means that requests can be send in parallel to the Feature Provider.

4.3 Correlation Model

So far, two parts of the CADS approach have been described in detail: the CADS conceptual model in Section 4.1 and the CADS architecture in Section 4.2. In the next step,

the correlation model is defined. It details how the Correlation Engine evaluates its Policy based on Feature instances that are retrieved from the Feature Provider. Algorithms are provided that specify how Signatures and Anomalies are evaluated. For the anomaly detection, the notion of training and testing phases as well as profiles for smartphones are introduced.

4.3.1 Policy Evaluation Overview

The general idea of the correlation model is that there is one Correlation Engine that evaluates one Policy. The Policy includes rules that encapsulate requirements for the secure integration of smartphones within an IT infrastructure. With respect to the scenarios defined in Section 2.2, these rules will enable to make smartphones visible throughout the IT infrastructure, enable context-related service provisioning, support to detect unwanted and malicious software as well as enable immediate reaction on identified threats. The same, single Policy is valid for all smartphones that are used within the respective IT infrastructure.

In essence, evaluation of the Policy is generally necessary when the Correlation Engine retrieves Feature instances for a smartphone from the Feature Provider. Evaluation is performed on a per smartphone device basis. The Correlation Engine evaluates the Policy for each smartphone separately and independently from each other. Based on the Feature instances that have changed, the Correlation Engine determines which Rules need to be evaluated. For each Rule, it checks whether its Condition (respectively its Signature and Anomaly components) use Features that have changed according to the last Feature instances retrieved from the Feature Provider. If that is true, the corresponding Rule is evaluated (that is all of its Signatures and Anomalies are evaluated). For each Rule whose Condition is true, the respective Actions are performed. Thus, it leads to a set of Feature instances that will be stored or deleted in the Feature Provider. Algorithm 1 summarizes the evaluation of the Policy.

The evaluation of a Rule's Condition requires to evaluate the Signatures and Anomalies that it is composed of. How these components are evaluated is detailed in the following.

Algorithm 1: Evaluation of the Correlation Engine's Policy

```
Data: Policy P, list of retrieved Feature instances featureList
Result: Actions that have been performed
foreach Rule r : P.ruleSet do
    if r.uses(featureList) then
        result = r.evaluate();
        if result == true then
            r.performActions();
        end
    end
end
```

Table 4.1: Exemplary Categories and Features.

Category	Cardinality	Features
app	N	Name, Rating
app.perm	N	Requested

4.3.2 Evaluation of Signatures

As defined in Section 4.1.3, a Signature is composed of Boolean and relational expressions which are formulated based on Features, more precisely their identifiers. However, the evaluation is performed against a set of Feature instances (with identifiers generated as discussed in Section 4.1.1). In order to evaluate a Signature, all Feature instances that match its Contexts must be considered. Depending on whether the Correlation Engine implements some form of caching for Feature instances, this might require additional retrieval requests that are sent to the Feature Provider. The resulting set of Feature instances that have to be considered can be organized as a tree structure. Nodes represent Categories, Features are represented as labels who are attached to the nodes. An example of such a Feature instance tree is given in Figure 4.5. It includes two Categories (each of cardinality N) with Features describing aspects of a smartphone app (name, rating, requested permissions). They are listed in Table 4.1. The tree depicts the situation when there are three apps, where the first one requests two permissions and the other two request one permission each.

In the following, examples are given that emphasize the problem of a reasonable Signature evaluation. They are summarized in Table 4.2. If the Signature is only composed of

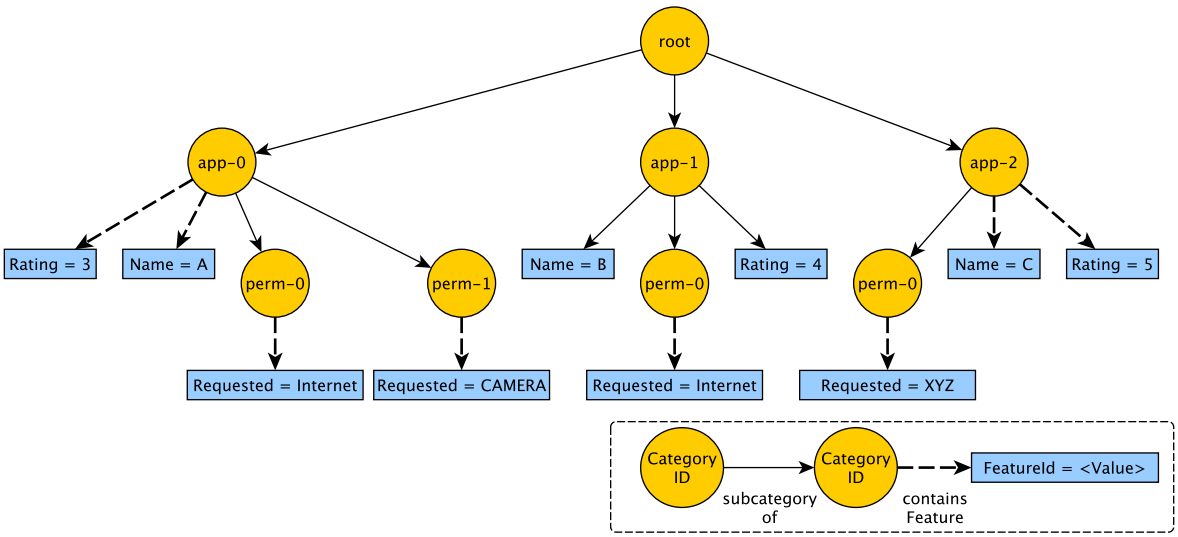

Figure 4.5: Tree representing Feature instances. Categories are depicted as orange circles. The text within the circles represents the identifier of the Category. Features are depicted as blue boxes. The text within the boxes represents the Feature identifier and the Feature's value.

Table 4.2: Signature expressions and their respective number of matches according to the Feature instance tree depicted in Figure 4.5.

Expression	Scope	Matches
app.Name = A	0	1
app.Rating > 3	0	2
app.perm.Requested = INTERNET	0	2
app.Name = A ∧ app.Rating > 3	0	0
app.Name = A ∨ app.Rating > 3	0	3
app.Name = A ∧ app.perm.Requested = INTERNET	0	1
app.perm.Requested = INTERNET ∧ app.perm.Requested = CAMERA	0	0
app.perm.Requested = INTERNET ∧ app.perm.Requested = CAMERA	1	1

one relational expression, the evaluation basically requires to traverse the Feature instance tree, searching for nodes that represent the Categories that are used by the expression. For each matching node, it is checked whether it has a label attached that fulfills the relational expression. For example, the relational expression $app.Name = A$ leads to a traversal of the tree that searches for nodes that represent an instance of the app Category. In the example above, there are three such nodes (app-0, app-1, app-2). Then, for each node it is checked whether it has a label that fulfills the relational expression. This is only true for one node (app-0). Thus, the Signature matches. A Rule whose Condition is only based on such a Signature would fire.

However, the evaluation of a Signature that is composed of more than one relational expression is more complex and introduces subtle challenges. The expression ($app.Name = A \wedge app.Rating > 3$) should match any app whose name is A *and* which has a rating that is higher than three. In the example above, there is no instance that matches this expression. This result is basically obtained by applying multiple traversals of the Feature tree. That is, like for the first simple expression, the tree is searched for nodes that fulfill the first relational expression. For each of these nodes, only their subgraph is traversed again in order to find a match for the second relational expression. This process is repeated for each relational expression. This behavior changes if instead of a logical $\wedge$ the logical $\vee$ is used. In this case, there is no subgraph reduction performed between the relational expressions. Instead, it is always traversed from the root node for each relational expression. That is, the Signature would match three times. If logical $\wedge$ and $\vee$ are mixed, $\wedge$ has a higher precedence than $\vee$.

The algorithm presented so far introduces a problem when a Signature should match any app that has the permissions INTERNET and CAMERA. The algorithm would search the first matching permission node (perm-0) and consider only its subgraph for the second relational expression. Thus, in contrast to what was expected by the Signature definition, there would be no match. In order to circumvent this problem, the notion of a scope for Signature evaluation is introduced. The scope defines how the evaluation algorithm determines the subgraph that should be traversed once a matching node was found when a logical $\wedge$ is used to concatenate two relational expressions. The default behavior described above uses a scope of zero. That is, only the subgraph whose root node is the matching node is considered. A scope that is larger than zero causes that a larger subgraph is considered for subsequent relational expressions. More precisely, the

scope equals the number of parent nodes that are "traced back" in order to determine the root node of the subgraph that is traversed for the subsequent evaluation. That is a scope of "1" causes to consider the subgraph whose root node is the parent of the node that caused the match. A scope of "2" makes the grandparent of the matching node the root for the subgraph traversal. Thus, the previously mentioned Signature that aims to find apps who have both the INTERNET and the CAMERA permission can be defined with a scope that is set to 1. After the first matching node was found (perm-0), the root for the subgraph that is considered for the next relational expression is set to (app-0). Thus, the Correlation Engine can find the second matching node (perm-1). In contrast, in order to specify a Rule that fires if there is at least one app that has the INTERNET permission and at least one app that has the CAMERA permission, two Signatures that are each composed of one single relational expression can be used.

In general, the exact number of matches for a certain Signature is not relevant to evaluate the Condition of a Rule. The Correlation Engine considers a Signature to be fulfilled in terms of evaluating the Boolean expression of the Condition if there is at least one match. However, if there is need to determine the exact number of matches for a certain Signature for further processing, the Correlation Engine can be configured to do so as well. This generally allows to define further expressions that also consider how often a certain Signature was matched against a certain Feature instance tree.

With the Signature evaluation as described in this section, it is possible to define complex patterns based on Features. When a logical $\wedge$ is used to concatenate two relational expressions A and B within one Signature, the result of the first expression determines the subgraphs that are considered in order to evaluate the second expression. This also holds if there are more than two expressions. How these subgraphs are composed can be adjusted by defining a scope parameter. The evaluation algorithm is sufficient to handle expressions that are needed for the scenarios that are addressed in this thesis. Note that efficiently evaluating complex Boolean expressions like those that are used within Signatures is a subject of current research [162]. Algorithm 2 summarizes the evaluation of Signatures at a high level.

Algorithm 2: Evaluation of a Signature

```
Data: Signature S, Feature Instance Tree FIT
Result: true or false
relationalExpressions = S.getRelationalExpressions();
listOfNodes = FIT.getRootNode();
foreach RelationalExpression rExp : relationalExpressions do
    listOfMatchingNodes = evaluateExpressionForNodesInList(rExp, listOfNodes);
    if S.nextBooleanOp == AND then
        listOfNodes = applyScopeForNodesInList(listOfMatchingNodes);
    else if S.nextBooleanOp == OR then
        if listOfMatchingNodes.size() > 0 then
            return true;
        end
        listOfNodes = FIT.getRootNode();
    else
        return listOfMatchingNodes.size() > 0;
    end
end
```

4.3.3 Evaluation of Anomalies

As defined in Section 4.1.4, an Anomaly is composed of Boolean and relational expressions which are formulated based on Hints. Thus, its evaluation involves a two step process. First, the Hints that are used in relational expressions by the Anomaly are evaluated. Second, the overall Boolean expression of the Anomaly can be evaluated as well (based on the results of the relational expressions).

The Anomaly defines which Contexts are relevant for the evaluation. That is, the same Contexts apply for each one of the Hints. However, the Hints themselves may be defined based on different Features and each can use a different Procedure. During the evaluation of an Anomaly component, it is thus necessary to forward the Feature instances to the respective Hints properly, which in turn will call their Procedure. That is, for each Hint all Feature instances that match the Anomaly's Contexts must be forwarded. Depending on the implementation of the Correlation Engine, more precisely whether it supports caching of Feature instances or not, this might require further requests for retrieval that are send to the Feature Provider. Once all necessary Feature instances are available, the Hint itself can be evaluated. This causes the Feature instances to be forwarded to the

Hints Procedure. The details of how the Procedure itself handles the Feature instances is out of scope. When the Procedure returns its scoring result, the relational expression the respective Hint is used in can be evaluated. Once all relational expressions are evaluated, the overall Boolean expression can be evaluated as well. Algorithm 3 summarizes the evaluation of an Anomaly.

Algorithm 3: Evaluation of an Anomaly

```
Data: Anomaly A
Result: true or false
listOfContexts = A.getContexts();
listOfHints = A.getHints();
foreach Hint h : listOfHints do
    featureIds = h.getFeatureSet();
    featureInstances = getFeaturesByContext(listOfContexts);
    score = h.evaluate(featureInstances, listOfContexts);
    h.setScore(score);
end
return A.evaluateBooleanExpression();
```

4.3.4 Training and Testing Phases

The previous section described the basic evaluation of Anomalies. The details of the anomaly detection techniques are part of the Procedure components. However, the general concept of anomaly detection introduces another requirement that must be met by the Correlation Engine: the ability to distinguish between training and testing phases. Training phases are used by some anomaly detection techniques in order to learn the normal behavior based on training data. Those are also referred to as supervised and semi supervised anomaly detection techniques [96]. After the training is done, those techniques can detect anomalies during the testing phase. In contrast to that, unsupervised techniques do not require training at all. That is, they can be used to detect anomalies directly during the testing phase. Nevertheless, in case training-based techniques should be used, the Correlation Engine must be able to handle those two phases properly.

The challenge for CADS and the scenario of smartphone security is that each smartphone will have its own, normal behavior. This is due to the fact that smartphones are normally used by a single user. However, the way each of them uses their smartphone

will vary. Thus, the training of anomaly detection techniques must be device specific. The CADS approach to employ proper training leverages the Correlation Engine's Policy that was engineered by a domain expert. For a given domain, the Policy especially mentions relevant Signatures and Anomalies, both in conjunction with associated Contexts. The training as employed by the Correlation Engine is based on this Policy. More precisely, for each smartphone device that has provided training data, the Policy is parsed for Condition statements that use Anomalies. Depending on these Anomalies (which are formulated based on Features via Hints), the training data can be searched for matching Feature instances. As already stated, defined Context instances can limit the set of Feature instances that are considered for each Anomaly (for example when only Features should be considered that were obtained during working hours).

Once the set of Features for an Anomaly has been identified, the Procedures that are used by the Hints can train the normal behavior (for example the number of SMS messages that are usually sent during working hours). This is done for each Rule that includes Anomalies within its Condition statement. Within the CADS approach, the basic algorithm that is used for training is the same as for evaluating an Anomaly (see Algorithm 3). There are just two major differences: (1) training is performed on training data and (2) each Hint must indicate to its Procedure that it is currently called during the training phase (for example by passing an appropriate flag).

The result of the training is a smartphone specific profile. It has the same structure as the Correlation Engine's Policy (that is the same set of Rules, Signatures, Anomalies, etc.), but references trained instances of Procedures. That is, for each smartphone exists a profile that encapsulates its normal behavior by means of trained Procedure instances. Procedures that do not need any sort of training are simply omitted during the training phase. Once a profile has been trained, it is used for any further evaluation tasks. That is when the Correlation Engine retrieves new Feature instances from the Feature Provider, it determines to which smartphone they belong to, looks up the respective profile and evaluates the contained rules as described in Section 4.3.1. The Correlation Engine is able to determine which Features belong to which smartphone based on the retrieval functionality of the Feature Provider. After training has been done for all smartphones, the training phase is finished and the testing phase starts. Note that it is not supported to switch between these two phases. That is, if the training should be redone (for example when better training data is available), this requires to end the testing phase and start

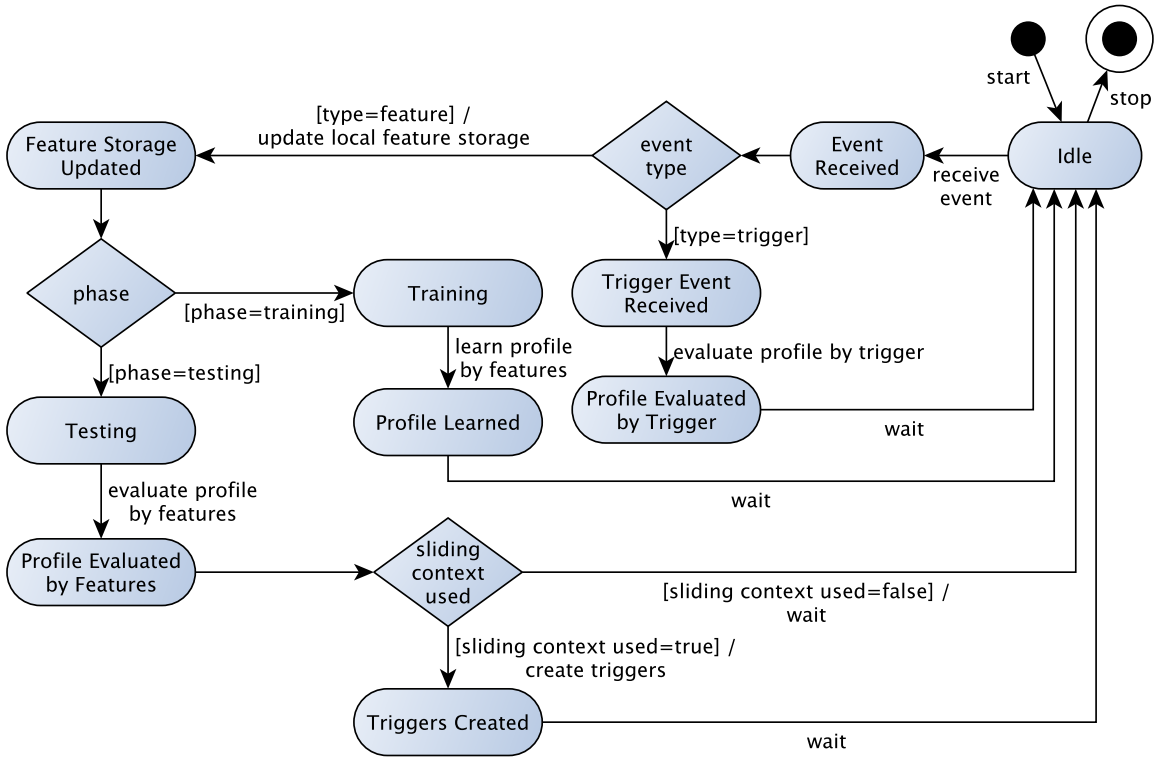

Figure 4.6: Correlation Engine State Machine (UML state machine diagram).

over again with a new training phase. Trained Procedures are reset at the end of the testing phase. That is, they do not maintain their state for the next training phase.

4.3.5 Correlation Engine Workflow

So far, the Correlation Model introduced the basic concept of evaluating Policies, Signatures and Anomalies. Furthermore, the differences between testing and training phases were discussed. However, it has not been defined yet how the Correlation Engine works internally and by what events the evaluation can be triggered. This is done in the following by providing a simplified state machine of the Correlation Engine. It is depicted in Figure 4.6. The state machine generally is separated into three branches: one for the training phase, one for the testing phase and one for handling so-called Triggers. When started, the Correlation Engine enters the “Idle” state. In this state, it basically waits for events that cause a state change. There are two types of events that can cause such

a state change: (1) events that indicate the availability of new Features (event type is "feature") and (2) events that express that a Trigger has fired (event type is "trigger"). Thus, after the Correlation Engine has received a new event, it enters the state "Event Received". From there on, it dispatches the further processing based on the event type it has received.

The first type of events occur when the Correlation Engine has retrieved new Feature instances from the Feature Provider. The details on how the new Feature instances are obtained are specific for the communication protocol that is used in order to implement the CADS approach. However, the Correlation Engine must be able to determine to which smartphone the Features belong to. As already stated in Section 4.3.1, the retrieval of new Features is the most common case that requires to evaluate the Policy. In this case, the Correlation Engine updates its own local storage of Feature instances and enters the state "Feature Storage Updated". The details on how the Correlation Engine implements its local storage are not specified.

The further processing depends on whether the current phase is training or testing. This is set as a configuration parameter in the Correlation Engine. That is it must be defined a priori and does not depend on the occurrence of certain events. The workflow in case the current phase is training is highlighted in Figure 4.7. In this case, the Correlation Engine enters the state "Training". Afterwards, it learns the profile for the respective smartphone device as described in Section 4.3.4. Again note that the evaluation of Policies and thus the learning of profiles is device specific. After the profile has been learned, the Correlation Engine enters the state "Idle" again and waits for the next event.

The workflow that takes place in case the current phase is testing is depicted in Figure 4.8. After the Correlation Engine has updated its local storage, it enters the state "Testing". After that, it evaluates the profile for the respective smartphone that was obtained from the Policy during the training phase. The general process of evaluating a Policy (respectively a profile that has been derived from it through training) was detailed in Section 4.3.1. In contrast to the training phase, this might lead to Actions that are performed when Conditions of Rules are fulfilled. After the profile has been evaluated, the last step is to check if so-called Triggers need to be established. Triggers are another mechanism that can cause the evaluation of a profile (besides the retrieval of new Features either during training or testing phase). Triggers address the fact that some rules might need

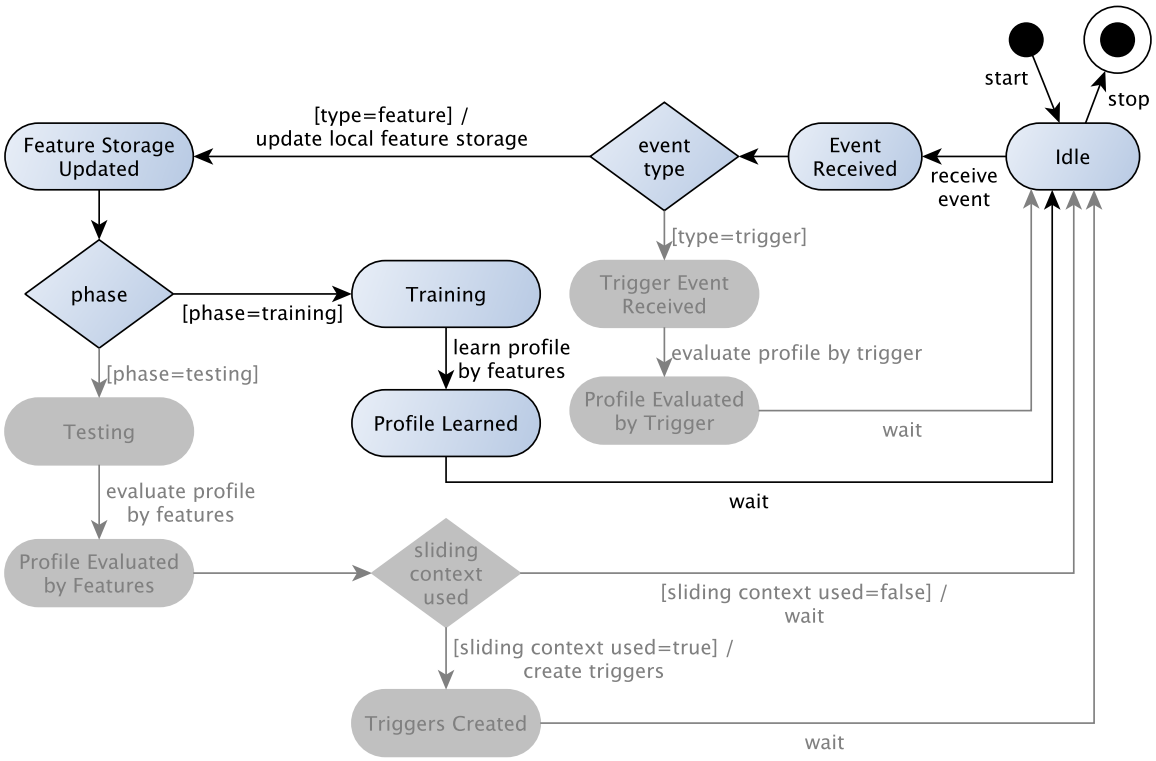

Figure 4.7: Correlation Engine Training Phase (UML state machine diagram).

to be reevaluated although there have been no updates of any Feature instances. This is always the case when Signatures or Anomalies use sliding Contexts (see Section 4.1.2).

Signatures and Anomalies reference Features. At evaluation time, the necessary Feature instances are retrieved from the Feature Provider. Contexts that are used by Signatures and Anomalies limit the set of Feature instances that are considered for further evaluation. Let F be the set of Feature instances that have to be considered for evaluation of a Signature or an Anomaly. If only normal Contexts are used, F does only change in the event that either a Feature Collector or the Correlation Engine uses the storage functionality of the Feature Provider. That is F only changes if either new Feature instances are created or existing ones are changed respectively deleted.

However, this is not true if sliding Contexts are used. Sliding Contexts are used in order to express temporal expressions that are relative to the current moment in time. For example, a normal Context can express that only Features who have been collected between 8:00AM and 18:00PM are considered for evaluation purposes. In contrast, a sliding Context can express that only Features who have been collected in the last 15 minutes should be considered. Thus, the set of Features F that are relevant during the evaluation of a Signature or Anomaly can change without any events that indicate the change of the Feature instances themselves.

Assume that a Signature or Anomaly which uses such a sliding Context was evaluated at time t_0, leading to the result r_{t_0}. At that time, the set of Feature instances that had to be considered is denoted as F_{t_0}. Further assume that an event that indicates a change of Feature instances does not occur before t_n, with $\Delta t_{n,0} = t_n - t_0$. Then, the result of the evaluation of the respective Signature or Anomaly can nevertheless change in the time between t_0 and t_n as the set of Feature instances that match the sliding Context can change. That is, at any time t_1 with $\Delta t_{1,0} = t_1 - t_0$ and $\Delta t_{1,0} < \Delta t_{n,0}$, evaluating the respective Signature or Anomaly can yield different results depending on the definition of the sliding Context. The question that arises is at which moments in time the Correlation Engine should reevaluate a certain Policy (respectively profile), although there have been no events that indicate the change of any Feature instances. Simply refusing to do additional evaluations and solely relying on the events of type “feature” is not sufficient. In this case, the results of the last evaluation might contradict the real situation. On the other hand, simply evaluating all profiles for all smartphones in an infinite loop would minimize the delay between a change of the Feature instance set and

the next evaluation process. However, this brute force approach is not practical as it will exhaust the resources available on the system that fulfills the role of the Correlation Engine.

The approach taken by CADS is a compromise of the two alternatives described above and relies on the use of so-called Triggers. After a profile has been evaluated, the Correlation Engine checks whether Signatures and Anomalies that use sliding Contexts were involved. For each such Signature or Anomaly, a Trigger is created. This Trigger basically renders the moment in time when the respective Signature or Anomaly (and thus the Rules that they are used in) should be reevaluated again. The exact amount of time that should pass before a Trigger fires can be configured at will. However, it should be taken into account that the lower this amount of time is, the more resources need to be available for the Correlation Engine. Coming back to Figure 4.8, if sliding Contexts were used, the necessary Triggers are created. After that, the Correlation Engine has reached the "Triggers Created" state. From there on, it enters the "Idle" state again and waits for the next event.

The last branch that is discussed here addresses the handling of Triggers that have fired (depicted in Figure 4.9). In this case, the Correlation Engine receives an event of type "trigger". Thus, it enters the state "Trigger Event Received". This causes the Correlation Engine to evaluate the respective profile. The evaluation basically works similar to the one that is carried out when an event of type "feature" would have been received. There are only two major differences: (1) the evaluation does only evaluate one single Rule (the one that initially caused the corresponding Trigger to be created) and (2) the evaluation does not lead to the creation of new Triggers. Instead, the existing Triggers remain valid and will fire again once the respective amount of time has passed again. Note that Triggers are only created if the Correlation Engine is working in the testing phase.

4.4 Domain-specific Mapping

The previous sections introduced the generic parts of the CADS approach: its conceptual model, its architecture and the correlation model. However, in order to actually use CADS within an IT infrastructure, a crucial part is still missing: the domain-specific mapping. That is, for any IT infrastructure that aims to secure their integration of smartphones based on the CADS approach, a specific domain instance of the generic parts must be

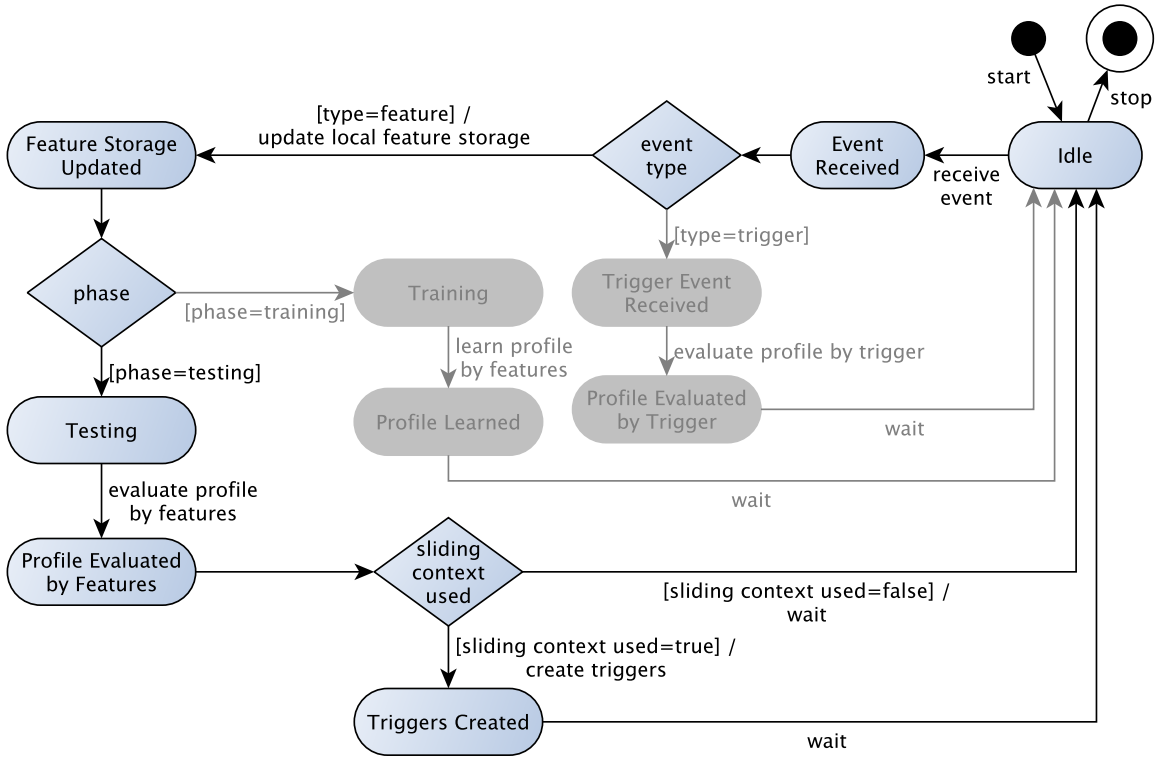

Figure 4.8: Correlation Engine Testing Phase Feature Event (UML state machine diagram).

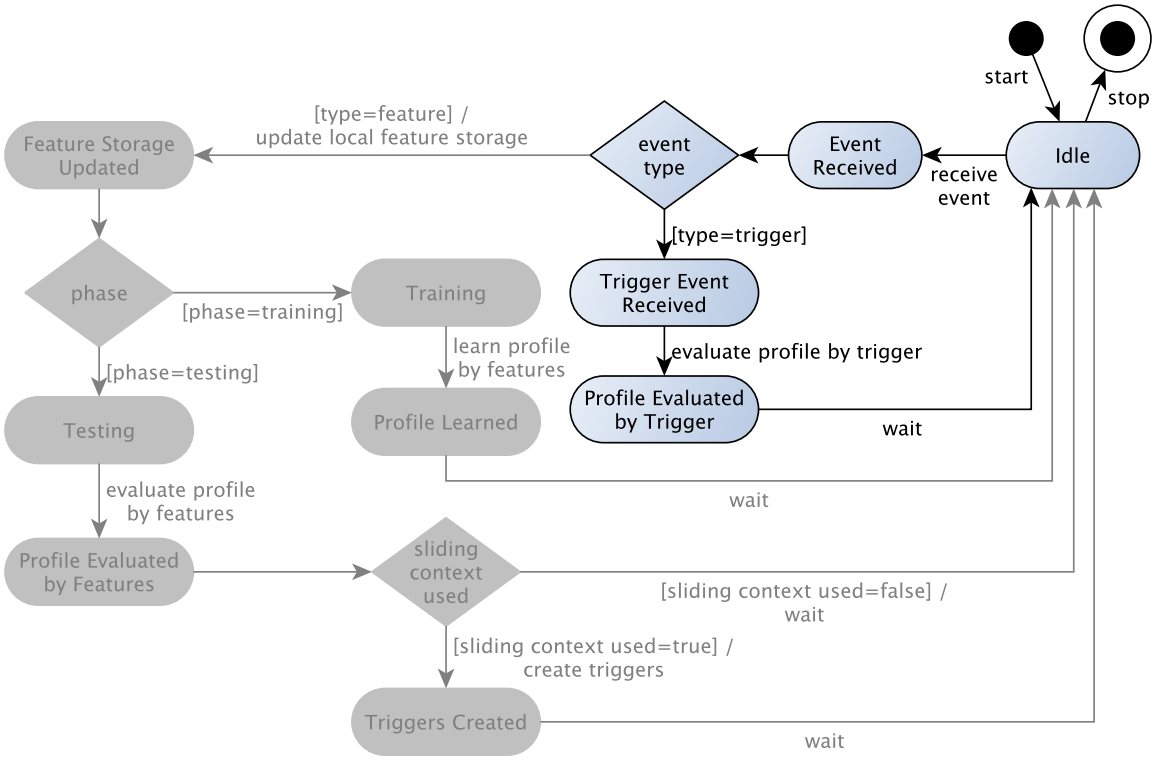

Figure 4.9: Correlation Engine Testing Phase Trigger Event (UML state machine diagram).

derived. The generic questions that need to be addressed in order to derive such a domain instance are detailed in Section 4.4.1 by defining a process model. Afterwards, a domain instance is derived for the reference IT infrastructure and the scenarios described in Chapter 2.

4.4.1 Process Model to Derive Domain Instances

Any domain instance generally addresses the following questions:

- What Features should be collected? That is, what is the basis for any Signature and Anomaly components that are used within the Policy of the Correlation Engine?
- How does the Correlation Engine react on Rules that have fired? That is, what Features are created by the Correlation Engine itself?
- What Context Parameters that are relevant for the target domain need to be considered?
- How are the logical roles of the CADS approach mapped to the target IT infrastructure? Especially Feature Collectors and Feature Consumers should be deployed in such a way that they can leverage the capabilities of existing services if possible.
- How does the Policy look like that is evaluated by the Correlation Engine? That is, encapsulating the demands that the maintainer of an IT infrastructure has regarding the secure integration of smartphones in appropriate Rules.

In order to answer these questions properly, a generic process model is specified that defines how domain instances for the CADS approach should be derived (depicted in Figure 4.10):

1. **Definition of Features**. In the first step, Features that are relevant for the specific domain are defined. As mentioned in Section 4.1.1, the Feature space is hierarchically structured by means of Categories. Since each domain will have its own specific Policy regarding the secure integration of smartphones, it is not possible to provide a single set of Features that is generally accepted. Instead, what Features are necessary depends on the scenarios and use cases that are addressed. However, two types of

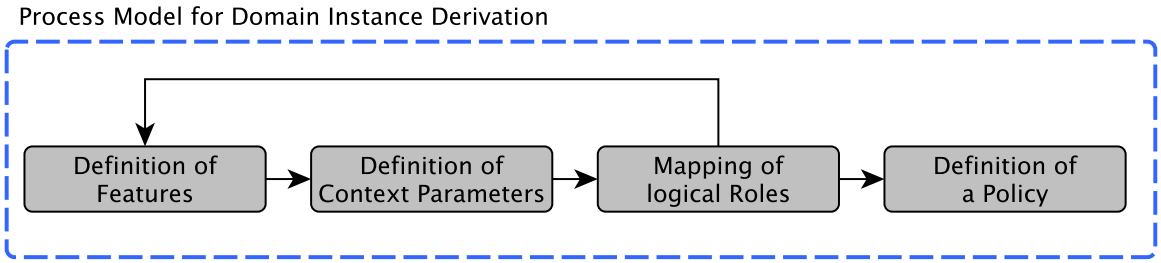

Figure 4.10: Process Model for deriving domain instances.

Features can generally be distinguished: (1) those that are collected by arbitrary Feature Collectors and (2) those that are created by the Correlation Engine when a Rule of its Policy fires.

2. **Definition of Context Parameters**. A set of relevant Context Parameters needs to be defined. Similar to the definition of Features in step 1, it cannot be assumed that there is a generally accepted way that defines which context information should be encapsulated by which Context Parameters. However, it is assumed that the set of reasonable Context Parameters will be less diverse among different domains compared to the set of Features.

3. **Mapping of logical roles**. In this step, the logical roles defined in Section 4.2 need to be mapped to the target IT infrastructure. It is expected that the Feature Provider and Correlation Engine roles will demand for new systems that are deployed. However, existing systems are expected to fulfill the role of a Feature Collector or a Feature Consumer. Based on this mapping, a gap analysis can be performed in order to detect Features that have been defined in step 1 but that cannot be collected in the absence of appropriate Feature Collectors. If there is a gap, new systems need to be added to the target IT infrastructure that work as Feature Collectors in order to collect the missing Features. If that is not possible, the definition of Features needs to be revised. Note that multiple Feature Collectors can be deployed in order to collect the same Feature.

4. **Definition of a Policy**. In the last step, the Policy that should be evaluated by the Correlation Engine is defined. This basically requires to define Signature, Anomaly

and Context components, make use of them as part of Rules and define appropriate Actions for Rules that fire. Note that although the notion of the abnormal behavior is mentioned in the Policy as part of the Anomaly definition, the concrete technique that is used depends on the implementation of the approach. For example, where the Policy states that an abnormal use of a smartphone's sensors should be detected, various methods (even combined ones) may be used within an implementation in order to detect this.

