Practical Attack Scenarios on Secure Element-enabled Mobile Devices

Michael Roland, Josef Langer
NFC Research Lab Hagenberg
University of Applied Sciences Upper Austria
{michael.roland, josef.langer}@fh-hagenberg.at

Josef Scharinger
Department of Computational Perception
Johannes Kepler University Linz
josef.scharinger@jku.at

Abstract—**Near Field Communication's card emulation mode is a way to put virtual smartcards into mobile phones. A recently launched application is Google Wallet. Google Wallet turns a phone into a credit card, a prepaid card and a tool to collect gift certificates and discounts. Card emulation mode uses dedicated smartcard chips, which are considered to fulfill high security standards. Therefore, card emulation mode is also considered to be safe and secure. However, an NFC-enabled mobile phone introduces a significantly different threat vector. Especially a mobile phone's permanent connectivity to a global network and the possibility to install arbitrary applications onto smart phones open up for several new attack scenarios. This paper gives an overview of the new risks imposed by mobile connectivity and untrusted mobile phone applications. The various APIs for secure element access on different mobile phone platforms and their access control mechanisms are analyzed. The security aspects of mobile phones are explained. Finally, two practical attack scenarios, a method to perform a denial of service (DoS) attack against a secure element and a method to remotely use the applications on a victims secure element without the victim's knowledge, are highlighted.**

I. Introduction

Near Field Communication (NFC) is a contactless communication technology standardized in ECMA-340 and ECMA-352. It is an advancement of inductively coupled proximity Radio Frequency Identification (RFID) technology and smartcard technology.

NFC has three operating modes: peer-to-peer mode, reader/writer mode and card emulation mode. The peer-to-peer mode is an operating mode specific to NFC and allows two NFC devices to communicate directly with each other. In reader/writer mode, NFC devices can access contactless smartcards, RFID transponders and NFC tags. In card emulation mode, an NFC device emulates a contactless smartcard and, thus, is able to communicate with existing RFID readers.

Technically, the card emulation mode could be handled by an NFC device in several different ways. The most notable difference is the communication standard. An NFC device could emulate one of either ISO/IEC 14443 Type A, ISO/IEC 14443 Type B or FeliCa (JIS X 6319-4). Support for either of these communication technologies is limited by the NFC chipset and, typically, also depends on the geographic region.

Another difference in handling the card emulation is the part of the device that performs the actual emulation. On the one hand, a card can be emulated in software (e.g. on the device's application processor). On the other hand, card emulation can be performed by a dedicated smartcard chip – a so-called secure element (SE). Such a chip can be a dedicated secure element IC (integrated circuit) that is embedded into the NFC device. Another possibility is the combination of the secure element functionality with another smartcard/security device that is used within the NFC device – like a UICC[1] (universal integrated circuit card) or an SD (secure digital) memory card.

Typical use-cases for card emulation are security critical applications like access control and payment. Therefore, emulation by software on a non-secure application processor is not widely used. As of today, only some dedicated NFC reader devices – like ACS's ACR 122U – and only a small number of NFC-enabled mobile phones – specifically those equipped with RIM's BlackBerry 7 operating system [1], [2] – support software card emulation.

The majority of mobile NFC devices use dedicated smartcard chips for card emulation. Examples are the Nokia 6131, the Nokia 6212, the Samsung GT-S5230N ("Player One") and the Samsung Nexus S. The Nokia 6131 and the Nokia 6212 have an embedded secure element. The Samsung GT-S5230N uses an SWP-enabled (Single Wire Protocol) UICC as secure element and the Samsung Nexus S has both, an embedded secure element and support for an SWP-enabled UICC. Only some recent NFC phones developed by Nokia have no support for card emulation at all [3].

Typical secure elements – like NXP's SmartMX – are standard smartcard ICs as used for contact and contactless smartcards. The only difference is the interface they provide: Instead of a smartcard interface according to ISO/IEC 7816-3 (for contact cards) or an antenna (for contactless cards), the secure element has either an NFC Wired Interface (NFC-WI) or a Single Wire Protocol (SWP) interface for the connection to the NFC controller.

As secure elements have the same hardware and software platforms as regular smartcards, they also share the same security standards. A secure element provides secure storage, a secure execution environment and hardware-based support for cryptographic operations. The IC is protected against various attacks that aim at retrieval or manipulation of stored data and processed operations. Smartcard chips and their design process are usually evaluated and certified according to high security standards – like those defined by the Common Criteria

[1]The UICC is often referred to as Subscriber Identity Module (SIM) card.

