Security of Smartphones

at the Dawn of their Ubiquitousness

Inauguraldissertation
zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
der Universität Mannheim

vorgelegt von

Michael Becher

aus Kleve, Deutschland

Mannheim, 2009

Dekan:	Prof. Dr.-Ing. Felix Freiling, Universität Mannheim
Referent:	Prof. Dr.-Ing. Felix Freiling, Universität Mannheim
Korreferent:	Prof. Dr.-Ing. Ulrike Meyer, RWTH Aachen

Tag der mündlichen Prüfung: 09.10.2009

Abstract

The importance of researching in the field of *smartphone security* is substantiated in the increasing number of smartphones, which are expected to outnumber common computers in the future. Despite their increasing importance, it is unclear today if mobile malware will play the same role for mobile devices as for common computers today. Therefore, this thesis contributes to defining and structuring the field mobile device security with special concern on smartphones and on the *operational side* of security, i.e., with mobile malware as the main attacker model. Additionally, it wants to give an understanding of the *shifting boundaries* of the attack surface in this emerging research field.

The first three chapters introduce and structure the research field with the main goal of showing what has to be defended against today. Besides introducing related work they structure mobile device attack vectors with regard to mobile malicious software and they structure the topic of mobile malicious software itself with regard to its portability.

The technical contributions of this thesis are in Chapters 5 to 8, classified according to the location of the investigation (on the device, in the network, distributed in device and network). Located in the device is MobileSandbox, a software for dynamic malware analysis. As another device-centric contribution we investigate on the efforts that have to be taken to develop an autonomously spreading smartphone worm. The results of these investigations are used to show that device-centric parts are necessary for smartphone security. Additionally, we propose a novel device-centric security mechanism that aims at reducing the attack surface of mobile devices to mobile malware.

The network-centric investigations show the possibilities that a mobile network operator can use in its own mobile network for protecting the mobile devices of its clients. We simulate the effectiveness of different security mechanisms.

Finally, the distributed investigations show the feasibility of distributed computation algorithms with security modules. We give prototypic implementations of protocols for secure multiparty computation as a modularized version with failure detector and consensus algorithms, and for fair exchange with guardian angels.

Zusammenfassung

Die Wichtigkeit der Forschung zur *Sicherheit von multifunktionalen Mobiltelefonen* (engl. “smartphones”) wird untermauert von ihrer steigenden Anzahl, die vermutlich die Anzahl von herkömmlichen Rechnern in Zukunft übersteigen wird. Trotz ihrer steigenden Bedeutung ist es unklar, ob mobile Schadprogramme (engl. “mobile malware”) dieselbe Rolle für mobile Endgeräte spielen werden, wie es herkömmliche Schadprogramme für herkömmliche Rechner tun. Aus diesem Grund leistet diese Arbeit einen Beitrag dazu, das Thema Sicherheit mobiler Endgeräte zu definieren und zu strukturieren. Insbesondere behandelt sie multifunktionale Mobiltelefone und die *operative Seite* der Sicherheit, die mobile Schadprogramme als das hauptsächliche Angriffsmodell hat. Die Arbeit möchte zusätzlich ein Verständnis von den *sich verschiebenden Grenzen* der Angriffsvektoren in diesem neuen Forschungsfeld geben.

Die ersten drei Kapitel führen in das Themenfeld ein, strukturieren es, und zeigen, wogegen man sich heute schützen muss. Neben der Nennung von verwandten Arbeiten werden Angriffsvektoren für mobile Endgeräte in Bezug auf mobile Schadprogramme, sowie mobile Schadprogramme in Bezug auf ihre Portabilität strukturiert.

Die technischen Beiträge dieser Arbeit folgen in den Kapiteln 5 bis 8, aufgeteilt durch den Ort der Untersuchung (auf dem Endgerät, im Netz, verteilt zwischen Endgerät und Netz). Im Endgerät angesiedelt ist MobileSandbox, ein Programm für die dynamische Analyse von Schadprogrammen. Ein weiterer Beitrag bezogen auf das Endgerät ist die Untersuchung der Aufwände für die Entwicklung eines sich autonom verbreitenden Wurms auf multifunktionalen Mobiltelefonen. Die Ergebnisse dieser Untersuchungen werden benutzt, um endgerätezentrierte Teile als für die Sicherheit von multifunktionalen Mobiltelefonen notwendig zu zeigen. Außerdem wird ein neuartiger endgerätezentrierter Sicherheitsmechanismus vorgestellt, der die Angriffsfläche der Endgeräte gegenüber mobilen Schadprogrammen reduziert.

Die netzzentrierten Untersuchungen zeigen die Möglichkeiten, die ein Mobilfunknetzbetreiber in seinem Mobilfunknetz benutzen kann, um die mobilen Endgeräte seiner Kunden zu schützen. Die Effektivität von verschiedenen Sicherheitsmechanismen wird simuliert.

Schließlich zeigen die verteilten Untersuchungen die Durchführbarkeit von Sicherheitsmodulen benutzenden Algorithmen zur verteilten Berechnung. Es werden vorgestellt: eine prototypische Implementierung von Protokollen zur sicheren verteilten Berechnung in einer modularisierten Version mit Ausfallerkennern und Konsensusalgorithmen. Außerdem eine Implementierung von fairem Austausch mit Sicherheitsmodulen.

Acknowledgments

A work like this cannot be done without the support of numerous people.

Thanks go to all the employees of Deutsche Telekom who supported me with their time for discussions, with hardware, and with insights into the mobile world and into my research field. Dr. Thomas Breitbach for his trust in me and for enabling the funding during my work on this thesis. Special thanks go to Dr. Uwe Wilhelm for being my all-purpose contact at Deutsche Telekom during all the years.

My proofreaders have contributed to an understandable text. Cordial thanks for carefully reading and commenting on every line of the draft chapters that I sent them: Alexander Becher, Andreas Sensen, Philipp Vorst. With valuable suggestions from their different perspectives they helped me a lot to refine my arguments and to make the text more understandable for a general computer science audience.

Additional thanks go to the scientific world. To all my students from Aachen and Mannheim for their interest in the world of smartphone security and for choosing me as their supervisor. To all the people in Mannheim who gave the work on this thesis also a social component. To my referee Prof. Dr. Ulrike Meyer, who carefully read the thesis and gave valuable feedback to me that has formed this final version of the thesis. And to my supervisor Prof. Dr. Felix Freiling, who also carefully commented on the chapters of this thesis, who gave me support in every phase of my scientific work, and who always shared his insights into science.

My family receives the most cordial thanks. They all know that this thesis would neither have started nor finished without them.

Contents

1 Introduction **1**

2 Related Work **9**
2.1 Mobile Devices 10
2.1.1 Definition 10
2.1.2 Security Modules 11
2.1.3 Specifics of Mobile Devices 15
2.2 Mobile Device Security 20
2.2.1 Definition of Security 21
2.2.2 Security in Application Frameworks 22
2.2.3 Security in Windows Mobile 24
2.2.4 Security in Symbian OS 29
2.3 Mobile Malicious Software 34
2.3.1 Definition of Malware 34
2.3.2 Surveys of Mobile Malware 35
2.3.3 Virus Scanners 37
2.4 Analytic Mechanisms 40
2.4.1 Dynamic Software Analysis 41
2.4.2 Simulation 42
2.5 Distributed Computation 43
2.5.1 Secure Multiparty Computation 44
2.5.2 Fair Exchange 47

3 Structuring Mobile Device Attack Vectors **51**
3.1 Attack Vector Classes 52
3.2 Hardware-Centric Attacks 53
3.2.1 Intercepting MNO Smartcard Communication 55
3.2.2 Attacking the Device 55
3.3 Device-Independent Attacks 56
3.3.1 Wireless Transmission Security 57
3.3.2 Backend Systems 61

3.4 Software-Centric Attacks 62
3.4.1 General 62
3.4.2 Operating System Modifications 64
3.4.3 Web Browser 66
3.5 "Layer 8": The User as Attack Vector 68
3.5.1 Security Awareness 69
3.5.2 Influence of the User 70
3.5.3 Social Engineering 71
3.5.4 Security & Usability 72
3.6 Conclusion 75

4 Structuring Mobile Malicious Software 77
4.1 Known Mobile Malware 78
4.1.1 Windows Type of Operating Systems 78
4.1.2 Symbian OS 81
4.1.3 Java Platform, Micro Edition (J2ME) 86
4.2 Phases of Malware 87
4.2.1 Infection 87
4.2.2 Malicious Functionality 88
4.2.3 Spreading 90
4.3 Mobile Malware Portability 90
4.3.1 Cross-Platform Malware 91
4.3.2 Portability between Mobile Operating Systems 93
4.4 Conclusion 95

5 Developing a Dynamic Malware Analysis Tool 97
5.1 Design 98
5.1.1 General Design Considerations 98
5.1.2 Environment 99
5.1.3 Logging 100
5.1.4 Analysis Duration 100
5.2 Implementation 100
5.2.1 Components of MobileSandbox 100
5.2.2 Prolog and Epilog 102
5.2.3 User-Level Hooking 102
5.2.4 Kernel-Level Hooking 104
5.2.5 Portability 107
5.3 Evaluation 107
5.3.1 Performance 108
5.3.2 Completeness 108
5.3.3 Malware Analyses 109

5.4 Conclusion 114

6 **Device-Centric Security Investigations** **115**
6.1 Developing a Smartphone Worm 116
6.1.1 Proof-of-Concept Malware for Mobile Devices 117
6.1.2 Constant Part: The Building Blocks 119
6.1.3 Variable Part: Measuring Resistance 123
6.2 Attack Surface Reduction in Mobile Device Security 125
6.2.1 Justifying Increased User Control 126
6.2.2 Policy Enforcement and Security Interfaces 130
6.2.3 Prototype Design 131
6.3 Conclusion 134

7 **Network-Centric Security Investigations** **135**
7.1 Mobile Dynamic Malware Analysis 136
7.1.1 Collecting Samples 136
7.1.2 Analyzing Samples 140
7.1.3 Responding to the Analysis 140
7.2 Simulation 142
7.2.1 Model 142
7.2.2 Implementation 144
7.2.3 Evaluation 145
7.3 Conclusion 148

8 **Distributed Security Investigations** **149**
8.1 Distributed Computation in Real-World Environments 150
8.1.1 Mobile Network Setting 150
8.1.2 Virtual Trusted Third Parties 152
8.2 Secure Multiparty Computation with Security Modules 155
8.2.1 Design 155
8.2.2 Implementation 156
8.2.3 Memory Requirements Evaluation 159
8.3 Fair Exchange with Security Modules 160
8.3.1 Design 160
8.3.2 Implementation 163
8.3.3 Evaluation 164
8.4 Conclusion 166

9 **Conclusion** **169**
9.1 Summary 169
9.2 Discussion 170

9.2.1 Future Entities 170
9.2.2 Security Requirements 171
9.3 Perspective 172
9.3.1 Future Validity of the Specifics 172
9.3.2 Future Challenges 173

Bibliography **175**

List of Figures

2.1 Structure of Command APDUs 15
2.2 Specifics of Mobile Devices 16
2.3 Vulnerability Lifetime 17
2.4 System Call Architecture of Windows Mobile 26
2.5 Operating Systems Targeted by Mobile Malware 36

3.1 Mobile Device Attack Vectors (High-Level) 53
3.2 Mobile Device Attack Vectors (Incarnations) 54
3.3 Web Browser Undermining the Security Model 67

4.1 Cross-Platform Malware 91

5.1 Structure of MobileSandbox 101
5.2 Import Address Table Patching 103
5.3 Kernel-Level System Call Hooking 105
5.4 Dust Source Code (Excerpt) 110
5.5 Analysis of Dust (Excerpt) 111
5.6 Analysis of Pmcryptic (Excerpt) 112

6.1 Exploitation with Staged Shellcode 120
6.2 Infected Device 122
6.3 Measuring Resistance 124
6.4 Policy Enforcer in the Mobile Device 127
6.5 Policy Enforcer System Architecture 132

7.1 File Collection Vectors 137
7.2 Simulation System Design 143
7.3 Applying Virus Scanners vs. User Education 146
7.4 Effects of User Education 147
7.5 Virus Scanner Update Times 147

8.1 System Architecture of SMC Solution 155

8.2 Application Protocol Data Units of SMC Solution 157
8.3 Content of Encrypted Messages 158
8.4 Design of Fair Exchange Solution 162

List of Tables

2.1 Access Rules for Data Caging 31

3.1 Personal Security Profile 74

4.1 Known Mobile Malware 79
4.2 Mobile Malware Portability 94

5.1 Performance of MobileSandbox 108

7.1 Detected Bluetooth Services 139

8.1 Memory Requirements of Failure Detector 159
8.2 Approximated Memory Requirements Java Card 159
8.3 Fair Exchange Adversaries 165
8.4 Fairness Evaluation Using InformedRandomAdversary 165
8.5 Average Execution Times Depending on Cryptography 166

Chapter 1

Introduction

Motivation

The goal of this thesis is contributing to the topic of *smartphone security*. This topic covers all mechanisms that are intended to increase the security of sophisticated mobile devices—called smartphones. Besides a connection to mobile phone networks, smartphones can be characterized as mobile devices having a large screen, reasonable processing power and memory, and an operating system that is extensible with third-party software.

The beginning of the smartphone era can be seen as beginning with the new millennium. Since then, numerous articles have been written about the topic of smartphone security and the potential of malicious software on smartphones. A recent study expected "that by the end of 2007, enough factors will have come together that the risk of mobile attacks will be much greater. Those factors include less heterogeneity in operating systems, more penetration of smartphones and a greater incidence of people actually accepting downloads and sending executables to one another on mobile devices, [...]" [115].

There are more articles that try to give a statement on the future of smartphone security, e.g., "The wireless epidemic" in 2007 [114], "Is it finally time to worry about mobile malware?" in 2008 [119], and others from 2000 to 2007 [120, 75, 42, 121, 100, 101].

What do all these statements mean? They mean that experts are expecting a major security incident with mobile phones ever since these devices began to become more powerful: with increased processing power and memory, increased data transmission capabilities of the mobile phone networks, and with open and third-party extensible operating systems. However, such an incident has not happened

until the time of this writing in 2009. The reasons are unclear, but the heterogeneity of mobile operating systems could be a reason. Contrary to the prediction quoted above, heterogeneity of mobile operating systems has even increased. Besides the operating systems Windows Mobile and Symbian OS, the mobile world has seen the advent of the iPhone OS and the Linux-based Android operating system during the last two years. Despite of their young age, both operating systems already gained their market share and they are predicted to even increase this market share [28]. Moreover, the view that mobile operating systems are sufficiently secure today [26] might be another reason why no major security incident has happened until now.

Smartphones and mobile phones in general have particular specifics. A first unique feature is the availability of trusted modules in every mobile device: the subscriber identity module (SIM card). This module gives additional possibilities for security mechanisms, as Chapter 8 will show. It is a deployed and currently available basis for a key infrastructure, which is a difficult problem for common computers. A second unique feature is the more centralized infrastructure that facilitates the network-centric solutions of Chapter 7. More of these differences between mobile phones and common computers can be seen in Chapter 2.

This thesis deals with smartphone security that is a subset of the broader topic mobile security. In particular, the thesis covers two aspects of smartphone security:

1. Analyzing, writing, and contributing to the defense against mobile malicious software (malware).

2. The feasibility of secure computation on mobile devices. These main contributions are placed into the more general context of mobile device security in Chapter 3.

Mobile Malicious Software Investigations. We investigate device-centric security mechanisms, network-centric security mechanisms, and something in-between: distributed security mechanisms. We aim at showing how the local view of these single security mechanisms can be brought together in a global view that the mobile device user accepts and is able to understand. As a basis of its investigations we see every real-world parameter as changeable, especially those that only the mobile network operator is able to change.

Mobile malware covers malware for mobile phones and smartphones in particular. Malware is the usual form in which vulnerabilities and the exploits of an operating system or an application manifest themselves to become security problems for a broad audience. Examples of malware are viruses, worms, and Trojan horses.

Investigating the damage potential of mobile malicious software is challenging today because this new kind of malware has the potential to have mobile phone users question the trust in the mobile telephony system as such.

Therefore, we see the main research tasks for mobile device security in the attacks that can be committed by mobile malicious software. The mobile device can be seen as a *Byzantine* node, i.e., having arbitrary and possibly malicious behavior. We call this the *operational side* of security. We will focus on mobile devices that work in today's mobile phone networks, that is, we cover mobile phones containing a smartcard that is controlled by the mobile network operator. This smartcard is used for regaining control over the otherwise Byzantine mobile devices: it is assumed that mobile malware cannot spread onto the smartcard.

Feasibility of Secure Multiparty Computation on Mobile Devices. This leads to the second area of contributions: the investigation of proactive mechanisms to reduce the influence that malicious software can have on smartphones. These mechanisms are distributed computation protocols that use a trusted module on the device of a participant to build a trusted subnet and to relieve the need for a trusted third party. The protocols are secure multiparty computation and fair exchange.

Secure multiparty computation is a hard problem. Informally, it is defined as: "A set of parties wants to compute a common function F on their local inputs, while keeping their local data as private as possible, but who do not trust each other, nor the communication channels" [37]. Fair exchange is defined as: two or more mutually untrusted parties want to exchange secrets in such a way that either all of them receive the desired information or none of them learns anything valuable. It is challenging to solve these problems on smartphones because of their limited resources compared to current implementations of these protocols.

In summary, we can take an approach in this thesis without being forced into specific parameters by real mobile device security incidents. The importance of the research topic is substantiated in its investigation subjects, which are becoming ubiquitous and which are expected to outnumber common computers in the future. As a final remark, we take the attacker point of view most of the times, which proved to be a valuable point of view for information security [79].

Contributions

The contributions in this thesis can be grouped into three areas. The first area is non-technical, the second area is about mobile malware (called *device-centric investigations* and *network-centric investigations*), and the third area is about

the feasibility of distributed computation algorithms on mobile devices (called *distributed investigations*).

One part of the device-centric investigations is MobileSandbox, a software for dynamic malware analysis of Windows Mobile executables. Another device-centric investigation is researching on the efforts that have to be taken to develop an autonomously spreading smartphone worm. The results of these investigations are used to show that device-centric parts are necessary for smartphone security and we propose a novel device-centric security mechanism that aims at reducing the attack surface of mobile devices to mobile malware. The network-centric investigations show the possibilities that a mobile network operator has to use its own mobile network for protecting the mobile devices of its clients. We simulate the effectiveness of different security mechanisms.

The distributed investigations show the feasibility of distributed computation algorithms with security modules. We give prototypic implementations of protocols for secure multiparty computation as a modularized version with failure detector and consensus algorithms, and for fair exchange with guardian angels.

In detail, the contributions of this thesis are:

- We give two non-technical contributions for structuring of the topic *smartphone security* as a conceptual framework. First, an attack model with four attack classes that argues for a clear distinction between its classes (Chapter 3), presenting in detail what we call the *operational side* of mobile device security, that is, vulnerabilities exploitable by mobile malware. Second, a survey on the current state of real-world examples and a projection of the potential of mobile malicious software, together with a classification concerning portability between different platforms with the main goal of showing what has to be defended against today (Chapter 4). It shows that even if most of today's mobile malware targets Symbian OS, most of these pieces of malware are portable to other mobile operating systems.

- We contribute a software for dynamic malware analysis of Windows Mobile binaries (Chapter 5, the core parts are already published [48]). This software has advantages over existing solutions: it solves the problem of logging a particular run of a Windows Mobile software sample for the first time. Most of the work in this area of dynamic analysis has been done for common Windows systems, and this chapter will point out why their approaches cannot be transferred to Windows Mobile. Either they use processor emulators with sophisticated interfaces (TTAnalyze [17]) or they use DLL overwriting techniques (CWSandbox [232]) that cannot be used for systems like Windows Mobile because they execute DLLs directly in ROM ("execute in place",

see Section 2.2.3). Abstracted from the different target operating systems, MobileSandbox has two conceptual advantages over the two solutions from above. First, it logs system calls not only at user level (CWSandbox) but even at the level of the kernel, enabling a more detailed system call log. Second, it can be integrated into a running device without any changes to the firmware of the device.

- As there are currently no incarnations of smartphone worms, we contribute on the efforts that have to be taken to develop an autonomously spreading smartphone worm for Windows Mobile by actively researching characteristics and countermeasures to learn more about its associated threats. The results are embedded into a three phases model of breaking a system and are used to derive a cost-to-break metric for Windows Mobile (Section 6.1, the results are already published [20]). This investigation shows that it is possible to come very close to the target of developing a smartphone worm with reasonable effort.

- We show device-centric parts as necessary for smartphone security, because some important security requirements can only be implemented on the mobile device itself. We use the MobileSandbox dynamic software analysis tool of Chapter 5 as a basis for a novel device-centric security mechanism—called the *policy enforcer*—that aims at reducing the attack surface of mobile devices concerning mobile malware. This security mechanism differs from recent related work in the area by relieving the need to be added to the device at manufacturing time (Section 6.2).

- Chapter 7 investigates the possibilities of increasing mobile device security within the mobile network itself. It uses discrete event simulation to simulate the effectiveness of various network-centric security mechanisms.

- Finally, we show the feasibility of distributed computation algorithms with security modules by giving prototypic implementations of protocols for secure multiparty computation in a modularized version and for fair exchange (Chapter 8). The implementation of secure multiparty computation is the first for more than two participants in an environment that is not synchronous. We show that this implementation only needs minimal resources compared to other systems for solving secure multiparty computation. The implementation of the probabilistic fair exchange with guardian angels protocol is shown to inherit the fairness properties of the underlying protocol by being resilient against a large number of attacker classes.

Besides its key contributions, this thesis wants to convey the following insights: in the fast-changing world of mobile device security, the boundaries of security

are constantly changing, e.g., with application frameworks and with mobile Web browsers. This makes it necessary to reduce the attack surface of the operating systems of mobile devices. Moreover, this thesis shows that many insights from security research remain valid in the context of smartphone security. This is especially true for the combination of systems that are secure for themselves but whose combination can lead to unexpected vulnerabilities.

Organization

We make a clear separation between related work and our own contributions in this thesis. Therefore, the preliminary knowledge of a chapter usually is not in direct vicinity, but can be found in Chapter 2. Besides reading this thesis from beginning to end it is possible to select particular chapters. For the latter case, the texts contains back references to the sections where a topic was introduced.

The first three chapters introduce and structure the topic with the main goal of showing what has to be defended against today. The following technical contributions are classified according to the location of the investigation: on the device, in the network, distributed in device and network.

The individual chapters are organized as follows:

- Chapter 2 introduces related work that is a necessary basis for understanding the solutions of this thesis. It is subdivided into the introduction of mobile devices, mobile device security, mobile malicious software, analytic mechanisms, and distributed computation algorithms.

- Chapter 3 defines what this thesis calls the *operational side* of mobile device security by structuring current mobile device attack vectors into the four classes hardware-centric attacks, device-independent attacks, software-centric attacks, and user-layer attacks. Special attention is on attack vectors that can be used by mobile malware.

- Malicious software for mobile devices is covered in Chapter 4 where current pieces of mobile malicious software are investigated according to their portability to other mobile operating systems.

- The technical contributions of this thesis start in Chapter 5 with the description of design and development of a tool for dynamic software analysis in Windows Mobile, which is able to log system calls of a particular execution of a software sample on the level of user-level system calls and kernel-level

system calls. It is evaluated by analyzing known malware for Windows Mobile.

- In Chapter 6 we investigate the resistance of Windows Mobile against fuzzing attacks aimed at understanding the current state of mobile device security. The chapter further argues for an attack surface reduction in mobile device security because the current diversity of mobile device security mechanisms is seen as inappropriate for the common mobile device user. Therefore, a *policy enforcer* is introduced.

- Chapter 7 investigates the possibilities of increasing mobile device security within the mobile network itself. It uses discrete event simulation to simulate the effectiveness of various network-centric security mechanisms.

- Chapter 8 investigates how algorithms for distributed computation can be applied in the setting of this thesis: the trusted module SIM card is used to solve two problems. First, secure multiparty computation in an implementation using J2ME and Java Card. Second, fair exchange with guardian angels in partially synchronous network settings using Java.

- Chapter 9 summarizes the content of this thesis and discusses the pros and cons of the security mechanisms that were introduced in the previous chapters.

Chapter 2

Related Work

This chapter introduces related work as a basis for the rest of this thesis and it connects this related work to our contributions.

Section 2.1 gives a definition of mobile devices—the investigation subjects. Closely connected to the investigated mobile devices is the trusted module—the mobile network operator (MNO) smartcard—that will be introduced, as well as well as specifics of mobile devices compared to desktop security.

Section 2.2 gives basic definitions of mobile device security. Afterwards, Windows Mobile, Symbian OS, and J2ME are introduced as examples of the current state of security in mobile device operating systems and application frameworks. This technical knowledge is necessary to understand the later parts of this thesis.

Section 2.3 introduces the definition and presents surveys of mobile malicious software. Moreover, it presents virus scanners as the defense side of malicious software.

Two analytic mechanisms are introduced in Section 2.4: dynamic software analysis and discrete event simulation. The topic of dynamic software analysis is a basis for the MobileSandbox dynamic malware analysis tool of Chapter 5. The methods of simulation are used in Chapter 7.

The distributed computation algorithms secure multiparty computation and fair exchange in Section 2.5 are the basis for understanding their implementation on mobile devices in Chapter 8. Both algorithms use security modules.

2.1 Mobile Devices

This section gives a definition of mobile devices—the investigation subjects. Closely connected to the investigated mobile devices is the trusted module—the mobile network operator smartcard—that will be introduced, as well as well as specifics of mobile devices compared to desktop security.

2.1.1 Definition

As a first approach, the investigation subject of this thesis is defined as: any mobile device that contains a smartcard that is controlled by a mobile network operator (MNO). Intuitively, this is the definition of a mobile phone.

This definition is mainly true, but there are mobile phones that are not in the focus of this thesis. These are mainly the kind of phones that can only be used for the phone functionality (plus text messaging and some basic other functionality), often aligned with a limited display size. They sometimes have proprietary operating systems and are not extensible with additional software. Even though the applications on these phones can be attacked, e.g., denial-of-service attacks with malformed SMS messages, they are not the typical attack target of mobile malicious software.

Other exceptions are some restricted environments that are not in the focus of this thesis either: USB sticks that enable laptops to use the mobile network are not covered. Moreover, there are some other devices with operator-controlled smartcards that are a restricted environment of their own (machine-to-machine types of communication). Both are not extensible with third-party software and the operating systems are proprietary developments.

Mobile devices also have other communication interfaces like WLAN and Bluetooth, and malicious software exists that only uses these interfaces for spreading. Consequently, devices can be imagined that do not have a connection to a mobile network, i.e., do not contain an operator-controlled smartcard, but are attackable by mobile malware. Fortunately, all relevant mobile device operating systems provide the interface to the mobile network together with the local communication interfaces. That is why the intuitive definition from the beginning still holds.

A more formal definition follows now as an important distinction concerning the possible security mechanisms.

MNO Smartcard: *An MNO smartcard is a smartcard inside the mobile device that is controlled by a mobile network operator (MNO).* Whenever this term is used

in this thesis, it can be used for all smartcards in mobile devices that are controlled by an MNO regardless of the actually used technology (see Section 2.1.2.2).

Smartphone: *A smartphone contains an MNO smartcard with a connection to a mobile network. Moreover, it has an open operating system that can be extended with third-party software.* These two properties in combination are the reason for this entire work and the smartphone is is the central attack target of this thesis.

The term “smartphone” as one word is chosen intentionally. It is supposed to denote that not only “smart phones” are under attack, but that the smartphone with its two main properties defines a complete new class of attack targets and protection needs, which takes place in a setting with mobile devices connected to the network over a wireless link and a more centralized environment of the network operators. Additional properties of these smartphones can be found in the literature [237].

Feature Phone: *A feature phone has a closed operating system that has pre-installed applications but that does not allow third-party software to be installed. Apart from that fact it is comparable to the smartphone because it has applications, large display and amenable processing power.* Therefore, feature phones are prone to the same attacks as smartphones, but they cannot easily be protected with security mechanisms like locally installed anti-virus software. As a side note: smartphones may be restricted to be feature phones by not making an SDK available.

The distinction smartphone vs. feature phone is only relevant in some parts of the thesis. Therefore, the investigation subject is abstracted in the rest of this thesis as *mobile device* or just *device*. When the connection to the mobile network is emphasized, it is called *mobile phone*. The mobile network is operated by the *mobile network operator* (*MNO*).

Mobile devices offer various services to its users. Popular is messaging as Short Message Service (SMS) or Multimedia Messaging Service (MMS). They use certain protocols that are explained in the literature [90].

In contrast to mobile devices, the traditional computers are called here *common computers*. When their fixed location is emphasized, they are called *desktop computers*.

2.1.2 Security Modules

This section will introduce the topic of security modules for mobile devices. Special focus is on solutions for secure multiparty computation with security modules,

which will be introduced in Section 2.5 and investigated in Chapter 8. The topic of security modules is also addressed by Anderson [5].

2.1.2.1 General Properties

Notation. Today, security modules usually have the form of a *smartcard*, i.e., a card with a specific form factor containing a chip with processor and memory. The spelling “smart card” is also in use, but as with smartphones above this thesis uses the more forward-looking spelling. This is based on the fact that the card is not only a “card that is smart” but defines a new class of entities.

Tamper-Proof. Security modules are trusted hardware inside of a mobile device. That is, they can contain data (e.g., cryptographic keys) that is only accessible via a defined interface of operations. This ensures that no tampering is possible even if a possible attacker has control over the entire device.

The main incarnation of a security module in mobile devices today is the MNO smartcard. Currently, there are other initiatives trying to bring new security modules into mobile devices, enabling more third-parties to provide applications with security modules.

Trustworthy. The smartcard is expected to behave according to the software that it was programmed with. A user has to trust the device manufacturer because he is hardly able to verify this assumption. The manufacturer publishes a specification of the smartcard functionality and promises the accordance of the implementation to this specification. The specification also lists possible interactions with the interface of the smartcard. It must not be possible to invoke operations that have not been pre-defined and documented.

Restricted Environment. Smartcards have restrictions in computing power and memory. Mobile devices have this restriction in general, but this is even more significant for smartcards. Therefore, it is crucial to limit the involvement of the card in any solution that makes use of the smartcard.

Exchange Frequencies. The question can be posed how new security features find their way into mobile devices. Two alternatives exist: waiting for more sophisticated MNO smartcards and waiting for new devices containing new security hardware. Replacing all the existing MNO smartcards is an expensive task and it

can be assumed that no new functionality of the MNO smartcard will be introduced by exchanging all existing cards, regardless of how sophisticated the functionality is. The assumption can be made that new (security) functionality is introduced more likely with new devices because they are supposed to be exchanged more frequently than MNO smartcards.

2.1.2.2 MNO Smartcard

The MNO smartcard was called “SIM card” in the GSM network (2G). It was a monolithic entity, being able to denote the physical smartcard or the logical functionality within the GSM network. In 3G networks the Universal Integrated Circuit Card (UICC) was introduced as the physical card. It contains logical applications for different use cases: a SIM application for communication with 2G networks and a USIM application for communication with 3G networks.

MNO smartcards are seen more and more as a server component in the recent time [113, 224]. While this is true for modern and advanced smartcards, the majority still has very limited capabilities (1 kilobyte of RAM, small EEPROM).

Provisioning of applications can only be done by the MNO. Therefore, the MNOs have control over third-party applications using the MNO smartcard as a security module on mobile devices. This is a reason why the security specifications exist that are introduced now.

The so-called *SIM lock* is a feature of a mobile device to only accept MNO smartcards of a specific MNO. This is used for subsidized devices and for devices that are bound to a specific MNO, e.g., the iPhone.

2.1.2.3 Security Specifications

There exist additional specifications for security modules in mobile phones. A more thorough introduction can be found in Pisko et al. [175].

Open Mobile Terminal Platform: Trusted Environment. The Open Mobile Terminal Platform (OMTP) defines requirements for trusted environments. The OMTP Trusted Environment [160] defines a model of the mobile device and the MNO smartcard. Additionally, a threat model is defined based on the device model. It is the basis for the OMTP Advanced Trusted Environment [164], where secure storage and secure boot are defined as enablers for future applications. Trusted device management is named as an example of these applications.

Trusted Computing Group: Mobile Security Specification. The Mobile Phone Working Group (MPWG) is part of the Trusted Computing Group (TCG) with the goal to define standards that get their security by using a trusted platform module in the mobile phone, called Mobile Trusted Module (MTM) [218]. They define a mobile reference architecture (RA) consisting of distinct trusted engines. The definition of this security module approaches the problem of mobile device security from a deeper technical level, because they have an attacker model with a more sophisticated attacker. A first implementation of this reference architecture based on SELinux was given by Aciicmez et al. [1].

The effects of the MTM for the overall security of mobile devices have been reflected by Leavitt in 2005 [122], thoughts about deploying these modules were published by Kasper et al. [189], and using these trusted modules for virtualizing the MNO smartcard has been investigated by Kasper et al. [190].

2.1.2.4 Java Card Framework

The Java Card Framework is a Java runtime environment for smartcards. It is a subset of the Java Platform, e.g., important data types like *float* and *string* are not available. The Java Card framework is the basis for our implementation of secure multiparty computation with security modules in Section 8.2. More information about technologies and using the Java Card framework can be found in the literature [32, 128].

Communication can take place with application protocol data units (APDU) or with a more sophisticated Java Card remote method invocation (JCRMI). Of most interest is APDU, because it is implemented more often than JCRMI in mobile devices (see Section 2.2.2).

Application Protocol Data Units. The application protocol data unit (APDU) is the common communication means between a Java Card applet and its host application. Communication is specified as ISO standard 7816. It is a simple request-response protocol (half duplex communication), where after a request (*command*) from the host a response (*response*) from the Java Card with a fixed return value length follows. The Java Card applet can indicate the availability of data with a special return value, which can be retrieved with a *GET RESPONSE* command.

There are two main protocols for APDU communication: T=0 and T=1. T=0 is a byte-level protocol and T=1 a block-level protocol.

Figure 2.1 shows the structure of a command APDU. The header consists of four bytes: class of instruction (CLA) selects the addressed application on the smartcard,

Header				Body		
CLA	INS	P1	P2	Lc	DATA	Le
1	1	1	1	1	0-*	1

Figure 2.1: Structure of Command APDUs

the instruction code (INS) selects the addressed command of the application, and the parameters P1 and P2 give additional information without the need to attach the optional body to the APDU. The Lc byte of the body specifies the length of the following DATA field that contains application-specific data. The Le byte specifies the length of the response APDU. If no answer is expected, the byte is omitted.

The response APDU consists of a body and a trailer. The length of the body was defined by the Le byte of the command APDU. The trailer consists of two status bytes that contain a status code of the command execution.

Java Card version 3. A draft for a new version 3 of the Java Card framework exists (released in March 2008). It will be split into two parts. The *classic edition* is an evolution of current version 2.2.2. The *connected edition* introduces the Web protocol HTTP to the Java Card world. Java Card applications can be a servlet that can be accessed via HTTP from the host application. With Java Card 3, MNO smartcards can be seen as in the process of developing from a multi-applications smartcard platform to a multi-interface network connected secure token [224].

2.1.3 Specifics of Mobile Devices

A central question for the solutions of this thesis comes from the scientific point of view: is researching on the security of mobile devices different from common security research? Could it not be possible to transfer known security solutions from common desktop computers to mobile devices? Could it possibly be the same, only with the additional word “mobile” in the title?

This thesis says: no, there are specifics of mobile device security that justify research on this topic. In its solutions, this thesis makes use of these unique features of mobile devices compared to common desktop computer security. They are the basis to novel security mechanisms especially designed for mobile devices and their infrastructure, and these mechanisms cannot be transferred from existing common computer security solutions. An overview of these differences is shown in Figure 2.2 and they will be introduced subsequently.

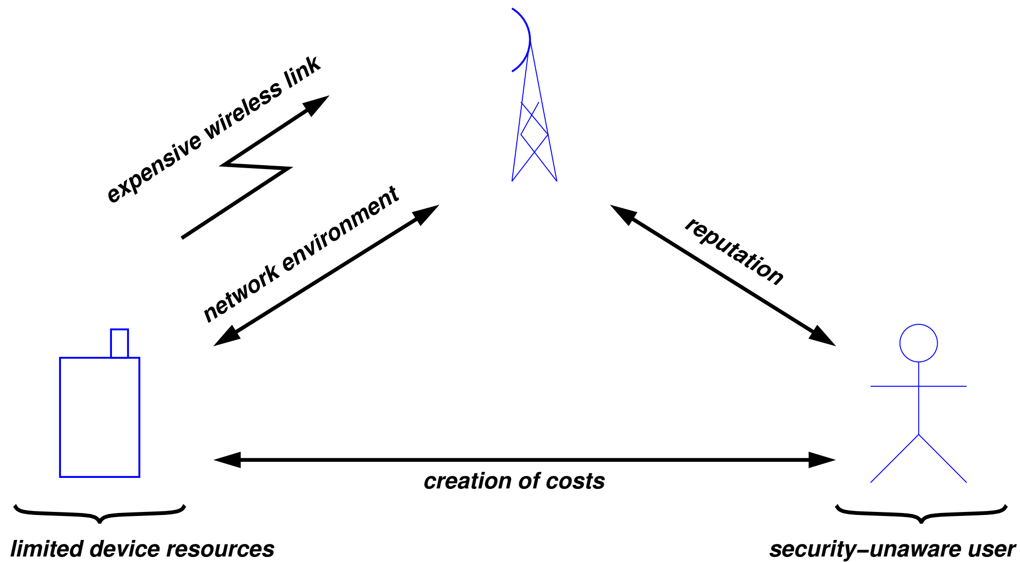


Figure 2.2: Specifics of Mobile Devices

2.1.3.1 Creation of Costs

The specific *creation of costs* is the inherent possibility for malware to generate costs for the user and revenue for the malware author. It has two aspects: events that are billed by the mobile network operator (phone calls, messages) and the arising payment systems.

Billed Events. The problem of billed events existed previously in PC security when dial-up connections via modem or ISDN were common. Malware could dial premium-rate numbers and with it directly benefit the malware author. With the appearance of DSL and flat rates this problem mostly vanished, because the connection to the telephone system was not available anymore. However, in mobile devices it will most likely be a problem for a long time. Even if flat rates for data or voice services become common, separately charged premium services will always be available.

Payment Systems. A first type of payment systems uses the messaging functionality of mobile phones as a trustworthy channel for transmitting authorization information, e.g., online banking with mobile transaction numbers or online payment services like mpass. In general, there are two communication channels that need to be compromised. However, the mobile device is the only channel that needs to be compromised if an attacker has access to the authentication information

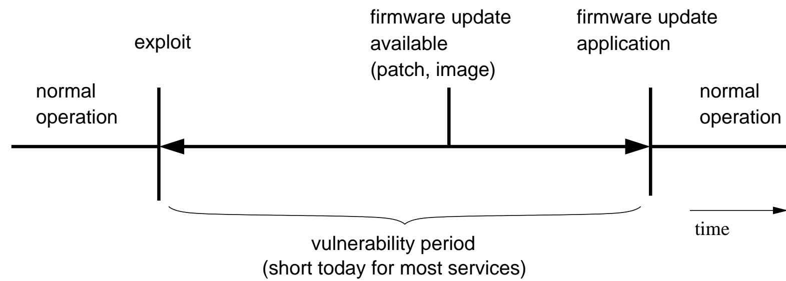


Figure 2.3: Vulnerability Lifetime

of the targeted account. Customized mobile malware might forward the messages to the attacker or respond to them in the expected form. However, the necessity of these attacks being customized makes it more probable that mobile malware will use the cost-creating functionality of the mobile network.

A second type of payment systems uses the mobile phone as payment device and physical proximity as part of the authorization (Near Field Communication, NFC). In this case, the required proximity to the receiver of the payment enhances the security and makes these attacks unlikely compared with directly using the mobile network cost-creating functionality.

2.1.3.2 Network Environment

The specific *network environment* consists of the three aspects strong connection, firmware update process, and remote device management.

Strong Connection. Strong connection means the presence of the mobile network operator and its influence on the device. Different from PCs where the network provider almost always has no influence on the user's computer, the mobile network operator has the MNO smartcard inside the mobile phone as a trusted device. It is possible to create trusted applications on the mobile phone with enhanced security. This specific remains true even despite of the facts that trusted platform modules start to appear in PCs and that third-party trusted modules are available for mobile devices, e.g., embedded into a memory card. Both do not have the unique owner that the MNO smartcard has.

Firmware Update Process. The process of updating the firmware of mobile devices changed rapidly during the last few years. A few generations of mobile phones ago, an update of a firmware could only be done in a local setting, possibly

only by the device manufacturer itself. With the rise of smartphones and extensible operating systems, more sophisticated hardware architectures have been used. These new architectures enable a firmware or third-party software update remotely.

Figure 2.3 shows the lifetime of a vulnerability. The time between the exploit of a vulnerability and the provision of a firmware update is the *vulnerability period*. As Section 3.4 will show, the time until a patch is provided for today's most important application layer vulnerabilities is sufficiently short. This makes two aspects important: the technical process of applying the update and the role of the mobile device user who sometimes has to play an active role in the update process.

Even though remote update is possible today and an update nowadays does not differ much from common computers, updating mobile devices remains a challenging task. If not connected to a host computer on a regular basis, an update process has to use the expensive wireless interface.

Updating the firmware over the air is an important functionality to update vulnerable parts of the mobile device's operating system. It is also a critical feature, because most update procedures cannot be interrupted without damaging the device. Instead of a complete firmware update, the exchange of single files of the operating system's file structure is better suited. This is especially true in terms of wireless communication and device resource costs. The firmware update process itself is considered secure in this thesis. Every part of the update is cryptographically signed and a secure boot process can ensure that only legitimate updates are executed.

An important application of automatic remote firmware updates are feature phones. They have a closed operating system and cannot be extended with virus scanners or other security software. Therefore, these type of devices need more attention of the mobile network operator.

An additional aspect is the entity that starts the update. This has traditionally been the mobile network operator, but only recently the manufacturers started to control the firmware update process themselves (examples are the iPhone and Android).

Remote Device Management. Remote device management provides methods to manage mobile devices when they are already in use. An important feature of mobile devices is the ability to be managed by a remote entity. This is due to the fact that usually some entity has more power over the device than in common computer environments, e.g., the mobile network operator, the device manufacturer, or the corporate IT department.

The common user experiences feature changes mainly as remote configuration, for example, when MMS or WAP settings are pushed to the device. Other feature

changes are mainly targeted at corporate environments, where the IT department has to enforce a corporate information security policy on the devices. Examples of these features are disabling the Bluetooth, WLAN, or memory card interfaces for preventing corporate data to leak from the protected device. An interesting feature in this context is the "remote wiping" functionality. Lost or stolen devices can be deleted completely by a remote entity. Relevant documents in this area are the "OMTP Advanced Device Management" [163] and the "OMTP User Experience Remote Service" [158].

2.1.3.3 Limited Device Resources

The specific *limited device resources* has the two aspects resource-limitedness and mobility.

Resource-Limited. This is the most obvious difference to common computers. Even though it is always said that mobile devices today have the computing power of desktop computers of "some years ago", they are still limited compared to common computers. The limiting factors are the central processing unit (CPU) and the main memory (RAM).

These two factors limit the sophistication of possible security solutions, e. g. , the sophisticated intrusion detection algorithms that hardly work for real-life application on common computers will never be transferable to mobile devices.

Mobile. The factor *battery* limits the resource needs for a security solution from the point of view of the general acceptance factor. Even though the common user might not notice this point, it is important that a security solution does not constantly need large portions of available CPU time, leading to battery exhaustion.

2.1.3.4 Expensive Wireless Link

An additional specific of mobile device security is the *expensive wireless link*. Expensive is meant here more in terms of computation costs for the algorithm than in terms of costs for the user.

Expensive Computation Costs. Compared to local computations on the device the wireless link is always expensive for an algorithm. Thus, solutions for increasing the security of mobile devices should try to avoid this communication.

However, transferring computation load from the device to the mobile network is desirable as the device resources are limited. This is an area of conflict between the limited device resources (processing power, memory), the design of security algorithms using the computing resources of the mobile network, and the expensive communication between these two.

Expensive Communication Costs. A minor aspect are the communication costs, i.e., the costs of using the mobile network. This is only a side aspect of the specific *expensive* in Figure 2.2 compared to the computation costs.

Communication cost means that either the user has to pay for the security solution or the network operator has to consider these communication costs in the calculation of its flat rate conditions.

2.1.3.5 Reputation

The specific *reputation* can be seen as a weak specific of mobile devices. The mobile network operator will invoice every event that generated costs, even though it might have been generated by malware. Therefore, it can be thought that the mobile network operator could be held responsible from the user's point of view. In case of a widespread mobile malware outbreak with several network operators involved, mobile malware might even have an impact on the reputation of the entire mobile phone system in general.

2.1.3.6 Security-Unaware User

A last specific is the *security-unaware user* of mobile devices. Connected with the reputation of the mobile network operator, this topic differs from common computer security. This unawareness is discussed in more detail in Section 3.5.

2.2 Mobile Device Security

This section gives a minimal definition of the important parts of security that are necessary for this thesis to be self-contained. However, we do not want to repeat large parts of the definitions of other authors. This section is based on the definitions of Gollmann [85] and Avižienis et al. [9] who can be referred to for a more comprehensive introduction.

