Practical Experiences on NFC Relay Attacks with Android: Virtual Pickpocketing Revisited*

José Vila[1] and Ricardo J. Rodríguez[2]

[1] DIIS, University of Zaragoza, Spain
pvtolkien@gmail.com
[2] Research Institute of Applied Sciences in Cybersecurity (RIASC)
University of León, Spain
rj.rodriguez@unileon.es

Abstract. Near Field Communication (NFC) is a short-range contactless communication standard recently emerging as cashless payment technology. However, NFC has been proved vulnerable to several threats, such as eavesdropping, data modification, and relay attacks. A relay attack forwards the entire wireless communication, thus communicating over larger distances. In this paper, we review and discuss feasibility limitations when performing these attacks in Google's Android OS. We show an experiment proving its feasibility using off-the-shelf NFC-enabled Android devices (i.e., no custom firmware nor root required). Thus, Android NFC-capable malicious software might appear before long to virtually pickpocket contactless payment cards within its proximity.

Keywords: NFC, security, relay attacks, Android, contactless payment

1 Introduction

Near Field Communication (NFC) is a bidirectional short-range (up to 10cm) contactless communication technology based on the ISO-14443 [1] and the Sony FeLiCa [2] Radio Frequency Identification (RFID) standards. It operates in the 13.56MHz spectrum and supports data transfer rates of 106, 216, and 424 kbps.

NFC defines three operation modes: peer-to-peer, read/write, and card-emulation mode. In peer-to-peer mode, two NFC devices communicate directly with each other. This mode is commonly used to exchange business cards, or credentials for establishing a network link. Read/write mode allows an NFC device to communicate with an NFC tag. Finally, card-emulation mode enables an NFC device to behave as a contactless smartcard, thus allowing to communicate with an NFC reader/writer.

Nowadays, NFC technology is widely used in a disparity of applications, from ticketing, staff identification, or physical access control, to cashless payment. In fact, the contactless payment sector seems the one where NFC has

* This work was partially supported by Spanish National Cybersecurity Institute (INCIBE) accordingly to the rule 19 of the Digital Confidence Plan (Digital Agency of Spain) and the University of León under the contract X43.

generated more interest, accordingly to market studies [3, 4]. As Fischer envisioned in 2009 [5], the confluence of NFC with smart phones can be the reason behind this fact since NFC is a way to bring "cards" to the mobile [6].

To date, almost 300 different smart phones are (or will be soon) available at the market [7]. Most of them are based on Google's Android OS (or Android for short), while other OS such as Apple's iOS, BlackBerry OS, or Windows Phone OS are less representative. For instance, Apple has just started to add NFC capabilities into its devices: Apple's iPhone 6 is the first model integrated with an NFC chip, although is locked to work only with Apple's contactless payment system [8]. As a recent market research states [9], this trend will keep growing up, expecting to reach more than 500 million of NFC payment users by 2019.

Unfortunately, NFC is insecure as claimed by several works [10–13], where NFC security threats and solutions have been stated. Potential threats of NFC are eavesdropping, data modification (i.e., alteration, insertion, or destruction), and relay attacks. Eavesdropping can be avoided by secure communication, while data modification may require advanced skills and enough knowledge about RF transmission, as well as ad-hoc hardware to perform the attack. A relay attack, defined as a forwarding of the entire wireless communication, allows to communicate over a large distance. A *passive relay attack* forwards the data unaltered, unlike an *active relay attack* [14]. In this paper, we focus on passive relay attacks.

Relay attacks were though to be really difficult or almost impossible to achieve, since the physical constraints on the communication channel. However, the irruption of NFC-enabled mobile phones (or devices) completely changes the threat landscape: Most NFC communication can be relayed – even NFC payment transactions – with NFC-enabled devices. Many works have proved this fact under different attack scenarios, as we reviewed in Section 5.