DOI 10.1109/NFC.2012.10
© 2012 IEEE

protection profiles for smartcard ICs[2]. The same applies to their operating systems[3]. Thus, the secure element fulfills the requirements necessary for security critical applications like access control and even payment.

Smartcards by themselves are considered safe and secure but their integration into NFC-enabled mobile phones significantly changes the presuppositions. A fundamental difference is, for example, that a mobile phone is permanently connected to a global network, while a regular smartcard is typically isolated from the surrounding world when it is not in use.

In this publication we evaluate the impact of embedding secure elements into NFC-enabled mobile phones. Starting with an introduction to smartcards, we investigate the additional risks imposed by mobile connectivity and untrusted mobile phone applications. We, therefore, take a close look at the APIs for secure element access provided by different mobile phone systems. We analyze the access control mechanisms used to protect these APIs. Finally, we explain two practical attack scenarios against secure elements and their applications. One is a method to perform a denial of service (DoS) attack against a secure element. The other is a method to remotely use the applications on a victims secure element without the victim's knowledge.

II. Smartcards

Smartcards are used, for example, as credit cards, access control cards, signature cards or electronic passports. Such application specific smartcards may use customized operating systems, and their application software is usually programmed into a read-only memory during the manufacturing process. Today, however, there also exist generic smartcard platforms which can be loaded with various applications. A single card can even contain multiple applications at the same time.

A. Communication Protocol

Contact smartcards, as well as contactless smartcards, use the same application level protocol which is defined in ISO/IEC 7816-4. Command and response pairs are called APDUs (application protocol data units). Commands are always sent from the reader to the card while responses are always sent from the card to the reader.

B. Java Card and GlobalPlatform

A standardized framework for multi-application smartcards is the Java Card platform. Java Card operating systems provide a common set of application programming interfaces (APIs) and a standardized runtime environment. This allows development of applications independent of the actual smartcard hardware and independent of the actual operating system.

Besides a common API, an application provider also needs a standardized interface to manage a smartcard's lifecycle and application software. *"The GlobalPlatform architecture is designed to provide card issuers with the system management architecture for managing these smart cards"* [4]. GlobalPlatform specifies interfaces, mechanisms and commands to allow secure smartcard application management. The management facilities are independent of the actual smartcard hardware and of the actual operating system, and are, thus, interoperable.

Most current secure elements for NFC-enabled mobile devices have a Java Card operating system and allow GlobalPlatform compliant card and application management. An example for a Java Card and GlobalPlatform compliant operating system is JCOP (Java Card Open Platform).

A GlobalPlatform compliant smartcard contains a *Card Manager*, which is the central component for card administration. It is responsible for managing the whole life cycle of a card and its applications (*card content*).

GlobalPlatform defines the following states for the life cycle of a smartcard [4]: OP_READY, INITIALIZED, SECURED, CARD_LOCKED, and TERMINATED. OP_READY is the card's initial state after production. During initialization with initial keys for card management the life cycle state irreversibly traverses from OP_READY via INITIALIZED to SECURED. In the state SECURED, the card is ready for issuance. When the card is in the state CARD_LOCKED, applications on the card can be used, but card content can no longer be managed (i.e. no applications can be added or removed). This state is reversible to SECURED. TERMINATED is similar to CARD_LOCKED, but transitions to it are irreversible. Thus, once a card reached this state, management of card life cycle and card content are no longer possible. As this state is permanent it is intended for cases where a severe security threat was detected or where a card has expired [4].

Many secure elements traverse to TERMINATED states for security reasons if the number of failed authentication attempts for card management reaches a certain threshold. For instance, the secure elements included in the Nokia 6131 and the Nokia 6212 mobile phones have a threshold of ten successive authentication failures [5].

III. APIs for Secure Element Access

The secure element in an NFC mobile phone can be accessed from two sides. In external mode, the secure element emulates a contactless smartcard to external RFID/NFC hardware. In internal mode, the secure element is accessible from mobile phone applications. The various mobile phone platforms define different APIs and access restrictions for the internal mode.