As examples of the current state of security in mobile device operating systems and application frameworks Windows Mobile, Symbian OS, and J2ME are introduced. These technical details of current mobile operating systems are necessary for understanding the attack vectors of Chapter 3, the description of mobile malware in Chapter 4, and the dynamic software analysis tool in Chapter 5 with its applications in Chapter 6.

2.2.1 Definition of Security

The notion *security* is closely connected with the *protection of assets*. It is commonly accepted to define three main properties of security from the view of how the protected assets can be compromised: confidentiality, integrity, and availability.

Confidentiality is about preventing unauthorized disclosure of information, *integrity* is about preventing of unauthorized modification of information, and *availability* is about preventing unauthorized withholding of information or resources.

Additional properties were defined. *Non-repudiation* is about providing unforgeable evidence that a specific action occurred. This property will be of importance for fair exchange protocols in Chapter 8.

When considering the topic of this thesis, we adopt the broadest definition by Gollmann [85]: “Computer Security: Concerned with the measures we can take to deal with *intentional* actions by parties behaving in some unwelcome fashion.”

A *vulnerability* is an internal fault of a system that enables an external fault. Common vulnerabilities of software systems are buffer overflows or race conditions. A vulnerability becomes important in the security context if it can be *exploited*, meaning to violate the security properties of a system.

Of special importance in the context of telecommunications networks are authentication, access control and accounting. With access control being called authorization, the abbreviation AAA is commonly used for this triple.

Authentication is about verifying the claim of one entity that it is possesses a particular secret.

Access control is about controlling the areas that a (previously authenticated) user can access. Besides the mobile network setting access control is also present on the devices itself.

Accounting refers to logging the use of resources and to the ability to link these logs with the entity that was responsible for this use of resources. In mobile networks, this information is mostly used for billing purposes.

2.2.2 Security in Application Frameworks

Application frameworks aim at defining an interface to the functions of the operating system, i.e., restricting what applications can do on the system. One example of such application frameworks is the Java Platform, Micro Edition (J2ME), which is introduced in this section. Other application frameworks are the Dotnet Compact Framework and the Binary Runtime Environment for Wireless (BREW). The advantage of J2ME over the other frameworks is its wide adoption in mobile devices. J2ME and its Security and Trust Services API are the technical basis for our implementation of secure multiparty computation in Chapter 8.

2.2.2.1 Common Properties of Application Frameworks

The executables of application frameworks do not contain machine code instructions but an intermediary format ("bytecode language"). Therefore, the executables can be interpreted on any hardware platform that provides a converter from bytecode language to the native machine code of the hardware.

The runtime environment has complete control over the executed application and can be seen as a security interface. This is sometimes called "sandboxing" and access to the system can efficiently be restricted. The functionality of an application framework is provided by APIs and similar APIs are often grouped.

2.2.2.2 Java Platform, Micro Edition (J2ME)

This is a very short introduction to the security of the Java Platform, Micro Edition. More comprehensive information—including its functional features—can be found in the literature [44].

J2ME is subdivided into configurations and profiles. The configuration in mobile phones usually is the Connected Limited Device Configuration (CLDC, JSR 139), which is currently available in version 1.1. The profile usually is the Mobile Information Device Profile (MIDP). The runtime environment of J2ME is the K Virtual Machine (KVM). A reference implementation and several other implementations exist [44].

Additional specifications for security communication with smartcards exist. They are specified in JSR 177: "Security and Trust Services API". It is included in JSR 248: "Mobile Service Architecture". All these specifications will be discussed shortly.

2.2.2.3 Mobile Information Device Profile (MIDP, JSR 37/JSR 118/JSR 271)

MIDP version 2 (JSR 118 [106]) introduced an enhanced security model with permissions, named oneshot, session, blanket. Currently, MIDP version 3 (JSR 271 [109]) enhances the security model of MIDP2. It is currently being specified and supposed to be published at the end of 2009.

The MIDP 3 specification "extends the expressiveness of the security policy" [109] by defining a format for writing a security policy that is based on the Java Standard Edition security policy format.

Another work is the extensive comparison of "implementation versus specification" of J2ME low level security [178]. This work contributes tools for low-level manipulation of MIDlet suites and an analysis of the robustness of current implementations against these low-level manipulations. Aspects of analyzing Java malware (not restricted to J2ME) and possible measures of virus writers to prevent an analysis of their code can be found [177]. Moreover, an analysis of J2ME with a risk assessment approach ("MEHARI" method) is available [44].

2.2.2.4 Security and Trust Services API for J2ME (SATSA, JSR 177)

The "Security and Trust Services API for J2ME" [107] defines the interface between J2ME and Java Card functionality within the mobile device. The specification consists of four parts that are all optional to implement: APDU, JCRMI, PKI, and CRYPTO.

APDU defines APIs for communicating with the smartcard using application protocol data units and JCRMI for communicating with the smartcard using the Java Card remote method invocation protocol. The CRYPTO package defines cryptographic functions like encryption and hash algorithms. The PKI package defines APIs for creating digital signatures from the keys within the smartcard and provides basic management methods for the keys.

Commonly implemented today are the APDU and the CRYPTO part. The APDU part is most important for the implementation of secure multiparty computation in Chapter 8.

2.2.2.5 Mobile Service Architecture (MSA, JSR 248/JSR 249)

The success of the Java Community Process led to a growing number of additional specifications, e.g., file access (JSR 75), Bluetooth (JSR 82), messaging (JSR 205). The claim of J2ME to be device-independent started to lose its validity because of

the numerous optional JSRs. A first attempt for defining a standard set of JSRs was "Java Technology for the Wireless Industry" (JTWI). The current approach is the "Mobile Service Architecture" (MSA), defined in JSR 248 as MSA 1.1 [108] and in JSR 249 as MSA 2.0 [105]. MSA defines profiles with corresponding mandatory JSRs, and devices have to implement these JSRs for being compliant to the profile.

MSA 1.1 defines a full set and a subset. JSR 177 is only included in the full set with APDU, CRYPTO, and PKI. MSA 2.0 defines a full set, a subset, and a limited set. JSR 177 is not included in the limited set, but is included in the subset with APDU and in the fullset with APDU, CRYPTO, and PKI.

2.2.2.6 Other Application Frameworks

Dotnet. The Dotnet framework uses the Microsoft Intermediate Language (MSIL) for its binaries. A special version for mobile devices exists, the Dotnet Compact Framework. An unofficial extension is the SmartDeviceFramework, which implements some of the parts that are not present in the Dotnet Compact Framework.

The Dotnet framework on mobile devices aims at being the standard framework for third-party software. When the operating system is closed for third-party software, many vulnerabilities of Windows Mobile would be relieved.

OMTP ASF. The Open Mobile Terminal Platform (OMTP) is a forum of mobile network operators [175]. They define requirements for mobile devices in several requirements documents and dedicate some of their activities to mobile device security, being signing of applications and the definition of an "application security framework" [165]. Contributions of the framework are the subdivision of API functions into different groups and the definition of several trust levels. Both are comparable to the J2ME definitions. Another notable contribution is the precise definition of run-time prompting, which can be summarized as "for unapproved applications prompt for every access" (oneshot).

2.2.3 Security in Windows Mobile

The technical knowledge on Windows Mobile in this section will be used to understand design and implementation of the MobileSandbox dynamic software analysis tool in Chapter 5, for the development of a proof-of-concept smartphone worm in Section 6.1, and for designing a means to reduce the attack surface of mobile devices in Section 6.2.

First, the environment of Windows Mobile is introduced. Following is information on system calls, protected server libraries, and kernel data structures. This discussion of Windows Mobile concludes with a survey on vulnerability research for Windows Mobile and with its exploited vulnerabilities.

These sections only give the most necessary information for understanding the operating system internals due to space restrictions of this chapter. A more detailed description of Windows CE can be found in the literature [149].

2.2.3.1 Windows Mobile Environment

The term *Windows Mobile* describes the collection of an operating system core together with productivity applications. The core is named *Windows CE*, partially being published as source code. Our solutions are designed for Windows CE version 5 that is basis for the current Windows Mobile versions 5 and 6. The most important concept concerning our implementation are the Windows type of *system calls*. All services of the operating system are accessible only by using system calls.

The central kernel-level process is *nk.exe*. Every system call on kernel-level first leads to this process in the kernel. We will exploit this fact for intercepting the system calls within *nk.exe*. A user-level process is able to run as kernel-level process by using the undocumented system call *SetKMode*.

ActiveSync is a synchronization program and protocol for common computers running the Windows type of operating systems.

2.2.3.2 Windows CE System Calls

From the user-level perspective, Windows CE provides the well known Win32 API interface with some minor exceptions. Therefore, many user space programs written for Windows NT based operating systems can be easily ported to Windows CE. In contrast to user space, the kernel is different from the kernels of the other Windows operating systems. Especially processing of system calls is different.

System calls on common computers are typically implemented by executing dedicated software interrupts like *int2e* in Windows NT. Some versions also use the special *sysenter* instruction that is provided by the x86 instruction set. Subsequently, a handler function is executed in the kernel, the requested system call is processed and finally, the kernel gives execution back to the initiator of the system call in user space. The requested function and the parameters are given by the parameters of the interrupt call and the user space stack.

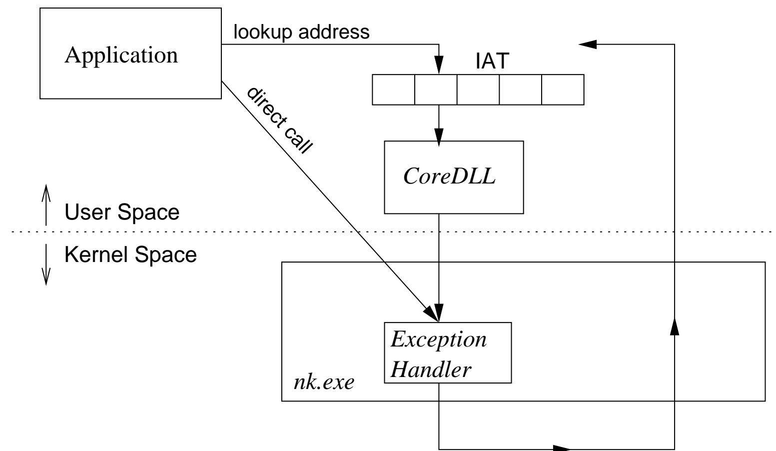


Figure 2.4: System Call Architecture of Windows Mobile

Windows CE uses a slightly different approach. Although the ARM processor architecture provides an interrupt instruction *SWI*, the transition from user space to kernel space is achieved by jumping to a specially crafted invalid memory address consisting of an architecture-dependent fixed offset, an *APISet* number and a method number. Consequently, the exception dispatcher is executed and checks whether or not the address is assigned to a certain system call. Therefore, a special area of the memory is reserved for such system call traps, denoted as the *kernel trap area*. On ARM processors, this area is located between the memory addresses 0xF0008000 and 0xF0010000, and kernel trap addresses can be computed by the formula

$$0xF0010000 – (APISetID \ll 8)|MethodID) \cdot 4$$

Figure 2.4 illustrates the way of a system call through the system. The direct way described above directly leads into kernel space.

The common way is by using the import address table (IAT). This is a table with the actual addresses of the system calls. System call addresses in executable files are indirect jumps to addresses that are in a fixed position of the IAT. This has to be done because the actual addresses of system calls can be different on different incarnations of the system. It is a task of the Windows loader to fill the IAT with the actual addresses.

With the correct address from the IAT, the call goes from the application to the designated export in a DLL (usually *CoreDLL),* where an exception is raised as mechanism to switch into kernel mode. The exception handler within the kernel

looks up the system call addresses in a *CINFO* structure and jumps to the start of the system call.

2.2.3.3 Protected Server Libraries

Windows CE loads device drivers as non-privileged user mode processes. As a consequence, system calls are processed in separate processes, whose executions must take place in kernel mode.

Each device driver process that exports system call APIs has to register its own *APISet* first by calling the special functions *CreateAPISet* and *RegisterAPISet*. The parameters consist of an arbitrary name with a length of four bytes, the number of exported functions, a method pointer table to the corresponding handler functions, and a pointer to a signature table being a bitmask of 32 bits, where the various bits indicate whether or not a certain argument is a pointer. The number of different *APISet*s is limited to 32, where the lower 16 identifiers are reserved for the kernel. In a traditional client/server model, the caller and the server run in separate threads. In contrast, Windows CE lets threads migrate between both processes in a system call for the sake of performance. Therefore, the current process of a thread does not necessarily have to be the thread's owner. This information can be obtained by calling *GetCurrentProcess*, *GetOwnerProcess* and *GetCallerProcess*. The latter returns the caller process of the current protected server library (PSL) API, while *GetOwnerProcess* obtains the process which really owns the thread performing the function call.

2.2.3.4 Internal Kernel Data Structures

To understand how it is possible to hook API calls on the kernel mode level, one has to know which relevant and modifiable data structures are maintained by the kernel.

Each *APISet* contains all its information in a *CINFO* structure. This includes all the parameters that were passed to *CreateAPISet*, as well as the dispatch type. Currently, Windows CE distinguishes *handle-based* from *implicit APISet*s, the former ones being direct system calls while the latter ones are attached to handles such as files, sockets, and so on.

An implicit API is identified by its *APISet* identifier and method identifier. In contrast, a handle-based API is given by its handle and the method identifier. In order to access the data of each implicit *APISet*, the kernel maintains an array that holds all *CINFO* structures. A pointer to this array can be found in the

UserKInfo array which is always located at the fixed offset 0xFFFFCB00 on the ARM architecture. As even the kernel mode *APISet*s are registered when the system boots, all the relevant pointers are contained in writable memory pages. Thus, they can simply be altered and redirected to different functions. On the other hand, a *CINFO* structure exists for each handle, which is allocated when the handle is created and deallocated when it is closed.

For the purpose of completely intercepting system calls, the attached *CINFO* pointer must be changed after its creation. As every handle is created in an implicit API call (such as *CreateFile*, *socket*, etc.) those functions will need some special handling in order to hook the method of the handle they return. This special handling does not prevent the hooking of all system calls.

2.2.3.5 Windows CE Vulnerability Research.

A high-level analysis with concern to electronic signatures has been given by Murmann and Roßnagel [148]. The operating system was rated according its possibility to be a trustworthy environment for a signature scheme. It was found as unsuitable because of its lack of encryption and its completely unprotected internal communication. The work concludes that the Perseus project[1] and the use of a trusted platform module would lead to a solution of the problems. The Perseus project is a security kernel with special focus on trustworthy internal communication and isolation of security-relevant processes.

An early publication on Windows CE low-level security was at Black Hat Europe 2003 by de Haas [43]. Besides some hardware information, the talk summarizes typical security flaws of that version and presents the “Wallaby Patch Tool” custom boot loader for the HTC Wallaby, which is able to copy device memory to the SD card and to remove the device PIN. In Black Hat talk “Pocket PC Abuse”, Fogie [74] presents a keyboard logger, the possibilities of hidden programs, and code to trigger a hard reset.

The topic of shellcode generation is dealt with in the year 2005 by Mulliner [145], Hurman [98], and in Phrack magazine [185]. These works use Windows CE version 4.2 as a basis. Finally, we investigated Windows Mobile version 5 recently in 2007 (Becher et al. [20]). This work adds a robust framework around the topic of low-level shellcode creation.

The work of Asselineau and Hospital [8] deals with the use of the C API in order to infect a process and with the limitations of exploiting Windows CE at the kernel level. It proposes the transfer of the concept of capabilities from Symbian OS as a

[1] http://www.perseus-os.org/

solution, and mentions a solution based on virtual machines. It concludes with a statement on currently available anti-virus and personal firewall programs. They would not offer sufficient protection, because:

1. the defensive process would have the same rights as the malicious process, therefore it could be terminated by the latter, and
2. the anti-virus engines would be insufficient and it would be impossible to deal with a sufficiently complete signature database. Even the behavior-based detection is discarded by the authors, because the operating system would already need too much of the limited resources of the device.

2.2.3.6 Exploited Vulnerabilities of Windows CE.

Only one remote code exploit is known so far for Windows Mobile. It is a buffer overflow in the handler program of MMS messages [147].

The user receives an MMS message with a specially crafted header field that will trigger a buffer overflow when the MMS is processed by the message handler process *tmail.exe*. The header field will be read when the user opens the MMS for reading. The buffer overflow is not executed when the MMS is unread in the inbox.

The exploit today is the only known example of an infection of malware requiring only common interaction (see Section 4.2). Updated versions for the message handler are available and can be downloaded for protecting the device against possible malware that exploits the vulnerability.

2.2.4 Security in Symbian OS

This section introduces the most basic knowledge about Symbian OS and its Platform Security Architecture (PSA). Only the most important parts are presented here that are necessary for understanding technical information on attack vectors in Chapter 3 and on the Symbian malware of Chapter 4. A more thorough introduction can be found in the literature [46, 95].

2.2.4.1 Operating System

A central notion for the modularity of Symbian OS is the provision of services through servers that can be accessed by clients. A *server* usually runs as an independent process and waits for commands. The *client* accesses the server

through a well-defined protocol, possibly with the help of a client-side library that defines *APIs* (application programming interface) as a basic unit for providing services. This modularity makes the introduction of security features easy, as can be seen in the following section.

Symbian OS uses the concept of *dynamic link libraries* (DLLs) just as other operating systems: they provide functionality that can be used by more than one process at the same time. A DLL is loaded into memory when a process uses one of the DLL's APIs. The DLL is removed from memory when it is no longer used by any process.

The installation packages of Symbian are called *SIS files*. They can contain several files like executables and resource files. An installation package also specifies a version number that will be important when another version of the package is already installed on the device. The installer recognizes installation packages by its type and not by its suffix, a property that is exploited by the Beselo malware. When unsigned software packages are installed the user has to confirm several security messages referring to the possible malicious nature of the package.

2.2.4.2 Platform Security Architecture

Beginning with Symbian OS version 9 in 2005, the Platform Security Architecture (PSA) has been introduced to increase the security of Symbian OS.

Capabilities. The capability model is a central concept of the Symbian PSA. Applications need capabilities to access certain system resources. They are defined within the installation packages. Holding a capability means for a process to possess an unforgeable data value that authorizes it against system processes when the process accesses sensitive functionality. The PSA specifies well-defined mappings between most of the APIs of Symbian OS and their associated capabilities [208].

There are 20 defined capabilities and they are discrete and orthogonal. That means, there is no hierarchy, every process must possess exactly the necessary capability for accessing a particular API.

Special attention is needed for DLLs, because they may be equipped with capabilities that may differ from the capabilities of their host process. Therefore, there exist rules for the interaction between DLLs and other processes. These *DLL rules* are defined as:

- A process holds a number of capabilities and these capabilities do not change during its lifetime. The capabilities are defined in the executable file of the process.

Table 2.1: Access Rules for Data Caging

Path	Capability needed for:	
	Reading	**Writing**
`\sys`	`AllFiles`	`Tcb`
`\resource`	none	`Tcb`
`\private\⟨ownSID⟩`	none	none
`\private\⟨otherSID⟩`	`AllFiles`	`AllFiles`
`\⟨other⟩`	none	none

- A process is only allowed to load a DLL, if the capabilities of the DLL are a subset of the process capabilities.
- It is not allowed to statically link a DLL to another DLL that possesses less capabilities.

Data Caging. Another central part of the PSA is data access control. This *data caging* is used to protect sensitive files. It is implemented as three access-restricted directories: `\sys`, `\resource`, and `\private`. All other directories provide unrestricted access for every process. The directories are shown together with their required capabilities in Table 2.1 and are explained now.

The directory `\sys` and its subdirectories are only accessible by the trusted kernel (`Tcb` capability). The directory contains all executable files of the system, ensuring that only the trusted kernel is allowed to create executable files or to load them into memory. The directory `\resource` is used for read-only resource files that do not need to be altered after installation.

The subdirectories of `\private` provide a private area for every application to store its data. The subdirectories are named after the *secure identifier* (SID), a value that uniquely identifies a running process. Processes can access their own private directory without any capabilities, but they need the capability `AllFiles` for accessing the private directories of other processes.

File System Environment. The file system defines certain rules referring to the addition of new files. With two exceptions, installation packages are not allowed to *overwrite* files. First, if the package is marked as an update for an already installed package, then overwriting of files is allowed. Second, if a trusted installation package tries to overwrite files of an untrusted package, then the user may be asked to allow the trusted package to overwrite the file, depending on how the PSA is configured on a particular device.

An *eclipsing situation* can occur, if two drives contain paths with equal name that contain files with equal name. The loader of Symbian OS searches for a file in reverse alphabetical order from drives `Y` to `A` and finally drive `Z`, until it finds the file. The SWInstaller tries to avoid security-relevant situations with the following eclipsing rules.

First, a SIS file tries to install a file that creates an eclipsing situation with a file of an untrusted package. The SWInstaller can ask the user if he wants to remove the untrusted file.

Second, creating an eclipsing situation with ROM files is always prevented. One exception exists: if the ROM contains a so-called "SIS stub file", the SWInstaller can identify the replacement file as legitimate file for updating the ROM.

2.2.4.3 Cryptographic Application Signing

Cryptographically signing of applications is most prominent in Symbian OS, because examples of recent mobile malware for this operating system exist that happened to be signed (FlexiSpy and Yxe, see Section 4.1.2). This led to malware samples that can use sensitive system calls without the user being informed. The reasons why this could happen are introduced here.

Preliminaries. Cryptographic signing of applications usually uses public key schemes, so every device needs a public key for verifying the signature of an application, the *root certificate*. The root certificates can, for example, be supplied by the device manufacturer or the mobile network operator. When signing schemes are applied, it might be the case that a signed application (e.g., signed with an operator certificate) is allowed to generate cost-creating events (e.g., sending messages) without asking the user for permission. It is not possible to override this trust relationship between the device and the operator-signed application. Operator-signed applications are usually connected with the provision of reliable sources for downloading the applications, for example, by using an operator-controlled download portal. This MNO view is documented in signing scheme [159] and application security [165] specification documents.

Certificates of the public key schemes are commonly classified into three different trust levels, depending on their issuer: device manufacturer, mobile network operator, and anyone else ("trusted third party", TTP). Applications can be signed by the owner of the certificate's secret key. These applications are then, for example, called "approved".

Cryptographic signing of applications is a good approach towards more trustworthy applications. However, it is difficult to put into practice for all programs, mainly because of the costs that come from the process.

Process. Basis for a successful implementation of cryptographic signing are certificates on actual mobile devices. In the best position for supplying these are the device manufacturers and the MNOs. For individual devices, extending the valid root certificates is also possible.

When an application is supposed to be signed, the author uses the appropriate signing program to retrieve a signature. This signature proves that the binary of the application is certified by the signing program, locally verifiable on every device with the corresponding root certificate. The necessary steps for passing the certification vary. Current signing programs mainly require contractual commitment and sometimes tests for functionality. Security is only part of the contractual agreement. It is desirable to have security properties tested contrary to have them only stated by a signed contract. This is current research in the MOBIUS project [136].

Depending on the policy of the signing program, a signed application may ask fewer questions to confirm access to security-relevant APIs. Or it may allow access to more APIs than unsigned applications have. Different trust levels might control access to different groups of similar APIs.

Symbian Signed. For Symbian OS, the *Symbian Signed* program delivers the infrastructure and the process for signing an application [211]. There are three possibilities to get an application signed: Open Signed, Express Signed, and Certified Signed.

Open Signed provides two different methods. The old method (which is no longer available) provided a developer certificate so that the developer is able to sign the application himself. Since the middle of 2008 it is no longer possible to get new certificates for this method. However, signing with existing certificates is still possible. The new and current method uses an online submission. A developer has to submit a SIS file and wait until the signing process is finished, reaching from some minutes to several hours. *Express Signed* allows self-signing as Open Signed, but these applications can be distributed to other devices afterwards. This is a fast method for signing SIS files with the restriction that not all capabilities can be granted. The most important option for commercial software development is *Certified Signed*. It requires that the application fulfills certain testing criteria and the application must be tested by an independent testing house.

Revocation. A current challenge of the mobile device world is establishing the use of certificate revocation. It is imaginable that a malicious application is able to fulfill the criteria for being signed, and actual examples exist [72, 220]. Reasons can be hidden functionality or insufficient testing. Therefore, it is useful to define means for revoking the signature in case of a signed malicious application. A device is able to check a valid signature against public revocation lists for increasing the possibility of avoiding a malicious application. Protocols are defined (Online Certificate Status Protocol - OCSP [151]) and they are implemented, but either they are not used (because of the costs of downloading revocation lists) or their check can be overridden by the user [15]. Whenever security incidents are going to matter in the mobile world, it will be discussed who is responsible for the damage. This can be of special of importance in the case of financial damage. Compared with the discussion on phishing for online banking, the mobile network operators might be held responsible for the damage unless they prove the user as guilty.

Effects. The main difference of signed applications compared to unsigned applications is the former being connectable to an author and the signing entity has seen at least the binary executable of the application. Additional expressiveness of a cryptographic signature depends on the signing program and varies. In their basic form, the author states in a contractual form some properties of his application. These properties are tested only in some cases. In any case, a benefit of signed applications is the possibility to revoke the signature.

As a first remark: one could ask what the signing programs are good for when even spyware can get signed [220]. And as a second remark: in case of damage—especially monetary damage—the affected parties (user or MNO) have to prove the cause of the damage. This is not possible with current signing schemes.

2.3 Mobile Malicious Software

This section introduces the definition and presents surveys of mobile malicious software. Moreover, it presents virus scanners as the defense side of malicious software.

2.3.1 Definition of Malware

Malware, composed of the words *malicious software*, is the usual form in which vulnerabilities and their exploits manifest themselves to become security problems

for a broad audience. For the most parts of this thesis, the umbrella term malware will be used for the investigated topics and samples. For completeness, this section shows definitions with more detail regarding the functionality of the malware.

A *virus* has its name according to its analogy in the real world: it infects other files that serve as a host to it. Its execution is bound to the execution of its host. Most of the times, the functionality of the host is not changed except for the additional virus parts.

A *Trojan horse* or *trojan* is a program with malicious side-effects. It fulfills its intended purpose, but sometimes it executes functionality that was not documented or advertised. A *backdoor* is a special type of Trojan horses whose malicious functionality is providing access to the phone for the attacker. The attacker can use this access for committing malicious actions from the phone, while at the same time hiding his identity from the victim.

A *worm* is characterized by its self-replication property. It does not infect other files like a virus and does not promise to provide legitimate functionality as a Trojan horse. It is possible to say that a worm *infects the entire host* rather than single files on the host.

The term *spyware* denotes a kind of malware that aims at gathering information about the mobile device user, contrary to creating revenue for the attacker or damaging the user's data on the device. It is common that spyware is commercial software.

It is possible to define the notion of a *malware family* that subsumes malware with similar functionality. It might be the case that every member of a malware family exploits the same vulnerability of an program for infection.

Chapter 4 will present additional classifications of malware, tailored to mobile malware there.

2.3.2 Surveys of Mobile Malware

This section names in a chronological order the most relevant surveys of mobile malware as of today. Peikari [170] gives an overview of Windows Mobile and Symbian OS malware. An extensive article covering nearly all malware of its time of writing was given by Shevchenko in the year 2005 [197]. Bachfeld [13] gives an overview of the filtering activities of German mobile operators of the year 2005 and assumes that the malware threat is currently only relevant from the perspective of anti-virus companies. It ends with a test of Symbian OS anti-virus programs. A book by Eren and Detken [52] lists the known malware samples until 2006, surveys the weaknesses of mobile operating systems, and describes much of the mobile

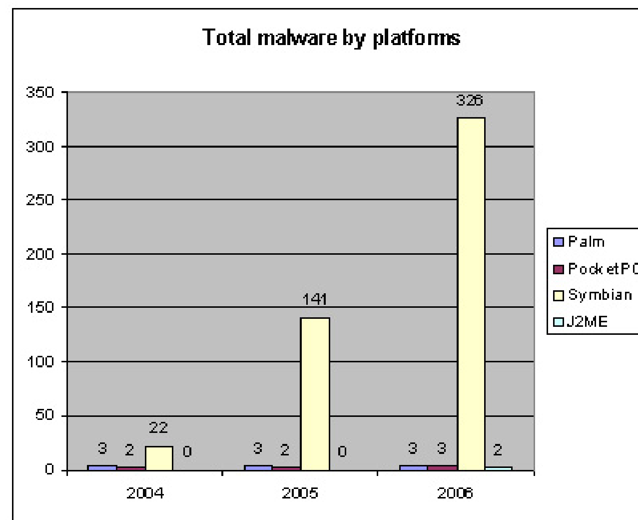


Figure 2.5: Operating Systems Targeted by Mobile Malware (according to F-Secure [48])

and the mobile device security knowledge of that time. Töyssy and Helenius [215] list infection routes and some examples of malware of the year 2006, but their focus is on countermeasures and media perception of mobile malware. F-Secure published statistics of operating systems that are targeted by mobile malware (see Figure 2.5).

Bontchev [25] notes mobile malware classification problems and chooses Symbian OS malware as an example. Although not explicitly stated, his findings can be generalized for malicious software on any operating system.

A survey in the "Scientific American" was given by Hypponen in 2006 [100]. Besides a summary of mobile device security knowledge of that time, it shows in an illustrative comic cartoon that many repetitions of an installation attempt (via Bluetooth) could even break down the resistance of a security-conscious user. A mobile malware summary was given by Hypponen at the Black Hat conference in the year 2007 [101].

McAfee published a study in 2007 as a result of surveying mobile network operators [129]. This survey shows how mobile network operators start preparing defenses against mobile malware. The most recent surveys as of the time of writing of this thesis are given in 2009 by Morales [48, Chapter 3] and Schmidt et al. [188].

2.3.3 Virus Scanners

The concept *malware detection* with the incarnation virus scanners is a reactive aspect of security. It is known to most of the users of information technology, even if they are not very security-conscious otherwise (see Section 3.5). Despite the criticism of virus scanner in this section, the topic has its place in the security landscape because the user has accepted the interface that virus scanning provides. Thus, novel security solutions can use the terms of virus scanners for increased acceptance.

This section introduces to the concept of malware detection, which is usually subdivided into two parts: signature-based and behavior-based [213]. The section will especially point out the weaknesses of the applied technologies, facts that the above-mentioned users usually do not know or do not take into consideration.

The classical formal treatments of malware detection are from the early nineties by Adleman [2] and Cohen [36]. Singh and Lakhotia [199] listed detection approaches in an annotated bibliography in the year 2002. More references to different detection algorithms as of 2004 are given by Christodorescu and Jha [34] and as of 2006 by Aycock [12].

2.3.3.1 Signature-Based Detection

Signature-based detection of malicious software was the prevalent approach of malware detection in the recent years. It is based on the assumption that every malware sample contains a unique signature that can be detected when scanning an infected file. Now, every file can be checked against a signature database. In theory, the only remaining problem is the time gap between the first appearance of a malware in the wild and the addition of a signature to the signature databases (cf. Figure 2.3). However, there are other deficiencies that are named now.

Detection Rate. Recent studies investigated the detection rate for a large number of malware samples. One result was that currently available virus scanners have poor detection rates for malware actually spreading in the wild [16, 142]. One explanation for this might be that not all signatures are checked in a normal run because of runtime efficiency and the efficiency of distributing an updated database via a communication network. Because of the lower resources of mobile devices (processing power, data communication speeds), an on-device virus scan is likely to have even more limitations.

Behavior-Preserving Binary Changes. Another deficiency of the signature-based approach are small changes in the malware sample that can even be applied to its binary version without access to the source code. This can be done in two levels of sophistication: string changes and replacing some machine code instructions by semantically equivalent instructions.

The first change can be as simple as changing a human-readable string within the malware binary. This can be the output of the malware and it might be the string that forms the signature [142]. This can be done with an ordinary editor. More semantically equivalent versions can be created with access to the source code. An example is a changed name of a variable.

In case the malware sample is not encrypted in some form, it is possible to replace machine code instructions by semantically equivalent instructions. Another possibility is the addition of NOP (no operation) instructions or some of their numerous equivalents like "add rx, 0" into the binary. These simple changes lead to malware variations with the same behavior as the original malware sample, but they have been found to be largely undetected by current virus scanners [34, 142, 144].

Conclusion. Signature-based malware detection still is the main component of current virus scanners today [27], mainly because alternatives usually have a higher false-positive rate. A currently active topic of research is behavior-based malware detection, which starts to be integrated in virus scanner products [191].

2.3.3.2 Behavior-Based Detection

With the rapidly increasing numbers of malware and the associated problems with signature-based detection, the topic of behavior-based detection became an important research topic. Current tests of virus scanners note that behavior-based detection is increasingly integrated into virus scanners [191], at least on common computers.

Behavior-based malware detection collects *behavior data* at some level, e.g., system calls. This data is classified or abstracted and finally used to distinguish legitimate behavior from malicious behavior.

The enforcement of security policies during runtime given by Schneider [192] can be seen as an implementation of behavior-based detection and a first work on the topic. Other early work came from the research area intrusion detection. Anderson [5] has broad view of intrusion detection as a general notion of computer security and defines virus scanners as an element of this notion. From host-based intrusion detection came contributions where attacks can be investigated at the deeper level of

the operating system. In different levels of sophistication, publications contributed to solve the fundamental of host-based intrusion detection systems that use system call detection [31, 133, 150]. These results can be useful for any solution that is able to log system calls (see Chapter 5).

A comprehensive introduction to more literature and to the current state of the topic in the year 2008 is given by Morales [141]. For this type of malware detection, a form of "normal behavior" has to be modeled. There are two possibilities: to describe the normal (legitimate) behavior ("whitelisting") and to model malicious behavior ("blacklisting"). The commonly used method of blacklisting (signature-based detection) is in general a bad concept [99] that is only of limited use, even if mainly applied in practice (virus scanners, firewalls). Behavior-based detection offers from its concept the possibility to implement whitelisting for virus scanners.

Behavior-based malware detection is a promising approach, because the restricted environment should lead to more precise models of legitimate or malicious behavior. A recent contribution by Bose et al. [27] provides a behavior-based detection especially targeted to mobile devices.

Approaches for "collaborative virus detection" were proposed (Cheng et al. [33]) that use the accumulated statistics of short message sending activity for detecting malicious behavior. This bird-eye view reduces the influence of the behavior of a single device to the algorithm.

A recent contribution [187] extends this approach to implement behavior-based malware filtering within the network by using feature vectors of device activity, e.g., CPU and RAM access activity. The task of the device is reduced to gathering the feature vectors. Contrary to the previously named message statistics, the vectors have to be sent regularly into the network for applying detection algorithms to the vectors. The overall goal is to identify a malware-infected device by evaluating these feature vectors, but the system increases the workload of the mobile device (that has only limited resources).

The drawbacks of behavior-based virus scanners can be summarized as: "While a behavior blocker knows which executable is the problem, unlike an integrity checker, it again cannot identify or disinfect the virus. Run-time overhead and false positives are a concern, as is the fact that the virus is already running on the system prior to being detected." [12]

2.3.3.3 Virus Scanners on Mobile Devices

Virus scanners on mobile devices differ from common virus scanners mainly in two points: first, there are two possible locations for scanning (device and mobile

network), and second, communication of the device with the network is expensive (in the meaning of computationally expensive).

Scanning on the mobile device can be done with the resources of the mobile device itself or with the help of a host computer, e.g., when the device is connected for synchronizing. The host computer provides more computing power and less battery dependence, thus being able to scan more efficiently. However, the time between two subsequent connections to the PC might be too long to effectively protect the device and the host computer might have a too restricted access to the file system of the mobile device.

Scanning within the mobile network creates privacy problems because the content of the scanned data has to be processed (e.g., the content of short/multimedia messages, e-mail). Another problem is the reaction when a message was identified as malicious: is the MNO allowed to discard the message, possibly without notifying the user? This is not advisable, because false positives may cause acceptance problems of such a service. Thus, it is at least necessary to notify the user of a detected message, possibly with a message preview as common virus scanners sometimes do. However, such a preview seems only feasible for e-mail messages, not for MMS messages. A possible solution is a notification with reference to a "malware preview page" in the MNO's Web frontends. With the upcoming rise of the mobile Web browser this page would even be accessible via the mobile device as a seamless service between the messaging application and the Web browser, but providing a safe environment for reviewing the message [162]. An additional drawback of scanning within the network is the inability to detect malware that infects the device from local interfaces like Bluetooth (this will be discussed in Chapter 7).

Incarnations of virus scanners on mobile devices are tested from time to time with poor results: the false-negative rates are almost 50 % [142].

2.4 Analytic Mechanisms

This section introduces the two analytic mechanisms dynamic software analysis and simulation. The topic of dynamic software analysis is a basis for the Mobile-Sandbox dynamic malware analysis tool of Chapter 5. The methods of simulation are used in Chapter 7.

2.4.1 Dynamic Software Analysis

The main idea of dynamic analysis is executing a given sample in a controlled environment, monitor its behavior, and obtain information about its nature and purpose. This can be done for analyzing normal software analysis or malware, and it is especially important in the field of malware research because a malware analyst must be able to assess the threat of an investigated program and to create proper countermeasures.

While static analysis might provide more precise results, the sheer mass of newly emerging malware each day makes it impossible to conduct a static analysis for even a small portion of today's malware, because static analysis is a manual task of the malware analyst. Dynamic analysis is an automatic task of a software. The malware analyst only has to analyze the results of this software and even parts of these tasks can be automated, e.g., extracting botnet access information. The numbers of malware are different for mobile devices today (especially for Windows Mobile, see Section 4.1), but it is always good to develop tools for the future (like MobileSandbox in Chapter 5).

Recently, much work has been done in this direction for Windows malware. CWSandbox [232] and TTAnalyze [17] monitor the execution of one sample and log the activities during one particular run. Even though this method misses more sophisticated malware that has some additional requirements for showing its malicious behavior, the absolute number of detected malware is still large because of the vast overall number of malware for Windows [91]. The adaption of these methods to Windows Mobile is possible [123].

Recent work enhances the dynamic analysis by being able to analyze multiple execution paths [143], which is a promising step towards a complete dynamic analysis, even though the approach has some limitations. They apply tainting to all variables that come from a defined set of external inputs of the program and are able to process all linear modifications of variables that are relevant for conditional branches. They use a linear constraint solver to achieve the goal of reaching the alternative branch, setting the emulated execution back to the relevant starting point. Another approach is "forced sampled execution" [231], where heuristics are applied to reach the alternative branches.

The overall goal of dynamic software analysis is being as complete as possible, because this would relieve all the drawbacks of static analysis like scrambled or self-modifying binaries.

2.4.2 Simulation

This section introduces the topic discrete event simulation and related work that uses simulation for investigating the spreading characteristics of mobile malware. This is the basis for our own simulation of mobile malware spreading characteristics in Chapter 7.

2.4.2.1 Discrete Event Simulation

The central notion of discrete event simulation is the *event*. Events are created by the active entities in the system and they are collected in the *event list*. Based on a *global clock* the simulation system processes events from the event list, leading to state changes of the system. The system state is defined by the set of *state variables*. A special state variable is the *end condition* that will end a simulation run if its logical value is true. Other important parts of the system are random number generators and the logging of events and state changes for statistic evaluation of a simulation run.

At the start of a simulation run the global clock is set to zero and the state variables and counters are set to their initial values. Additionally, the end condition is set to false and a first event is created for being processed.

This was the most basic knowledge on discrete event simulation for understanding the contribution of Chapter 7. A more detailed introduction can be found in the literature [202].

2.4.2.2 Simulation in Mobile Device Security

Various recent articles on investigating the spreading of mobile malware exist. They are the basis for our own investigations.

A preliminary investigation of worm infections in a Bluetooth environment was given in 2006 by Su et al. [204]. They simulated the spreading of Bluetooth worms by varying the parameters *number of infection seeds* and *initial time of the outbreak*. They conclude their simulation that Bluetooth worms have the potential of "infecting a population of 10,000 devices over a few days only".

An application of simulation for investigating malware propagation in mobile phone networks was given in 2007 by Fleizach et al. [73]. The authors have developed a simulation environment and a network topology generator, both for mobile phone networks. They build a virtual network and simulate the spreading of malware over a VoIP client and MMS with different parameters like propagation

speed and size of address books. With their simulation system design they conclude that one danger with aggressively spreading mobile malware are bottlenecks of the capacity of internal links.

An effort for quantifying the effectiveness of mobile phone virus response mechanisms was given in 2007 by Ruitenbeek et al. [183]. They have a simulation system with four different types of MMS viruses with varying spreading aggressiveness. Six response mechanisms are simulated with these viruses, subdivided into three groups: at the point of reception, at the point of infection, and at the point of dissemination.

The first group is virus response mechanisms at the point of reception: virus scan of all MMS attachments in an MMS gateway and virus detection algorithm in an MMS gateway (more general than the former because even unknown viruses can be detected). The second group is virus response mechanisms at the point of infection: phone user education and immunization using software patches. The third group is virus response mechanisms at the point of dissemination: monitoring for anomalous behavior and blacklist phones suspected of infection.

The elements of the first group are common approaches, also using software patches of the second group. The effects of educating phone users are discussed throughout this thesis, mostly in Section 3.5. The elements of the last group belong to the part anomaly detection of intrusion detection research. Their success in a real-world scenario is questionable because of the inherent problems of intrusion detection systems. They conclude that "an optimal response strategy must incorporate mechanisms to counteract a wide variety of virus behaviors".

A recent contribution for understanding the spreading patterns of mobile phone viruses was given in 2009 by Wang et al. [225].

2.5 Distributed Computation

This section introduces to the topic of distributed computation problems. It presents secure multiparty computation (SMC) and fair exchange. They are presented in this chapter because solutions to solving distributed computation problems help to increase the security of the participants by providing confidentiality, integrity, and sometimes non-repudiation.

There are two possible approaches for solving distributed computation problems: by using a *trusted third party* (*TTP*) and by only using the participants. Having a TTP makes solutions to SMC easier. The more elegant solutions of SMC virtualize the TTP among the participants.

In the setting of this thesis, the mobile network operator could serve as a TTP (also proposed by Jøsang and Sanderud [111]). However, even in simple use cases participants with different MNOs are involved, requiring administrative efforts between the MNOs for the solution. Therefore, solutions with a virtual TTP are worthwhile. This section introduces algorithms that use a virtual TTP. An implementation of these algorithms on mobile devices using the MNO smartcard as trusted module (cf. Section 2.1.2) will be presented in Chapter 8.

2.5.1 Secure Multiparty Computation

2.5.1.1 Problem Definition

Secure multiparty computation is a hard problem. Informally, it is defined as: "A set of parties wants to compute a common function F on their local inputs, while keeping their local data as private as possible, but who do not trust each other, nor the communication channels" [37]. A formal definition follows (with an *F-result* defined as the result of the function F).

Definition: A protocol solves SMC if it satisfies the following properties:

- *Validity*: If a process receives an F-result, then F was computed with at least the inputs of all correct processes.
- *Agreement*: If some process p_i receives F-result r_i and some process p_j receives F-result r_j then $r_i = r_j$.
- *Termination*: Every correct process eventually receives an F-result.
- *Privacy*: Faulty processes learn nothing about the input values of correct processes apart from what is given away by the result r and the input values of all faulty processes.

A *synchronous system* has known and bounded timing parameters for the network. In an *asynchronous system*, the timing parameters can be arbitrarily long. A *partially synchronous system* means that bounds on all important network parameters eventually hold [37].

2.5.1.2 SMC in Synchronous Systems

There are three interesting implementations of SMC for synchronous systems today: FairPlay, Proactive-Reactive Recovery, and TrustedPals.

FairPlay. The FairPlay system was implemented by Malkhi et al. [125] for a two-party computation setup. An extension for multiparty computation was given with FairPlayMP in 2008 by Ben-David et al. [21]. The main drawback of the FairPlay implementations are the computational resources that the system needs and the special programming language SFDL (secure function definition language), which is used for programs that instruct a virtual trusted third party.

TrustedPals. TrustedPals solves SMC for an arbitrary number of participants by using smartcards. The system is subdivided into two parts: the untrusted host system and the trusted system inside the smartcard (security module). The security modules build a virtual trusted third party. A first version of TrustedPals [76] used a monolithic approach and assumed a synchronous network setting. A second version was designed for asynchronous network settings, which will be introduced below.

Proactive-Reactive Recovery. Sousa et al. [201] propose a system that is similar to TrustedPals but with a different attack model. They assume attacks on the hosts as only possible from outside the hosts. In TrustedPals, the host itself can be the attacker (the Byzantine node).

2.5.1.3 SMC in Asynchronous Systems

A second version of TrustedPals takes a modular approach with a consensus algorithm and a failure detector algorithm, enabling a use in network settings with less synchrony [37]. Necessary notions for this version of TrustedPals are defined subsequently.

In- and Out-Connected Processes. An important definition are in- and out-connected processes. They are defined as *processes that can only receive messages* (in-connected) and *processes that can only send messages* (out-connected). This distinction can be made in TrustedPals because of its setup of the failure detector.

Consensus. Every process proposes a value and correct processes must eventually decide on some common value that has been proposed [169]. It is defined by the properties termination, validity, and agreement, which are slightly modified for the setup in TrustedPals.

- *Termination*: Every in-connected process eventually decides some value.

- *Integrity*: Every process decides at most once.
- *Uniform Agreement*: No two processes decide differently.
- *Validity*: If a process decides v, then v was proposed by some process.