Based on the domain instance, the CADS approach can be implemented for a target IT infrastructure.

4.4.2 An Example for a Domain Instance Derivation

The process model described above is now applied to derive a domain instance for the reference IT infrastructure and the scenarios defined in Chapter 2. Concerning the different available smartphone platforms, the domain instance focuses on Google Android.

Definition of Features

The difficulty to define a set of reasonable Features is closely related to the scenarios that should be addressed within a certain domain. For example, in order to come up with Features for the scenarios "Smartphone Visibility", "Context-related Service Provisioning" and "Policy-based Enforcement" is pretty straight forward. However, to tackle scenario "Detection of Malicious and Unwanted Apps" is more challenging as will be shown later in this section. In the following, a brief overview of some Features and the respective Categories is given. The hierarchy of the Categories is depicted in Figure 4.11, starting from a virtual root Category. The sub Category relationship is denoted by directed arrows. A full list of all defined Features for the domain is given in Table A.2.

In the scenario "Smartphone Visibility", it is required that certain services are not allowed to be accessed by a smartphone. Thus, the service itself must be able to determine whether a certain request was issued from a smartphone or not. Rendering this fact as a Feature is trivial. It simply requires one single Feature acting like a flag to indicate that a certain device is a smartphone. For this domain mapping, the Feature is defined as *correlationresult.smartphonevisibility.IsASmartphone* whose value can either be "true"

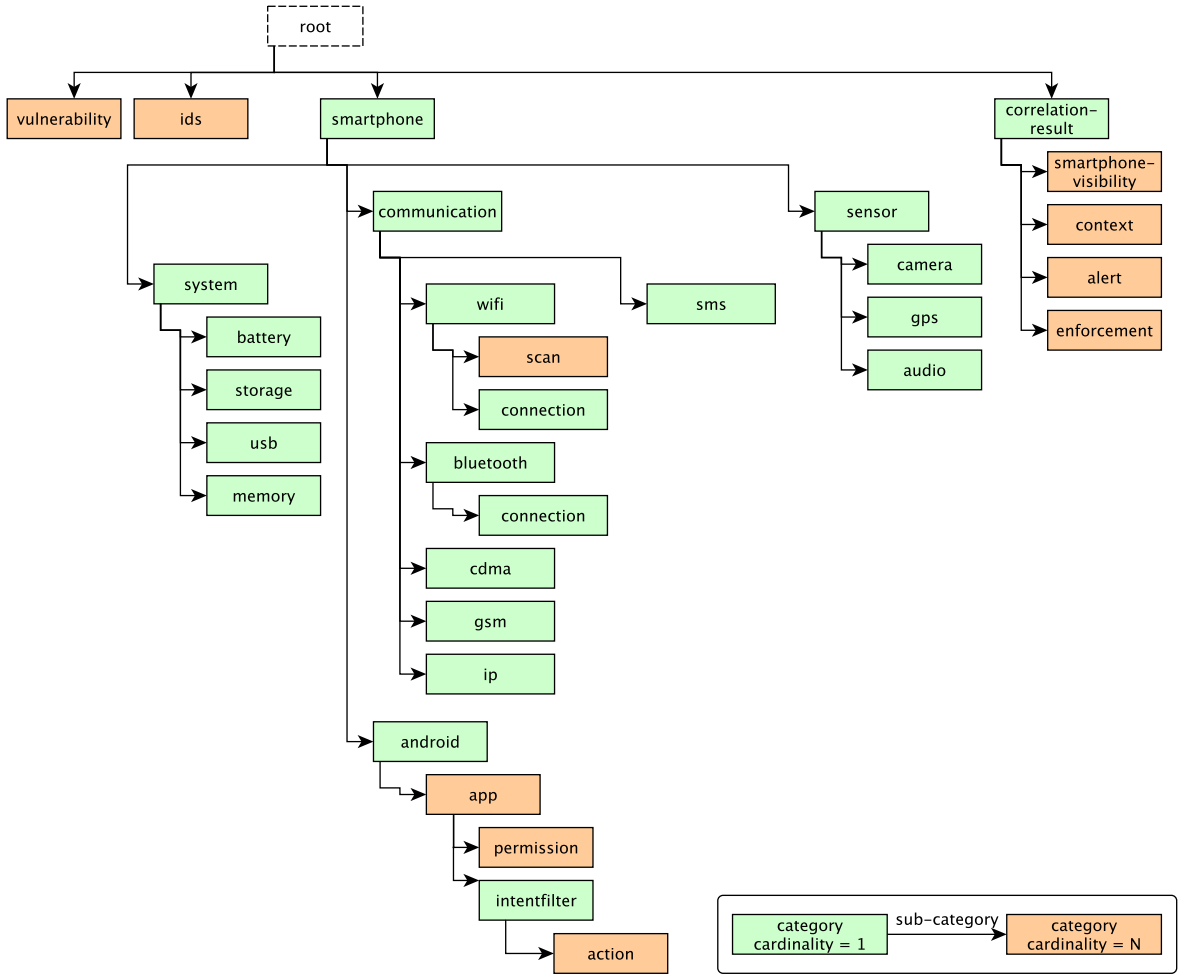

Figure 4.11: Categories of the exemplary domain instance.

or "false". The Correlation Engine is responsible for creating this Feature based on other Features that indicate the respective device is a smartphone, such as the presence of an IMSI or IMEI number. The mapping which Features need to be present in order to flag the respective device as a smartphone is defined in the Policy of the Correlation Engine by defining appropriate Rules.

Scenario "Context-related Service Provisioning" is similar to the first one. However, in this case a service that is accessed must consider the Context of the requesting smartphone. In order to provide the latest values for the defined Context Parameters, a dedicated "dummy" Feature is defined. That is, the value of this Feature is generally irrelevant, it is only used in order to communicate updated values for Context Parameters from a Feature Collector to the Feature Provider. Thus, it is referred to as *smartphone.ContextPing*. The Correlation Engine can retrieve this Feature in order to check if defined Contexts are fulfilled or not. It does so by defining Signatures that make use of the *ContextPing* Feature and exactly one Context that is of interest. Depending on whether the Signature matches, the respective Context is fulfilled or not. By creating appropriate Rules that use the Signature, the Correlation Engine can create new Features that render the outcome of the respective Rule checks. For this purpose, two Features are defined: *correlationresult.context.IsFulfilled* and *correlationresult.context.IsNotFulfilled*. The values of the Features encapsulate the identifier for the respective Context. A service that also fulfills the role of a Feature Consumer can retrieve these Features in order to reason about the Context of the requesting smartphone.

In fact, both of the two described scenarios could also be realized without the participation of the Correlation Engine at all. As there are no complex Signatures required or anomaly detection techniques involved, a Feature Consumer could easily implement logic that allows to reason about whether a device is a smartphone or not and what Context it is currently in on his own. However, involving the Correlation Engine provides the benefit that the knowledge about what Features identify a smartphone and what Contexts are relevant is maintained in one single Policy.

The scenario "Policy-based Enforcement" generally requires that an enforcement can take place upon the detection of a threat. In terms of CADS, that means that appropriate Features that render the desired enforcement action need to be created when Rules that encapsulate threats fire. In order to address this, a Feature called *enforcement.EnforcementAction* is defined. Its value encapsulates the type of enforcement that

should be employed. This value must be set in such a way that the receiving Feature Consumer is able to interpret it correctly. For the given scenario, it is assumed that the enforcement is done by a packet filter like iptables. Thus, the value to the Feature is set to the command line string that should be executed. For example, it could trigger the execution of a shell script that blocks traffic for the IP address that is currently used by the smartphone. Of course, this requires that the Correlation Engine knows the IP address of the smartphone, and thus an appropriate Feature for it is necessary as well. If only alert messages should be distributed to Feature Consumers, Features of the Category *correlationresult.alert* can be used.

Clearly, the most challenging scenario to address is "Detection of Malicious and Unwanted Apps". As pointed out in Section 3, various approaches exist that aim to detect malicious apps. However, the set of features[1] that have been used for detection tasks is diverse. As the scenario aims to primarily detect the presence of sensory malware, Features that encapsulate information about installed apps and the status of the smartphone's built-in sensors are primarily necessary. In general, Features are included that (1) are obtained on the smartphone itself and (2) Features that originate from security and management services that are present in the reference IT infrastructure.

Some of the presented Features are specific for the Android platform (such as those to encapsulate an app's requested permissions). However, the notion of those Features can generally be adapted to other platforms as long as they provide a similar, permission based access control model. The chosen Features are motivated by previous work that has been published recently, primarily inspired by Enck et al. [129, 14], Barrera et al. [87] and Shabtai et al. [18] for Android specific Features and Schmidt et al. [147] for user specific and general platform features. The integration of events originated from network-based monitoring systems (like an IDS), results of vulnerability scanners (like OpenVAS) and Features obtained from app stores like Google Play is a contribution of this thesis. Although the general idea to integrate both host-based and network-based approaches for smartphone malware detection has already been proposed by Miettinen et al. [95], their work does not provide any detailed concepts, especially no holistic model to capture both smartphone and network-based Features as this thesis does.

[1] This refers to the general meaning of the term feature, not to the Feature defined as part of the conceptual model.

In order to benefit from Features that can be contributed by intrusion detection systems (IDS) and vulnerability scanner, two dedicated categories are defined. Both have cardinality N and include Features that describe events generated by IDS respectively information about vulnerability reports generated by vulnerability scanners. The most complex category tree is started by the Category *smartphone*. It includes various sub-categories to describe various Features of a smartphone. This especially includes basic information about the smartphone (*smartphone.system*), its sensors (*smartphone.sensor*), its communication capabilities (*smartphone.communication*) and the installed apps (*smartphone.android.app*).

Note that in general there is no "right or wrong" regarding the definition of Features and Categories. That is, there will always be multiple possibilities to address a specific scenario. However, the way the Features are defined and organized by means of Categories affects the definition of the Policy.

Definition of Context Parameters

For the exemplary domain instance, there are only three Context Parameters defined:

- *Timestamp*: The moment in time when the respective Feature was measured.
- *Longitude*: The longitude coordinate from GPS position.
- *Latitude*: The latitude coordinate from GPS position.

These Context Parameters are sufficient in order to select Feature based on two aspects: when they were measured and where they were measured. This only requires to specify appropriate Context instances. The timestamp parameter is expected to be available at any Feature Collector. However, the GPS coordinates will likely be only available for Features that are measured on a smartphone.

Mapping of Logical Roles

In this step, the logical roles are mapped to the reference IT infrastructure described in Section 2.1. The same infrastructure is depicted in Figure 4.12, with the logical roles of the CADS approach added to it.

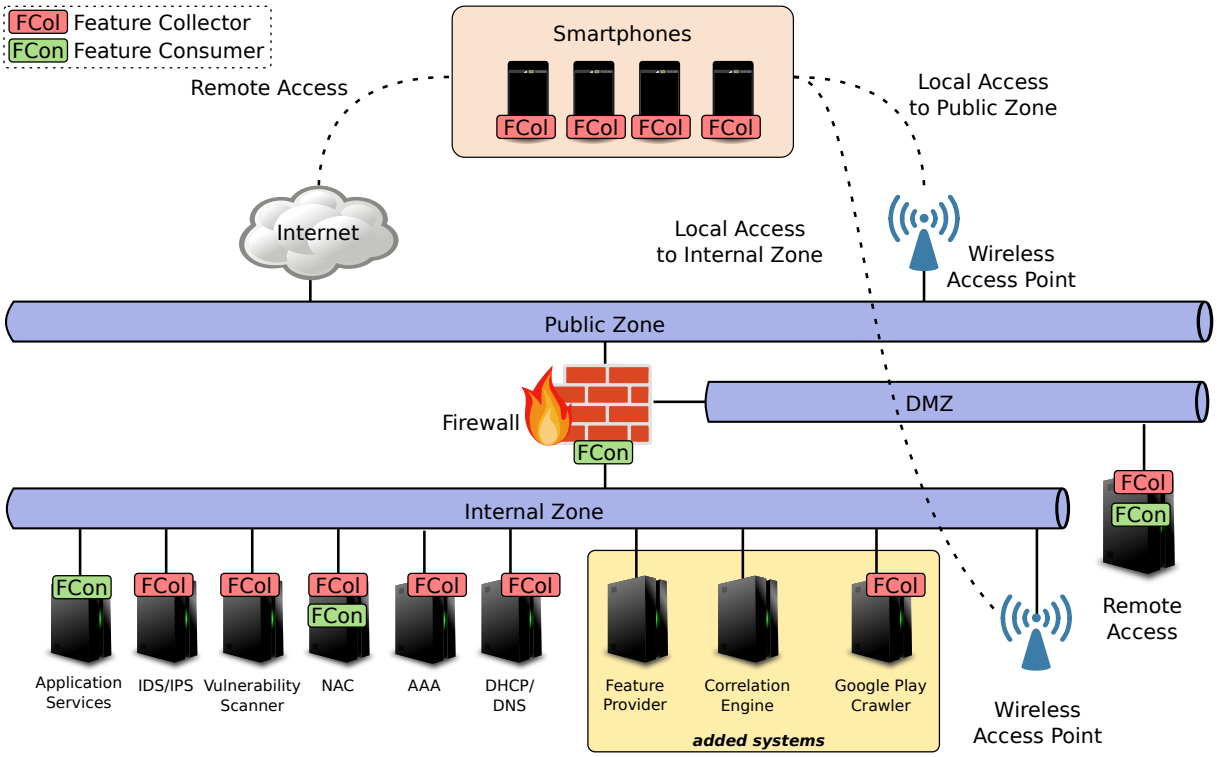

Figure 4.12: Reference IT infrastructure extended with CADS roles. Icons taken from Openclipart [27].

Feature Collectors are mapped to services that can contribute Features that were defined in the first step. This especially includes security services like the IDS and the vulnerability scanner (which are responsible to collect the Features of the Categories *ids* and *vulnerability*). Smartphones themselves are responsible for collecting Features of the Category *smartphone*. However, not all of these defined Features can directly be collected on a smartphone. For example, the Features *smartphone.android.app.Rating* and *smartphone.android.app.Downloads* are not directly available on a smartphone. Instead, it is necessary to query the Google play app store to obtain up-to-date values for the respective Features. Thus, a new system that hosts a service to crawl the Google Play app store was added to the reference IT infrastructure. The Correlation Engine and the Feature Provider were added as new systems as well as none of the existing services is expected to be able to fulfill these roles.

Feature Consumers are mapped to two types of services: (1) application services that need to process results of the Correlation Engine in order to determine if a certain request originated from a smartphone and in what Context the smartphone is currently operating in. (2) Services that can employ enforcement actions. Besides the Firewall, these also includes the NAC and Remote Access services (as they can choose to interrupt the connection of a smartphone). Note that NAC and Remote Access Services are also expected to work as Feature Collectors as they can obtain Features like the current IP address of a smartphone. This leads to a situation where the same Feature can generally be collected by numerous Feature Collectors. There are generally three options to solve this issues:

1. Choose only one Feature Collector that is responsible to collect the respective Feature.

2. Allow multiple Feature Collectors to collect the same Feature. That is, Feature Collector A can update the Feature that was collected by Feature Collector B and vice versa. However, this approach will lead to a higher rate of Feature updates, and thus to a higher amount of processing that must be done by the Feature Provider and the Correlation Engine. As a consequence, this option should be omitted if possible.

3. Refactor the modeling of Features and Categories so that there is no collision anymore. This can for example be achieved by introducing sub Categories for each Feature Collector.

In general, the question whether a certain service should act as Feature Collector and/or Feature Consumer can only be answered for a concrete domain. For example, the application services themselves could generally also act as Feature Collectors if there is a scenario in the concrete domain that demands it. For example, the scenario "Smartphone Visibility" states that some services are not allowed to be accessed by smartphones, requiring them to fulfill the role of a Feature Consumer. However, building on top of that, another scenario might require that smartphones who frequently try to access services they are not allowed to access are identified. In this case, the respective services can also act as Feature Collectors in order to collect Features that render unsuccessful access requests performed by smartphones.

Definition of a Policy

The last step is to define a Policy that makes use of the previously defined Features, Categories and Context Parameters. The following example will focus on the detection of one type of sensory malware as described in Section 2.2.3. That is, an app aims to capture sensor data and tries to sent this data to a remote destination under the control of the attacker. The Policy depicted in listing 4.1 is formulated in pseudocode. A grammar for the Policy language is defined as part of the prototype implementation in Chapter 5.

```
1  // define a context for working hours
2  context.WorkingHours := Timestamp > "08:00" and Timestamp < "20:00";
3  // define an anomaly to capture excessive outgoing traffic within working hours
4  ano.HighTraffic := hint.HighTrafficWifi > "0.5" OR
5                     hint.HighTraffic3g > "0.5", context.WorkingHours;
6  hint.HighTrafficWifi := "smartphone.communication.ip.TxOther" <procedure a>;
7  hint.HighTraffic3g := "smartphone.communication.ip.Tx3g" <procedure a>;
8  // define some signatures
9  // is the camera currently used
10 sig.Camera := "smartphone.sensor.camera.IsUsed" = "true", context.WorkingHours;
11 // is there an app with suspicious permissions that can leak sensor data
12 sig.SuspiciousApp (scope=1) :=
13   "smartphone.android.app.permission.Requested" = "RECEIVE_BOOT_COMPLETED" AND
14   "smartphone.android.app.permission.Requested" = "CAMERA" AND
15   "smartphone.android.app.permission.Requested" = "INTERNET", context.WorkingHours;
16 // open port detected
17 sig.OpenPortDetected := "vulnerability.Name" = "Open Port", context.WorkingHours;
18 // define a condition that uses the signatures and anomalies
19 condition.SensorLeakage := ano.HighTraffic AND sig.Camera AND
20                            sig.SuspiciousApp AND sig.OpenPortDetected;
21 // define an enforcement action
22 action.DropClient :=
```

```
23   create "correlationresult.enforcement.EnforcementAction" = "./drop-client.sh";
24 // define a rule that puts all things together
25 if (condition.SensorLeakage) do action.DropClient;
```

Listing 4.1: Example Policy to detect one kind of sensory malware (pseudocode).

For this domain example, only violations that happen during working hours are assumed to be relevant. Thus, an appropriate Context is defined (line 2). Any other Anomaly and Signature definitions make use of this Context. Afterwards, an Anomaly is defined in order to detect excessive outgoing traffic originating from a smartphone (lines 4 to 5). The assumption here is that captured sensor data needs to be transmitted to a remote server, and thus will increase the outgoing traffic in an abnormal manner. The Anomaly is formulated by using two Hints (lines 6 to 7), each one referring to a specific Feature that encapsulates outgoing traffic. Note that each of the Hints makes use of the same Procedure for detecting the abnormal behavior. However, this is not a must. A concrete anomaly detection technique is not specified at this point. Chapter 5 will demonstrate how abnormal traffic can be detected by means of statistical methods. The Anomaly is defined in such a way that if one of the two Hints returns a score that is above 0.5, the behavior is considered to be abnormal.

Abnormal traffic is not the only indicator that is used to identify a sensory malware. In addition, a couple of Signatures are defined. They check if the camera of the smartphone is currently in use (line 10), if an app is installed that has a suspicious set of permissions (lines 12 to 15) and if the smartphone has an open port on which it accepts incoming connections (line 17). For this Signature, the scope is adjusted (set from 0 to 1) in order to match any app that has all of the permissions requested (as defined in Section 4.3.2).

In order to actually evaluate the Signature and the Anomaly, an appropriate rule is defined (line 25). Its Condition (line 19 to 20) makes use of the previously defined Signatures and the Anomaly. The associated Action that should be performed if the rule fires is defined in line 22 and 23. It causes the creation of a new Feature which is stored in the Feature Provider and retrieved by a Feature Consumer in order to drop the respective smartphone from the network. In this example, this is done by encapsulating a command that should be executed by the Feature Consumer (in this case a shell script).

4.5 Assessment

In the following, an assessment of the CADS approach is performed. The purpose of this assessment is twofold:

1. First, the CADS approach is compared to the requirements defined in Section 2.5. It is shown that CADS fulfills these requirements better than any other related approach.

2. Second, the inherent drawbacks of the CADS approach are discussed.

It is important to note that the assessment does not cover any benefits or drawbacks of CADS that are related to implementation details.

4.5.1 Fulfillment of Requirements

R-01 Detection of unwanted and malicious configurations of smartphones The CADS approach explicitly addresses this requirement by means of the Signature Components defined within the conceptual model (see Section 4.1.3). Signatures are patterns based on Features. When the Correlation Engine evaluates a Signature, the pattern is searched in the set of relevant Feature instances as described in Section 4.3.2. In order to detect a certain configuration (whether it is malicious or unwanted) with CADS, two aspects need to be met: (1) Features that are suitable to express the configuration must be available and (2) an appropriate Signature must be defined as part of the Correlation Engine's Policy. Since Features can be added transparently to the CADS approach, virtually any configuration can be detected. However, Feature Collectors must be available that collect the respective Features. Thus, the requirement is fulfilled.

R-02 Detection of abnormal smartphone behavior Abnormal behavior of smartphones can be detected by means of the CADS Anomaly Detection Components (defined in Section 4.1.4). The normal behavior can be flexibly defined by composing an Anomaly of several Hints, that in turn define which Features should be analyzed by which Procedure. The Correlation Engine supports to process Features both for training and testing. Training is performed based on the Correlation Engine's Policy in order to derive device-specific profiles. Again, Features need to be available that can be used for a reasonable

anomaly detection. Furthermore, an implementation of the CADS approach must provide concrete anomaly detection techniques. Thus, the requirement is fulfilled as well.

R-03 Consideration of context information for detection This requirement is addressed by the Context-related Components of the CADS conceptual model (see Section 4.1.2). In essence, Features are "tagged" with Context Parameters. Contexts can be specified based on these Context Parameters. This allows to (1) easily derive the Contexts a smartphone is currently in and (2) limit the set of Feature instances that are considered during the evaluation of the Correlation Engine's Policy based on the Context they were obtained in. The variety of Contexts is only limited by the set of available Context Parameters. CADS is the first approach that enables both context-related signature and anomaly detection for smartphones. However, as Context Parameters are set by Feature Collectors during the measurement of a Feature, some of them might not be reasonably set by all of the Feature Collectors (like GPS coordinates). Nevertheless, the requirement can generally be considered as fulfilled.

R-04 Policy-based reaction on detection results This requirement is primarily addressed by the Policy Components of the CADS conceptual model (see Section 4.1.5). It allows to use Signature and Anomaly components to specify simple IF-THEN Rules. For each Rule that fires, the associated Action is performed. This allows to flexibly react on any detected configuration or abnormal behavior. The design of the Action component allows to create, update or delete Features for the respective smartphone. This "feedback" provides two main benefits: (1) it allows to disseminate detection results and thus requests to employ an enforcement action to virtually any Feature Consumer, (2) it allows to define Signatures and Anomalies that work based on Features which are created by the Correlation Engine as part of performed Actions. The first benefit was used in order to address the scenario "Policy-based Enforcement" in the exemplary domain-specific mapping. The second benefit was not used yet. However, the general requirement to allow policy-based reactions is fulfilled.

R-05 Dynamic analysis at runtime This requirement is primarily addressed by the CADS architecture (Section 4.2) and the Correlation Model (Section 4.3). Feature Collectors collect Features at runtime and store them in the Feature Provider. The Correlation

Engine can retrieve the stored Features for each smartphone, evaluate the Policy and create new Features if necessary. These Features are again stored properly in the Feature Provider. Feature Consumers can retrieve those Features and react accordingly. Thus, the requirement is generally met. However, a training phase should be performed before the testing phase (and thus the dynamic analysis) starts. Furthermore, although the approach allows dynamic analysis on a conceptual level, the real latency between Feature measurement, Policy evaluation and the reaction that is employed by a Feature Consumer depends on the concrete implementation, especially on the communication protocol that is used.

R-06 Extensibility of processed data and used methods A major drawback of existing approaches was the lack of extensibility, both in terms of data that is processed and methods that are used for processing. In contrast, the CADS approach only names generic concepts both for defining the structure of data that is processed and the methods employed for anomaly detection. Data is expressed by means of Features, Categories and Context Parameters. Techniques for anomaly detection are encapsulated as Procedures. The only part of the Correlation Model that specifies a concrete algorithm for detection purposes is the evaluation of Signatures (Section 4.3.2). Thus, the requirement is fulfilled.

R-07 Ability to integrate the approach in existing environments This requirement is primarily addressed by the distributed CADS architecture. The logical roles define functionality that must be present in any IT infrastructure that aims to implement the CADS approach. The goal is to benefit from existing services by adding additional functionality to them so that they can fulfill the role of a Feature Collector or a Feature Consumer. Since CADS is a network-oriented approach and thus does not rely on extensive changes to the smartphones themselves, this requirement is generally fulfilled on the conceptual level. However, it depends on the concrete implementation how easy an integration into a real IT infrastructure really is. Even more, a concrete implementation can cause the CADS approach to fail this requirement. For example, if Features should be collected on smartphones that require customized versions of the smartphone platform. In this case, it is a domain-specific design decision to include such Features while sacrificing an easy integration. Thus, the requirement is fulfilled as well on the conceptual level.

4.5.2 Drawbacks

Despite the fact that the presented CADS approach fulfills all necessary requirements at the conceptual level, there are also some inherent drawbacks that should be briefly mentioned here.

Complexity of architecture integration The CADS architecture is composed of four logical roles. Two of them are expected to be fulfilled by new systems that are added to an IT infrastructure (Feature Provider and Correlation Engine). Feature Collectors and Consumers on the other hand should be realized by extending existing systems and services. Depending on the target IT infrastructure, the integration of these roles can become a complex task.

Necessity of domain knowledge In order to detect unwanted configurations and abnormal behavior with the CADS approach, knowledge about the domain CADS is used in is required. In this respect, the term knowledge refers to the fact that the domain-specific mapping as described in Section 4.4 basically requires to specify two aspects:

1. What data is relevant and should be collected and processed. This is done by defining Features and Categories.

2. How should the collected data be processed. This is done by defining a Policy with Rules whose Conditions are formulated based on Signatures and Anomalies.

The capability of CADS to enable a secure integration of smartphones into existing IT infrastructures relies on the definition of an adequate Policy that renders the domain knowledge. At the moment, there is no way to automatically generate a Policy.

Further drawbacks and limitations which are related to implementation details are discussed in Chapter 5.

4.6 Summary

This chapter presented a novel, network-oriented approach for smartphone security. The approach is referred to as **CADS: Context-related Signature and Anomaly Detection for Smartphones**. It is composed of four main parts.

1. The CADS conceptual model was presented in Section 4.1. It defines the main building blocks and the relationships between them, especially the notion of Features and Categories and how they are used to express Signatures and Anomalies.

2. The CADS architecture was detailed in Section 4.2. It is mainly composed of logical roles and their responsibilities. Components that fulfill these logical roles need to be present in an IT infrastructure in order to implement the CADS approach.

3. The CADS correlation model that defines the general processing of collected data was presented in Section 4.3. This part primarily defines the internal mechanisms and algorithms that are used by the Correlation Engine to evaluate Signatures and Anomalies.

4. Finally, a process model that defines how the first three parts of the CADS approach can be mapped to a specific problem domain was presented in Section 4.4.

An assessment of the CADS approach was presented in Section 4.5. The results show that CADS fulfills the requirements that were identified in Chapter 2 at a conceptual level. The feasibility of the CADS approach will be demonstrated by presenting a prototype implementation in Chapter 5, followed by an evaluation in Chapter 6.

5 Implementation

"Beware of bugs in the above code; I have only proved it correct, not tried it."

(Donald Ervin Knuth)

Contents

In the following, a proof of concept implementation of the CADS approach is presented. The prototype uses IF-MAP (introduced in Section 3.4) as communication protocol. First, Section 5.1 details how the generic CADS approach presented in Chapter 4 can be realized based on IF-MAP. Afterwards, Section 5.2 describes the software components that were developed for the prototype implementation. In the end, Section 5.3 mentions existing drawbacks and limitations of the prototype.

5.1 IF-MAP as Communication Protocol

The IF-MAP protocol for network security was chosen as communication protocol for the CADS prototype implementation. It meets all requirements that were identified in Section 4.2.2. Furthermore, its original purpose is to enhance network security by enabling existing systems and services to share data at runtime. Thus, it matches the general idea of the CADS approach which aims to improve smartphone security in enterprise environments by leveraging existing systems and services.

Section 5.1.1 details why IF-MAP meets the requirements demanded by a communication protocol that is used for realizing the CADS approach. Section 5.1.2 explains how Features are mapped to the IF-MAP data model. Section 5.1.3 elaborates how the CADS logical roles are mapped to those defined as part of the IF-MAP specification. For the prototype implementation, version 2.0 of the IF-MAP protocol is used [163]. An update of the core protocol to version 2.1 [20] published in May 2012 introduced some improvements that are considered in Section 5.1.5.

5.1.1 Fulfillment of Requirements

As already stated, the IF-MAP protocol fulfills all requirements that were mentioned in Section 4.2.2. This is detailed in the following.

Request-Response Interaction IF-MAP uses SOAP via HTTPS and thus follows the request-response paradigm.

Proper Transmission of Features IF-MAP is based on XML [164]. Thus, it allows proper encapsulation of Features as long as they are represented as XML documents. In

order to distinguish Features for numerous smartphones, IF-MAP identifiers can be used (as detailed in Section 5.1.2).

Support for Remote Procedure Calls IF-MAP is based on remote procedure calls. Its publish-search-subscribe operations can be used to store, retrieve or delete Features.

Secure As the use of HTTPS is mandatory for any IF-MAP communication, the protocol is sufficiently secure regarding the integrity and confidentiality of Features in transit. However, the level of security varies depending on which method is used to authenticate MAP clients to the MAP server (HTTP basic authentication versus certificate-based authentication).

Efficient and Scalable The protocol is intended to be used in large IT infrastructures to enable communication of thousands of network devices, with thousands of messages communicated per second. Thus, it is designed to be able to scale in such environments. However, the actual performance depends on the concrete implementation of the protocol.

It is important to note that other protocols and techniques could have been used to build the prototype of CADS as well. This includes complex event processing Engines like Esper[1] or message oriented middlewares based on the Advanced Message Queuing Protocol (AMQP)[2].

5.1.2 Encapsulation of Features within IF-MAP

In order to transmit Features via the IF-MAP protocol, they need to be properly mapped to the IF-MAP data model. That is, the core components Category and Feature as well Context Parameters must be expressed as IF-MAP data types. As the other components of the CADS conceptual model are only used locally by the Correlation Engine, they do not need to be expressed by means of IF-MAP. As explained in Section 3.4, the IF-MAP data model is composed of identifiers, links and metadata[3] objects that are organized as a graph structure. The structure of identifiers and metadata is specified via XML Schema documents [165].

[1] http://esper.codehaus.org/
[2] http://www.amqp.org/
[3] The meaning of the term metadata with respect to IF-MAP was explained in Section 3.4.

General Extensibility of IF-MAP

IF-MAP provides several mechanisms to extend and customize both the base protocol and the metadata types that are transmitted. However, changes to the base protocol should be omitted by third parties in order to ensure interoperability among arbitrary IF-MAP implementations. Basically, there are two ways that can be used to map the components of the CADS conceptual model to IF-MAP:

1. `identity` identifiers of type "other". IF-MAP version 2.0 supports five distinct types of identifiers. One of them is the `identity` identifier that is primarily used to represent users who are connected to a network. However, its XML Schema definition (depicted in Listing 5.1) includes two attributes that can be leveraged to encapsulate arbitrary data within such an identifier: `type` and `other-type-definition`. The `type` attribute must be set to "other". The `other-type-definition` attribute must be set to a non-empty string that unambiguously identifies the type of the `identity` identifier. Its value must take one of two forms:

   a) "Name": In case the type is defined by the TCG, it is identified by specifying an appropriate name.

   b) "Vendor-ID:Name": In case the type is defined by another, third party (referred to as vendor), the name of the type is prefixed by a so-called Vendor-ID. It is a Structure of Management Information (SMI) Private Enterprise Number [166, 167] owned by the respective vendor that defines the type.

   These two attributes provide the semantics to interpret the value of the `name` attribute of the identifier.

```
<xsd:complexType name="IdentityType">
  <xsd:attribute name="administrative-domain" type="xsd:string"/>
  <xsd:attribute name="name" type="xsd:string" use="required"/>
  <xsd:attribute name="type" use="required">
  <xsd:simpleType>
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="aik-name"/>
    <xsd:enumeration value="distinguished-name"/>
    <xsd:enumeration value="dns-name"/>
    <xsd:enumeration value="email-address"/>
    <xsd:enumeration value="kerberos-principal"/>
    <xsd:enumeration value="username"/>
    <xsd:enumeration value="sip-uri"/>
    <xsd:enumeration value="tel-uri"/>
```

```
15      <xsd:enumeration value="hip-hit"/>
16      <xsd:enumeration value="other"/>
17    </xsd:restriction>
18    </xsd:simpleType>
19  </xsd:attribute>
20  <xsd:attribute name="other-type-definition" type="xsd:string"/>
21  </xsd:complexType>
```

Listing 5.1: IF-MAP XML Schema for `identity` identifier [163].

2. Vendor-specific Metadata. New types of metadata can be added by specifying appropriate complex types in XML Schema documents. There are only two requirements that must be met: (1) the new complex types must be specified within an XML namespace which is different from the default namespace used by the TCG and (2) each newly defined complex type must reference one of two attribute groups that are defined by the TCG (either `singleValueMetadataAttributes` or `multiValueMetadataAttributes`).

A third way that was considered in the first place was to extend existing metadata definitions by adding new attributes to an attribute group that is defined as part of the IF-MAP base protocol. The XML Schema that defines the respective attribute group is depicted in Listing 5.2.

```
1 <xsd:attributeGroup name="metadataAttributes">
2 <xsd:attribute name="ifmap-publisher-id"/>
3 <xsd:attribute name="ifmap-timestamp" type="xsd:dateTime"/>
4 <xsd:anyAttribute/>
5 </xsd:attributeGroup>
```

Listing 5.2: IF-MAP metadata attribute group [163].

The element `<xsd:anyAttribute/>` allows to add any number of additional attributes to the group. However, the specification explicitly denies to use this element for adding vendor-specific attributes [20] p.48. Thus, adding attributes to standard metadata types is not an option to map Features, Categories or Context Parameters to the IF-MAP data model. Instead, they must be mapped by either using `identity` identifiers of type “other” or by defining new, vendor-specific metadata.

The mapping approach that is presented in the following uses both of these two alternatives. Features are mapped to new metadata types defined by an XML Schema document.

Context Parameters are modeled as a new attribute group which is referenced by the complex type of the Feature. Categories however are modeled by leveraging the previously introduced `identity` identifier. Any newly defined metadata types use the namespaces depicted in Listing 5.3.

```
<?xml version="1.0" ?>
<xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ifmap="http://www.trustedcomputinggroup.org/2010/IFMAP/2"
xmlns="http://www.esukom.de/2012/ifmap-metadata/1"
targetNamespace="http://www.esukom.de/2012/ifmap-metadata/1">
```

Listing 5.3: Target namespace declaration for CADS components.

Mapping of Context Parameters

Context Parameters are mapped to a new XML Schema attribute group (depicted in Listing 5.4). In the example, only the three Context Parameters that were introduced in Section 4.4 are included. However, further attributes can be added if necessary due to the `xsd:anyAttribute` element.

```
<xsd:attributeGroup name="contextParameters">
  <xsd:attribute name="ctxp-timestamp" type="xsd:dateTime"/>
  <xsd:attribute name="ctxp-longitude" type="xsd:string"/>
  <xsd:attribute name="ctxp-latitude" type="xsd:string"/>
  <xsd:anyAttribute namespace="##targetNamespace"/>
</xsd:attributeGroup>
```

Listing 5.4: Mapping of Context Parameters to XML attributes.

Mapping of Features

The complex type that is used to encapsulate a Feature is depicted in Listing 5.5. It defines three elements to render the Features id, its type and its value. Furthermore, it makes use of two attribute groups. The first one is used to include Context Parameters as defined in the previous section. The second one is used to include further attributes that are necessary for any IF-MAP metadata type. Note that the description of a Feature is not mapped to the XML complex type. The description provides semantic background regarding the Feature for human beings and is defined as part of the domain-specific

instance. However, it is not intended to be processed by any logical role of the CADS approach at runtime. Thus, it is omitted in the complex type definition.

```
<xsd:element name="feature">
<xsd:complexType>
<xsd:sequence>
  <xsd:element name="id" type="xsd:string"/>
  <xsd:element name="type">
    <xsd:simpleType>
    <xsd:restriction base="xsd:string">
    <xsd:enumeration value="quantitive"/>
    <xsd:enumeration value="qualified"/>
    <xsd:enumeration value="arbitrary"/>
    </xsd:restriction>
    </xsd:simpleType>
  </xsd:element>
  <xsd:element name="value" type="xsd:string"/>
</xsd:sequence>
<xsd:attributeGroup ref="contextParameters"/>
<xsd:attributeGroup ref="ifmap:multiValueMetadataAttributes"/>
</xsd:complexType>
</xsd:element>
```

Listing 5.5: Metadata to represent a Feature.

Basically, this complex type definition is already sufficient to use Features within IF-MAP. That is, Features could be published by a MAP client to the MAP server, either attaching it to an identifier or on a link between two identifiers. However, one part that is still missing is the mapping of the hierarchical structure defined by means of Categories.

Mapping of Categories

In the CADS conceptual model, Categories are used to hierarchically structure the set of defined Features. That is, they do not actually contain values that have been measured. Instead, they provide the semantic background for Features. The relationship between IF-MAP identifiers and metadata is similar: identifiers are the structuring, rather constant components, whereas metadata components are used to render data that was obtained at runtime and that frequently changes.