A. JSR 177

The Security and Trust Services API (SATSA, JSR 177 [6]) specifies a number of Java programming interfaces for integrating secure elements into Java applications. Specifically the sub-package SATSA-APDU is designed for APDU-based communication with secure elements. JSR 177 is defined for Java ME (Java Platform, Micro Edition), which is a Java platform specifically designed for devices with limited processing and storage capabilities – like mobile phones.

In today's Java ME capable devices SATSA-APDU is mainly used for access to the mobile phone's UICC. An interface `APDUConnection` is provided for creating connections to

[2]E.g. "Smartcard IC Platform Protection Profile", Version 1.0, July 2001 and "Protection Profile Smart Card IC with Multi-Application Secure Platform", Version 2.0, November 2000.

[3]E.g. "Java Card™ System Protection Profile Collection", Version 1.0b, August 2003 and "Java Card™ System Protection Profile Open Configuration", Version 2.6, April 2010.

specific card applications (*applets*) on the secure element. Connections are opened based on an applet's application identifier (AID). The interface has methods for verification of PIN codes, for retrieval of the card's answer to reset (ATR) and for exchange of arbitrary APDUs with the selected applet.

Access to the SATSA-APDU API is protected by Java ME permissions. The permissions for smartcard access are only granted to signed applications. Applications in the manufacturer domain and the operator domain are automatically granted the permission while applications in the trusted third party domain may require additional user interaction in order for the permissions to be granted.

As an addition to this basic access control scheme, the SATSA specification recommends a more sophisticated access control model in order to protect the secure element from malicious mobile phone applications. The *Recommended Security Element Access Control* [6] defines two mechanisms for fine-grained access control to secure element applications. The first mechanism extends the security domains of the Java ME device in that only applications signed with certificates that chain back to a root certificate provided by the secure element are granted access. The second mechanism is an access control scheme based on access control lists (ACLs). The secure element as a whole and each applet can have their own access control entries (ACEs). The access control scheme grants access based on the APDU header information and the mobile phone application's security domain (manufacturer, operator or trusted third party).

The SATSA specification makes some important assumptions for the access control model to be secure: Mobile phone applications are bound by all secure element access restrictions and both the mobile phone application and the applet trust the mobile device platform [6].

B. Nokia's Extensions to JSR 257

The Contactless Communication API (JSR 257 [7]) specifies a Java programming interface for access to contactless targets (RFID/NFC tags and visual tags). Consequently, this API provides access to NFC's reader/writer mode. For their first NFC phones (specifically Nokia 6131 and Nokia 6212), Nokia developed some extensions to the Contactless Communication API in order to support more features of NFC. Besides support for further RFID tag types and for some limited peer-to-peer functionality, Nokia's extensions to JSR 257 also provide access to the embedded secure element of their mobile phones.

Both JSR 177 and JSR 257 provide access to smartcards. While JSR 177 is intended for access of specific applets on secure elements connected to or integrated into a mobile device, JSR 257 is intended for access to any contactless smartcards that are accessed through a device's RFID/NFC reader. JSR 257 provides an interface `ISO14443Connection` for creating connections to contactless smartcards. The interface has a single method for exchange of arbitrary APDUs with the card. As opposed to JSR 177 this connection is not limited to one specific applet.

With Nokia's extensions a connection can also be established to the phone's embedded secure element. This compensates for the missing support of access to the embedded secure element through JSR 177 on their devices.

Opening an `ISO14443Connection` is subject to protection by Java ME permissions. On Nokia's mobile phones, however, the permission for contactless smartcard access is granted to any application by default. Even applications in the untrusted third party domain may freely access this API. For secure element access through the API extensions an additional security scheme has been introduced. This scheme requires that an application is in the manufacturer, the operator or the trusted third party security domain. Therefore, only applications that are signed with trusted certificates are granted access to the secure element.

C. BlackBerry 7

BlackBerry uses an interface similar to SATSA-APDU for secure element communication. Additional helper classes provide information on the available secure elements and allow for easy instantiation of `APDUConnection` objects.

Access to the secure element API is restricted to applications that are signed with BlackBerry Java code signing keys. Code signing keys are provided to developers free of charge but registration with Research In Motion (RIM) is required.

D. Android

While Android-based NFC-enabled mobile phones, like the Nexus S, have an embedded secure element and also support a UICC-based secure element, Android still does not have a public API for secure element access. Since Android 2.3.4 access to the embedded secure element is possible through an API called `com.android.nfc_extras`, but this interface is not included in the public software development kit (SDK).