The consensus algorithm is subdivided into four phases:

- *Phase 1*: All processes send their input value to the round leader.
- *Phase 2*: If the round leader gets an input value from the majority of all processes, it sends the most current input value to all processes. If the round leader does not receive a sufficient number of messages, it will send a NEXT message to all processes. This leads to the start of a new round with the next process as round leader.
- *Phase 3*: If processes receive a value from the round leader, they reply with an ACK message to the round leader.
- *Phase 4*: If the round leader receives an ACK message from the majority of all processes, it sends the value again with the addition to decide that value. Afterwards, all processes decide the value of phase 4, if they have received the last message.

Consensus cannot be solved in an asynchronous system, because very slow processes cannot be distinguished from stopped processes. However, it can be solved in a synchronous or partially synchronous system [49].

Failure Detector. The task of a failure detector is to create synchrony in an asynchronous system. That is, it encapsulated timing assumptions. It holds information about the status of the other processes. The TrustedPals version uses $n \times n$ matrices for n processes instead of simple "alive" messages, enabling a process to gain more information about communication possibilities between the processes.

TrustedPals uses the "eventually perfect" failure detector. It fulfills the following three properties:

- *Strong completeness*: Every process that is not out-connected will not be permanently considered as out-connected by any in-connected process.
- *Eventual strong accuracy*: Eventually every process that is out-connected will be permanently considered as out-connected by every in-connected process.
- *In-connectedness*: Eventually every process will permanently consider itself as in-connected iff it is in-connected.

Weak SMC. Compared to only one bit as an input value, SMC in completely asynchronous systems is a difficult problem. SMC with arbitrary input values in asynchronous systems can only be performed as a weak SMC [132]. Each input value requires its own consensus round r_i, where the input value of process p_i is distributed. In case of asynchronous systems it may happen that a process p_i is not able to distribute its value in round r_i. Even though p_i is not crashed, it was not able to participate in the SMC. This leads to different success probabilities for the participating processes, depending on their position i in the consensus rounds.

2.5.2 Fair Exchange

Fair exchange can intuitively be defined as: two or more mutually untrusted parties want to exchange secrets in such a way that either all of them receive the desired information or none of them learns anything valuable.

2.5.2.1 Problem Definition

A fair exchange protocol fulfills the following requirements [7, 168]:

- *Effectiveness*: If both parties follow the protocol and do not want to abandon the exchange and the items match their known description, then A has i_B, B has i_A, and both reach a success state upon completion.
- *Termination*: A party which behaves according to the protocol will eventually reach either a *success* or *abort* termination state.
- *Strong Fairness*: If at least one party does not follow the protocol or an item does not match its description, then no honest participant wins or loses anything of value.

An additional requirement can be [7]:

- *Non-repudiability*: After an effective exchange, any party A will be able to prove *non-repudiability of origin*, meaning that the item i_B originated from B, and *non-repudiability of receipt*, meaning that B received i_A.

Non-Repudiability is useful in subsequent disputes after a fair exchange, where it may be necessary to solve the conflict outside of the system, e.g., by a court of justice.

Also, the exchanged items can be classified:

- *Generatability*: An item is generatable if it can be produced by a third party (an arbiter) upon request.
- *Revocability*: An item is revocable if it can be revoked by the arbiter and thus rendered useless for the receiving party.

2.5.2.2 Impossibility Result of Fair Exchange

Even and Yacobi [54] formally proved the *impossibility result* of fair exchange that no protocol exists that solves fair exchange in the following setting: A and B want to exchange digital items in a sequence of communication rounds. At some point during the exchange—namely after n rounds—B will have received sufficient information to reconstruct the desired item i_A. However, due to mutual distrust A will only give away that amount of information and participate in that round if it has already received enough information to reconstruct i_B in a previous communication round $m < n$. This contradicts the fairness requirement of fair exchange.

Therefore, it is only possible to specify a fair exchange protocol by utilizing the help of a *trusted third party* (TTP), also referred to as the *trustee*. Of course, the involvement of another participant poses new problems to be solved. An arbiter must possess certain properties in order to be eligible as a TTP. Two aspects are particularly relevant, namely *trustworthiness* and *availability* [168].

Concerning trustworthiness, both parties A and B need to be sure that the trustee neither teams up with the other party, nor that it has an agenda of its own, e.g., stealing the items or eavesdropping on the conversation between A and B. This means it has to be ensured that the TTP follows its specified protocol precisely. Concerning availability, the parties need assurance that the TTP will be available to carry out the exchange as expected. This is especially important in the exchange of time-sensitive items. In asynchronous systems, the TTP must ensure to be at least *eventually up* and able to serve all incoming requests. The higher the reliability requirements an application puts on the TTP, the harder it is to ensure a certain implementation will be able to fulfill these demands. Trust is therefore usually not really towards a TTP itself, but rather towards its manufacturer or independent experts.

2.5.2.3 Fair Exchange Protocol Classes

As solving fair exchange without a trusted third party is impossible and the introduction of a TTP as a new entity in the protocols leads to increased complexity,

research in the area of fair exchange focuses on how to reduce the involvement of the trustee to a minimum.

Early solutions for fair exchange fall into one of two categories: third party protocols with an *online* trusted third party and *gradual exchange protocols* [7]. In online TTP protocols, both parties A and B send their items to the trustee. After verifying the correctness of the items, the trustee forwards both items to the corresponding recipients. It can easily be seen that this solution causes a high workload on the trustee and heavily relies on its availability. In gradual exchange protocols, the probability of correctness is increased over several rounds of communication. This leads to a communication overhead, because items are only transferred partially in each round.

Recent work introduced other possible solutions: *optimistic third party protocols* use an offline TTP. In these optimistic protocols, the participants perform the exchange of their digital items without outside help. In case of a system failure or in case of one party suspecting unfair behavior, the TTP is asked to collect information on the exchange and—if possible—lead the exchange to a state where fairness is achieved. Unfortunately, optimistic protocols require items that are either generatable or revocable. As such, optimistic protocols cannot be used in any fair exchange scenario, at least not without tolerating loss of fairness or termination.

Until now, the presented fair exchange protocols rely on the continuous availability of a trusted third party, either actively involved in the exchange or available for dispute resolution. This has several drawbacks. Apart from the difficulties of designing an external host in such a way that it reliably provides trustworthiness and availability, factors such as network topology and user conduct may pose even greater challenges to third-party protocols. Therefore, it is beneficial to explore how the duties of a TTP might be delegated to other entities that operate closer to the parties involved in the exchange.

One approach that has gained attention in literature is the use of *tamper-proof hardware* or *security modules* [10, 11, 222, 223]. By using security modules, it is possible to solve fair exchange of time-sensitive items as well as fair exchange of arbitrary items (which are not generatable or revocable), and fair exchange can be reduced to other well-known problems in distributed computing. For example, Avoine et al. [10] introduced the *gracefully degrading fair exchange* protocol that offers probabilistic fairness in case of a honest majority. The following fair exchange protocol with guardian angels provides equal properties in the case of two-party fair exchange, where the honest majority is only given when both participants are honest.

2.5.2.4 Fair Exchange with Guardian Angels

Avoine and Vaudenay [11] proposed an approach to have security modules assist in the construction of a fair exchange protocol. They proposed a model of *pirates* (the hosts) and *guardians* (the smartcards). By reducing the fair exchange problem to a synchronization problem known as *non-blocking atomic commit*, they are able to construct a protocol that offers probabilistic fairness without assuming a trusted third party.

Communication between the pirates is suspected to be insecure and to provide no timeliness guarantees. In contrast, every host may communicate to its local smartcard in a secure and timely fashion. Guardians are even able to communicate to each other in a secure manner over the unprotected communication network. This is possible by implementing message confidentiality, integrity, and authenticity with cryptographic techniques.

Avoine and Vaudenay introduced a *keep-in-touch* (*KiT*) synchronization protocol. It basically enables two participants A and B to agree on a specific time to terminate their connection. A sends a termination request to B, stating an amount c of messages to be exchanged before termination. Both parties then start sending (empty) messages back and forth until the number c is reached. In that case, they both end in a success state. In case of a time-out while expecting a message, a participant ends in a failure state and stops sending messages. In this case, the other party also times out and considers the protocol to have failed. The probability of successfully manipulating this protocol depends on c and the probability distribution that c was derived from.

An attacker would have to correctly guess the number c of messages and disrupt the network connection when the last message is sent. Only in that case, the sending party would suspect success while the waiting party would time out, leading to a failure state. Since the communication between A and B is suspected to be secure, an adversary may not learn c from eavesdropping, neither may he insert fake messages or disturb communication in any way other than disabling communication completely.

The authors then apply this synchronization protocol to fair exchange as follows: both participants send their respective items, as well as a description of the desired item to their local guardian. The guardians exchange the items over their secure communication channel. After verifying the correctness of the items received, they perform the keep-in-touch protocol. If this is successful, they deliver the items to their local hosts. Otherwise, the exchange has to be re-initiated. Therefore, the fair exchange protocol delivers probabilistic fairness equal to the probability of successfully running the KiT protocol.

Chapter 3

Structuring Mobile Device Attack Vectors

While the previous chapter used common denominations for its topics, this chapter contributes an attack model with four attack classes and argues for a clear distinction between the classes. This attack model in Section 3.1 serves as the framework for the division of the rest of this chapter. Its parts concerning attack vectors exploitable by mobile malware will be used in the rest of this thesis together with the specifics of mobile device security in Section 2.1.3. The chapter shows that the increasing functionality of mobile devices (leading to smartphones, cf. Section 2.1) brings more possible attack vectors to them. It presents in detail what we call the *operational side* of mobile device security, that is, vulnerabilities exploitable by mobile malware. The attack vector classes can be used as a framework to evaluate future vulnerabilities and their exploits according to their potential to be used by mobile malware.

The MOBIUS project defined a threat model for mobile devices [137, Chapter 3]. Besides a MIDP threat model it defines (according to the focus of the project) common attacks on information flow security and common attacks on resource control. Instead of a classification of attack vectors, this work is more focused on the *potential damage* that an attacker can commit on the mobile device. This chapter extends this work for *attack vectors*.

The relevant document from an industry perspective is the OMTP "Security Threats on Embedded Consumer Devices" [166]. It defines six threat categories: software modification threats, software opportunistic threats, and four type of hardware threats (external, terminal intrusive, component invasive, hardware cloning). The software-opportunistic threats are comparable to the software-centric attacks of this chapter, the hardware threats are comparable to the hardware-centric attacks. As

another related work: an informal brainstorming on finding attack vectors of mobile device security was done by Whitehouse [229]. To the best of our knowledge, no other classifications of mobile device attack vectors specifically aimed at finding out mobile malware attack vectors exist. This chapter differs from the previous two pieces of related work by giving examples of real-world incidents for each of the attack vectors.

The chapter is structured according to the classes of mobile device attack vectors of Section 3.1. Section 3.2 lists local attacks that are only relevant to a particular mobile device but not suited to be exploited by mobile malware. The remote attacks of Section 3.3 are not related to the security level of a particular mobile device, but they cannot be exploited by mobile malware either.

The following sections show the field where mobile malware can attack: vulnerabilities of mobile device applications in Section 3.4 and the impact of the mobile device user on mobile device security in Section 3.5.

3.1 Attack Vector Classes

This section presents a classification of mobile device attack vectors as a framework for the rest of the chapter. Its intention is to show the relevant attack vectors that can be used by mobile malware.

Mobile device threats are classified here as belonging to one out of four classes: *hardware-centric attacks*, *device-independent attacks*, *software-centric attacks*, and *user layer attacks*. From the point of view of defending against vulnerabilities, every layer is separate from the other and needs its own security mechanisms.

Figure 3.1 shows a high-level view of these four attack vector classes and Figure 3.2 shows the actual attack vectors of these classes that will be introduced in the rest of the chapter.

Hardware-centric attacks belong to mobile device security only from a broader point of view but not from the definition of this thesis. Even though they are suited to violate security properties (e.g., confidentiality of personal data violated by forensic analysis) they are not suited to be exploited by mobile malware, because these vulnerabilities cannot be exploited remotely, but only with physical access to the mobile device.

Device-independent attacks directly belong to the protection targets of the mobile device user: eavesdropping on the wireless connection or leaking mirrored personal data on backend systems both violate confidentiality of the user's personal data. However, just as device-centric attacks they are not vulnerabilities that can be

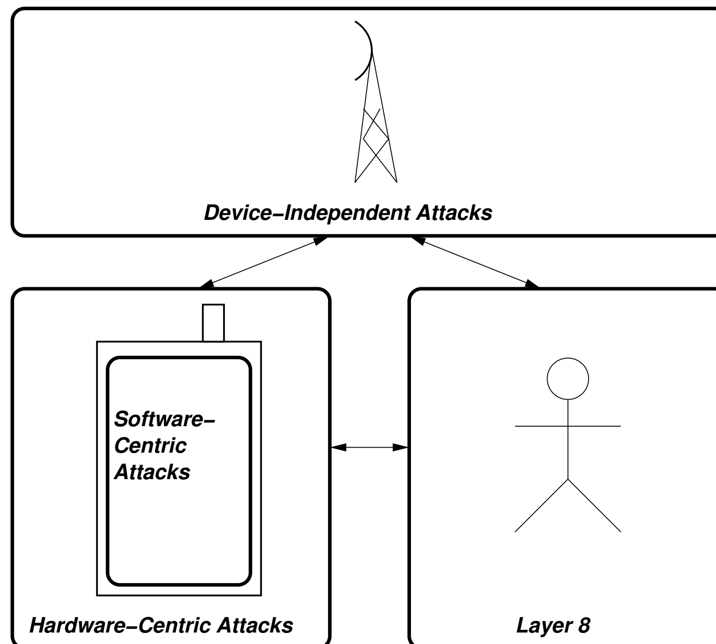


Figure 3.1: Mobile Device Attack Vectors (High-Level)

solved with the solutions of this thesis, as these threats are independent from the security level of a particular mobile device.

In the context of this thesis, the most important class of technical vulnerabilities for mobile devices are *software-centric attacks*. Especially the rise of the—hardly security-specified—mobile Web browser led to various exploited vulnerabilities in the recent time.

User layer attacks contain every exploit that is not of technical nature. As Chapter 4 will show that many of today's mobile malware samples are not based on a technical vulnerability, but trick the user into overriding all technical security mechanisms. This is an important class of vulnerabilities, even if not of technical nature.

3.2 Hardware-Centric Attacks

These hardware-centric attacks belong to the security of mobile devices only from a broader point of view. Even though they are suited to violate security properties (e.g., confidentiality of personal data violated by forensic analysis), they are not suited to be exploited by mobile malware because these vulnerabilities cannot be exploited remotely, they need physical access to the mobile device. They are

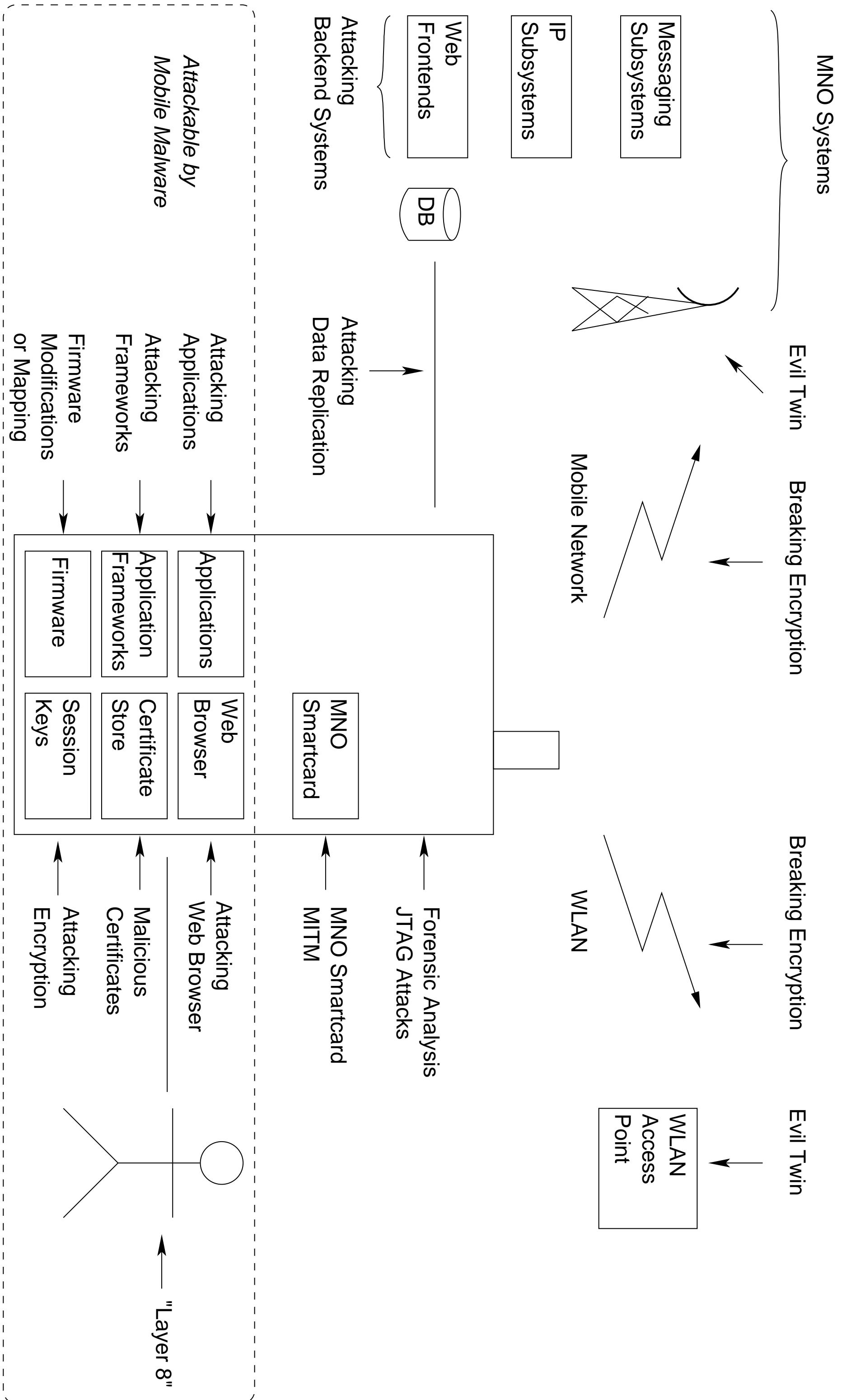


Figure 3.2: Mobile Device Attack Vectors (Incarnations)

subdivided here into attacks on the removable security module of the mobile device (the MNO smartcard) and into attacks on the device itself.

3.2.1 Intercepting MNO Smartcard Communication

Communication between the mobile device and the MNO smartcard is unencrypted, because a man-in-the-middle (MITM) attack on this communication was considered as infeasible when this interface was specified

A product named TurboSIM successfully implements the MNO smartcard MITM attack of Figure 3.2. TurboSIM is a product of the Czech company Bladox [24] and is offered since 2004. It is a small chip that intercepts the communication between the MNO smartcard and the mobile device. It is added to a mobile device by removing a small part of the smartcard's plastic frame. TurboSIM was successfully applied to removing the SIM lock of the iPhone [22]. As the hardware interface is the same for 2G SIM cards and 3G UICCs, it is possible to use TurboSIM for both settings.

Without regarding the limitations of the actual implementation of TurboSIM, in general, such a MITM attack can change every communication between MNO smartcard and mobile device, or even inject new communication. The only relief for this attack would be to encrypt the communication. This would work because this piece of hardware has access to the internals neither of the MNO smartcard nor of the mobile device.

It is difficult to close this attack vector with billions of vulnerable devices worldwide. From a high-level point of view it is a task for engineering. Fortunately for the context of this thesis, this attack is not targeted to damage other users but to enhance the functionality of the attacker's phone. As it is an attack with the introduction of additional hardware, it cannot be exploited by mobile malware either. Therefore, it is no threat that will be dealt with in this thesis.

3.2.2 Attacking the Device

Hardware-centric attacks that target the mobile device itself can be subdivided according the the status of the mobile device: switched on (JTAG attacks) or switched off (forensic analysis).

3.2.2.1 JTAG Attacks

Joint Test Action Group (JTAG) is a standard for testing and debugging hardware. Even though this debugging functionality is no longer necessary in mobile devices that are sold to the users, the JTAG functionality is sometimes still accessible. This functionality allows inspecting the device on a deep level being able to lead to exploitable vulnerabilities.

This threat is addressed by industry requirements documents [164] and as it is an attack that needs physical access to the device, this attack vector cannot be exploited by mobile malware.

3.2.2.2 Forensic Analysis

The forensic analysis of mobile devices is an attack vector targeting the confidentiality of the stored data. It is an unexpected attack vector and it is only valid in the case of an attacker getting physical access to the device. There are two possibilities for that: an attacker that takes the device for a limited period of time without the owner noticing, and a legitimate change of ownership. Especially the second case is common today, as some studies show: it encompasses data from personal conversations [78] to confidential corporate data [219].

From a high-level point of view, this attack vector can be closed quite easily by just adding sound encryption schemes to the data. In both cases from above, an encryption scheme is sufficient that encrypts the data whenever the device is switched off, because a thorough forensic analysis requires removing the device's battery (access to the data using means like data cables should also be restricted). Dealing with the solution in more detail leads to the consideration that cryptographic functions need the limited device resource processing power, leading to increased battery usage. Therefore, the two alternatives must be weighed against each other.

3.3 Device-Independent Attacks

These vulnerabilities directly belong to the protection targets of the mobile device user: eavesdropping on the wireless connection (Section 3.3.1) or leaking mirrored personal data on backend systems (Section 3.3.2) both violate the confidentiality of the user's personal data. However, just as the local attacks of Section 3.2 they are not vulnerabilities that can be solved with the solutions of this thesis, as these threats are independent from the security level of a particular mobile device. Just as the device-centric attacks of Section 3.2 they cannot be exploited by mobile

malware either. An exception could be the wireless pairing process, which could be influenced by a mobile malware, e.g., by forcing the device to connect to an evil twin access point. However, this is seen as irrelevant here, because WLAN only is a side aspect in this thesis.

3.3.1 Wireless Transmission Security

Wireless transmission security belongs to the confidentiality of the wireless link between the mobile device and the MNO, be it voice calls or a data transmission. The topic is subdivided into cryptanalytic attacks against the encryption mechanisms and into imitating parameters of a legitimate communication access node.

3.3.1.1 Breaking Encryption

Breaking Mobile Network Encryption. The encryption in the GSM system uses the A5 family of algorithms. These algorithms have been defined without public review prior to their implementation ("security by obscurity"). Mainly two variations of the algorithm are deployed today: A5/1 and A5/2, the latter being a cryptographic weaker version for countries with legal restrictions on usage of cryptographic schemes. The cryptographic stronger version A5/1 was reverse-engineered and published in 1998 [200]. Since then it has been investigated by the public cryptanalytic community and several attacks have been published with the recent addition by Gendrullis et al. [84] of increasing the implementation speed of an attack to only six hours for revealing the internal state of the A5/1 algorithm by using a low-cost special-purpose hardware. The publication also introduces to the related work of this research area, and the interested reader is referred to their article.

It has to be noted that the mobile network encryption key is not only endangered by cryptanalysis. Session keys are derived from the master key and stored outside the MNO smartcard. Thus, mobile malware could access these keys for decrypting the communication without breaking the algorithm. Practical restrictions exist with this attack vector, because the storage location of these session keys is not specified, meaning that it is dependent on the implementation of a specific operating system. Therefore, it is not clear if mobile malware is able to access these keys at all.

As a side-note: from a high-level lifetime view of cellular mobile networks: one could argue that with the speed at which the weaknesses in the A5/1 algorithm are discovered, the algorithm fulfilled its purpose over its lifetime. With the third generation (3G, UMTS) mobile networks, new encryption algorithms have been introduced (UEA1, UEA2) and they are backported to the GSM systems as A5/3

and A5/4. Contrary to A5/1, these algorithms are public. The algorithms UEA2 (A5/4) were specified precautionary for being able to replace UEA1 (A5/3) in case of a successful attack against the latter. Details of these algorithms can be found at the European Telecommunications Standard Institute [53].

UMTS introduced additional mechanisms for protecting the wireless link (e.g., see Pütz et al. [176]). It introduces an integrity key for protecting signaling information and sequence numbers for preventing replay attacks. Additionally, the cipher and integrity keys are changed on a regular basis, depending on time or the amount of already encrypted data.

A remaining problem is the introduction of new encryption algorithms. The devices have to support them as well as the mobile network itself. The network side requires hardware changes in the base stations, so that the actual introduction could take too long for A5/1 being secure. However, application-layer encryption can be used even without secure network-level algorithms, e.g., virtual private networks for data communication and speech encryption systems for voice communication as an additional layer in case of legal interception.

Breaking WLAN Encryption. Wireless local area networks (WLAN) have been a hot topic in real-world security since their beginnings. Either they have been used without any encryption at all, by using flawed encryption schemes, or with weak passwords. The following paragraphs shortly summarize the status of this part of mobile security. Technical information on the schemes and the attacks can be found in the literature [52].

In the beginning WLAN deployments, especially privately operated networks did not use encryption. The access points even offered a DHCP server to automatically supply the attacker with valid network settings for IP address, name server, and gateway.

Wireless equivalent privacy (WEP) was the first encryption protocol that has been widely adopted. Its weaknesses are a short encryption key that is the same for every connected client. Additionally, several reductions in search space have been found by cryptanalysts.

Today, WLANs are protected by Wi-Fi Protected Access (WPA) security protocols. WPA is based on the Temporal Key Integrity Protocol (TKIP) that was designed to allow the migration from WEP to WPA on already deployed hardware. The main security enhancements are dynamic encryption keys, sequence numbers for packets (for preventing replay attacks), and message integrity checks. For small networks, WPA has a “pre-shared key” mode that is based on a shared key.

The encryption might be vulnerable because of a short key length of the shared key. With limited input possibilities, e.g., only a number keypad, users of mobile devices might set short encryption keys that only contain numbers. This raises the probability that an attacker is successful with a brute force attack.

The successor of WPA is called WPA2. It replaces the RC4 cipher of WEP/WPA with the AES encryption standard. Moreover, it replaces TKIP (in Counter Mode) with Cipher Block Chaining Message Authentication Code Protocol (CCMP). For the purposes of this thesis, WPA/WPA2 are assumed to be sufficiently secure and attacks are considered to be brute-force attacks only.

Conclusion. Currently available security protocols for WLANs are considered secure, if used appropriately. As an additional security measure, even additional cryptographic measures can be taken on application layer.

3.3.1.2 Evil Twin Radio Access Nodes

The wireless connection of a device and a radio access node of a mobile network is always endangered by entities in the same physical area that can act as *evil twin*: they imitate the parameters of the legitimate communication partner. This attack vector is always possible when a communication partner does not need to authenticate itself. This sections shows the examples of GSM base stations and WLAN access points.

Evil Twin WLAN Access Points. More sophisticated WLAN clients remember wireless networks, to which they were connected. This is a convenience feature, but it may be a security weakness. An attacker would set up a WLAN access point with the same parameters as the legitimate access point (SSID, probably the active channels) in the same physical area as the legitimate access point. Whenever a mobile device is in the access point's coverage area and recognizes the access point, it might switch to use this (cheap) link for data connections. If the mobile device user uses applications that transfer data over unencrypted connections, the attacker is able to intercept the data.

There are several constraints that have to be met before this type of attack can take place. Encrypted wireless networks are not vulnerable to this type of attack, only if the attacker was able to figure out the encryption key. Moreover, corporate wireless networks often do only allow access via virtual private network connections, even though an association with the access point might be possible without restrictions and without encryption. As a consequence, the only type of wireless networks

susceptible to this type of attack are unencrypted wireless networks. The main use case for them is the provision of Internet services in cafés and some public places, where uncomplicated access to the wireless network is seen as a feature of usability.

For these public networks, recent work proposed a solution for avoiding an evil twin attack by using two additional colored indicator lights in the access point with a client application showing the simultaneously submitted flashing sequence of the lights [182]. The user could verify the sequence by comparing its local display with the physically visible access point. This system was tested for usability with intriguing simplifications compared to the original propositions.

Rogue WLAN Access Points. The security topic connected to this term does not contribute to the mobile device security in the context of this thesis, but it is inserted for clarification. A *rogue access point* is defined as a WLAN access point that is added to a cable-based computer network without the permission of the network operator. A typical example is an employee who wants to enable access to the corporate network that is not physically bound to his desk. More details can be found in the literature [172]. These rogue access points do not have to be confused with the concept of evil twin access points.

Evil Twin GSM Base Stations. It is possible in the GSM network to simulate a base station. When the requirement of physical proximity is fulfilled, the so-called "IMSI Catcher" is able to simulate the base station, to connect to the real network, and to forward the authentication requests to the attacked mobile device. This is a successful man-in-the-middle attack without the need to break the underlying encryption, because the mobile device is requested to switch off encryption, a feature of the GSM network. For more information on the topic see Fox [77]. They call this a man-in-the middle attack, because the IMSI Catcher exchanges data between the victim and the mobile network. For terminology uniformity with WLAN evil twins, this thesis calls this attack evil twin GSM base station. This is also called "false base station attack" in the literature [85, 176].

A second aspect of evil twin base stations is the possibility to commit a man-in-the-middle attack on the UMTS network. In GSM networks only the mobile device has to authenticate itself, and for increased security UMTS was designed to provide mutual authentication of mobile device and the network. Additionally, signaling information is integrity-protected as a means to prevent evil twin base stations [176].

However, UMTS was also designed to be compatible to GSM, whenever no sufficient UMTS coverage can be provided. This compatibility makes a roll-back

attack possible, where the compatibility mechanisms between these two mobile networking standards are exploited [130].

Conclusion. In summary, the evil twin attack vector can break the confidentiality of the mobile device user's data, most prominently access passwords to services like e-mail or Web accounts. As always, application layer encryption helps in these cases.

3.3.2 Backend Systems

This section adds an attack vector to mobile device security that is not obvious at first glance, but with the example of a year 2005 security incident it is shown how insecure backend systems can even compromise the privacy of a mobile device user.

3.3.2.1 Danger Hiptop / T-Mobile Sidekick

The Hiptop device (named "Sidekick" in the T-Mobile version) of United States based company Danger is a feature phone with a closed operating system (cf. Section 2.1). It differs from other mobile phones in storing its media data not only on the device itself, but mirroring the data in the MNO's network for Web accessibility. The data is protected by a password only. That means, it is possible to break a user's on-device data confidentiality by not attacking the mobile device at all.

The incident took place in the T-Mobile network of the United States of America in 2005 and led to the publication of phone numbers and private data of prominent United States citizens. It is reported by the Washington Post [117] to have been a combination of Web application attacks (cf. Section 6.2.1.2) and social engineering attacks (see Section 3.5).

The Web applications had a vulnerability that allowed to reset the access password to the mirrored data, resulting in locking the legitimate user out of its own account and giving the new password to the attacker. The only necessary piece of information for this attack was the mobile phone number.

The social engineering part of the attack was finding the mobile phone number of a prominent client of the MNO. This was achieved with a social engineering attack on an MNO's store, tricking the employees to reveal an access password for internal systems of the MNO. From this starting point it was possible to map names to phone numbers.

3.3.2.2 Other Backend Systems

Attacks on backend systems also comprise GPRS attacks [233] or attacks on the MMS infrastructure [89]. Moreover, the upcoming outsourcing of computation ("cloud computing") might lead to new privacy concerns and to new solutions for ensuring privacy. The solutions could possibly be transferred to solve the problems of backend systems with mobile devices.

3.3.2.3 Conclusion

The previous examples demonstrated the possibility of unexpected attack vectors, in this case to break the confidentiality of on-device data. The Sidekick incident might be non-repeatable because of the coincidental combination of vulnerabilities that led to the successful attack. Together with the fact that on-device data usually is not mirrored on network servers this attack vector will be uncommon and improbable in the future, except for users in corporate environments.

3.4 Software-Centric Attacks

Software-centric vulnerabilities are the most important class of vulnerabilities for mobile devices from the attack model of this thesis. Especially the rise of the—hardly security-specified—mobile Web browser led to various exploited vulnerabilities in the recent time.

3.4.1 General

3.4.1.1 SMS Vulnerabilities

An incident of the early times of mobile phones (not even smartphones at that time) was an implementation bug in the SMS parser of a Siemens S55 when receiving an SMS with Chinese characters, leading to a denial-of-service [29]. A denial-of-service also affected some Nokia phones when Nokia sent an invitation to the CeBIT computer fair with a picture SMS [124].

Both bugs required a local firmware update, forcing the users to bring or send in their device to customer service. This class is expected to be of less importance in the future, because modern smartphone architectures are increasingly allowing local or remote firmware updates.

A recent denial-of-service attack is the "curse of silence" short message, which was published at the end of 2008 [51]. It is caused by an omitted sanity check of input data: A rarely used function of the short message service is an e-mail to SMS gateway function. The standard defines a maximum length of 32 characters for the e-mail address. Some implementation of this function in Nokia phones assumes this length as verified by the mobile network and does not perform a sanity check on the device, but the mobile network did not limit the e-mail address length. This led to a failure of the messaging functionality of the affected Nokia phones, and they had to be reset to factory settings.[1] This is a clear violation of the principle to never trust (resp. assume anything of) an input value (see, e.g., Huseby [99]). A solution for this vulnerability would have been proper adherence to software development best practice.

The "curse of silence" exploit shows that both denial-of-service attacks and vulnerabilities in operating system applications are not only of historical interest, as the dates of the other listed exploits show, but support the hypothesis that there are always exploitable vulnerabilities in operating systems.

3.4.1.2 MMS Vulnerabilities

In the year 2006, a remote code execution exploit for mobile phones using MMS as the attack vector was published [147]. It exploited a buffer overflow in the MMS handling program of Windows Mobile. The targeted return address was known for a specific device except for the current memory slot (cf. Section 2.2.3), which had to be guessed, therefore lowering the success probability of the exploit.

As being the first of its kind, it supported the public fear of that time that mobile devices would start to become commonly attacked. The exploit received some attention by a technical audience and the mobile network operators who published patches for affected devices. Anti-virus companies added the exploit to their signature databases, but the exploit never appeared as part of mobile malware.

There are two possible explanations for this fact. The first one is the probability of succeeding with the message by using the correct memory slot. A second and more probable explanation is the actuality of the affected devices. Windows CE 4.2 was already succeeded by Windows CE 5 at the time when the exploit was published. Therefore, this vulnerability only affected comparatively dated devices.

[1] Nokia published a removal tool one month after the publication. This tool enabled to delete only the malicious messages, therefore avoiding the need to lose other data.

3.4.1.3 Application Framework Vulnerabilities

At the end of 2004, implementation vulnerabilities in the virtual machine implementation of some devices were presented [88]. It was possible to bypass the bytecode verifier and to access methods of the underlying native operating system directly. Unfortunately, the results have not been published in detail.

3.4.1.4 Sophisticated Applications

Different from the general software-centric attack vectors named so far are sophisticated applications like the e-mail program or the Web browser. They differ from SMS/MMS programs because of their extended functionality, leading to an increased attack surface. This is also true compared to application frameworks because these frameworks usually have a more sophisticated security architecture and the separation from the operating system is their main reason for existence.

The Web browser as a software of rapidly increasing importance in mobile devices has its own discussion in Section 3.4.3.

3.4.2 Operating System Modifications

Sometimes it is possible to evade security features by modifying the operating system itself. As real-world examples, this section covers firmware manipulation and malicious signing certificates.

Not all operating system modifications that are named here belong to the software-centric attack class. Especially the manipulation of the firmware image and the introduction of malicious certificates on the mobile device are attacks that are hardly to be committed by mobile malware. However, the possibility exists.

3.4.2.1 Manipulating the Firmware Image

As real-world example, a vulnerability in Symbian OS is described here. The Symbian OS Platform Security Architecture (PSA) has a central configuration file SWIPolicy (cf. Section 2.2.4). This file is set by the manufacturer or the mobile network operator and cannot be configured by the user, but changing this file is necessary for a user to gain extended control over his device.

It was possible in 2008 to manipulate a Nokia firmware image before installing it on the device. Nokia allowed users to perform a firmware update with their own computers. The user had to download a firmware image from Nokia, which was

installed on the phone afterwards. Some versions of the firmware image happened to contain the SWIPolicy in human-readable text [127]. Therefore, it was possible to modify the image file.

Two remarks on the real-world applicability of this topic are necessary: first, even though it was possible to modify the SWIPolicy, it was necessary to maintain the length of the firmware image to satisfy basic sanity checks by the firmware installer. This posed an additional complexity to the exploit, but it was solved. Second, this vulnerability was quickly fixed by Nokia, i.e., the vulnerability did not have any long-lasting impact.

This adds another attack vector to mobile device security. However, it is not considered in this thesis, because it is in its current incarnations no attack vector that is accessible by mobile malware. And Chapter 2 introduced that this firmware update process is well-secured, leading to additional effort for an attacker.

3.4.2.2 Mapping ROM to Writable Memory

The problem of mapping read-only memory to writable memory does not arise on common computers, because all files reside on the hard disk there. Mobile devices have an inherent advantage here, because all files residing in read-only memory cannot be corrupted by mobile malware. Unfortunately, there are exceptions.

Symbian OS allows to override files in ROM when a file with the same name resides on the memory card (cf. Section 2.2.4) and Windows Mobile allows to change a central pointer to lead the operating system into using files from writable memory. This effectively leads to the same implications as the manipulation of the firmware image from above.

3.4.2.3 Runtime Memory Manipulation

Similar to the static process of mapping system files during installation from above, manipulating the memory of a process at runtime allows to get advanced control over the process.

There is an example for Symbian OS where a (signed) runtime debugger on the device was allowed to read and write the memory of other processes [212]. If the signature gives sufficient rights to the signed program, even reading and writing of system process memory is possible.

3.4.2.4 Using Malicious Certificates

Whenever a signed application is installed, the signature is verified against a list of valid certificates (cf. Section 2.2.4). That means, there are two ways to illegally get a valid signature: computing the signature without a valid certificate and adding an own certificate to the list. The first option is equivalent to breaking the underlying cryptography scheme and is considered unfeasible here. The second option is of more interest and has a real-world example.

An anonymous author posted a Symbian signing certificate to a Web forum [6]. This certificate enabled a user to sign his own applications, after he was able to add the certificate to the list of valid certificates. Valid certificates are recognized in the directory `c:\resource\swicertstore\dat`. Usually, this directory is only readable, but getting write permissions was possible with a runtime exploit (described later in this chapter). Afterwards, using the signing certificate was possible.

3.4.3 Web Browser

3.4.3.1 Evolution of the Mobile Web Browser

The mobile Web browser is an emerging attack vector in mobile devices. Just as common Web browsers, mobile Web browsers are being extended from pure Web browsing software to complete application frameworks with widgets or completely browser-based mobile devices [157]. It can be expected that even security-relevant functions of the operating system are accessible in the near future.

Industry requirements documents even include these security-relevant features. An example is the browser requirements document of OMTP [162]. Requirement BR-2540 demands: "The browser MUST support the making of voice calls and video calls from a URI / IRI".

Requirement BR-2570 suggests appropriate security mechanisms in the implementation of this requirement: "The browser SHOULD ask for user confirmation before initiating any call from a hyperlink". Even though, a possible complying implementation is the vulnerable iPhone Web browser, which enables browser-based dialers to create costs for the user without necessary confirmation (see below).

Therefore, the mobile Web browser as an application framework of its own is able to undermine the mobile device's security model: the original—and possibly secure—model of signed applications is replaced by the security model of the Web browser developer. This is visualized in Figure 3.3. Examples for successful

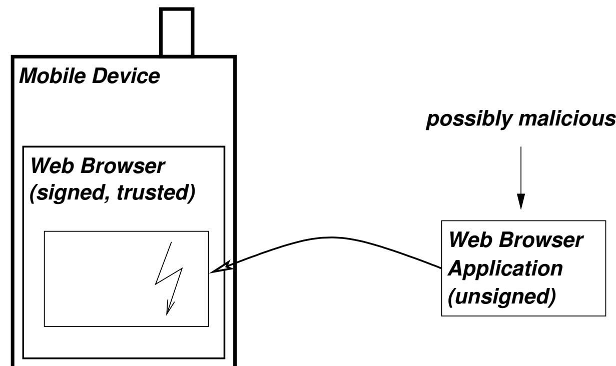


Figure 3.3: Web Browser Undermining the Security Model

attacks—besides denial-of-service attacks on the mobile Internet Explorer [41]—are the jailbreak of the iPhone, hacking the Android browser, and using the iPhone browser as a dialer.

After having read the subsequent details of mobile Web browser exploits, one might ask the question, how to solve the problem of an increasing number of security mechanisms. These questions touch the next class of mobile device attack vectors, the mobile device user (see Section 3.5). A possible solution will be discussed in more detail in Section 6.2, where a technology-independent security solution solution is proposed.

3.4.3.2 iPhone Jailbreak

Unlocking the iPhone with firmware 1.1.1 was completely based on vulnerabilities of the Web browser. Therefore, the exploit of this vulnerability is documented here as an example of how important the attack vector Web browser is for the security of the mobile device. A documentation with more technical details can be found in the literature [48, 140]. The iPhone firmware version 1.1.1 was supposed to re-enforce the jailbreak of previous firmware versions, but the developers used an old library in the iPhone firmware.

The affected library is named libtiff that is used for displaying TIFF images. It had a stack-based buffer overflow vulnerability [40] that was fixed in version 3.8.2 of March 2006. Firmware 1.1.1 was published in September 2007, using a still vulnerable version of libtiff. Ten days later, a browser-based jailbreak possibility was published. The authors already used this vulnerability to exploit the Sony Playstation Portable, so it was only necessary to port the exploit code. Now it was possible to remove the third-party software restriction by visiting a specially prepared Web site.

3.4.3.3 Android Vulnerability

A vulnerability in the Android Web browser was discovered in October 2008. Like the iPhone vulnerability above, it was due to a deprecated and vulnerable library module. An important difference to the iPhone vulnerability was the sandboxing architecture of Android that restricted the effects of this vulnerability to the Web browser process. Despite the restriction to a single process: being able to control the Web browser process is a rewarding target.

3.4.3.4 iPhone Web Browser Dialer

A recent vulnerability [146] of the iPhone Web browser allows a Web page to initiate a phone call without user interaction. The vulnerability is based on the interaction of different applications.

URIs with beginning with "tel:" allow Web pages to initiate phone calls. As a security mechanism, a confirmation dialog is shown. Unfortunately, when the confirmation dialog is active and when the Web page requests a second application to launch (e.g., with a "sms:" URI), the Web browser closes the active dialog with the option "call" instead of "cancel". For increasing the chances of succeeding with the phone call, it is even possible to freeze the GUI for some seconds with a long number after the "sms:" prefix.

This vulnerability is an example of the combination principle of security that even when two systems are secure, the combination of these systems is not as secure as the systems themselves.

3.5 "Layer 8": The User as Attack Vector

The user is sometimes seen as an additional layer on top of the application layer. When considering the ISO OSI network reference model [238] with its seven layers from the physical layer 1 to the application layer 7, it is natural to assign the term *layer 8* to the user. Colloquially, a "layer 8 problem" consists of a working technical system with the erroneous behavior of the system's user. Scientific discussion also assigns the term "semantic layer" to the user [111]. This point of view that the users of technical systems are sometimes the only problem of malfunction is especially important in the security field. Schneier introduces the security of a system as being defined by the weakest link, in many cases being represented by the system's user [193]. Gollmann [85] even calls this the

“fundamental dilemma of computer security” based on the number of users dealing with security topics having increased from a few organizations to every user of the Internet: “*Security-unaware* users have specific security requirements but usually no security expertise.”

Many studies have been performed to evaluate the security knowledge of the common user. Most of them show what already the well-known study of Whitten and Tygar [230] found out: the common user is not able to use security mechanisms in a correct way. Different attempts have been made to simplify the interface for security settings, but even the very simple Windows security slider with only four possible positions was not completely understood and therefore set wrong by users who rated themselves as “IT proficient” [82].

Therefore, the question arises what these numerous security mechanisms are useful for, when the user does not understand them. Even if he understands to work with one mechanism, another mechanism might be incompatible with his understanding. People working in the security field always want to achieve the desirable goal of more security-conscious users, but the will, the interest, and the time to learn more than one security mechanism is likely to be limited for the common user.

This section shows in detail important topics of the user’s role in security. Section 3.5.1 starts with the security awareness that is hardly available. Section 3.5.2 shows the contrast, how much influence the user has on security-critical decisions. Section 3.5.3 presents the “user as attack vector” as the topic of social engineering. Finally, Section 3.5.4 introduces to the research on *security and usability* that came into life in the recent years for relieving the situations presented in this section thus far.

3.5.1 Security Awareness

When it comes to the question who is responsible for the security of mobile devices, there are contradicting results in the literature. Furnell and Clarke found out in 2005 that “customers are indeed security conscious, but perceive the issue to be handset-dependent rather than operator-related” [35]. A study by the OMTP group of MNOs in 2006 showed that “participants stated that their mobile operator was primarily responsible for mobile security” [161]. In 2008, F-Secure found a contrary result that “over half of those questioned felt it was up to the individual user to ensure their phone was protected” and “a third expected this to be taken care of by their mobile phone carrier, with the US putting the greater emphasis on third-party responsibility. Only 11 per cent of Germans believed their mobile phone provider should be in charge of security, compared with over 32 per cent in France.“ [71]

These results—of course—do not mean that the MNO should leave security to its clients. They can be explained with the fact that there are currently no widely used operator-based security mechanisms. Therefore, the participants might not have been able to imagine how these security mechanisms would look like. If the MNOs start to proactively provide security mechanisms, the results could change towards the MNO seen as a good entity for introducing security mechanisms.