Mobile malicious software (i.e., malware) usually target user data (such as user credentials or mobile device information), or perform fraud through premium-rate calls or SMS, but we believe that the rise of NFC-enabled devices put NFC in the spotlight for malware developers [15]. To the best of our knowledge, to date there not exist any malware with NFC capabilities although they might appear before long. To prove whether an *NFC-capable malware* might exist nowadays, in this paper we study the feasibility of passive relay attacks in Android. Android is used since it leads the global smartphone market [16] and provides a broad set of freely resources for the developers.

The contribution of this paper is twofold: First, we discuss the implementation alternatives to perform NFC relay attacks in Android; Second, we show a practical implementation of these attacks using two NFC-enabled mobile phones running an off-the-shelf (OTS) Android (i.e., no custom firmware nor root permissions). Our findings put in evidence that these scenarios are nowadays feasible, requiring permission only of NFC and relay communication link chosen (e.g., Bluetooth, WiFi, or GPRS). This issue clearly supposes a high security risk: An NFC-capable malware installed on an Android device can interact with any contactless payment cards in its proximity, being able to conduct illegal

transactions. Current limitations and feasibility of some malware attack scenarios are also introduced.

The outline of this paper is as follows. Section 2 introduces previous concepts. In Section 3, we analyse and discuss practical issues of alternatives provided by Android to perform an NFC passive relay attack. Section 4 describes a practical implementation of this attack using OTS Android NFC-enabled devices, discusses threat scenarios, and introduces countermeasures. Section 5 reviews related works. Finally, Section 6 states conclusions and future work.

2 Background

This section first briefly introduces the ISO/IEC 14443 standard [17] since contactless payment cards rely on it. Then, relay attacks and mafia frauds are introduced. Finally, we review the history of NFC support in Android.

2.1 ISO/IEC 14443 Standard

ISO/IEC 14443 is a four-part international standard for contactless smartcards operating at 13.56 MHz [17]. Proximity Integrated Circuit Cards (PICC), also referred to as *tags*, are intended to operate up to 10cm of a reader antenna, usually termed as Proximity Coupling Device (PCD).

Part 1 of the standard defines the size, physical characteristics, and environmental working conditions of the card. Part 2 defines the RF power and two signalling schemes, Type A and Type B. Both schemes are half duplex with a data rate of 106kbps (in each direction). Part 3 describes initialisation and anticollision protocols, as well as commands, responses, data frame, and timing issues. A polling command is required for waking both card types up and start communication. Part 4 defines the high-level data transmission protocols. A PICC fulfilling all parts of ISO/IEC 14443 is named *IsoDep* card (for instance, contactless payment cards). As response to the polling phase, a PICC reports whether Part 4 is supported. Apart from specific protocol commands, the protocol defined in Part 4 is also capable of transferring Application Protocol Data Units (APDUs) as defined in ISO/IEC 7816-4 [18] and of application selection as defined in ISO/IEC 7816-5 [19]. ISO/IEC 7816-4 and ISO/IEC 7816-5 are part of ISO/IEC 7816, a fifteen-part international standard related to contacted integrated circuit cards, especially smartcards.

2.2 Relay Attacks and Mafia Frauds

Relay attacks were initially introduced by John Conway in 1976 [20], where he explained how a player without knowledge of the chess rules could win to a Grandmaster. To relate these attacks in detail, let us recall here to the well-known people in the security community Alice, Bob, and Eve.

Consider Grandmasters Alice, Bob, who are both challenged by Eve at the same time to play a correspondence chess game (i.e., chess movements are received and sent via email, postal system, or others long-distance correspondence methods). Eve, who ignores the rules, selects a different colour pieces in each match. When playing, she only needs to relay movements received/sent from one match to the other, until a match ends. Fig. 1 shows the scenario described by Conway. Note that both Alice, Bob, think they are playing against Eve, but in fact they are playing between them.