This API contains two classes: `NfcAdapterExtras` and `NfcExecutionEnvironment`. `NfcAdapterExtras` is used to enable and disable external card emulation and to retrieve an instance of the secure element's `NfcExecutionEnvironment` class. `NfcExecutionEnvironment` is used to exchange APDUs with the embedded secure element.

In Android 2.3.4 the NFC-Extras API could be accessed by any application that held the permission to use NFC. In later versions this has been changed to a special permission named `com.android.nfc.permission.NFCEE_ADMIN`. This special permission is only granted to applications which are signed with the same certificate as the NFC system service. Consequently, access to the secure element is restricted to applications trusted by the manufacturer/provider of the NFC system service.

E. SEEK for Android

The SEEK for Android project[4] provides a standardized smartcard API for the Android platform. The API is compliant to the Open Mobile API defined by SIMalliance. This API is intended to provide access to any kind of secure element available in a mobile phone.

SEEK for Android defines a sophisticated security scheme [8] to prevent smartcard access from unauthorized applications. The scheme is similar to JSR 177's *Recommended Security Element*

[4]At the moment SEEK for Android is only a proposed solution and is not integrated into the Android system release.

Table I
COMPARISON OF SECURE ELEMENT API ACCESS CONTROL

	JSR 177	JSR 257 ext.	BlackBerry 7	Android 2.3.4	Android 2.3.5+	SEEK
Valid application certificate	●	●	●	●	●	●
Trusted application certificate	●	●	●		●	●
App. cert. matches applet's policy	●					●
Manufacturer only					●	
APDU filter	●					●

Access Control. An access control enforcer (ACE) reads an access policy from an access control applet (ACA) on the secure element. This policy defines access rules based on an applet's AID and an Android application's signing certificate. Additionally, access can be limited based on APDU header information. While the access control policy is stored and managed on the secure element, the ACE is part of the smartcard service on the mobile phone. Consequently, the policy is only enforced when the SEEK smartcard API is used to access the secure element.

F. Comparison of API Access

The examined programming interfaces have diverging access control mechanisms. Table I shows a comparison of these mechanisms. All APIs require an application to be signed with a valid code-signing certificate. On Android 2.3.4 this is the only requirement to request the permission for secure element access. The other APIs additionally require the code-signing certificate to be trusted by the operating system. On Android 2.3.5 and later this trust is established by comparing the certificates of the application and the NFC service. Only if those were created with the same signing-keys the application is granted access. Thus, only the system manufacturer can grant this permission to an application. With JSR 257 extensions any valid code-signing certificate can be used. For the BlackBerry 7 API, special code-signing keys, provided by the device manufacturer on request, are necessary. JSR 177 and SEEK for Android have the most sophisticated access control. Both APIs allow the definition of access control policies that consider specific applets, the mobile phone application's certificate, and specific APDUs.

Yet, all APIs assume that the underlying operating system and the mobile phone hardware can be trusted. As a consequence, the secure element may not be sufficiently protected if an application can bypass security measures of the operating system.

IV. RELATED WORK AND DISCUSSION

A. Privilege Escalation and Malicious Software

For many mobile phone platforms there exist methods to circumvent security measures. Popular techniques used on many smart phones are "jail breaking" and "rooting". Jail breaking refers to escaping the restrictions imposed by the operating system, so that an application can access resources it usually could not access. Rooting refers to an even sever scenario where the user or an application gains full access to the whole system. Both methods are often used intentionally by device owners/users to circumvent digital rights management or to gain "improved" control over a device.

However, similar exploits can be integrated in virtually any application. That way an application could elevate its permissions without the user's knowledge. Intentional jail breaking and rooting by users could even open further vulnerabilities that might be abused to gain access to restricted resources.

Lately the topic of mobile phone security experiences significantly increasing awareness. Recent research activities include the assessment of vulnerabilities and threats and the uncovering of actual attack scenarios.

Jeon et al. [9] analyzed the vulnerabilities and threats in smart phone security. They identified vulnerabilities caused by implementation errors, incompatibilities, user unawareness, improper configuration, social engineering, loss of smart phones and a smart phone's interaction with its environment. While smart phones' connectivity features (internet, wireless networks...) make them "*useful and most popular*", these same features open up various paths for intruders [9]. The threats identified by Jeon et al. comprise malware, attacks through (wireless) networks, denial of service, break-in attacks, malfunction, phishing, loss of devices and platform alteration.