An approach to increasing security awareness is education. There are documented attempts for the mobile world where the network operators provide educational material for their clients [215]. However, their impact is limited and the material contains notions that are more difficult to understand than the Windows security slider from above. Additionally, other work vigorously proposes that users do not need to become security experts at all, because it is the task of the security experts to protect the common user. Additionally, educating users would not raise their security awareness [92].

Denning [45] and Anderson [5] give examples of poor password use, documenting low security awareness, because the relationship between guessable passwords, successful attacks, and the role of the user is often not clear to a particular user. Users compromise security by rating convenience in a particular situation higher than an abstract security threat. This leads to short passwords, and especially on mobile devices to passwords only consisting of numbers, as they are most convenient to enter on the mobile device keyboard.

Additionally, it is assumed here that the appreciation of the mobile device is lower than for desktop PCs, and that it is more seen as a disposable item [161]. The security awareness with regard to their mobile devices can be assumed lower than for PCs as well. Therefore, the cost of security solutions must be lower than for PCs, the solutions should especially work automatically and should not need much operator maintenance like adding new signatures to anti-virus databases.

In summary: this thesis assumes that the average user does not have extensive security awareness. Even if his security awareness should be raised by media coverage of security incidents like worm or virus outbreaks, it seems questionable if he is able to differentiate between different classes of security products to use them correctly. An additional assumption is: even if the user is security-aware, his will to get into the depths of security research and products will be close to zero.

3.5.2 Influence of the User

Opposite to the low security awareness of the common user is his influence over the mobile device: “the human factor can negate all technical mechanisms taken to

secure systems” [197]. Chapter 4 will show that there are only few cases where harm to the user happens without any interaction of the mobile phone (resp. the malware) with the user. So in most of the cases, the user is able to prevent harm to himself, but he must be able to understand the security solutions of his mobile phone.

It might be the case that the implications of security mechanisms like application frameworks (e.g., the Java framework J2ME) and signature schemes for different trust levels might not be understood by the average user. An example is a phone that was locked for third-party software. This was a good measure from the security point of view, but this measure was not accepted by its users for usability reasons, forcing the provider Orange to unlock the devices, thus decreasing the security of the unlocked devices [167]. Recently, this story was repeated with the iPhone. These are indications to expect that devices will be open to extensions, and therefore open to malware, in the future. This leads to the definition of two problems of usable security on mobile devices.

Permission Flat Rate Problem: Users are not able to see the full extent of security decisions with persistent effects. Current security mechanisms (e.g., on the J2ME application framework) sometimes give the possibility to the user to permit a security-relevant action conveniently not only for the current instance, but for all future instances. This is a problem because of the direct correlation between security-naive users and decisions that are not remembered afterwards.

Security Acceptance Problem: Whenever users are restricted too much by security, they will open the doors and bypass the security mechanisms.

These two problems point into the direction of security services that are embedded into the system (also discussed by Kuper and Gannon [118]): security mechanisms should be invisible and unchangeable by the user. This view shows that most of the current security mechanisms (cf. Chapter 2) probably have dealt with the wrong attack vectors, where the user should be seen as the most influential attack vector.

3.5.3 Social Engineering

Social engineering is the act of leading a user to override technical security mechanisms. A classical essay on this topic is the book “The Art of Deception” [135] and Anderson discusses the topic in his book “Security Engineering” [5]. Here, the two dimensions *abuse of trust relationships* and *imitation of functionality* are shortly shown for the domain of mobile devices.

Social engineering becomes most important when there are no more technical vulnerabilities to exploit. So in general, it is a good cause that security is only dependent on the user. It means that technical security mechanisms are effective and sufficient.

3.5.3.1 Abuse of Trust Relationships

Trust relationships exist between known persons and these relationships are commonly documented in the address book of the mobile device. In unsecured systems, mobile malware can access the address book and spread by sending itself to the contacts. The Commwarrior MMS worm is an example of this technique.

An important additional condition for this type of attack to suffice are the communication expectations of the targeted user. If the language is different than the common communication language, the targeted user might become suspicious. Even if the language is correct, there are additional aspects of the message that might raise the targeted user's suspicion. Only if everything fits, he probably takes the supposed actions like installing the delivered software, thereby infecting his own device.

3.5.3.2 Imitation of Functionality

There are cases where even a security-aware user cannot distinguish a social engineering attack from legitimate functionality. This is especially true for incoming messages, mainly via the short-range Bluetooth technology.

Not quite common in Europe but widespread in Asia is marketing via Bluetooth. Shops can have a Bluetooth sending entity, sending messages via the Bluetooth protocol to persons who are passing. Sometimes, the message might indicate that the prospected client should install a game for gaining access to vouchers. Whenever such a system is active, an attacker can set up a similar system that sends imitated legitimate messages, but with malicious content. A more detailed introduction can be found in the literature [180].

A similar case are large events where participants are open to receive additional information. Some successful outbreaks of the Cabir worm are documented in Europe during sports events and concerts [13].

3.5.4 Security & Usability

This section introduces the research field "security & usability" and proposes an easy-to-use interface to security afterwards.

3.5.4.1 The Research Field

Security & usability is a research field that came out of research on human-computer interaction (HCI). It started with the famous study by Whitten and Tygar in 1999 [230] and gained some attention in the following years [39, 83]. User studies regularly find out that security mechanisms are not understood or used correctly by the majority of their users [230, 80, 81, 82]. Moreover, some authors propose to embed security in products [118] and in the development process [92] rather than having it stand-alone. Usability heuristics have been developed by Nielsen [154] and Shneiderman and Plaisant [198]. They are a good starting point for usability of security solutions.

3.5.4.2 Interface Proposal

The following interface proposal is based on the results of user studies [161], especially that participants were more willing to take responsibility for their own security when given the means that would enable them to do this, e.g., security settings. Additionally, the participants suggested that their phone should be equipped with rudimentary security aids, similar to an antivirus, or to have an “overarching” security setting that they could configure if required.

The proposed interface uses the idea of a *personal security profile* [19], where the user only has to answer a few simple questions to define his profile. These questions should be self-explanatory and understandable, so that the user can set the answers according to his protection goals. This establishes a direct means for the user to communicate his security needs.

This simple interface complies with two of Nielsen’s principles for user interface design [154]: “Match between system and the real world: The system should speak the users’ language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. [...]” And: “User control and freedom”.

Possible questions are shown in Table 3.1. This interface is very simple and the questions are completely focused on the user’s protection goals. Therefore, the system will provide a baseline of protection in every possible situation. It might even continue to be useful when sophisticated policy systems fail because of complexity and unpredicted security conditions, both of them being a major danger when the transition is made from example policies to real-world application.

The first setting of Table 3.1 is a virus scan within the network of all files that are downloaded to the device. This setting may reduce the subjective feeling of privacy for the user, but it has the benefit of increased security.

Table 3.1: Personal Security Profile

Option	*Settings*
Allow anti-virus scan by operator	yes/no
Limit data traffic	yes/no/# per time unit
Limit number of allowed messages	yes/no/# per time unit
Restrict sending of messages to messaging application	yes/no
Restrict access to personal data	yes/no

The second and third row allow to specify upper limits for data traffic and the number of messages. When the limit is reached, the user will be informed and every new event of the corresponding type has to be confirmed explicitly. This leads to the definition of a third problem of usable security on mobile devices.

Paradox of Security mechanisms: Even when the restrictions "number of events per time unit" are commonly used, it can be expected that attacks try to create less malicious events per time unit for not triggering the security restrictions. If they are successful with this strategy, the only remaining means for detecting monetary damage is the monthly invoice. Or the user must monitor his sensitive assets more often than once a month. A possible solution could be that the MNO sends an SMS with this information on a weekly basis.

The fourth row defines that only the messaging application is allowed to send messages without user confirmation. Whenever an application, be it a J2ME MIDlet or a Web browser widget, tries to access this central function for creating costs, the user will be warned before the costs are created. The problems of implementing such a solution are vulnerabilities in the implementation. Even if sending messages is restricted, an exploitable messaging application undermines this security setting. This is the same problem as for signed applications or operating system modules.

The last row considers the problem of preventing illegitimate information flow. This is a difficult problem that is currently being attacked [152]. Of course, this problem cannot be solved with this simple setting from a theoretical point of view, because it is easily possible that a legitimate program accesses the data and forwards it to an unsecured location where a malicious program accesses the data. However, this solution should work for the majority of mobile malicious software.

This interface proposal leaves room for extensions. Advanced users could be supplied with additional log entries, for example, a trace of sent messages, regardless whether the messaging application sent them or another program by using a system call.

As a conclusion, whenever mobile device security incidents start to increase, it is not the minority of security-aware users that will define the view on these incidents, but the majority of the common users. It is more useful then to have a simple solution in place that will prevent or remedy the majority of possible damage instead of having several sophisticated security mechanisms in place that are not understood and therefore not used. The design of a policy enforcer for the proposed interface will be presented in Section 6.2.

3.6 Conclusion

This chapter contributed to the question how to shape the research topic mobile device security by presenting incidents of mobile device security of the recent years. It showed that the increasing functionality of mobile devices (leading to smartphones) brings more possible attack vectors to them. It presented in detail the operational side of mobile device security, that is, vulnerabilities exploitable by mobile malware.

The chapter contributed an attack model with four attack classes (hardware-centric, device-independent, software-centric, user layer), two of them exploitable by mobile malware: software-centric attacks and user layer attacks.

Of special importance is the user of mobile devices. The average user does not have extensive knowledge of security. Even if his security-awareness should be increased by media coverage of security incidents like worm or virus outbreaks, it seems questionable if he is able to differentiate between different classes of security products to use them correctly. An additional proposition is that even if the user is security-aware, his will to get into the depths of security research and products might be close to zero. Therefore, it can reasonably be assumed that the common user of mobile devices will never become perfectly security-aware and never become able to use sophisticated security mechanisms. Especially techniques of social engineering can be expected to be successfully applicable for an indefinite amount of time. This is an inadequate situation, because security issues threaten every user of mobile devices and not just the few security-proficient users.

Because of these facts, it is proposed that most of the users need a solution that is embedded into the normal handling of the used device rather than a separate solution. Together with the specifics of mobile device security in Section 2.1.3, this chapter is the basis for the technical solutions of this thesis beginning in Chapter 5.

Chapter 4

Structuring Mobile Malicious Software

This thesis on smartphone security sees mobile malicious software (malware) as the main attacker model. The previous chapter presented the attack vectors that mobile malware is able use. This chapter surveys the current state of real-world examples of mobile malware. It will show that mobile malware is not a topic of the same importance as it is in the common computer world today. Even though, malware for mobile devices has been a topic of increasing importance since the middle of 2004, when the worm Cabir for Symbian OS and the virus Dust for Windows Mobile appeared.

Since their first appearance, mobile malicious software has been a topic of scientific investigation (Section 2.3.2) and of mainstream media coverage [226]. Section 4.1 presents notable examples of mobile malware together with their main functionality, with the goal of showing the status of what has to be defended against today.

This chapter contributes a coherent presentation of mobile malware for the time of this writing in August 2009. It extends related presentations because the description are presented with regard to portability of the malware. It shows that even if most of today's mobile malware targets Symbian OS, most of these pieces of malware are portable to other mobile operating systems.

General properties of mobile malware are presented in the two following sections. The behavior of a malware sample is categorized into the three phases *infection*, *malicious functionality*, and *spreading* in Section 4.2. Finally, Section 4.3 discusses the portability of today's malware.

4.1 Known Mobile Malware

Figure 2.5 shows the absolute numbers of mobile malware samples for different operating systems over the years. It ends in 2006, but the trend is visible: almost all malware targets the Symbian operating system and only few malware targets the J2ME application framework or Windows Mobile ("Pocket PC"). Complete and current lists of mobile malware names including descriptions are published by the anti-virus companies (e.g., F-Secure, Kaspersky [221]) and by other authors in the surveys of Section 2.3.2. Therefore, this section does not want to cover all of today's mobile malware. Instead, this section highlights aspects of the presented mobile malware that are important in the context of this thesis. Table 4.1 summarizes the following enumeration of mobile malware and is ordered by the year of its first appearance.

Section 4.1.1 lists mobile malware for Windows Mobile, equivalently Section 4.1.2 for Symbian OS. Section 4.1.3 presents mobile malware for the application framework J2ME. These samples can run on any mobile operating system that implements a J2ME runtime environment.

4.1.1 Windows Type of Operating Systems

This section contains descriptions of malware for the Windows type of operating systems. Besides mobile malware targeting Windows Mobile it includes some malware running on the desktop variants of the Windows operating system. The connection of this cross-platform malware will be analyzed in more detail in Section 4.3. Section 5.3.3 will give a detailed dynamic analysis of the Dust and Pmcryptic malware samples with our dynamic software analysis tool MobileSandbox, which will reveal a more complete view of their functionality.

Dust. Dust appeared in the year 2004 and is the first virus for Windows Mobile. It is a proof-of-concept file infector virus, which asks the user for permission to perform its malicious functions. If allowed to do so, it will infect all executable files in the root directory of the mobile device. It was subject to a detailed static analysis shortly after its appearance [173].

Brador. The malware Brador was found shortly after Dust. It is a backdoor program, which "copies itself to the startup folder, mails the IP address of the PDA to the backdoor author and starts listening commands on a TCP port" [56]. A detailed analysis can be found in the literature [174].

Table 4.1: Known Mobile Malware (ordered by year)

Name	Type	Year	Operating System
Dust	virus	2004	Windows Mobile
Brador	backdoor	2004	Windows Mobile
Cabir	worm	2004	Symbian OS
Mosquitos	trojan	2004	Symbian OS
Skulls	trojan	2004	Symbian OS
MetalGear	trojan	2004	Symbian OS
Lasco	virus	2005	Symbian OS
Locknut	trojan	2005	Symbian OS
Feakk	worm	2005	Symbian OS
Commwarrior	worm	2005	Symbian OS
Cardblock	virus	2005	Symbian OS
CardTrap	virus	2005	Symbian OS
Blankfont	trojan	2005	Symbian OS
Crossover	virus	2006	Dotnet
Letum	worm	2006	Dotnet
Fontal	trojan	2006	Symbian OS
Mobler	worm	2006	Symbian OS
Redbrowser	trojan	2006	J2ME
Wesber	trojan	2006	J2ME
FlexiSpy	spyware	2006	“multi OS” (see text)
Acallno	spyware	2006	Symbian OS
Beselo	worm	2007	Symbian OS
InfoJack	trojan	2008	Dotnet
Pmcryptic	worm	2008	Windows Mobile
Yxe	worm	2009	Symbian OS

Crossover. The Crossover virus is an example of a malware that is able to cross the borders of a platform. It achieves this by using the intermediate bytecode language of the Dotnet application framework (cf. Section 2.2.2.6). It is able to run on common Windows systems and on Windows Mobile systems. More information on cross-platform malware will be given in Section 4.3.

Crossover has different operation modes for the different platforms, the functionality is chosen by using a runtime query API of Dotnet. In case of Windows Mobile, the process waits for an ActiveSync connection and in case of an established connection, copies itself on the connected computer and inserts itself as auto start program. In case of common Windows it is vice versa. A more detailed analysis can be found in the literature [171].

Letum. Another example for Dotnet is the worm Letum. It was discovered in 2006 and spreads via e-mail and usenet technology [206]. This worm is written for the Dotnet runtime environment, and therefore it can be executed on mobile devices running Windows Mobile. It is unclear, whether the worm is able to execute as intended on mobile devices, but because of the differences between the Dotnet framework and the Dotnet Compact Framework, it can be assumed that the malicious functionality of the worm is restricted to common Windows systems. As an example, the worm inserts itself in the auto start part of the registry with a path containing the ‘‘`C:`’’ device. This alone renders the worm useless in an unmodified form on Windows Mobile devices.

InfoJack (Infomeiti). The InfoJack malware was discovered as the fourth Windows Mobile malware in February 2008 [70]. It is a trojan that comes in two parts. The first part spreads on the devices by adding itself to installation packages of other software. In case of an available Internet connection, the malware will connect itself to a home server and download the second part. It is a noteworthy feature of InfoJack that it disables security settings in the registry of the Windows Mobile device, enabling software to be installed without security warnings or restrictions, a special vulnerability of the Windows Mobile operating system, which we will use in our own proof-of-concept worm in Section 6.1. Therefore, InfoJack is not portable to other mobile operating systems. The InfoJack malware is described in more detail in the literature [48, Chapter 10].

Pmcryptic. The Pmcryptic worm was discovered in November 2008 [207]. It spreads via memory cards and its malicious functionality is dialing a premium-rate phone number. Its infection functionality comprises hiding directories, adding files, and modifying the registry. Dialing phone numbers and spreading on memory

cards is portable behavior. However, as it is unlikely that other operating systems allow such a program to modify settings of the system in the registry, this malware is seen as not portable.

4.1.2 Symbian OS

Cabir. The first version of the Cabir malware family was released in June 2004 as the first malware for Symbian OS, written as a proof-of-concept worm [205]. It was developed as a Trojan horse within a Symbian OS security package. It spreads over the Bluetooth wireless connection. Many variants have been developed (F-Secure counts more than 30 variants [48]), because the source code was published in December 2004. The original Cabir sample adds itself as an auto start program, therefore it is started during every device boot process. Then it activates Bluetooth and starts scanning for another device in proximity. When a device was found, the malware sends its executable file over Bluetooth. No vulnerability is exploited, the targeted victim has to confirm that he is willing to receive the file. The original sample only sends its executable file out once for every boot process of the infected device, some of the succeeding variants implement more aggressive spreading mechanisms, more targeted to causing real damage.

One of the noteworthy variants of Cabir is the malware Mabir. It keeps the Cabir functionality of spreading via Bluetooth, but adds spreading via MMS by listening for incoming messages (SMS, MMS). Whenever the device receives a message, Mabir replies with an infected MMS. This procedure exploits the increased acceptance ratios when a user expects the owner of the infected device to send a message (cf. Section 3.5).

Mosquitos. The Mosquitos trojan (found in August 2004) was embedded into a game of the same name [134]. It sent out SMS messages to a premium-rate number, creating costs at the infected device. The creator of this trojan functionality turned out to be the developing company itself with the goal to embed a copy protection into the game. The copy protection did not work correctly in all cases and affected legitimate owners of the game. Therefore, this piece of software fulfilled the classification criteria for malware. The developing company removed the copy protection from later versions of the game.

Skulls. The Skulls trojan was found in November 2004 [58]. It is a special malware in two regards. First, it is bound to the Symbian operating system, because it uses a special vulnerability. The second point is the vulnerability itself:

in Symbian OS it is possible to overwrite system files, and overwritten system files influence the stability of the system. Even though the system files are in read-only memory, Symbian OS will disregard these files, if it finds a file with the same name on the (writable) device `C:\` (cf. Section 2.2.4).

The Skulls trojan exploited this vulnerability to overwrite the menu icon of every installed application to a picture of skulls and crossbones. This action sufficed to render the system applications useless, the only remaining functionality was using the device for phone calls. Its malicious functionality, compared to PC viruses, was summarized as: "In terms of damage caused and technical sophistication, viruses from this class are analogous to DOS file viruses which executed the command 'format c:\'" [197]. This demonstrates the early state of mobile malware (as of 2004, but valid for the year 2009 as well): malware does not aim at generating money for criminal subjects like the majority of common computer malware today, but is more proof-of-concept, even in terms of malicious functionality.

MetalGear (MGDropper). This trojan from December 2004 uses the same Symbian OS vulnerability as Skulls to disable applications, especially targeting virus scanners [57]. Additionally, it installs the Cabir worm on the mobile device. Because of using the same Symbian OS specific vulnerability as Skulls, MetalGear is not easily portable to other operating systems either (see Section 4.3).

Lasco. Lasco is a file infector virus. It searches all installation packages on the mobile device and adds itself as a part of the installation. Whenever an installation package was transferred to another Symbian OS device and installed afterwards, the virus succeeds in its spreading functionality. To this regard, Lasco is the Symbian OS equivalent to InfoJack on Windows Mobile, but without the registry change, which is specific for Windows Mobile.

Locknut (Gavno). This trojan was discovered in February 2005 [61]. It uses another vulnerability of the Symbian operating system: it creates entries for a new application in the application directory. The files only contain text instead of application information. Despite their invalid format, Symbian OS tried to execute the files when advised to do so, leading to a system freeze [197]. Because of the file's position within the directory structure (auto start folder), the device tried to execute it during every boot process. This led to a denial-of-service of the complete device, because it was no longer able to boot. Like MetalGear, Locknut also dropped a version of the Cabir worm.

Feakk. Feakk is a proof-of-concept malware that has been developed as a research project. It sends SMS messages to all contacts in the address book, if it finds an address book entry named “HACKME”. The most interesting thing about it is the timeline of events. It was developed and presented in March 2005 [93], and its source code was released in October 2006 [227]. However, only in April 2007 anti-virus company F-Secure added this malware to its mobile malware list [68]. It is unclear, with which date the Feakk malware found its way into the “malware curve” (cf. Figure 2.5). In case of the last date (March 2007) this curve would be something like a self-fulfilling prophecy: when more work is put into finding new malware samples, especially when also adding proof-of-concept malware to the lists, then a rising number of known malware samples is a natural phenomenon.

Commwarrior. The Commwarrior worm appeared in March 2005 [55]. It uses the two spreading vectors Bluetooth and MMS. Spreading via Bluetooth was already common at that time, but it was the first piece of malware spreading via MMS. Once installed, the malware permanently scans for other devices in Bluetooth range, but leaving the device’s Bluetooth indicator switched off [155]. The malware uses social engineering by using only the device’s address book for choosing its spreading targets.

Commwarrior only replicates itself and has no malicious functionality. Because of its spreading vector, the activity of this worm could be measured within the mobile networks. Its main activity was between the years 2005 and 2006 [195].

Cardblock. The Cardblock malware was discovered in 2005 [59]. Besides deleting several files, it encrypts the storage card of the device with a random password, leading to a loss of data. This type of malware can be extended to a business model for malware authors by setting a deterministic recovery password and selling the password to the victims. This development would be equivalent to common computer malware (e.g., the Gpcode malware [112]). Regarding portability, this malware uses standard functionality to access the storage card. Therefore, it is portable to other operating systems.

CardTrap. The CardTrap malware of September 2005 is an example of malware for different target operating systems [60]. The Symbian OS part of the malware copies a version of a Windows worm (Wukill) on the card, which will be executed by the auto start feature of Windows operating systems for removable storage cards. The mobile device part of the malware is portable, as only functionality for accessing the storage card is used. The auto start functionality of the dropped

Windows malware is not portable in any case. However, this is not part of the portability thoughts.

Fontal, Blankfont. These trojan pieces of malware cause a denial-of-service: for Fontal, the device will no longer start when it is rebooted [63]. For Blankfont, the device will start, but it will not display any font [67]. Both effects are due to a replacement of system font files with invalid files: font files, but from a different language version of the operating system. These two pieces of malware exploit the same vulnerability of Symbian OS as Locknut and MetalGear: the ability to replace system files in ROM by user-given files of arbitrary content.

Mobler. Mobler is a cross-platform malware targeting Symbian OS and the PC variants of the Windows operating system. As the formats of executable files of these two systems differ, it uses a dropping mechanism (see Section 4.3) for crossing the system borders. The Symbian OS executable has two main functions. First, it tries to copy files on the removable storage card. These files target the auto start mechanism of the Windows operating system for removable storage cards [216]. Second, it overwrites a number of files for disabling their corresponding applications, e.g., virus scanners. Besides the malicious behavior, the Windows part drops a Symbian OS installation package into different directories of the Windows system, most notably into removable drives [217].

When considering portability, the dropping mechanism is portable. However, it is difficult to say whether all other operating systems allow the overwriting of application files and whether they support an auto start mechanism. Therefore, Mobler is seen as not portable.

Acallno (GSM surveillance phone). Acallno is an example of the *spyware* kind of malware that aim at gathering information about the mobile device user, contrary to creating revenue for the attacker or damaging the user's data on the device. Acallno is a commercial software that is classified by the anti-virus companies as malware, for the first time in 2006. It sends copies of incoming and outgoing SMS messages. Acallno has two interesting properties: it hides itself from the user and it is bound to only one mobile phone per sample, identified by the IMEI [64]. Associated with this piece of malware name are a number of commercial mobile phone spying programs for listening to phone calls, sending location information, and informing about phone restarts or changes of the MNO smartcard [203].

The most interesting function is the SMS forwarding functionality, because it can lead to financial loss besides being spied. If the user uses mobile banking

with transaction numbers (TANs) or other mobile payment schemes sent out via SMS/MMS, an attacker might be able to get the message and use the TAN before the legitimate user does so. This way, an attack vector leading to financial loss is enabled, which is different from the common ways to lose money with the mobile phone.

Considering portability, the message forwarding functionality is likely to be available on all operating systems. However, the hiding functionality is likely to be not portable. This assumption is supported by the fact that the spyware is only offered for some Symbian OS devices.

FlexiSpy. FlexiSpy is spyware and a commercial service like Acallno. It started as a Symbian OS program. Currently, it is also available with different properties for Windows Mobile, Blackberry, and the iPhone [220]. Anti-virus companies categorized it as mobile malware shortly before Acallno [62]. It was chosen as an investigation subject for a detailed static analysis in the literature [48, Chapter 10].

Most interesting about this software is its signature from the Symbian Signed program. Even though it was classified as malicious software, the installation package received a signature of the Symbian Signed program, enabling messaging and phone functions without user acknowledgment [228].

Beselo. The Beselo worm appeared in December 2007, and as a malware family it is said to be “very similar to the Commwarrior family but contains enough differences in the code base and behavior that it is counted as separate family” [69]. Beselo does not have malicious functionality besides its cost-creating spreading functionality via MMS, but it has two noteworthy attributes. First, it is a new piece of malware to appear after some months without the discovery of a new mobile malware. Media attention of the mobile malware threat had decreased at this time, having its peak in the year 2006 [14].

Second, it uses another functionality of the Symbian operating system. Its executable does not have the extension of a Symbian OS installer package (SIS), but comes with media file extensions (MP3, JPG). Together with appealing file names, it tries to convince the user to open the file. It used a functionality of the Symbian OS media file handler, which recognized the file as an installer package despite its extension. Therefore, the user will see the common installation prompts (cf. Section 2.2.4). Inexperienced users could fail to notice that media files usually do not need to be installed.

Yxe. A recent addition of the year 2009 is the worm Yxe [72]. Its new contribution to the mobile malware world is its spreading vector short message service (SMS). The message content consists of an appealing message and of the URL of a helper program that uses the address book to send the helper program's address to all contacts in the address book. This malware sample uses trust between message sender and receiver together with an appealing message. Thus, it is another example of malware using social engineering to trick the users into overriding all technical security mechanisms. The malware uses standard techniques like accessing the contact list and sending text messages. Therefore, the malware can be ported to other operating systems.

As an additional aspect, the program has a valid signature of the Symbian Signed program. That means, the operating system will not ask the user for confirmation before the malware sends its spreading messages. The signing certificate has been revoked, so this malware is a reason to start focusing the currently not used certificate revocation processes (OCSP, cf. Section 2.2.4). In July 2009, a new variant (named Yxe.D) was found that used another vendor name for the signature, thus increasing the number of Symbian Signed approved malware vendors [72].

4.1.3 Java Platform, Micro Edition (J2ME)

A J2ME malware sample has the advantage of being able to run on almost any mobile device at the time of this writing. It is only restricted, if it exploits implementation vulnerabilities of a particular virtual machine. As shown in Section 2.2.2, J2ME offers most of the necessary functionality that today's malware for other operating systems uses: messaging functions, address book access, networking functions.

No J2ME malware had existed until the year 2006 (cf. Figure 2.5). The two samples Redbrowser and Wesber are described shortly here. In 2008, Kaspersky reported an increase in Trojan horses for J2ME counting fifty different samples with almost the same functionality: sending short messages to premium numbers [87]. Despite the total number of J2ME malware samples increasing, there are only few malware families for J2ME.

Redbrowser. The Redbrowser trojan appeared in March 2006. It promises to be a browser (for WAP, the Wireless Application Protocol) that enables free browsing by using free SMS. After the program is started, its only function will be to continuously send SMS messages [65]. The security architecture of J2ME will ask the user for confirmation, because the MIDlet is not signed. The user might

assume that these messages are free of charge. Due to the social engineering, the user is likely to approve the confirmation requests before an SMS is sent.

Redbrowser is a good example that technical security mechanisms fail when the user is able and willing to override them (cf. Section 3.5). The attacked user is likely to approve at least the first few confirmation requests before he might get suspicious.

Wesber. The Wesber trojan of September 2006 is similar to Redbrowser [66]. It is only worthwhile to mention, because it is the second sample of J2ME malware. Like Redbrowser, it sends out SMS messages to a premium-rate number. The messages are sent without country prefix and the premium-rate number was only valid in Russia, so the trojan cannot cause financial loss outside of Russian mobile networks. Neither does it employ social engineering, nor does it possess any other noteworthy features.

4.2 Phases of Malware

After describing mobile malware examples in the previous section, this section now abstracts from specific examples. The main phases of malware are seen here as *infection of the device*, *spreading*, and *malicious functionality*. These are mainly the phases of a self-replicating worm. In general, a taxonomy of malware is difficult [12] and the phases *trigger* and *making itself permanent* could be added [94], while at the same time the phase *spreading* can be combined with the phase *infection*, which is most useful for viruses. The chosen subdivision of this section is more general and is effective for the proof-of-concept malware in Section 6.1.

4.2.1 Infection

Infection is the phase when the malware infiltrates the device. Chapter 3 listed attack vectors, e.g., a technical vulnerability of the device or a social engineering attack.

Malware infection for such devices can be categorized according to the degree of user interaction that is necessary for the malware to infect the system. This results in four distinct classes of decreasing required user interaction:

1. *Explicit permission.* The most benign interaction is asking the user, whether it is allowed to infect the device, clearly indicating its potential malicious behavior. This is the typical behavior of proof-of-concept malware.

2. *Implicit permission*. The next category are the standard questions at installation procedures for (unsigned) software. The user might be accustomed to them because of previously performed installation procedures. This is the standard way how Trojan horses get installed, usually by seducing the user with social engineering techniques, so that he really wants to install the offered software (e.g., with a text like: “Free World Cup After-Party Ticket - just install”).

3. *Common interaction*. The third category is an action that is common behavior when using a mobile phone. An example is the MMS buffer overflow of Section 3.4.1.2, which only requires the user to open the MMS message.

4. *No interaction*. The most dangerous type of malware is a *smartphone worm*, which is able to infect a device without any user interaction. This would be the worst case concerning mobile phone security, but as of today, no such type of malware is known.

4.2.2 Malicious Functionality

Once on the device, the malware can commit its malicious actions. The possibilities for these actions under Windows Mobile comprise the entire system functionality [20]. For Symbian OS since version 9 with the Platform Security Architecture it is assumed to be more difficult for malware to perform malicious actions on the device, as the sensitive APIs are protected by the Symbian OS capabilities (cf. Section 2.2.4). However, if a signature of the Symbian Signed program can be obtained (like the spyware programs of Section 4.1.2), the protection of the PSA is undermined in some way. More formal treatments of possible malicious functionality can be found in the literature: from a scientific point of view [137] and from a mobile network operator point of view [166].

Malicious functionality is subdivided here into *monetary damage*, *data damage*, and *hidden damage*. Real-world incarnations of monetary damage and data damage have been given in the mobile malware descriptions of Section 4.1.

4.2.2.1 Monetary Damage

Monetary damage is the malicious functionality that the common user will notice in any case, because he has to pay for it. Other malicious functionality like stolen personal data, the use of the device as mobile botnet, or even denial-of-service might go mainly unnoticed: in case of a denial-of-service attack because the attack

might be attributed to a general failure of the device. Therefore, monetary damage is a malicious functionality that deserves the highest attention.

The most common functionality of current mobile malware is sending messages to premium-rate SMS numbers, because the malware author can earn money with this procedure. The same is true for dialing premium-rate phone numbers. As most current malware uses the specified API for accessing this functionality, a security-conscious user can prevent monetary damage most of the times.

An interesting consideration are “business models” for malicious software authors. Services like premium-rate SMS/phone numbers are not anonymous, because they must be registered with the mobile network operator. Calling phone numbers in foreign countries might be an option for increased anonymity and might be feasible by abusing the way how today’s billing models are constructed. However, the possibilities are finite and calling these numbers can be prevented with an appropriate security policy. Such a mechanism is provided by our solution in Section 6.2.

4.2.2.2 Data Damage

Data damage can be subdivided into *data theft* and *data destruction*. Data on the mobile device can comprise e-mails from the e-mail software or personal data like short messages, pictures, video and audio recordings, and in general the internal file system or memory cards with all personal or possibly corporate data. Data theft will try to get hold of this data.

Data destruction is the simplest way for a malware author to show that he is able to do something malicious. Besides the data types from above, also device configuration settings or specific parts of the file system can be targeted for advanced denial-of-service attacks like a non-reacting user interface or a device that is unable to boot properly.

4.2.2.3 Hidden Damage

Most interesting is malicious functionality that only makes sense in the mobile world, because the mobile device is carried with its owner most of the time. This functionality is intended to be more hidden from the victim than currently implemented denial-of-service or data destruction functionality.

There have been thoughts about advanced versions of spyware, e.g., audio recording software, backdoors into the corporate network [43], and video-capturing software

[234]. Also, location tracking is possible when a malware is able to access and forward location information (e.g., GPS signal or cell identification).

A transfer of common malicious functionality is the use of mobile devices as bots in botnets. The limited data network bandwidth is acceptable for most botnet functionality. Sending unsolicited bulk e-mails is possible and even denial-of-service attacks, because small individual bandwidths quickly accumulate to a large overall bandwidth. These botnets could be used in the traditional way against computers in the Internet or against the infrastructure of the MNO.

4.2.3 Spreading

In the spreading phase, a malware wants to distribute itself. It has to reduce the huge number of possible targets in the chosen spreading channel to a feasible selection. Channels for spreading are local connections like Bluetooth or wireless LAN, remote connections over the phone network (phone numbers), remote connections over the data network (IP addresses), and e-mail.

Spreading in the wireless LAN is easy for devices running Windows Mobile, as they announce themselves when connecting to the network. For other devices it is necessary to actively scan for new targets. The different characteristics of the spreading process are discussed by Mickens and Noble [131], who propose an extension of the commonly used Kephart-White model for modeling spreading between locally related devices (WLAN or Bluetooth). Other work is based on this new model, e.g., Zheng et al. [236].

When the malware uses telephone numbers as targets (e.g., when replicating via MMS), a reasonable choice is using the address book of the infected device. This strategy has a social engineering component, because the recipient knows the sender of the message and is more willing to open the message.

4.3 Mobile Malware Portability

The term “portability” has two dimensions, answering two important questions: Which properties does malware need to possess for spreading on different operating systems (Section 4.3.1)? And can mobile malware be ported between different mobile operating systems (Section 4.3.2)?

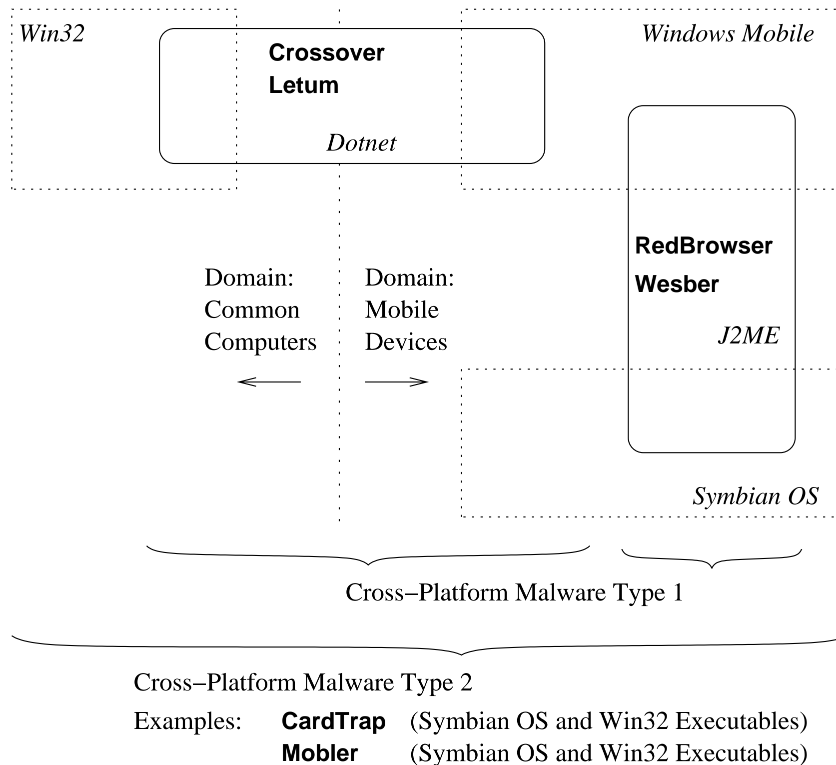


Figure 4.1: Cross-Platform Malware

4.3.1 Cross-Platform Malware

The topic of cross-platform malware is illustrated in Figure 4.1, answering the important question, which properties malware needs to spread on different platforms. This question has two answers, leading to two types of cross-platform malware: the first type of cross-platform malware uses application frameworks that have runtime environments on different platforms. The second type carries executables of different platforms and spreads them when it is able to use a data connection to a suitable target operating system.

4.3.1.1 Platforms

The term *platform* is used here in the common definition as "an environment that allows software to run". This definition used to mean a "hardware platform" with the corresponding machine code format, but the definition is extended with application frameworks (see below). An executable belongs to a platform, if the runtime environment of the platform is able to interpret the binary format of the

executable. That means for the upcoming two cross-platform malware types:

- The Windows EXE container format is not specific to a platform (e.g., Windows with Intel CPU, Windows Mobile with ARM CPU), but the contained machine code instructions are. Therefore, common computer EXE files cannot be executed on Windows Mobile, despite their common suffix.
- Application Frameworks provide an intermediary format ("bytecode language", cf. Section 2.2.2). Therefore, the executables can be interpreted on any hardware platform that provides a converter from bytecode language to the native machine code of the hardware. Examples are J2ME and Dotnet. Application Frameworks shift the boundaries of the traditional platform definition.

4.3.1.2 Cross-Platform Malware Type 1

Type 1 of cross-platform malware uses application frameworks that have runtime environments on different platforms. Examples are known for the Java Platform, Micro Edition (J2ME), and for the Dotnet environment. Not included in this type is malware exploiting a vulnerability of the virtual machine. Despite their ability to be executed in all runtime environments, the malicious functionality would be specific to a platform.

The classical J2ME malware is named Wesber and Redbrowser (cf. Section 4.1.3). These malware samples only use the specified programming interface and completely rely on social engineering. The malware Crossover (cf. Section 4.1.1) uses the Dotnet application framework. As Figure 4.1 shows, it crosses the platform borders between common computers running the Windows operating system and mobile devices running the Windows Mobile operating system.

The portability is restricted by the availability of libraries. Even though the application frameworks share the same binary format, they provide different libraries for programs. This has been introduced for J2ME in Section 2.2.2.2, the additional libraries could be the File Access API (JSR 75) and the Bluetooth API (JSR 82). As this diversity also affects "normal" software, it can be expected that initiatives like MSA (JSR 248/249) will provide a common runtime environment for J2ME software—and also for malware—in the future. Libraries of the Dotnet environment have been introduced in Section 2.2.2.6. Of special concern is the Smart Device Framework that tries to add the library functions of common Windows systems to mobile devices. Crossover circumvents the problem by using a runtime query to choose platform-specific functionality, this way having platform-specific code in a platform-independent container.

4.3.1.3 Cross-Platform Malware Type 2

Type 2 of cross-platform malware carries executables of different platforms and spreads them when it is able to use a data connection to a suitable target operating system.

Examples are Cardtrap and Mobler (cf. Section 4.1.2). This type of cross-platform malware can theoretically cross any platform borders, because it uses a dropper functionality to deliver an executable that fits to the target operating system. In practice, this type of malware needs a connection between the two operating systems to use as an attack vector. Memory cards can be used for this purpose or other file transmitting functionality like synchronization software to which the malware has access.

Most of today's cross-platform malware type 2 uses existing mobile malware samples as dropped malware and adds a custom wrapper to them. An example is the Cardtrap virus that carries the Windows worm Wukill.

4.3.2 Portability between Mobile Operating Systems

A second dimension of mobile malware portability is the dependency of current mobile malware on a specific mobile operating system. Most of today's mobile malware targets the Symbian operating system (cf. Figure 2.5), so it must be asked, if Symbian OS is less secure than the other operating systems. Therefore, this section contributes to the question, if the bias towards Symbian OS, which is illustrated in Figure 2.5, can be traced back to increased weaknesses of this operating system.

4.3.2.1 Common API

The question of portability between different mobile operating systems is the availability of relevant system calls. So the question is: what are the relevant system calls for mobile malware? An important functionality (cf. Section 4.2) is the spreading and the damaging mechanism. Both of these can be realized by sending MMS messages (e.g., realized by the Commwarrior worm). Spreading and making itself permanent on the device can be done by writing to the file system or the external memory card. It can reasonably be assumed for all this functionality to be accessible via using system calls by programs written for the operating system. As a subset of all programs, also malware is able to use these system calls.

Exceptions are Skulls, Locknut, or Blankfont, which are not portable. They all use the same vulnerability of the Symbian operating system. InfoJack uses a

Table 4.2: Mobile Malware Portability

operating system (OS)	*not portable*	*portable to other mobile OS*
Windows Mobile	InfoJack Pmcryptic	Dust Brador
Symbian OS	Skulls MetalGear Locknut Blankfont Fontal Mobler Acallno Beselo	Cabir Mosquitos Lasco Feakk Commwarrior Cardblock CardTrap FlexiSpy Yxe
J2ME	-	Redbrowser Wesber

vulnerability of the Windows Mobile registry that allows software to change the security settings for software installation themselves.

The portability of Commwarrior as an example will be presented in detail subsequently. Table 4.2 shows the portability of the mobile malware that has been introduced in Section 4.1.

4.3.2.2 Case Study Commwarrior

The Commwarrior worm for the Symbian operating system used MMS to spread its binary to other devices (cf. Section 4.1.2). It is an interesting question, if this functionality can be transferred to other operating systems. Especially Windows Mobile was of interest, because its native MMS composer software does not allow executables or installation packages to be added to the MMS message.

Our tests with a virtual MMS system setup (technically based on Mulliner and Vigna [147]) revealed: it is possible to send executables and installation packages via MMS messages to other Windows Mobile devices. The file suffix is most relevant for the MMS handling application: it is only recognized as executable, when the suffix is either EXE or CAB. Additionally, the MIME type has to be of "application/*", where the asterisk is a true wild card. These facts can contribute to the design of filters in the mobile networks.

4.4 Conclusion

This chapter contributed to the definition of the investigation subject mobile malware. Mobile malware is not a topic of the same importance as it is in the common computer world today, but has received continuous interest since the first version of modern mobile malware in the year 2004. The chapter listed surveys of mobile malware and presented notable examples of mobile malware together with their main functionality, with the goal of showing the status of what has to be defended against today.

Concerning portability of today's available mobile malware, it can be stated that most of today's mobile malware does not use vulnerabilities of its host operating system. Therefore, the dominance of Symbian OS malware in current statistics is not due to technical and security aspects.

An interesting dimension of malware portability is cross-platform malware. This type of malware can cross platform borders in nearly any direction, either because of using intermediate languages of application frameworks or by carrying different executables for each target platform. This leads to problems when these types of malware are supposed to be classified. There are only few incarnations of this type of malware today, but these blended threats may become a common type of malware in the future.

A bottom line of the current state of mobile malware: first, today's mobile malware incarnations are simple, mostly being Trojan horses attacking the user without the need to use a technical vulnerability of the mobile device. Second, mobile malware has many possibilities to commit its malicious actions once it is on the device. This is especially true, because the mobile device user sometimes overrides technical security mechanisms (cf. Section 3.5).

Chapter 5

Developing a Dynamic Malware Analysis Tool

The previous chapters laid the conceptual framework for the technical solutions of this thesis. This first technical solution chapter brings dynamic software analysis with the investigation of one particular run (cf. Section 2.4.1) to the Windows Mobile operating system. This chapter makes two contributions. First, it provides a dynamic analysis solution with advantages over existing solutions. Second, it is the basis for a novel device-centric solution in the next chapter.

Concerning the advantages over existing solutions: this tool solves the problem of logging a particular run of a Windows Mobile software sample for the first time. Most of the work in this area of dynamic analysis has been done for common Windows systems, and this chapter will point out why their approaches cannot be transferred to Windows Mobile. Either they use processor emulators with sophisticated interfaces (TTAnalyze [17]) or they use DLL overwriting techniques (CWSandbox [232]) that cannot be used for systems like Windows Mobile because they execute DLLs directly in ROM ("execute in place", cf. Section 2.2.3).