Thus, it is reasonable to map Categories to IF-MAP identifiers. More precisely, an `identity` identifier of type “other” is used. The attribute `other-type-definition` is set to the value “32939:category”. The number “32939” is the SMI Private Enterprise Number of the Hochschule Hannover.

It is important to note that two requirements must be considered by this mapping. First, Categories can have a cardinality of either 1 or N. That is, any Feature that is contained by a Category whose cardinality is N can have N instances at the same time. These instances must be distinguishable within IF-MAP. Second, Features will be collected for various smartphones simultaneously. That is, Feature instances must also be distinguishable according to the smartphone they belong to.

In order to address both requirements, the attributes of an `identity` identifier that represents a Category are set as follows:

- The attribute `name` is set to the fully qualified identifier of the Category instance (as defined in Section 4.1.1).
- The attribute `administrative-domain` is set to a value that unambiguously identifies the smartphone to which the respective Features and Categories belong. It can either be chosen randomly or derived from information that is suitable to identify a smartphone like its International Mobile Station Equipment Identity (IMEI).

An example of an `identity` identifier that represents a Category instance of the CADS conceptual model is depicted in Listing 5.6. The identifier generally refers to the category *smartphone.android.app.permission* as defined in Section 4.4. As both *app* and *permission* are of cardinality N, appropriate instance counters are attached to them. The counters are separated by a colon from the name of the respective Category. Thus, the example refers to the Category instance that represents the 6th permission of the 4th app on the smartphone which is identified by the string "f498ad8bdcc209dd".

```
<identity type="other"
  other-type-definition="32939:category"
  <!-- unique id of smartphone -->
  administrative-domain="f498ad8bdcc209dd"
  <!--  fully qualified identifier of category -->
  name="smartphone.android.app:3.permission:5" />
```

Listing 5.6: Mapping of Categories to IF-MAP identifiers.

Thus, Features that are expressed as IF-MAP metadata (leveraging the complex type definition from above) can be attached to the `identity` identifier that represents their respective Category. However, the tree structure that is defined by the hierarchy of Categories cannot be maintained yet within IF-MAP. Although the hierarchy is encapsulated

as part of the `name` attribute, there is no way to associate two `identity` identifiers in order to express their sub-category relationship. This makes traversing the IF-MAP graph hard.

In order to address this issue, another new metadata type is introduced. Its only purpose is to express the sub-category relationship between two Categories within IF-MAP. That is, the metadata is published on a link between two `identity` identifiers that represent a Category. Its complex type definition is depicted in Listing 5.7.

```
<xsd:element name="subcategory-of">
<xsd:complexType>
<xsd:attributeGroup ref="ifmap:singleValueMetadataAttributes"/>
</xsd:complexType>
</xsd:element>
```

Listing 5.7: Metadata to model sub-category relationship.

Distinguishing Features from Multiple Smartphones in IF-MAP

As already stated, numerous Features will be collected from multiple smartphones at the same time by multiple Feature Collectors. In addition to be able to distinguish multiple instances of the same Features, collected Features need also be distinguishable on a per smartphone basis.

This is achieved by leveraging the standard `device` identifier of the IF-MAP data model. For each smartphone that is used in an IT infrastructure, at least one such `device` identifier is expected to be present. In order to associate this `device` identifier with the `identity` identifiers for the Categories (and thus with the Feature metadata that is attached to them), another new metadata type is necessary. Its structure is defined in Listing 5.8. It is intended to be placed on a link between the smartphone's `device` identifier and any identifier that represents a top-level Category. In the exemplary domain-specific mapping presented in Section 4.4, there are four such Categories (*vulnerability, ids, smartphone, correlationresult*).

```
<xsd:element name="device-category">
<xsd:complexType>
<xsd:attributeGroup ref="ifmap:singleValueMetadataAttributes"/>
</xsd:complexType>
</xsd:element>
```

Listing 5.8: Metadata to associate a `device` identifier with a Category identifier.

Example for an IF-MAP Graph

Figure 5.1 depicts a sample IF-MAP graph that was created based on the mapping described above. Standard identifiers and metadata types that are defined by the IF-MAP specifications are depicted in yellow tones. Although the concrete structure depends on the actual MAP clients that are used within an infrastructure, the Figure depicts an example that is common.

Standard identifiers and standard metadata are primarily used to express which endpoints are currently connected to a network. In the example, two smartphones have established a connection to the network of the IT infrastructure. Each smartphone is represented by a `device` identifier ("smartphone-23b3" and "smartphone-af0f"). The connections that they have established are represented by `access-request` identifiers. The relationship between a smartphone's `device` identifier and its `access-request` identifier is expressed by means of `access-request-device` metadata that is placed on the link between them. The graph further depicts that both connections have been authenticated by another device referred to as "pdp". This device fulfills the role of a Policy Decision Point (PDP) as defined by the IF-MAP specification (as explained in Section 3.4). A single PDP is responsible for handling connection requests of multiple endpoints. Normally, the PDP authenticates the user and optionally the device before access to the protected network is granted. In the following, the IF-MAP graph for the smartphone "smartphone-23b3" is detailed. The subgraph for the second (or any other) smartphone looks similar.

The standard metadata is used to render basic information about the connected smartphone. The example graph depicts which IP address is currently in use, the smartphone's MAC address and the user that was authenticated while the connection was established (in this case named "bob"). The set of standard metadata discussed so far is usually published by the PDP himself.

Identifiers that are used to represent Categories and newly defined metadata types are depicted in blue tones. All Features for a single smartphone are contained within one single subtree of the IF-MAP graph. The root node of this subtree is defined by the smartphone's `device` identifier. Any `identity` identifier that represents a top-level Category is associated to the smartphone's `device` identifier by a link that has `device-category` metadata attached to it. Going further, any `identity` identifier that represents a sub Category is

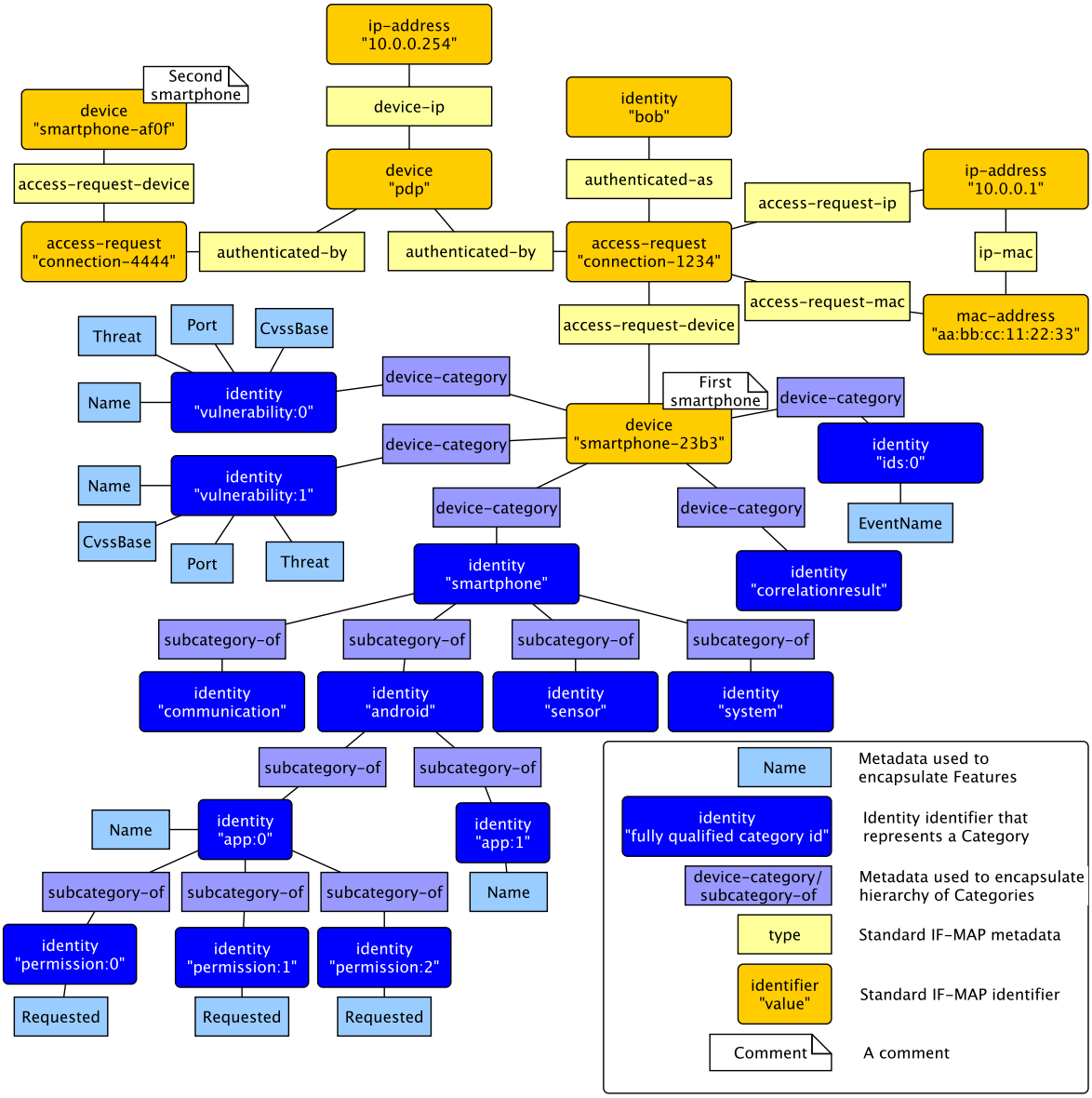

Figure 5.1: IF-MAP graph with CADS specific metadata and identifiers. Note that the value of IF-MAP identifiers that represent Categories is set to the fully qualified category ID. However, only the short form of the Category IDs is depicted in the sample graph in order to ensure readability.

associated to its parent Category by a link that has `subcategory-of` metadata attached. Feature metadata can be published to any `identity` identifier that represents a Category.

It must be ensured that the attribute `administrative-domain` is set properly for each `identity` identifier that is used to represent a Category. More precisely, its value must be set to the same value that is used by the `device` identifier of the respective smartphone. Referring to the example graph, each blue identifier has set its `administrative-domain` attribute to the value "smartphone-23b3".

Note that the example only depicts a small extract of an IF-MAP graph in order to emphasize how the CADS components are encapsulated within IF-MAP. For example, it only depicts very few Features for two apps on the respective smartphone. Features that might have been created by the Correlation Engine are omitted as well.

5.1.3 Mapping the CADS Architecture to IF-MAP

So far, the necessary parts of the CADS conceptual model have been mapped to the IF-MAP data model. In the next step, the logical roles of CADS are mapped to those defined in the IF-MAP specification. That is, it is defined if a logical role of the CADS architecture is either realized by implementing a MAP client (MAPC) or a MAP server (MAPS).

For most of the logical roles, this mapping is straightforward. Feature Collectors and Feature Consumers are mapped to the MAP client role, whereas the Feature Provider is mapped to the MAP server role. A MAP server provides all the functionality that is required by a Feature Provider, which is to store, retrieve and delete Features. Storage and deletion of Features can be performed by leveraging the IF-MAP publish operation. Retrieval of Features can be performed by using IF-MAP search, subscribe and poll operations. Feature Collectors that act as MAP clients can marshal measured Features according to the IF-MAP data model (as described above) and use the IF-MAP publish operation to transmit them to the MAP server. Feature Consumers on the other hand can leverage IF-MAP search, subscribe and poll operations to retrieve measured Features from the MAP server, unmarshal them and process them.

To answer the question how the Correlation Engine should be mapped to the IF-MAP roles is not that straightforward. There are basically two options: (1) to implement it as an extension within the MAP server and (2) to implement it as another MAP client. Both

of these alternatives have their own benefits and drawbacks. Realizing the Correlation Engine as an extension to the MAP server might sound as the most reasonable approach. As the MAP server handles all measured Features of all smartphones, a module that implements the Correlation Engine could easily have access to them as well. There would be no overhead in terms of network communication between the Correlation Engine and the MAP server. The Correlation Engine could work based on the internal data structures that are used by the MAP server to represent measured Features.

However, extending a MAP server by a Correlation Engine module also introduces some drawbacks. First, the implementation would be specific to a single MAP server. That is, the module that realizes the Correlation Engine would not be able to work with arbitrary MAP servers (despite the fact that IF-MAP was chosen as protocol to ensure interoperability). Second, the Correlation Engine is required to create, change or delete new/existing Features when Rules within their Policy fire. If it is realized as module, this would require to "spawn" metadata within the MAP server. That is, metadata is changed within the MAP server without the involvement of a MAP client. Although technically feasible, it violates the operational model of the IF-MAP protocol.

In order to ensure compliance with the IF-MAP protocol and interoperability among the implemented software components, the Correlation Engine is realized as another MAP client. The drawbacks that are introduced by this approach are taken into account: namely the processing overhead for marshalling and unmarshalling of Features and the latency introduced by transmitting the marshalled Features over the network between the MAP server and the Correlation Engine acting as MAP client. Figure 5.2 depicts the CADS architecture with the adoption of IF-MAP roles.

5.1.4 Coordination of Multiple MAP Clients

Organizing Features for a smartphone under a single `device` identifier introduces some challenges regarding the coordination of multiple MAP clients. When MAP clients that act as Feature Collectors are distributed in the respective IT infrastructure are involved, it cannot generally be assumed that all of them have access to information that is suitable for unambiguously identifying a smartphone. For example, the IMEI can be used by a Feature Collector that runs on the smartphone itself. However, it cannot be used if the Feature Collector resides on a service in the IT infrastructure (like an IDS or a vulnerability

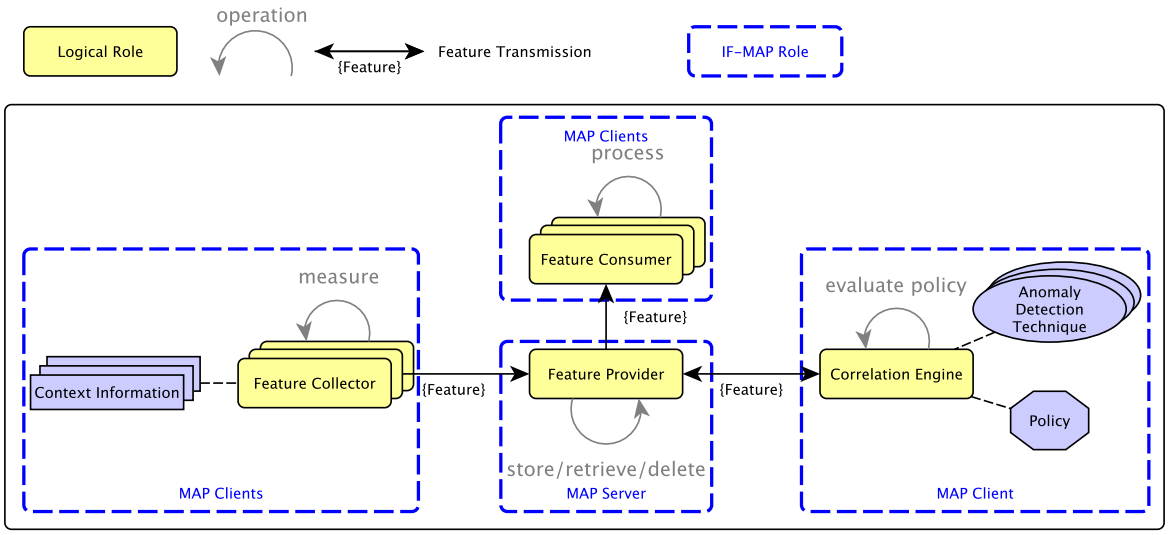

Figure 5.2: CADS architecture with IF-MAP roles.

scanner). That introduces the question how arbitrary Feature Collectors can manage to publish Features under a single `device` identifier.

This problem can be tackled by leveraging the inherent capabilities of the IF-MAP protocol. When an endpoint (whether it is a smartphone or not) connects to the network, it is assumed that a Policy Decision Point (PDP) publishes a minimum set of standard metadata to the MAP server [154]. This includes

- `access-request-device` metadata to associate the `access-request` with the endpoint's `device` identifier,
- `authenticated-by` metadata to associate the `access-request` identifier with the `device` identifier that represents the PDP,
- `access-request-mac` and `access-request-ip` metadata to associate the `access-request` with the endpoint's `mac-address` and `ip-address` identifiers if available,
- optionally, `authenticated-as` metadata in order to associate the `access-request` with the `identity` identifier that represents the authenticated user of the endpoint.

Thus, a PDP is responsible for creating an appropriate `device` identifier for each connected smartphone. As already stated, the value of the `device` identifier may be chosen randomly

or derived from information that identifies a smartphone (if such information is available to the PDP). In any case, other Feature Collectors can search the IF-MAP graph in order to find the matching `device` identifier for a smartphone. The only requirement is that the Feature Collector must provide an IF-MAP identifier that represents the root of the search query. For example, a Feature Collector that extends an IDS or a vulnerability scanner can use the IP address of the respective device. As the `ip-address` identifier is associated to the `access-request` identifier, which in turn is associated to the smartphone's `device` identifier, the Feature metadata can be published appropriately.

However, there may also be Feature Collectors that have no means to specify a reasonable IF-MAP identifier as root for a search query based on the data they process. Consider a crawler of an app store that aims to publish Features such as number of downloads and rating for apps that are installed on a smartphone. As already stated in Section 4.4.2, such a Feature Collector can be realized as a service that is deployed within the target IT infrastructure. In this case, it cannot be assumed that the service knows an IP address, MAC address or other kind of information based on the data it processes that is suitable to specify an appropriate IF-MAP identifier for a search query. The only solution is that such Feature Collectors need additional configuration parameters that explicitly name IF-MAP identifiers that can be used for search queries in order to find the respective smartphone's `device` identifier. For the prototype implementation, a `device` identifier that represents the PDP will be used for this purpose. That is, any Feature Collector is at least capable of searching the IF-MAP graph with the PDP's `device` identifier as root.

The same issue is also relevant for MAP clients that fulfill the role of a Feature Consumer or the Correlation Engine. As with Feature Collectors, configuration parameters ensure that those MAP clients can at least search the IF-MAP graph based on the `device` identifier of the PDP. However, a Feature Consumer that resides on a OSI layer 3 packet filter might as well search the IF-MAP graph based on IP addresses he has seen traffic for. Coordination of MAP clients is a general challenge that is inherent to the IF-MAP protocol. However, it is beyond the scope of the respective specification to define concrete mechanisms for it.

5.1.5 Further Improvements based on IF-MAP Version 2.1

The most recent version 2.1 of the IF-MAP protocol was published in May 2012 [20]. It introduced the concept of *extended identifiers*. Previous versions were limited to five types of identifiers. However, it turned out that numerous use cases exist that would benefit from customizable identifier types. That is, a mechanism to add new types of identifiers similar to the way new metadata types can be added was desired.

Extended identifiers provide the desired flexibility. Basically, new types of identifiers can be defined by XML Schema documents. The only requirement that must be met is that new identifiers extend the `IdentifierType` complex type depicted in Listing 5.9. Otherwise, they can contain virtually any XML data.

```
<xsd:complexType name="IdentifierType">
<xsd:attribute name="administrative-domain" type="xsd:string" use="required"/>
</xsd:complexType>
```

Listing 5.9: Complex type for extended identifiers in IF-MAP 2.1 [20].

Extended identifiers can be used to implement an alternative mapping of Categories. The structure of an extended identifier that represents a Category is depicted in Listing 5.10.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:base-id="http://www.trustedcomputinggroup.org/2012/IFMAP- IDENTIFIER/1"
  xmlns="http://www.esukom.de/extended-identifiers"
  targetNamespace="http://www.esukom.de/extended-identifiers">
<xsd:element name="category">
<xsd:complexType>
  <xsd:complexContent>
    <xsd:extension base="base-id:IdentifierType">
    <xsd:attribute name="id" type="xsd:string" use="required"/>
  </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
</xsd:element>
```

Listing 5.10: Extended identifier for Categories.

It basically adds an attribute `id` to the base type which should be set to the fully qualified identifier of the respective Category instance. An example identifier that represents the same Category instance as in Listing 5.6 is depicted in Listing 5.11.

```
<category
  <!-- unique id of smartphone -->
```

```
3   administrative-domain="f498ad8bdcc209dd"
4   <!--  fully qualified identifier of category -->
5   id="smartphone.android.app:3.permission:5" />
```

Listing 5.11: Mapping of Categories to IF-MAP identifiers.

Thus, the use of extended identifiers seems to be a reasonable alternative compared to the use of `identity` identifiers as described in Section 5.1.2. There should be less overhead as the attributes specific to `identity` identifiers can be omitted (`type` and `other-type-definition`). Furthermore, the element name `<category>` matches the purpose of the identifier more appropriately. However, these benefits are not existent given the way how the IF-MAP specification version 2.1 encapsulates extended identifiers in IF-MAP request and response messages. In order to ensure backwards compatibility with existing implementations, they demand that extended identifiers are again encapsulated by `identity` identifiers whose `type` is set to "other". The `other-type-definition` attribute is set to the string "extended". Finally, the `name` attribute of the respective `identity` identifier is set to a canonicalized form of its XML data. The details of how the canonicalized form is generated are given in the IF-MAP specification [20] Section 3.2.3.3.

Thus, the use of extended identifiers as defined by version 2.1 of the IF-MAP specification would not provide any major benefits. In fact, it would make the creation and processing of `identity` identifiers that represent Categories more complex and cumbersome. Once the IF-MAP protocol allows to use extended identifiers without the need to encapsulate them as `identity` identifiers, the CADS prototype implementation should be changed accordingly.

5.1.6 Alternative Mapping Approaches

This section presented an approach to use IF-MAP as communication protocol for the CADS prototype implementation. Categories, Features and Context Parameters have been mapped to the IF-MAP data model. Furthermore, the logical roles of the CADS architecture have been mapped to those defined within the IF-MAP specification. Several design decisions had to be made, such as whether the Correlation Engine should be realized as an extension to the MAP server or as dedicated MAP client. The benefits and drawbacks of the two alternatives were discussed.

Even more alternatives exist considering the mapping of Categories, Features and Context Parameters to the IF-MAP data model. They have not been discussed in detail. The taken approach enables to update the values of single Feature instances at a minimum cost in terms of traffic on the network and processing power required by the components that implement the logical roles. An alternative that quickly comes to mind is to encapsulate all Features, Categories and Context Parameters within one single complex type. The corresponding metadata could be published directly to the `device` identifier of a smartphone. This approach does not require to render the hierarchy of Categories by means of IF-MAP identifiers. Instead, it is encapsulated within the complex type definition of the new metadata type. However, the main drawback of this approach becomes obvious when the value of a single Feature instance needs to be updated. As the value is encapsulated as part of a larger XML document instance, any other Features of the same XML document need to be republished as well (even if their values have not changed). That is, although the structure of the IF-MAP graph would be less complex (considering the number of identifiers and links that are used), updating values of single Features would be too cumbersome. It is expected that updates of single Feature values will occur frequently (for example when the status of a smartphone's sensors change).

The coordination of multiple MAP clients was identified as a general challenge. It is inherent to the IF-MAP protocol. The prototype implementation of the CADS approach tackles this challenge by means of configuration parameters that can be used to specify IF-MAP identifiers that are used as root nodes for search queries. The use of extended identifiers as defined by version 2.1 of the IF-MAP specification does not provide any major benefits compared to IF-MAP version 2.0.

5.2 Software Components

This section gives an overview of the software components that were developed in order to realize the CADS prototype based on the IF-MAP protocol for network security.

5.2.1 Background

Most of the software components that are used for the prototype implementation have been developed by the Trust@FHH research group. Some components were exclusively

developed for this thesis (such as the Correlation Engine "irondetect"). Others were developed as part of the group's overall effort to study the specifications that are published by the TCG (such as the MAP server "irond"). In order to ensure platform interoperability, the Java programming language was chosen for development. Any component that is presented below is available as open source software licensed under the Apache License version 2.0 [49]. They can be downloaded from the Trust@FHH website [168], the Trust@FHH GitHub account [169] or the website of the ESUKOM research project [21].

It should be noted that the CADS approach can be used in any IT infrastructure where the software components that are described in the following are deployed.

5.2.2 MAP Server irond as Feature Provider

The MAP server "irond" is used to fulfill the role of the Feature Provider. irond is an experimental MAP server that supports IF-MAP version 2.0 and 2.1. Its development began in 2009 as part of a project of bachelor students at the University of Applied Sciences and Arts in Hannover. irond is now updated and maintained by the Trust@FHH research group. It is one of the first MAP servers that were officially certified by the TCG [170] for their specification compliance and interoperability among implementations of different vendors.

irond is written in Java. One design goal was to rely on the Java Standard Edition (JavaSE) and to omit the use of external libraries if possible. Thus, only three dependencies to external libraries exist:

- Apache Commons Codec[4] primarily used for Base64 encoding and decoding,
- Apache HttpCore[5] used to implement basic HTTP server functionality and
- Apache log4j[6] for logging.

Further details regarding the design and implementation of irond are provided as part of the archives that are available for download. It is important to note that any other MAP server implementation that supports IF-MAP 2.0 could be used as Feature Provider as well. That is, implementations from different vendors can be used in order to seamlessly

[4] http://commons.apache.org/codec/
[5] http://hc.apache.org/httpcomponents-core-ga/
[6] http://logging.apache.org/log4j/1.2/

replace irond. This especially includes omapd[7], another open source MAP server, as well as commercial products from Juniper Networks[8] and Infoblox[9]. The other software components of the CADS prototype will not be affected by this and thus do not need to be changed in any way.

5.2.3 MAP Client Library ifmapj

In order to ease the development of MAP clients, a general purpose library called "ifmapj" was implemented. It is basically a port of the C++ library libifmap2c[10]. It provides a simple interface to communicate with IF-MAP servers. Thus, clients that are implemented based on ifmapj do not need to handle the details of the IF-MAP protocol themselves. In fact, in order to publish Features according to the mapping defined in Section 5.1.2, developers only need to create appropriate Document Object Model (DOM) [171] XML documents within their application which can then be passed as parameters to the respective methods provided by the ifmapj library.

ifmapj aims to provide a lightweight, yet flexible way to develop MAP clients. Similar to irond, its design aims to rely only on a minimum set of external libraries. More precisely, only Apache HttpCore is used in order to ease the handling of HTTP requests and responses. ifmapj can be used on any platform that supports a Java runtime environment. This also includes the Google Android smartphone platform. ifmapj is used for any MAP client that is implemented as part of the CADS prototype.

5.2.4 Correlation Engine

The main contribution of the prototype implementation is the development of the Correlation Engine "irondetect". This section provides an overview of its functionality.

IF-MAP Communication

The Correlation Engine uses IF-MAP subscribe and poll operations in order to get notified when Feature metadata is changed in the MAP server. The challenge here is that all

[7] http://code.google.com/p/omapd/
[8] http://www.juniper.net/
[9] http://www.infoblox.com/en/products/orchestration-server-if-map.html
[10] http://code.google.com/p/libifmap2c/

Feature metadata from all smartphones that are connected to the respective network must be obtained. For this purpose, irondetect generally holds two types of subscriptions. The first one is used to get notified in the event that a new smartphone joins the network. In this case, it is assumed that a PDP publishes a set of standard metadata as mentioned in Section 5.1.4. This especially includes `access-request-device` metadata that associate the `access-request` identifier of the smartphone with its `device` identifier. The essential parts of the subscription request that is used for this purpose and its parameters are depicted in Listing 5.12.

```
<subscribe>
<update name='subscription-pdp'
    max-depth='2'
    match-links='authenticated-by or access-request-device'>
<device>
  <name>pdp</name>
</device>
</update>
</subscribe>
```

Listing 5.12: Subscription that allows irondetect to notice smartphones that join the network. Namespace declarations and operational attributes are omitted for readability.

The root identifier for the subscription is specified by the `<device>` element. It represents the `device` identifier of the PDP who is named "pdp". As outlined in Section 5.1.4, the actual name of the PDP is a configuration parameter that needs to be provided to the Correlation Engine. Further parameters are specified as attributes to the `<update>` element. Their meaning is as follows:

- `name`: A name for the subscription. This is used internally by the MAP server in order to distinguish multiple subscriptions. It must be chosen uniquely among all subscriptions of a single MAP client.

- `max-depth`: The maximum depth of the subscription is set to 2. That is, starting from the `device` identifier of the PDP, only the subgraph that is composed of links and identifiers that have a distance of at most 2 are covered by the subscription. The distance is measured by the number of links that are traversed. That is, a max-depth of 0 only covers the root identifier. A max-depth of 1 additionally covers all identifiers that are associated to the root identifier by one single link. Thus, a max-depth of two

covers identifiers that can be reached from the root identifier by traversing at most two links. Given the set of standard metadata as depicted in Figure 5.1, the distance between a PDP's `device` identifier and the `device` identifiers of smartphones that joined the network is 2.

- `match-links`: In order to further constrain which links (and thus which identifiers) are covered by a subscription, the `match-links` attribute is used. For example, in order to get notified about new smartphones that join the network, it is only necessary to consider links that have either the metadata `authenticated-by` or `access-request-device` attached. Starting from the `device` identifier of the PDP, following a link that has `authenticated-by` metadata attached will lead to the `access-request` identifier of a smartphone. From there on, following a link that has the `access-request-device` metadata attached will lead to the `device` identifier of the respective smartphone. The rest of the IF-MAP graph can be ignored. Thus, the attribute is set to express such a filter. The filter syntax is detailed in the IF-MAP specification [20].

For each smartphone that has been noticed based on the first subscription, another new subscription is created by the Correlation Engine. This second subscription is used in order to get notified about any changes that are made to the sub graph that contains the respective smartphone's Feature metadata. An example of such a subscription is depicted in Listing 5.13.