Verdult and Kooman [10] describe a method to inject malicious software into Nokia's NFC-enabled S40 phones. Based on their method, an attacker can use a combination of NFC and Bluetooth to install software on a victim's phone. Due to an issue with Nokia's proprietary PC Suite interface – which is exposed through Bluetooth – an application's privileges can be elevated into the operator or the manufacturer domain [10]. Exploit code is even available on the internet[5].

Davi et al. [11] investigated the security model of the Android platform. Android's security mechanism consists of a combination of discretionary access control for file system access, sandboxing for application execution, mandatory access control for inter-component communication, component encapsulation and application signing. They state that "*Android does not deal with transitive privilege usage, which allows applications to bypass restrictions imposed by their sandboxes.*" Consequently, an application with lower privileges could potentially access an application with higher privileges in such a way, that it could use these higher privileges for its own purposes.

Höbarth and Mayrhofer [12] introduce a framework for the Android platform that can use arbitrary exploits to achieve permanent privilege escalation. Based on any existing or future exploit that gains temporary *root* level privileges their framework modifies the system in such a way that these *root* level privileges are permanently retained.

Based on such frameworks attackers have an easy platform to integrate the newest exploit code into their malware applications. According to Kaspersky Lab's monthly malware statistics [13] the trend towards threats and malware for Android (and mobile platforms in general) has dramatically increased within the last year. Therefore, it seems unlikely that this trend will be interrupted any time soon.

[5]http://code.google.com/p/nokicert/

As a result, software-based protection of the secure element APIs is likely to be circumvented. On Nokia's NFC-enabled S40 phones, the restrictions of the JSR 257 extensions can be completely avoided. The same applies to the basic access control of JSR 177. We also assume that there will continue to be new privilege escalation exploits for the Android platform in the future.

B. Attacks on Contactless Smartcards

In 2005, Hancke [14] first presented a successful relay attack against ISO/IEC 14443 smartcards. His system directly transmitted the bits (digital information) of the data link layer between two UHF transceivers. The relay system could bridge a distance of up to 50 meters but had significant timing issues during anti-collision of multiple cards [14]. Hancke explains that even cryptographic protocols for confidentiality, authentication and integrity have no influence on the possibility of relay attacks.

Kfir and Wool [15] describe a similar system. They also show that the relay device used to access the victims card can be up to 50 centimeters away from the card when using additional amplification and filtering.

As existing cryptographic protocols on the application layer cannot prevent relay attacks, several methods have been identified to prevent or hinder relay attacks [14]–[16]:

1) The card's radio frequency interface can be shielded with a Faraday cage when not in use.
2) The card could contain additional circuitry for physical activation and deactivation.
3) Additional passwords or PIN codes could be used for two-factor authentication.
4) Distance bounding protocols can be used on fast channels to determine the actual distance between the card and the reader.

Other measures – like measurement of command delays to detect additional delays induced by relay channels – have been identified as not useful. For instance, Hancke et al. [16] conclude that the timing constraints of ISO/IEC 14443 are too loose to provide adequate protection against relay attacks.

Our analysis of the proposed methods reveals that not all of them can be applied to internal or external card emulation:

1) Shielding with a Faraday cage is only possible for external card emulation. For internal card emulation the equivalent of shielding is the API access policy, which we showed to have weaknesses on some platforms.
2) Physical activation and deactivation of card emulation (or actually the secure element) would be possible for both external and internal mode. For instance, the Nokia NFC phones require the user to explicitly allow external card emulation. On the Android platform external card emulation can also be enabled and disabled through software. However, none of these systems allow physical deactivation (e.g. by means of a button) of internal card emulation.
3) We assume that two-factor authentication can reliably prevent relay attacks as long as the PIN codes are entered on the reader side. If they are entered on the mobile phone, an attacker might be able to monitor and reuse previously entered PIN codes. However, at the moment many access control systems and payment solutions do not require PIN codes for contactless transactions.
4) Distance bounding protocols would require an additional fast communication channel (e.g. ultra-wideband [16]). Such a channel is neither defined in current NFC standards nor available in current NFC chipsets.