Abstracted from the different target operating systems, MobileSandbox has two conceptual advantages over the two solutions from above. First, it logs system calls not only at user level (CWSandbox) but even at the level of the kernel, enabling a more detailed system call log. Second, it can be integrated into a running device without any changes to the firmware of the device, a property that will be used in the next chapter.

One note on terminology: the word "sandbox" can be confusing, because of different meanings in other contexts. The word is also used in the context of application frameworks (cf. Section 2.2.2). For example, sandboxing in the context of the Java virtual machine means that a Java applet is not able to escape its

restricted environment. Another meaning is an environment that can be reset after use. This is especially useful for public computer terminals, which are able to provide a clean environment for every user. A third use of the word refers to the technique of *software-based fault isolation* [192]. Sandboxing in the context of the MobileSandbox tool refers to the topic of dynamic software analysis, the *execution of a program in a controlled environment* (as introduced in Section 2.4.1).

This chapter is structured as follows: Section 5.1 shows the design of the MobileSandbox tool. Section 5.2 gives some details of the implementation. Finally, the tool is evaluated in Section 5.3.

5.1 Design

5.1.1 General Design Considerations

When designing a sandbox, the question arises as to which extent behavioral data of a sample is detected and logged. A commonly used concept is monitoring the interaction between the sample's process and the operating system environment, that is, intercepting the system calls in either or both the user space and the kernel space. In the case of user-level interception, calls to the corresponding system libraries are monitored (for Windows Mobile this is mainly *CoreDLL.dll*), while in the case of kernel-level hooking, intercepting system calls inside the operating system kernel is required.

The following list shows examples for interesting operating system functionality, being implemented as system calls. Especially the last item is specific for the mobile world, which make this world unique:

- Which DLLs have been loaded?
- Which files have been created, opened, changed?
- Has the registry been read, has it been changed?
- Did the sample affect other processes?
- Did it start new processes?
- Was network activity present? What about messaging activity? Voice call? Bluetooth?

Designing a sandbox for a mobile device is similar to designing a sandbox for common computers. Windows Mobile provides nearly a full compatibility to the common Win32 API layer, so user-space-only sandbox solutions for Windows PC operating systems could be ported to Windows Mobile. A difference is the ROM-based "execute in place", because large parts of the operating system code reside in ROM. This restricts the number of feasible implementations when it comes to system call hooking, because it implicitly involves modifying data or code in memory that is only readable in the case of Windows Mobile.

For MobileSandbox, we have chosen to create a sandbox that is injected into the sample's process and the kernel process and monitors all system calls on both user-level and kernel-level. It works on any given device that runs Windows Mobile 5 or later.

5.1.2 Environment

An important design decision is the environment, in which the sandbox works. Most solutions execute the sample in a real operating system environment and let the sandbox inject into the running sample process. In this case, system call interception works on the same level as the sample. This has some implications. A positive aspect of this solution is that it is well-established and usually very fast. On the negative side, the sandbox can easily be detected by the sample, because both share the same permissions. One solution to this problem is implementing the sandbox only as kernel mode process and to implement some kind of rootkit functionality, which allows to completely hide its presence. However, this would also have a negative impact on the expressiveness of the analysis results, since the operating system implements certain system calls in user space only for improving performance and the information that we get on kernel-level is generally less informative.

In this context, the most powerful solution is emulating the entire hardware environment (examples for the common computer world are names like QEMU or Bochs). These emulators are generally slower than a real system, and for the ARM-based Windows Mobile no emulator exists. The only emulator that is known to us is the standard Windows Mobile emulator, but it does not provide an API for reading or writing memory, so it cannot be used for the purposes of dynamic analysis for the Windows Mobile environment.

Therefore, we decided to implement a solution that executes the samples in a real operating system environment, in user space and in kernel space. With this decision, MobileSandbox is able to run on the device emulator and on real devices.

5.1.3 Logging

Another design decision is defining a place to store the log data. There are two general possibilities: logging on the device or logging to a remote location. A local log file on the device is the easiest solution, because it can be read after the analysis has finished. This has two main drawbacks. First, a log on the same device as the analyzed sample is accessible by the sample itself and therefore the log file might be compromised before it is read. Second, malware can render the system unusable, e.g., by causing a continuous reboot as Fontal.A does. The log file cannot be read in this case and is unusable as well. These problems are solved when logging to a remote location, as soon as the log information is present. It can be implemented in a variety of ways, as the main purpose of mobile devices is communicating with the outside world. IP is a good choice for achieving this, because the IP protocol is implemented on top of nearly every transmission technology today.

MobileSandbox uses a TCP connection with a host computer over an ActiveSync connection. Additionally, it stores a log file on the device, in textual form or in a binary format for large analyses.

5.1.4 Analysis Duration

A remaining question is, for how much time a sample should be analyzed. In general, the malware sample can be executed for an arbitrary duration. As Section 4.2 said, it is likely that malware shows its most interesting behavior quite at the beginning of execution, because it can never be sure to get a second chance or more time to perform its malicious actions.

MobileSandbox analyzes a sample for two minutes by default. However, if the analyst assumes that the investigated malware requires certain user interaction or other actions to show its malicious behavior, he might choose to increase the analysis duration.

5.2 Implementation

5.2.1 Components of MobileSandbox

The sandbox consists of the following files (illustrated in Figure 5.1):

- *MSandboxDLL.dll*: This is where the user-level hooking and the main part of the hook-handling are implemented. The DLL is injected into each analyzed process. See Section 5.2.3.

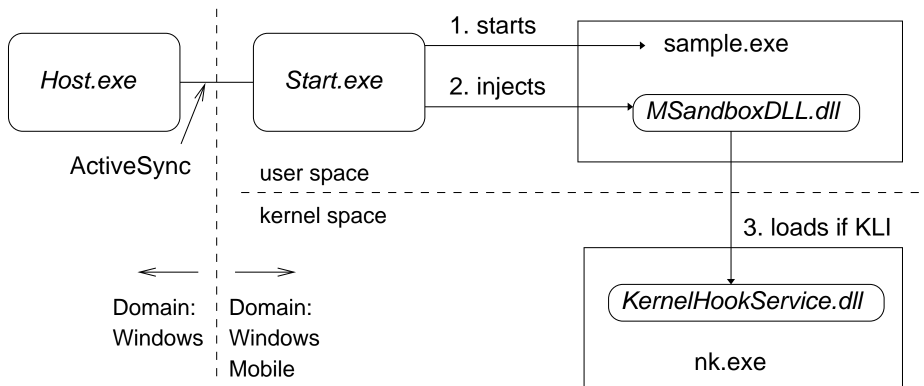


Figure 5.1: Structure of MobileSandbox

- *KernelHookService.dll*: This DLL contains all the kernel-level hooking code. It is injected into the kernel process *nk.exe*. See Section 5.2.4.
- *Start.exe*: This program initializes the process which should then be analyzed and thus performs the injection of *MSandboxDLL*.
- *Host.exe*: In contrast to the already mentioned files, *Host.exe* is a Win32 PC program. It holds a TCP connection to an attached Windows Mobile device via ActiveSync. It is responsible for the initialization of an analysis and receives log data directly from the device's *MSandboxDLL*.

Four stubs are set up at runtime for every system call (see Figure 5.2):

- *PreProlog*: This stub prepares the entry of the general prolog function *MainProlog*.
- *PostProlog*: After returning from *MainProlog*, we jump to the actual system call that the caller requested and set up the CPU registers so that the call will return to *PreEpilog*.
- *PreEpilog*: After the system call has executed successfully, it returns to this method. *PreEpilog* sets up certain parameters and enters *MainEpilog* afterwards.
- *PostEpilog*: After returning from *MainEpilog*, control is handed back to analyzed program with a jump.

Each stub is made up of a small number of ARM assembler instructions. This is necessary, because we need direct access to the CPU registers for not corrupting the parameters, which would inevitably lead to program inconsistencies.

5.2.2 Prolog and Epilog

Our design approach uses system call hooking (cf. Section 2.4.1) that substitutes certain pointers of the system with pointers to our solution . The original call and the hooked call will take different ways through the system, as soon as the executed code is outside of the investigated application. This will be explained in technical detail for the two cases of import address table patching and kernel-level hooking in Section 5.2. Abstracted from this implementation, the handling of the hooked call is similar and will be described here (the structure with the involved entities is visualized in Figures 5.2 and 5.3a).

Control over the system is handed over to MobileSandbox before and after the original system call in the central functions *MainProlog* and *MainEpilog*. These functions only exist once within the system, therefore every hooked system call has individual stubs that prepare the entrance of *MainProlog* and *MainEpilog* and perform cleanup operations when the hook is finished. They are replicated for every API function and these replicated stubs only differ in the individual data that is passed as parameters to *MainProlog* and *MainEpilog*.

MainProlog is responsible for logging the intercepted system call and for intercepting system calls that need to be handled to ensure the integrity of the sandbox. Examples are system calls that create a new process (e.g., *CreateProcess* and *ShellExecuteEx*) or return pointers to other system calls (e.g., *GetProcAddress*). If a new process is created, the sandbox has to prepare the new process for being hooked. If a pointer to another system call is returned, MobileSandbox needs to change the returned address and let it point to our corresponding *PreProlog* stub instead.

MainEpilog first logs the return value of the system call and might also modify the return value, for example, when *GetProcAddress* was called. As previously said, we then set up new stubs and let the return value point to the new *PreProlog*, because otherwise MobileSandbox would miss the subsequent calls of this system call.

5.2.3 User-Level Hooking

This section explains the user-level hooking of system calls. MobileSandbox uses the technique of import address table patching. This is a basic technique for intercepting system calls, but malware can easily evade this hooking type, requiring the kernel-level hooking techniques of Section 5.2.4. Nevertheless, import address table patching is useful to keep the logs readable. User-level system calls sometimes

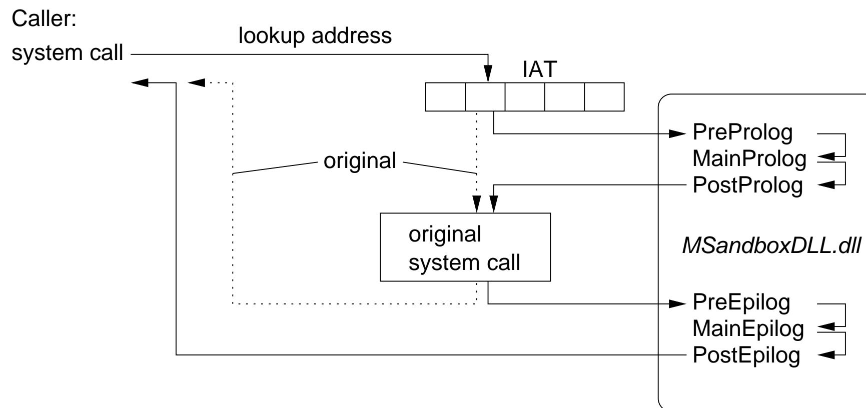


Figure 5.2: Import Address Table Patching

lead to several kernel-level system calls, where the user-level call is more concise and much more expressive.

This import address table patching part of MobileSandbox is similar to CWSandbox from the Win32 world, but it uses a different method to intercept the system calls: CWSandbox rewrites the first portion of the method in the DLLs. This is impossible in Windows Mobile because many DLLs are saved in read-only memory. We use another standard method instead, patching the import address table (IAT).

When an executable starts, the Windows loader looks up the addresses of each used system call and inserts them into the IAT, because these addresses are not known at compile time. A system call in the program reads the address of the system call out of the IAT and jumps to this address afterwards. This normal way is represented by the dashed line in Figure 5.2.

After the Windows loader filled the IAT, MobileSandbox does some steps that will lead to the way of the solid lines in Figure 5.2. The address of every entry in the IAT is changed. For every changed address four functions are set up (*PreProlog*, *PostProlog*, *PreEpilog*, *PostEpilog*). They handle saving and restoring the current processor state and calling the two main functions of MobileSandbox (*MainProlog* and *MainEpilog*). The IAT entry for each system call now points to its corresponding *PreProlog* function, which is the unique entry point for every system call.

A malware sample is able to circumvent this user-level hooking, because a program may calculate the system call address itself and does not need to use the IAT . Whenever it wants to use a system call, it sets the address and sets the system into kernel mode. User-level hooking is not able to log this event, because it has no access to the kernel structures. So, an extension within the kernel is necessary that

is described in the following section.

5.2.4 Kernel-Level Hooking

The deeper level of intercepting system calls at the kernel level is explained here. As already said, this is necessary when programs do not use the import address table and it is important for hiding the presence of a sandbox solution from the analyzed sample.

As previously introduced, our sandbox solution consists of two different DLLs (cf. Figure 5.1), one being responsible for user-level hooking (*MSandboxDLL*) and the other one taking care of kernel-level hooking (*KernelHookService*). This separation is a consequence of the architecture of Windows Mobile system calls and the fact that there might be several sandboxed processes at a time. The kernel-level DLL is loaded on initialization of a sandboxed process by the user-level DLL.

It is a vital point, where the kernel-level DLL is positioned within the system kernel processes. As already explained, system calls are executed in many different processes. Therefore, the kernel-level hooking code has to be accessible from every system kernel process, because the kernel switches to the address space of the system kernel process before performing the call. One solution is to inject the DLL into every PSL process. MobileSandbox uses a different solution by injecting its kernel-level DLL *KernelHookService* only into the *nk.exe* process. Because the kernel switches a user-level thread into kernel mode before performing a kernel-level system call, our code in *nk.exe* will be accessible, because Windows Mobile allows all kernel-level threads to access each other. In order to inject into *nk.exe,* MobileSandbox uses the undocumented *PerformCallback4* function, which executes code in another process just like in a system call. By using this technique, MobileSandbox executes the *LoadLibrary* function in the process of *nk.exe* with a global pointer which points to the name of our kernel-hooking DLL.[1]

To go into detail, a system call goes through the following stages when intercepted at kernel-level (illustrated in Figure 5.3b):

Start. When the system call is processed by the exception handler, the corresponding function address is extracted from the method pointer that was previously patched by the sandbox. Thus, our individual stub instructions are executed rather than the intended function. As explained above, the kernel switches to the address

[1] The *CreateRemoteThread* API would have been useful for this task, but is not available on Windows Mobile.

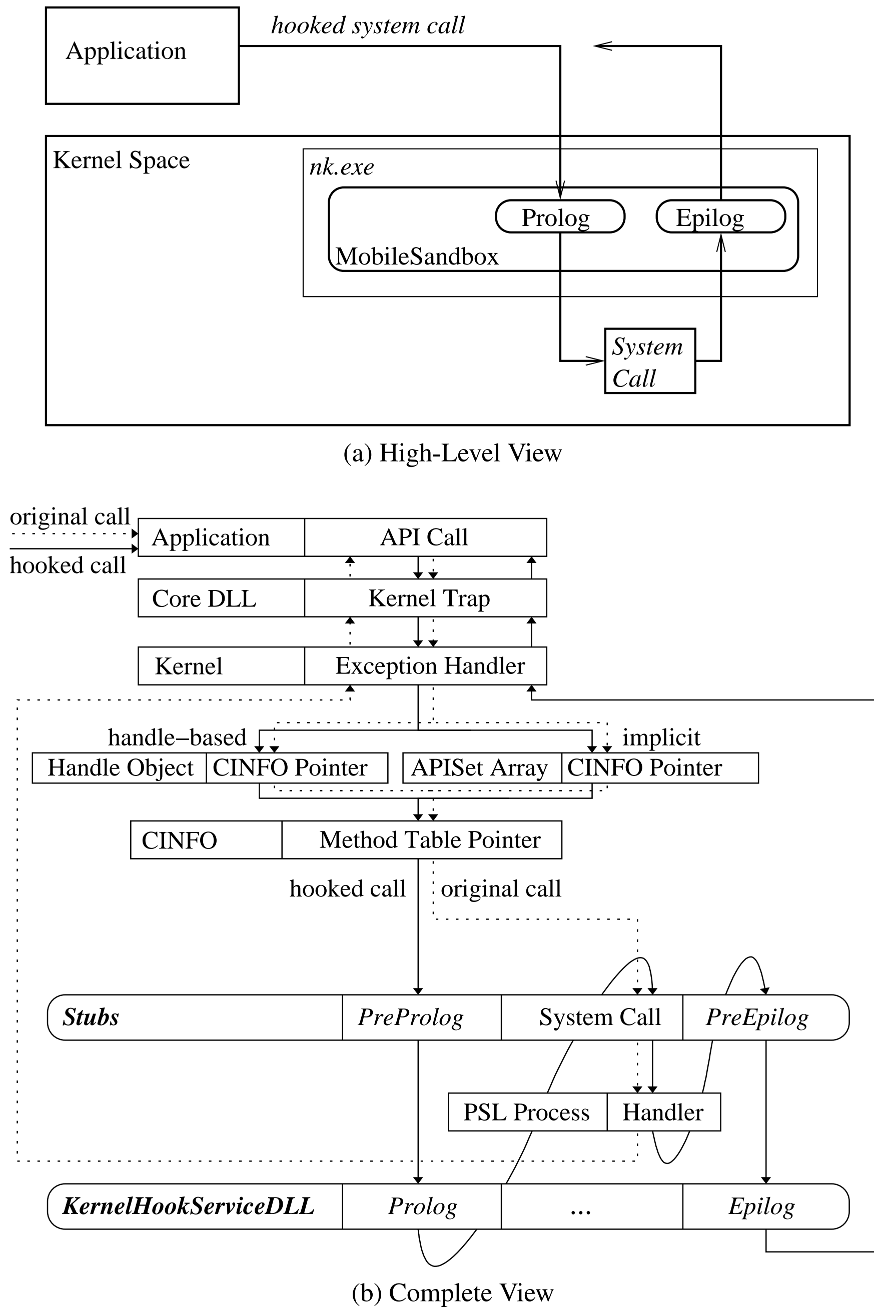


Figure 5.3: Kernel-Level System Call Hooking

space of the PSL process, this has to be taken into account when dealing with pointers.

Stub. The task of the stub is to prepare and call our unique *MainProlog* function. Care has to be taken to not alter registers values, because they might hold some of the arguments of the system call and might be used later on. Therefore, the first step is to save all registers to the stack, followed by setting up the arguments of the prolog, which are the *APISet* identifier and method identifier as well as a pointer to the current stack where the register values were stored. On ARM processors, the first four arguments are passed in the general purpose registers R0-R3, whereas the rest is stored on the stack. Additionally, the contents of the registers R4-R12 have to be preserved through function calls.

Prolog. The prolog first checks, which process has initiated the system call. If this process is not sandboxed, it returns immediately. Additionally, kernel-hooking might be deactivated for single threads under certain circumstances. In this case, the prolog also returns. As an example, we only intercept the first level of system calls, because system calls within system calls are not of interest. We only care about the sandboxed application and not the way how system calls are implemented in a PSL, so this is ignored. As another example, a system call might already have been intercepted in user space by IAT. Special care had to be taken of the correct implementation of these conditions, because the kernel might quickly hang in an endless loop otherwise, when an intercepted system call performs system calls itself.

If the prolog has decided to intercept the call, it writes the parameters of the call to a shared memory region and indicates that there is a system call to be executed by triggering a special event, causing the special thread in the application address space to further process the interception. This includes extracting and logging the parameter information of the call. When finished, a second event is triggered that wakes up the sleeping kernel mode system call. Events are indicated using standard inter-process communication functions such as global mutexes or global events. Eventually, the prolog returns, register values are restored from the stack, and the original system call is executed.

Epilog. In case the system call was hooked, the stub also prepares the entry of a generic epilog hook function after the call was executed. The epilog goes through the same stages as the prolog. In some situations, it might also modify the return value of a system call. For instance, this could be necessary when the sandbox wants to hide its presence.

5.2.5 Portability

The portability of the presented approach to other mobile operating systems is an interesting question. Our approach is based on the facts that Windows Mobile allows untrusted processes to inject DLLs into other processes and to enter kernel mode. This functionality is specific to Windows Mobile with its common security-unaware architecture. However, if relaxing the requirement of being able to run on actual devices, there are possibilities because of the device emulator architectures, running on common computers.

Symbian OS. As an example we consider Symbian OS with its Platform Security Architecture. Access to operating system functions is only possible by using the specified interfaces of the servers (cf. Section 2.2.4) and access to them is generally restricted. This can be seen, when even accessing file system APIs with anti-virus or encryption software needs a sophisticated Symbian DevKit [210] and the TCB capability, which can only be granted by the phone manufacturer. In old versions of Symbian OS, system files could be overwritten, which is exploited by a number of mobile malware samples, e.g., Skulls and Blankfont, (cf. Section 4.1.2). Therefore, it could be possible to implement a dynamic software analysis solution even on real devices, at least on the devices where the above named mobile malware runs as intended.

Another option is to restrict the dynamic analysis to the emulator. With the Symbian emulator, it is possible to exchange system DLLs with custom DLLs. The official tool HookLogger [209] uses this method for enabling advanced debugging functionality. One option is to log all API calls, which proves the desired functionality to be implementable.

That means, porting MobileSandbox to Symbian OS is possible, at least for the device emulator. Either the tool HookLogger can be used or the architecture of MobileSandbox can serve as a basis for a Symbian OS sandbox, but major changes in the implementation can be expected.

5.3 Evaluation

This section evaluates the MobileSandbox tool. It will investigate the performance of the solution, discuss the completeness of the logged system calls, and present analyses of some current malware samples for Windows Mobile.

Table 5.1: Performance of MobileSandbox
(average/median/standard deviation in milliseconds)

System Call	*Normal Time*	*User-Level Hooking*	*User-Level & Kernel-Level Hooking*
Sleep	104/101/9	113/109/6	116/115/9
GetSystemInfo	0/0/0	25/15/22	24/16/32

5.3.1 Performance

This section shows how the use of MobileSandbox influences the performance of the investigated program. We have chosen to measure at the level of single system calls for eliminating side effects. The *Sleep* system call is a good indicator of performance loss, because its execution time is defined. The other system call is *GetSystemInfo*, which is considered to return its result as fast as possible and without any significant overhead. Table 5.1 shows the results of the measurements. The first column shows measured times without any system call hooking at all, the second column with import address table patching, and the third column with both hooking methods. We used a Windows Mobile device emulator running in a virtual machine for our measurements. We called both system calls ten times and the presented numbers are the average value, the median, and the standard deviation.

The *GetSystemInfo* system call has large values for standard deviation, so it is not useful for measuring despite its simplicity. The results for the *Sleep* system call indicate that our system adds an overhead of of about 10 percent for user-level hooking and about 15 percent for combined user-level and kernel-level hooking. This is an acceptable result together with the intuitive feeling that MobileSandbox does not delay execution times for complete samples.

5.3.2 Completeness

There are two aspects when considering completeness: interception of every system call and recognition of the signature of the system call (that is, its parameters). The solution for both aspects is described subsequently.

5.3.2.1 Interception

The most important part is to see every system call. This is achieved through the technique depicted in Figure 5.3. MobileSandbox changes the central pointer for

the data structures to point to its own data structures, and there is no other way for a program to enter kernel mode when using system calls. However, there are several special cases to consider: handle-based system calls and the services of MobileSandbox itself.

Handle-based system calls load the kernel space addresses at the handle's creation time. Therefore, it is necessary to change the addresses there, so that these system calls do not circumvent our system. An example system call is *CreateFile*, where pointers to handle-based system calls (such as *ReadFile*, *WriteFile*) are maintained in an individual *CINFO* structure which is connected to the handle object. Therefore, one has to patch the handle right after it was created.

Another special case is the *KernelHookService* DLL of MobileSandbox. It provides some services that are necessary for the system but that are not intercepted.

5.3.2.2 Signature Recognition

The signatures of the system calls can be found in the header files of the shared Windows CE source code that is distributed with the Platform Builder. These header files have a unique format that can be parsed. The system calls are grouped into different *APISets*. These are documented as comments in the header files. The source code can be parsed with a tool like doxygen and the actual signatures can be assigned to the system call in its corresponding *APISet*. Some system calls are undocumented, therefore they are not present in the shared source header files. Typical examples are the GWES (graphics, window and event subsystem) API functions. All of these are intercepted, but it might happen that their signature is unknown. This case requires manual effort to locate the signature. This can be solved by using a disassembler (like IDA Pro) and decompiling the library file.

5.3.3 Malware Analyses

This section shows excerpts from the analyses of current Windows Mobile malware and compares the results with the public analyses that were done with manual analysis techniques. This should show the strengths and weaknesses of MobileSandbox for real-world malware. Short descriptions of the investigated malware samples were already given in Section 4.1.1.

5.3.3.1 Malware "Dust"

Dust is a good example for demonstrating the usefulness of MobileSandbox for dynamic malware analysis, because its source code has been published and it has

```
mov lr, pc
ldr pc, [r11, #-24] ; find first file
```

Figure 5.4: Dust Source Code (Excerpt)

been analyzed thoroughly [173], so our analysis can directly be compared with the source code. Because of that, it is the most interesting piece of native malware for Windows Mobile. Dust does not use the import address table to access the system calls, so a sandboxing solution with only IAT patching would not see the interesting parts of its behavior. Instead, Dust calculates the addresses with the formula of Section 2.2.3 and directly jumps to these addresses. An example are two lines of the source code (Figure 5.4), where the program counter is set to a value of the stack that was previously set to the address of *FindFirstFileW*. This corresponds to API call ID #3 in Figure 5.5, where the direct jump is indicated by the "(Kernel API)" statement in the log.

We used a device emulator with Windows CE version 5. The original Dust sample did not start here, because it was written for Windows CE version 4, where programs were automatically started in kernel mode. An addition of a call to *SetKMode* and a compilation of the source code was a necessary step to make the malware execute in MobileSandbox.[2] More of the analysis can be seen in Figure 5.5. You can see the proof-of-concept nature of Dust in API call ID #2, where the program asks, if it is allowed to spread. For this analysis, we chose "Yes" in the message box. The program then searches for files with the pattern `*.exe` in the root directory. The first returned file is *Start.exe,* which happens to be a file of MobileSandbox itself, but this fact does not influence the analysis. The program then starts the infection (calls #3 to #6) and finishes it after a few other actions in call #17. Calls #18 to #21 show that no other files are found and Dust terminates itself.

5.3.3.2 Malware "Pmcryptic"

Pmcryptic is the most current Windows Mobile malware, found in November 2008. Its main malicious functionality is hiding files and dialing premium-rate phone numbers. Figure 5.6 shows an excerpt, but the complete analysis is available.[3] The analyzed file is named after its MD5 hash value:
`0x33aea5c280aff3dff0badfba9a01f925`.

[2]We are aware that we created a new malware variant with this proceeding.

[3]http://mobilesandbox.org/analysis_pmcryptic.html

ID	API call	Arguments	Return value
2	USER_MessageBoxW (Kernel API)	hWnd=0 lpText=Dear User, am I allowed to spread? lpCaption=WinCE4.Dust by Ratter/29A uType=4	6
3	FS_FindFirstFileW (Kernel API)	lpFileName=*.exe lpFindFileData=639770832	3987199810
4	SC_CreateFileForMapping (Kernel API)	lpFileName=Start.exe dwDesiredAccess=3221225472 dwShareMode=0 lpSecurityAttributes=0 dwCreationDisposition=3 dwFlagsAndAttributes=0 hTemplateFile=0	229103390
5	SC_CreateFileMapping (Kernel API)	hFile=229103390 lpsa=0 flProtect=4 dwMaxSizeHigh=0 dwMaxSizeLow=42496 lpName=0	3449191462
6	SC_MapViewOfFile (Kernel API)	hMap=3449191462 fdwAccess=6 dwOffsetLow=0 dwOffsetHigh=0 cbMap=42496	1200619520
	...	...	
15	SC_UnmapViewOfFile (Kernel API)	lpvAddr=1200619520	1
16	SC_MapCloseHandle (Kernel API)	hMap=3449191462	1
17	FindNextFileW (Kernel API)	hFindFile=406608 lpFindFileData=639770832	1
18	FindNextFileW (Kernel API)	hFindFile=406608 lpFindFileData=639770832	0
19	FindClose (Kernel API)	hFindFile=406608	1
20	SC_ProcTerminate (Kernel API)	hProc=66 dwExitCode=765234978	1906388608

Figure 5.5: Analysis of Dust (Excerpt)

ID	API call	Arguments	Return value
5	wcscmp	Arg0=\system.exe Arg1=\33aea5c280aff3dff0badfba9a01f925xx.exe	1
6	_wcslwr	Arg0=1174096	1174096
7	wcsstr	Arg0=\33aea5c280aff3dff0badfba9a01f925xx.exe Arg1=autorun.exe	0
	...	...	
13	CopyFileW	lpExistingFileName=\33aea5c280aff3dff0badfb... lpNewFileName=\system.exe bFailIfExists=0	1
14	Sleep	dwMilliseconds=1000	0
15	SetFileAttributesW	lpFileName=\system.exe dwFileAttributes=2	1
	...	...	
22	CreateProcessW	pszImageName=\system.exe pszCmdLine=0 psaProcess=0 psaThread=0 fInheritHandles=0 fdwCreate=0 pvEnvironment=0 pszCurDir=0 psiStartInfo=0 pProcInfo=0	

Figure 5.6: Analysis of Pmcryptic (Excerpt)

The first interesting logged system call is the fifth call: The program checks, if its own previously retrieved filename equals `\system.exe`. After a lowercase conversion of a string in call #6, the program checks in call #7, if its own filename contains the string `autorun.exe`. Both of these comparisons will fail, because of the non-matching file names. It can be assumed that the result of these checks will lead to different behavior of the malware. Figure 5.6 shows the case of initial infection mode.

Call #13 copies the original malware executable to the file *\system.exe* in the root folder. The next call #14 stops the program execution for one second, before call #15 sets the file attributes to *FILE_ATTRIBUTE_HIDDEN* (=`0x02`). That means, the file will not be included in an ordinary directory listing.

The last logged call #22 is *CreateProcessW*. It executes the newly created hidden file *\system.exe*. Unfortunately, the analysis ends here, because MobileSandbox does not follow newly created processes at the moment. Another deficiency could be seen in call #6: the call takes a *wchar_t** pointer as an argument and returns a pointer of the same type. Unfortunately, the pointers are not dereferenced, so the string's content cannot be seen.

5.3.3.3 Analysis Extensions

The malware "Dust" shows its malicious behavior every time it is started, but some malware—be it mobile or common—is dependent on certain user actions or other conditions. Especially trojans show no special behavior most of the time, but might send a text message to a premium-rate number from time to time. So it might be necessary to run the suspicious piece of software for some time and to interact with it. MobileSandbox supports this approach, because the device (or the device emulator) can be used as usual when run with a local version of MobileSandbox.

The following list shows some useful extensions that certainly will improve the analysis. They are not implemented at the moment, but they are on the wish list. They give an idea of the possible future of dynamic analysis:

- Logging system calls over an extended period of time leads to new challenges for the analysis environment, because the log can easily grow very large. This can be solved by implementing some kind of "capture filters" that do not log every system call, but only the calls the analyst is interested in, for example networking and messaging activities.
- A sample of mobile malware might not show its malicious behavior directly but install other programs that are no direct child processes, but instead are

triggered by an external condition. In the environment of mobile devices this can be a messaging filter. They are triggered if an incoming message (SMS or MMS) matches its filter condition. If such a filter is installed, its actions should be logged just as the original sample. Now, the analyst can send messages to the device and find out what the filter does, for example by watching the filter conditions or text comparisons of the code.

- Loops in the source code lead to long log traces that are very similar but that cannot be recognized as related at a first look. An example is the presented Dust analysis, where Dust scans every file in the root folder. An important extension is combining logs of loops in a readable way, maybe even graphically. This can easily be achieved by using the program counter values of the logs.

- Two more improvements are useful when the sample changes the user interface or the file system. These changes could be added to the log, user interface changes in the form of screenshots and file changes in the form of diffs to the previous version. Both extensions would simplify the task of a malware analyst even more, but may increase the log size considerably.

- In case of an increasing number of malware samples, a framework can be built around MobileSandbox for a massively automated analysis of malware samples. This heads into the direction of CWSandbox today.

5.4 Conclusion

This chapter presented MobileSandbox, a tool for dynamic software analysis of Windows Mobile executables, i.e., the execution of a program in a controlled environment. It gave insights into design and implementation, and proved its performance penalty to be acceptable. Most interesting is its application on real Windows Mobile malware. Despite some rough edges, the tool proved itself to be functional, enabling the dynamic analysis of Windows Mobile software without any technical knowledge (like being able to use a debugger or disassembler).

The MobileSandbox tool will be used in Section 6.2 as a building block for a policy enforcement solution and in Chapter 7 as a part of network-centric security mechanisms, where the question is investigated how restrictive the limitation of being able to log only one sample execution is for a large-scale system.

Chapter 6

Device-Centric Security Investigations

This is the second technical solution chapter of this thesis. It gives two contributions: developing a smartphone worm and providing a mechanism to protect the device against this threat.

First, Section 6.1 investigates the efforts that have to be taken to develop an autonomously spreading smartphone worm for Windows Mobile. A smartphone worm is the most dangerous category of mobile malware. Fortunately, there are currently no incarnations of this category, which was the reason for actively researching characteristics and countermeasures to learn more about its associated threats.

The focus on a worm restricts the possible attack vectors to interfaces that have connections to other devices, e.g., Bluetooth, Infrared, IP over WLAN or over the mobile network, of which IP over WLAN was chosen because it appeared to be most promising. The possible vulnerabilities include buffer overflows, heap overflows, and race conditions. Buffer overflows are chosen here as a vulnerability that was seen as the most dangerous programming error of the recent years [38], that is easier to exploit compared to heap overflows, and that has been found in Windows Mobile 2003 (cf. Section 3.4.1.2). We utilize the approach of fuzz testing for finding a vulnerability because the previously named buffer overflow was found with fuzz testing.

The results of our investigation are embedded into a three phases model of breaking a system and are used to derive a cost-to-break metric for Windows Mobile. This investigation shows that it is possible to come very close to the target of developing a smartphone worm with reasonable effort.

The second part of the chapter in Section 6.2 shows device-centric parts as necessary for smartphone security, because some important security requirements can only be implemented on the mobile device itself. It uses the MobileSandbox dynamic software analysis tool of the previous chapter as a basis for a novel security mechanism—the *policy enforcer*—on mobile devices that aims at reducing the attack surface of mobile devices to mobile malware.

The policy enforcer extends related work [102] in two ways. First, it is applied at the deeper level of the device's operating system, not on a virtual machine level. Therefore, it can be used for native programs, enabling a broader application. The boundaries are equal to the physical device, which is an important property of usability. Second, it can be applied to existing devices without changing the operating system. That means, we implement the concept of a reference monitor as a flexible solution that can be applied to existing devices as an add-on in contrast to a major change of the operating system.

6.1 Developing a Smartphone Worm

This section focuses on malware for Windows Mobile. Section 4.1.1 showed that only few instances of malware for Windows Mobile are known today (a complete list in August 2009: Dust, Brador, Crossover, Infojack, Pmcryptic). According to the classification of Section 4.2.1, neither any of these programs can be called a smartphone worm, nor the existing malware for other mobile operating systems. However, according to the classification, a smartphone worm is most dangerous concerning infection, thus we need to actively research characteristics and countermeasures to learn more about the associated threats. This part of the thesis makes most use of taking the attacker's point of view [79].

The remainder of this section is structured as follows: our approach of developing a proof-of-concept worm for Windows Mobile is presented in Section 6.1.1. The development of building blocks as the constant part of smartphone worm development is presented in Section 6.1.2. The search for buffer overflows by employing fuzz testing as the variable part of development will be presented in Section 6.1.3. Fuzz testing or *fuzzing* means automatically searching for vulnerabilities in software. The attacker generates input that complies with the structure of the expected input, but contains random data that is intended to make the targeted process fail. If this happens, the next step is analyzing the possibility of using this failure for injecting malicious code into the process.

6.1.1 Proof-of-Concept Malware for Mobile Devices

Many standard techniques for malware development known from the desktop and server world do not work in the mobile environment. As an example, the hardware architecture used in smartphones (ARM) is very different from the well-understood Intel x86 family, a fact that leads to additional demands for shellcode development. Operating system concepts like processes, scheduling and virtual memory are also quite different in mobile devices, even between different versions of mobile operating systems. Thus, it is neither clear how easy it is to stage standard attacks from the PC world in mobile environments, nor how easy it is to transfer known attacks for one version of Windows Mobile to the next.

In this Section 6.1, we investigate the question of how easy or difficult it is to create a smartphone worm for Windows Mobile version 5. We do this by measuring the effort needed by a skilled individual to create such a worm using standard methods of penetration testing, standard security tools, and software engineering methods. Other proof-of-concept malware for mobile devices has been developed by Molitor [139], Jamaluddin et al. [103], Haas [93], Emm [50], and Cheng et al. [33]. However, none of these publications measured the effort of developing the malware.

This section aims at measuring the effort that is needed to find a vulnerability. The intrusion process of breaking a system can be seen as subdivided into three phases (according to Jonsson and Olovsson [110]). The first phase is a *learning phase*, where information about the targeted system is collected. The *standard attack phase* consists of a number of fairly straightforward actions and standard tools for breaking a system, with a rather high probability of success. The *innovative phase* consists of inventing new methods for breaking a system. The time between successful breaches increases, because it can never be said, after how much time the ideas of an attacker will lead to an effective attack. This model can be applied to our situation. When starting our efforts, we were in the innovative phase of the model. The situation changes with the publication of our results, because parts of them can now be taken as successful (i.e., standard) attacks on the system.

We managed to build a prototype worm that can spread autonomously in case a vulnerability is given in a Windows Mobile service that is accessible over the network. The working effort encompasses the following tasks:

1. Building a worm toolkit, i.e., a software library that contains shellcode and worm spreading functionality. This is what we call the *constant part* of the attack. It is developed once and can be used for any future vulnerability, possibly with slight modifications that can in terms of time still be regarded

as constant compared to the variable part. The effort required for achieving this was about 13 weeks of full time work.

2. Finding a vulnerability in the network interfaces of Windows Mobile 5. This task included attempts to execute known attacks for Windows Mobile 2003 as well as applying the technique of fuzzing to the WLAN network stack of Windows Mobile 5 to find buffer overflows. This is what we call the *variable part* of the attack.

This separation might increase the needed time for development slightly, but it is favorable in terms of effort, because the attacker is also able to incorporate vulnerabilities, that have been found by others. So, using this procedure of modularization—the classic software engineering approach—the attacker maximizes the effects of his work by enabling the reuse of his code.

Both parts are presented in the section, but the focus is clearly on the constant part, i.e., on the building block for a smartphone worm. The variable part is only shown as an example here. In both parts, the efforts for developing them was measured. The effort necessary for the second task can only be approximated because we terminated our efforts after forty hours of work without finding a new vulnerability that could be exploited by a worm. We however found two denial-of-service vulnerabilities, on which we report.

To summarize, we found that it takes roughly 600 work hours (14 weeks of full time work) to come very close to the target of building a smartphone worm. The benefit of our work is that it helps to understand the difficulties adversaries face when attacking Windows Mobile. It also highlights the strengths and weaknesses of Windows Mobile version 5 over version 2003 as well as the general difficulties of attacking ARM-based architectures.

Most importantly, however, the insights from our study can be used to *estimate* lower bounds of cost-to-break metrics for current and future versions of Windows Mobile. Cost-to-break metrics are usually used to assess the security of safes. The situation is different in computer security because the tools are different and an attacker can be lucky to find an exploitable vulnerability in a short time, which is unlikely for real-world safes. This topic of IT security metrics is further discussed in the literature [186] .

Even if many parameters are likely to vary in different environments (skilled individual, tools available, exploit finding), at least the constant part of our smartphone worm can be used to estimate the lower bound in case of attacking an operating system where such a worm toolkit is not available in advance. We are unaware of any other work that has measured attack effort as precisely as we have.

6.1.2 Constant Part: The Building Blocks for a Smartphone Worm

This section documents the development of the building blocks for a smartphone worm. It describes the infection functionality, the spreading functionality, and the steps for assembling the building blocks.

6.1.2.1 Infection

The infection routine is the first part of the building block. It injects code into the mobile device by exploiting a security vulnerability, and gains access to the operating system and its resources. The injected code is the *shellcode* and its development is the main task in this section. Besides the covered topics, problems of zero-free shellcode and self-modifying code on the ARM architecture arise. They are addressed in an extended version of this work [20].

Vulnerable Application. For the development of shellcode for a smartphone worm, a program is needed that can be exploited. The program contains a stack-based buffer overflow, a security vulnerability that is a very common programming error in programming languages like C/C++. The vulnerability is used to create a proof-of-concept for exploitation of this class of vulnerabilities.

Stack-based buffer overflows. Exploiting stack-based buffer overflows in Windows Mobile 5 works in principle the same way as it does in other operating systems. A local buffer in a function is filled with user data that exceeds the buffer boundary. This way, memory behind the buffer can be overwritten. That is also true for the memory address that contains the saved return address of the function. If this address can be manipulated, the program will continue its execution at this address after the running function has returned. The return address may now point to an address that is controlled by the user. Usually, this is the address of the buffer that was overwritten. Therefore, machine code is put at the beginning of the buffer. The machine code (or shellcode) can then be executed by the CPU.

Shellcode stages. The user input that fills the buffer is often limited to a defined size. Therefore, shellcode must be as small as possible. Very small shellcodes exist for operating systems like Linux and UNIX. 40 bytes are sufficient to take over such a system [214]. Shellcode for Windows Mobile is bigger for several reasons. All processor instructions in the ARM architecture are four bytes long.

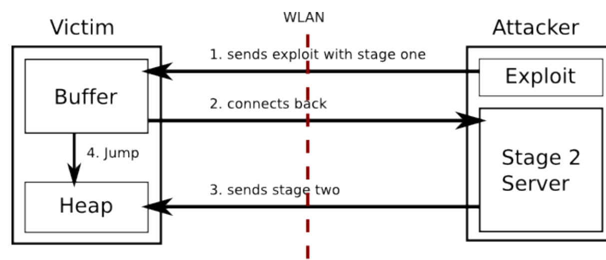


Figure 6.1: Exploitation with Staged Shellcode

Note that in the x86 architecture some simple instructions just need two bytes. Another reason is the load-store architecture used by ARM processors. Memory can only be altered, if it is first transferred to a register, then manipulated, and finally written back to memory. That is why all modifications on memory take at least three instructions.

The interface between user processes and the operating system is the last reason for bigger shellcode. In Linux, a shell interpreter for the execution of arbitrary commands exists. Unfortunately, in Windows Mobile (and other Microsoft systems) the command interpreter is not as powerful and useful as a shell interpreter in Linux [116]. Access to the Windows API is absolutely essential and many instructions are required to use it.

If there is not enough space in the buffer, then the size and complexity of payloads is limited. Because a worm is a complex program, a solution for that problem is needed. The solution is to divide the shellcode in two stages. The first stage contains the smaller part of the shellcode. Its only purpose is to download the second stage from the attacking host, save it to heap memory, and execute it. The second stage is practically not limited in size, and may hold even large worm programs. Figure 6.1 depicts the functioning of the staged exploitation.

Second stage shellcode. Now any payload of worm code can be appended to the second stage. To simplify the development of the worm payload, the high-level language C is used. The result is an executable file (EXE).

The second stage is now separated into an executable starter and the executable payload (the EXE file). On execution of the second stage, the starter saves the payload to the file system of the mobile device. Afterwards, the executable file is executed in a new process by using the system call *CreateProcess*. Finally, the process of the exploited program is killed.

Whenever an executable is started, the operating system validates the signature of the executable. If the executable is not signed or the signature is invalid, the

operating system prompts the user for a confirmation. This behavior is not desired by a smartphone worm, because now the infection depends on user interaction. However, it is possible to circumvent this by manipulating the Windows registry. The registry features a security policy named "Unsigned Prompt Policy" that defines the behavior when loading an unsigned executable.

```
; Unsigned Prompt Policy
[HKEY_LOCAL_MACHINE\Security\Policies\Policies]
    "0000101a"=dword:0
```

The default policy for Pocket PCs is the user prompt (value 0). If the value is changed to 1, the prompt will be disabled. This can be done with the registry functions from the Windows API (*RegOpenKeyEx*, *RegSetValueEx*, and *RegCloseKey*). Note that these functions may be restricted when the process runs in normal execution mode on Smartphones. This is no limitation for Windows CE up to version 5, because all processes run in or can set themselves to privileged execution mode.

6.1.2.2 Spreading

At this point any worm payload can be developed in a high level language and ARM assembler is no longer needed. The focus is now on the spreading functionality of the worm.

The task of the worm's spreading function is finding mobile devices on the network and to start the infection. Our smartphone worm spreads over the wireless network (WLAN) via the IP protocol. The question is, how IP addresses of other mobile devices as possible victims can be found.

The simplest algorithm would be to canonically enumerate all IP addresses of connected networks and start infection for every IP address. Of course, this is a time-consuming task. In a test, the infection of a whole class C network (254 hosts) took about 30 seconds to finish. For a class B network (65534 hosts) the algorithm would take more than two hours. This is a strong limitation for the spreading of a smartphone worm.

Fortunately, the problem can be solved for Windows Mobile because of its specific behavior within data networks. Every Windows Mobile device that connects to local network initially sends a series of NetBIOS packets on UDP port 137 to the network broadcast address. This is used by the devices to notify their participation in the "NetBIOS Name Service". The behavior can be exploited to find out the IP addresses of devices on the network. To do this, it is necessary to listen on UDP port 137 for incoming packets. In Windows Mobile, it is possible to *steal* a port from a listening process [20].