```
<subscribe>
<update name='subscription-smartphone-23b3'
    match-links='device-category or subcategory-of'>
<device>
  <name>smartphone-23b3</name>
</device>
</update>
</subscribe>
```

Listing 5.13: Subscription that covers Feature metadata for a single smartphone. Namespace declarations and operational attributes are omitted for readability.

Three major differences compared to the first subscription are important:

- The root identifier of the subscription is the `device` identifier of the smartphone. This identifier was noticed thanks to the first subscription.

- A maximum depth is not given.
- The `match-links` attribute is set in such a way that only links who have either `device-category` or `subcategory-of` metadata attached are considered by the subscription.

Thus, the subscription covers the complete sub graph that contains the Feature metadata for a single smartphone. Referring to Figure 5.1, this is the sub graph that is depicted in blue tones, with the `device` identifier for the smartphone "smartphone-23b3" as root identifier.

In order to receive changes that affect one of the subscriptions as soon as possible, irondetect ensures that there is always a blocking poll operation present. This is also referred to as a *pending* poll. For this purpose, irondetect establishes an ARC to the MAP server and issues a poll. If it receives a response on a pending poll, the data that is contained in the response is forwarded internally for further processing. Immediately after that, another request that contains a poll operation is issued via the ARC. Thus, changes of the IF-MAP graph that match any of the subscriptions created by the Correlation Engine are directly received.

Each time the results of a poll operation are received, the Policy of the Correlation Engine is evaluated as described in Section 4.3.1. The Correlation Engine can easily dispatch to which smartphone the received Feature metadata belongs to. This is possible as the response contains the name of the subscriptions that have caused the results. For example, consider that Feature metadata changes for the smartphone "smartphone-23b3". As the Correlation Engine holds a subscription for the respective `device` identifier, it will get notified about the changes. All received changes will be labeled according to the corresponding subscription, in this case "subscription-smartphone-23b3".

Note that instead of using the subscribe/poll mechanisms, the Correlation Engine could also make use of search operations. Analog to the approach above, one search is responsible for finding `device` identifiers of new smartphones, whereas the second type of search operations is used to retrieve the Feature metadata for a single smartphone. Although this approach would generally work as well, it has a major drawback. The responses to search operations would always include the complete set of Feature metadata that is present in the IF-MAP graph for a specific smartphone. That is, even Feature metadata that has not been changed between two subsequently issued search operations would be encapsulated

in the respective response. In contrast to that, the subscribe/poll approach only communicates changes of Feature metadata between the MAP server and the Correlation Engine. If there are no changes, the poll operation blocks (that is the poll is pending). Since only changes of Feature metadata are communicated, the Correlation Engine maintains its own history of Feature metadata that it has received.

Policy Parser

The prototype implementation of irondetect supports to parse a basic Policy that follows the structure as defined in Section 4.1.5 and the example given in Section 4.4.2. That is, it is composed of IF-THEN Rules. The Condition of each Rule is formulated based on Signatures and Anomalies. Actions are performed for each Rule that fires. The parser for the Policy was generated by using the Java Compiler Compiler (JavaCC)[11]. The specification for the grammar of the Policy is given in Listing A.1.

Note that the Policy language was designed in order to fulfill the basic requirements for the prototype implementation. That is, simple rules should be expressed that work based on Signatures and Anomalies. However, the language that is used to specify the Policy for irondetect was not the main focus of the prototype implementation. During the course of the development, the grammar for the Policy was subsequently extended in oder to provide the required functionality. It might be beneficial to revise the policy language in the future in order to improve its expressiveness and usability. Concrete examples for policies that can be parsed by irondetect will be given as part of the evaluation of the CADS approach in Chapter 6.

Supported Anomaly Detection Techniques

The CADS approach demands that arbitrary anomaly detection techniques can be used to detect abnormal behavior. Thus, irondetect supports a plug-in mechanism for anomaly detection techniques. Implementations of anomaly detection techniques must be provided as Java Archives (JAR files). The archive must contain at least one class that implements the `Procedureable` interface as it is defined by irondetect. Otherwise, it is ignored.

At startup, irondetect scans a predefined folder for appropriate JAR files. Each class that implements the respective interface is loaded as a Procedure. Thus it can be used

[11] http://javacc.java.net/

within irondetect's policy in order to express hints and anomalies. In order to identify a certain Procedure, the fully qualified name of the class that implements the `Procedureable` interface is used. The interface basically defines methods that allow (1) to load the configuration for a Procedure, (2) to train the Procedure in order to learn the profile for a smartphone, (3) to calculate the result of a Procedure during testing mode and (4) to tear down a Procedure when irondetect is terminated.

The prototype implementation of irondetect provides a set of Procedures that work based on simple mathematical statistics. They are implemented based on the Apache Commons Math library[12]. The Procedures allow to calculate the arithmetic mean, the variance and the standard deviation of quantitative Features. Furthermore, one Procedure supports to employ a simple linear regression [172] based on quantitative Features. The result of the Procedure is calculated based on the slope of the fitted line. The Procedure will be used during the evaluation in order to detect excessive outgoing traffic that originates from a smartphone.

In general, all of the implemented Procedures calculate a value based on the available Features during testing mode. The value is then compared against an expected value. The expected value has either been trained or was predefined. In order to obtain a Procedure result (and thus a result for the respective Hint that uses the Procedure), an output mapping must be performed as described in Section 4.1.4. That is, based on the calculated and the expected value, a score in the range of $[-1, 1]$ must be returned. The prototype implementation allows to flexibly define how this mapping should be done. The default output mapping works based on two predefined values for the tolerance and the threshold:

- If the distance of the calculated and the expected value is less than the tolerance, a score of 0 is returned.
- If the distance of the calculated value is greater than the tolerance but lesser than the threshold, a score of 0.5 is returned.
- If the distance of the calculated value is greater than the threshold, a score of 1.0 is returned.

Note that this basic output mapping can be customized if necessary.

[12] http://commons.apache.org/math/

Training and Testing Mode

The prototype implementation also supports two operating modes for training and testing as defined in Section 4.3.4. Training data can be provided to irondetect in two ways: (1) via the Feature Provider and (2) via db4o[13] database files that contain a set of previously measured Features. In case the database files are used, a single file must be provided for each smartphone. Based on the included Features, the Correlation Engine can create smartphone-specific profiles. The option to provide training data in the form of database files was implemented in order to allow easy collection of Features on smartphones, even when they are not connected to the respective IT infrastructure, and thus cannot publish the collected Features to the Feature provider.

The actual details on how the training is performed are specific for each Procedure. For example, the mean Procedure simply calculates the mean over all Features that are available for training. On the other hand, the Procedure that implements the simple linear regression does not process all available Features at once for training. Instead, it sorts the available Features based on their timestamp and sequentially performs a linear regression for a subset of the Features. The default configuration is to use 10 Features for each linear regression step. For each linear regression step, the calculated slope is recorded. After all regression steps have been performed, the average of all calculated slope results is stored as trained value for the respective Procedure.

5.2.5 Feature Collectors

For the prototype implementation, three kinds of Feature Collectors have been developed:

1. Feature Collectors that are deployed on smartphones with the Google Android operating system,
2. a Feature Collector that extends the open source vulnerability scanner OpenVAS and
3. a collection of command line tools that is capable of publishing arbitrary Feature metadata based on command line arguments.

They are detailed in the following.

[13] http://en.wikipedia.org/wiki/Db4o

Feature Collectors for Google Android

Most of the Features that are used within the prototype implementation are directly collected on the smartphones themselves. Thus, Feature Collectors that can be deployed on smartphones are necessary. The Google Android platform was chosen for development as (1) it is the predominant platform for modern smartphones and (2) the Android Software Development Kit[14] is freely available. The Feature Collectors are implemented as ordinary apps for Google Android. Thus, no modifications to the smartphone platform are necessary. Two separate Feature Collectors have been developed for the Android platform: one that is used to collect Features during the testing phase and another one that is used to collect Features for the training phase. Although both Feature Collectors share similar functionality, there are some differences that will be mentioned here.

Feature Collector for the Testing Phase The Feature Collector that is used to collect Features during the testing phase was developed in collaboration with partners of the ESUKOM research project [21], especially with the DECOIT GmbH. An except of the set of Features that can be collected with it is depicted in Table 5.1. It is a subset of Features that were defined as part of the exemplary domain instance derivation listed in Table A.2. The focus of the implementation was to include Features that are most important in order to address the scenarios defined in Section 2.2. Once started, the app runs in the background and continuously measures the respective Features.

Table 5.1: A list of Features and Categories that are collected by the Feature Collector for Google Android.

Category/Feature	**Description**
smartphone	Top level Category for smartphone Features
ContextPing	Used to get updated Context Parameters
smartphone.system	Low level Features of the smartphone
MacAddress	MAC address of the WiFi interface
IpAddress	IP address of the interface used to access the Internet
Imei	The smartphone's IMEI
smartphone.android	Category for Android specific Features
KernelVersion	The version of the Linux kernel
FirmwareVersion	The version of Android
smartphone.android.app	Category that includes Features of an Android app

[14] http://developer.android.com/sdk/index.html

Category/Feature	Description
Name	The name of the app
Installer	The app store it was installed from
VersionName	A string value that represents the release version of the application code
VersionCode	An integer value that represents the version of the application code
IsRunning	Indicates whether the app is running or not
smartphone.android.app.permission	Category to encapsulate Features that describe an app's permissions
Requested	A permission that was requested by the app
smartphone.sensor.camera	Category for Features that describe the smartphone's camera
IsUsed	Indicates that the camera is currently in use by an app
smartphone.communication.ip	Category that includes Feature to describe the traffic that is received and transmitted
Rx3g	Traffic received via 3G
Tx3g	Traffic transmitted via 3G
RxOther	Traffic received via Wifi (or other interfaces)
TxOther	Traffic transmitted via Wifi (or other interfaces)
smartphone.system.battery	Category that includes Features to describe the battery status of the smartphone
Level	Battery level in percent

Features are transmitted to the Feature Provider via IF-MAP. Several aspects of the Feature Collector can be configured within the app. This includes the set of Features that actually should be collected and the interval at which periodic updates are sent to the Feature Provider. The configuration parameters can be used to reduce the amount of traffic that is generated by the Feature Collector. By default, the collected Features are published to a `device` identifier which is derived from the smartphone's IMEI.

Feature Collector for the Training Phase Features that are used as input for the Correlation Engine in order to perform a training are collected with another app: the FHH Device Analyzer. The development started as part of a master's thesis [173] and is now maintained by the Trust@FHH research group.

The FHH Device Analyzer does not make use of the IF-MAP protocol. Instead, it stores collected Features in db4o database files. It is an exception compared to any other Feature Collector. The training data should be collected over a potentially long period of time

(several weeks). For this purpose, it is not necessary to transmit the collected Features immediately to the Feature Provider, respectively the MAP server. Instead, the set of Features can be provided at once when the collection of training data is finished.

The FHH Device Analyzer allows to capture all Features of the Category *smartphone* and its sub Categories as depicted in Table A.2, excluding those that can only be obtained by crawling the Google Play app store[15]. In order to unambiguously identify the smartphone to which the collected Features belong to, an ID is derived from its IMEI. The algorithm is the same as the one that is used by the other Feature Collector for Android. Thus, Features that are obtained for training and those that are obtained during testing mode can be mapped by the Correlation Engine whether they belong to the same smartphone or not. The database files are sent to a remote server located in the infrastructure of the Hochschule Hannover. From there, they can be copied to a location that is accessible by the Correlation Engine. The user of the smartphone has to confirm that it wishes to share the collected data. Further details on the app can be found at the official project website [174].

Thus, there are two separate Feature Collectors for Google Android, each of them having a different purpose. However, they share similar functionality, especially regarding the code that measures Features on Android smartphones. Merging the functionality of both Feature Collectors into a single app is a subject of future work.

Feature Collector for OpenVAS

In order to demonstrate the capability to integrate existing security services, a Feature Collector for the OpenVAS vulnerability scanner was developed called “ironvas”. The implementation began as part of a bachelor thesis [175] and is now maintained by the Trust@FHH group. ironvas is capable of fetching the vulnerability reports that are stored in an OpenVAS server, maps them to appropriate Feature metadata and publishes them to the Feature Provider. That is, the latest results of vulnerability scans are reflected as Features within the Feature Provider. As each vulnerability report contains the IP address at which the vulnerability was detected, the correct `device` identifier to which the

[15]These Features are: GooglePlayCategory, Rating and Downloads.

respective Feature metadata must be published to can be obtained[16]. Table 5.2 highlights the Features that can be collected for each report that is stored on the OpenVAS server.

Table 5.2: A list of Features that are collected by the Feature Collector ironvas.

Category/Feature	**Description**
vulnerability	Category for vulnerability scan results from OpenVAS
Name	The name of the Network Vulnerability Test (NVT)[17] that was performed
Port	The port at which the vulnerability was detected
CvssBase	The CVSS[18] base score that was defined for the vulnerability. Ranges from 0 to 10.
Threat	The threat level of the vulnerability as determined by OpenVAS
CveIdentifier	The CVE[19] number that identifies the vulnerability

The integration of OpenVAS allows to include information about the amount of known vulnerabilities when reasoning about the security status of smartphones. Further details on the configuration of ironvas can be found at the project documentation website [176].

General Purpose Feature Collector based on Command Line Tools

The third Feature Collector that was developed consists of a set of command line tools referred to as "ifmapcli". They allow to script the behavior of other security services (like an IDS or a NAC solution) without adding the complexity to integrate even more existing software components. There are basically three kinds of commands supported by ifmapcli: (1) commands that allow to publish IF-MAP standard metadata to a MAP server, (2) commands that allow to publish Feature metadata and (3) commands that allow to search and subscribe for metadata.

ifmapcli will be used in the remainder of this thesis for two purposes: (1) to simulate a PDP that publishes standard IF-MAP metadata for smartphones that access the network and (2) to simulate an IDS that can publish Feature metadata of the Category *ids*. All Features that have been defined for the *ids* Category are listed in Table 5.3.

[16] Starting a search from the `ip-address` identifier with depth 2 following links that have either `access-request-ip` or `access-request-device` metadata attached will lead to the smartphone's `device` identifier.
[17] http://www.openvas.org/openvas-nvt-feed.html
[18] https://nvd.nist.gov/cvss.cfm
[19] http://cve.mitre.org/

Table 5.3: List of Features of the Category *ids* (derived from [154]).

Category/Feature	**Description**
ids	Category that encapsulates Features from an IDS
EventType	Type of an event
EventName	Name of an event
EventSource	Source of the event
EventThreat	Threat level that is associated with the event
EventConfidence	Confidence of the IDS that the event occurred
EventImpact	Impact of the event
HighTraffic	Special event that indicates that there was a high amount of traffic

5.2.6 Feature Consumer

One generic Feature Consumer was developed as part of the prototype implementation. It is referred to as "ironmonitor". It provides means that allow the Correlation Engine to trigger the execution of arbitrary commands on any system that has this respective software component installed. ironmonitor includes basic functionality to issue subscriptions based on single `device` identifiers that represent smartphones. The subscription can be configured to match any type of Category and Feature metadata.

In the remainder of this thesis, ironmonitor will be used to process Features that have been created by the Correlation Engine. That is, Features of the Category *correlationresult*. Within the prototype implementation, this is used to invoke the iptables command in order to adjust the configuration of a Linux-based firewall depending on the results of the Correlation Engine.

5.3 Identified Issues and Limitations

During the course of the prototype implementation, some limitations and issues were encountered. They are briefly mentioned in the following.

First, ensuring the consistency of the identifiers of Features and Categories was cumbersome. This became especially obvious during the parallel development of the Feature Collectors for Android. Often, Features and Categories with the same semantics were

named slightly differently (for example MAC vs. MacAddress). That is, implementers of the CADS approach should employ consistency checks of the defined set of Features and Categories on a regular basis.

Second, the effort that is necessary to coordinate multiple MAP clients should not be underestimated. For the CADS prototype, this is achieved by leveraging the IMEI of smartphones in order to derive a unique `device` identifier for each smartphone. MAP clients that have no access to the IMEI need to search the IF-MAP graph for the respective `device` identifier. If they cannot specify an appropriate root identifier for the search operation based on the data they are processing, configuration parameters that are set manually are needed.

Third, the collection of training data is currently limited to those Features that are obtained directly on smartphones. For this purpose, a dedicated Feature Collector, the FHH Device Analyzer, was developed. It does not make use of the IF-MAP protocol. Instead, collected Features are (1) stored locally on the device in db4o database files and (2) sent to a dedicated remote server via HTTP. This allows to collect a large set of Features over a long period of time, without requiring access to a MAP server at any time. In order to trigger the training of profiles by the Correlation Engine, the db4o database files need to be manually copied to a location that is accessible by the Correlation Engine.

5.4 Summary

This chapter described a prototype implementation of the CADS approach based upon the IF-MAP protocol. The fact that IF-MAP is a suitable communication protocol for implementing the CADS approach was detailed in Section 5.1. In order to use IF-MAP, Features, Categories and Context Parameters of the CADS conceptual model were mapped to the IF-MAP data model. Furthermore, the CADS logical roles were mapped to those roles that are defined by the IF-MAP specification. The mapping was detailed in Section 5.1. An overview about the implemented software components and their supported functionality was given in Section 5.2. The main contribution was made by implementing the Correlation Engine irondetect, the set of Feature Collectors and the Feature Consumer ironmonitor. The existing MAP server irond could be used to fulfill the role of the Feature Provider without any additional modifications. Issues that have been identified during the course of the implementation were detailed in Section 5.3. Although these issues should

be tackled by future work, the presented implementation of the CADS prototype allows to successfully address the scenarios described in Section 2.2. This will be proven by an evaluation of the CADS approach in the next chapter.

6 Evaluation

"I love deadlines. I like the whooshing sound they make as they fly by."

(Douglas Adams)

Contents

In this chapter, an evaluation of the CADS approach is performed. Section 6.1 presents an analysis of data that has been collected by the FHH Device Analyzer. Its purpose is to provide insights regarding the question how users actually use their smartphones. After that, the overall performance of the CADS prototype implementation is investigated. As the MAP server is the single point responsible for storing and retrieving collected Features, its performance is crucial. Thus, a performance analysis of the two currently available open-source MAP servers is performed in Section 6.2. One of the Feature Collectors developed as part of the prototype implementation is deployed directly on the respective smartphones. In order to investigate if the amount of data that is needed to transmit the measured Features to the Feature Provider is within reasonable bounds, Section 6.3 details how much overhead in terms of generated traffic is caused by this Feature Collector. That the CADS approach is capable of detecting sensor sniffing attacks is demonstrated in Section 6.4. Finally, the flexibility of the CADS approach is proven in Section 6.5 by specifying a Policy that leads to a similar functionality as provided by the Kirin security service for Android [14].

6.1 Analysis of Data Collected by the FHH Device Analyzer

The FHH Device Analyzer was used to gather data from 13 smartphones over a period of 2.5 months between August 8 2012 and October 22 2012. All of the participants were affiliated with the Trust@FHH research group. In total, 1.3 GB of data were collected. The amount of Features that were collected from the participants varies greatly. Some of the participants chose to restrict the set of Features that should be collected. Figure 6.1

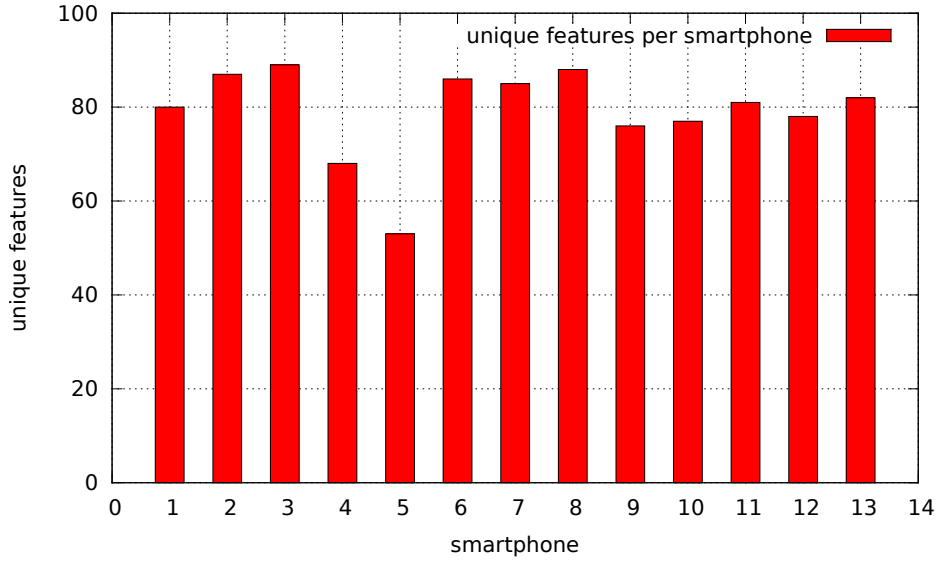

Figure 6.1: Unique Features that have been measured for participating smartphones.

depicts how many different, unique Features have been measured for each participating smartphone. For most of the smartphones, more than 70 different Features were measured. Only two smartphones (device 4 and 5) contributed fewer Features. The number of Feature instances that were measured is depicted in Figure 6.2. In total, more than 4.5 million Feature instances haven been measured. Most of the smartphones uploaded more than 100,000 Feature instances. One smartphone (device 8) even uploaded more than one million Feature instances. Only two smartphones (device 5 and 9) contributed less than 20,000 Feature instances.

As already stated, not all of the collected Features are directly used within the scenarios that are targeted in this thesis. However, a preliminary analysis of relevant collected Features is performed in the following. More precisely, the gathered data will be analyzed in order to find out (1) what apps were installed on the smartphones, (2) how the smartphones behaved in terms of traffic consumption and (3) in how many different environments they were used. The latter is derived from analyzing how many different WAPs a smartphone has "seen" respectively scanned while being used.

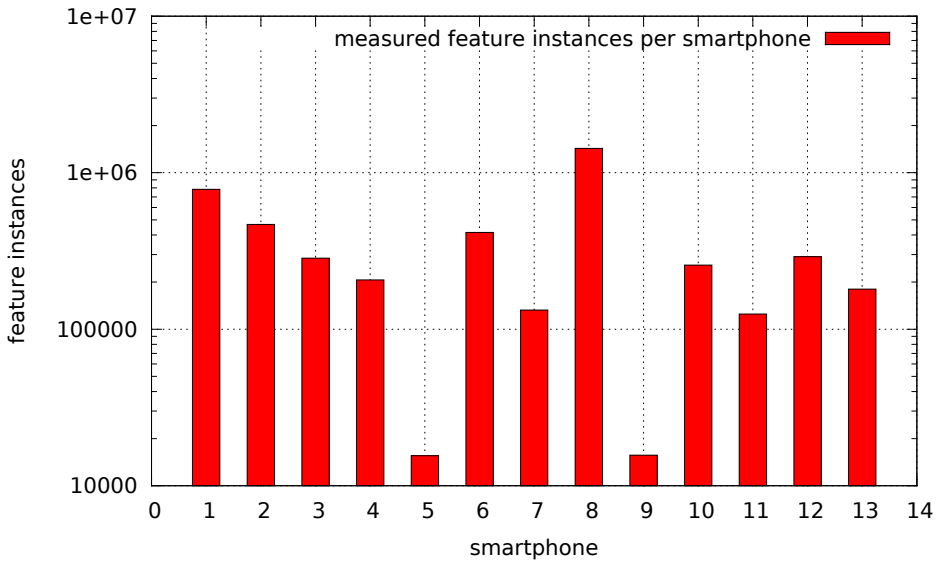

Figure 6.2: Feature instances that have been measured for participating smartphones.

6.1.1 Installed Apps and Requested Permissions

The use of apps is one key aspect of modern smartphones. Thus, an analysis was performed regarding how many apps were installed on the participating smartphones, and how these apps make used of the Android permission system. Figure 6.3 depicts the distribution of installed apps among the participating smartphones, and how many permissions these apps requested for use.

Most of the the participants had more than 40 apps installed. Device 1 represents the maximum with 369 apps. On the other hand, smartphone number 5 only had two third-party apps installed. The sum of installed apps on all devices was 1103, which gives an average of 84 apps per device. Together, they requested 6079 permissions, which is an average of 5.5 permissions per app. However, this set includes a lot of duplicates, meaning that many apps tend to request similar permissions. In total, the installed apps requested 283 different permissions. However, more than 40% of them were only requested by just one of the installed apps at a time. This is due to the fact that many permissions are vendor respectively app specific. For example, the permission `com.symantec.permission.ACCESS_NORTON_SECURITY` was only requested by a single app in the whole dataset. Another permission that was only requested once revealed that

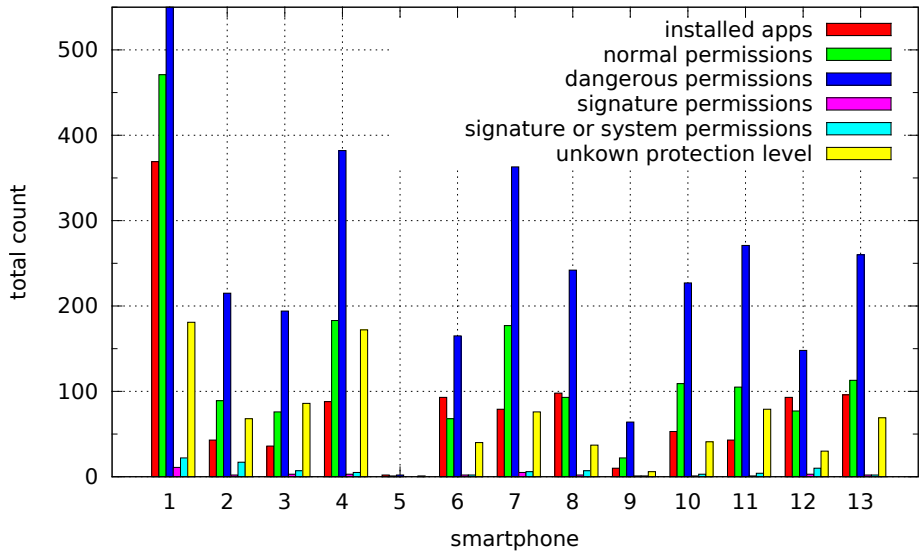

Figure 6.3: Installed apps and their requested permissions. Note that the exact number of requested dangerous permissions for the first smartphone is 1792. The y-axis has been truncated for readability.

the author of the corresponding app likely made a typing error during the development phase (typing `andorid` instead of `android` in the permission label).

In contrast to those permissions that are only requested once, there are some permissions that are requested far more often compared to others. The ten most frequently requested permissions are listed in Table 6.1. As expected, the `INTERNET` permission is requested most often.

An interesting observation could be made in terms of requested permissions and their protection levels. Most of the requested permissions have a protection level of "dangerous", indicating that they are potentially harmful (as explained in Section 3.2.4). Note that the exact number of requested dangerous permissions for the first smartphone is 1792. The maximum value of the y-axis was limited to 550 in order to ensure readability. Permissions with a protection level of "normal" are roughly requested half as often. Permissions with a protection level of "signature" and "signature or system" are not widely used compared to the other protection levels. For a significant number of permissions, the protection level could not be determined. This is the case when developers have defined their own permissions. The FHH Device Analyzer does not support to obtain the correct protection

Table 6.1: The ten most frequently requested permissions in the dataset.

Permission	Total Count	%
android.permission.INTERNET	629	57
android.permission.ACCESS_NETWORK_STATE	528	47
android.permission.WRITE_EXTERNAL_STORAGE	497	45
android.permission.WAKE_LOCK	303	27
android.permission.READ_PHONE_STATE	261	23
android.permission.VIBRATE	252	22
android.permission.ACCESS_WIFI_STATE	225	20
android.permission.ACCESS_FINE_LOCATION	192	17
android.permission.ACCESS_COARSE_LOCATION	173	15
android.permission.BLUETOOTH	147	13

level for these permissions. However, it can be stated that developers make frequent use of defining their own permissions.

6.1.2 Traffic Consumption

The FHH Device Analyzer supports to collect Features of the Category *smartphone.communication.ip* which allow to analyze how much data a smartphone transmits and receives. More precisely, the Features encapsulate how much bytes were transmitted and received either via the mobile network interface (*Rx3g, Tx3g*), or via any other interface, including WiFi (*RxOther, TxOther*). The results were rendered as box plots [177]. The colored box spans from the lower quartile (Q1, 25 percentile) to the upper quartile (Q3, 75 percentile). The band inside the box represents the median (Q2, 50 percentile). The ends of the whiskers represent the lowest / highest values that are still within 1.5 times of the interquartile range (IQR)[1].

An overview of the received and transmitted amounts of data is depicted in Figure 6.4. The box plots of devices 4, 7 and 10 have been truncated at 1,000 KB. Their upper whiskers are at 1,250 KB, 2,300 KB and 1,100 KB respectively. Again, the results are diverse. Independently of the interface that is being used, smartphones receive more data than they transmit. However, peaks in terms of the amount of transmitted and received data exist. This is especially true for device 7. Except for two devices (4 and 8), smartphones made use of both the WiFi and the mobile network interface (3g) for transmitting and

[1] http://en.wikipedia.org/wiki/Interquartile_range

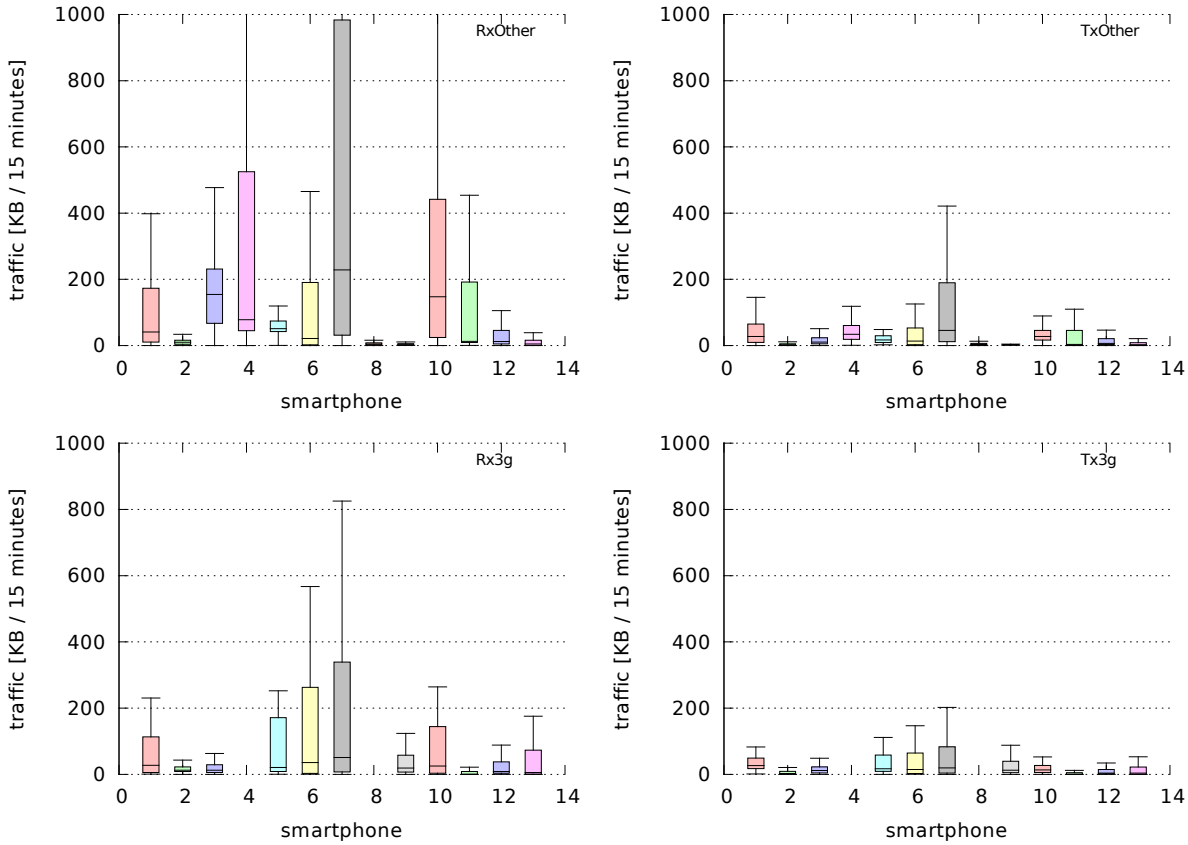

Figure 6.4: Overview of received and transmitted data. The box plots of devices 4, 7 and 10 have been truncated at 1,000 KB. Their upper whiskers are at 1,250 KB, 2,300 KB and 1,100 KB respectively.

receiving data. The details of the box plots of each Feature are depicted in Figure 6.5, 6.6, 6.7 and 6.8. Note that each of them has a different scaling for the y-axis.

Outliers are omitted in the figures. Most of them were caused by very low measurement values (below 0.1 KB) at times when the respective smartphone was not used at all (at night). Upper outliers have been detected as well. Regarding the upper outliers for the received data, they were likely caused when the smartphone was used for receiving emails with attachments, installing respectively updating of installed apps or for consuming services like Google Maps or YouTube which are expensive in terms of generated traffic. The upper outliers of the outgoing traffic were mainly caused by the FHH Device Analyzer itself (for example when the user waited a long time before uploading the first dataset of collected Features).

Overall, the amount of data that is received and especially that is transmitted is low (compared to the technical capabilities of modern smartphones). Median values of all smartphones are lower than 300 KB per 15 minutes, which is a rate of 0.33 KB/s. When only the Feature *TxOther* is considered, median values are always below 50 KB per 15 minutes. This corresponds to a rate of 0.06 KB/s. However, the upper whiskers indicate that there are peaks in terms of a smartphone's traffic consumption. The fact that smartphones usually do not transmit large amounts of data will be used for the detection of a sensor sniffing attack in Section 6.4.

6.1.3 Scanned Wireless Access Points

In order to understand in what different environments smartphones are used, an analysis based on WAPs that have been scanned by smartphones was conducted. Android smartphones scan for nearby WAPs on a regular basis as long as the WiFi interface is enabled. The FHH Device Analyzer app gets notified after each scan and renders the results to Features of the Category *smartphone.communication.wifi.scan*. This especially includes the Basic Service Set Identification (BSSID) and the Service Set Identification (SSID) for each scanned WAP. The number of different access points that have been scanned by each participating smartphone is depicted in Figure 6.9. The results are diverse. Three of the smartphones have scanned more than 1000 different access points. Device 1 even scanned almost 2000 different access points. These numbers were higher than expected. On the other hand, three devices scanned less than 10 different access points (device 5, 9 and

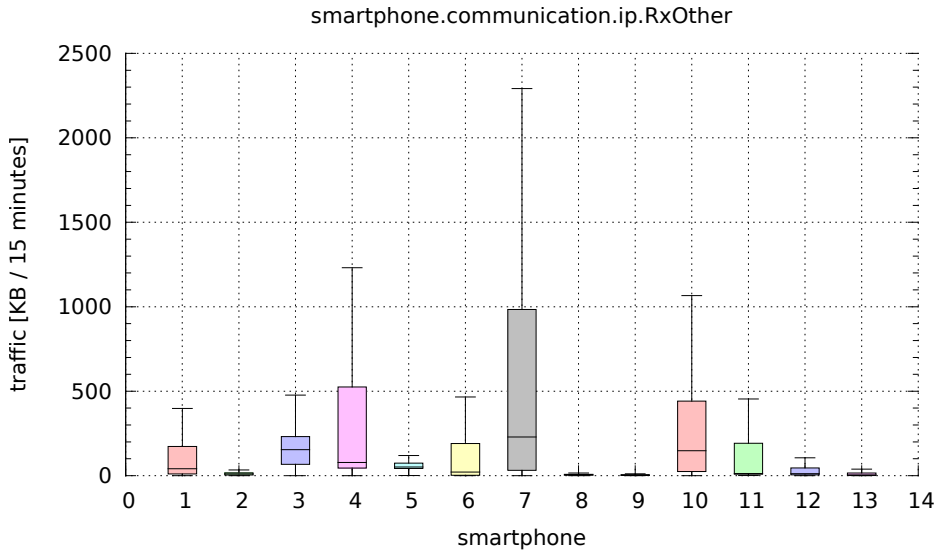

Figure 6.5: Received traffic via Wifi or other interface except 3g.

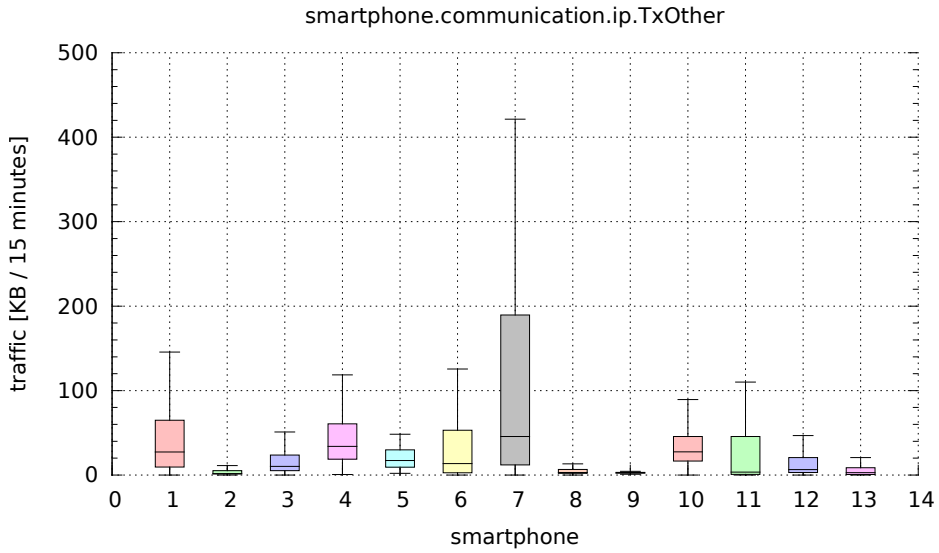

Figure 6.6: Transmitted traffic via Wifi or other interface except 3g.

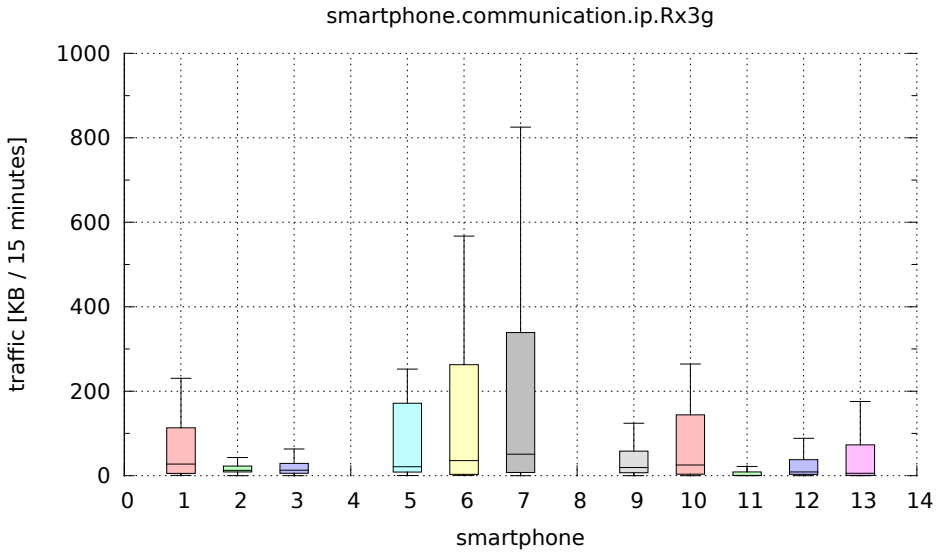

Figure 6.7: Received traffic via 3g.

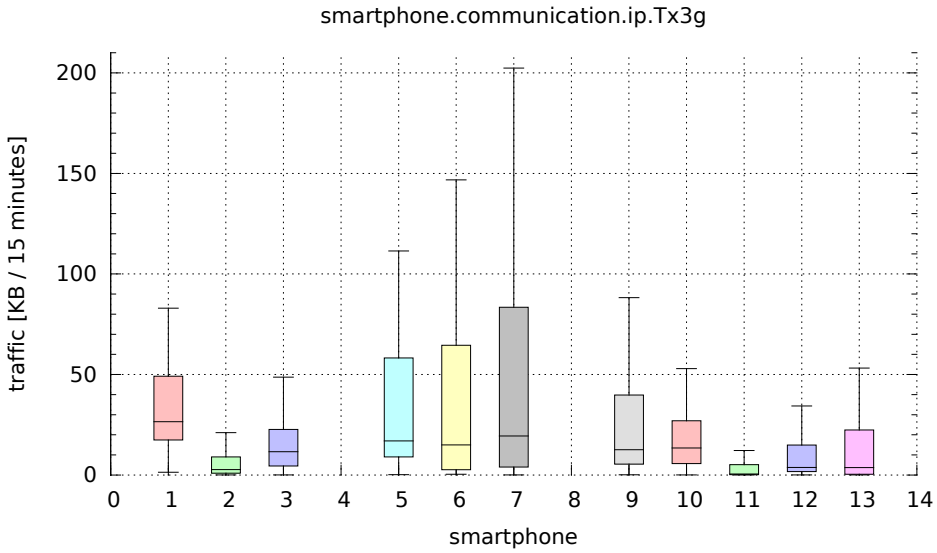

Figure 6.8: Transmitted traffic via 3g.

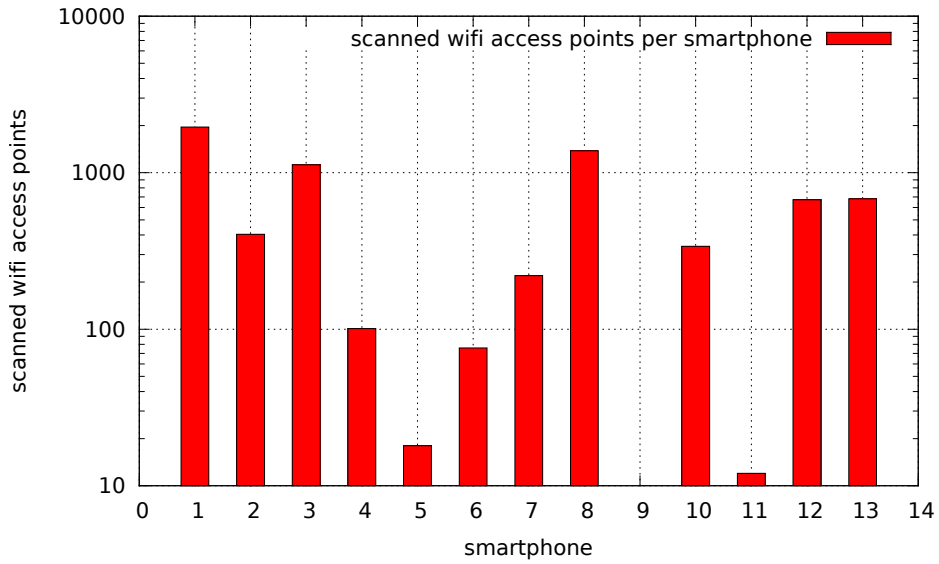

Figure 6.9: Scanned WAPs per smartphone.

11). All scans were performed in the area of Hanover, Germany. The results proof that individual smartphones are frequently being used in different environments. Note that the scan results do not imply that the smartphone has also established a connection to the respective WAP.

As a side effect, the analysis also revealed the configuration details of the scanned WAPs (via the Feature *smartphone.communication.wifi.scan.Capabilities*). These results are not relevant for the remainder of the thesis. Nevertheless, one aspect should be mentioned here. More than half of the scanned access points had support for WiFi Protected Setup (WPS) enabled. This protocol is prone to brute force attacks, as was recently proven by Viehböck [178]. More details on the FHH Device Analyzer app are provided by the master's thesis of Tobias Ruhe [173].

This section provided an analysis of the data that was collected with the FHH Device Analyzer. It provided some insights in terms of how smartphones are actually used. The main conclusions that can be drawn and that is relevant for the remainder of this thesis is that smartphones usually transmit and receive only low amounts of data compared to the theoretically available bandwidth.

6.2 Performance Analysis of MAP Servers

IF-MAP was chosen as communication protocol for the CADS prototype. The respective specifications demand that the protocol is scalable and efficient in order to be used in environments with thousands of MAP clients where thousands of metadata updates occur per second at a single MAP server [20]. The crucial bottleneck in terms of performance is the MAP server. In order to check how a MAP server scales when Feature metadata is published and retrieved for numerous smartphones, a performance analysis was conducted for the two currently available open source MAP servers. irond, which is mainly used as part of the CADS prototype implementation, and omapd.

6.2.1 Definition of Test Case

Within the CADS prototype implementation, there are generally two sorts of metadata published to the MAP server: (1) standard metadata mainly published by a PDP and (2) Feature metadata that is published by arbitrary Feature Collectors. The general structure of the resulting MAP graph was already depicted in Figure 5.1. The test case aims to simulate the interaction between the MAP server, a Feature Collector that publishes numerous Feature metadata for a single smartphone and the Correlation Engine. The general sequence of steps is as follows:

- The MAP server and the Correlation Engine are started. In order to get notified when a new smartphone is authenticated by the PDP, the Correlation Engine creates a single subscription for the PDP's `device` identifier with max-depth set to "2" and initiates a poll operation.

- For a certain number of different smartphones, the following operations are performed in order:

  1. The set of PDP standard metadata is published (including `authenticated-by` and `access-request-device` metadata). This links the PDP's `device` identifier to the `device` identifier of the respective smartphone. The publish operation must be acknowledged by the MAP server.

  2. A set of Feature metadata is published to the `device` identifier of the respective smartphone. Again, the MAP server must acknowledge the successful

publish operation before the test proceeds with publishing metadata for the next smartphone.

For the performance tests, a MAP client that publishes both the standard and the Feature metadata for an arbitrary number of smartphones was implemented. The wall time is measured for each of the two publish operations per simulated smartphone. It is important to note that after the first publish operation was performed, the Correlation Engine creates a new, smartphone specific subscription using the smartphone's `device` identifier. The details on the subscription handling of the Correlation Engine were discussed in Section 5.2.4. Furthermore, it is expected that a publish operation is not acknowledged until all currently available subscriptions have been processed by the MAP server. Both irond and omapd work this way.

The following parameters can be adjusted in order to customized the test case:

- *Smartphones*: The number of smartphones for which metadata should be published.
- Three parameters can be used in order to change the size and the structure of the sub graph that includes the Feature metadata and the Category identifiers:
  - *Categories*: The number of child Category identifiers that are created for each identifier in the sub tree, starting from the `device` identifier of the respective smartphone.
  - *Features*: The number of Feature metadata that is published to each Category identifier.
  - *Depth*: The depth of the sub tree. A depth of "0" means that only a single level of Category identifiers is created which are directly linked to the `device` identifier of the respective smartphone.

  Note that the three parameters are set once and thus are valid for all smartphones that are simulated by the test.

The performance tests were done with three different sets of parameters as listed in Table 6.2. All tests were run by simulating 256 smartphones. The size of the sub graphs is given in number of metadata objects. For the first test case, 401 metadata objects need to be stored by the MAP server (400 metadata objects of type `feature` and a single metadata object of type `device-category`). For the second test case, the number is almost equal.

Table 6.2: Size of the sub graph depending on the parameters of the performance test program. The size is given in metadata objects that must be stored by the MAP server.

#	Smartphones	Categories	Features	Depth	Size of sub graph	
					one	all
1	256	1	400	0	401	102,656
2	256	1	200	1	402	102,656
3	256	2	10	4	682	174,592

Only one additional `subcategory-of` metadata object is needed in order to realize the depth of "1".

For the third test case, 682 metadata objects need to be stored per sub graph. As the *categories* parameter is set to "2", there are two links with `device-category` metadata attached that connect each smartphone's `device` identifier with a corresponding Category identifier. Considering the sub graph that starts with one of these Category identifiers, there are $2^{depth+1}-1 = 31$ identifiers and $2^{depth+1}-2 = 30$ links in it. Each link has attached a metadata object of type `subcategory-of`. Furthermore, each Category identifier has "10" metadata objects of type `feature` attached. Thus, the number of metadata objects for the sub graph is $31 * 10 + 30 = 340$. Since the *categories* parameter is set to "2", there are two such sub graphs. Considering the two metadata objects of type `device-category` mentioned in the beginning, the total number of metadata objects for the sub graph of a single smartphone is $(2 * 340) + 2 = 682$. In this case, the subgraph has the structure of a full binary tree: each identifier has two children, except those that are the leaves of the tree.

Note that in addition to the metadata objects, the MAP server is also required to store the identifiers in an appropriate way.

6.2.2 Testing Environment

All tests were run on a single virtual machine with 2048 MB RAM and 4 vCPUs (Intel(R) Xeon(R) CPU E5520). The guest system was Debian Wheezy with Sun JRE (1.6.0_26-b03) and g++ (4.7.1). The MAP servers irond-0.3.4 and omapd-0.7.3 have both been certified by the TCG. In contrast to irond, omapd is written in C++ using the Qt Framework from Nokia. The MAP client for performance testing, the MAP servers as well as the

Correlation Engine were executed on the same machine. That is, network communication between MAP clients and MAP servers was done by using the loopback interface only. In order to ensure that irond does not run out of memory, the heap size was set to 768 MB.

6.2.3 Results

The results of the performance tests are depicted in Figure 6.10 and 6.11. They include the wall times measured by the performance testing MAP client for the three test cases described above.

irond performs well in all of the three test cases. The first publish operations are rather slow. Standard metadata takes approximately 0.8 seconds, Feature metadata takes almost 1.1 seconds to be published. However, after metadata has been published for the first 25 smartphones, the wall time has lowered significantly and remains rather constant for both the standard metadata (approx. 0.01s) and Feature metadata (approx. 0.15s to 0.25s). The performance improvement is mainly caused by optimizations of the Java HotSpot Virtual Machine[2]. Wall times for the first two test cases are almost identical. This is reasonable as the structure of the sub graph that contains the Feature metadata only changes slightly, and thus the amount of metadata objects that need to be stored is almost the same. However, the wall times for the third test case differ from the rest. Interestingly, the wall time for the standard metadata in this case is lower than for the first two test cases, although the same amount of metadata has been published. As expected, the wall time for the Feature metadata is higher compared to the rest. This is reasonable as the size of the sub graph per smartphone is bigger (682 compared to 401 respectively 402 metadata objects for the first two test cases). At the end of the third test case, starting with smartphone 245, the wall times tend to increase. This is because irond is going to run out of heap space. In fact, constantly publishing new metadata to irond without deleting any of it will ultimately lead to a `OutOfMemoryError`[3] thrown by the Java virtual machine. However, as long as enough memory is available, irond has a constant wall time for publishing both standard and Feature metadata for an arbitrary number of smartphones as defined by the test cases. The subscriptions that are created by the Correlation Engine have no negative impact on the overall performance.

[2] http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136373.html
[3] http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/OutOfMemoryError.html

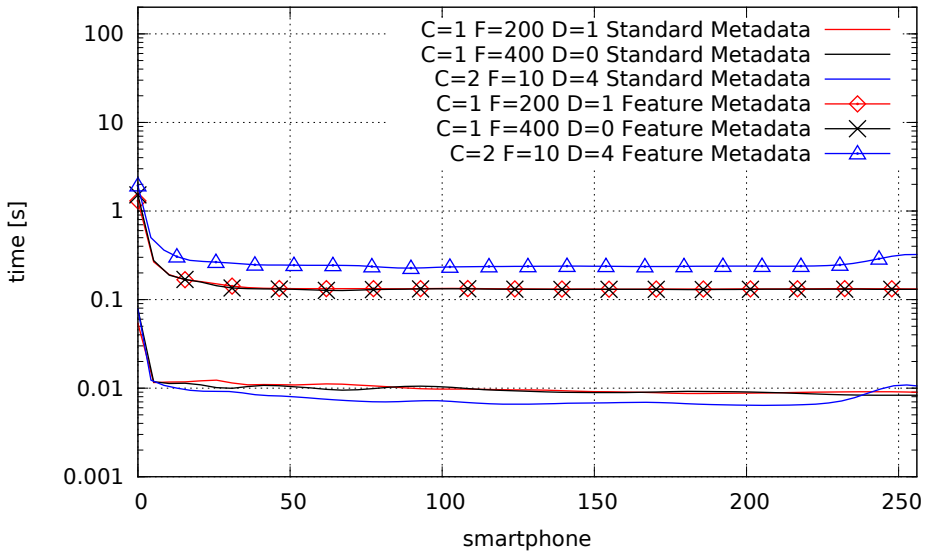

Figure 6.10: Performance analysis irond.

The MAP server omapd performs inferior compared to irond. It manages to fulfill the first two test cases in a reasonable amount of time. The first publish operations even outperform irond. However, the wall time increases steadily the more smartphones are involved, clearly exceeding those of irond. Starting from smartphone 50, the standard set of metadata even takes longer to be published compared to the Feature metadata. The situation is even worse for the third test case. Wall times increased exponentially and the test case was aborted after metadata was published for smartphone number 23. At that time, publishing Feature metadata for a single smartphone took more than 100 seconds. It is assumed that the weak performance is caused by a poor handling of multiple subscriptions. In the third test case, the sub graph for each smartphone is rather complex. The Correlation Engine issues a single subscription that explicitly targets each of those sub graphs. Thus, when Feature metadata is published for a single smartphone, only one subscription actually needs to be evaluated. However, omapd seems to evaluate all subscriptions that are currently active. That is, it cannot determine in advance that the published Feature metadata only affects a single sub graph. Considering these results, the tested version of omapd is not suitable to be used for implementing the CADS approach. However, this is a problem which is caused by the respective implementation, not by the IF-MAP protocol itself.

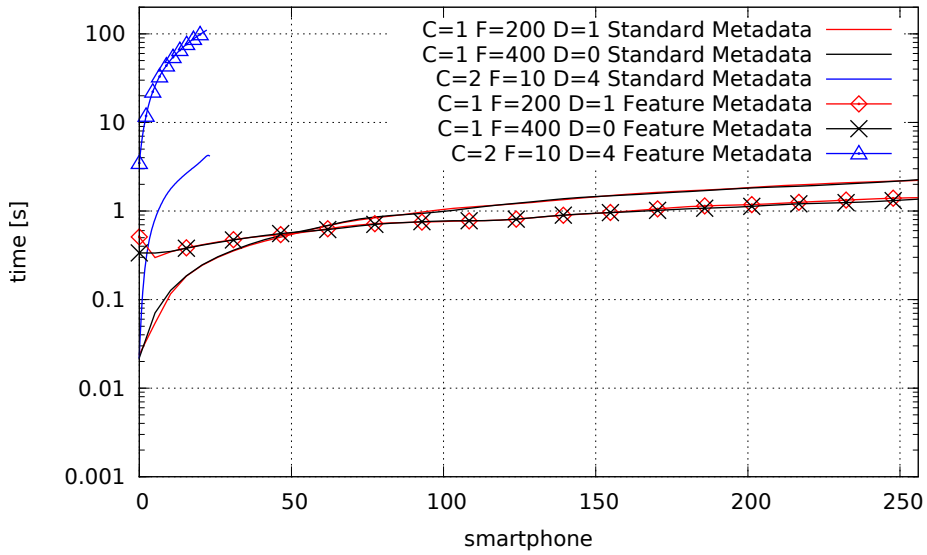

Figure 6.11: Performance analysis omapd.

To summarize, the performance tests that were conducted with the two currently available open source MAP servers yielded diverse results. Concerning the amount and structure of metadata that is published when implementing the CADS approach, irond performed well. Although irond is just a prototype implementation, this proofs that the IF-MAP protocol can be implemented in an efficient and scalable manner, and is thus suitable for being used as communication protocol for CADS. On the other hand, the tests conducted with the omapd MAP server yielded poor results. It was generally able to process the standard and Feature metadata that is used within the CADS prototype. However, the wall time for publishing metadata was not acceptable when the sub graph that contains Feature metadata was more complex. Thus, although any MAP server that is compliant to the specification can generally process metadata that is used by CADS correctly, it depends on the concrete implementation whether the performance is sufficient or not.

6.3 Traffic Consumption of the Feature Collector for Android

In the following, an analysis is performed that aims to discover how much traffic is generated by the Feature Collector for Android that is used in the testing phase.

6.3.1 Definition of Test Case

In order to get accurate results, the Feature Collector on the smartphone was tested in an isolated testing network. No route to any external service or the Internet was configured. Only the MAP server irond acting as Feature Provider was configured and running in the testing network. The smartphone was cold booted. Before the test was started, all apps that were listed under Settings→Applications manager→Downloaded were stopped via the "Force Stop" button. After the Feature Collector was started, the smartphone was not used in any way. Gzip compression for IF-MAP was enabled. Bluetooth was turned off. 36 third-party apps were installed on the smartphone at the time of testing. The interval between two IF-MAP publish operations was set to 20 s. Both the outgoing (TxOther) and the incoming (RxOther) traffic for the Wifi interface were measured after a successful IF-MAP publish operation was performed. More precisely, a successful IF-MAP publish operation constitutes of a request message sent from the MAP client to the MAP server that encapsulates the Feature metadata that should be changed and a response message that is sent from the MAP server to the MAP client that states whether the operation was successful or not. Measurements were made directly by the Feature Collector and logged by leveraging the Android logging system (logcat)[4].

6.3.2 Testing Environment

The testing was performed with a Samsung Galaxy S3 running Android version 4.1.1. It was connected to a Lancom L54-g wireless access point. The MAP server irond was running on a separate machine connected to the same wireless network. The smartphone was connected to a laptop via Universal Serial Bus (USB). This way, the logged measurements could easily be obtained by using the Android Debug Bridge (ADB).

[4] http://developer.android.com/tools/help/logcat.html

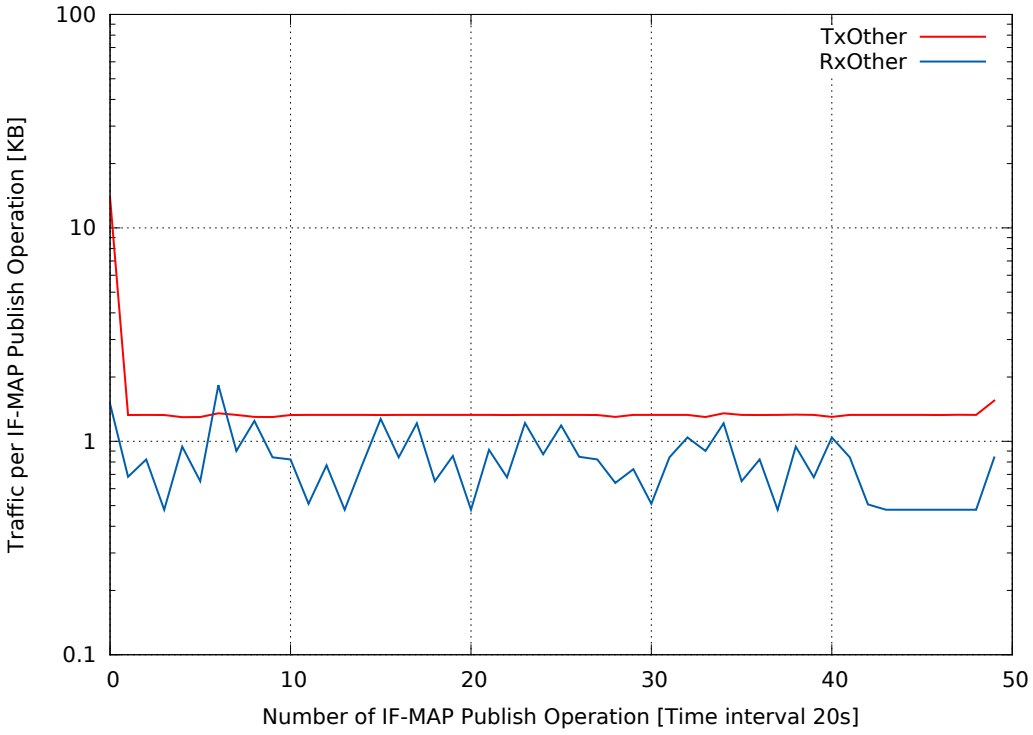

Figure 6.12: Traffic consumption for the Android Feature Collector.

6.3.3 Results

The results of the test are depicted in Figure 6.12. The traffic was measured for 50 IF-MAP publish operations that were executed with a time delay of 20 s. The y-axis depicts the outgoing and the incoming traffic after each publish operation. It is obvious that the first operation causes the largest increase of the outgoing traffic (approximately 14 KB), whereas the subsequent publish operations only need less than 2 KB of outgoing traffic. This is due to the fact that the Feature Collector only transmits Feature metadata for Features whose values have changed. For the first publish operation, all Features have updated values. Thus, respective Feature metadata must be created and transmitted to the Feature Provider for all of them. In subsequent publish operations however, for most of the Features no metadata has to be transmitted again as their respective values have not changed. Considering the Features that are supported to be measured by the Feature Collector for Android (as detailed in Section 5.2.5), most of them are rather static. Only some of them like those of the Category *smartphone.communication.ip* are more dynamic and thus need to be transmitted to the MAP server within each publish operation.

Most of the Feature instances that need to be created refer to the third-party apps that are installed on the respective smartphone and their permissions. Those do not change after the first publish operation, hence the outgoing traffic drops. Incoming traffic remains rather constant at approximately 1 KB per publish operation. This makes sense as the response to a successful publish request is always the same, no matter how much Feature metadata has been published.

To summarize, the prototype Feature Collector for Android causes approximately 2.5 KB of traffic for each IF-MAP publish operation in average (1.6 KB of outgoing and 0.9 KB of incoming traffic). One exception is the initial publish operation which causes approximately 15 KB of traffic. The actual traffic consumption of the Feature Collector can be customized by adjusting the time delay between two subsequent publish operations. Assuming that this interval is set to a value in the range of 30 s to 60 s (which still allows a reasonable reaction time on identified threats), the traffic consumption is negligible in environments were the smartphone is connected via Wifi. When the smartphone is connected to the IT infrastructure via a mobile network carrier, it depends on the respective rate for using mobile data whether the amount of generated traffic is acceptable or not. Parameters that can be adjusted in order to lower the traffic consumption are

1. the update interval that determines how often IF-MAP publish requests are sent,
2. the number of Features that are measured and
3. the number of Context-Parameters that are given for each measured Feature.

6.4 Detection of Sensor Sniffing

6.4.1 Overview

In the following, the ability of CADS to detect and to react on an exemplary sensor sniffing attack is described. The assumption is that a benign app is used within the IT infrastructure of an enterprise to capture video data by leveraging a smartphone's built-in sensors. The captured data is transmitted to a remote location, that is it leaves the smartphone device. In terms of smartphone security, it is reasonable to address this kind of threat as smartphones are the only devices that come with numerous built-in sensors. Once the policy violation is detected, it should be countered immediately. That is, the outgoing traffic that originates from the smartphone should be blocked.

In order to perform the sensor sniffing, the app IP Webcam[5] is used. It is a benign app that is available via the official Google Play store. It provides basic functionalities in order to use a smartphone as a web cam. That is, it implements an HTTP server and allows to stream both audio and video data. The app can be configured to automatically start itself when the smartphone finished booting. Furthermore, it can be configured to run in the background, without being visually noticeable by the user.

The app itself cannot be classified as being malicious. If users want to use their smartphone as a web cam, they are free to do so. However, within the context of an enterprise environment, the respective company might consider the streaming of video and audio data within their infrastructure as a potential threat, and thus as an act that violates their policy. It is important to note that general anti virus scanning solutions are not applicable for this use case, as the respective app is not malicious. For the detection, the process how and why the app was installed on the respective smartphone is irrelevant. The user might have willingly chosen to install it or might have been fooled to do so by other means (such as via scanning of faked QR codes [179]).

[5] https://play.google.com/store/apps/details?id=com.pas.webcam&hl=de

The CADS prototype implementation can detect and react on the threat imposed by such an app by collecting and analyzing Features from numerous Feature Collectors. More precisely, the following aspects are checked.

- It is detected if an app that requests a suspicious set of permissions is present on a smartphone. Suspicious in this example refers to a combination of permissions that allows the app to capture video data and to transmit it to a remote location.
- It is detected whether the camera of the smartphone is currently used or not.
- It is detected if the smartphone itself accepts incoming IP connections on any ports.
- The amount of outgoing traffic that originates from the smartphone is monitored. It is expected that the transmission of captured sensor data will increase the amount of outgoing traffic to an abnormal level.

If all of the above aspects are fulfilled by a smartphone, it is assumed that a sensor sniffing attack is going on. As a consequence, an enforcement should be employed that blocks any incoming or outgoing traffic from the smartphone.

6.4.2 Evaluation Environment

The detection will be performed in a virtualized environment. The basic layout is depicted in Figure 6.13. It is a simplified version of the reference IT infrastructure defined in Chapter 2. Virtualization is performed by leveraging VirtualBox using OS X 10.8.2 as host operating system and Ubuntu 12.04.1 as guest operation system. As a smartphone, the Samsung Galaxy S3 with Android 4.1.1 is used. It is connected to the virtualized environment via a Lancom L-54 g wireless access point. The Feature Collector for Android is installed on the smartphone.

The network topology is composed of two subnets. The first one (External 10.0.0.0/24) is used by the smartphone in order to establish a connection to the virtualized environment via the wireless access point. The second one (Internal 192.168.2.0/24) is primarily used by the CADS software components in order to communicate with each other. The environment is composed of three virtual machines (VM):

Router VM The VM is responsible for routing between the external and the internal network. Furthermore, it hosts an iptables packet filter and the ironmonitor Feature

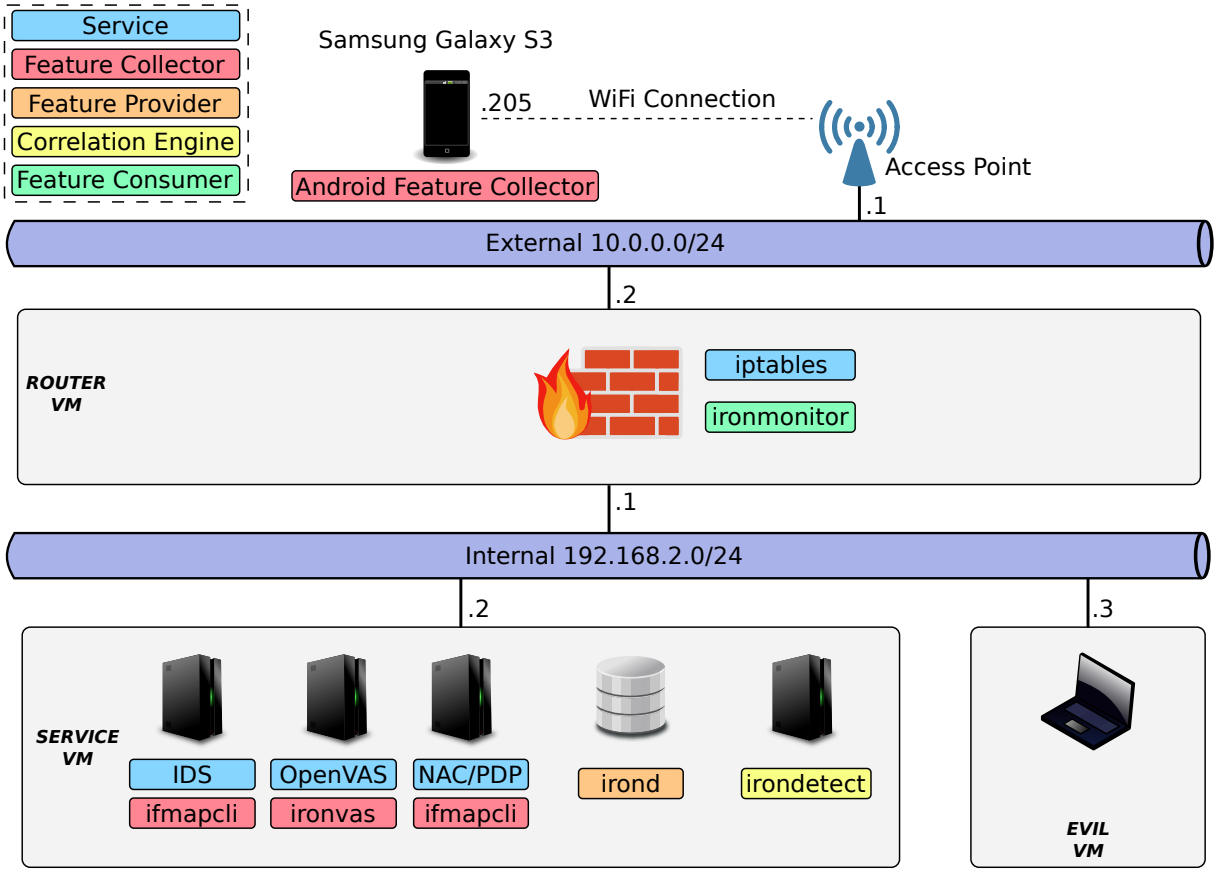

Figure 6.13: Evaluation environment for detecting sensor sniffing attacks. The red, orange, yellow and green colored boxes indicate software components that fulfill logical roles as defined by the CADS architecture. The blue colored boxes indicate services that have been integrated either by leveraging ironvas, ironmonitor or the ifmapcli command line tools. Icons taken from Openclipart [27].

Consumer. By default, the packet filter is configured to allow any traffic between the two networks. ironmonitor will be used to react on the results of the Correlation Engine. More precisely, when a policy violation is detected (in this case the sensor sniffing), ironmonitor will change the configuration of the packet filter in order to block traffic that originates from or is targeted at the smartphone's IP address.

Service VM The service VM hosts all of the CADS software components that are used for the evaluation. This includes the MAP server "irond" which acts as Feature Provider, the Correlation Engine "irondetect" and three Feature Collectors. One of them actually integrates into an existing security service: ironvas retrieves the results of the latest vulnerability scans from OpenVAS and transmits them to the Feature Provider. The other two are simulated based on the ifmapcli command line tool. First, ifmapcli is used to simulate a PDP that is part of a Network Access Control solution which supports the Trusted Network Connect standards, especially the IF-MAP protocol. That is, for each connected smartphone, it publishes a set of standard metadata to the MAP server. The second simulated component represents an intrusion detection system like Snort[6]. It will be used in the remainder to publish Features that represent events generated by an IDS to the MAP server. The service VM is connected to the internal network.

Evil VM The third VM is used in order to represent a malicious component that aims to retrieve sensor data fro the smartphone. That is, this component will be responsible for actually receiving the captured sensor data from the smartphone. It is also connected to the internal network. This topology was chosen in order to allow an easy exemplary enforcement based on the iptables packet filter.

The exact versions of the CADS software components that were used for evaluation are listed in Table 6.3.

It should be noted that although a specific network topology was chosen for the evaluation, this does not affect the general applicability of the CADS approach. More precisely, it is only assumed that there is a separation in two different zones (internal and external) for the evaluation. This topology is basically in accordance with the topology of the reference IT infrastructure that was introduced in Section 2.1, with two minor exceptions:

[6] http://www.snort.org/

Table 6.3: CADS software components of the virtualized evaluation environment.

Software Component	Version
irond	0.3.4
irondetect	0.0.1
ironvas	0.1.0
ifmapcli	0.0.2
irondmonitor	0.0.1

1. The evaluation environment does only provide two subnets (external and internal). There is no dedicated subnet for the DMZ. An additional subnet could easily be added if necessary. However, this would not introduce any benefits concerning the detection of sensor sniffing attacks.

2. The evaluation environment is limited to smartphones that are connected via WAPs to the external zone. Access via VPNs is not supported. However, this could be easily added by installing an appropriate VPN solution.

At a conceptual level, these network topology details do not affect the detection capabilities of the CADS approach. Even if a VPN is used and smartphones connect remotely to the IT infrastructure, the detection of sensor sniffing attacks would still be possible. Even more, the enforcement could be done by leveraging the VPN gateway instead of the iptables packet filter.

The only, general requirement that must be met is that Feature Collectors which are distributed within the IT infrastructure are able to collect their respective Features and to transmit them to the Feature Provider.

6.4.3 OpenVAS Vulnerability Scans of Android Smartphones

The vulnerability scanner OpenVAS is expected to contribute Features that help to detect the sensor sniffing attack. In order to exactly determine which Features are suitable for detection (and should thus be included in the Correlation Engine's policy file), a set of vulnerability scans have been performed in advance for three different Android smartphones. For each smartphone, two tests have been performed: one directly after the smartphone has been booted (and no apps have been started yet) and a second where the respective

IP webcam app is running. The relevant configuration parameters of OpenVAS are given in Table 6.4:

- The version of the OpenVAS scan engine that was used (in this case version 4.0.6)
- The version of the Network Vulnerability Tests (NVTs) that were used. NVTs are modules that perform the actual vulnerability tests. They can be updated independently from the OpenVAS scan engine. Updates are obtained via the publicly available OpenVAS NVT feed [7].
- The scan config that was used. It determines which of the available NVTs are actually executed during a certain scan. In this case, the pre-configured scan config "full and fast" was chosen. For the respective version of OpenVAS, this resulted in 28843 NVTs to be executed for each vulnerability scan that is run. None of the NVTs were specifically targeted to detect vulnerabilities on Android smartphones.
- A target that identifies the smartphone to be scanned. For this purpose, the IP address of the smartphones and the port range that should be scanned must be specified. Optionally, credentials can be provided in order to log into the target and execute NVTs that require host-based access. The protocol that is used to log into the target machine depends on the concrete NVT implementation. The tested smartphones did not provide any service for remote login (such as telnet or ssh). Thus, no login credentials were specified.

The results of the vulnerability scans are depicted in Table 6.5. First of all it is obvious that the Android version that is used has no effect on the the scan results. This is likely due to the fact that no host-based NVTs are executed since no login credentials have been provided during the definition of the scan target.

The second point that is obvious is that there are more vulnerabilities found when the IP webcam app is running. If the app is not running, only 9 vulnerabilities of the threat level "log" are found. These are actually no vulnerabilities that impose a threat at all. They purely serve informational purposes. For example, they state that the target responded to Internet Control Message Protocol (ICMP) requests and that there were no open ports found on the target.

[7] http://www.openvas.org/openvas-nvt-feed.html

Table 6.4: OpenVAS configuration details.

Configuration Parameter	**Value**
OpenVAS Scanner Version	4.0.6
NVT Feed Version	201211231334
Scan Config	Full and fast
Target	IP address of smartphone, default port range, no login credentials were provided

Table 6.5: Vulnerabilities found by OpenVAS on smartphones with stock Android versions. "App Running" refers to whether the IP webcam app was actually started and ready to stream data or not.

Model	**Version**	**App Running**	**Vulnerabilities**			
			High	Medium	Low	Log
Samsung Galaxy S 1	2.3.3	no	0	0	0	9
Samsung Galaxy S 1	2.3.3	yes	0	1	4	14
Samsung Galaxy S 3	4.1.1	no	0	0	0	9
Samsung Galaxy S 3	4.1.1	yes	0	1	4	14
Samsung Galaxy Nexus	4.2.1	no	0	0	0	9
Samsung Galaxy Nexus	4.2.1	yes	0	1	4	14

When the IP webcam app is running, more vulnerabilities are detected. The overall threat level of the respective smartphone rises from "none" to "medium". This is mainly because the app implements a simple web server that accepts connections on port 8080. This is detected by OpenVAS. Since an open port is detected, further NVTs are executed that aim to investigate the vulnerabilities imposed by the provided service. The leads to a number of vulnerabilities with the threat level "low". The single vulnerability that caused the threat level to rise to "medium" refers to the fact that the provided service is prone to denial of service attacks.

For the detection of a sensor sniffing attack as described in this scenario, the main contribution that is provided by the OpenVAS vulnerability scans is the fact that open ports on a smartphone are detected. Actual vulnerabilities of services (such as the fact that the IP webcam app is prone to denial of service attacks) are not considered. The NVT that is responsible for detecting open ports and provided services is named "Services". Thus,

the Feature Collector “ironvas” is configured to publish Feature metadata that represents vulnerability reports that were created by the “Services” NVT[8].

6.4.4 Policy Definition

The definition of a Policy that is able to detect the described sensor sniffing is rather complex. There are many options to actually craft the Rules based on Signatures, Anomalies and Context definitions. The details of the Policy that was used for this example is depicted in Listing 6.1. It is basically an extended version of the Policy described as part of the domain-specific mapping in Section 4.4. Furthermore, it strictly follows the grammar that is used by the Correlation Engine in order to verify its syntax.