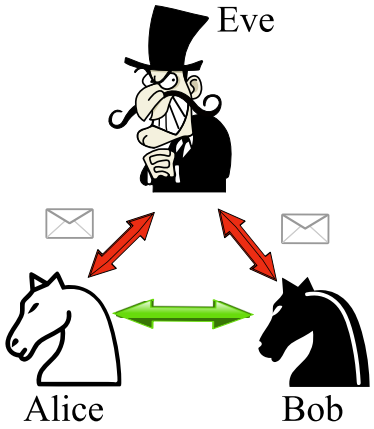


Fig. 1. Chess game relay scenario.

To the best of our knowledge, the first work introducing relay attacks in systems where security is based on the proximity concern is [21]. Desmedt defined the *mafia fraud* as a relay attack where a dishonest prover $\mathcal{P}'$ and a dishonest verifier $\mathcal{V}'$ act together to cheat to a honest verifier $\mathcal{V}$ and a honest prover $\mathcal{P}$, respectively, as: $\mathcal{V} \Longleftrightarrow \mathcal{P}' \ll$ *communication link* $\gg \mathcal{V}' \Longleftrightarrow \mathcal{P}$, where $\mathcal{P}', \mathcal{V}'$, communicates one each other through a communication link. Herein, we adhere to this terminology to refer to the contactless payment card (*honest prover*), the legitimate Point-of-Sale (PoS) terminal (*honest verifier*), and the two NFC-enabled Android devices used to perform an NFC passive relay attack (*dishonest prover* and *dishonest verifier*).

2.3 Evolution of NFC Support in Google's Android OS

NFC support in Android began with Android version 2.3 (*Gingerbread* codename), where only peer-to-peer (used by Android Beam) and read/write operation modes were natively supported. This initial limitation was overcome by the following version updates, being incrementally completed up to gaining an NFC full support and a really comprehensive API for developers in Android version 4.4 (*KitKat* codename). In read/write operation mode, different communication protocols and tags are supported, depending on the NFC chip manufacturer (e.g., ISO/IEC 14443-3 Type A, Type B, and IsoDep tags, to name a few).

Card emulation is the mode with more substantial changes among versions. In the early NFC-enabled Android versions, this mode was only provided via hardware using Secure Elements (SEs) following GlobalPlatform specifications [22]. A SE provides a tamper-proof platform to securely store data and execute applications, thus maintaining confidential data away from an untrusted host. With SE card emulation, the NFC controller directly routes all communication from external reader (e.g., a PoS terminal) to the tamper-resistant chip, without passing through the OS. That is, NFC communication remains transparent to the OS. SEs are capable to communicate not only with the NFC controller but also with mobile applications running within the OS (such as electronic wallets) and *over-the-air* with the Trusted Service Manager (TSM).

TSM is an intermediary authority acting between mobile network operators and phone manufacturers (or other entities controlling a SE) and enabling the service providers (such as banks) to distribute and manage their applications remotely. This closed, distributed ecosystem guarantees a high level of confidence and has been mainly used by the payment sector. However, it excludes other developers from using the card emulation mode. As a result, many developers asked for an easier access to this resource.

The first solution was provided by BlackBerry Limited (formerly known as Research In Motion Limited) company, which included software card emulation (or "soft-SE") mode in BlackBerry 7 OS [23]. This mode allowed the interaction of RFID readers and mobile phone's applications directly, thus completely opening the card emulation to any developer. Basically, soft-SE (also named as Host Card Emulation, HCE) allowed the OS to receive commands from the NFC controller and to response them by any applications instead of by applets installed on a SE.

HCE feature was unofficially supported to Android in 2011, when Doug Year released a set of patches for Android CyanogenMod OS (version 9.1 onward). These patches provided an HCE mode and a middleware interface adding two new tag technologies (namely, IsoPcdA and IsoPcdB). However, this implementation worked only for a specific NFC controller (particularly, NXP PN544), included at those dates in Google's Android devices such as the Samsung Galaxy Nexus or Asus Nexus 7 (first generation). Finally, Android officially supports HCE mode since October 2013, when Android KitKat was released.