Figure 6.2: Infected Device

6.1.2.3 Assembling the smartphone worm

The worm program is a Windows PE executable for ARM processors. It was developed with the gcc compiler for the target system *arm-wince-pe*. It uses three threads:

1. a main thread,
2. a thread to provide the second stage shellcode (cf. Section 6.1.2.1),
3. a thread for spreading (cf. Section 6.1.2.2).

The main thread does the following:

1. show a confirmation dialog (see Figure 6.2) at startup,
2. use port stealing to be able to receive NetBIOS packets,
3. launch a thread for the providing of the second stage shellcode and thread for spreading.

When the worm program is executed on a Windows Mobile 5 device, it first shows a confirmation dialog and asks the user if it should proceed (see Figure 6.2). If the user confirms, port stealing is used to listen on the NetBIOS port 137. Then the program launches the thread for providing the second stage shellcode. After that, the spreading thread is launched. The spreading thread now waits for incoming NetBIOS packets from victim devices and sends, on event, the exploit with the first stage shellcode back to the victim device. The victim device executes the first stage shellcode and connects back to receive the second stage. Finally, the second stage shellcode is executed on the victim device, and again the confirmation dialog of Figure 6.2 is shown.

6.1.2.4 Conclusion

The research showed that it is possible to develop a smartphone worm (the fourth malware category of Section 4.2.1) for devices running Windows Mobile 5. Infection and spreading work reliably and effectively. The exploited security vulnerability is not a real vulnerability, and does not exist in default software for Windows Mobile devices. The results can be applied to the other malware categories according to user interaction as well. The third category (common behavior) can use the infection part (Section 6.1.2.1), because the staged execution and the shellcode work is useful there. The second category (Trojan horse) can use the spreading part, e.g., to send an installation version of itself to the other device.

The research and development of the worm program was done in a time of about thirteen weeks. About nine weeks were spent for developing and testing the shellcode, the remaining four weeks for the spreading algorithms and for assembling the worm. This is just the constant part needed by smartphone worm. A real security vulnerability is still needed for the worm to become a real threat.

6.1.3 Variable Part: Measuring Resistance

The variable part of an autonomously spreading worm is finding an actual vulnerability of the investigated system. Finding vulnerabilities in an operating system like Windows Mobile 5 can be subdivided into several stages. The focus on a worm restricts the possible attack vectors to interfaces that have connections to other devices, e.g., Bluetooth, Infrared, IP over WLAN or over the mobile network, of which IP over WLAN was chosen. The possible vulnerabilities include buffer overflows, heap overflows, and race conditions. Buffer overflows are chosen here as a vulnerability that was seen as the most dangerous programming error of the recent years [38] and that has successfully been exploited in Windows Mobile 2003 (cf. Section 3.4.1.2). So the focus was on searching for buffer overflows that can be exploited through the WLAN interface. The following steps are illustrated in Figure 6.3.

6.1.3.1 Finding the attack vectors

The first step is port scanning the device (one hour of work). This resulted in five open UDP ports: 68 (DHCP), 137, 138 (both NetBIOS), 1034 ("activesync-notify"), and 2948 (WAP Push). Then, research on the structure of protocol messages is necessary (resulting in 17 hours of work). Some protocols are sufficiently documented, others (like "activesync-notify") lack a documentation, implying that

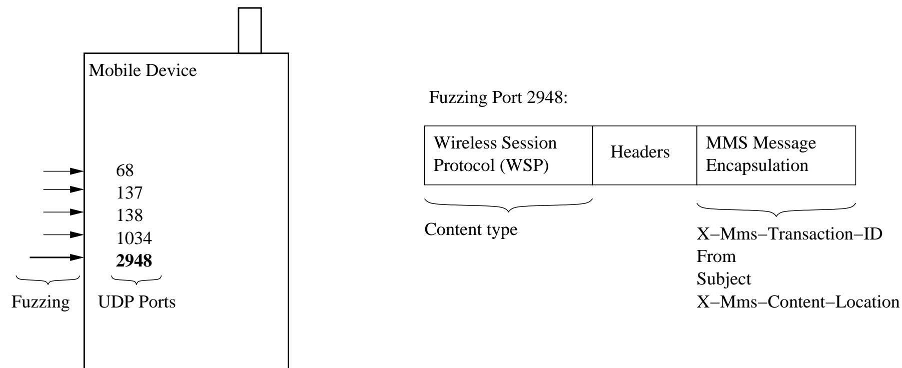


Figure 6.3: Measuring Resistance

the structure can only be found by reverse engineering sniffed protocol messages, or by giving input without any structure at all.

The string variables in the protocol messages are the attack vectors that are used during fuzzing sessions. The last preparation step is incorporating the protocol message structure into the fuzzing framework (work time included in the fuzzing time). Now the actual fuzzing process can take place.

6.1.3.2 Fuzzing

The fuzzing part includes varying the input to the attack vectors (i.e., the strings of protocol messages). Any unusual behavior of the investigated device, especially a crash of the input handling process, can be a sign that a candidate input for a buffer overflow has been found. After fuzzing sessions with each of the attack vectors (21 hours of work), no candidate unusual behavior has been found, but two denial-of-service possibilities and the insight that security cookies are used in the investigated processes (five hours of work).

One denial-of-service possibility was found in the MMS handling process. For inputs in a wrong format, the device displays a warning message that the user must confirm. If these inputs are sent repeatedly to the attacked device, it is rendered unusable. Another denial-of-service possibility was found in the DHCP handling process. When unsolicited DHCP ACK messages are sent to the device in a fast sequence, the reaction times of the user interface decelerate, possibly rendering the device unusable.

6.1.3.3 Results

Searching for a buffer overflow adds another week (exactly 44 hours) of full-time work to the efforts. After that time, searching for buffer overflows in the protocol fields of WLAN visible interfaces of applications in Windows Mobile 5 was terminated.

The search was done with standard techniques for finding buffer overflows. This time was needed to find two possible denial of service attacks and to verify that the system was compiled with security cookies.

As already said, the situation for an attacker changes with the publication of our results. So the measured effort will be invalid for the investigation of Windows Mobile 5 beginning with their publication time. Their value is that they can be used to predict the resistance of other operating systems under the simplifying assumption that all mobile operating systems are largely comparable.

Our results can serve as a basis for further work in Windows Mobile malware research, where it will be—as the results are known now—in the second phase of the three phases model, the standard attack phase.

The next section will now use these results as insights into the current state of mobile device security and propose a novel security solution for mobile devices.

6.2 Attack Surface Reduction in Mobile Device Security

This section shows device-centric parts as necessary for any mobile device security strategy, because some important security requirements can only be implemented on the mobile device itself. It uses the MobileSandbox dynamic software analysis tool of the previous chapter as a basis for a novel security mechanism on mobile devices that aims at reducing the attack surface of mobile devices to mobile malware.

The term *attack surface* was defined by Wing et al. (the most current use in 2009 [126]). It is connected with measuring the attack surface. This section does not measure the attack surface but uses the term in an intuitive way. Alternatively, the term *security boundary* [33] can be used to describe the topic of this section.

This section first justifies why the user of mobile devices should have more control over his own device. Then it lists the foundations for implementing such an interface. Finally, it shows the design of such an interface by using MobileSandbox as a basis.

6.2.1 Justifying Increased User Control

6.2.1.1 Control Requirements

The previous section showed the simplicity with which an attacker can control the mobile device once he has successfully attacked the device. Moreover, Chapter 3 introduced that the increasing functionality of mobile devices—leading to smartphones—brings more and more possible attack vectors to them. An example for these shifting boundaries of common notions is the mobile Web browser, which evolves into a complete application framework (cf. Section 3.4.3). Therefore, it seems necessary to install an umbrella security mechanism on the mobile device, which is reliable to the user, regardless of any possible new functionality of his device. The classical concept of a reference monitor will be used to present the design of such a solution based on our tool MobileSandbox of Chapter 5.

The concept of a *reference monitor* [4] is a useful approach for mobile devices. The main task of a reference monitor is being called every time that a system call is used. Additionally, it must be protected against manipulation and only be a small piece of code for security analyses. Schneider introduces the concept of *security automata* for enforceable security policies as a model for a reference monitor [192]. Reference monitors are also implemented in the Symbian OS operating system [46, 95] and many security problems can be solved with their help. Every system call can now be restricted to a certain policy. Examples are restrictions for messaging, phone and data network functions. Messages and phone calls can be restricted to certain numbers. All of these functions can be limited in numbers per time unit. Reference monitors are an important part of security today (cf. Gollmann [85]) and solutions with them are under active research [23, 102].

As Section 3.5 argued, most of the users are sensible to monetary damage. Therefore, it is reasonable to implement the following two requirements as representatives for the whole set of possible requirements (these requirements were categorized together with other requirements by the MOBIUS project [138]):

1. “It is possible to restrict the amount of data/short messages that a program can transmit per time unit.”
2. “No program sends short messages except the SMS program.”

The implementation of only these two requirements already leads to an attack surface reduction in the security of mobile devices and adds a new entity to the mobile device: the *policy enforcer*. It is visualized in Figure 6.4

Recent work proposes a reference monitor for the Java Platform, Micro Edition (J2ME), an application framework that is present on almost all of today’s mobile

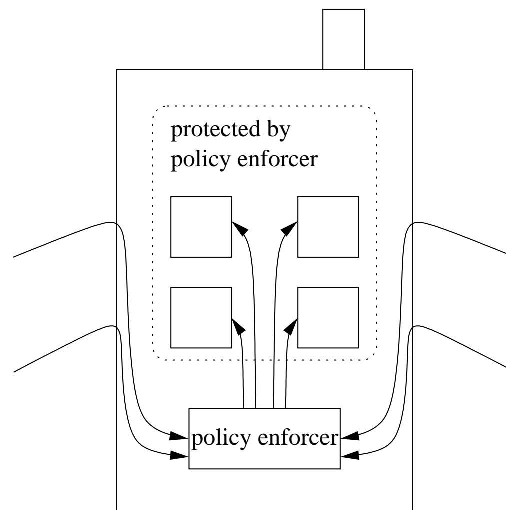


Figure 6.4: Policy Enforcer in the Mobile Device

devices. They implement an extension of an open-source J2ME virtual machine. The extension must be present at compile time of the virtual machine and must afterwards be incorporated into the device [102]. Also, permissions with multiplicities for J2ME in the context of proof-carrying codes have been defined, allowing access to particular system calls for a defined number of times with one single confirmation [23]. Moreover, Schneider has shown that information flow, not being a *safety property,* is not enforceable by the use of reference monitors [192].

Our solution in this section is different in two ways from the J2ME contribution above [102]. First, it is applied at the deeper level of the device's operating system, not on a virtual machine level. Therefore, it can be used for native programs, enabling a broader application. The boundaries are equal to the physical device, which is an important property of usability. Second, it can be applied to existing devices without changing the operating system. That means, we implement the concept of a reference monitor as a flexible solution that can be applied to existing devices as an add-on in contrast to a major change of the operating system.

From the general point of view of this thesis arises the question, why the mobile device itself should be the right starting point. Let us assume the mobile network is the place for implementing the two requirements from above. For requirement #1 it would be possible to collect the amount of already transmitted data within the mobile network. However, the network cannot prevent data transmissions beginning at some threshold, because the transmission has already been performed when the network recognizes it. Thus, resources have already been consumed, even if the network discards the transmitted data. Requirement #2 is even more clear,

because the mobile network cannot distinguish, which process of the mobile device sent a message. Already these two simple requirements show that the mobile device itself has to be part when considering the security of mobile devices.

Additionally, the concept of increasing user control through the use of a policy enforcer can only be introduced in the mobile setting with its strong connection between mobile network operators and the devices. Implementing such a concept for a majority of devices in the common computer world seems not feasible, because it has not happened until now.

6.2.1.2 Firewalls

The common security mechanism firewall together with its evolution over the recent years is introduced now, because this history of firewalls is a good example of the shifting boundaries of security and how security mechanisms adapt to these changes.

Evolution of Firewalls in the Data Network Firewalls have been applied in computer networks for a long time with the intention to separate different networks or network segments from each other.

The first firewalls were *packet filters*. These have a single IP packet as their maximum inspected unit, based on the headers like source and target IP addresses. For protecting servers, firewalls can restrict access to the TCP ports that are intended to be publicly accessible. The limitation to a single packet means that these firewalls are not able to define policies for semantics between different packets. Especially, these firewalls are not able to follow connections (TCP or in some way also UDP, e.g., when stateless UDP packets all belong to a logical connection like multimedia streaming on application layer). *Stateful firewalls* are able to recognize the connection context of single packets and they are able to determine if the packet is related to the current connection state. This state of firewalls was comprehensive for successfully separating networks up to OSI layer 4.

A common setup today is the Web server as the only network entity that is exposed to the public Internet. Fortunately for attackers, Web applications as frontends for the now closed backend systems became increasingly important. So, a publicly accessible Web application uses internal systems (like central database servers) for accessing necessary data. Attackers could now use the clear-text protocols of the World Wide Web, and they did successfully with a plethora of attacks because of poorly written Web applications [99]. Network-level firewalls could not help

in this situation because they only allowed to either close the Web port (usually TCP port 80) or to open it completely. Additionally, many protocols started to be tunneled through the Web port for being able to evade network-level firewalls.

In this situation, the shifted boundaries of attack vectors had to be reflected and *application-level firewalls* (ALF) started to be of importance. They also inspect the traffic on the application layer (OSI layer 7), thus having to understand the application protocols. Even as this leads to more computational complexity of these firewalls, the concept of application-level filtering is an established layer of defense in the data network today.

A common setup for data networks is the separation into a *militarized zone* (MZ) and a *demilitarized zone* (DMZ). As another shifted boundary, this distinction is no longer applicable for application-layer attacks, because the DMZ system usually only forwards the requests. The interested reader is referred to our more thorough explanation of this fact and our more detailed investigation of Web application firewalls [18].

Firewalls for Other Purposes Users are accustomed to the firewall security mechanism. Therefore, it is also used in different contexts where a generalization of the notion firewall is applied implicitly.

Personal firewalls are programs that a user can install on his computer. They fulfill at least the tasks of network-level firewalls with the exception of not being a dedicated device in the network. Personal firewalls are proven to fulfill usability criteria and at least the basic functionality is used correctly by the users [96].

From the definition above, firewall solutions on mobile devices are also personal firewalls. Current incarnations copy the network-level functionality of common computer personal firewalls. These firewalls filter IP traffic in both directions. For example, this was necessary for devices running Windows Mobile (up to version 5.1), because they exposed some UDP ports at all interfaces (GSM, WLAN), leading to vulnerabilities [20]. Also, design and implementation of a simple port blocking firewall was published [97]. In general, mobile devices do not offer network servers so that a firewall for incoming traffic is not necessary.

Current firewall incarnations on mobile devices should be extended to more functionality, for example, filtering message traffic. As with behavior-based virus detection, the limited use cases of the mobile device are advantageous here. See Section 6.2 for this.

6.2.2 Policy Enforcement and Security Interfaces

6.2.2.1 Security Interfaces

The policy enforcer as a new entity in the mobile device adds a new interface for security. This section shows that the implementation is almost not restricted even on currently existing mobile devices. It adds a new line of defense to the mobile device with many possible functions, of which the most useful have to be defined. The most important point is that the policy enforcer is developed independently of the functional evolution of the mobile device and any technology on it.

With regard to the security awareness of mobile devices users (cf. Section 3.5), such a solution with a stable interface promises to be a more effective solution than the security mechanisms of particular mobile device technologies. When reflecting upon the evolution of firewalls from packet-level over stateful and stream-oriented to application-level firewalls (cf. Section 6.2.1.2), we have an example that it is useful to include higher protocol layers for being able to handle more incidents.

Regarding the usability of the solution, the interface of personal firewall is related to this interface here. The effectiveness of this interface was investigated [96] and the propositions of this section have successfully been applied for the successful ZoneAlarm personal firewall, where—besides user-centric and user-driven design—especially the design principle "eliminate complexity" was used [39, Chapter 27].

6.2.2.2 Policy Languages

Currently out of scope is the question of how the security policy finds its way from the specification into our system. At the moment, we simply assume that it is present as a reasonable assumption, because the two following relevant aspects of it are solvable. The first aspect is parsing a configuration text file which is a standard task. The second aspect is the content of the configuration file, the security policy's format. There are several current solutions that are all applicable. They are briefly surveyed afterwards (aspects of usability have already been discussed in Section 3.5.4).

The sophisticated Symbian OS Platform Security Architecture was already mentioned. This is a mechanism at the operating system level. Also already mentioned, Schneider [192] introduces *security automata* as a formalism. Another formalism is the Security Policy Language (SPL) [179]. Other work tries to provide some kind of security certificate together with the application. The first one was proof-carrying code [153] that creates formal proofs for certain security properties. This approach was not easily applicable, so extensions were developed. Model-carrying code [196] and abstraction-carrying code [3] go into the direction of more

human-readable policies. Security-by-contract [47] adds semantics to the signing of mobile code. For completeness: similar systems have been thought about some years ago from an industry perspective [104].

It is an interesting question, why there is such a big gap between research results and real-world implementations. Currently available security policy implementations on mobile devices only enable coarse-grained policies. Especially, there are only few possibilities to restrict data network usage. The current expressiveness is to disallow an access to a certain API completely, to allow it unrestricted (for every run of the program or for the current session), and to ask every time that an API is called. Especially asking every time can be very tedious for the user when the program uses many consecutive calls to a certain API. It would be useful to enhance the expressiveness of security policies to allow at once a certain number of events, e.g., a number `n` of messages or amount `x kilobytes` of data (as already said above).

6.2.2.3 Definiteness of Policy Rules

An important question in the context of policy enforcement is how definite the settings in the policy enforcer are. It is imaginable that certain applications of the mobile network operator or the device manufacturer needs certain restricted functionality, but are sufficiently trusted to bypass the policy enforcer. Technically, this would add a list of processes with superior rights to the policy enforcer, and a checkbox for answering the question of definiteness can be added as well.

It is then a question of the interaction between device manufacturers, mobile network operators, and users of mobile devices, if these choices can be changed by the user and how they are set initially. It could be imaginable that certain operator-signed or manufacturer-signed applications are able to override the settings of the user.

6.2.3 Prototype Design

The implementation of the idea of attack surface reduction is now sketched here, based on the tool MobileSandbox of Chapter 5.

It is important to prevent other programs from switching into kernel mode for ensuring that our reference monitor implementation runs with superior rights compared to other processes except for the operating system processes. Fortunately, there are only a limited number of ways for switching into kernel mode, as shown in Section 2.2.3. The separation between user mode and kernel mode is effective in

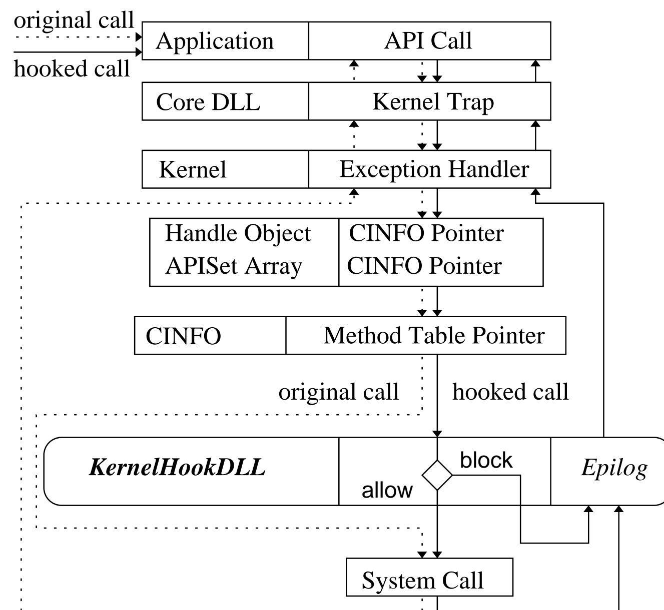


Figure 6.5: Policy Enforcer System Architecture (cf. Figure 5.3)

Windows CE, so the only way to switch into kernel mode is to use a system call. As all system calls are hooked by our solution, we are always able to prevent a program from switching into kernel mode if all ways into kernel mode are intercepted. It can simply return an appropriate error code for an unsuccessful system call. Normal programs do not need to switch into kernel mode, therefore it is no restriction of functionality when disallowing kernel mode.

We use the technique of kernel-level system call interception for accessing the operating system. This well-known technique can easily be used in Windows Mobile, because this operating system implements almost no security mechanisms. This is one of the reasons that our implementation is most needed for this operating system.

Without a loss of generality, we use the example of data network access in the following description. We explained that a kernel-level process is able to intercept all system calls. Of course, this includes any system calls to access the data network and every access to the data network must use a system call. Therefore, our solution will see every access to the data network and is able to apply access restrictions by a security policy.

Figure 6.5 shows the two possible ways for a system call in our implementation. The main decision is made in the *KernelHookDLL*, where a system call either is allowed to be executed or is blocked. When the system call is allowed, we set the

return address to point again to our own system. Then, control is passed back to the operating system. After the system call execution finished, we are able to modify the return value in the Epilog, and then pass control back to the calling application. When the system call is blocked, we jump to the Epilog directly. This poses two questions. First: how exactly can a system call be blocked, so that the calling application does not recognize a difference to a failed system call? And second: how is our system set up, that is, how does our *KernelHookDLL* find its way into the operating system kernel? Both questions will be answered subsequently.

The first question has a simple answer from a conceptual point of view: the system only needs to create a valid fail return condition for every blocked system call. As an example, for handle-based system calls it is a value of -1 most of the times. Processes get the specific error condition usually by calling *GetLastError* after a failure of the system call was indicated. Our system needs to set this value with special care.

The answer to the second question is DLL injection. To put our system in place as a reference monitor, we have chosen to inject our DLL into the *nk.exe* process. As explained in Section 2.2.3, the kernel switches a process into kernel mode before performing the system call, so our code will always be accessed. We show the DLL injection procedure in Section 5.2. In order to inject into *nk.exe*, our system uses the undocumented *PerformCallback4* function that executes code in another process just like in a system call. Therefore, we execute the *LoadLibrary* function in the process of *nk.exe* with a global pointer which points to the name of our kernel-hooking DLL.

For implementing the requirement of only allowing certain processes to call certain system calls, it is necessary to match the calling process. This can be done with the system call *GetCallerProcess* in a reliable way. The system is now able to enforce policies like restricting the sending of messages to the messaging application.

Now, a reference monitor is in place that can be used to effectively enforce security policies. Moreover, this reference monitor implementation is able to perform additional functions. A fundamental problem in the Windows CE world is the lack of any reasonable type of security policy. Other security systems like Symbian OS or J2ME implement concepts like private persistent data spaces for every program, API grouping and several trust levels, with selective access to API groups depending on the trust level. Our system is able to add these concepts to Windows CE. With the same techniques as restricting data network usage, it is possible to restrict access to certain parts of the persistent storage to certain processes and therefore programs.

Additionally, it is possible to restrict the access to cost-generating functions like messaging or phone functions. So, it is useful to globally restrict access to these

functions. For example, the Internet browser on mobile devices is likely to be an increasingly important attack vector with the increasing prevalence of data network usage. However, the Internet browser does not need to access messaging (or phone) functions, and likely does no other program with the notable exception of signed—and therefore trusted—programs. So it is useful to restrict the possibility to send messages to the messaging process and the possibility to initiate phone calls to the phone handler process. All other calls to these functions can either be prevented or prompted to be confirmed by the user. This way, many of the expected problems of upcoming mobile malware can be solved, and our solution is able to accomplish this.

6.3 Conclusion

In the first part of this chapter, we investigated the resistance of Windows Mobile 5 against the threat of an autonomously spreading smartphone worm as the innovative part of the three phases model of breaking a system. Our results are that the constant part can be implemented as a reliable basis. The search for a buffer overflow by using fuzzing was not successful. The entire needed effort for creating the constant part (building block) and the variable part (fuzzing) was approximately 600 hours (fourteen weeks of full-time work, without the time for documenting the results).

In the second part of the chapter we showed device-centric parts as necessary for smartphone security, because some important security requirements can only be implemented on the mobile device itself. We proposed a policy enforcer on mobile devices that aims at reducing the attack surface of mobile devices towards mobile malware. It was developed based on two simple security requirements that already cope with large parts of relevant smartphone security. Finally, we showed a possible design of this policy enforcer.

Chapter 7

Network-Centric Security Investigations

This chapter investigates the possibilities of increasing mobile device security within the mobile network itself. This encompasses all the possibilities that a mobile network operator can provide with its special position for increasing mobile device security. This view is supported by a year 2005 Gartner study that encouraged a different approach for mobile device security: the mobile world should not repeat the mistakes of the personal computer world. Security mechanisms should be installed on network-level first, the mobile devices should be the last resort [115]. Moreover, Schneier proposed that the security industry would shift from having direct contact with the mobile device user to being a supplier for the mobile network operators, "just like new automotive technologies are marketed to automobile manufacturers, rather than individual car owners" [194]. Both views make it worthwhile to investigate the influence that a mobile network operator can have on mobile device security.

As we have discussed in Section 3.5, studies of mobile device users indicate that most users do not see the mobile network operator responsible for protecting their mobile devices. Even though the user acceptance is not given as a prerequisite, this chapter investigates the possible advantages of the more centralized environment of mobile networks, and mobile network operators may be well advised to proactively install these mechanisms in their networks, because they could legally be forced to refund any monetary damage that customers experience because of malware. Indications in this direction for mobile devices are present from the year 2007 [181] and have long been known for banking attacks.

If the mobile network operator is legally forced, it could be justifiable to invade the user's privacy and analyze every piece of installable software that reaches the

user via the network. It could even be required that installable software from local sources like memory cards or a Bluetooth/infrared connection must be sent to the network operator before installing.

The analysis system here uses the mobile network as analysis place rather than the mobile device for two reasons. First, the mobile network has more computing power to perform a more thorough analysis (cf. Section 2.1.3). Only this makes the presented mobile dynamic malware analysis feasible. Second, it is assumed that most software will be delivered via the mobile networks, in part because of the easier handling compared to dealing with local connections.

The contribution of this chapter is the simulation of some of the proposed network-based security mechanisms. With our own simulation system we reproduce some of the results of Ruitenbeek et al. [183] and provide evidence that the protection of a virus scanner helps in reducing spreading velocity, even when only a small percentage of users are protected. As an add-on contribution we document the results of a Bluetooth honeypot case study that confirms the results of similar case studies [30, 204].

Section 7.1 will give a high-level overview of a mobile dynamic malware analysis system, subdivided into the three main parts collecting samples, analyzing samples, and responding to the analysis. Emphasis of the section is on the mechanisms of analyzing the samples, because the effectiveness of these mechanisms is investigated with discrete event simulation in Section 7.2.

7.1 Mobile Dynamic Malware Analysis

Three parts are important for mobile dynamic malware analysis: collecting the software samples as complete as possible (Section 7.1.1), analyzing the samples as complete as possible (Section 7.1.2), and taking certain actions as a response to the analysis (Section 7.1.3).

7.1.1 Collecting Samples

There are various ways, how installable software can reach the mobile device (illustrated in Figure 7.1): messaging, data network, WLAN, local connections via Bluetooth, infrared, memory cards and PC connections. Only few of them are under the control of the network operator (messaging, data network) and are instantly ready to be analyzed. These local software installation vectors are a general drawback of all network-based solutions, that is, a network-based analysis service

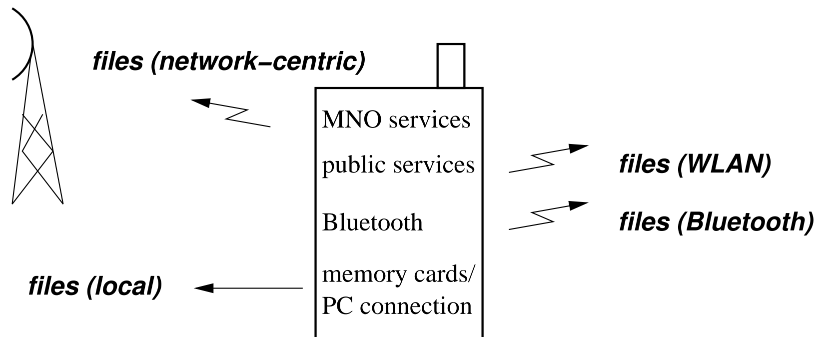


Figure 7.1: File Collection Vectors

could miss a malware sample that has been installed via a local installation vector. Therefore, the part of collecting the possible malicious samples has to employ heuristic methods. Mechanisms for collecting software samples are described subsequently.

7.1.1.1 Backbone Network Scanning

The backbone network of an MNO is the place where all data from the mobile device is available that was transmitted using wireless links of the MNO (mobile network or WLAN hotspots). These are the network-centric files of Figure 7.1.

Messaging Systems. Chapter 4 showed that there are examples for malware spreading via MMS (Commwarrior) and SMS (Yxe). The messaging systems are centralized and under the control of the MNO. Therefore, it is reasonable to collect software there.

IP Systems. Data traffic is increasing steadily. Like the messaging systems, the IP systems are under the control of the MNO and a reasonable place for collecting software. Collecting in IP systems has increased effort, because there is more traffic and more application protocols. However, scanning the application protocols is necessary, because malware can use any of them (e.g., Web traffic to the mobile browser, e-mail) for attacking the mobile device.

7.1.1.2 Collecting Locally Distributed Software

Chapter 4 showed that mobile malware also uses local attack vectors like Bluetooth or memory cards for spreading. An MNO is not able to monitor or to influence this

communication. However, in order to protect the mobile devices of its clients, the MNO must become aware of these samples.

A first solution would be to extend the installer of the mobile device: whenever a software is going to be installed, the installer sends a hash value of the installation package to the mobile network. There are two possibilities: if the hash value is known, the mobile network returns the threat assessment of this software. If the hash value in unknown, the mobile device might send the installation package to the mobile network for an analysis. This would be a straightforward extension of the installer, because the installer is already supposed to implement online certificate status checking. Sending a hash value would only be a minor addition to the functionality of the installer.

However, there are executable files that are executed immediately without using the installer. This would be a task for the executable loader of the mobile device to check the executable against a local list of threat assessments or—if necessary—to add this software to the list with the procedure described above.

7.1.1.3 Bluetooth Honeynets

To be informed about current malware spreading via Bluetooth, implementing a Bluetooth honeynet could be a good idea. Especially crowded areas promise to be a good source of malware [30]. When proactively analyzing samples, the analysis of a particular submitted software might already be finished when the user submits a sample via the extended installer or the executable loader.

Of the local connections, PC connections and memory cards require active involvement of the user. Infrared connections need intervisibility. So all of them can be expected to play only a minor role in mobile malware propagation. Only Bluetooth with a wireless communication range of several meters has the potential of being a major source for malware propagation, even though the mobility of the devices could successfully prevent some infections [235].

We performed a case study of using a Bluetooth honeypot during several weeks in the city of Mannheim. This case study showed no connection attempt of other devices at all, meaning that no mobile malware within our reception area spread via Bluetooth.

For estimating the potential of Bluetooth mobile malware, we measured the available Bluetooth devices at crowded places in the city and in the main station of Mannheim, each for two hours. In these two hours, 151 resp. 170 devices with active Bluetooth were detected. As a rough estimation of the number of active Bluetooth devices, we estimated a Bluetooth transmission range of 10 meters and

Table 7.1: Detected Bluetooth Services (321 devices in total)

Service	*Number of Detection*
OBEX Object Push/OBEX File Transfer	156
Dial-Up Networking	150
Headset/Hands-free Audio Gateway	123

estimated about 500 persons during two hours, leading to roughly 30% resp. 34% of the persons carrying an active Bluetooth device. The absolute number of devices are similar to related studies [30, 204] with the tendency to have slightly more devices with active and visible Bluetooth detected.

Table 7.1 shows the detected Bluetooth services. The reason that only about half of the devices offer any Bluetooth service is the mobility of the devices: they are already out of the reception area before the services could be detected (cf. Carettoni et al. [30]).

As part of the offensive approach of this thesis we also wanted to find out how persons with Bluetooth devices react to receiving unsolicited messages via Bluetooth. At the main station of Mannheim we sent contact information (vCard) to 25 devices, where the name of the contact was the following text (translated from German): "Welcome to the main station of Mannheim". Seven transmission were accepted. The other 18 transmission attempts were either rejected or not recognized in the reception area of our device, this cannot be distinguished. The number of transmission attempts was low and we only sent a contact instead of a software (because of legal considerations). However, the number of successful transmissions indicates a tendency of users towards accepting unsolicited transmissions on the mobile device.

7.1.1.4 Conclusion

The numerous local installation vectors are an argument for solutions like local anti-virus software, but as we have seen it is possible to send the software to the network for analysis before actually installing it. This needs time and will create costs, but it could be necessary to ensure security for mobile device users, because it enables more sophisticated analysis (see next section). And it can be expected that data traffic will be more common in the future or it could be possible not to bill certain traffic in connection with the security system. In order to minimize costs it could be useful to send a hash value of the software first to see if the network already knows this particular software.

7.1.2 Analyzing Samples

The collected samples must now be analyzed. This part of the system can incorporate different analysis modules as its sources, for example, an anti-virus product. Another example of an analysis module is MobileSandbox of Chapter 5.

7.1.2.1 Virus Scanners

One common method to assess the treat of a software is scanning with a virus scanner. Current anti-virus scanners analyze most of the times based on their signature database with all their known drawbacks regarding completeness and susceptibility to small binary changes in the sample (cf. Section 2.3.3).

Even though, virus scanners give a quick threat assessment of a given software. The reaction can be based on that assessment.

7.1.2.2 Dynamic Analysis

The use of formal verification methods could be thought of, but it can be expected that no formal specification for a software is available most of the times. Methods of static analysis could be used, but they are limited (cf. Section 2.4.1).

A promising approach is an automatic dynamic analysis where system calls are logged and afterwards analyzed for malicious behavior. Because of mobile device limitations this cannot be done efficiently on the mobile device itself.

The optimal case is a complete analysis that can be rated against several security profiles like the user's personal security profile. An additional network-wide security profile can exist, e.g., where known malicious behavior like malicious premium-rate services can be stored. However, even the solutions with one run (like MobileSandbox) are able to assess the threat in a certain way.

In the example where messaging functionality is restricted to the messaging program (cf. Section 6.2), every access to messaging functions is a violation of the profile and therefore malicious behavior.

7.1.3 Responding to the Analysis

The mobile network operator can choose from several mechanisms based on the analysis. These are the topic of this section and these mechanisms are explained subsequently.

7.1.3.1 Removing the Malware

A simple and common way is to remove the malware. If it is already installed on the device, then the device must be disinfected. If the malware is still under the control of the MNO (e.g., in its backbone network), the message or packet might be discarded and an explanatory message might be sent to the user.

7.1.3.2 User Interface and Privacy

A problem in the setup of today's mobile networks is the inability to assess the costs of premium-rate services in the operational systems. This is due to the fact that this information is only available in the billing systems. Thus, it is not possible to design an easy prompt for the user, where the device displays all relevant information on the upcoming action, enabling the user to take an informed decision. Unfortunately, this inability also in some way limits the upper monetary limit for transactions like sending messages.

The MNO is able to disallow the installation of the software, send a message to the user that the program violates the security profile of the user or of the network, maybe even as a voice confirmation of the user, depending on how severe the damage is expected to be. From an operator's point of view, the proceeding of asking the user for confirmation and recording his reply could relieve the operator of refund claims.

7.1.3.3 Remote Policy Adaption

In the context of this thesis, we have the policy enforcer of Section 6.2 available as a security entity on the mobile device. This policy enforcer defines many interfaces to security functionality. As an example, it is able to restrict access to the phone functionality, even on the level of single phone numbers.

A security analysis might find out phone numbers that are used in a malicious way. The network-based security mechanism of remote policy adaption can send this security information to the mobile device for informing the user or restricting access to these numbers. Keeping such a system with a global view on malicious players—leading to *malicious phone numbers*—in the mobile network is especially useful because of the inability to assess the costs of premium-rate services prior to their use (cf. above). This procedure is comparable to application revocation, because in both cases the previously available trust into an entity (application or phone number) has to be reverted and this new status has to be conveyed to the mobile device users.

7.2 Simulation

This section measures the spreading velocity of mobile malware with varying the parameters *number of users protected with a network-based virus scanner, update time of virus signatures,* and *user acceptance ratio*.

7.2.1 Model

Before a simulation can take place, a model has to be created from the real-world system that is supposed to be simulated. This modeling consists of the two steps problem definition and a description of the events.

7.2.1.1 Problem Definition

The first step towards a working model is defining the problem. It is here the spreading velocity of mobile malware. The simulation is supposed to answer the following questions: can network-based technical security mechanisms decrease the spreading velocity of mobile malware? Is it necessary to educate a subset of the users? This is answered here for the particular parameters *number of protected users* and *user acceptance ratio*. Another question concerns the necessity for fast signature updates when a previously unknown virus starts spreading.

For answering these questions, the mobile network has to be modeled. The design of our simulation system can be seen in Figure 7.2. It consists of the following modules: virus scanner, messaging servers, users (protected and unprotected), and the two active message sending modules MMS message and virus.

Figure 7.2 shows that all users are connected to the messaging server. Both types of users have a probability of accepting a message. We assume that protected users are more probable to accept messages, because they rely on the network-based security system. Messages for protected users are routed through the virus scanner. This adds a certain delay to message delivery, especially when the virus scanner has no more capacity.

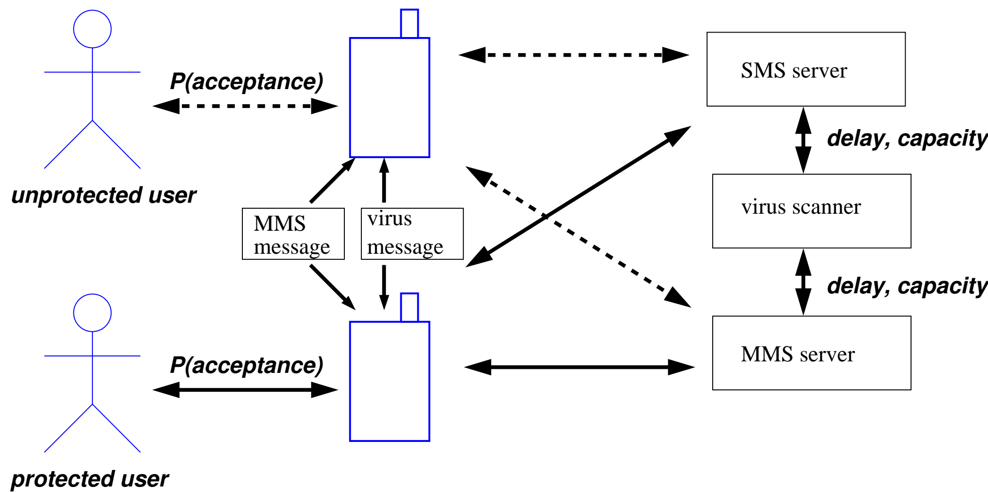


Figure 7.2: Simulation System Design

7.2.1.2 Event Description

Having defined the problem, it is now necessary to structure the elements of the system and to define the relationships among them. Our simulation consists of one MNO that provides the messaging server and the virus scanner. The events of the system are described now.

Messages. The messaging module and the virus module continuously send messages of variable length to the users of the system. They are passing the MNO systems and are delivered to the user. When the message reaches the user, he processes the message according to his behavior.

Mobile Network Operator. The messages pass the systems of the MNO and are routed through the virus scanner if the user subscribed to the network-based virus scanning service. If no malware is detected, the message is delivered to the user. The virus scanner is updated in certain time intervals.

User. The users are subdivided into the *group of protected users* and the *group of unprotected users*. All users have a contact list with a random number of other users. The entries of the contact list are a bijective relation: if user A has user B in his contacts, then user B has user A in his contacts. Users are classified into three classes according to their message acceptance ratio: The *unconcerned users*

accept every message, the *paranoid users* do not accept any message, and the *normal users* have specific personal rules for accepting messages, these rules being abstracted as a probability p in our simulation.

Malware. In case of a successful infection, the malware controls the infected user. Depending on the spreading strategy of the actual malware module, the malware will try to attack other users.

7.2.2 Implementation

7.2.2.1 OMNeT++

For our discrete, event-oriented simulation, we use the tool OMNeT++ [156]. It is often used for simulating protocols in communication networks. A model in OMNeT++ consists of modules that communicate by exchanging messages. Modules are in a hierarchy, with *simples modules* being at the lowest level and being embedded into *compound modules*.

A model is based on the topology description and on the message definitions. The modules can be flexibly extended by embedding C++ language into them. The parameters for the actual simulation run are provided in configuration files. Examples are the simulation duration, the number of mobile devices, the number of protected devices, and the contact list of each mobile device.

7.2.2.2 Entities

Virus. This virus spreads by sending a copy of itself to one random phone number. This is only successful with a probability of 53.3% (derived from the number of connected phones in Germany compared to the maximum possible number of phone numbers) and if unsuccessful, this particular instance of the virus stops spreading. In the following, this virus type has two incarnations that are independent of each other and that are intermittently sent from the outside to devices of our simulated system. They are denoted Virus 1 and Virus 2.

Virus Scanner. The behavior of the virus scanner is dependent on the simulation time. Within the first three simulated days, only Virus 1 is detected. An imagined update at the beginning of day four enables the virus scanner to detect Virus 1 and Virus 2.

The scanning time is set to be between one and three seconds. Moreover, the virus scanner only has a limited capacity of 2500 kbytes per second. If this capacity is exceeded, the scanning process is delayed. The virus scanner module keeps track of several events: overall number of scanned messages, detected malware, undetected malware.

Server. The server module is responsible for routing the message. If a message contains Virus 1 or Virus 2, the message is only processed further with a probability of 53.3% (because of the random target phone number). Afterwards, for a protected recipient the message is forwarded to the virus scanner. For an unprotected recipient, the message is delivered immediately.

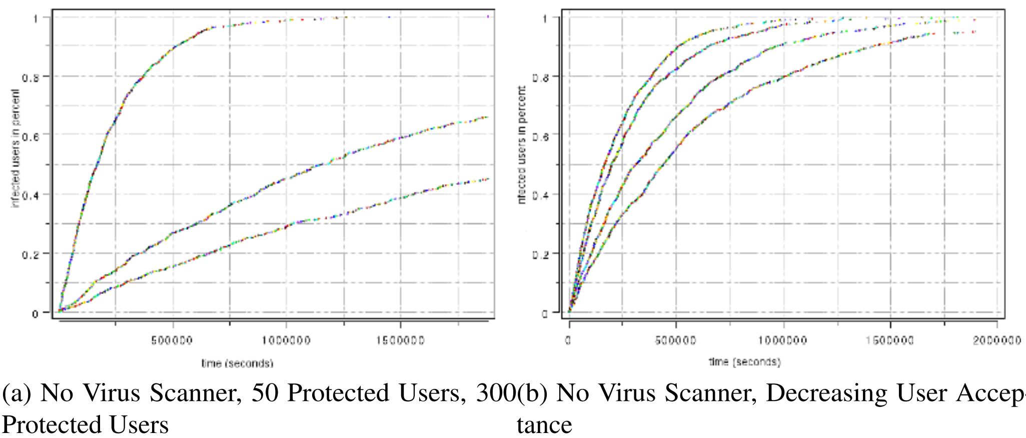
User. The user is modeled to answer to received messages with a Gaussian distribution with a mean of thirty minutes. If a user becomes infected with a malware, the malware takes over the control over this user. That is, it will send messages according to its own spreading mechanism.

A protected user has a higher probability to accept messages and install contained applications than an unprotected user, because the protected user does not expect to receive infected messages. The exact values for these probabilities are varied in the evaluation.

7.2.3 Evaluation

For validating our solution, we take the same parameters as Ruitenbeek et al. [183] and compare their results to ours. The parameters are: 1000 users, 40% acceptance probability for unprotected users, 60% acceptance probability for protected users, percentage of protected phones 5% and 30%. An additional parameter of our system is a variable message size uniformly distributed between 30 kbytes and 300 kbytes. Protected users have a higher probability of accepting a message because they rely on the network-centric security system.

The simulations run for approximately 23 simulated days (the exact value is two million seconds). This proved to be a time interval in which all relevant results can be seen. For rating the simulation results we see the line of 10% infected users as the main indicator for success of the varied parameters. 10% of our 1000 simulated users are 100 infected users. Compared with real-world numbers (e.g., 30 million users for one MNO) 10% are already 3 million infected users. Even if the convergence behavior of these curves it interesting for other purposes, the relevant indicator for our investigation is the 10% threshold.