```
# Context definitions
context {
  ctxLastMinutes := SLIDING = "00:15:00" ;
  ctxWorkingHours := DATETIME > "06:00" and DATETIME < "20:00";
}
# Hint definitions
hint {
  hintTrafficSmartphone := "smartphone.communication.ip.txother"
    "de.fhhannover.inform.trust.irondetectprocedures.TrendByValueCW"
    "100";
  hintTrafficIds := "ids.event.hightraffic"
    "de.fhhannover.inform.trust.irondetectprocedures.TrendByValueCW"
    "100";
}
# Anomaly definitions
anomaly {
  anoHighTrafficSmartphone := hintTrafficSmartphone > 0.5 and hintTrafficIds > 0.5;
}
# Signature definitions
signature {
  sigCamera := "smartphone.sensor.camera.isused" = "true" ctxWorkingHours;
  sigSuspiciousApp := "smartphone.android.app.permission.requested!1" = "android.
      permission.RECEIVE_BOOT_COMPLETED"
    and "smartphone.android.app.permission.requested!1" = "android.permission.CAMERA"
    and "smartphone.android.app.permission.requested!1" = "android.permission.INTERNET"
        ctxWorkingHours;
  sigPortOpen := "vulnerability.name" = "Services"
  ctxLastMinutes;
}
# Condition definitions
condition {
```

[8]The respective Features were already listed in Table 5.2 in the previous chapter.