3 Practical Implementation Alternatives in Android

This section first introduces the Android NFC architecture. Then, we focus on implementation issues of read/write and card-emulation operation modes in Android as these modes allow to act as a dishonest verifier and a dishonest prover, respectively, in an NFC attack relay. Finally, we point three limitations out when performing these attacks in Android, and discuss its feasibility.

3.1 Android NFC Architecture: NCI Stack

Android NFC development offers an event-driven framework and extensive API on two native implementations that depends on the NFC chip within the device:

- `libnfc-nxp`, which provides support for NXP PN54x NFC controllers and NXP MIFARE family products. These products are not supported by all NFC-enabled devices, since they use a proprietary transmission protocol (that is, MIFARE family cards are not IsoDep cards).
- `libnfc-nci`, which provides support for any NFC Controller Interface (NCI) [24] compliant chips, such as Broadcom's BCM2079x family.

Nowadays, NCI leads the NFC development by several reasons: (i) it provides an open architecture not focused on a single family chip; (ii) it offers an

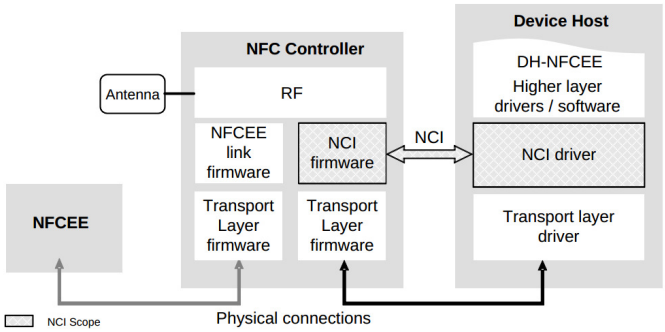


Fig. 2. NFC architecture (taken from [24]).

open interface between the NFC Controller and the DH; and (iii) it is a standard proposed by NFC Forum, a non-profit industry association that develops NFC specifications, ensures interoperability, and educates the market about NFC technology. In this paper, we focus on NCI by the same reasons.

NCI specification aims at making the chip integration of different manufacturers easier by defining a common level of functionality among the components of an NFC-enabled device. It also provides a logical interface that can be used over different physical channels, such as UART, SPI, and I2C.

A noteworthy fact is that Google dropped NXP in favour of Broadcom's NFC stack in their latest Nexus devices (from LG Nexus 4 onward). Moreover, the Android Open Source Project does not support HCE mode for old NXP PN544 chipsets since they lack of AID dynamic routing capabilities, being only possible to use it in devices with closed-source factory ROMs (e.g., Sony Xperia Z1, Sony Compact Z1, and Sony Z Ultra). Latest NFC chipsets developed by NXP, such as NXP PN547, support HCE mode since they are NCI-compliant.

Three main actors are distinguished in a NCI-compliant NFC scenario, as depicted in Fig. 2: the *NFC Execution Environment* (NFCEE), which is a hardware module in most cases (e.g., embedded SEs or Single Wire Protocol enabled SD-Cards); the *NFC Controller* (NFCC); and the *Device Host* (DH), which refers to the main processor, i.e., the own NFC-enabled device.

The NFCC transmits and receives information over the RF channel and is part of a system-on-chip. It maintains a connection with the DH and others NFCEE using the NCI, which defines a logic interface between them independently of the physical layer. NCI also deals with data packet fragmentation, according to the MTU defined by the physical layer.

NCI defines two message types, packed and transmitted over a particular physical channel: (1) **Control messages**, subdivided in *commands*, *responses*, and *notifications*. Commands are only sent from the DH to the NFCC, whereas the others travel in both directions; and (2) **Data messages**, which carry the information addressed to the NFC endpoint (i.e., a remote tag or reader). These messages are only sent once the logic channel has been established. During initial-

isation, the default communication channel is set to the RF connection, although more channels can be created with others NFCEEs.