(a) No Virus Scanner, 50 Protected Users, 300 Protected Users

(b) No Virus Scanner, Decreasing User Acceptance

Figure 7.3: Applying Virus Scanners vs. User Education

7.2.3.1 Introducing Virus Scanners

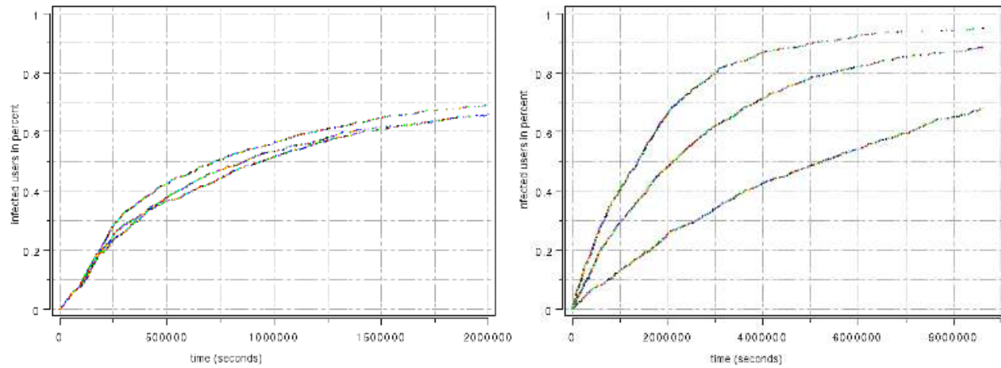
The simulation without any protection leads to the infection of all users after 22 days (topmost curve of Figure 7.3a). This shows that even our virus with polite spreading characteristics infects on the average 45 users per day.

The results of Figure 7.3a are comparable to the referenced paper. They show the decrease in spreading velocity for 50 and for 300 protected users. The curves show that even a small number of protected users (5% for 50 protected users) decreases the overall number of infected users after 22 days. Of special importance is the 10% threshold. It is reached in less than one day for no virus protection. For 5% protected users it is reached after approximately two days and for 30% protected users after approximately three days. This gives more time to every involved entity for reacting to the current malware.

7.2.3.2 Effects of User Education

The effects of introducing a virus scanner (Figure 7.3a) can be compared to the effects of educating the users. This is shown in Figure 7.3b. The topmost curve is the same (no protection) and every curve below has a reduced acceptance probability by 10%.

Figure 7.4 shows the effects with 5% protected users and a acceptance probability that is reduced by 10% for every curve. Figure 7.4a shows a decreasing acceptance probability for protected users only with almost no effects. Figure 7.4b shows decreasing acceptance probability for all users. This shows less effects than the technical security measures.



(a) Only Protected Users Decrease Acceptance Probability

(b) All Users Decrease Acceptance Probability

Figure 7.4: Effects of User Education

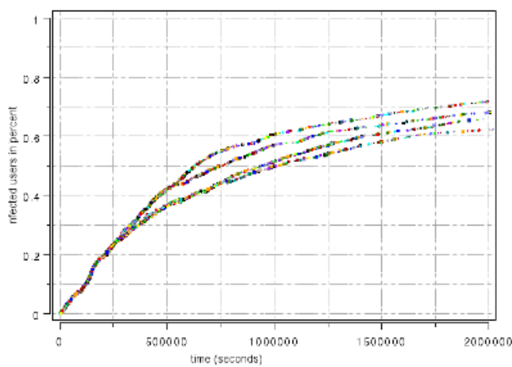


Figure 7.5: Virus Scanner Update Times

7.2.3.3 Update Times

The curves up to now showed the ideal case of a virus scanner: every virus in the system is detected. As a last investigation, we simulate the appearance of a virus that is not in the virus database (Virus 2). It is introduced after one simulated day.

As a final simulation result, Figure 7.5 shows the spreading velocity of our virus while we varied the update times of the virus database: 1/2/4/7 days before the new virus is detected by the system. As we can easily see, there are only minor differences in the curves, especially there are almost no difference at the 10% threshold.

7.3 Conclusion

This chapter investigated the possibilities of increasing mobile device security within the mobile network itself. Section 7.1 gave a high-level overview of a mobile dynamic malware analysis system, subdivided into the three main parts collecting samples, analyzing samples, and responding to the analysis. Emphasis of the section was on the mechanisms of analyzing the samples, because the effectiveness of these mechanisms was investigated with discrete event simulation in the subsequent Section 7.2.

The contribution of this chapter was the simulation of some of the proposed network-based security mechanisms. With our own simulation system we reproduced some of the results of Ruitenbeek et al. [183] and provided evidence that the protection of a virus scanner helps in reducing spreading velocity, even when only a small percentage of users are protected. As an add-on contribution we documented the results of a Bluetooth honeypot case study that confirms the results of similar case studies [30, 204].

Based on our simulation system, further measurements could be done with a new virus variant with a more aggressive spreading behavior: the virus uses the contact list of the infected device for increasing the success probability of choosing a valid phone number. Moreover, the virus spreads more than once per instance. An additional advantage of using the contact list is an increased acceptance probability, as the attacked user might think that this message is a personal message from a known contact.

Chapter 8

Distributed Security Investigations

This chapter investigates a field in-between the device-centric and the network-centric focus of the previous two chapters. Connected to the question of the thesis what the future entities of mobile device security are, this chapter makes use of the MNO smartcard, the trusted module within the mobile phone. This is based on the insight that protecting a mobile device against the influence of mobile malware needs parts in the device that cannot be attacked by mobile malware. As a consequence, this chapter explores the feasibility of using these security modules.

One question related to these modules is their applicability to other tasks than the ones that they already perform. Fortunately, protocols are in place that use the potential of the available security modules for distributed computation problems. This chapter studies the feasibility of of distributed computation protocols with the use of security modules. The first implemented protocol is secure multiparty computation as a modularized version with failure detector and consensus algorithms in the trusted module. The second implemented protocol is fair exchange with guardian angels.

Section 8.2 shows an implementation of secure multiparty computation (SMC) that uses security modules for building a trusted subnet. We selected this protocol because it is the first implementation of SMC on mobile devices with their restricted environment. It extends previous work because it is the first implementation of SMC for more than two participants in an environment that is not synchronous. Our result is that this implementation only requires minimal resources compared to other systems for solving SMC.

Section 8.3 shows an implementation of the probabilistic fair exchange with guardian angels protocol. We selected this protocol because it has never been implemented in an asynchronous setting with special regard to the resource constraints of mobile devices. Our results are that the implementation inherits the

fairness properties of the underlying protocol by being resilient against a large number of attacker classes.

The general applicability of these protocols in mobile real-world scenarios is discussed in Section 8.1. The necessary basic knowledge for this chapter was introduced in Section 2.5, covering definitions of secure multiparty computation and fair exchange together with algorithms and their terminology. This back reference to the early parts of this thesis is intentional because of the clear separation between related work and our own contributions in this thesis.

8.1 Distributed Computation in Real-World Environments

This section discusses the properties of distributed computation when they are applied in real-world environments. This is the mobile network setting and the use of security modules as virtual trusted third parties, both discussed with a bias towards the application of fair exchange.

8.1.1 Mobile Network Setting

The properties of the mobile network setting are important when accessing a trusted third party (TTP). Relevant properties are synchrony in mobile networks and increased time-sensitivity in mobile settings where the location of a device is only temporary.

8.1.1.1 Synchrony in Mobile Networks

The setting of this chapter is a number of participants who communicate in a partially synchronous communication network. The participants are subject to Byzantine (i.e., arbitrary and possibly malicious) behavior. There are two properties of communication networks that are denoted as *centralized* and *open*. Centralized networks are under the control of a single point of responsibility, while open networks are the interconnection of several independent service providers.

Synchronous systems with an upper bound for the message transmission delay are infeasible in most practical settings. It may be possible in centralized networks to achieve a high probability of synchrony by careful monitoring of the network infrastructure and redundant system design. In open networks, where we are facing not only unreliable links possibly with malicious behavior, synchrony is rendered

impossible without assuming a time-out that would be out of the scope of every practical application. Therefore, it is of great importance to select protocols that are built to cope with restrictions of the given infrastructure. Complex protocols and applications may fail in unpredictable ways if their requirements are not met.

Mobile networks belong to the centralized category, but only for the cable-based part. The wireless link is a problem in these networks. Disconnects are encountered regularly, e.g., when switching from one area of wireless coverage to another or when network availability is limited. Moreover, the travel speed of the mobile device may lead to reduced connectivity.

A final factor for the synchrony of mobile networks are the network transitions to other operators. It cannot be assumed in general that the participants use the same (assumable as centralized) mobile network. As a conclusion, even though the centralized environment of mobile networks gives the possibility of them being in the category centralized networks, the wireless link and network transitions make them be seen as open networks in the context of this thesis. In terms of message delivery, they are asynchronous or at most partially synchronous networks.

8.1.1.2 Increased Time-Sensitivity in Mobile Settings

Connected with the mobile setting is an increased time-sensitivity. Time-sensitive items are goods that lose their value over time (e.g., concert tickets, stock market data). It may seem as if many digital items do not belong to this category, thus the delivery delay imposed by a temporarily unavailable TTP would be of minor importance. However, the time-sensitivity may depend on the user's point of view. In protocols with an active trustee, the availability of that trustee is the decisive factor. If the chosen TTP has a high workload, a user may not be able to use its services at the desired time.

Even if the TTP is only needed in the rare case of dispute resolution in optimistic fair exchange protocols, these protocols assume the TTP to be available at least eventually. While failures may occur during protocol execution, the TTP will at some time in the future be able to serve all requests that have previously been issued. While the assumption of an eventually available TTP is perfectly valid for achieving theoretical fairness, the "eventual" part of the observation above may be too long in a mobile setting.

Time-sensitivity is especially true for location-based services (such as route planning or restaurant guides) where the perceived value of the service is connected to the (temporary) location of the mobile device, because the user will likely move on while the delivery takes place. The same may be true for access keys to a local wireless network. Imagine a user who is in a foreign town and who wants to use

his mobile phone for browsing the Web. He may encounter a pay-per-use network service at some place where he orders access for 15 minutes. If the delivery of access credentials takes too long, he may already have moved to a different location. While the credentials will still be valid, the user may never return to the coverage area of the wireless network, thus the item is perceived as useless now.

8.1.2 Virtual Trusted Third Parties

This section first discusses the properties of trusted third parties regarding trustworthiness and availability. Afterwards, their virtualization is discussed.

8.1.2.1 Properties of Trusted Third Parties

Trustworthiness. It is difficult for a user to place trust into a TTP, as he is usually not able to evaluate the set-up and implementation of such a service. Therefore, trust actually has to be placed into either the manufacturer/provider of the arbiter service or an external evaluation authority. While the latter is obviously more trustworthy, it is still infeasible to assume perfect trustworthiness. This is less due to inaccuracy in the verification of the machine's technical specification but more to missing or uncertain knowledge about possible connections between such service providers and untrusted participants in an exchange. Even if the trusted third party is an important factor of fair exchange protocols, a formal analysis of the requirements to be met by the TTP is not available.

Availability. It is impossible to construct a fair exchange protocol without the help of a trusted third party (Even and Yacobi [54], cf. Section 2.5). Literature has since focused on reducing the necessary involvement by the TTP in order to guarantee fairness. This direction of research is advantageous also regarding the specifics of the mobile network setting introduced above. When standard protocols make use of an active TTP on a regular basis, they run the risk that the TTP becomes a bottleneck. When considering the operation of a TTP in a real-world—possibly mobile—scenario, it is reasonable that one TTP will be responsible for supporting numerous exchanges simultaneously. This places a high burden on the system, both in terms of communication as well as computation demands. As an additional remark, this central TTP can be subject to denial-of-service attacks.

A major improvement has been achieved by proposing *optimistic protocols*. Those solutions rely on the observation that most exchanges are executed without malicious intent and terminate successfully. For the rare case of failure, a TTP is

available to aid the conflicting parties in dispute resolution. While the optimistic approach proves to be much better suited for actual deployment, all solutions based on external third parties need to address serious challenges. Any entity willing to offer such services will have to be recognized by all participants as trustworthy. Up to now, these aspects have been mostly ignored in literature.

8.1.2.2 Relaxing the Trustworthiness Requirement

A way to increase the level of trust between the participants in an exchange and to simplify and improve fair exchange protocols is to utilize trusted hardware as security modules. Chapter 2 showed that the MNO smartcard of today's mobile devices can serve as a basic security module, with possible extensions by using more sophisticated smartcards.

Such hardware-based solutions offer multiple advantages: placing a smartcard close to the participant makes the communication between those two entities more trustworthy. It can be assumed that the security module is locally attached to the user's device and that the communication takes place fast and synchronously.

The security module may also be able to perform part of the duties that would usually fall to the central TTP. This lowers the risk of the trustee forming a bottleneck, since the load on that server may be significantly reduced. The danger of the local security device being overloaded is minimal: each participant has access to its own device and the workload is restricted to his own computations.This is an advantage compared to a central TTP that has to handle potentially hundreds of participants simultaneously.

It also has to be noted that an adversary trying to disturb the exchange may find less potential attack vectors when secure hardware is used. This is mainly due to the fact that many critical operations take place within a well-defined environment with restricted access possibilities.

8.1.2.3 Benefits and Disadvantages of Utilizing Trusted Hardware

Optimistic fair exchange protocols are able to guarantee deterministic strong fairness if at least one of the exchanged items is either revocable or generatable. Fair exchange protocols using security modules achieve probabilistic strong fairness for any digital item. Even though we exchange the deterministic strong fairness with a probability, this is a major step forward for the applicability of fair exchange protocols, because digital items are only rarely revocable and only sometimes generatable (e.g., digital signatures). A fair exchange framework offering protocols

that guarantee strong fairness without regard to the goods exchanged opens up more possibilities to use this technique in practice. Furthermore, trusted hardware enables timely delivery even when parts of the network infrastructure, e.g., the connection to the central TTP, are unreliable.

Hardware-based protocols have been proposed that ensure this important property. A locally equipped smartcard may even be able to deliver an item although the item to be delivered in return has not been sent yet. Literature investigates solutions such as incentive-based fair exchange (Vogt et al. [222]). The basic idea here is to reward fair behavior and penalize efforts to cheat. For example, the smartcard may be locked while an exchange has not been completed successfully, thus making it useless for its possessor.

Another interesting property of solutions based on security modules is the introduction of anonymous exchanges. It may be desirable in certain settings for a participant to not expose his identity to an outside entity. This is not possible in classical fair exchange protocols as soon as dispute resolution is needed. Secure hardware can be a solution for this problem, because the customer may reliably use its security module as an arbiter that deals with the vendor. As the smartcard is expected to work as specified, the vendor may trust that a request to resolve a dispute or to cancel a contract has indeed originated at the customer. Still, he has no way to learn of that person's true identity.

As a final advantage, the participants do no longer need to trust an external entity with unknown and unobservable relationships to other—possibly malicious—entities. Even though both TTP settings require trust in the producer of the hardware and the provider of the software of the TTP, the behavior of a local TTP is more observable than the behavior of an external TTP.

A drawback of applying security modules is their limited computational power and memory. For important applications, such hardware seems to be well suited, as the rest of this chapter will show. Research already investigates how to overcome performance constraints by securely outsourcing parts of the smartcard's workload onto its host computer [222]. An additional possibility is the virtualization of the trusted module. These results of trusted computing research use the hardware trusted module for providing a software trusted module that has access to the full capabilities of the host device while at the same time being as inaccessible as its hardware counterpart [190]. A successful virtualization of a security module would relieve most of the restrictions of Section 8.2.3.

As a conclusion, even by using security modules it is not possible to deliver deterministic strong fairness. The impossibility result by Even and Yacobi still holds and third parties are needed to achieve such requirements. Though, the notion of probabilistic fairness as introduced in the fair exchange with guardian angels

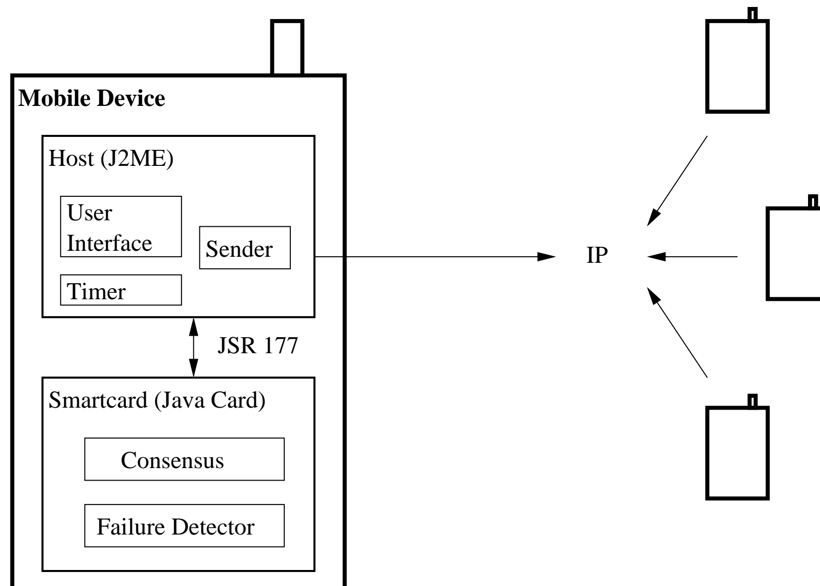


Figure 8.1: System Architecture of SMC Solution

and gracefully degrading fair exchange protocols are a promising approach for real-world application scenarios. While it is more than desirable for fair exchange systems to be able to accomplish as much dispute resolution as possible, absolute fairness, at least in a legal sense, will always only be possible by calling upon a legal entity.

8.2 Secure Multiparty Computation with Security Modules

Section 2.5 introduced a solution to solving SMC with a consensus and a failure detector. This section will show an implementation of this algorithm for mobile devices with MNO smartcard, using the technologies J2ME and Java Card. As also said in Section 2.5, we implement a *weak SMC*, because we have a *weak failure detector*.

8.2.1 Design

The design of TrustedPals is subdivided into the untrusted host system and the trusted smartcard system. Figure 8.1 shows the architecture of the developed system.

8.2.1.1 Host Application

The host application is responsible for the communication between the participants, for the user interface, and for providing a clock signal to the smartcard, as the smartcard does not have access to a clock signal.

After the user confirmed his input value, the participants synchronize and start the consensus. Upon successful completion, the result is given to the user.

8.2.1.2 Smartcard Application

With the consensus and the failure detector, the most important parts of the SMC algorithm reside on the smartcard. Both parts were introduced in Section 2.5.

The failure detector is the first implementation of the algorithm that was proposed by Cortiñas et al. [37]. It continuously sends and receives failure detector matrices and builds a process liveness table from the received matrices. The consensus with its four phases will be used as proposed. For computing the common function F, consensus is executed once for the input value of each participant.

An important security feature of the algorithm is attaching the consensus messages to the recurrently sent failure detector messages. If no consensus message has to be sent, the consensus part of the failure detector message is filled with random content. As the failure detector messages are sent in encrypted form, outside attackers cannot learn the behavior of the consensus algorithm.

Triggered by the clock signal, a failure detector sends its state messages to the other failure detectors. Thus, the observable communication has fixed time intervals when assuming a regular clock signal. Additionally, messages only leave the trusted smartcard in an encrypted form. Unencrypted sensitive data is only available in the secure memory of the smartcard.

8.2.2 Implementation

8.2.2.1 APDU Communication

Communication with the smartcard is done with APDUs. The most important command APDUs are for starting the SMC, for sending messages, and for sending clock signals to the smartcard. They are illustrated in Figure 8.2.

Start APDU. The Start APDU has the instruction byte 0x30 and carries the user input from the host to the smartcard. The successful start of the SMC will be confirmed with a response of value of 1.

Start APDU:	CLA	0x30	0	0	1	estimate	1
Message APDU:	CLA	0x40	0	0	ML	message	-
Clock APDU:	CLA	0x50	0	0	-	-	-

Figure 8.2: Application Protocol Data Units of SMC Solution

Message APDU. Incoming messages from other participants are forwarded with the message APDU to the smartcard. The instruction byte is 0x40 and the message length (ML) is dependent on the number of participants. No response is expected, so there is no Le byte that would indicate the length of the response APDU.

Clock APDU. The clock APDU is responsible for triggering the step counter of the smartcard part. As there is no additional data and no expected response, no Le and no data bytes are part of this APDU.

8.2.2.2 Host Application

The implementation of the host application parts is straightforward. The user interface is simple, because its only functionality is retrieving the desired input value from the user, starting the SMC, and displaying the result.

The timer recurrently sends the clock APDUs to the smartcard and the sender is responsible for scrambling the packets and sending them to the other participants.

8.2.2.3 Smartcard Application

General. Implementations of the Java Card application on real devices can be done with the Nokia Series 40 5th edition, because it implements the APDU part of the JSR 177 specification. These devices can access the Java Card functionality of the MNO smartcard. Another possibility is a special Java Card hardware module. Such a module exists at the moment in Nokia phones with Near Field Communication (NFC) technology (e.g., Nokia 6121).

The implementation uses the cryptographic methods that Java Card provides in the javacard.security package. Available are asymmetric encryption schemes and key agreement for symmetric encryption schemes.

1	1	1	$\lceil \frac{n^2}{8} \rceil$	n	1	1	1	1
sender	receiver	time	failure detector matrix	failure detector version	round	estimate (consensus)	age	estimate round (consensus)

Figure 8.3: Content of Encrypted Messages (Size in Bytes)

Consensus. The consensus algorithm is the main part for distributing the input values of the participating processes. As said in Section 8.2, it is executed once for each participating process and messages are embedded into a continuous message stream (the *estimate* parts of Figure 8.3).

Failure Detector. The failure detector holds the following data: lists of in- and out-connected processes, time-out counters for each process, the time stamp for the next expected message of each process, its own step counter, as well as various flags for newer versions of the failure detector matrix. A new failure detector message from another process is processed by comparing the own failure detector matrix with the received matrix. Differences are adopted when they have a larger version number than the own failure detector matrix.

Then, the failure detector matrix is raised to the power of the number of participating processes. The failure detector matrix is the adjacency matrix of a directed graph. Entry (p,q) equals 1 iff an edge from node p to node q exists. The exponential matrix informs about the connectedness of the processes. A process is in-connected, if it is able to receive messages from at least $\lceil \frac{(n+1)}{2} \rceil$[3] processes. Therefore, a process p is in-connected, if the exponential matrix has in row p at least $\lceil \frac{(n+1)}{2} \rceil$ non-zero entries.

Message Box. Figure 8.3 shows the message that is created and sent when a clock signal is received from the host. The messages for the other processes are concatenated and sent to the host in a single response APDU. The total length of this message is $(9+n+\lceil \frac{n^2}{8} \rceil) \cdot (n-1)$[6] bytes for n participating processes. With a maximum APDU length of 256 bytes, this leads to a number of 9 participating processes possible in one APDU. A solution can be extra-long APDUs that are sent with offsets, but when assuming the size of 1 kilobyte overall memory of the smartcard more participating processes are prevented anyhow.

Table 8.1: Memory Requirements of Failure Detector

Attribute	*Memory Requirements (in Bytes)*	
	n Participants	*9 Participants*
Matrix	$\lceil \frac{n^2}{8} \rceil$	9
Exponential Matrix	$\lceil \frac{n^2}{8} \rceil$	9
In- and Out-Connected Processes	$2 \cdot \lceil \frac{n^2}{8} \rceil$	4
Version List	n	9
Constant Overhead	20	20
Total	$4 \cdot \lceil \frac{n^2}{8} \rceil + n + 20$	51

Table 8.2: Approximated Memory Requirements Java Card (9 Participants)

Function	*Memory Requirements (in Bytes)*
FD and Consensus	80
Messages	250
Other	50
Total	380

8.2.3 Memory Requirements Evaluation

The memory size of currently existing smartcards usually is 1 kilobyte, sometimes up to 2 kilobytes and in rare cases even more. These values show that smartcard-based applications need to respect the low available memory. With the given number of 9 participants almost a quarter of the available memory is filled with the buffer (256 bytes). Incoming messages have a size of $9 + \lceil \frac{n^2}{8} \rceil + (n-1)$ bytes (equals 28 bytes for nine participating processes). As such a message can be processed directly, the memory buffer is available again immediately after processing the buffer and is only needed once.

The failure detector needs memory for its matrix, for the exponential matrix, for the version list, and for the lists of in- and out-connected processes. Table 8.1 shows the memory requirements for n participating processes and for the fixed number of 9 processes. There is an overhead of 20 bytes that is independent of n.

Table 8.2 show the complete memory requirements for the smartcard part. The consensus needs 10 bytes statically and two lists of length n. This totals at least 380 bytes memory for 9 participating processes and leaves approximately 600 bytes as buffer for incoming messages. With the numbers from above this buffer can store 21 messages, or approximately three messages per participating process.

Intuitively, these numbers seem to be too low for robustness of the algorithm, and attack scenarios can be constructed to let this buffer overflow. An example is a Byzantine participant that increases the clock signal of its smartcard, leading to an increased number of failure detector messages with future round numbers. When the other participants receive these messages, they have to buffer them.

Worthwhile future work for our implementation is investigating how these theoretical values influence the robustness of the multiparty computation in the presence of real attackers.

8.3 Fair Exchange with Security Modules

This section describes our implementation of the probabilistic fair exchange with guardian angels protocol, subdivided into design, implementation, and evaluation.

8.3.1 Design

This design part first describes how the fair exchange with guardian angels protocol is mapped to our solution. Then it introduces the design of the host system part and of the smartcard application part.

8.3.1.1 General Mapping of Fair Exchange Algorithm

The fair exchange with guardian angels protocol is mapped to the three phases key agreement, item exchange, and keep-in-touch protocol.

Phase 1: Diffie-Hellman Key Agreement. Symmetric cryptographic techniques are used to encrypt the data exchanged between the security modules. Since it is infeasible to assume that all security modules share a single common secret key, a symmetric key is generated using the Diffie-Hellman key agreement scheme. To perform the key agreement, both participants need access to the other's public Diffie-Hellman key. Therefore, A first sends its public key to B, who answers with his own public key. When both parties are in possession of the remote public key, they can start the agreement and end up with the same common secret without further communication. This common secret is then used to encrypt all subsequent communication between the two participants.

Phase 2: Item Exchange. Once that *A* has computed the secret key, *A* starts the second phase of the fair exchange protocol. A message object is constructed to hold the plain text that *A* wants to exchange for a non-repudiation of receipt (NRR) token. This item is then sent to *B*, who verifies that it matches the hash value proposed during exchange initiation. *B* in turn delivers his NRR token to the smartcard of *A*, thus completing the item exchange.

Phase 3: The Keep-in-Touch Protocol. The third and final phase implements the keep-in-touch (KiT) protocol proposed by Avoine and Vaudenay. *A* starts this phase by sending a fake message to *B*. This fake message includes no explicit contents except for a number *k*, which has been chosen randomly from a range of 3 to *MaxFakeRounds* (one of the global parameters). For testing purposes, we set this parameter to 10.

When receiving such an initialization message for the KiT protocol, *B* saves the total number of rounds proposed and sends a new fake message back to *A*. The fake messages include a round count—namely “two” if it is the first message sent by *B*—which can be checked by the recipient. If the round count included does not match the state of the smartcard’s internal counter, the exchange is aborted. As long as the count does match, *A* and *B* keep sending such messages back and forth. When this message exchange has lasted for the saved number of total rounds, both *A* and *B* can be sure that at least probabilistic fairness has been achieved. The internal exchange object is updated to reflect the successful outcome and the smartcard signals the success to its host application. The host may then choose to have the item delivered from the smartcard’s internal memory.

8.3.1.2 Host System Application

Figure 8.4 shows the design of the fair exchange system. The abstract base classes *Exchange*, *FXSmartCard*, and *FXSmartCardMessage* define functionality of any fair exchange implementation. These abstract interface definitions include an object model in *Exchange*, a smartcard definition in *FXSmartCard*, and a message format for the communication between the host and the security module in *FXSmartCardMessage*. All concrete implementations of fair exchange solutions should be derived from this set of interface definitions and extend their functionality where needed.

The host uses the *PacketCollector* interface to listen for messages that carry a *GAFXSmartCardMessage* payload. When such a message is encountered, its (encrypted) content is recovered from the XML representation and the smartcard’s method to handle the message is called. The smartcard method verifies the origin

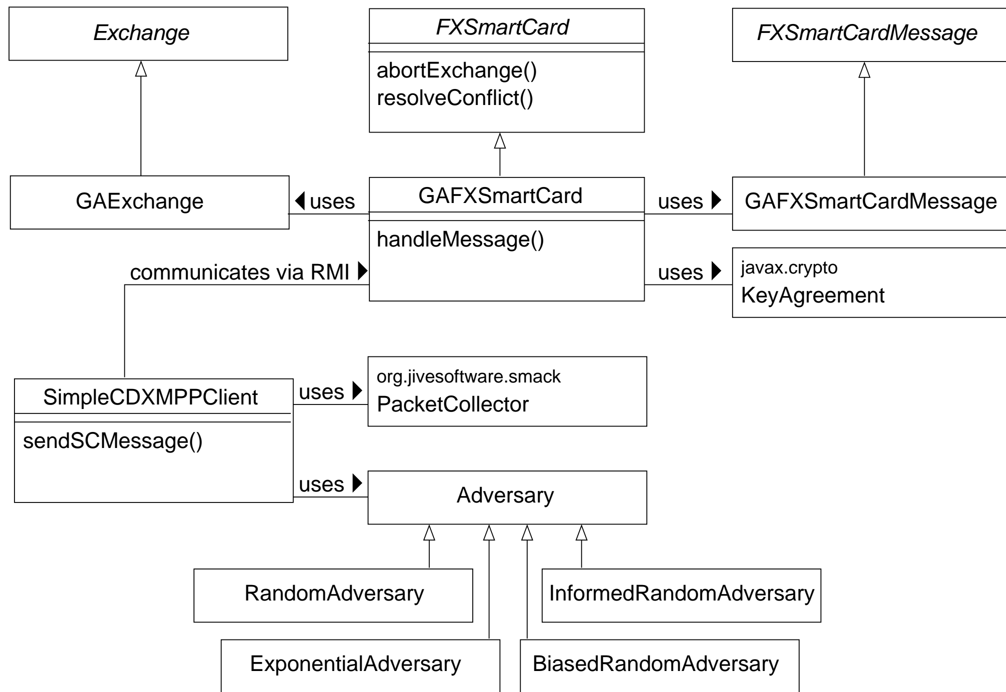


Figure 8.4: Design of Fair Exchange Solution

of the message and processes it depending on the current phase of the protocol execution. If a message does not pass the verification or if it belongs to a protocol phase other than the one currently in execution, it will be dropped and not regarded any further. This is a simple security mechanism to prevent message spoofing and deliberate attacks on the protocol run.

8.3.1.3 Smartcard Application

Our solution only simulates the smartcard in an object. For communicating with the simulated smartcard it uses method calls. When using a real smartcard, this would be equivalent to using the sophisticated remote method invocation (RMI) services instead of the APDU communication in the implementation of secure multiparty computation in Section 8.2. RMI abstracts from the APDU message format and provides an environment for accessing the Java card that is more adapted to the concepts of the Java language.

8.3.2 Implementation

8.3.2.1 Host System

Our implementation uses the extensible message and presence protocol (XMPP [184]). This XML-based protocol is mainly used for the Jabber instant messaging clients, but it has been defined as a general technology for communication-related tasks. Therefore, it is a reasonable choice as basis for the implementation. The Jabber client application is implemented in the class *SimpleCDXMPPClient*. It takes care of XMPP connection management and basic Jabber chat functionality as well as the management of the smartcard device and delivering messages to and from the smartcard.

When its *sendSCMessage* method is called, the specified payload is attached to a custom XMPP message and delivered to the receiver's XMPP client. When a custom smartcard XMPP message is received by the client, its payload is extracted from the XML envelope and passed on to the module's *handleMessage* method implementation. The custom payload is encrypted and includes status and identity information about the exchange as well as additional data depending on the current protocol phase (e.g., public keys, digital items, dummy payloads). It is stored in a *GAFXSmartCardMessage* object that can be serialized to and deserialized from a byte array representation using Java's Serialization API and then be transported as part of an ASCII encoded XML message.

Communication between the parties in all three phases of the exchange uses the blocking *PacketCollector* mechanism while awaiting an answer. This is necessary to implement the time-out functionality that our solution uses to detect process failures. This blocking of the user interface is acceptable in our proof-of-concept environment. In production environments it is desirable to restrict operations that block the user interface. For adding non-blocking behavior to our solution, it is possible to implement the failure detector in a thread of its own. The user interface would stay responsive and communicate with the failure detector thread via signaling mechanisms.

8.3.2.2 Smartcard

Our smartcard simulation provides a number of publicly invokable methods. The methods address the provision of general status information concerning both the smartcard itself as well as fair exchanges currently being executed, but they also provide means to actively modify the exchange execution, e.g., through the methods *abortExchange* and *resolveConflict*.

8.3.3 Evaluation

8.3.3.1 Fairness Analysis

Adversaries. To test the fairness provisions of our prototype, we modeled different attackers and included them in our system. Attackers are able to compromise the communication channels or the host computers in such a way that they can arbitrarily disrupt such communication and drop any messages at their own choosing (Byzantine behavior). They may not, however, make an informed decision whether to drop a message based on its contents, since this would violate the confidentiality assumption of secure channels.

Our attackers are included in the adversaries package and follow different strategies when deciding whether to drop a given message or not.

- The *RandomAdversary* uses a random number generator and drops messages with a probability of 50%.
- The *ExponentialAdversary* keeps track of the number of messages received and drops all messages that have a count of 2^n.
- The *BiasedRandomAdversary* may be initialized with a custom threshold value in order to change the probability that messages are dropped.

Other strategies, especially combinations of those described here, may be interesting to investigate. However, performing one hundred test runs, we were not able to reach a fairness breach even once. Whenever a message was dropped by an adversary, the expecting party would reach its timeout limit and abort the exchange. The remote party would also time out as expected.

There is a more sophisticated attack scenario that will have to be investigated further. Having knowledge of the protocol execution, an attacker will know that there are exactly four messages exchanged before the keep-in-touch protocol starts (not counting the exchange proposal). An attacker may also know the maximum number of fake rounds that a smartcard will support, and make a partly informed decision about dropping messages based on this knowledge. So while in theory, choosing the parameter k of fake messages may render the chance to successfully attack the protocol arbitrarily low, it has to be considered that a practical implementation of this protocol will always have to make a trade-off between security and applicability. In a real-world network setting it is infeasible to assume the exchange of more than a few fake messages, and an attacker may be able to deduce the range inside which that parameter can be found by carefully observing the protocol executions.

Table 8.3: Fair Exchange Adversaries

Name	*Knowledge*	*Strategy*
RandomAdversary	none	drop randomly
BiasedRandomAdversary	none	drop randomly
ExponentialAdversary	none	counting
InformedRandomAdversary	KiT interval	drop random KiT message

Table 8.4: Fairness Evaluation Using InformedRandomAdversary

max_k	10	20	30	40	50	60	70	80	90	100
Fairness Breaches	8	4	4	2	3	1	1	1	3	0
Theoretical Values	10	4	3.3	2.5	2	1.7	1.4	1.3	1.1	1

We have implemented such an attacker in the *InformedRandomAdversary* class. This implementation provides a constructor method which is called with a lower and a upper bound on the expected number of keep-in-touch messages. The adversary will then randomly choose a specific message to be dropped from that interval and increase an internal message counter each time a message is sent or received. When the counter equals the pre-determined message number, that message will be dropped. All other messages will be passed on. An overview of all modeled adversaries is given in Table 8.3.

The described informed attacker should succeed in a fairness breach if and only if the randomly chosen message is the last message in the keep-in-touch protocol phase. If the attacker drops messages too early, the exchange should abort for both parties. If the attacker drops them too late, the protocol will complete successfully before any message is dropped. After each exchange, the adversary's internal message counter is reset to zero.

Fairness Breaches. We varied the maximum number of KiT rounds $\max_k$ and performed one hundred test runs each. The obtained results are shown in Table 8.4. The theoretical breach rate for each unit of measurement is $\frac{100}{\max_k}\%$. Although this cannot be exactly verified with the results at hand, there seems to be a distinct trend towards lower breach rates for higher values of $\max_k$. Therefore, we conclude that our implementation indeed inherits the probabilistic fairness results of the guardian angels protocol.

There is one more potential security issue to be noted here. Right now, the receiver accepts any number of KiT rounds k that is proposed by the originator's smartcard. If that device only offers a rather weak implementation of the guardian angels

Table 8.5: Average Execution Times Depending on Cryptography (in milliseconds)

max_k	10	20	30	40	50	60	70	80	90	100
Crypto On	120	126	131	140	144	156	157	172	182	189
Crypto Off	32	35	45	45	60	55	74	75	83	91

protocol, the receiving party might unknowingly suffer from insufficient ranges for k. It should therefore be allowed to reject certain values. In our opinion, the best approach would be to agree on an allowed range during some kind of feature negotiation phase prior to starting the actual exchange.

8.3.3.2 Performance Analysis

The Diffie-Hellman key agreement is one of the most complex operations performed during an exchange. Therefore, we wanted to identify the impact of the cryptographic primitives on execution speed. So we removed all cryptographic operations (key agreement, message encryption and decryption) from our implementation and measured the performance. The results for a local setup with negligible communication times are shown in Table 8.5.

8.4 Conclusion

Based on the insight that increasing the security of a mobile device by protecting it against the influence of mobile malware needs parts that cannot be attacked by mobile malware, this chapter explored the feasibility of using trusted modules in mobile devices for advanced distributed computation problems. It showed the general applicability of these protocols in mobile real-world scenarios. Afterwards, it presented prototypic implementations of two distributed computation protocols: secure multiparty computation with security modules and fair exchange with guardian angels.

The implementation of secure multiparty computation with security modules is the first implementation of SMC for more than two participants in an environment that is not synchronous. We showed that this implementation only needs minimal resources compared to other systems for solving SMC.

Avoine and Vaudenay's fair exchange protocol with guardian angels was the basis of our implementation for a two-party fair exchange scenario as certified message delivery for the Extensible Messaging and Presence Protocol (XMPP). We were able to show that the implementation inherits the fairness properties provided by

the underlying fair exchange protocol. It successfully withstood different adversary models which behaved according to the attack model laid out for the study of hardware-based protocols. We also showed how we were able to meet the desired security requirements of our communication infrastructure using cryptographic techniques.

Chapter 9

Conclusion

9.1 Summary

The importance of researching on the smartphone security topic is substantiated in the increasing number of smartphones, which are expected to outnumber common computers in the future. Despite their increasing importance, it is unclear today if mobile malware will play the same role for mobile devices as for common computers today. Therefore, this thesis contributed to defining and structuring the topic mobile device security with special concern on smartphones and on the *operational side* of security, i.e., with mobile malware as the main attacker model.

The first three chapters introduced and structured the topic with the main goal of showing what has to be defended against today. First, an attack model with four attack classes that argues for a clear distinction between its classes. Second, a survey on the current state of real-world examples and a projection of the potential of mobile malicious software, together with a classification concerning portability between different platforms with the main goal of showing what has to be defended against today.

We subdivided the technical investigations of this thesis into the three parts device-centric security mechanisms, network-centric security mechanisms, and something in-between: distributed security mechanisms.

The device-centric investigations contributed MobileSandbox, a software for dynamic malware analysis of Windows Mobile binaries with the following advantages over existing solutions: first, it logs system calls not only at user level but even at the level of the kernel, enabling a more detailed system call log. Second, it can be integrated into a running device without any changes to the firmware of the device.

As additional device-centric investigations we contributed on the efforts that have to be taken to develop an autonomously spreading smartphone worm for Windows Mobile by actively researching characteristics and countermeasures to learn more about its associated threats. Moreover, we showed that device-centric parts are necessary for smartphone security, because some important security requirements can only be implemented on the mobile device itself. We proposed a policy enforcer as a security mechanism that aims at reducing the attack surface of mobile devices to mobile malware. This security mechanism differs from recent related work in the area by relieving the need to be added to the device at manufacturing time.

The network-centric investigations showed the possibilities that a mobile network operator can use in its own mobile network for protecting the mobile devices of its clients. We simulated the effectiveness of different security mechanisms and came to the conclusion that the protection of a virus scanner helps in reducing spreading velocity, even when only a small percentage of users are protected.

Finally, the distributed investigations showed the feasibility of distributed computation algorithms with security modules by giving prototypic implementations of protocols for secure multiparty computation as a modularized version with failure detector and consensus algorithms in the trusted module and for fair exchange with guardian angels. The implementation of secure multiparty computation is the first for more than two participants in an environment that is not synchronous. We showed that this implementation only needs minimal resources compared to other systems for solving SMC. The implementation of the probabilistic fair exchange with guardian angels protocol was shown to inherit the fairness properties of the underlying protocol by being resilient against a large number of attacker classes.

9.2 Discussion

This discussion synthesizes the essence of the previous chapters. Future entities of smartphone security are shown in a coherent section. This is equally done for the security requirements that are reasonable for a future with secure smartphones.

9.2.1 Future Entities

Besides its key contributions, this thesis conveyed the following insight: in the fast-changing world of mobile device security, the boundaries of security are constantly changing. This makes it necessary to reduce the attack surface of the operating systems of mobile devices. This is especially true when the combination of systems

that are secure for themselves lead to unexpected vulnerabilities in combination with other systems.

The device-centric investigations argued for the future entity *policy enforcer*. Under the aspect of security and usability the policy enforcer is a sophisticated combination of a device firewall, an extended installer, and a remote device management receiver.

The network-centric investigations indicated that network-based virus scans are a reasonable measure for decreasing the propagation velocity of mobile malware. Indications exist that update times of signature databases are not critical in the order of days.

One future entity is seen in *proactive security protocols*. They evade the security problems of this thesis in a way by using a secure subnet built by trusted modules that can even withstand a compromised host device acting as a Byzantine node in the protocol.

One possible future entity was not named until now, because it only makes sense in the combination of the contributions of this thesis. This entity is a *fraud database* within the mobile network under the control of the mobile network operator and connected to the mobile devices. This fraud database would at least contain the malicious phone numbers that were introduced in Chapter 7. This is a network-centric security mechanism in combination with the device-centric policy enforcer that will enforce the policies of the fraud database. This database can architecturally be combined with the certification revocation databases.

9.2.2 Security Requirements

We see three main security requirements for mobile devices that contribute to increase their security: address space randomization, an active policy enforcer, and the necessity for process logging.

Address Space Randomization: The technical parts of this thesis—especially the development of our smartphone worm—showed the effects that the simple security measure address space randomization can have. Malware can use hard-coded addresses of system calls that are currently valid on similar devices. With a randomized address space, the hard-coded addresses are only valid for one device. This would lead to an increased effort for malware authors who want to hide their malware from the system by using hard-coded addresses.

Active Policy Enforcer: The policy enforcer of Chapter 6 can be used to limit costs and for other useful security functionality on the mobile device. As this enforcer is proposed to be independent of particular technologies, it can be demanded as a security requirement.

Process Logging: It can be demanded to log the process that initiated a security-relevant event on the mobile device. This is especially true for monetary events. Chapter 6 proved this functionality as feasible. The logs can be used for two reasons. First, they can be used for dispute resolution between the MNO and a user who was a victim to monetary damage caused by mobile malware. Second, this data could be used as input for fraud detection algorithms.

9.3 Perspective

This section discusses the future validity of the fundamentals that underlay the investigations of this thesis. Ultimately, it indicates challenges for future research in mobile device security.

9.3.1 Future Validity of the Specifics

We recall the mobile device security specifics of Figure 2.2. They were the framework for the investigations in this thesis. Important for future research is knowing whether these specifics remain valid.

We see the following directions:

- *Creation of costs* is a specific that will even be of more importance in the future. As more and more services are introduced to mobile devices, it is likely that some of them will be payment services and other services with monetary effects.
- The *network environment* is likely to stay the same. With the observation of the recent years, a tendency towards an increased usage of remote device management and remote firmware update can be expected.
- The *expensive wireless link* will decrease in importance. The communication costs will decrease because of more available bandwidth of mobile networks. The computation costs for algorithms on the device will also decrease because the fourth generation of mobile networks (4G) is designed for high bandwidth and low latency for data traffic.

- The *limited device resources* are likely to decrease in a way, leading to more processing power and more memory. The battery as the most important factor in mobile devices seems probable to remain a factor that has to be considered. Altogether, *limited device resources* remain a specific of mobile device security.
- The *security-unaware user* might become a little more security-aware when mobile device security moves into common attention. This is likely to be connected to the *reputation*, because more understanding for the problems of mobile device security might relieve the MNOs from unsubstantiated claims from the users.

As a conclusion: smartphones are rapidly closing the gap on common computers in terms of processing power, display size, and versatility of operating systems. However, they have inherent constraints that will remain valid in the future. Therefore, the specifics of Figure 2.2 can be used as a framework for future mobile device security research.

9.3.2 Future Challenges

As smartphones are subject to a fast-paced change it can be expected that the shifting boundaries of mobile device security will continue to be a challenge for the future. The shifting boundaries started with the mobile Web browser that evolved into a complete application framework. For this reason, the policy enforcer was introduced as an entity that is independent of functional progress. However, even the security contributions of this concept might be insufficient in the future.

A challenge for realization is application revocation of mobile device applications. This topic came into focus in 2009 when two signed pieces of malware were found.

A challenge for future offensive research can be the abuse of firmware flashing functionality. Even if this was assumed to be secure in the context of this thesis, the process might be attacked successfully. As this attack would lead to devices that are completely Byzantine, they might not only attack other devices but even the mobile network instead.

Two challenges concerning the user interface were not covered with the solutions of this thesis. First, it is possible that the user interface does not display the message that the program or the operating system expects. Examples are APIs for dialog boxes that accept strings of an arbitrary length for the message to be displayed. This can be exploited if these string are related to the other elements of the dialog box. Second and even more challenging is malware that is able to simulate actions

of the user, e.g. by automatically reacting to security confirmations. Some common computer operating systems provide APIs for this task and it can be imagined that mobile operating systems will provide this functionality in the future. It becomes even more intriguing in combination with mobile malware that re-flashed some parts of the operating system.