```
30   conDataLeakDetected := sigSuspiciousApp and sigCamera and sigPortOpen and
         anoHighTrafficSmartphone;
31 }
32 # Action definitions
33 action {
34   enforcementIsolate :=
35     "correlationresult.enforcement.enforcementaction" "./drop-client.sh"
36     "correlationresult.enforcement.$1" "@smartphone.system.ipaddress";
37 }
38 # Rule definitions
39 rule {
40   dataLeakage := if conDataLeakDetected do enforcementIsolate;
41 }
```

Listing 6.1: Example Policy for the detection of sensor sniffing attacks.

Contexts

For the evaluation, two Context definitions are used. Both make use of the temporal Context Parameter *Timestamp* that renders the moment in time when a Feature has been measured[9]. The two Contexts allow to restrict the set of Feature instances that are considered for the evaluation. In the example, a sliding Context is defined that only matches Features who have been measured in the past 15 minutes. The second Context is used to consider only Features who have been measured during working hours. Further Contexts could have been defined that work on other Context Parameters (like the location or the presence of other devices). However, this has been omitted in order to keep the example simple. It is important to note that Contexts are solely used to restrict the set of Features that are considered during the evaluation of a Policy. However, they do not change the way the evaluation is performed once the set of relevant Feature instances has been determined.

Signatures

Signatures are used in order to express the static characteristics of the sensor sniffing attack. For this example, there are three Signatures used. The first one checks if the camera of the smartphone is currently in use. The second one searches for apps that have a suspicious set of requested permissions. Note that the scope parameter has been set to

[9]In the irondetect version that has been used for evaluation, the Context Parameter is referred to as DATETIME in the policy language.

"1" for this Signature[10]. The third Signature finally checks if there are any open ports reported by the OpenVAS vulnerability scanner. This Signature makes use of a sliding Context. That is, only Features that have been measured in the past 15 minutes are considered. Furthermore, the Signature is reevaluated every 15 minutes, even when there were no new Features received by the Correlation Engine. This mechanism can be used to ensure that only Features who have a certain "freshness" are used during the evaluation of a Policy. To set the interval of a sliding Context to a reasonable value is a domain-specific decision.

Anomalies

A single Anomaly component is defined in order to analyze the amount of data that is transmitted by the smartphone. It is composed of two Hints. Each of them references a single Feature that encapsulates outgoing traffic (*TxOther* and *HighTraffic*). The first Hint is used in order to evaluate Features measured on the smartphone, whereas the second one works on Features that have been measured by the simulated IDS. Both Hints make use of the same Procedure called "TrendByValueCW". As already mentioned in Section 5.2.4, this procedure performs a simple linear regression [172] on the Feature values and determines the slope of the fitting line. The Procedure always considers the last 10 Feature instances that have been measured. The calculated slope is compared to the trained value. If no training was performed, it is compared against the value that is given in the Policy as the last parameter of the Hint definition. For this example, no training was performed. Instead, the expected value was set a priori based on the findings discussed in Section 6.1.2. More precisely, the expected value was set to "100" for both Hints, which corresponds to 100 KB/s of traffic given the Features and the Procedure that are used by the Hints. Any smartphone that exceeds this rate is considered to behave abnormal in this respect. The output mapping was configured as follows:

- If the slope exceeds the expected value by more than 50%, return 0,
- if the slope exceeds the expected value by more than 100%, return 1,
- else return -1.

[10] The implementation of the Policy parser currently requires to define the scope for each Feature that is used in the Signature as follows: FeatureId!<scope>

This allows to detect if a smartphone transmits data via its WiFi interface at an alarming rate. The Hints are combined by a logical "and". That is, only if both the Features collected on the smartphone and those that are collected by the IDS indicate the presence of an abnormal behavior, the respective Anomaly is fulfilled.

Note that each of the Hints might as well consider more than one single type of Features. For example, in order to also capture data that is transmitted by the smartphone via its mobile network interface, the Hint could have been defined to also evaluate the Feature (*smartphone.communication.ip.Tx3g*). Another option is to define another Hint for the respective Feature and to integrate it in the definition of the Anomaly. This level of opportunities to chose from emphasizes that the definition of reasonable Policies is a complex and challenging task.

Rules and Actions

The previously defined components are combined within a single Rule. If the three Signatures and the Anomaly match, the associated Action is performed. In this case, this triggers the creation of two Features:

- EnforcementAction: The value is set to the name of a Shell script.
- $1: The value is set to a parameter that is passed to the Shell script. In this case, this is the IP address of the smartphone the Policy is currently evaluated for[11].

These two Features are intended to be processed by the Feature Consumer ironmonitor.

The presented Policy allows to detect sensor sniffing attacks by combining both Signature and Anomaly detection mechanisms. It is important to note that although a special app is used within the evaluation to perform the sensor sniffing (IP webcam app), the Policy and thus the detection capabilities are not specific for this single app. More precisely, any sensor sniffing attack that shares the same characteristics can be detected as well.

6.4.5 Interaction of CADS Software Components

The interaction between the CADS software components that take place in order to detect the sensor sniffing attack is rather complex. In order to understand how the individual

[11]The fact that the Correlation Engine should look up the smartphones IP address is denoted by the '@'.

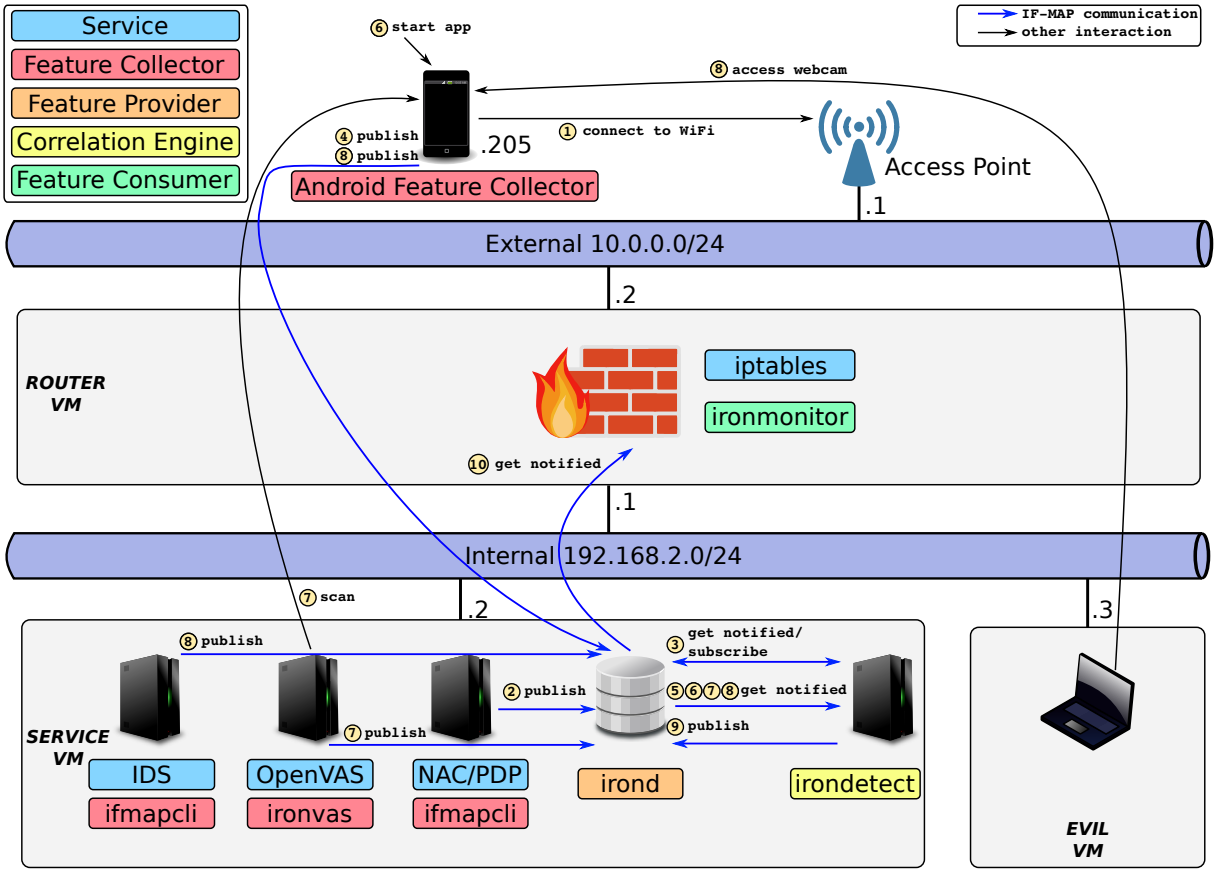

Figure 6.14: Interaction between software components in the sensor sniffing example. Icons taken from Openclipart [27].

systems work together, their interaction is detailed in the following. It is assumed that all components are properly configured. The Feature Provider must be running. The Correlation Engine and the Feature Consumer have issued their initial subscriptions in order to retrieve new Feature metadata when it is published to the Feature Provider.

The single steps that are performed and described in the following are depicted in Figure 6.14.

1. At first, the smartphone establishes a connection to the WAP, and thus to the virtualized environment.

2. The PDP (simulated by ifmapcli) publishes a set of standard metadata to the Feature Provider. This especially includes `access-request-device` and `authenti-`

`cated-by` metadata which links the `device` identifier of the smartphone to the `device` identifier of the PDP.

3. The Correlation Engine gets notified about the new smartphone. It issues a new subscription with the smartphone's `device` identifier as root in order to get notified when new feature metadata is published for the smartphone.

4. The Feature Collector on the smartphone is started. It periodically publishes Feature metadata to the Feature Provider. For the evaluation, an update interval of 30 seconds was chosen.

5. The Correlation Engine is notified about the new Feature metadata published by the smartphone. Thus, it evaluates the Policy. In the example, the first Signature of the Condition matches as the IP webcam app requests the respective set of permissions. However, as the camera of the smartphone is currently not used, the second Signature does not match and evaluates to false. The version of irondetect that was used in this example stops the evaluation of a Condition that only has logical "and" conjunctions upon the first Signature or Anomaly that is false. Thus, the results of the third Signature and the Anomaly are not considered at that moment.

6. In the next step, the respective IP webcam app is started. However, the video stream that it provides is not yet accessed by the evil VM. After the respective Feature metadata has been transmitted to the Correlation Engine, the Signature which checks if the camera is currently used evaluates to "true". However, the third Signature which expresses whether open ports have been found or not does not match yet. This is because the respective Feature Collector (ironvas) has not yet published the Feature metadata which renders the results of the last vulnerability scan performed for the smartphone.

7. In this step, OpenVAS scans the smartphone and ironvas publishes the respective Feature metadata that render the results of the last vulnerability scan. Within the evaluation environment, both OpenVAS and ironvas were configured to perform their tasks (executing vulnerability scans for smartphones respectively transmitting the results of the last scans to the Feature Provider) at a fixed interval. As the IP

webcam app is now running, OpenVAS detects the open port and ironvas transmits this information rendered in appropriate Feature metadata to the Feature Provider. Thus, the third Signature matches as well. Note that the third Signature uses a sliding Context. This causes that only Features which have been measured in the last 15 minutes are considered and that the Signature (and thus the Condition) is reevaluated every five minutes. Although all Signatures are now fulfilled, the traffic that is transmitted by the smartphone itself is still considered as being normal. This is because the web server which is provided by the app is not accessed by the evil VM yet.

8. In the next step, a web browser from within the evil VM is used to access the video stream provided by the IP webcam app. This causes a drastically increase of the outgoing traffic. In the evaluation environment, two Features are used to express this increase: the first one is actually measured by the Feature Collector on the smartphone (*smartphone.communication.ip.TxOther*). The second one is created on behalf of the simulated IDS. That is, it is not actually measured but scripted by leveraging the ifmapcli command line tools. It simply publishes the Feature *ids.event.Hightraffic* periodically with a value that expresses the amount of data that has been sent by the smartphone. The main purpose of integrating a simulated IDS in the evaluation environment is to emphasize the capability of the CADS approach to detect Anomalies based on Features that are collected by different Feature Collectors.

   The effect that the streaming of video data has in terms of transmitted data is depicted in Figure 6.15. As already mentioned, the slope has been calculated by performing a simple linear regression based on the last 10 Features that have been measured. The results are depicted for the Feature that has been directly measured on the smartphone (*TxOther*). Each calculated value is marked with a red cross. The moments in time when the video steam was accessed are labeled as 'start', the moments when access was stopped are labeled as 'stop'. The first access happened after three minutes. Prior to this moment, the amount of outgoing traffic is negligible. However, when the video stream is accessed, the amount transmitted data increases. It exceeds the expected/trained value at $t = 338$ s. The configured tolerance is exceeded at $t = 405$ s. At this moment, the result of the respective

Hint evaluation is 0. Due to the definition of the Anomaly component in the Policy which requires a Hint result greater than 0.5, the outgoing traffic is still considered as being normal.

At $t = 472$ s, the slope exceeds the threshold the first time. At this moment, the respective Hint result is 1.0. As only the last 10 Features are considered for calculating the slope, its value drops and increases accordingly depending on whether the evil VM stops to access the video stream (at $t = 539$ s and $t = 942$ s) or starts to use it again ($t = 640$ s).

9. When both of the defined Hints return a result of 1.0, the respective Anomaly evaluates to "true". Thus, the Condition of the defined Rule evaluates to "true" as well. As a consequence, the Correlation Engine itself creates two new Features and transmits them to the Feature Provider. In this example, the Features render a command that should be executed in order to react on the identified threat.

10. The Features that were created by the Correlation Engine are received by the Feature Consumer located in the router VM. It processes them and changes the configuration of the iptables packet filter in such a way that the evil VM cannot access the video stream anymore. As the IP address of the smartphone was encapsulated within one of the Features ($1), it is possible to add a rule to the iptables packet filter that explicitly blocks the traffic for that particular smartphone. Of course, it should be ensured that the Feature Collector on the smartphone is still able to transmit its measured Features to the Feature Provider.

A screenshot of the irondetect graphical user interface that summarizes the single detection steps is depicted in Figure 6.16. It displays the evaluation results for each Rule (upper left), Condition (lower right), Signature (upper right) and Anomaly (lower left). The first evaluation was done at 09:35:36. At that time, only one single Signature was fulfilled (*sigSuspiciousApp*). After the IP webcam app was started, the Signature *sigCamera* was fulfilled as well. However, the Signature that indicates an open port was not fulfilled at that moment. The reason is that the respective Feature Collector (ironvas) had not yet published its Feature metadata. That is, there is a time gap between the actual opening of a port caused by the IP webcam app and its detection by the vulnerability scanner. However, once the vulnerability scanner has detected the open ports, the respective Features

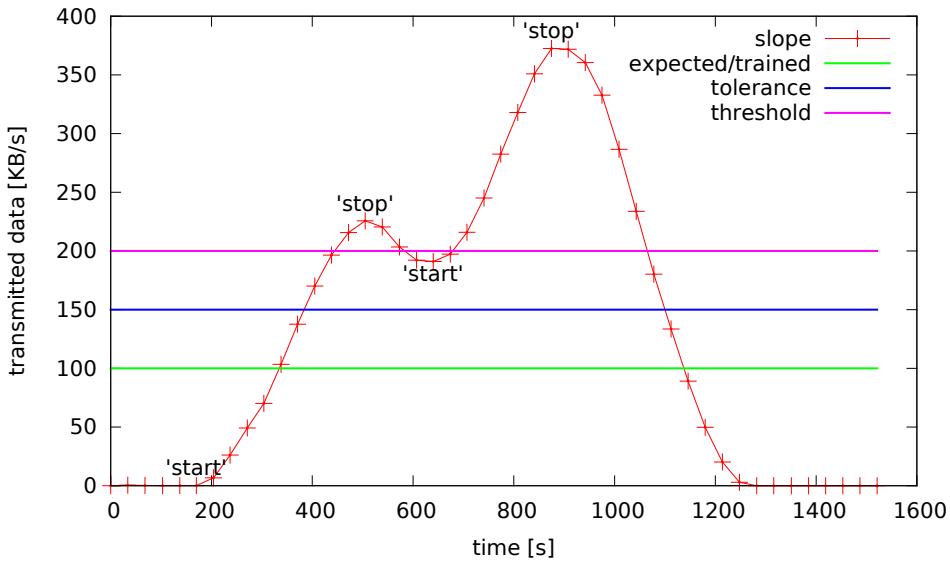

Figure 6.15: Traffic anomaly caused by accessing the IP webcam app. Depicted are the results for performing a simple linear regression on the measured values of the Feature *TxOther*.

are published by ironvas. The Features are received by the Correlation Engine, which in turn evaluates the Policy again (steps labeled as #6, #7, #8). Once all Signatures are fulfilled, the Anomaly is evaluated as well. In the depicted example, the webcam server was accessed after the Anomaly was evaluated the second time (labeled as #7). After three minutes, the amount of transmitted data exceeded the configured threshold. Since all Signatures and the Anomaly were fulfilled at that time, the Condition evaluates to "true", and thus the respective Rule fires.

6.4.6 Results

The previous section described how a sensor sniffing attack can be detected and mitigated with the CADS approach by analyzing Features that were gathered from multiple Feature Collectors. A Policy for the Correlation Engine was defined that makes use of Signature and Anomaly components for detection. The attack was illustrated by leveraging a benign app from the official Google Play store, thus rendering any approach that solely relies on detecting malicious apps inappropriate. It is important to note that although the example was illustrated by using a certain app (IP webcam), the detection would work for any other

irondetect - Rules

#	Device	ID	Value	Timestamp
10	af0f	dataLeakage	☑	2012-12-13T09:37:02+01:00
9	af0f	dataLeakage	☐	2012-12-13T09:36:53+01:00
8	af0f	dataLeakage	☐	2012-12-13T09:36:42+01:00
7	af0f	dataLeakage	☐	2012-12-13T09:36:32+01:00
6	af0f	dataLeakage	☐	2012-12-13T09:36:29+01:00
5	af0f	dataLeakage	☐	2012-12-13T09:36:21+01:00
4	af0f	dataLeakage	☐	2012-12-13T09:36:11+01:00
3	af0f	dataLeakage	☐	2012-12-13T09:36:06+01:00
2	af0f	dataLeakage	☐	2012-12-13T09:35:56+01:00
1	af0f	dataLeakage	☐	2012-12-13T09:35:36+01:00

irondetect - Signatures

#	Device	ID	Value	Timestamp
29	af0f	sigPortOpen	☑	2012-12-13T09:37:02+01:00
28	af0f	sigCamera	☑	2012-12-13T09:37:02+01:00
27	af0f	sigSuspiciousApp	☑	2012-12-13T09:37:02+01:00
26	af0f	sigPortOpen	☑	2012-12-13T09:36:53+01:00
25	af0f	sigCamera	☑	2012-12-13T09:36:53+01:00
24	af0f	sigSuspiciousApp	☑	2012-12-13T09:36:53+01:00
23	af0f	sigPortOpen	☑	2012-12-13T09:36:42+01:00
22	af0f	sigCamera	☑	2012-12-13T09:36:42+01:00
21	af0f	sigSuspiciousApp	☑	2012-12-13T09:36:42+01:00
20	af0f	sigPortOpen	☑	2012-12-13T09:36:32+01:00
19	af0f	sigCamera	☑	2012-12-13T09:36:32+01:00
18	af0f	sigSuspiciousApp	☑	2012-12-13T09:36:32+01:00
17	af0f	sigPortOpen	☑	2012-12-13T09:36:29+01:00
16	af0f	sigCamera	☑	2012-12-13T09:36:29+01:00
15	af0f	sigSuspiciousApp	☑	2012-12-13T09:36:29+01:00
14	af0f	sigPortOpen	☑	2012-12-13T09:36:21+01:00
13	af0f	sigCamera	☑	2012-12-13T09:36:21+01:00
12	af0f	sigSuspiciousApp	☑	2012-12-13T09:36:21+01:00
11	af0f	sigPortOpen	☑	2012-12-13T09:36:11+01:00
10	af0f	sigCamera	☑	2012-12-13T09:36:11+01:00
9	af0f	sigSuspiciousApp	☑	2012-12-13T09:36:10+01:00
8	af0f	sigPortOpen	☑	2012-12-13T09:36:06+01:00
7	af0f	sigCamera	☑	2012-12-13T09:36:06+01:00
6	af0f	sigSuspiciousApp	☑	2012-12-13T09:36:06+01:00
5	af0f	sigPortOpen	☐	2012-12-13T09:35:56+01:00
4	af0f	sigCamera	☑	2012-12-13T09:35:56+01:00
3	af0f	sigSuspiciousApp	☑	2012-12-13T09:35:56+01:00
2	af0f	sigCamera	☐	2012-12-13T09:35:36+01:00
1	af0f	sigSuspiciousApp	☑	2012-12-13T09:35:36+01:00

irondetect - Anomalies

#	Device	ID	Value	Timestamp
8	af0f	anoHighTrafficSmartphone	☑	2012-12-13T09:37:02+01:00
7	af0f	anoHighTrafficSmartphone	☐	2012-12-13T09:36:53+01:00
6	af0f	anoHighTrafficSmartphone	☐	2012-12-13T09:36:42+01:00
5	af0f	anoHighTrafficSmartphone	☐	2012-12-13T09:36:32+01:00
4	af0f	anoHighTrafficSmartphone	☐	2012-12-13T09:36:29+01:00
3	af0f	anoHighTrafficSmartphone	☐	2012-12-13T09:36:21+01:00
2	af0f	anoHighTrafficSmartphone	☐	2012-12-13T09:36:11+01:00
1	af0f	anoHighTrafficSmartphone	☐	2012-12-13T09:36:06+01:00

irondetect - Conditions

#	Device	ID	Value	Timestamp
10	af0f	conDataLeakDetected	☑	2012-12-13T09:37:02+01:00
9	af0f	conDataLeakDetected	☐	2012-12-13T09:36:53+01:00
8	af0f	conDataLeakDetected	☐	2012-12-13T09:36:42+01:00
7	af0f	conDataLeakDetected	☐	2012-12-13T09:36:32+01:00
6	af0f	conDataLeakDetected	☐	2012-12-13T09:36:29+01:00
5	af0f	conDataLeakDetected	☐	2012-12-13T09:36:21+01:00
4	af0f	conDataLeakDetected	☐	2012-12-13T09:36:11+01:00
3	af0f	conDataLeakDetected	☐	2012-12-13T09:36:06+01:00
2	af0f	conDataLeakDetected	☐	2012-12-13T09:35:56+01:00
1	af0f	conDataLeakDetected	☐	2012-12-13T09:35:36+01:00

Figure 6.16: Screenshot of irondetect after the detection of a sensor sniffing attack. Checkmarks indicate that the respective component defined by the Policy evaluated to “true”.

app or combination of apps as long as the performed attack shares the same characteristics. That is

- there must be an app that has a set of suspicious permissions,
- the camera must be used to obtain the video data,
- the captured data must be provided by a web server on the smartphone and
- retrieving the data from the smartphone must increase its outgoing traffic.

Of course, apps can be developed that are still able to perform a sensor sniffing attack and will not be detected by the example Policy. It is a subject of future work to develop further domain-specific Policies in order to detect more variants of sensor sniffing attacks.

6.5 Using CADS to Mimic Kirin

In order to prove the versatility of the CADS approach, this section demonstrates how it can be used to provide a similar functionality as an existing, host-based security service for Android.

6.5.1 Overview

The Kirin security service for Android [129, 14] was already discussed in Section 3.3.3 as part of the analysis of related work in the field of smartphone security. It provides an extension to the Android platform that checks security properties of apps at installation time against a blacklist defined in a specific policy. Kirin supports to classify an app as being malicious or not based on two kinds of static properties: (1) the app's requested permissions and (2) the app's ability to receive Intents with a certain action string. If one of the rules defined in the policy matches, the user is warned that the app which is about to be installed is potentially malicious. He can then choose to abort or to continue the installation. Kirin is a host-based approach, thus requires custom modifications of the Android platform. The CADS approach is able to provide similar functionality as the Kirin security service, while providing a number of benefits (like omitting the need for modifications of the Android platform).

6.5.2 Translating Kirin Policies to CADS

Enck et al. [129, 14] provide a sample policy that is composed of nine rules. It is depicted in Listing 6.2. The first eight rules simply express a combination of requested permissions that are considered as being malicious. For example, rules 4 and 5 aim to detect location tracker apps that leak the location of the smartphone to a remote party. Rule number 9 is special as it is not only formulated based on requested permissions. Instead, it combines a permission label with the fact that an app can receive and process a certain Intent. It aims to prevent a malicious app from replacing the default voice call dialer app without the user's knowledge.

```
1 (1) An application must not have the SET_DEBUG_APP permission label.
2 (2) An application must not have PHONE_STATE, RECORD_AUDIO, and INTERNET permission
      labels.
3 (3) An application must not have PROCESS_OUTGOING_CALL, RECORD_AUDIO, and INTERNET
      permission labels.
4 (4) An application must not have ACCESS_FINE_LOCATION, INTERNET, and
      RECEIVE_BOOT_COMPLETE permission labels.
5 (5) An application must not have ACCESS_COARSE_LOCATION, INTERNET, and
      RECEIVE_BOOT_COMPLETE permission labels.
6 (6) An application must not have RECEIVE_SMS and WRITE_SMS permission labels.
7 (7) An application must not have SEND_SMS and WRITE_SMS permission labels.
8 (8) An application must not have INSTALL_SHORTCUT and UNINSTALL_SHORTCUT permission
      labels.
9 (9) An application must not have the SET_PREFERRED_APPLICATION permission label and
      receive Intents for the CALL action string.
```

Listing 6.2: Example policy for Kirin in KSL syntax [14].

In the following, the example policy for Kirin will be translated to the policy language used by the CADS approach. Kirin is limited to express malicious patterns based on an app's requested permissions and its Intent Filters. Thus, the rules of a Kirin policy can easily be mapped to Signature definitions within the CADS policy language. The only requirement that must be met is that appropriate Features have been defined in order to render requested permissions and action strings of Intent Filters. The Feature which represents a permission that is requested by an app has already been used in the previous example: *smartphone.android.app.permission.Requested*. The Feature that represents the action string of an Intent Filter is referred to as *smartphone.android.app.intentfilter.action.ActionString*. Given these two Features, the example policy for the Kirin system can be expressed by a set of Signature definitions in the CADS policy language. They are depicted in List-

ing 6.3. The resulting policy can be evaluated by the Correlation Engine. The Feature Collector for Android is responsible for measuring the necessary Features.

```
# signature definitions
# the categories smartphone.android.app are omitted for readability
# the prefix "android.permission" for permission labels is omitted for readability
signature {
  sigKirinRuleOne := "permission.requested!1" = "SET_DEBUG_APP";
  sigKirinRuleTwo := "permission.requested!1" = "PHONE_STATE" and
    "permission.requested!1" = "RECORD_AUDIO" and
    "permission.requested!1" = "INTERNET";
  sigKirinRuleThree := "permission.requested!1" = "PROCESS_OUTGOING_CALL" and
    "permission.requested!1" = "RECORD_AUDIO" and
    "permission.requested!1" = "INTERNET";
  sigKirinRuleFour := "permission.requested!1" = "ACCESS_FINE_LOCATION" and
    "permission.requested!1" = "INTERNET" and
    "permission.requested!1" = "RECEIVE_BOOT_COMPLETE";
  sigKirinRuleFive := "permission.requested!1" = "ACCESS_COARSE_LOCATION" and
    "permission.requested!1" = "INTERNET" and
    "permission.requested!1" = "RECEIVE_BOOT_COMPLETE";
  sigKirinRuleSix := "permission.requested!1" = "RECEIVE_SMS" and
    "permission.requested!1" = "WRITE_SMS";
  sigKirinRuleSeven := "permission.requested!1" = "SEND_SMS" and
    "permission.requested!1" = "WRITE_SMS";
  sigKirinRuleEight := "permission.requested!1" = "INSTALL_SHORTCUT" and
    "permission.requested!1" = "UNINSTALL_SHORTCUT";
  sigKirinRuleNine := "permission.requested!1" = "SET_PREFERRED_APPLICATION" and
    "intentfilter.action.ActionString!1" = "CALL";
}
```

Listing 6.3: Kirin example policy translated to the CADS policy language. Only the use of Signatures is mandatory for the translation.

6.5.3 Discussion

Kirin classifies apps as being malicious or not solely based on two aspects of an app: its requested permissions and the action strings that are defined as part of its Intent Filter. To realize the same classification of apps with the CADS approach is possible. However, there are differences that need to be considered:

Host- vs. Network-based Approach CADS is a lightweight, network-based approach. This introduces both a benefit and a drawback. The drawback is that CADS cannot prevent the installation of potentially malicious apps. It does not hook itself into the installation process of apps as Kirin does. Instead, an ordinary app is responsible for

measuring the necessary Features, which in turn can be evaluated by the Correlation Engine. The benefit is that CADS can be used on smartphones that run standard Android versions. The only requirement is that the Feature Collector for Android is installed on the device.

Detection Capabilities The Kirin approach is limited to classify an app as being malicious or benign based on their requested permissions and the action strings defined in their Intent Filter. That is, Kirin policies define patterns based on permissions and action strings that are considered to be malicious. As a consequence, Kirin policies that are translated to the CADS policy language only make use of Signatures for classifying apps. However, CADS allows the definition of further components, including Contexts and Anomalies. They can be used to enrich the basic detection capabilities that are provided by the Kirin service. More precisely, CADS allows (1) to consider other Features than the two that represent requested permissions and action strings of Intent Filters, (2) to integrate the Context of a smartphone and (3) to reason about the behavior of a smartphone by means of defining Anomaly components. That these extended capabilities can actually provide a benefit was proven in the previous Section during the detection of a sensor sniffing attack. Using Kirin, it would only have been possible to check if an app is going to be installed that has a set of suspicious permissions. However, to consider the status of the smartphone's camera, the presence of vulnerability reports or the amount of outgoing traffic at the moment when the smartphone is actually used within the infrastructure of a company would not have been possible.

Reaction Capabilities Kirin informs the user about any app that is considered as being malicious at install-time. The user can choose to abort or to continue the installation. This is not possible with the CADS approach. Instead, it allows to define arbitrary Actions as part of the Correlation Engine's Policy. In order to provide a similar functionality as Kirin does, an additional Feature Consumer could be placed on the smartphone in order to retrieve the results of the Correlation Engine. This Feature Consumer can in turn show appropriate warning messages to the user, suggesting that a certain app should be uninstalled because it was classified as malicious. However, it is important to note that CADS is not limited to this type of reaction. Instead, it is also possible to employ an enforcement as presented in Section 6.4 in

order to deny access to certain parts of the network as long as the malicious app(s) are installed.

To summarize, CADS can be used to provide a similar functionality as the Kirin security service. However, CADS is not limited to classify apps as being malicious or benign based on a fixed set of aspects. It also provides means to reason about the behavior of a smartphone and the Contexts it is currently used in. Furthermore, CADS can be used on smartphones with standard Android versions and does not require any modifications to the smartphone platform itself.

6.6 Summary

This chapter described the evaluation of the CADS approach. An analysis of data that was collected by the FHH device analyzer app over a period of 2.5 months was presented in Section 6.1. The collected data can help to understand how users actually use their smartphones. Furthermore, it can generally be used in order to learn the normal behavior of smartphones. The performance of MAP servers was investigated in Section 6.2. The goal was to investigate if their performance is sufficient to be used within a CADS prototype implementation. For irond, this is true. For omapd however, this is not true. The poor performance of omapd is mainly caused by a rudimentary implementation of handling multiple subscriptions. As irond performs far better than omapd, this is no drawback of the IF-MAP protocol itself. The amount of traffic that is generated by the Feature Collector for Android was analyzed in Section 6.3. The results indicate that after the first IF-MAP publish operation was performed, the generated traffic remains constant and is negligible. Section 6.4 provided an example that proofs the capability of the CADS approach to detect and to react on sensor sniffing attacks. The attack was detected by performing a Context-related detection of Signatures and Anomalies. As reaction, an enforcement was employed that stops the attack by denying outgoing traffic for the respective smartphone. In order to proof the versatility of the CADS approach, Section 6.5 described how it can be used to mimic the functionality of an existing, host-based security extension for Android.

7 Conclusion and Future Work

"I never think of the future. It comes soon enough."

(Albert Einstein)

Contents

Within this thesis, a novel, network-based approach for smartphone security was proposed called **CADS: Context-related Signature and Anomaly Detection for Smartphones**. The main results are summarized in Section 7.1. Based on these results, the research questions that were defined as part of the introduction in Section 1.2 are discussed in Section 7.2. Finally, areas that are subject of future work are mentioned in Section 7.3.

7.1 CADS: A Network-based Approach for Smartphone Security

While developing CADS, the field of smartphone security was tackled from the perspective of a company that aims to securely integrate smartphones into their existing IT infrastructure. Four scenarios were described in Chapter 2 that provide the motivating background for this thesis:

1. Scenario I: Smartphone Visibility, which is about to determine at the network side whether a certain request originates from a smartphone or not,

2. Scenario II: Context-related Service Provisioning, which is about allowing or denying the use of services depending on a smartphone's context,

3. Scenario III: Detection of Malicious and Unwanted Apps, which addresses the threat of third-party apps and

4. Scenario IV: Policy-based Enforcement, which is about to perform policy-based reactions to mitigate detected threats.

Based on these scenarios, seven requirements were derived that must be fulfilled by an approach that aims to securely integrate smartphones into existing IT infrastructures:

1. R-01: Detection of unwanted and malicious configurations of smartphones,

2. R-02: Detection of abnormal smartphone behavior,

3. R-03: Consideration of context information for detection,

4. R-04: Policy-based reaction on detection results,

5. R-05: Dynamic analysis at runtime,

6. R-06: Extensibility of processed data and used methods,

7. R-07: Ability to integrate the approach in existing environments.

An analysis of related work was performed in Chapter 3. The analysis revealed that existing approaches fail to meet all the requirements derived from the scenarios. Especially since most of the existing approaches focus primarily on the detection of malicious apps and follow a host-based approach. As host-based approaches tend to require extensive modifications of the respective smartphone platform, it is hard to integrate them into existing IT infrastructures, especially when companies follow a strategy like Bring Your Own Device (BYOD).

The main contribution of this thesis is the development of **CADS**, a novel, network-based approach for smartphone security that enables **C**ontext-related **A**nomaly and Signature **D**etection for **S**martphones. The concept of the approach was detailed in Chapter 4. It is composed of four parts:

1. A conceptual model that defines its main building blocks and the relationships between them.

2. An architecture that defines logical roles that must be fulfilled by components in order to support the distributed collection and central analysis of data about smartphones.

3. A correlation model that defines how the collected data is analyzed.

4. A process model that defines how so-called domain-specific instances can be specified.

CADS is a novel, network-based approach that allows to reason about the security status of smartphones by analyzing three aspects: their current configuration in terms of installed software and available hardware, their behavior and the context they are currently used in. The necessary data is collected in a distributed manner. CADS is a lightweight approach, which means that it does not rely on modifications that are made to a certain smartphone platform.

An implementation of the approach based on the Android platform and the IF-MAP protocol for network security was presented in Chapter 5. The implementation demonstrates that IF-MAP, although not specifically targeted at the domain of smartphone security, can be used as communication protocol for CADS in order to securely integrate smartphones into existing IT infrastructures. Finally, the CADS approach was evaluated in Chapter 6. It was shown that CADS can be used to detect sensor sniffing attacks, even when they are not performed by a malicious app but rather by leveraging a benign app from an official app store. Furthermore, the flexibility of CADS was demonstrated by mimicking the functionality of Kirin, a host-based security service for the Android platform.

7.2 Discussion of Research Questions

Based on the results that were achieved within this thesis, the research questions that were presented in Section 1.2 are discussed in the following.

Question 1: What approach is appropriate to enable a secure integration of smartphones into existing IT infrastructures? This question was primarily addressed in Chapter 2. Four scenarios from the perspective of a company that aims to securely integrate smartphones into their own IT infrastructure have been described.

- Scenario I: Smartphone Visibility emphasizes that existing services that are provided within an IT infrastructure must be able to determine if they are accessed by a smartphone or not.
- Scenario II: Context-related Service Provisioning extends the first scenario, expressing the need for services to reason about the context of a smartphone that tries to access them.
- Scenario III: Detection of Malicious and Unwanted Apps addresses the fact that third-party apps which are installed on a smartphone can violate the security policy of a company. It is important to note that this scenario is not just limited to apps that are generally considered as being malicious. Instead, even benign apps can violate the security policy of a company under certain circumstances.
- Scenario IV: Policy-based Enforcement expresses the need for a flexible mechanism in order to react on detected policy violations that were caused by smartphones.

The scenarios were defined as part of the analysis phase within the ESUKOM research project. Based on these scenarios, requirements were derived that define the set of functionalities that must be supported by the developed approach, namely:

- To detect unwanted respectively malicious configurations.
- To detect if a smartphone behaves abnormally.
- To consider the context of a smartphone for detection purposes.
- To enable policy-based reactions that are employed based on the detection results.
- To support dynamic analysis at runtime, that is at the time when a smartphone is actually used within an IT infrastructure that should be protected.
- To ensure extensibility, both in terms of what data is considered for detection tasks as well as what methods are actually used for analyzing the data.

- To support a seamless integration of the approach into existing IT infrastructures. This requirement explicitly prohibits that the approach relies on modifications which are made to a specific smartphone platform.

The CADS approach fulfills all of the mentioned requirements at a conceptual level.

Question 2: What data should be collected and how should the collected data be analyzed in order to determine the security status of smartphones? This question is primarily addressed in Chapter 4. Regarding the first part of the question (what data should be collected), the CADS approach introduces the notion of Features and Categories, the so-called Core Components. These components are necessary in order to model relevant data at an abstract level, as a general answer to the question that explicitly names certain types of data cannot be given. Instead, the answer depends on the concrete domain, more precisely the company who aims to integrate smartphones into their IT infrastructure and their security policy. This is taken into account by the CADS approach by defining a process that allows to derive so-called domain-specific instances. These instances model the set of relevant data based on the generic Category and Feature components.

Within the thesis, such an exemplary domain-instance was derived. The set of Categories and Features represent various issues that are of interest in order to determine the security status of smartphones. Of special importance are Features that (1) describe properties of apps that are installed on the respective smartphones (such as their requested permissions) and (2) Features that express how much outgoing and incoming traffic is received respectively transmitted by a smartphone.

The second part of the question (how should the collected data be analyzed) is addressed by the correlation model that was developed as part of the CADS approach. It allows to combine both Signature detection (in order to find patterns based on Features) and Anomaly detection techniques (in order to detect abnormal behavior of smartphones based on Features). Signatures are primarily used to detect malicious or unwanted configurations of smartphones (such as if a suspicious app from an unofficial app store is currently installed). On the other hand, Anomaly detection allows to monitor the behavior of smartphones and to detect deviations that exceed a certain tolerance (such as when a smartphone starts to transmit large amounts of data to a remote location). In

general, the CADS approach is not limited to a certain Anomaly detection technique. The prototype implementation presented in Chapter 5 supports simple statistical methods.

The Features of the exemplary domain-instance and the combination of Signature and Anomaly detection was used to successfully detect a sensor sniffing attack that was employed by leveraging a benign app from the official Google Play app store (as detailed in Chapter 6).

Question 3: How can the context of smartphones be obtained and used in order to contribute to their secure integration into existing IT infrastructures? Again, this question is primarily addressed in Chapter 4. As discussed in Section 2.3.2, the notion of a context is defined as *"the situation in which something happens and that helps you to understand it"* [19]. As CADS is in essence based on the distributed collection and central analysis of Features, it provides means to capture and to express the context of individual Features, rather than of smartphones as a whole. This is achieved by defining two dedicated components as part of the conceptual model, referred to as Context Parameter and Context. Context Parameters are referenced by Features. A Context Parameter encapsulates data that describes the physical environment at the moment in time when the value of its respective Feature was set (such as the current time or the geographical location of the Feature Collector). On the other hand, a Context Component is a Boolean expression that is formulated based on Context Parameters. This way, Features can be effectively selected based on the values of their respective Context Parameters.

This concept is used for two purposes: (1) to check if a smartphone is in a specific Context or not and (2) to enable a Context-related detection of Signatures and Anomalies. For the first purpose, it is sufficient to check the Context Parameters of a dedicated Feature that was collected directly on a smartphone. Within the exemplary domain-instance presented in this thesis, this Feature was referred to as *smartphone.ContextPing*. This functionality can be used in order to realize Context-related provisioning of services within the IT infrastructure. For the second purpose, the conceptual model of CADS allows that Signatures and Anomalies reference defined Contexts. This way, the set of Features that is considered for detecting Signatures and Anomalies can easily be restricted to those that fulfill the referenced Contexts. For example, this allows to use CADS in order to detect Signatures and Anomalies only if a smartphone (1) is used on-site at a company compound and (2) is used during working hours. This functionality is especially important

when the use of certain apps is considered unwanted only in certain areas of the company's IT infrastructure.

7.3 Future Work

The CADS approach that is presented in this thesis allows a secure integration of smartphones into existing IT infrastructures. Although it meets all requirements that have been defined in Section 2.5, there is clearly room for future work. Five subjects that are especially important are mentioned in the following.

Cross device analysis The correlation model of the CADS approach is currently limited to work on single smartphone devices. That is, all Signatures and Anomalies are evaluated on a per smartphone basis, independently from each other. The training of profiles in order to capture the normal behavior of smartphones is performed device-specific as well. As a consequence, threats that manifest itself in Signatures and Anomalies across multiple smartphones cannot be detected. Future work should address this drawback and find ways to enrich the correlation model of CADS in such a way that it can detect Signatures and Anomalies which span across multiple smartphones.

Evaluation of more sophisticated anomaly detection techniques Within this thesis, simple statistical methods have been employed for anomaly detection. They were used to detect if a smartphone transmits large amounts of data to a remote location. However, numerous anomaly detection techniques exist that have not been considered (see [96] for an overview). Thus, another direction of future work is to analyze if other anomaly detection techniques can be used in order to detect abnormal behavior of smartphones.

Further analysis of domain-specific Features A set of domain-specific Features has been defined within this thesis. Only a subset of them was actually used during the evaluation of the CADS approach. That is, most Features that have been considered reflect properties of installed apps and the traffic consumption of smartphones. Furthermore, Features that were collected from existing services such as an Intrusion Detection System (IDS) and a vulnerability scanner were used as well. However, there are lots of Features whose effectiveness in terms of smartphone security has not been investigated yet. This

especially includes many low level Features that are collected on smartphones (those defined as part of the Category *smartphone.system*) as well as Features that are obtained by crawling app stores such as Google Play. Future work should address this issue.

Improvement of the CADS policy language The language that is used in order to define the policy for the Correlation Engine should be improved. As the focus of this thesis was the development of a network-based approach for smartphone security, and not the development of a new policy language, its structure and expressiveness is rather elementary. Furthermore, the process of manually defining example policies for the Correlation Engine turned out to be complex and error prone. Future work should address this issue, aiming to develop a language that is more flexible and yet easier to use.

Integration of other smartphone platforms The CADS approach was implemented and evaluated based on the Google Android smartphone platform. Other popular platforms, such as Apple iOS, Microsoft Windows Phone and BlackBerry, were not considered in detail. In order to use the CADS approach with other smartphone platforms, two things need to be done: First, Features that are specific for the respective smartphone platform need to be defined at a conceptual level. In this thesis, all Features that are specific to the Android platform are defined in the Category *smartphone.android* or one of its sub-categories. For example, the Feature *smartphone.android.app.permission.Requested* is used to express that a certain permission is requested by an app. Windows Phone 8 supports a similar permission concept. However, privileges that are granted to apps are referred to as *capabilities* instead of permissions. In order to use CADS with Windows Phone 8, Features that appropriately express capabilities need to be defined. Second, an app for the respective smartphone platform needs to be developed. The app must support the measurement of required Features and the transmission of these Features to the Feature Provider via an appropriate communication protocol. In this thesis, IF-MAP was used for this purpose. However, it should be noted that the CADS approach does not depend on the IF-MAP protocol. Instead, another communication protocol can be used as well, as long as it meets the requirements that were specified in Section 4.2.2.

Integration with existing host-based approaches Another area of future work is to analyze how CADS can be integrated with existing, host-based approaches for smart-

phone security. One example for such an host-based approach is CRePE [38]. Both CADS and CRePE provide context-related enforcement of policies. CADS on the network-side, and CRePE on the smartphone device itself. Generally, both approaches can be used in conjunction. CADS could be used to restrict access to certain services based on the smartphone's context. Furthermore, CRePE can be used to enforce context-related policies, even when the respective smartphone is not connected to the IT infrastructure that is protected by CADS.

Investigation of legal issues Legal issues have not been addressed in this thesis at all. Only minor aspects were considered in order to protect the user's privacy (such as to anonymize the smartphone's IMEI before it is transmitted to the Feature Provider). As CADS relies on the distributed collection of data, future work should investigate what data is legally allowed to be collected without violating the user's privacy. This will require an analysis of existing laws, such as the Federal Data Protection Act [180]. Furthermore, the recent efforts of the European Commission in terms of data protection should be considered as well [181].

To summarize, a novel, network-based approach for smartphone security called CADS was presented in this thesis. The approach enables a Context-related detection of Signatures and Anomalies in order to securely integrate smartphones into existing IT infrastructures. In fact, although CADS was designed to specifically address smartphones, it can generally be used for the secure integration of other devices as well. In contrast to existing approaches in the field of smartphone security, CADS combines both Signature and Anomaly detection techniques while considering a smartphone's Context, has the ability to integrate existing security services and supports to employ arbitrary reactions based on detection results.

A Appendix

A.1 Publications

A number of publications have been published within the context of this thesis. Their titles are listed in the following. Detailed bibliographical references are given as part of the bibliography.

- TCADS: Trustworthy, Context-related Anomaly Detection for Smartphones [182]
- On Remote Attestation for Google Chrome OS [183]
- Trusted Service Access with Dynamic Security Infrastructure Configuration [184]
- Trustworthy Anomaly Detection for Smartphones (Poster) [185]
- Automatisches Erkennen mobiler Angriffe auf die IT-Infrastruktur [186]
- Towards Permission-Based Attestation for the Android Platform [187]
- Konsolidierung von Metadaten zur Erhöhung der Unternehmenssicherheit [188]
- Interoperable device identification in Smart-Grid environments [189]
- Towards Trustworthy Networks with Open Source Software [190]
- Interoperable remote attestation for VPN environments [191]
- ESUKOM: Smartphone Security for Enterprise Networks [192]
- Countering Phishing with TPM-bound Credentials [193]
- tNAC - Trusted Network Access Control (Poster) [194]

- Privacy Enhanced Trusted Network Connect [195]
- Towards Trusted Network Access Control [196]
- Towards real Interoperable, real Trusted Network Access Control: Experiences from Implementation and Application of Trusted Network Connect [197]

A.2 History of Android Versions

Table A.1: History of Android versions.

Version	Release Date	Important Changes
1.0	09-2008	Initial release
1.1	02-2009	Bug fixes
		Cupcake
1.5	04-2009	User interface improvements
		Donut
1.6	09-2009	Support for 802.1X and Virtual Private Networks (VPNs)
		Éclair
2.0	09-2009	Support for multiple account synchronization, Bluetooth 2.1
2.0.1	12-2009	Bug fixes
2.1	01-2010	HTML5, IPv6
		Froyo
2.2	05-2010	Android Cloud to Device Messaging (C2DM)
2.2.1	09-2010	Bug fixes
2.2.2	01-2011	Bug fixes
2.2.3	11-2011	Bug fixes
		Gingerbread
2.3	12-2010	ext4 filesystem, Google TV, near field communication (NFC)
2.3.1	12-2010	Bug fixes
2.3.2	01-2011	Bug fixes
2.3.3	02-2011	Bug fixes
2.3.4	04-2011	Google Talk voice and video chat
2.3.5	07-2011	Bug fixes
2.3.6	09-2011	Bug fixes
2.3.7	09-2011	Google Wallet
		Honeycomb
3.0	02-2011	Optimized for tablets, simplified multitasking
3.1	05-2011	User interface improvements
3.2	07-2011	User interface improvements
3.2.1	09-2011	Bugfixes
		Ice Cream Sandwich
4.0	10-2011	Merge of Android 2.x and 3.x, Android Beam, Face Unlock
4.0.1	10-2011	Bug fixes
4.0.2	11-2011	Bug fixes
4.0.3	12-2011	Bug fixes
4.0.4	12-2011	Bug fixes
		Jelly Bean
4.1	07-2012	Google Now, Google Chrome as default web browser
4.1.1	07-2012	Bugfixes
4.1.2	10-2012	Bugfixes
4.2	11-2012	SELinux, security scans for third-party apps

A.3 Complete List of Defined Features and Categories

The following table lists the complete set of Features and Categories that have been defined within this thesis. Note that only a subset has actually been used for the evaluation of the presented approach.

Table A.2: Features and Categories that have been defined within the scope of this thesis. Note that not all of them have been actually used in the evaluation.

Category/Feature	**Card/Type**	**Description**
C: smartphone	1	top level Category for smartphones
F: ContextPing	arbitrary	dummy Feature to get updated Context Parameters
C: smartphone.system	1	low level Features of a smartphone
F: Imei	arbitrary	the smartphone's IMEI
F: Imsi	arbitrary	the smartphone's IMSI
F: Model	arbitrary	the smartphone's model name as displayed to the user
F: FirmwareVersion	arbitrary	the smartphone's firmware version
F: Product	arbitrary	the name of the smartphone product
F: Manufacturer	arbitrary	the manufacturer of the smartphone
F: BasebandVersion	arbitrary	version of the smartphone's radio code
F: KernelVersion	arbitrary	the version of the kernel
F: BuildNumber	arbitrary	the build number of the smartphone
F: Os	arbitrary	version of the operating system
F: Sdk	arbitrary	the SDK version of the system
F: IpAddress	qualified	current IP address
F: MacAddress	qualified	current MAC address
F: SystemBoot	quantitative	last time the smartphone completed to boot
F: SystemShutdown	quantitative	last time the smartphone was shut down
F: AdbEnabled	qualified	indicates if Android Debug Bridge (ADB) is enabled
F: TetheringEnabled	qualified	indicates if tethering is currently enabled
F: NonMarketInstall	qualified	indicates if apps can be installed from unofficial app stores

Category/Feature	Card/Type	Description
F: DataRoamingEnabled	qualified	indicates if data roaming option is enabled
F: AirplaneMode	qualified	indicates if airplane mode is enabled
F: SimState	qualified	the state of the subscriber identity module (SIM)
F: ServiceState	qualified	the service state of the smartphone (e.g. indicates if only emergency calls are possible)
F: UsbMassStorageEnabled	qualified	indicates if USB storage is enabled
F: PhoneType	arbitrary	string that describes the phone type
F: DisplayBrightness	quantitative	the current brightness of the display
F: DisplayWidth	quantitative	the width of the display
F: DisplayHeight	quantitative	the height of the display
F: MediaImageCount	quantitative	number of images found on the smartphone
F: MediaVideoCount	quantitative	number of videos found on the smartphone
F: MediaAudioCount	quantitative	number of audio tracks found on the smartphone
F: RingMode	qualified	ringmode setting
F: VibrateSetting	qualified	indicates if smartphone's vibrator is activated
F: Screen	arbitrary	
F: ProcessCount	quantitative	number of running processes
C: smartphone.system.memory	1	Features regarding the smartphone's memory status
F: MemoryAvailable	quantitative	available memory in bytes
F: MemoryLow	qualified	indicates that the smartphone is in low memory state
F: MemoryThreshold	quantitative	Indicates which threshold (in bytes) has to be reached, before the system is in low memory state and starts killing processes.
C: smartphone.system.battery	1	Features related to the smartphone's battery
F: Power	qualified	Indicates if the smartphone is currently connected to a power supply
F: Level	quantitative	the current power level in percent

Category/Feature	Card/Type	Description
F: Voltage	quantitative	the current voltage
F: Status	qualified	the status of the battery
C: smartphone.system.storage	1	Features that describe the smartphone's storage capabilities
F: MediaState	qualified	state of primary external storage media
F: MediaInternalSize	quantitative	total size of internal storage in bytes
F: MediaInternalFree	quantitative	free internal storage in bytes
F: MediaExternalSize	quantitative	total size of external storage in bytes
F: MediaExternalFree	quantitative	free external storage in bytes
C: smartphone.system.usb	1	USB related Features
F: State	qualified	indicates if a USB device is connected
C: smartphone.sensor	1	Features related to the smartphone's built-in sensors
C: smartphone.sensor.gps	1	Features related to the GPS sensor
F: IsUsed	qualified	indicates if the sensor is currently used
C: smartphone.sensor.camera	1	Features related to the built-in camera sensor
F: NewPicture	qualified	indicates that a new picture was taken
F: IsUsed	qualified	indicates if the camera is currently used
C: smartphone.sensor.audio	1	Features related to the built-in microphone
F: IsUsed	qualified	indicates if the microphone is currently used
C: smartphone.communication	1	Features related to communication capabilities of the smartphone
C: smartphone.communication.ip	1	Features related to IP based communication
F: Rx3g	quantitative	amount of received bytes via mobile network interface
F: Tx3g	quantitative	amount of transmitted bytes via mobile network interface

Category/Feature	Card/Type	Description
F: RxOther	quantitative	amount of received bytes via other interfaces except the mobile network interface
F: TxOther	quantitative	amount of transmitted bytes via other interfaces except the mobile network interface
C: smartphone.communication.gsm	1	GSM related Features
F: CellId	arbitrary	ID of the current GSM cell
F: CellLac	arbitrary	local area code of the current GSM cell
C: smartphone.communication.cdma	1	CDMA related Features
F: CellBaseStationId	quantitative	base station identification number of current CDMA cell
F: CellNetworkId	quantitative	network identification number of current CDMA cell
F: CellSystemId	quantitative	system identification number of current CDMA cell
F: CellLatitude	quantitative	GPS latitude value for the current CDMA cell
F: CellLongitude	quantitative	GPS longitude value for the current CDMA cell
C: smartphone.communication.wifi	1	Features related to the smartphone's WiFi interface
C: smartphone.communication.wifi.connection	1	Features related to a specific WiFi connection that was established
F: Bssid	arbitrary	the access point's Basic Service Set Identification (BSSID)
F: Ssid	arbitrary	the access point's Service Set Identification (SSID)
F: SsidHidden	qualified	indicates of the SSID is hidden
F: LinkSpeed	quantitative	speed of connection in Mbps
F: IpAddress	quantitative	IP address used for the connection
F: MacAddress	qualified	the access point's MAC address
F: WifiState	qualified	state of the WiFi interface
C: smartphone.communication.wifi.scan	N	Features related to scanned wireless networks
F: Bssid	arbitrary	BSSID of scanned access point
F: Ssid	arbitrary	SSID of scanned access point

Category/Feature	Card/Type	Description
F: Frequency	quantitative	frequency in MHz of the access point
F: Level	quantitative	signal level in dBm
F: Capabilities	arbitrary	the access point's supported security capabilities
C: smartphone.communication.bluetooth	1	Bluetooth related Features
F: State	qualified	state of the Bluetooth adapter
F: Scanmode	qualified	scan mode of the Bluetooth adapter
C: smartphone.communication.bluetooth.connection	1	Features related to a Bluetooth connection
F: Name	arbitrary	name of the remote device
F: Address	arbitrary	address of the remote device
F: ConnectionState	qualified	state of the connection
F: BondState	arbitrary	bond state of the Bluetooth adapter
C: smartphone.communication.sms	1	Features related to smartphone's Short Message Service (SMS) system
F: LastOutgoingSms	qualified	last time a SMS was sent
F: LastIncomingSms	qualified	last time a SMS was received
C: smartphone.android	1	Android specific Features
C: smartphone.android.app	N	Features related to an installed app
F: Name	arbitrary	the name of the app
F: PackageName	arbitrary	the apps package name
F: Installer	arbitrary	app store used for installing the app
F: VersionName	arbitrary	name of the app's version
F: VersionCode	quantitative	code of the app's version
F: MinSdk	quantitative	minimum SDK version required to run the app
F: Installed	quantitative	moment in time when the app was installed
F: LastUpdate	quantitative	moment in time when the app was updated
F: IsRunning	qualified	indicates if the app is currently running
F: GooglePlayCategory	qualified	category of the app in the Google Play app store
F: Rating	quantitative	rating of the app in the Google Play app store

Category/Feature	Card/Type	Description
F: Downloads	quantitative	number of downloads of the app in the Google Play app store
C: smartphone.android.app.permission	N	Features related to an app's permissions
F: Required	arbitrary	a permission that is required in order to use a component of the app
F: Requested	arbitrary	a permission that is requested by the app
F: ProtectionLevel	quantitative	indicates the protection level of the required permission
C: ids	N	top level Category for IDS events
F: EventType	arbitrary	Type of an event
F: EventName	arbitrary	Name of an event
F: EventSource	arbitrary	Source of the event
F: EventThreat	quantitative	Threat level that is associated with the event
F: EventConfidence	arbitrary	Confidence of the IDS that the event occured
F: EventImpact	quantitative	Impact of the event
F: HighTraffic	quantitative	Special event that indicates that there was a high amount of traffic. Encapsulates the amount of traffic in bytes.
C: vulnerability	N	top level Category for vulnerabilities
F: Name	arbitrary	The name of the Network Vulnerability Test (NVT)[1] that was performed
F: Port	arbitrary	The port at which the vulnerability was detected
F: CvssBase	quantitative	The CVSS[2] base score that was defined for the vulnerability. Ranges from 0 to 10.
F: Threat	quantitative	The threat level of the vulnerability as determined by OpenVAS
F: CveIdentifier	arbitrary	The CVE[3] number that identifies the vulnerability

[1] http://www.openvas.org/openvas-nvt-feed.html
[2] https://nvd.nist.gov/cvss.cfm
[3] http://cve.mitre.org/

Category/Feature	Card/Type	Description
C: correlationresult	1	top level Category for correlation results
C: correlationresult.smartphonevisibility	N	correlation result Features for the smartphone visibility scenario
F: IsASmartphone	qualified	indicates if the respective device is a smartphone or not
C: correlationresult.context	N	correlation result Features to express that a smartphone is in a certain Context
F: IsFulfilled	arbitrary	represents a Context that is fulfilled
F: IsNotFulfilled	arbitrary	represents a Context that is not fulfilled
C: correlationresult.alert	N	Category that encapsulates correlation result that represent alerts
F: AlertName	arbitrary	the name of an alert
F: AlertSeverity	qualified	the severity of the alert
C: correlationresult.enforcement	N	Category to represent Features that encapsulate enforcement actions
F: EnforcementAction	arbitrary	string that identifies the enforcement action
F: $1	arbitrary	technical Feature used in the prototype implementation to pass an additional parameter to the enforcement action

A.4 Grammar for the CADS Policy Language