V. New Attack Scenarios

Our research on related work revealed that contactless smartcards are vulnerable to several types of attack. Besides the relay attack scenario, other scenarios – like jamming and destruction with electromagnetic waves or interception of radio signals – are possible. All these attack scenarios and their proposed solutions also apply to external card emulation.

However, none of these research papers cover the effects of card emulation's internal mode in a mobile phone environment. Above, we showed that secure element APIs are not adequately protected on some mobile phone platforms. Therefore, based on the assumption of unrestricted access to the secure element's internal mode, we analyzed possible attack scenarios. We focused on two classes of attack: denial of service and relay of communication. Based on this analysis, we found two new practical attacks that could be applied to the existing NFC-enabled mobile phones Nokia 6131, Nokia 6212 (both with the latest firmware) and Samsung Nexus S (with Android 2.3.4).

A. Denial of Service (DoS)

The first attack scenario is a denial of service attack. With current embedded secure elements, the card is put into TERMINATED state after ten successive authentication failures for card management. Once the card is in TERMINATED state, all installed applets will continue to be available but card management is no longer possible. As a result, applets cannot be installed or removed.

An authentication attempt to the card manager (also referred to as *issuer security domain*) consists of a sequence of three APDU commands: SELECT (issuer security domain), INITIALIZE UPDATE, and EXTERNAL AUTHENTICATE. For an embedded secure element, card management must be available through the secure element API. Otherwise, over-the-air management of the secure element and its applications would not be possible. As a consequence, every application with access to the secure element API can perform an authentication attempt. Malicious code for permanently locking the secure element could be injected into any (harmless looking) application (e.g. a game).

We consider this a critical problem as a permanently locked (TERMINATED) embedded secure element will render the NFC device unusable for card emulation applications. This may lead to a significant decrease in reputation and user satisfaction. Moreover, it might result in costly product recalls.

B. Relay Attack

"*If a contactless card could be read while in a pocket, purse or wallet, a thief might be able to engage in the act of digital* *pickpocketing while standing next to or merely walking past the victim*" (Hancke [14]).

This type of *virtual pickpocketing* becomes even easier on a secure element-enabled mobile phone. Instead of being close-by to victim's phone, the attacker simply needs to install an application onto the victim's mobile phone. As with the denial of service attack, relevant exploit code could be packed into virtually any application.

Our proposed relay attack scenario consists of a relay application on the victim's mobile phone. This application waits for APDU commands on a network socket and forwards these APDUs to the secure element. The responses are then sent back through the network socket. On the other end of the network socket a card emulator forwards the APDU commands (and responses) from (and to) a smartcard reader. That smartcard reader could, for instance, be part of an access control system or a point-of-sale terminal.

In comparison to existing relay scenarios, where bits are relayed at the data link layer, our attack scenario relays command and response packets (APDUs) at the application layer. Due to this high protocol level, there are practically no timing constraints on the delay through the relay channel. Therefore, Bluetooth, Wi-Fi, or even the mobile phone network could be used as a relay channel.

Any device that supports software card emulation can be used as a card emulator. For example, ACS's ACR 122U NFC reader and the BlackBerry NFC mobile phones support card emulation on APDU level. Especially a BlackBerry mobile phone seems to be an ideal card emulator platform. It not only has the same form factor that is expected for NFC contactless transactions (i.e. a mobile phone), but it also has the same network connectivity as the attack target (i.e. Bluetooth, Wi-Fi, mobile phone network). Thus, it would be possible to directly relay the smartcard communication between the victim's secure element and a BlackBerry mobile phone. At a point-of-sale or an access control gate, nobody would suspect that the communication is actually relayed to a remote device.

VI. Conclusion

We analyzed several available APIs for secure element access in mobile phones. We found that most APIs provide protection against malicious usage but the level of protection varies widely. Yet, with all APIs it is necessary that the underlying operating system and the mobile phone hardware can be trusted. Based on various existing exploits that circumvent such protection mechanisms, we conclude that a secure element may not be sufficiently protected on certain device platforms.

Further, we discovered two new attack scenarios applicable to secure element-enabled mobile phones. The first attack scenario, a denial-of-service attack, potentially allows to permanently terminating the lifecycle of a secure element. Thus, it becomes impossible to manage (install, uninstall) applications on the secure element. This scenario is possible on many GlobalPlatform-compliant secure elements.