NCI modules, such as the *RF Interface modules*, are a key part of the NCI stack. An RF Interface module defines how the DH communicates with a given NFC endpoint through the NCI. Each RF Interface supports a specific RF protocol and determines how the NCI data-message payload fits on its respective RF message. This layered design allows a modular addition of interfaces implementing new protocols.

3.2 Read/Write Operation Mode

Android applications are not allowed to directly set the device into read/write mode. However, this mode is indirectly reached as follows: First, it registers the set of NFC tags of interest to be detected through the `AndroidManifest.xml` file; and then, the Android NFC service selects and starts the registered application whether a tag of interest is discovered (i.e., it enters in its proximity communication range). Applications can also ask preference for a discovered tag when they are in foreground mode.

The tag can be discovered by means of the NFCC, which polls the magnetic field. Once a tag is detected, the NFCC first determines the protocol and the technology used. Then, it sends an NCI notification message to the DH with the tag details. The DH (indeed, the `NfcService` in Android) handles this notification and fills a `Tag` object with the data received from the NFCC. Finally, the DH creates and emits an `Intent` with the `EXTRA_TAG` field, which is received by the registered Android application with higher preference.

Android NFC API offers an specific class per RF protocol (i.e., to work with different type of NFC tags) built on top of `TagTechnology` interface. These classes implement the high-level I/O blocking `transceive(byte[] data, boolean raw)` method, which is used to communicate a DH with an NFC tag.

Such a method can work with old NFC implementations (`raw` boolean flag) where some preprocessing is needed before transmitting through the RF channel (e.g., to compute and add CRC payloads to messages). Current `libnfc-nci` implementation ignores this flag, while `libnfc-nxp` implementation uses it to distinguish between ISO/IEC 14443 or NXP proprietary commands. In fact, the `transceive` method executes via Inter Process Communication (IPC) a remote invocation on `TagService` object, defined in `NfcService`. This object is also associated to a `TagEndpoint` object, which references to the remote tag and whose implementation relies on the native library used (`libnfc-nci` or `libnfc-nxp`). Thus, the Android NFC API offers an abstraction of its internal implementation.

Lastly, let us briefly describe low-level issues. The `libnfc-nci` implementation uses a reliable mechanism of queues and message passing named General Kernel Interface (GKI) to easily communicate between layers and modules: Each task is isolated, owning a buffer (or *inbox*) where messages are queued and processed on arrival. This mechanism is used to send messages from the DH to the NFCC chip, and vice versa.

3.3 Host-Card Emulation Operation Mode

Android applications can directly use HCE operation mode by implementing a service to process commands and replies, unlike read/write operation mode, restricted to Android activities. The aforementioned service must extend the `HostApduService` abstract class and implement the `processCommandApdu` and `onDeactivated` methods.

The `processCommandApdu` method is executed whenever an NFC reader sends an APDU message to the registered service. Recall that APDUs are the application-level messages exchanged in an NFC communication between a reader/writer and an IsoDep tag, defined by the ISO/IEC 7816-4 specification (see Section 2.1). In fact, an IsoDep-compliant reader initiates the NFC communication by sending an explicit `SELECT` APDU command to the smartcard to choose a specific Application ID (AID) to communicate with. This command is finally routed by the NFCC to the registered application for the specified AID.

Therefore, each application must previously register the AID list of interest, in order to receive APDU commands. As in the previous case, registration is performed by means of `AndroidManifest.xml` file. Fortunately, Android version 5.0 (*Lollipop* codename) onward enables to dynamically register AIDs.

The Android NFC service (i.e., `NfcService`) populates during initialisation an AID routing table based on the registered AIDs of each application. This table is basically a tree map of AIDs and services.

When the DH receives a `SELECT` command, the routing table is checked and the corresponding service is set to `defaultService`. Thereafter, subsequent APDU commands are routed to that service until other `SELECT` command is received, or a deactivation occurs (by a `DESELECT` command or a timeout). Routing process is performed by `Messenger` objects sent between the `NfcService` and the `HostApduService`: When the `NfcService` needs to route a message, it sends a `MSG_COMMAND_APDU` to `HostApduService`, which extracts the APDU payload and executes `processCommandApdu`. Similarly, an analogous process occurs with a response, but handling a `MSG_RESPONSE_APDU` instead.