```
(***KEYWORDS***)
CONTEXTKEY    = "context"
HINTKEY       = "hint"
ANOMALYKEY    = "anomaly"
SIGNATUREKEY  = "signature"
CONDITIONKEY  = "condition"
ACTIONKEY     = "action"
RULEKEY       = "rule"
IFKEY         = "if"
DOKEY         = "do"
COUNTKEY      = "count"
GETKEY        = "get"

(***OPERATORS***)
DEFINEOP = ":="
BOOLOP   = "and" | "or"
COMPOP   = "<" | ">" | "<=" | ">=" | "=" | "!="

(***CONSTANTS***)
FCTXPARAMTYPE =
   "LOCATION" | "DATETIME" | "OTHERDEVICES" | "SLIDING" | "TRUSTLEVEL"

(***DATATYPES***)
DIGIT       = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
NUMVAL      = [ "-" ] {DIGIT} | {<DIGIT>} "." {DIGIT}
LETTER      =
   "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" |
   "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z" |
   "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" |
   "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"
STRSYMBOL  = "/",":",".","_","!","-","@"," ","$","!"]
STRVAL     = '"' LETTER | STRSYMBOL | DIGIT {LETTER | STRSYMBOL | DIGIT} '"'
IDENTIFIER = LETTER | DIGIT {LETTER | DIGIT}

(***LANGUAGE PRODUCTIONS***)
POLICY           = CONTEXTBLOCK HINTBLOCK ANOMALYBLOCK SIGNATUREBLOCK CONDITIONBLOCK
    ACTIONBLOCK RULEKEY "{" {IDENTIFIER RULE} "}"

CONTEXTBLOCK    = CONTEXTKEY "{" {CTX} "}"
HINTBLOCK       = HINTKEY "{" {HINT} "}"
ANOMALYBLOCK    = ANOMALYKEY "{" {ANOMALY} "}"
SIGNATUREBLOCK  = SIGNATUREKEY "{" {SIGNATURE} "}"
CONDITIONBLOCK  = CONDITIONKEY "{" {CONDITION} "}"
ACTIONBLOCK     = ACTIONKEY "{" {IDENTIFIER ":=" ACTION ";"} "}"

CONDITION       = IDENTIFIER DEFINEOP IDENTIFIER {BOOLOP IDENTIFIER} ";"
ANOMALY         = IDENTIFIER DEFINEOP HINTEXP    {BOOLOP HINTEXP} {IDENTIFIER} ";"
```