The second attack scenario, a relay attack, potentially allows the remote usage of any secure element. By simply installing a malicious application on the victim's mobile phone, communication with the secure element can be relayed across a network (e.g. the Internet) to a card emulator. This card emulator can then be used in the same way as if the victim's secure element was integrated into it. Thus, it performs card emulation with the remote secure element. Existing mobile phones with software card emulation capabilities even provide an attacker with a ready-made card emulator platform that cannot be distinguished from legitimate secure element-enabled devices.

Acknowledgment

This work is part of the project "4EMOBILITY" within the EU programme "Regionale Wettbewerbsfähigkeit OÖ 2007–2013 (Regio 13)" funded by the European regional development fund (ERDF) and the Province of Upper Austria (Land Oberösterreich).

References

[1] S. Clark, "RIM releases BlackBerry NFC APIs," *Near Field Communications World*, May 2011. [Online]. Available: http://www.nfcworld.com/2011/05/31/37778/rim-releases-blackberry-nfc-apis/

[2] *Blackberry API 7.0.0: Package net.rim.device.api.io.nfc.emulation*, RIM, 2011. [Online]. Available: http://www.blackberry.com/developers/docs/7.0.0api/net/rim/device/api/io/nfc/emulation/package-summary.html

[3] H. McLean, "Nokia: No mobile wallet support in current NFC phones," *Near Field Communications World*, Jul. 2011. [Online]. Available: http://www.nfcworld.com/2011/07/21/38715/nokia-no-mobile-wallet-support-in-current-nfc-phones/

[4] *GlobalPlatform Card Specification*, GlobalPlatform Spec., Version 2.2.1, Jan. 2011.

[5] *Nokia 6131 NFC SDK: User's Guide*, Forum Nokia, Version 1.1, Jul. 2007.

[6] *Security and Trust Services API (SATSA)*, Java Community Process Spec. JSR 177, Version 1.0.1, Aug. 2007.

[7] *Contactless Communication API*, Java Community Process Spec. JSR 257, Version 1.1, Jun. 2009.

[8] *Security Concept: Security considerations for the SmartCard API*, SEEK for Android, Jun. 2011. [Online]. Available: http://code.google.com/p/seek-for-android/wiki/SecurityConcept

[9] W. Jeon, J. Kim, Y. Lee, and D. Won, "A Practical Analysis of Smartphone Security," in *Human Interface and the Management of Information. Interacting with Information*, ser. Lecture Notes in Computer Science, vol. 6771/2011. Springer Berlin Heidelberg, 2011, pp. 311–320.

[10] R. Verdult and F. Kooman, "Practical Attacks on NFC Enabled Cell Phones," in *Proceedings of the Third International Workshop on Near Field Communication (NFC 2011)*, Hagenberg, Austria, Feb. 2011, pp. 77–82.

[11] L. Davi, A. Dmitrienko, A.-R. Sadeghi, and M. Winandy, "Privilege Escalation Attacks on Android," in *Information Security*, ser. Lecture Notes in Computer Science, vol. 6531/2011. Springer Berlin Heidelberg, 2011, pp. 346–360.

[12] S. Höbarth and R. Mayrhofer, "A framework for on-device privilege escalation exploit execution on Android," in *Proceedings of IWSSI/SPMU*, Jun. 2011. [Online]. Available: http://www.medien.ifi.lmu.de/iwssi2011/

[13] A. Gostev, "Monthly Malware Statistics: August 2011," Kaspersky Lab, Sep. 2011. [Online]. Available: http://www.securelist.com/en/analysis/204792190/Monthly_Malware_Statistics_August_2011

[14] G. P. Hancke, "A Practical Relay Attack on ISO 14443 Proximity Cards," University of Cambridge, Computer Laboratory, Jan. 2005. [Online]. Available: http://www.rfidblog.org.uk/hancke-rfidrelay.pdf

[15] Z. Kfir and A. Wool, "Picking Virtual Pockets using Relay Attacks on Contactless Smartcard," in *Proceedings of the First International Conference on Security and Privacy for Emerging Areas in Communications Networks (SECURECOMM'05)*, Sep. 2005, pp. 47–58.

[16] G. P. Hancke, K. E. Mayes, and K. Markantonakis, "Confidence in smart token proximity: Relay attacks revisited," *Computers & Security*, vol. 28, no. 7, pp. 615–627, 2009.