Both of these user interface challenges are on another level than the device-centric security policy enforcer. A first solution to these problems is introducing Turing tests (CAPTCHAs) for every security-relevant event on the mobile device in order to prove that an event was confirmed by the human user. The task for future research could be to explore the portion of security problems that remain when this solution is applied.

An important question for the future is how criminals can earn money with mobile devices. At the moment, premium-rate services or foreign country calls are a working method. In the future, smartphone-based payment systems could be exploited. With an abuse database and with the enforcement of this abuse database on a mobile device with the policy enforcer we expect some of the currently working methods to cease.

The challenge for mobile network operators is contributing to the cessation of the current potential to earn money with exploiting mobile device security vulnerabilities, especially concerning premium-rate and foreign country services. And the challenge for future research in mobile device security is identifying the kind of successful attacks that cannot be solved with the security entities that were presented in this thesis.

Bibliography

[1] ACIICMEZ, O., LATIFI, A., SEIFERT, J.-P., AND ZHANG, X. A Trusted Mobile Phone Prototype. In *5th IEEE Consumer Communications and Networking Conference* (Jan. 2008).

[2] ADLEMAN, L. M. An abstract theory of computer viruses. In *CRYPTO '88: Proceedings of the 8th Annual International Cryptology Conference on Advances in Cryptology* (1990).

[3] ALBERT, E., PUEBLA, G., AND HERMENEGILDO, M. Abstraction-Carrying Code: a Model for Mobile Code Safety. *New Generation Computing 26*, 2 (Feb. 2007), 171–204.

[4] ANDERSON, J. Computer Security Technology Planning Study. Technical Report ESD-TR-73-51, U.S. Air Force Electronic Systems Division, 1972.

[5] ANDERSON, R. J. *Security Engineering: A Guide to Building Dependable Distributed Systems*, 2nd ed. Wiley Publishing, Apr. 2008.

[6] ANONYMOUS. How to install ANY applications using platform hack! `http://www.ipmart-forum.com/showthread.php?t=247062` (Accessed 08/2009).

[7] ASOKAN, N., SHOUP, V., AND WAIDNER, M. Asynchronous Protocols for Optimistic Fair Exchange. *Security and Privacy, IEEE Symposium on* (1998).

[8] ASSELINEAU, R., AND HOSPITAL, J.-M. Bewertung der Sicherheit von mobilen Endgeräten unter Windows CE. *Misc 1* (2006), 17–23.

[9] AVIZIENIS, A., LAPRIE, J.-C., RANDELL, B., AND LANDWEHR, C. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing 01*, 1 (2004), 11–33.

[10] AVOINE, G., GÄRTNER, F. C., GUERRAOUI, R., AND VUKOLIC, M. Gracefully Degrading Fair Exchange with Security Modules. In *EDCC* (2005), M. D. Cin, M. Kaâniche, and A. Pataricza, Eds., vol. 3463 of *Lecture Notes in Computer Science*, Springer, pp. 55–71.

[11] AVOINE, G., AND VAUDENAY, S. Fair Exchange with Guardian Angels. In *The 4th International Workshop on Information Security Applications - WISA* (2003), no. 2908 in Lecture Notes in Computer Sciences, pp. 188–202.

[12] AYCOCK, J. *Computer viruses and malware*. Advances in information security; 22. Springer, New York, NY, 2006.

[13] BACHFELD, D. Wurmflug. *c't 13* (2006), 156–163.

[14] BACHFELD, D. Warnung vor neuem Handy-Wurm, Jan. 2008. `http://www.heise.de/security/news/meldung/102303` (Accessed 08/2009).

[15] BADURA, T., AND BECHER, M. Testing the Symbian OS Platform Security Architecture. In *AINA '09: Proceedings of the 23rd International Conference on Advanced Information Networking and Applications* (May 2009), IEEE Computer Society.

[16] BAILEY, M., OBERHEIDE, J., ANDERSEN, J., MAO, Z. M., JAHANIAN, F., AND NAZARIO, J. Automated Classification and Analysis of Internet Malware. In *RAID* (2007), pp. 178–197.

[17] BAYER, U., MOSER, A., KRUEGEL, C., AND KIRDA, E. Dynamic Analysis of Malicious Code - TTAnalyze. *Journal of Computer Virology* (2006).

[18] BECHER, M. *Web Application Firewalls - Applied Web Application Security*. VDM Verlag, 2007.

[19] BECHER, M., AND FREILING, F. C. Towards Dynamic Malware Analysis to Increase Mobile Device Security. In *Proc. of SICHERHEIT* (2008).

[20] BECHER, M., FREILING, F. C., AND LEIDNER, B. On the Effort to Create Smartphone Worms in Windows Mobile. In *Information Assurance and Security Workshop* (June 2007), pp. 199–206.

[21] BEN-DAVID, A., NISAN, N., AND PINKAS, B. FairplayMP: a system for secure multi-party computation. In *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security* (New York, NY, USA, 2008), ACM, pp. 257–266.

[22] BERKA, J. Turbo SIM add-on allows full iPhone unlocking, Aug. 2007. http://arstechnica.com/apple/news/2007/08/turbo-sim-add-on-allows-full-iphone-unlocking.ars (Accessed 08/2009).

[23] BESSON, F., DUFAY, G., AND JENSEN, T. P. A Formal Model of Access Control for Mobile Interactive Devices. In Gollmann et al. [86], pp. 110–126.

[24] BLADOX, S.R.O. Turbo SIM. http://www.bladox.com/ (Accessed 08/2009).

[25] BONTCHEV, V. SymbOS Malware Classification Problems. In *Virus Bulletin Conference* (Aug. 2006).

[26] BONTCHEV, V. Virusability of Modern Mobile Environments. In *Virus Bulletin Conference* (Sept. 2007).

[27] BOSE, A., HU, X., SHIN, K. G., AND PARK, T. Behavioral detection of malware on mobile handsets. In *MobiSys '08: Proceeding of the 6th International Conference on Mobile Systems, Applications, and Services* (New York, NY, USA, 2008), ACM Press, pp. 225–238.

[28] BRIEGLEB, V. Marktforscher: Offene Handy-Betriebssysteme werden wichtiger, Feb. 2009. http://www.heise.de/mobil/newsticker/meldung/132476 (Accessed 08/2009).

[29] BUGTRAQ. Siemens M Series SMS DoS Vulnerability, Mar. 2003. http://www.securityfocus.com/bid/7004/ (Accessed 08/2009).

[30] CARETTONI, L., MERLONI, C., AND ZANERO, S. Studying Bluetooth Malware Propagation: The BlueBag Project. *IEEE Security and Privacy 5*, 2 (2007), 17–25.

[31] CHARI, S. N., AND CHENG, P.-C. BlueBoX: A policy-driven, host-based intrusion detection system. *ACM Trans. Inf. Syst. Secur. 6*, 2 (2003), 173–200.

[32] CHEN, Z. *Java Card Technology for Smart Cards*. Addison-Wesley Professional, 2000.

[33] CHENG, J., WONG, S. H., YANG, H., AND LU, S. SmartSiren: Virus Detection and Alert for Smartphones. In *MobiSys '07: Proceedings of the 5th International Conference on Mobile Systems, Applications, and Services* (New York, NY, USA, 2007), ACM Press, pp. 258–271.

[34] CHRISTODORESCU, M., AND JHA, S. Testing malware detectors. *SIGSOFT Softw. Eng. Notes 29*, 4 (2004), 34–44.

[35] CLARKE, N. L., AND FURNELL, S. Authentication of users on mobile telephones - A survey of attitudes and practices. *Computers & Security 24*, 7 (2005), 519–527.

[36] COHEN, F. B. *A short course on computer viruses*, 2nd ed. John Wiley & Sons, Inc., New York, NY, USA, 1994.

[37] CORTIÑAS, R., FREILING, F. C., GHAJAR-AZADANLOU, M., LAFUENTE, A., LARREA, M., PENSO, L. D., AND ARRIOLA, I. S. Secure Failure Detection in TrustedPals. In *SSS* (2007), pp. 173–188.

[38] COWAN, C., WAGLE, P., PU, C., BEATTIE, S., AND WALPOLE, J. Buffer overflows: attacks and defenses for the vulnerability of the decade. *Foundations of Intrusion Tolerant Systems, 2003* (2003), 227–237.

[39] CRANOR, L., AND GARFINKEL, S. *Security and Usability: Designing Secure Systems That People Can Use*. O'Reilly Media, Inc., 2005.

[40] CVE-2006-3459. Multiple stack-based buffer overflows in the TIFF library, July 2006. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-3459 (Accessed 08/2009).

[41] CVE-2007-0685. Denial of Service for Internet Explorer on Windows Mobile 5.0 and Windows Mobile 2003 and 2003SE for Smartphones and PocketPC, Feb. 2007. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0685 (Accessed 08/2009).

[42] DAGON, D., MARTIN, T., AND STARNER, T. Mobile Phones as Computing Devices: The Viruses are Coming! *IEEE Pervasive Computing 3*, 4 (2004), 11–15.

[43] DE HAAS, J. The phone in the PDA - Pocket PC Phone edition security. In *Black Hat Briefings Europe* (May 2003).

[44] DEBBABI, M., SALEH, M., TALHI, C., AND ZHIOUA, S. *Embedded Java Security: Security for Mobile Devices*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[45] DENNING, D. E. R. *Information Warfare and Security*. Addison-Wesley, 1999.

[46] DIVE-RECLUS, C., DIXON, J., AND JAKL, M. Symbian OS Platform Security Architecture - Evolution of the Security Architecture for Symbian OS v9. In *IWWST: International Workshop In Wireless Security Technologies* (2005).

[47] DRAGONI, N., MASSACCI, F., NALIUKA, K., AND SIAHAAN, I. Security-by-Contract: Toward a Semantics for Digital Signatures on Mobile Code. In *EuroPKI* (2007), pp. 297–312.

[48] DUNHAM, K., BECHER, M., ABU-NIMEH, S., , FOGIE, S., HERNACKI, B., MORALES, J. A., AND WRIGHT, C. *Mobile Malware Attack and Defense*. Syngress Media, Nov. 2008.

[49] DWORK, C., LYNCH, N., AND STOCKMEYER, L. Consensus in the presence of partial synchrony. *J. ACM 35*, 2 (1988), 288–323.

[50] EMM, D. The Changing Face of Malware. In *IWWST: International Workshop In Wireless Security Technologies* (2005).

[51] ENGEL, T. Remote SMS/MMS Denial of Service - "Curse Of Silence" for Nokia S60 phones, Nov. 2008. http://berlin.ccc.de/~tobias/cos/s60-curse-of-silence-advisory.txt (Accessed 08/2009).

[52] EREN, E., AND DETKEN, K.-O. *Mobile Security*. Hanser, 2006.

[53] EUROPEAN TELECOMMUNICATIONS STANDARD INSTITUTE. 3GPP Security Algorithms. http://www.etsi.org/WebSite/OurServices/Algorithms/3gppalgorithms.aspx (Accessed 08/2009).

[54] EVEN, S., AND YACOB, Y. Relations among public key signature systems. Tech. Rep. Technical Report 175, Technion - Israeli Institute of Technology, 1980.

[55] F-SECURE. Malware Information Pages: Worm:SymbOS/Commwarrior. http://www.f-secure.com/v-descs/commwarrior.shtml (Accessed 08/2009).

[56] F-SECURE. Virus Descriptions: Brador, Aug. 2004. http://www.f-secure.com/v-descs/brador.shtml (Accessed 08/2009).

[57] F-SECURE. Virus Descriptions: MGDropper.A, Dec. 2004. http://www.f-secure.com/v-descs/mgdropper.shtml (Accessed 08/2009).

[58] F-SECURE. Virus Descriptions: Skulls.A, Nov. 2004. http://www.f-secure.com/v-descs/skulls.shtml (Accessed 08/2009).

[59] F-SECURE. Virus Descriptions: Cardblock.A, Sept. 2005. http://www.f-secure.com/v-descs/cardblock_a.shtml (Accessed 08/2009).

[60] F-SECURE. Virus Descriptions: Cardtrap.A, Sept. 2005. http://www.f-secure.com/v-descs/cardtrap_a.shtml (Accessed 08/2009).

[61] F-SECURE. Virus Descriptions: Locknut.A, Feb. 2005. http://www.f-secure.com/v-descs/locknut_a.shtml (Accessed 08/2009).

[62] F-SECURE. Malware Information Pages: Flexispy.A, Mar. 2006. http://www.f-secure.com/v-descs/flexispy_a.shtml (Accessed 08/2009).

[63] F-SECURE. Malware Information Pages: Fontal.A, Jan. 2006. http://www.f-secure.com/v-descs/fontal_a.shtml (Accessed 08/2009).

[64] F-SECURE. Spyware Information Pages: Acallno.A, Aug. 2006. http://www.f-secure.com/sw-desc/acallno_a.shtml (Accessed 08/2009).

[65] F-SECURE. Trojan Information Pages: Redbrowser.A, Mar. 2006. http://www.f-secure.com/v-descs/redbrowser_a.shtml (Accessed 08/2009).

[66] F-SECURE. Trojan Information Pages: Wesber.A, Sept. 2006. http://www.f-secure.com/v-descs/wesber_a.shtml (Accessed 08/2009).

[67] F-SECURE. Malware Information Pages: Blankfont.A, Sept. 2007. http://www.f-secure.com/v-descs/blankfont_a.shtml (Accessed 08/2009).

[68] F-SECURE. Malware Information Pages: SMS-Worm:SymbOS/Feak, Apr. 2007. http://www.f-secure.com/v-descs/sms-worm_symbos_feak.shtml (Accessed 08/2009).

[69] F-SECURE. Malware Information Pages: Worm:SymbOS/Beselo, Dec. 2007. http://www.f-secure.com/v-descs/worm_symbos_beselo.shtml (Accessed 08/2009).

[70] F-SECURE. Malware Information Pages: Trojan:WinCE/InfoJack, Feb. 2008. http://www.f-secure.com/v-descs/trojan_wince_infojack.shtml (Accessed 08/2009).

[71] F-SECURE. Mobile Users Do Not Take Security Precautions, Mar. 2008. http://www.f-secure.com/en_US/about-us/pressroom/news/2008/fs_news_20080304_1_eng.html (Accessed 08/2009).

[72] F-SECURE. Virus Descriptions: Worm:SymbOS/Yxe, 2009. http://www.f-secure.com/v-descs/worm_symbos_yxe.shtml (Accessed 08/2009).

[73] FLEIZACH, C., LILJENSTAM, M., JOHANSSON, P., VOELKER, G. M., AND MEHÉS, A. Can you infect me now? Malware propagation in mobile phone networks. In *WORM '07: Proceedings of the 2007 ACM workshop on Recurring malcode* (New York, NY, USA, 2007), ACM, pp. 61–68.

[74] FOGIE, S. Pocket PC Abuse. In *Black Hat* (July 2004).

[75] FOLEY, S. N., AND DUMIGAN, R. Are handheld viruses a significant threat? *Commun. ACM 44*, 1 (2001), 105–107.

[76] FORT, M., FREILING, F. C., PENSO, L. D., BENENSON, Z., AND KESDOGAN, D. TrustedPals: Secure Multiparty Computation Implemented with Smart Cards. In Gollmann et al. [86], pp. 34–48.

[77] FOX, D. Der IMSI-Catcher. *Datenschutz und Datensicherheit 26*, 4 (2002).

[78] FREILING, F., HOLZ, T., AND MINK, M. Reconstructing Peoples Lives: A Case Study in Teaching Forensic Computing. In *IMF: IT-Incidents Management & IT-Forensics* (2008).

[79] FREILING, F. C. Vom Wert offensiver Methoden - Ein Blick auf IT-Sicherheit aus Angreiferperspektive. *DuD - Datenschutz und Datensicherheit 33*, 4 (2009).

[80] FURNELL, S. Why users cannot use security. *Computers & Security 24*, 4 (2005), 274–279.

[81] FURNELL, S. Making security usable: Are things improving? *Computers & Security 26*, 6 (2007), 434–443.

[82] FURNELL, S., JUSOH, A., AND KATSABAS, D. The challenges of understanding and using security: A survey of end-users. *Computers & Security 25*, 1 (2006), 27–35.

[83] GARFINKEL, S. L. *Design principles and patterns for computer systems that are simultaneously secure and usable*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2005.

[84] GENDRULLIS, T., NOVOTNÝ, M., AND RUPP, A. A Real-World Attack Breaking A5/1 within Hours. In *CHES* (2008), pp. 266–282.

[85] GOLLMANN, D. *Computer Security*, 2nd ed. John Wiley & Sons, Inc., New York, NY, USA, Jan. 2006.

[86] GOLLMANN, D., MEIER, J., AND SABELFELD, A., Eds. *Computer Security - ESORICS 2006, 11th European Symposium on Research in Computer Security, Hamburg, Germany, September 18-20, 2006, Proceedings* (2006), vol. 4189 of *Lecture Notes in Computer Science*, Springer.

[87] GOLOVANOV, S., GOSTEV, A., AND MASLENNIKOV, D. Kaspersky Security Bulletin 2008: Malware Evolution January - June 2008, Sept. 2008. http://www.viruslist.com/analysis?pubid=204792034 (Accessed 08/2009).

[88] GOWDIAK, A. Java 2 Micro Edition security vulnerabilities. In *Hack in the Box Security Conference* (Oct. 2004).

[89] GSMA. MMS Security Considerations, Sept. 2003. http://www.3gpp1.com/ftp/tsg_sa/WG3_Security/TSGS3_31_Munich/Docs/PDF/S3-030694.pdf (Accessed 08/2009).

[90] GUTHERY, S. C., AND CRONIN, M. J. *Developing MMS Applications - Multimedia Messaging Services for Wireless Networks*. McGraw-Hill Professional, June 2003.

[91] GÖBEL, J., HOLZ, T., AND WILLEMS, C. Measurement and Analysis of Autonomous Spreading Malware in a University Environment. In *DIMVA* (2007), B. M. Hämmerli and R. Sommer, Eds., vol. 4579 of *Lecture Notes in Computer Science*, Springer, pp. 109–128.

[92] GÖRLING, S. The Myth of User Education. In *Virus Bulletin Conference* (Oct. 2006).

[93] HAAS, P. Cellular Phone Viruses, Mar. 2005. http://www.cs.ucsb.edu/~seclab/projects/smartphones/2005_haas_cellviruses.pdf (Accessed 08/2009).

[94] HARLEY, D., AND SLADE, R. *Viruses Revealed*. McGraw-Hill Companies, 2001.

[95] HEATH, C. *Symbian OS Platform Security*. John Wiley & Sons, 2006.

[96] HERZOG, A., AND SHAHMEHRI, N. Usability and Security of Personal Firewalls. In *Proceedings of the IFIP TC-11 22nd International Information Security Conference (SEC)* (2007), pp. 37–48.

[97] HUANG, J., ANG, R. J., MCDONALD, C. A. G., AND SUSILO, W. Personal Firewall for Pocket PC 2003: Design & Implementation. In *AINA '05: Proceedings of the 19th International Conference on Advanced Information Networking and Applications* (Washington, DC, USA, 2005), IEEE Computer Society, pp. 661–666.

[98] HURMAN, T. Exploring Windows CE Shellcode, June 2005. http://www.pentest.co.uk/documents/exploringwce/exploring_wce_shellcode.html (Accessed 08/2009).

[99] HUSEBY, S. H. *Innocent Code: A Security Wake-Up Call for Web Programmers*. John Wiley & Sons, 2004.

[100] HYPPONEN, M. Malware goes Mobile. *Scientific American* (2006), 70–77.

[101] HYPPONEN, M. Status of Cell Phone Malware in 2007. In *Black Hat Briefings USA* (July 2007).

[102] ION, I., DRAGOVIC, B., AND CRISPO, B. Extending the Java Virtual Machine to Enforce Fine-Grained Security Policies in Mobile Devices. In *ACSAC '07: Proceedings of the 23nd Annual Computer Security Applications Conference* (2007), IEEE Computer Society, pp. 233–242.

[103] JAMALUDDIN, J., ZOTOU, N., EDWARDS, R., AND COULTON, P. Mobile Phone Vulnerabilities: A New Generation of Malware. In *IEEE International Symposium on Consumer Electronics* (2004).

[104] JANSEN, W. A., GAVRILA, S. I., AND KOROLEV, V. A Unified Framework for Mobile Device Security. In *Security and Management* (2004), H. R. Arabnia, S. Aissi, and Y. Mun, Eds., CSREA Press, pp. 9–14.

[105] JAVA COMMUNITY PROCESS. JSR 249: Mobile Service Architecture 2. http://jcp.org/en/jsr/detail?id=249 (Accessed 08/2009).

[106] JAVA COMMUNITY PROCESS. JSR 118: Mobile Information Device Profile 2.1, June 2006. http://jcp.org/jsr/detail/118.jsp (Accessed 08/2009).

[107] JAVA COMMUNITY PROCESS. JSR 177: Security and Trust Services API for J2ME, Aug. 2007. http://jcp.org/en/jsr/detail?id=177 (Accessed 08/2009).

[108] JAVA COMMUNITY PROCESS. JSR 248: Mobile Service Architecture, Feb. 2008. http://jcp.org/en/jsr/detail?id=248 (Accessed 08/2009).

[109] JAVA COMMUNITY PROCESS. JSR 271: Mobile Information Device Profile 3, Mar. 2008. http://jcp.org/en/jsr/detail?id=271 (Accessed 08/2009).

[110] JONSSON, E., AND OLOVSSON, T. A Quantitative Model of the Security Intrusion Process Based on Attacker Behavior. *IEEE Trans. Softw. Eng. 23*, 4 (1997), 235–245.

[111] JØSANG, A., AND SANDERUD, G. Security in mobile communications: challenges and opportunities. In *ACSW Frontiers '03: Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003* (Darlinghurst, Australia, Australia, 2003), Australian Computer Society, Inc., pp. 43–48.

[112] KASPERSKY. Virus.Win32.Gpcode.ak, June 2008. http://www.viruslist.com/en/viruses/encyclopedia?virusid=313444 (Accessed 08/2009).

[113] KEHR, R. *Mobile Security with Smartcards*. PhD thesis, Department of Computer Science, Darmstadt University of Technology, Darmstadt, Germany, May 2002.

[114] KLEINBERG, J. The wireless epidemic. *Nature 449*, 20 (Sept. 2007), 287–288.

[115] KOTADIA, M. Major smartphone worm 'by 2007', Gartner Study, June 2005.

[116] KOZIOL, J., LITCHFIELD, D., AITEL, D., ANLEY, C., EREN, S., MEHTA, N., AND HASSELL, R. *The Shellcoder's Handbook: Discovering and Exploiting Security Holes*. John Wiley & Sons, 2004.

[117] KREBS, B. Paris Hilton Hack Started With Old-Fashioned Con, May 2005. http://www.washingtonpost.com/wp-dyn/content/article/2005/05/19/AR2005051900711.html (Accessed 08/2009).

[118] KUPER, P. The State of Security. *IEEE Security & Privacy 3*, 5 (2005), 51–53.

[119] LAWTON, G. Is It Finally Time to Worry about Mobile Malware? *IEEE Computer 41*, 5 (2008), 12–14.

[120] LEAVITT, N. Malicious Code Moves to Mobile Devices. *IEEE Computer 33*, 12 (2000), 16–19.

[121] LEAVITT, N. Mobile Phones: The Next Frontier for Hackers? *IEEE Computer 38*, 4 (2005), 20–23.

[122] LEAVITT, N. Will Proposed Standard Make Mobile Phones More Secure? *IEEE Computer 38*, 12 (2005), 20–22.

[123] LEMAN, D. Spy: A Windows CE API Interceptor, Oct. 2003. http://www.ddj.com/architect/184405459 (Accessed 08/2009).

[124] LÖDING, T. Nokias CeBIT-Einladung per SMS bringt eigene Handys zum Absturz, Mar. 2004. http://www.heise.de/newsticker/meldung/45400 (Accessed 08/2009).

[125] MALKHI, D., NISAN, N., PINKAS, B., AND SELLA, Y. Fairplay—a secure two-party computation system. In *SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium* (Berkeley, CA, USA, 2004), USENIX Association.

[126] MANADHATA, P. K., KARABULUT, Y., AND WING, J. M. Report: Measuring the Attack Surfaces of Enterprise Software. In *ESSoS '09: Proceedings of the 1st International Symposium on Engineering Secure Software and Systems* (Berlin, Heidelberg, 2009), Springer-Verlag, pp. 91–100.

[127] MANKO. Goodbye S60 Platform Security, Hello CAPABILITIES!, Oct. 2007. http://www.symbaali.info/2007/10/goodbye-s60-platform-security-hello.html (Accessed 08/2009).

[128] MAYES, K., AND MARKANTONAKIS, K., Eds. *Smart Cards, Tokens, Security and Applications*. Springer, 2008.

[129] MCAFEE. Handy-Viren grassieren: Bereits 83 Prozent aller Mobilfunkbetreiber betroffen, Feb. 2007. http://www.mcafee.com/de/about/press/corporate/2007/20070212_174646_p.html (Accessed 08/2009).

[130] MEYER, U., AND WETZEL, S. A man-in-the-middle attack on UMTS. In *WiSe '04: Proceedings of the 3rd ACM workshop on Wireless security* (New York, NY, USA, 2004), ACM, pp. 90–97.

[131] MICKENS, J. W., AND NOBLE, B. D. Modeling epidemic spreading in mobile environments. In *WiSe '05: Proceedings of the 4th ACM workshop on Wireless security* (New York, NY, USA, 2005), ACM Press, pp. 77–86.

[132] MIDDELHAUFE, H. Secure Multiparty Computation with Mobile Phones. Master's thesis, RWTH Aachen, Feb. 2009.

[133] MIETTINEN, M., AND HALONEN, P. Host-Based Intrusion Detection for Advanced Mobile Devices. In *AINA '06: Proceedings of the 20th International Conference on Advanced Information Networking and Applications* (Washington, DC, USA, 2006), IEEE, pp. 72–76.

[134] MILLER, J. A. Mosquito Trojan Bites Developer Back, Aug. 2004. http://www.smartphonetoday.com/articles/2004/8/2004-8-13-Mosquito-Trojan-Bites.html (Accessed 08/2009).

[135] MITNICK, K. D., AND SIMON, W. L. *The Art of Deception: Controlling the Human Element of Security*. John Wiley & Sons, Inc., New York, NY, USA, 2003.

[136] MOBIUS - MOBILITY, UBIQUITY AND SECURITY. http://mobius.inria.fr/twiki/bin/view/Mobius (Accessed 08/2009).

[137] MOBIUS PROJECT. Deliverable D1.1, Resource and Information Flow Security Requirements, Mar. 2006. http://mobius.inria.fr/twiki/pub/DeliverablesList/WebHome/Deliv1-1.pdf (Accessed 08/2009).

[138] MOBIUS PROJECT. Deliverable D1.2, Framework- and Application-Specific Security Requirements, Mar. 2006. http://mobius.inria.fr/twiki/pub/DeliverablesList/WebHome/Deliv1-2.pdf.

[139] MOLITOR, S. Mobiles unter Beschuss - Viren und Verwandte auf Handys, June 2004. http://www.heise.de/mobil/artikel/50820 (Accessed 08/2009).

[140] MOORE, H. D. Cracking the iPhone, Oct. 2007. http://blog.metasploit.com/2007/10/cracking-iphone-part-1.html (Accessed 08/2009).

[141] MORALES, J. A. *A Behavior Based Approach to Virus Detection*. PhD thesis, Florida International University, Mar. 2008.

[142] MORALES, J. A., CLARKE, P. J., DENG, Y., AND KIBRIA, B. M. G. Testing and evaluating virus detectors for handheld devices. *Journal in Computer Virology 2*, 2 (2006), 135–147.

[143] MOSER, A., KRUEGEL, C., AND KIRDA, E. Exploring Multiple Execution Paths for Malware Analysis. In *SP '07: Proceedings of the 2007 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2007), IEEE Computer Society, pp. 231–245.

[144] MOSER, A., KRUEGEL, C., AND KIRDA, E. Limits of Static Analysis for Malware Detection. In *ACSAC '07: Proceedings of the 23nd Annual Computer Security Applications Conference* (Dec. 2007), pp. 421–430.

[145] MULLINER, C. Exploiting PocketPC. In *What The Hack* (Jul 2005).

[146] MULLINER, C. iPhone Safari phone-auto-dial (vulnerability), June 2009. http://mulliner.org/security/advisories/iphone_safari_phone-auto-dial_vulnerability_advisory.txt (Accessed 08/2009).

[147] MULLINER, C., AND VIGNA, G. Vulnerability Analysis of MMS User Agents. In *ACSAC '06: Proceedings of the 22nd Annual Computer Security Applications Conference* (2006), IEEE Computer Society, pp. 77–88.

[148] MURMANN, T., AND ROSSNAGEL, H. Sicherheitsanalyse von Betriebssystemen für Mobile Endgeräte. In *SICHERHEIT* (2005), H. Federrath, Ed., vol. 62 of *LNI*, GI, pp. 129–138.

[149] MURRAY, J. *Inside Microsoft Windows CE*. Microsoft Press, Redmond, WA, USA, 1998.

[150] MUTZ, D., VALEUR, F., VIGNA, G., AND KRUEGEL, C. Anomalous system call detection. *ACM Trans. Inf. Syst. Secur. 9*, 1 (2006), 61–93.

[151] MYERS, M., ANKNEY, R., MALPANI, A., GALPERIN, S., AND ADAMS, C. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. RFC 2560 (Proposed Standard), June 1999.

[152] NAIR, S. K., SIMPSON, P. N. D., CRISPO, B., AND TANENBAUM, A. S. A Virtual Machine Based Information Flow Control System for Policy Enforcement. *Electron. Notes Theor. Comput. Sci. 197*, 1 (2008), 3–16.

[153] NECULA, G. C. Proof-carrying code. In *POPL '97: Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (New York, NY, USA, 1997), ACM Press, pp. 106–119.

[154] NIELSEN, J. Ten Usability Heuristics, 2005. http://www.useit.com/papers/heuristic/heuristic_list.html (Accessed 08/2009).

[155] O2. How to recognise Comwarrior. http://www.o2.com/cr/comwarrior.asp (Accessed 08/2009).

[156] OMNET++ COMMUNITY. OMNeT++. http://omnetpp.org (Acessed 08/2009).

[157] OMTP LIMITED. OMTP BONDI. http://bondi.omtp.org (Accessed 08/2009).

[158] OMTP LIMITED. Functional Requirements for Remote Service Provisioning - Version 2, Jan. 2006.

[159] OMTP LIMITED. Mobile Application Security - Requirements for Mobile Applications Signing Schemes - Version 1.23, Dec. 2006.

[160] OMTP LIMITED. Trusted Environment - OMTP TR0, Mar. 2006.

[161] OMTP LIMITED. User Research for Application Security Prompting Focus Groups and Usability Study, May 2006.

[162] OMTP LIMITED. Browser, Oct. 2007.

[163] OMTP LIMITED. Advanced Device Management, Jan. 2008.

[164] OMTP LIMITED. Advanced Trusted Environment - OMTP TR1, May 2008.

[165] OMTP LIMITED. Application Security Framework - Version 2.2, June 2008.

[166] OMTP LIMITED. Security Threats on Embedded Consumer Devices, May 2008.

[167] ORANGE. Certificate Security Disabling / Enabling. http://developer.orangews.com/orgspv/comdefq.aspx.

[168] PAGNIA, H., VOGT, H., AND GÄRTNER, F. C. Fair Exchange. *Comput. J. 46*, 1 (2003), 55–75.

[169] PEASE, M., SHOSTAK, R., AND LAMPORT, L. Reaching Agreement in the Presence of Faults. *J. ACM 27*, 2 (1980), 228–234.

[170] PEIKARI, C. PDA attacks, part 2: airborne viruses - evolution of the latest threats. *(IN)SECURE Magazine 4* (Oct. 2005), 32–41.

[171] PEIKARI, C. Analyzing the Crossover Virus: The First PC to Windows Handheld Cross-Infector, Mar. 2006. http://www.informit.com/articles/article.aspx?p=458169 (Accessed 08/2009).

[172] PEIKARI, C., AND FOGIE, S. *Maximum Wireless Security*. Sams Publishing, 2002.

[173] PEIKARI, C., FOGIE, S., AND RATTER/29A. Details Emerge on the First Windows Mobile Virus, Sept. 2004. http://www.informit.com/articles/article.aspx?p=337069 (Accessed 08/2009).

[174] PEIKARI, C., FOGIE, S., RATTER/29A, AND READ, J. Reverse-Engineering the First Pocket PC Trojan, Oct. 2004. http://www.informit.com/articles/article.aspx?p=340544 (Accessed 08/2009).

[175] PISKO, E., RANNENBERG, K., AND ROSSNAGEL, H. Trusted Computing in Mobile Platforms - Players, Usage Scenarios, and Interests. *DuD - Datenschutz und Datensicherheit 29*, 9 (2005), 526–530.

[176] PÜTZ, S., SCHMITZ, R., AND MARTIN, T. Security Mechanisms in UMTS. *Datenschutz und Datensicherheit 25*, 6 (2001).

[177] REYNAUD-PLANTEY, D. Reverse Engineering and Java Viral Analysis. In *Virus Bulletin Conference* (Oct. 2005).

[178] REYNAUD-PLANTEY, D. The Java Mobile Risk. *Journal in Computer Virology 2*, 2 (2006), 101–107.

[179] RIBEIRO, C., ZUQUETE, A., FERREIRA, P., AND GUEDES, P. SPL: An Access Control Language for Security Policies and Complex Constraints. In *Proceedings of the Network and Distributed System Security Symposium* (2001).

[180] ROGGE, M. Bluetooth als Einfallstor - Wie Bluetooth-Marketing Anwender für mobile Viren desensibilisiert, Dec. 2006. http://www.heise.de/security/artikel/81447.

[181] ROOS, U. Handynutzer haften nicht bei Angriffen, July 2007. http://www.heise.de/security/news/meldung/92881 (Accessed 08/2009).

[182] ROTH, V., POLAK, W., RIEFFEL, E. G., AND TURNER, T. Simple and effective defense against evil twin access points. In *WiSec '09: Proceedings of the second ACM conference on Wireless network security* (2008), pp. 220–235.

[183] RUITENBEEK, E. V., COURTNEY, T., SANDERS, W. H., AND STEVENS, F. Quantifying the Effectiveness of Mobile Phone Virus Response Mechanisms. In *DSN '07: Proc. 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (June 2007), T. Courtney, Ed., pp. 790–800.

[184] SAINT-ANDRE, P. Extensible Messaging and Presence Protocol (XMPP): Core. RFC 3920 (Proposed Standard), Oct. 2004.

[185] SAN. Hacking Windows CE. *Phrack Magazine 6*, 63 (July 2005).

[186] SCHECHTER, S. E. *Computer Security Strength & Risk: A Quantitative Approach*. PhD thesis, Harvard, 2004.

[187] SCHMIDT, A.-D., PETERS, F., LAMOUR, F., AND ALBAYRAK, S. Monitoring Smartphones for Anomaly Detection. In *MOBILWARE* (2008), ACM Press.

[188] SCHMIDT, A.-D., SCHMIDT, H.-G., BATYUK, L., CLAUSEN, J. H., CAMTEPE, S. A., ALBAYRAK, S., AND YILDIZLI, C. Smartphone Malware Evolution Revisited: Android Next Target? In *Proceedings of the 4th IEEE International Conference on Malicious and Unwanted Software (Malware 2009)* (2009), IEEE, pp. 1–7.

[189] SCHMIDT, A. U., KUNTZE, N., AND KASPER, M. On the deployment of Mobile Trusted Modules. In *WCNC '08: Proceedings of the Wireless Communications and Networking Conference* (Apr. 2008), IEEE Computer Society, pp. 3163–3168.

[190] SCHMIDT, A. U., KUNTZE, N., AND KASPER, M. Subscriber Authentication in Cellular Networks with Trusted Virtual SIMs. In *Proceedings of the 10th International Conference on Advanced Communication Technology* (Feb. 2008), vol. 2, IEEE Computer Society, pp. 903–908.

[191] SCHMIDT, J. Wachwechsel - 10 Antiviren-Programme im Test. *c't 23* (2008), 146–155.

[192] SCHNEIDER, F. B. Enforceable Security Policies. *Information and System Security 3*, 1 (2000), 30–50.

[193] SCHNEIER, B. *Secrets and Lies: Digital Security in a Networked World*. John Wiley & Sons, 2004.

[194] SCHNEIER, B. The Death of the Security Industry. *IEEE Security and Privacy 5*, 6 (2007), 88.

[195] SCHUOLER, T. Sicherheit im mobilen Netz, June 2006. http://www.voev.ch/dcs/users/2/VoeV_IT_2_Schuoler.pdf (Accessed 08/2009).

[196] SEKAR, R., VENKATAKRISHNAN, V., BASU, S., BHATKAR, S., AND DUVARNEY, D. C. Model-carrying code: a practical approach for safe execution of untrusted applications. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles* (New York, NY, USA, 2003), ACM Press, pp. 15–28.

[197] SHEVCHENKO, A. An overview of mobile device security, Sept. 2005. http://www.viruslist.com/en/analysis?pubid=170773606 (Accessed 08/2009).

[198] SHNEIDERMAN, B., AND PLAISANT, C. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, 4th ed. Pearson Addison Wesley, 2004.

[199] SINGH, P. K., AND LAKHOTIA, A. Analysis and detection of computer viruses and worms: an annotated bibliography. *SIGPLAN Not. 37*, 2 (2002), 29–35.

[200] SMARTCARD DEVELOPER ASSOCIATION. SDA Releases GSM Voice-Privacy Algorithm A5/1. http://www.scard.org/gsm/ (Accessed 08/2009).

[201] SOUSA, P., BESSANI, A., CORREIA, M., NEVES, N., AND VERISSIMO, P. Highly Available Intrusion-Tolerant Services with Proactive-Reactive Recovery. *Parallel and Distributed Systems, IEEE Transactions on forthcoming* (2009).

[202] SPANIOL, O., AND HOFF, S. *Ereignisorientierte Simulation. Konzepte und Systemrealisierung*. Thomson Publishing, 1995.

[203] SPYPHONES.BIZ. SMS CATCHER. http://www.spyphones.biz/index.php?categoryID=81 (Accessed 08/2009).

[204] SU, J., CHAN, K. K. W., MIKLAS, A. G., PO, K., AKHAVAN, A., SAROIU, S., DE LARA, E., AND GOEL, A. A preliminary investigation of worm infections in a bluetooth environment. In *WORM '06: Proceedings of the 4th ACM workshop on Recurring malcode* (New York, NY, USA, 2006), ACM Press, pp. 9–16.

[205] SYMANTEC. SymbOS.Cabir, June 2004. http://www.symantec.com/security_response/writeup.jsp?docid=2004-061419-4412-99 (Acessed 08/2009).

[206] SYMANTEC. MSIL.Letum.A@mm, Apr. 2006. http://www.symantec.com/security_response/writeup.jsp?docid=2006-040813-2610-99.

[207] SYMANTEC. Security Response: WinCE.Pmcryptic.A, Nov. 2008. http://www.symantec.com/security_response/writeup.jsp?docid=2008-111111-4000-99 (Acessed 08/2009).

[208] SYMBIAN. Functions listed by capability. http://www.symbian.com/developer/techlib/v9.2docs/doc_source/guide/platsecsdk/GT_9.3/FunctionsByCapablity.html.

[209] SYMBIAN. HookLogger. http://developer.symbian.com/main/tools_and_sdks/developer_tools/supported/hook_logger/index.jsp (Acessed 08/2009).

[210] SYMBIAN. What Symbian OS Development Kit Do I Need?, Dec. 2005. https://developer.symbian.com/wiki/pages/viewpage.action?pageId=1859 (Acessed 08/2009).

[211] SYMBIAN FOUNDATION. Symbian Signed. http://www.symbiansigned.com/ (Acessed 08/2009).

[212] SYMBIAN FREAK. Mission Accomplished S60 3rd Edition FP1 hacked. http://www.symbian-freak.com/news/008/03/s60_3rd_ed_feature_pack_1_has_been_hacked.htm (Acessed: 08/2009).

[213] SZOR, P. *The Art of Computer Virus Research and Defense*. Addison-Wesley Professional, 2005.

[214] THE METASPLOIT PROJECT. http://www.metasploit.org (Acessed 08/2009).

[215] TÖYSSY, S., AND HELENIUS, M. About malicious software in smartphones. *Journal in Computer Virology 2*, 2 (2006), 109–119.

[216] TREND MICRO. SYMBOS_MOBLER.B, Sept. 2006. http://www.trendmicro.com/vinfo/virusencyclo/default5.asp?VName=SYMBOS_MOBLER.B (Acessed 08/2009).

[217] TREND MICRO. WORM_MOBLER.B, Sept. 2006. http://www.trendmicro.com/vinfo/virusencyclo/default5.asp?VName=WORM_MOBLER.B (Acessed 08/2009).

[218] TRUSTED COMPUTING GROUP. Mobile Trusted Module Specification, version 1.0, revision 6, June 2008. http://www.trustedcomputinggroup.org/developers/mobile/specifications (Acessed 08/2009).

[219] UNIVERSITY OF GLAMORGAN. Disk Study 2008-2009, May 2009. http://isrg.weblog.glam.ac.uk/2009/5/7/disk-study-2008-2009 (Acessed 08/2009).

[220] VERVATA CO LTD. Flexispy Web page. http://www.flexispy.com (Acessed 08/2009).

[221] VIRUSLIST.COM. http://viruslist.com/ (Acessed 08/2009).

[222] VOGT, H., GÄRTNER, F. C., AND PAGNIA, H. Supporting Fair Exchange in Mobile Environments. *Mob. Netw. Appl. 8*, 2 (2003), 127–136.

[223] VOGT, H., PAGNIA, H., AND GÄRTNER, F. C. Using Smart Cards for Fair Exchange. In *WELCOM '01: Proceedings of the Second International Workshop on Electronic Commerce* (London, UK, 2001), Springer-Verlag, pp. 101–113.

[224] VÉTILLARD, E., AHMAD, S., AND TOURNIER, F. Next-Generation Java Card Technology for Secure Mobile Applications. Java One, 2007. http://developers.sun.com/learning/javaoneonline/2007/pdf/TS-5686.pdf (Acessed 08/2009).

[225] WANG, P., GONZALEZ, M. C., HIDALGO, C. A., AND BARABASI, A.-L. Understanding the Spreading Patterns of Mobile Phone Viruses. *Science* (April 2009).

[226] WEISSENBORN, R. Langzeitbombe Handyvirus. Spiegel online, May 2008. http://www.spiegel.de/netzwelt/mobil/0,1518,555019,00.html (Acessed 08/2009).

[227] WHITEHOUSE, O. University of Santa Barbara Release Source Code for Symbian Worm, Oct. 2006. https://forums2.symantec.com/t5/blogs/blogarticlepage/blog-id/mobile_wireless/article-id/18 (Accessed 08/2009).

[228] WHITEHOUSE, O. Even Symbian 9 Spyware Can Get Signed, July 2007. http://www.symantec.com/connect/blogs/even-symbian-9-spyware-can-get-signed (Accessed 08/2009).

[229] WHITEHOUSE, O. Mobile Device Threats (Poster), May 2007. https://forums2.symantec.com/t5/Mobile-Wireless/A-Picture-is-Worth-a-Thousand-Words-And-I-Only-Have-to-Type-300/ba-p/306396 (Accessed 08/2009).

[230] WHITTEN, A., AND TYGAR, J. D. Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0. In *SSYM'99: Proceedings of the 8th conference on USENIX Security Symposium* (Berkeley, CA, USA, 1999), USENIX Association, pp. 14–14.

[231] WILHELM, J., AND CHIUEH, T. A Forced Sampled Execution Approach to Kernel Rootkit Identification. In *RAID* (2007), pp. 219–235.

[232] WILLEMS, C., HOLZ, T., AND FREILING, F. Toward Automated Dynamic Malware Analysis Using CWSandbox. *IEEE Security and Privacy 5*, 2 (2007), pp. 32–39.

[233] XENAKIS, C. Malicious actions against the GPRS technology. *Journal in Computer Virology 2*, 2 (2006), pp. 121–133.

[234] XU, N., ZHANG, F., LUO, Y., JIA, W., XUAN, D., AND TENG, J. Stealthy video capturer: a new video-based spyware in 3G smartphones. In *WiSec '09: Proceedings of the second ACM conference on Wireless network security* (New York, NY, USA, 2009), ACM Press, pp. 69–78.

[235] YAN, G., AND EIDENBENZ, S. Modeling Propagation Dynamics of Bluetooth Worms (Extended Version). *IEEE Transactions on Mobile Computing 8*, 3 (Mar. 2009), pp. 353–368.

[236] ZHENG, H., LI, D., AND GAO, Z. An Epidemic Model of Mobile Phone Virus. In *Pervasive Computing and Applications, 2006 1st International Symposium on* (Aug. 2006), IEEE Computer Society, pp. 1–5.

[237] ZHENG, P., AND NI, L. M. The Rise of the Smart Phone. *IEEE Distributed Systems Online 7*, 3 (2006), p. 3.

[238] ZIMMERMANN, H. OSI reference model—The ISO model of architecture for open systems interconnection. pp. 2–9.