```
47 HINT          = IDENTIFIER DEFINEOP FEATURE {FEATURE } PROCEDURE ";"
48 SIGNATURE     = IDENTIFIER DEFINEOP EXPR {BOOLOP EXPR} IDENTIFIER {IDENTIFIER} ";"
49 CTX           = IDENTIFIER DEFINEOP FCTXPARAMTYPE COMPOP VALUE {BOOLOP FCTXPARAMTYPE
       COMPOP VALUE} ";"
50 HINTEXP       = IDENTIFIER COMPOP NUMVAL
51
52 PROCEDURE     = STRVAL STRVAL
53 FEATURE       = VALUE | COUNTKEY "(" STRVAL ")" | GETKEY "(" STRVAL ")"
54 RULE          = IFKEY IDENTIFIER DOKEY {IDENTIFIER}
55 ACTION        = {KEY VALUE}
56 EXPR          = FEATURE COMPOP VALUE
57
58 KEY           = STRVAL
59 VALUE         = STRVAL | NUMVAL
```

Listing A.1: Grammar that defines the Policy used by irondetect.

Bibliography

[1] "PlayStation Network Restoration Begins," News on Company Website, Sony Computer Entertainment, May 2011. [Retrieved: 05-Mar-2013] http://uk.playstation.com/psn/news/articles/detail/item369506/PSN-Qriocity-Service-Update/

[2] "Hackers in China Attacked The Times for Last 4 Months," News on Company Website, The New York Times, Jan. 2013. [Retrieved: 05-Mar-2013] http://www.nytimes.com/2013/01/31/technology/chinese-hackers-infiltrate-new-york-times-computers.html?pagewanted=all

[3] S. Gorman, "Cyber Combat: Act of War," The Wall Street Journal, May 2011. [Retrieved: 05-Mar-2013] http://online.wsj.com/article/SB10001424052702304563104576355623135782718.html

[4] A.-d. Schmidt and S. Albayrak, "Malicious software for smartphones," Tech. Rep., 2008. [Retrieved: 05-Mar-2013] https://www.dai-labor.de/fileadmin/files/publications/smartphone_malware.pdf

[5] R. T. Llamas and W. Stofega, "Worldwide Smartphone 2012–2016 Forecast and Analysis," International Data Corporation, Mar. 2012. [Retrieved: 05-Mar-2013] http://www.idc.com/getdoc.jsp?containerId=233553

[6] "The Mobile Movement - Understanding Smartphone Users," Google Inc./Ipsos OTX MediaCT, Apr. 2011. [Retrieved: 05-Mar-2013] http://www.gstatic.com/ads/research/en/2011_TheMobileMovement.pdf

[7] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos, "Paranoid Android: Versatile Protection For Smartphones," in *Proceedings of the 26th Annual Computer Security Applications Conference*. ACM, 2010, pp. 347–356. [Retrieved: 05-Mar-2013] http://dl.acm.org/citation.cfm?doid=1920261.1920313

[8] D. Dagon, T. Martin, and T. Starner, "Mobile Phones as Computing Devices: The Viruses are Coming!" *IEEE Pervasive Computing*, vol. 3, no. 4, pp. 11–15, Oct. 2004. [Retrieved: 05-Mar-2013] http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1369156

[9] L. Davi, A. Dmitrienko, A. Sadeghi, and M. Winandy, "Privilege escalation attacks on android," *Information Security*, pp. 346–360, 2011. [Retrieved: 05-Mar-2013] http://www.springerlink.com/index/D275570090NG72JT.pdf

[10] R. Schlegel, K. Zhang, X. Zhou, M. Intwala, A. Kapadia, and X. Wang, "Soundcomber: A Stealthy and Context-Aware Sound Trojan for Smartphones," in *Proceedings of the 18th Annual Network and Distributed System Security Symposium (NDSS)*, 2011, pp. 17–33. [Retrieved: 05-Mar-2013] https://www.cs.indiana.edu/~kapadia/papers/soundcomber-ndss11.pdf

[11] N. Xu, F. Zhang, Y. Luo, W. Jia, D. Xuan, and J. Teng, "Stealthy video capturer: a new video-based spyware in 3G smartphones," in *Proceedings of the second ACM conference on Wireless network security*. ACM, 2009, pp. 69–78. [Retrieved: 05-Mar-2013] http://portal.acm.org/citation.cfm?id=1514274.1514285

[12] D. Bachfeld, "ZeuS-Trojaner befällt Android," Jul. 2011. [Retrieved: 05-Mar-2013] http://www.heise.de/security/meldung/ZeuS-Trojaner-befaellt-Android-1278449.html

[13] ——, "Google entfernt über 50 infizierte Apps aus dem Android Market," Mar. 2011. [Retrieved: 05-Mar-2013] http://www.heise.de/security/meldung/Google-entfernt-ueber-50-infizierte-Apps-aus-dem-Android-Market-1200662.html

[14] W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," in *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 2009, pp. 235–245. [Retrieved: 05-Mar-2013] http://portal.acm.org/citation.cfm?id=1653691

[15] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel, "Semantically Rich Application-Centric Security in Android," *2009 Annual Computer Security Applications Conference*, pp. 340–349, Dec. 2009. [Retrieved: 05-Mar-2013] http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5380692

[16] W. Enck, P. Gilbert, B. Chun, L. Cox, J. Jung, P. McDaniel, and A. Sheth, "TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones," in *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2010. [Retrieved: 05-Mar-2013] http://appanalysis.org/tdroid10.pdf

[17] A. Felt, H. Wang, A. Moshchuk, S. Hanna, E. Chin, K. Greenwood, D. Wagner, D. Song, M. Finifter, J. Weinberger, and Others, "Permission Re-Delegation: Attacks and Defenses," in *20th Usenix Security Symposium, San Fansisco, CA*, 2011. [Retrieved: 05-Mar-2013] http://research.microsoft.com/en-us/um/people/helenw/papers/permissionredelegation.pdf

[18] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, ""Andromaly": a behavioral malware detection framework for android devices," *Journal of Intelligent Information Systems*, Jan. 2011. [Retrieved: 05-Mar-2013] http://www.springerlink.com/index/10.1007/s10844-010-0148-x

[19] "Oxford Advanced Learners Dictionary," 2012. [Retrieved: 05-Mar-2013] http://oald8.oxfordlearnersdictionaries.com/dictionary/context

[20] TCG Trusted Network Connect Work Group, "TNC IF-MAP Binding for SOAP, Version 2.1, Revision 15," Trusted Computing Group, May 2012. [Retrieved: 05-Mar-2013] http://www.trustedcomputinggroup.org/resources/tnc_ifmap_binding_for_soap_specification

[21] ESUKOM Project Consortium, "ESUKOM Project Website." [Retrieved: 05-Mar-2013] http://www.esukom.de

[22] "BSI Short Information - IT Baseline Protection: the Basis for IT Security," Bundesamt für Sicherheit in der Informationstechnik. [Retrieved: 05-Mar-2013] https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/FB/F14itbas_en_pdf.pdf?___blob=publicationFile

[23] "BSI-Standard 100-1: Managementsysteme für Informationssicherheit (ISMS)," Bundesamt für Sicherheit in der Informationstechnik, May 2008, version 1.5. [Retrieved: 05-Mar-

2013] https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/ITGrundschutzstandards/standard_1001_pdf.pdf?__blob=publicationFile

[24] "BSI-Standard 100-2: IT-Grundschutz-Vorgehensweise," Bundesamt für Sicherheit in der Informationstechnik, May 2008, version 2.0. [Retrieved: 05-Mar-2013] https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/ITGrundschutzstandards/standard_1002_pdf.pdf?__blob=publicationFile

[25] "BSI-Standard 100-3: Risikoanalyse auf der Basis von IT-Grundschutz," Bundesamt für Sicherheit in der Informationstechnik, May 2008, version 2.5. [Retrieved: 05-Mar-2013] https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/ITGrundschutzstandards/standard_1003_pdf.pdf?__blob=publicationFile

[26] "BSI-Standard 100-4: Notfallmanagement," Bundesamt für Sicherheit in der Informationstechnik, Nov. 2008, version 1.0. [Retrieved: 05-Mar-2013] https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/ITGrundschutzstandards/standard_1004_pdf.pdf?__blob=publicationFile

[27] "Openclipart Website." [Retrieved: 05-Mar-2013] http://openclipart.org/

[28] "Media Access Control (MAC) Bridges and Virtual Bridge Local Area Networks," IEEE Computer Society, Aug. 2011, IEEE Std 802.1Q. [Retrieved: 05-Mar-2013] http://standards.ieee.org/getieee802/download/802.1Q-2011.pdf

[29] C. Rigney, S. Willens, A. Rubens, and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)," RFC 2865 (Draft Standard), Internet Engineering Task Force, Jun. 2000, updated by RFCs 2868, 3575, 5080. [Retrieved: 05-Mar-2013] http://www.ietf.org/rfc/rfc2865.txt

[30] V. Fajardo, J. Arkko, J. Loughney, and G. Zorn, "Diameter Base Protocol," RFC 6733 (Proposed Standard), Internet Engineering Task Force, Oct. 2012. [Retrieved: 05-Mar-2013] http://www.ietf.org/rfc/rfc6733.txt

[31] H. Zimmermann, "Osi reference model–the iso model of architecture for open systems interconnection," *Communications, IEEE Transactions on*, vol. 28, no. 4, pp. 425 – 432, apr 1980.

[32] "IEEE 802.1X Port-Based Network Access Control," IEEE Computer Society, Feb. 2010, IEEE 802.1X-2010. [Retrieved: 05-Mar-2013] http://standards.ieee.org/findstds/standard/802.1X-2010.html

[33] K. Scarfone and P. Mell, "Guide to Intrusion Detection and Prevention Systems (IDPS)," National Institute of Standards and Technology (NIST), Feb. 2007. [Retrieved: 05-Mar-2013] http://csrc.nist.gov/publications/nistpubs/800-94/SP800-94.pdf

[34] "Network Access Protection," Microsoft Corporation, 2012, Microsoft Technet. [Retrieved: 05-Mar-2013] http://technet.microsoft.com/en-us/network/bb545879.aspx

[35] "Getting Started with Cisco NAC Network Modules in Cisco Access Routers," Cisco Systems Inc., Nov. 2012. [Retrieved: 05-Mar-2013] http://www.cisco.com/en/US/docs/security/nac/appliance/installation_guide/netmodule/nacnmgsg.pdf

[36] TCG Trusted Network Connect Work Group, "TNC Architecture for Interoperability, Version 1.5, Revision 3," Trusted Computing Group, May 2012. [Retrieved: 05-Mar-2013] https://www.trustedcomputinggroup.org/resources/tnc_architecture_for_interoperability_specification

[37] R. Shirey, "Internet Security Glossary, Version 2," RFC 4949 (Informational), Internet Engineering Task Force, Aug. 2007. [Retrieved: 05-Mar-2013] http://www.ietf.org/rfc/rfc4949.txt

[38] M. Conti, V. Nguyen, and B. Crispo, "CRePE: context-related policy enforcement for android," in *Proceedings of the 13th international conference on Information security*. Springer, 2011, pp. 331–345. [Retrieved: 05-Mar-2013] http://www.springerlink.com/index/574882RN4T65364M.pdf

[39] B. Lampson, M. Abadi, M. Burrows, and E. Wobber, "Authentication in distributed systems: theory and practice," *ACM Trans. Comput. Syst.*, vol. 10, no. 4, pp. 265–310, Nov. 1992. [Retrieved: 05-Mar-2013] http://doi.acm.org/10.1145/138873.138874

[40] A. Cummings, T. Lewellen, D. McIntire, M. A. P., and R. Trzeciak, "Insider Threat Study: Illicit Cyber Activity Involving Fraud in the U.S. Financial Services Sector," Carnegie Mellon Software Engineering Institute, Tech. Rep., Jul. 2012, (CMU/SEI-2012-SR-004). [Retrieved: 05-Mar-2013] http://www.sei.cmu.edu/reports/12sr004.pdf

[41] M. Becher, "Security of smartphones at the dawn of their ubiquitousness," Ph.D. dissertation, Universität Mannheim, 2009. [Retrieved: 05-Mar-2013] http://ub-madoc.bib.uni-mannheim.de/2998

[42] P. Zheng and L. M. Ni, "Spotlight: The rise of the smart phone," *IEEE Distributed Systems Online*, vol. 7, no. 3, pp. 3–, Mar. 2006. [Retrieved: 05-Mar-2013] http://dx.doi.org/10.1109/MDSO.2006.22

[43] J. Stempel, "Lawsuit: Amazon Fights Apple Over 'App Store' Trademark Claim," Huffington Post Website, Sep. 2012. [Retrieved: 05-Mar-2013] http://www.huffingtonpost.com/2012/09/26/lawsuit-amazon-apple-app-store-trademark_n_1917899.html

[44] A. Dmitrienko, A.-R. Sadeghi, S. Tamrakar, and C. Wachsmann, "Smarttokens: Delegable access control with nfc-enabled smartphones," in *International Conference on Trust & Trustworthy Computing (TRUST)*. Springer, Jun 2012.

[45] S. Tamrakar, J.-E. Ekberg, and N. Asokan, "Identity verification schemes for public transport ticketing with NFC phones," *Proceedings of the sixth ACM workshop on Scalable trusted computing - STC '11*, p. 37, 2011. [Retrieved: 05-Mar-2013] http://dl.acm.org/citation.cfm?doid=2046582.2046591

[46] International Data Corporation (IDC), "Android marks fourth anniversary since launch with 75.0% market share in third quarter," Nov. 2012. [Retrieved: 05-Mar-2013] http://www.idc.com/getdoc.jsp?containerId=prUS23771812

[47] "iPhone 5 First Weekend Sales Top Five Million," Apple Press Info, Sep. 2012. [Retrieved: 05-Mar-2013] http://www.apple.com/pr/library/2012/09/24iPhone-5-First-Weekend-Sales-Top-Five-Million.html

[48] Koekkoek, Hendrik, "Distimo Publication Full Year 2011," Dec. 2011. [Retrieved: 05-Mar-2013] http://www.distimo.com/publications/archive/Distimo%20Publication%20-%20Full%20Year%202011.pdf

[49] "Apache License, Version 2.0," The Apache Software Foundation, Jan. 2004. [Retrieved: 05-Mar-2013] http://www.apache.org/licenses/LICENSE-2.0

[50] F. Scherschel, "Google plans to ease the Android update problem," Jun. 2012. [Retrieved: 05-Mar-2013] http://www.h-online.com/open/news/item/Google-plans-to-ease-the-Android-update-problem-1628721.html

[51] Google Inc., "Google Dashboards," Nov. 2012. [Retrieved: 05-Mar-2013] http://developer.android.com/about/dashboards/index.html

[52] R. Meier, *Professional Android 4 Application Development.* John Wiley & Sons, Inc., 2012.

[53] "Google Says 700,000 Applications Available for Android," Oct. 2012. [Retrieved: 05-Mar-2013] http://www.businessweek.com/news/2012-10-29/google-says-700-000-applications-available-for-android-devices

[54] Google Inc., "Android API Guides," Nov. 2012. [Retrieved: 05-Mar-2013] http://developer.android.com/guide/components/index.html

[55] W. Enck, M. Ongtang, and P. McDaniel, "Understanding Android Security," *IEEE Security & Privacy Magazine*, vol. 7, no. 1, pp. 50–57, Jan. 2009. [Retrieved: 05-Mar-2013] http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4768655

[56] E. Chin, A. Felt, K. Greenwood, and D. Wagner, "Analyzing inter-application communication in Android," in *Procs. of the 9th Annual International Conference on Mobile Systems, Applications, and Services*, 2011. [Retrieved: 05-Mar-2013] http://www.eecs.berkeley.edu/~emc/papers/mobi168-chin.pdf

[57] C. S. Nachenberg, "A Window Into Mobile Device Security," Symantec, Tech. Rep., 2011. [Retrieved: 05-Mar-2013] http://www.symantec.com/about/news/release/article.jsp?prid=20110627_02

[58] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, and C. Glezer, "Google Android: A Comprehensive Security Assessment," *IEEE Security & Privacy Magazine*, vol. 8, no. 2, pp. 35–44, Mar. 2010. [Retrieved: 05-Mar-2013] http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5396322

[59] Android Open Source Project, "Security Technical Information," Nov. 2012. [Retrieved: 05-Mar-2013] http://source.android.com/tech/security/index.html

[60] J. Daemen and V. Rijmen, *The design of Rijndael: AES — the Advanced Encryption Standard*. Springer-Verlag, 2002.

[61] "Specification for the advanced encryption standard (aes)," Federal Information Processing Standards Publication 197, 2001. [Retrieved: 05-Mar-2013] http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[62] S. Bugiel, L. Davi, A. Dmitrienko, S. Heuser, A. Sadeghi, and B. Shastry, "Practical and lightweight domain isolation on Android," in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*. ACM, 2011, pp. 51–62. [Retrieved: 05-Mar-2013] http://dl.acm.org/citation.cfm?id=2046624

[63] Google Inc., "The Android Developers Guide - Manifest.permission," Nov. 2012. [Retrieved: 05-Mar-2013] http://developer.android.com/reference/android/Manifest.permission.html

[64] A. Felt, K. Greenwood, and D. Wagner, "The effectiveness of application permissions," in *2nd USENIX Conference on Web Application Development*, 2011, p. 75. [Retrieved: 05-Mar-2013] http://www.usenix.org/event/webapps11/tech/final_files/webapps11_proceedings.pdf#page=83

[65] "<permission>," Android API Guides, 2012. [Retrieved: 05-Mar-2013] http://developer.android.com/guide/topics/manifest/permission-element.html

[66] K. Au, Y. Zhou, Z. Huang, and P. Gill, "Short paper: a look at smartphone permission models," in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, 2011, pp. 63–67. [Retrieved: 05-Mar-2013] http://dl.acm.org/citation.cfm?id=2046626

[67] P. C. v. O. Alfred J. Menezes and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 2001.

[68] H. Lockheimer, "Android and Security," Google Mobile Blog, Feb. 2012. [Retrieved: 05-Mar-2013] http://googlemobile.blogspot.de/2012/02/android-and-security.html

[69] N. J. Percoco and S. Schulte, "Adventures in BouncerLand," *Black Hat USA*, 2012. [Retrieved: 05-Mar-2013] http://media.blackhat.com/bh-us-12/Briefings/Percoco/BH_US_12_Percoco_Adventures_in_Bouncerland_WP.pdf

[70] X. Jiang, "An Evaluation of the Application ("App") Verification Service in Android 4.2," Personal Website, Dec. 2012. [Retrieved: 05-Mar-2013] http://www.cs.ncsu.edu/faculty/jiang/appverify/

[71] N. Leavitt, "Malicious code moves to mobile devices," *Computer*, vol. 33, no. 12, pp. 16–19, Dec. 2000. [Retrieved: 05-Mar-2013] http://dl.acm.org/citation.cfm?id=619058.621603

[72] S. N. Foley and R. Dumigan, "Are handheld viruses a significant threat?" *Commun. ACM*, vol. 44, no. 1, pp. 105–107, Jan. 2001. [Retrieved: 05-Mar-2013] http://doi.acm.org/10.1145/357489.357516

[73] C. Guo, H. Wang, and W. Zhu, "Smart-phone attacks and defenses," in *ACM Workshop on Hot Topics in Networks*, 2004. [Retrieved: 05-Mar-2013] http://research.microsoft.com/en-us/um/people/helenw/papers/smartphone.pdf

[74] J. Jamaluddin, N. Zotou, and P. Coulton, "Mobile phone vulnerabilities: a new generation of malware," in *IEEE International Symposium on Consumer Electronics, 2004.* IEEE, 2004, pp. 199–202. [Retrieved: 05-Mar-2013] http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1375935

[75] N. Leavitt, "Mobile phones: The next frontier for hackers?" *Computer*, vol. 38, no. 4, pp. 20–23, Apr. 2005. [Retrieved: 05-Mar-2013] http://dx.doi.org/10.1109/MC.2005.134

[76] M. Hypponen, "State of cell phone malware in 2007," 2007, talk given at the 16th Usenix Security Symposium, Boston, MA. [Retrieved: 05-Mar-2013] http://static.usenix.org/event/sec07/tech/hypponen.pdf

[77] M. Becher, F. C. Freiling, J. Hoffmann, T. Holz, S. Uellenbeck, and C. Wolf, "Mobile Security Catching Up? Revealing the Nuts and Bolts of the Security of Mobile Devices," in *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, ser. SP '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 96–111. [Retrieved: 05-Mar-2013] http://dx.doi.org/10.1109/SP.2011.29

[78] J. Oberheide and F. Jahanian, "When mobile is harder than fixed (and vice versa): demystifying security challenges in mobile environments," in *Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications*. ACM, 2010, pp. 43–48. [Retrieved: 05-Mar-2013] http://portal.acm.org/citation.cfm?id=1734583.1734595

[79] B. Dixon, Y. Jiang, and A. Jaiantilal, "Location based power analysis to detect malicious code in smartphones," in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, 2011, pp. 27–32. [Retrieved: 05-Mar-2013] http://dl.acm.org/citation.cfm?id=2046620

[80] S. Egelman, A. P. Felt, and D. Wagner, "Choice Architecture and Smartphone Privacy : There ' s A Price for That," in *Workshop on the Economics of Information Security (WEIS) 2012*, 2012. [Retrieved: 05-Mar-2013] http://www.guanotronic.com/~serge/papers/weis12.pdf

[81] E. Chin, A. P. Felt, V. Sekar, and D. Wagner, "Measuring user confidence in smartphone security and privacy," in *Proceedings of the Eighth Symposium on Usable Privacy and Security - SOUPS '12*, no. 1. New York, New York, USA: ACM Press, 2012. [Retrieved: 05-Mar-2013] http://dl.acm.org/citation.cfm?doid=2335356.2335358

[82] W. Shin, S. Kiyomoto, K. Fukushima, and T. Tanaka, "Towards Formal Analysis of the Permission-Based Security Model for Android," *2009 Fifth International Conference on Wireless and Mobile Communications*, pp. 87–92, 2009. [Retrieved: 05-Mar-2013] http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5279458

[83] ——, "A Formal Model to Analyze the Permission Authorization and Enforcement in the Android Framework," *2010 IEEE Second International*

Conference on Social Computing, pp. 944–951, Aug. 2010. [Retrieved: 05-Mar-2013] http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5591409

[84] W. Shin, S. Kwak, S. Kiyomoto, K. Fukushima, and T. Tanaka, "A small but non-negligible flaw in the android permission scheme," in *Policies for Distributed Systems and Networks (POLICY), 2010 IEEE International Symposium on*, july 2010, pp. 107 –110. [Retrieved: 05-Mar-2013] http://dx.doi.org/10.1109/POLICY.2010.11

[85] E. Fragkaki, L. Bauer, L. Jia, and D. Swasey, "Modeling and enhancing android's permission system," in *ESORICS*, ser. Lecture Notes in Computer Science, S. Foresti, M. Yung, and F. Martinelli, Eds., vol. 7459. Springer, 2012, pp. 1–18. [Retrieved: 05-Mar-2013] http://dx.doi.org/10.1007/978-3-642-33167-1_1

[86] N. Hardy, "The Confused Deputy: (or why capabilities might have been invented)," *ACM SIGOPS Operating Systems Review*, vol. 22, no. 4, pp. 36–38, Oct. 1988. [Retrieved: 05-Mar-2013] http://portal.acm.org/citation.cfm?doid=54289.871709

[87] D. Barrera, H. Kayacik, P. van Oorschot, and A. Somayaji, "A methodology for empirical analysis of permission-based security models and its application to android," in *Proceedings of the 17th ACM conference on Computer and communications security*, no. 1. ACM, 2010, pp. 73–84. [Retrieved: 05-Mar-2013] http://portal.acm.org/citation.cfm?id=1866307.1866317

[88] T. Kohonen, "The self-organizing map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990. [Retrieved: 05-Mar-2013] http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=58325

[89] J. Saltzer and M. Schroeder, "The protection of information in computer systems," *Proceedings of the IEEE*, vol. 63, pp. 1278–1308, 1975. [Retrieved: 05-Mar-2013] http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1451869

[90] A. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 627–638. [Retrieved: 05-Mar-2013] http://dl.acm.org/citation.cfm?id=2046779

[91] T. Vidas, N. Christin, and L. Cranor, "Curbing android permission creep," in *Proceedings of the Web 2.0 Security and Privacy 2011 workshop (W2SP 2011)*, vol. 2, Oakland, CA, 2011. [Retrieved: 05-Mar-2013] http://www.andrew.cmu.edu/user/nicolasc/publications/VCC-W2SP11.pdf

[92] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: user attention, comprehension, and behavior," in *Proceedings of the Eighth Symposium on Usable Privacy and Security*, ser. SOUPS '12. New York, NY, USA: ACM, 2012, pp. 3:1–3:14. [Retrieved: 05-Mar-2013] http://doi.acm.org/10.1145/2335356.2335360

[93] A. P. Felt, S. Egelman, M. Finifter, D. Akhawe, and D. Wagner, "How to ask for permission," in *Proceedings of the 7th USENIX conference on Hot Topics in Security*, ser. HotSec'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 7–7. [Retrieved: 05-Mar-2013] http://dl.acm.org/citation.cfm?id=2372387.2372394

[94] M. Grace, Y. Zhou, Z. Wang, and X. Jiang, "Systematic detection of capability leaks in stock Android smartphones," in *Proceedings of the 19th Network and Distributed System Security Symposium (NDSS)*, Feb. 2012. [Retrieved: 05-Mar-2013] http://www.csc.ncsu.edu/faculty/jiang/pubs/NDSS12_WOODPECKER.pdf

[95] M. Miettinen and P. Halonen, "Host-Based Intrusion Detection for Advanced Mobile Devices," in *20th International Conference on Advanced Information Networking and Applications - Volume 1 (AINA'06)*. Ieee, 2006, pp. 72–76. [Retrieved: 05-Mar-2013] http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1620356

[96] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly Detection: A Survey," *ACM Computing Surveys*, vol. 41, no. 3, pp. 1–58, Jul. 2009. [Retrieved: 05-Mar-2013] http://portal.acm.org/citation.cfm?doid=1541880.1541882

[97] C. Kruegel and G. Vigna, "Anomaly detection of web-based attacks," in *Proceedings of the 10th ACM conference on Computer and communications security*, ser. CCS '03. New York, NY, USA: ACM, 2003, pp. 251–261. [Retrieved: 05-Mar-2013] http://doi.acm.org/10.1145/948109.948144

[98] Q. Xu, J. Erman, A. Gerber, and Z. Mao, "Identifying diverse usage behaviors of smartphone apps," in *Proceedings of the 2011 ACM SIGCOMM conference*

on Internet measurement conference (IMC'11), 2011. [Retrieved: 05-Mar-2013] http://dl.acm.org/citation.cfm?id=2068816.2068847

[99] W. Enck, D. Octeau, P. Mcdaniel, S. Chaudhuri, and I. Infrastructure, "A Study of Android Application Security," in *Proceedings of the 20th USENIX Security Symposium*, San Francisco, 2011. [Retrieved: 05-Mar-2013] http://www.enck.org/pubs/enck-sec11.pdf

[100] M. Egele, C. Kruegel, E. Kirda, and G. Vigna, "PiOS: detecting privacy leaks in iOS applications," in *Network and Distributed System Security Symposium (NDSS)*, 2011. [Retrieved: 05-Mar-2013] http://www.seclab.tuwien.ac.at/papers/egele-ndss11.pdf

[101] A. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A survey of mobile malware in the wild," in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*. ACM, 2011, pp. 3–14. [Retrieved: 05-Mar-2013] http://dl.acm.org/citation.cfm?id=2046618

[102] S. Fahl, M. Harbach, T. Muders, M. Smith, L. Baumgärtner, and B. Freisleben, "Why eve and mallory love android: an analysis of android ssl (in)security," in *Proceedings of the 2012 ACM conference on Computer and communications security*, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 50–61. [Retrieved: 05-Mar-2013] http://doi.acm.org/10.1145/2382196.2382205

[103] B. Sanz, I. Santos, and C. Laorden, "On the Automatic Categorisation of Android Applications," in *Proceedings of the 9th IEEE Consumer Communications and Networking Conference (CCNC)*, 2012. [Retrieved: 05-Mar-2013] http://paginaspersonales.deusto.es/isantos/publications/2012/Sanz_2012_CCNC_Android_Apps_Categorisation.pdf

[104] J. Pearl, "Reverend Bayes on inference engines: A distributed hierarchical approach," in *Proceedings of the American Association of Artificial Intelligence National Conference on AI*, Pittsburgh, PA, 1982, pp. 133–136.

[105] J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, Mar. 1986. [Retrieved: 05-Mar-2013] http://dx.doi.org/10.1023/A:1022643204877

[106] Fix, Evelyn and Hodges, Jr, J. L., "Discriminatory Analysis - Nonparametric Discrimination: Small Sample Performance," Project 21-49-004, Report Number 11, Tech. Rep., 1952.

[107] M. Hearst, S. Dumais, E. Osman, J. Platt, and B. Scholkopf, "Support vector machines," *Intelligent Systems and their Applications, IEEE*, vol. 13, no. 4, pp. 18 –28, jul/aug 1998.

[108] W. Zhou, Y. Zhou, X. Jiang, and P. Ning, "Detecting repackaged smartphone applications in third-party android marketplaces," in *Proceedings of the second ACM conference on Data and Application Security and Privacy*, ser. CODASPY '12. New York, NY, USA: ACM, 2012, pp. 317–326. [Retrieved: 05-Mar-2013] http://doi.acm.org/10.1145/2133601.2133640

[109] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, ser. SP '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 95–109. [Retrieved: 05-Mar-2013] http://dx.doi.org/10.1109/SP.2012.16

[110] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets," in *Proceedings of the 19th Network and Distributed System Security Symposium (NDSS)*, Feb. 2012. [Retrieved: 05-Mar-2013] http://www.csd.uoc.gr/~hy558/papers/mal_apps.pdf

[111] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "RiskRanker: scalable and accurate zero-day android malware detection," in *Proceedings of the 10th international conference on Mobile systems, applications, and services*, ser. MobiSys '12. New York, NY, USA: ACM, 2012, pp. 281–294. [Retrieved: 05-Mar-2013] http://doi.acm.org/10.1145/2307636.2307663

[112] T. Vidas, D. Votipka, and N. Christin, "All your droid are belong to us: A survey of current android attacks," in *Proceedings of the 5th USENIX Workshop on Offensive Technologies (WOOT)*. USENIX Association, 2011, pp. 10–10. [Retrieved: 05-Mar-2013] http://www.usenix.org/event/woot/tech/final_files/Vidas.pdf

[113] TCG Trusted Platform Module Work Group, "TPM Main Part 1 Design Principles," Mar 2011, specification Version 1.2 Level 2 Revision 116. [Retrieved: 05-Mar-2013] http://www.trustedcomputinggroup.org/resources/tpm_main_specification

[114] ——, "TPM Main Part 2 TPM Structures," Mar 2011, specification Version 1.2 Level 2 Revision 116. [Retrieved: 05-Mar-2013] http://www.trustedcomputinggroup.org/resources/tpm_main_specification

[115] ——, "TPM Main Part 3 Commands," Mar 2011, specification Version 1.2 Level 2 Revision 116. [Retrieved: 05-Mar-2013] http://www.trustedcomputinggroup.org/resources/tpm_main_specification

[116] A.-D. Schmidt, H.-G. Schmidt, L. Batyuk, J. H. Clausen, S. A. Camtepe, S. Albayrak, and C. Yildizli, *Smartphone malware evolution revisited: Android next target?* IEEE, Oct. 2009. [Retrieved: 05-Mar-2013] http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5403026

[117] J. Bickford, R. O'Hare, A. Baliga, V. Ganapathy, and L. Iftode, "Rootkits on smart phones: attacks, implications and opportunities," in *Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications*, ser. HotMobile '10. New York, NY, USA: ACM, 2010, pp. 49–54. [Retrieved: 05-Mar-2013] http://doi.acm.org/10.1145/1734583.1734596

[118] L. Liu, X. Zhang, G. Yan, and S. Chen, "Exploitation and threat analysis of open mobile devices," *Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems - ANCS '09*, p. 20, 2009. [Retrieved: 05-Mar-2013] http://portal.acm.org/citation.cfm?doid=1882486.1882493

[119] P. Traynor, M. Lin, M. Ongtang, and V. Rao, "On cellular botnets: Measuring the impact of malicious devices on a cellular network core," in *Proceedings of the 16th ACM conference on Computer and communications security.* ACM, 2009, pp. 223–234. [Retrieved: 05-Mar-2013] http://portal.acm.org/citation.cfm?id=1653662.1653690

[120] K. Singh, S. Sangal, N. Jain, P. Traynor, and W. Lee, "Evaluating bluetooth as a medium for botnet command and control," *Detection of Intrusions and*

Malware, and Vulnerability Assessment, pp. 61–80, 2010. [Retrieved: 05-Mar-2013] http://www.springerlink.com/index/k767q0424076444t.pdf

[121] S. Checkoway, L. Davi, A. Dmitrienko, A. Sadeghi, H. Shacham, and M. Winandy, "Return-oriented programming without returns," in *Proceedings of the 17th ACM conference on Computer and communications security*. ACM, 2010, pp. 559–572. [Retrieved: 05-Mar-2013] http://portal.acm.org/citation.cfm?id=1866370

[122] A. Egners, U. Meyer, and B. Marschollek, "Messing with android's permission model," in *Proceedings of the 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, ser. TRUSTCOM '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 505–514. [Retrieved: 05-Mar-2013] http://dx.doi.org/10.1109/TrustCom.2012.203

[123] T. Luo, H. Hao, W. Du, Y. Wang, and H. Yin, "Attacks on webview in the android system," in *Proceedings of the 27th Annual Computer Security Applications Conference*, ser. ACSAC '11. New York, NY, USA: ACM, 2011, pp. 343–352. [Retrieved: 05-Mar-2013] http://doi.acm.org/10.1145/2076732.2076781

[124] C. Orthacker, P. Teufl, S. Kraxberger, G. Lackner, M. Gissing, A. Marsalek, J. Leibetseder, and O. Prevenhueber, "Android security permissions – can we trust them?" in *Security and Privacy in Mobile Information and Communication Systems*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, R. Prasad, K. Farkas, A. Schmidt, A. Lioy, G. Russello, and F. Luccio, Eds. Springer Berlin Heidelberg, 2012, vol. 94, pp. 40–51. [Retrieved: 05-Mar-2013] http://dx.doi.org/10.1007/978-3-642-30244-2_4

[125] A. Felt and D. Wagner, "Phishing on Mobile Devices," in *W2SP 2011*, 2011. [Retrieved: 05-Mar-2013] http://www.cs.berkeley.edu/~afelt/felt-mobilephishing.pdf

[126] D. Muthukumaran, A. Sawani, J. Schiffman, B. Jung, and T. Jaeger, "Measuring integrity on mobile phone systems," in *Proceedings of the 13th ACM symposium on Access control models and technologies*. ACM, 2008, pp. 155–164. [Retrieved: 05-Mar-2013] http://portal.acm.org/citation.cfm?id=1377836.1377862

[127] U. Shankar, T. Jaeger, and R. Sailer, "Toward automated information-flow integrity verification for security-critical applications," in *Proceedings of the 2006 ISOC Networked and Distributed Systems Security Symposium*. Citeseer, 2006. [Retrieved: 05-Mar-2013] http://www.cse.psu.edu/~tjaeger/papers/ndss06.pdf

[128] K. Biba, "Integrity considerations for secure computer systems," MITRE CORP BEDFORD MA, Tech. Rep., 1977. [Retrieved: 05-Mar-2013] http://www.dtic.mil/dtic/tr/fulltext/u2/a039324.pdf

[129] W. Enck, M. Ongtang, and P. McDaniel, "Mitigating Android software misuse before it happens," Technical Report NAS-TR-0094-2008, Pennsylvania State University, Tech. Rep. November, 2008. [Retrieved: 05-Mar-2013] http://www.enck.org/pubs/NAS-TR-0094-2008.pdf

[130] H. Yin, D. Song, M. Egele, C. Kruegel, and E. Kirda, "Panorama: capturing system-wide information flow for malware detection and analysis," in *Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 2007, pp. 116–127. [Retrieved: 05-Mar-2013] http://portal.acm.org/citation.cfm?id=1315261

[131] P. Hornyack, S. Han, J. Jung, and S. Schechter, "These aren't the droids you're looking for: retrofitting android to protect data from imperious applications," in *Proceedings of the 18th ACM conference on Computer and communications security*. New York, New York, USA: ACM Press, 2011, pp. 639–652. [Retrieved: 05-Mar-2013] http://dl.acm.org/citation.cfm?doid=2046707.2046780

[132] M. Nauman, S. Khan, and X. Zhang, "Apex: extending Android permission model and enforcement with user-defined runtime constraints," in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security (ASIACCS)*. ACM, 2010, pp. 328–332. [Retrieved: 05-Mar-2013] http://portal.acm.org/citation.cfm?id=1755732

[133] M. Ongtang, K. Butler, and P. McDaniel, "Porscha: policy oriented secure content handling in Android," in *Proceedings of the 26th Annual Computer Security Applications Conference*. ACM, 2010, pp. 221–230. [Retrieved: 05-Mar-2013] http://portal.acm.org/citation.cfm?id=1920295

[134] D. Boneh and M. Franklin, "Identity-based encryption from the Weil pairing," in *SIAM Journal on Computing*, vol. 32, no. 3. Springer, 2001, pp. 213–229. [Retrieved: 05-Mar-2013] http://www.springerlink.com/index/bf5j8nhdp32pxqgy.pdf

[135] M. Dietz, S. Shekhar, Y. Pisetsky, and A. Shu, "Quire: Lightweight Provenance for Smart Phone Operating Systems," in *20th USENIX Security Symposium*, San Francisco, 2011. [Retrieved: 05-Mar-2013] http://arxiv.org/abs/1102.2445

[136] L. Xie, X. Zhang, J. Seifert, and S. Zhu, "pBMDS: a behavior-based malware detection system for cellphone devices," in *Proceedings of the third ACM conference on Wireless network security*. ACM, 2010, pp. 37–48. [Retrieved: 05-Mar-2013] http://portal.acm.org/citation.cfm?id=1741874

[137] Y. Zhou and X. Zhang, "Taming information-stealing smartphone applications (on Android)," in *Trust and Trustworthy Computing*, no. November 2009, 2011. [Retrieved: 05-Mar-2013] http://www.springerlink.com/index/K1874275275088L2.pdf

[138] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, and A.-R. Sadeghi, "XManDroid: A New Android Evolution to Mitigate Privilege Escalation Attacks," CASED: Center for Advanced Security Research Darmstadt, Darmstadt, Tech. Rep., 2011. [Retrieved: 05-Mar-2013] http://www.informatik.tu-darmstadt.de/fileadmin/user_upload/Group_TRUST/PubsPDF/xmandroid.pdf

[139] S. Bugiel, L. Davi, A. Dmitrienko, and T. Fischer, "Towards Taming Privilege-Escalation Attacks on Android," in *19th Annual Network & Distributed System Security Symposium (NDSS)*, 2012. [Retrieved: 05-Mar-2013] http://www.trust.informatik.tu-darmstadt.de/fileadmin/user_upload/Group_TRUST/PubsPDF/NDSS_2012_Towards_Taming_Privilege-Escalation_Attacks_on_Android.pdf

[140] P. Pearce, A. P. Felt, G. Nunez, and D. Wagner, "Addroid: privilege separation for applications and advertisers in android," in *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS '12. New York, NY, USA: ACM, 2012, pp. 71–72. [Retrieved: 05-Mar-2013] http://doi.acm.org/10.1145/2414456.2414498

[141] M. C. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi, "Unsafe exposure analysis of mobile in-app advertisements," in *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*, vol. 067, no. Section 2, 2012, pp. 101–112. [Retrieved: 05-Mar-2013] http://dl.acm.org/citation.cfm?id=2185464

[142] J. Cheng, S. Wong, and H. Yang, "SmartSiren: virus detection and alert for smartphones," in *Proceedings of the 5th international conference on Mobile systems, applications and services*. ACM, 2007, pp. 258–271. [Retrieved: 05-Mar-2013] http://portal.acm.org/citation.cfm?id=1247690

[143] H. Kim and J. Smith, "Detecting energy-greedy anomalies and mobile malware variants," in *Proceeding of the 6th international conference on Mobile systems, applications, and services*. ACM, 2008, pp. 239–252. [Retrieved: 05-Mar-2013] http://portal.acm.org/citation.cfm?id=1378600.1378627

[144] T. Buennemeyer, T. Nelson, L. Clagett, J. Dunning, R. Marchany, and J. Tront, "Mobile device profiling and intrusion detection using smart batteries," in *Hawaii International Conference on System Sciences, Proceedings of the 41st Annual*. IEEE, Jan. 2008, pp. 296–296. [Retrieved: 05-Mar-2013] http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4439001

[145] J. Oberheide, K. Veeraraghavan, E. Cooke, J. Flinn, and F. Jahanian, "Virtualized in-cloud security services for mobile devices," in *Proceedings of the First Workshop on Virtualization in Mobile Computing*. ACM, 2008, pp. 31–35. [Retrieved: 05-Mar-2013] http://portal.acm.org/citation.cfm?id=1622103.1629656

[146] J. Newsome and D. Song, "Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software," in *In Proceedings of the Network and Distributed System Security Symposium (NDSS 2005)*, 2005. [Retrieved: 05-Mar-2013] http://valgrind.org/docs/newsome2005.pdf

[147] A.-D. Schmidt, F. Peters, F. Lamour, C. Scheel, S. A. Camtepe, and S. Albayrak, "Monitoring Smartphones for Anomaly Detection," *Mobile Networks and Applications*, vol. 14, no. 1, pp. 92–106, Nov. 2008. [Retrieved: 05-Mar-2013] http://www.springerlink.com/index/10.1007/s11036-008-0113-x

[148] T. Bläsing, L. Batyuk, A. Schmidt, S. Camtepe, and S. Albayrak, "An Android Application Sandbox system for suspicious software detection," in *Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on*. IEEE, 2010, pp. 55–62. [Retrieved: 05-Mar-2013] http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5665792

[149] I. Burguera and U. Zurutuza, "Crowdroid: behavior-based malware detection system for Android," in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, 2011, pp. 15–25. [Retrieved: 05-Mar-2013] http://dl.acm.org/citation.cfm?id=2046619

[150] A.-D. Schmidt, R. Bye, H.-G. Schmidt, J. Clausen, O. Kiraz, K. a. Yuksel, S. a. Camtepe, and S. Albayrak, "Static Analysis of Executables for Collaborative Malware Detection on Android," *2009 IEEE International Conference on Communications*, pp. 1–5, Jun. 2009. [Retrieved: 05-Mar-2013] http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5199486

[151] E. Frank and I. H. Witten, "Generating accurate rule sets without global optimization," in *Proceedings of the Fifteenth International Conference on Machine Learning*, ser. ICML '98. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, pp. 144–151. [Retrieved: 05-Mar-2013] http://dl.acm.org/citation.cfm?id=645527.657305

[152] J. Cendrowska, "Prism: An algorithm for inducing modular rules." *International Journal of Man-Machine Studies*, vol. 27, no. 4, pp. 349–370, 1987. [Retrieved: 05-Mar-2013] http://dx.doi.org/10.1016/S0020-7373(87)80003-2

[153] M. Nauman, S. Khan, X. Zhang, and J. Seifert, "Beyond Kernel-Level Integrity Measurement: Enabling Remote Attestation for the Android Platform," *Trust and Trustworthy Computing*, pp. 1–15, 2010. [Retrieved: 05-Mar-2013] http://www.springerlink.com/index/7646533475760130.pdf

[154] TCG Trusted Network Connect Work Group, "TNC IF-MAP Metadata for Network Security, Version 1.1, Revision 8," Trusted Computing Group, May 2012. [Retrieved: 05-Mar-2013] http://www.trustedcomputinggroup.org/resources/tnc_ifmap_metadata_for_network_security

[155] ——, "TNC IF-MAP Metadata for ICS Security, Version 1.0, Revision 39," Trusted Computing Group, October 2012. [Retrieved: 05-Mar-2013] http://www.trustedcomputinggroup.org/resources/tnc_ifmap_metadata_for_ics_security

[156] E. Rescorla, "HTTP Over TLS," RFC 2818 (Informational), Internet Engineering Task Force, May 2000, updated by RFC 5785. [Retrieved: 05-Mar-2013] http://www.ietf.org/rfc/rfc2818.txt

[157] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246 (Proposed Standard), Internet Engineering Task Force, Aug. 2008, updated by RFCs 5746, 5878, 6176. [Retrieved: 05-Mar-2013] http://www.ietf.org/rfc/rfc5246.txt

[158] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication," RFC 2617 (Draft Standard), Internet Engineering Task Force, Jun. 1999. [Retrieved: 05-Mar-2013] http://www.ietf.org/rfc/rfc2617.txt

[159] M. Boisot and A. Canals, "Data, information and knowledge: have we got it right?" *Journal of Evolutionary Economics*, vol. 14, no. 1, pp. 43–67, Jan. 2004. [Retrieved: 05-Mar-2013] http://dx.doi.org/10.1007/s00191-003-0181-9

[160] "Code Conventions for the Java TM Programming Language," Oracle Corporation, Apr 1999. [Retrieved: 05-Mar-2013] http://www.oracle.com/technetwork/java/javase/documentation/codeconvtoc-136057.html

[161] L. Svobodova, "Client/server model of distributed processing," in *Proceedings on Kommunikation in Verteilten Systemen I.* London, UK, UK: Springer-Verlag, 1985, pp. 485–498. [Retrieved: 05-Mar-2013] http://dl.acm.org/citation.cfm?id=645663.664729

[162] M. Fontoura, S. Sadanandan, J. Shanmugasundaram, S. Vassilvitski, E. Vee, S. Venkatesan, and J. Zien, "Efficiently evaluating complex boolean expressions," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, ser. SIGMOD '10. New York, NY, USA: ACM, 2010, pp. 3–14. [Retrieved: 05-Mar-2013] http://doi.acm.org/10.1145/1807167.1807171

[163] TCG Trusted Network Connect Work Group, "TNC IF-MAP Binding for SOAP, Version 2.0, Revision 47," Trusted Computing Group, May 2012. [Retrieved: 05-Mar-2013] http://www.trustedcomputinggroup.org/resources/tnc_ifmap_binding_for_soap_specification

[164] W3C, "Extensible Markup Language (XML) 1.0 (Fifth Edition)," Feb. 2008. [Retrieved: 05-Mar-2013] http://www.w3.org/TR/2008/PER-xml-20080205/

[165] ——, "XML Schema Part 0: Primer Second Edition," Oct. 2004. [Retrieved: 05-Mar-2013] http://www.w3.org/TR/xmlschema-0/

[166] K. McCloghrie, D. Perkins, and J. Schoenwaelder, "Structure of Management Information Version 2 (SMIv2)," RFC 2578 (Standard), Internet Engineering Task Force, Apr. 1999. [Retrieved: 05-Mar-2013] http://www.ietf.org/rfc/rfc2578.txt

[167] Internet Assigned Numbers Authority (IANA), "PRIVATE ENTERPRISE NUMBERS," Dec. 2012. [Retrieved: 05-Mar-2013] http://www.iana.org/assignments/enterprise-numbers

[168] Trust@FHH Research Group, "Trust@FHH Website." [Retrieved: 05-Mar-2013] http://www.trust.inform.fh-hannover.de

[169] ——, "Trust@FHH GitHub Account." [Retrieved: 05-Mar-2013] https://github.com/trustatfhh

[170] TCG, "Trusted Computing Group Offers Security Assurances for Systems and Networks with Certified Products," News on Website, Nov. 2012. [Retrieved: 05-Mar-2013] https://www.trustedcomputinggroup.org/media_room/news/269

[171] W3C, "Document Object Model (DOM)." [Retrieved: 05-Mar-2013] http://www.w3.org/DOM/

[172] J. F. Kenney and E. S. Keeping, *Linear Regression and Correlation.* Van Nostrand, 1964, Ch. 15 in Mathematics of Statistics, Pt. 1, 3rd ed. Princeton, NJ.

[173] T. Ruhe, "Development of an Android Usage Study System," Master's Thesis, Hochschule Hannover, Sep. 2012.

[174] Trust@FHH Research Group, "Android Usage Study Project Website." [Retrieved: 05-Mar-2013] https://trust.inform.fh-hannover.de/trust_redmine/projects/android-usage-study-2012

[175] R. Steuerwald, "Integration von OpenVAS in IF-MAP," Bachelor's Thesis, Hochschule Hannover, Jun. 2011.

[176] Trust@FHH Research Group, "How to setup and use ironvas." [Retrieved: 05-Mar-2013] https://trust.inform.fh-hannover.de/trust_redmine/projects/iron/wiki/How_to_setup_and_use_ironvas

[177] J. W. Tukey, *Exploratory Data Analysis*. Addison-Wesley, 1977.

[178] S. Viehböck, "Brute forcing Wi-Fi Protected Setup," Dec. 2012. [Retrieved: 05-Mar-2013] http://sviehb.files.wordpress.com/2011/12/viehboeck_wps.pdf

[179] "That square QR barcode on the poster? Check it's not a sticker," News on Website, The Register, Dec. 2012. [Retrieved: 05-Mar-2013] http://www.theregister.co.uk/2012/12/10/qr_code_sticker_scam/

[180] "Federal Data Protection Act (Bundesdatenschutzgesetz - as of 1 September 2009)." [Retrieved: 05-Mar-2013] http://www.bfdi.bund.de/EN/DataProtectionActs/Artikel/BDSG_idFv01092009.pdf?___blob=publicationFile

[181] "Commission proposes a comprehensive reform of data protection rules to increase users' control of their data and to cut costs for businesses," Jan. 2012. [Retrieved: 05-Mar-2013] http://europa.eu/rapid/press-release_IP-12-46_en.htm?locale=en

[182] I. Bente, G. Dreo, B. Hellmann, J. Vieweg, and J. von Helden, "TCADS: Trustworthy, Context-related Anomaly Detection for Smartphones," *Presented at the 15th International Conference on Network-Based Information Systems (NBiS 2012)*, 2012, Best Paper Award.

[183] I. Bente, B. Hellmann, T. Rossow, J. Vieweg, and J. von Helden, "On Remote Attestation for Google Chrome OS," *Presented at the 15th International Conference on Network-Based Information Systems (NBiS 2012)*, 2012.

[184] R. Marx, N. Kuntze, C. Rudolph, I. Bente, and J. Vieweg, "Trusted Service Access with Dynamic Security Infrastructure Configuration," *Presented at the 18th Asia-Pacific Conference on Communications (APCC 2012)*, 2012.

[185] I. Bente, G. Dreo, B. Hellmann, J. Vieweg, and J. von Helden, "Trustworthy Anomaly Detection for Smartphones," *Poster presented at the 13th Workshop on Mobile Computing Systems and Applications (HotMobile 2012)*, 2012.

[186] K.-O. Detken, D. Scheuermann, I. Bente, and J. Westerkamp, "Automatisches Erkennen mobiler Angriffe auf die IT-Infrastruktur," in *D.A.CH Security 2012: Bestandsaufnahme, Konzepte, Anwendungen und Perspektiven*, P. Schartner and J. Taeger, Eds. syssec Verlag, 2012.

[187] I. Bente, G. Dreo, B. Hellmann, S. Heuser, J. Vieweg, J. Helden, and J. Westhuis, "Towards Permission-Based Attestation for the Android Platform," in *Trust and Trustworthy Computing*, ser. Lecture Notes in Computer Science, J. McCune, B. Balacheff, A. Perrig, A.-R. Sadeghi, A. Sasse, and Y. Beres, Eds. Springer Berlin Heidelberg, 2011, vol. 6740, pp. 108–115. [Retrieved: 05-Mar-2013] http://dx.doi.org/10.1007/978-3-642-21599-5_8

[188] K.-O. Detken, D. Dunekacke, and I. Bente, "Konsolidierung von Metadaten zur Erhöhung der Unternehmenssicherheit," in *D.A.CH Security 2011: Bestandsaufnahme, Konzepte, Anwendungen und Perspektiven*, P. Schartner and J. Taeger, Eds. syssec Verlag, 2011.

[189] N. Kuntze, C. Rudolph, I. Bente, J. Vieweg, and J. von Helden, "Interoperable device identification in Smart-Grid environments," in *Power and Energy Society General Meeting, 2011 IEEE*, july 2011, pp. 1 –7.

[190] I. Bente, J. Vieweg, and J. Helden, "Towards Trustworthy Networks with Open Source Software," in *Horizons in Computer Science Research*, T. S. Clary, Ed. Nova Publishers, 2011, vol. 3.

[191] I. Bente, B. Hellmann, J. Vieweg, J. von Helden, and A. Welzel, "Interoperable remote attestation for VPN environments," in *Proceedings of the Second international conference on Trusted Systems*, ser. INTRUST'10. Berlin,

Heidelberg: Springer-Verlag, 2011, pp. 302–315. [Retrieved: 05-Mar-2013] http://dx.doi.org/10.1007/978-3-642-25283-9_20

[192] I. Bente, J. Vieweg, and J. Helden, "ESUKOM: Smartphone Security for Enterprise Networks," in *ISSE 2011 Securing Electronic Business Processes*, N. Pohlmann, H. Reimer, and W. Schneider, Eds. Vieweg + Teubner Verlag | Springer Fachmedien Wiesbaden GmbH, 2012.

[193] ——, "Countering Phishing with TPM-bound Credentials," in *ISSE 2010 Securing Electronic Business Processes*, N. Pohlmann, H. Reimer, and W. Schneider, Eds. Vieweg+Teubner, 2011, pp. 236–246. [Retrieved: 05-Mar-2013] http://dx.doi.org/10.1007/978-3-8348-9788-6_23

[194] I. Bente, J. Vieweg, J. von Helden, M. Jungbauer, and N. Pohlmann, "tNAC - Trusted Network Access Control," *Poster presented at the 19th Usenix Security Symposium (USENIX '10)*, 2010.

[195] I. Bente, J. Vieweg, and J. Helden, "Privacy Enhanced Trusted Network Connect," in *Trusted Systems*, ser. Lecture Notes in Computer Science, L. Chen and M. Yung, Eds. Springer Berlin Heidelberg, 2010, vol. 6163, pp. 129–145. [Retrieved: 05-Mar-2013] http://dx.doi.org/10.1007/978-3-642-14597-1_8

[196] I. Bente and J. Helden, "Towards Trusted Network Access Control," in *Future of Trust in Computing*, D. Gawrock, H. Reimer, A.-R. Sadeghi, and C. Vishik, Eds. Vieweg+Teubner, 2009, pp. 157–167. [Retrieved: 05-Mar-2013] http://dx.doi.org/10.1007/978-3-8348-9324-6_17

[197] J. Helden and I. Bente, "Towards real Interoperable, real Trusted Network Access Control: Experiences from Implementation and Application of Trusted Network Connect," in *ISSE 2008 Securing Electronic Business Processes*, N. Pohlmann, H. Reimer, and W. Schneider, Eds. Vieweg+Teubner, 2009, pp. 152–162. [Retrieved: 05-Mar-2013] http://dx.doi.org/10.1007/978-3-8348-9283-6_16

List of Figures

List of Tables

List of Listings

Glossary

ADB Android Debug Bridge. 194, 232

AES Advanced Encryption Standard. 39

AOSP Android Open Source Project. 33

API application programming interface. 31, 35

ASCII American Standard Code for Information Interchange. 91, 93

ASLR address space layout randomization. 33, 38

BSSID Basic Service Set Identification. 184, 235

BYOD Bring Your Own Device. 3, 27, 220

DAC discretionary access control. 38

DDoS Distributed Denial of Service. 58

DHCP Dynamic Host Configuration Protocol. 12

DMZ Demilitarized Zone. 11, 13, 14, 201

DNS Domain Name System. 13

DoS Denial of Service. 47, 58

GPS Global Positioning System. 16, 31, 93

ICC inter-component communication. 37, 40

ICMP Internet Control Message Protocol. 202

IDC International Data Corporation. 2, 32

IDS Intrusion Detection System. 13, 64, 155, 157, 172, 173, 225

IMEI International Mobile Station Equipment Identity. 150, 155, 169–171, 174

IPsec Internet Protocol Security. 14

JavaCC Java Compiler Compiler. 166

JDK Java Development Kit. 35

JNI Java Native Interface. 35, 57

MAC Media Access Control. 15

MITM Man-in-the-Middle. 54

MMS Multimedia Messaging Service. 63, 64, 66

MNO mobile network operator. 30

NAC Network Access Control. 14, 18, 172

NFC near field communication. 31, 53, 231

NIST National Institute of Standards and Technology. 13

OSI Open Systems Interconnection. 13, 14

PDK Platform Development Kit. 34

PDP Policy Decision Point. 156, 157, 163, 164, 172, 200

PIN personal identification number. 39, 57

PSN Sony PlayStation Network. 1

RADIUS Remote Authentication Dial In User Service. 13

RIM Research in Motion. 32

SIIS Systems and Internet Infrastructure Security Laboratory. 60, 62

SIM subscriber identity module. 30, 47, 233

SMI Structure of Management Information. 146, 149

SMS Short Message Service. 30, 37, 41, 47, 53, 55, 58, 62–68, 119, 236

SSID Service Set Identification. 184, 235

TCB Trusted Computing Base. 24

TCG Trusted Computing Group. 13, 14, 70, 103, 146, 147, 161, 190

TCP Transmission Control Protocol. 15

TNC Trusted Network Connect. 13, 14, 103

TPM Trusted Platform Module. 57, 71, 81

UDP User Datagram Protocol. 15

UML Unified Modeling Language. 20, 120, 122, 125, 126

USB Universal Serial Bus. 194

VLAN Virtual Local Area Network. 12, 19

VPN Virtual Private Network. 12, 14, 15, 18, 201, 231

WAP Wireless Access Point. 12, 13, 15, 16, 19, 179, 184, 187, 201, 208

WLAN Wireless Local Area Network. 12, 31

XML Extensible Markup Language. 38, 144