Lastly, remark that the NFCC also maintains a routing table populated during each `NfcService` activation. In this case, it stores the destination route according to a set of rules: Technology, Protocol, or specific AID. Destination route can be either the DH (default option) or other NFCEE, such as a SE.

3.4 Limitations and Discussion

As previously described, Android NFC architecture is composed by different layers acting at different levels. Table 1 summarises these layers in top-down developer-accessible order, giving also details of implementation languages, on whom depend them, and whether the code is open source software (OSS).

We envisioned three major limitations when performing an NFC relay attack with dishonest parties (i.e., prover and verifier) in Android.

Description	Language(s)	Dependency	OSS
NFC developer framework (`com.android.nfc` package)	Java, C++	API level	Yes
System NFC library (`libnfc-nxp` or `libnc-nci`)	C/C++	Manufacturer	Yes
NFC Android kernel driver	C	Hardware and manufacturer	Yes
NFC firmware (`/system/vendor/firmware` directory)	(unknown)	Hardware and manufacturer	No

Table 1. NFC architecture levels in Google's Android OS.

Limitation 1. The scenario envisioned by this limitation depicts an NFC-enabled device with a non-NXP NFCC (the device, for short) acting as a dishonest verifier and communicating with a legitimate proprietary tag such as MIFARE Classic smartcards. The device must be in read/write mode. Since `libnfc-nci` implementation does not allow sending raw ISO/IEC 14443-3 commands, the device is unable to communicate with these proprietary tags. Note that the standard defines a CRC field for each command before sent. We have empirically verified where Android computes this CRC value. After some debugging and code review, we concluded that the entity responsible of this computations is, in fact, the ISO/IEC 14443-3 RF Interface module of the NFCC.

Therefore, this limitation is very unlikely to be circumvented, unless NFCC is modified. On the contrary, since contactless payment cards are IsoDep cards (i.e., fully ISO/IEC 14443-compliant), this issue does not really occur at all.

Limitation 2. In this case, we considered an NFC-enabled device acting as a dishonest prover that communicates with a honest verifier. The device must be in HCE mode, which is natively supported from Android KitKat onward (see Section 2.3). Besides, the specific emulated AID has to be known in advance (see Section 3.3). However, since the routing process is performed in the first layer, it could be surrounded in a device with root permissions. Indeed, there exists an Xposed framework module that addresses this issue. The Xposed framework provides a way to make system-level changes into Android without installing a custom firmware, but needs root permissions instead.

Therefore, an OTS Android (version 4.4) onward, dishonest prover can communicate with a honest verifier emulating any AID when this value is known in advance. Otherwise, an Android device with root permissions is needed.

Limitation 3. Finally, we envisioned a complete NFC passive relay attack scenario where a dishonest prover and a dishonest verifier communicate through a non-reliable peer-to-peer relay channel. Note that the success of the relay attack relies on the delay introduced by this communication link. A good review on NFC relay timing constraints can be found in [25]. ISO/IEC 14443-3 [17] specifies a timings values that were found too low for software emulation on mobile devices [26]. However, ISO/IEC 14443-4 [27] defines the Frame Waiting Time

as $FWT = 256 \cdot (16/f_c) \cdot 2^{FWI}, 0 \leq FWI \leq 14$, where $f_c = 13.56$MHz (i.e., the carrier frequency). Thus, FWT ranges from $500\mu s$ to 5s, which practically solves most timing problems in any relay channel. Moreover, the standard also allows to request additional computation time by the PICC with a `WTX` message, which might potentially be implemented on Android to give an attacker additional time for a relay. We aim at further studying the feasibility of these attacks in Android that we named as *timed-extension NFC relay attack*.

Therefore, an NFC relay attack on ISO/IEC 14443-4 is feasible whether the delay on the relay channel is less than 5 seconds. Recall that ISO/IEC 14443-4 is the protocol used by the contactless payment cards.

Let us remark that FWI is completely generated by the NFCC and not configurable by the Android NFC service on HCE mode. Although official documentation defines a $FWI \leq 8$ (i.e., $FWT \leq 78$ms), some PoS devices ignore FWT command sent by the PICC to provide better user responses (and thus, a longer time window). From a read/write mode point of view, this constraint does not happen since Android allows to set a timeout up to 5s.

Conclusions. To summarise, any NFC-enabled device running OTS Android version 4.4 onward is able to perform an NFC passive relay attack at APDU level when the specific AID of the honest prover is known.

4 Relay Attack Implementation

In this section, we first perform a proof-of-concept experiment of relay attack at contactless payment cards. Then, we foresee threat scenarios of NFC-capable Android malware that may appear before long. Finally, countermeasures are briefly discussed.

4.1 Proof-of-Concept Experiment

Recall that three parts are involved in an NFC relay attack (see Section 2.2): (i) a peer-to-peer relay communication channel; (ii) a dishonest verifier, which communicates with the honest prover (i.e., a contactless payment card in this case); and (iii) a dishonest prover, which communicates with the honest verifier (i.e., a PoS in this case). The dishonest verifier works in read/write operation mode, while the dishonest prover relies on HCE operation mode.

For ethical reasons we used our own PoS device. Namely, an Ingenico IWL280 with GPRS and NFC support. As dishonest prover and verifier, we used two off-the-self Android NFC-enabled mobile devices executing an Android application developed for testing purposes (about 2000 Java LOC and able to act as dishonest verifier/prover, depending on user's choice), having a single constraint: The dishonest prover must execute, at least, an Android KitKat version (first version natively supporting HCE mode, see Section 2.3). The experiment has been successful tested on several mobile devices, such as Nexus 4, Nexus 5 as dishonest provers, and Samsung Galaxy Nexus, Sony Xperia S as dishonest verifiers.

$\mathcal{V} \rightarrow \mathcal{P}$	00A4 0400 0E32 5041 592E 5359 532E 4444 4630 3100
$\mathcal{P} \rightarrow \mathcal{V}$	6F2D 840E 3250 4159 2E53 5953 2E44 4446 3031 A51B BF0C 1861 164F 07A0 0000 0003 1010 500B 5649 5341 2042 414E 4B49 4190 00
$\mathcal{V} \rightarrow \mathcal{P}$	00A4 0400 07A0 0000 0003 1010 00
$\mathcal{P} \rightarrow \mathcal{V}$	6F33 8407 A000 0000 0310 10A5 2850 0B56 4953 4120 4241 4E4B 4941 9F38 189F 6604 9F02 069F 0306 9F1A 0295 055F 2A02 9A03 9C01 9F37 0490 00
(messages omitted for privacy issues)	
$\mathcal{V} \rightarrow \mathcal{P}$	80AE 8000 2B00 0000 0000 0100 0000 0000 0007 2480 0000 8000 0978 1502 2400 37FB 88BD 2200 0000 0000 0000 0000 001F 0302 00
$\mathcal{P} \rightarrow \mathcal{V}$	7729 9F27 01XX 9F36 02XX XX9F 2608 XXXX XXXX XXXX XXXX 9F10 12XX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XX90 00

Table 2. Trace excerpt of a MasterCard contactless payment relayed.

The experiment execution workflow is as follows: First, each dishonest party chooses its role (prover or verifier); Then, a peer-to-peer relay channel is established. Finally, the dishonest verifier connects with a contactless payment card, and the dishonest prover relays every APDU from the PoS to the card, and vice versa. Thus, a successful payment transaction is conducted.

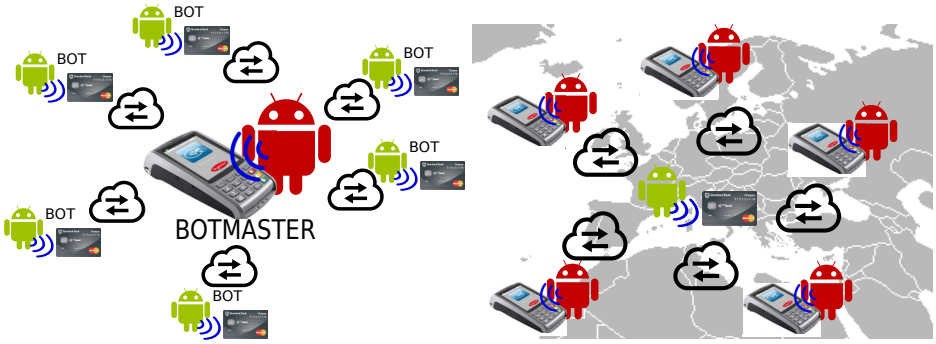
Table 2 shows a time-ordered trace excerpt of a MasterCard contactless transaction, indicating the message flow. First, `SELECT` command (`00A4`) is sent to ask the payment applications within the card. Then, MasterCard application is selected. We omit messages that contain sensitive parameters. Similarly, we obfuscate some bytes relative to cryptogram application generation (last message shown). A video demo of the experiment is publicly available at `URL_HIDDEN_TO_REMAIN_ANONYMOUS`.

4.2 Threat Scenarios

Fig. 3 depicts a pair of threat scenarios using this attack vector. In Fig. 3(a), we envisioned a network of Android infected devices (i.e., a botnet) that communicate with the bot master when a contactless payment card is detected. The bot master can use this smartcard to conduct illegal transactions with a honest verifier, or even multiple transactions at the same time collaborating with multiple dishonest verifiers. We named this attack as *distributed mafia fraud*. Fig. 3(b) foresees the same scenario than before, but with multiple dishonest provers committing fraud at the same time, as a way to hide their real location.

Note that contactless payment cards implement security mechanisms such as asking for a PIN after several uses and checking of atypical paying locations. These mechanisms clearly minimise the impact of the second threat scenario envisioned.

Lastly, let us remark a limitation of these attacks performed as an Android malware. Since read/write operation mode in Android is only reachable by activities (see Section 3.2), an NFC-capable Android malware detecting a smartcard in its proximity range starts execution in foreground. That is, whether the infected device is screen-locked, the malware cannot begin its execution until screen is unlocked. However, note that a malware with root permissions or behaving as a fake app can bypass this limitation.



(a) Scenario 1: Distributed mafia fraud (b) Scenario 2: Hiding fraud locations

Fig. 3. Threat scenarios of NFC passive relay attacks by Android malware.

4.3 Relay Attack Resistant Mechanisms

Relay attack resistant mechanisms proposed in several works [10, 14, 23] are also applicable in these scenarios. For the sake of space, we briefly describe *distance-bounding protocols*, *timing constraints*, and propose *hardware-fingerprinting identification* as practical countermeasure.

Distance-bounding protocols aim at upper bounding the physical distance between two communicating parties based on the Round-Trip-Time of cryptographic challenge-response pairs. Similarly, enforcing timing constraints (e.g., timeout responses) in communication protocols can detect the delay of a relayed communication. Unfortunately, timing constraints are not nowadays enforced in NFC-capable systems. Besides, timing extensions allowed by ISO/IEC 14443 might also be problematic. However, these solutions may partially solve a relay attack scenario using a high latency channel or long-distance dishonest parties.

Usually, an NFC chip uses a random unique identifier (UID) for the emulated card in HCE mode. Thus, whitelisting (or blacklisting) UIDs in an NFC-capable system may solve a relay attack scenario. However, this solution is unfeasible in practice since it breaks legitimate mobile payments as well (card UIDs should be previously known). Nonetheless, it may apply in ad-hoc scenarios.

5 Related Work

Relay attacks on NFC have been widely studied in the literature [14,23,25,28–33]. First works on this topic [14,28] built specific hardware to relay the communication between a smartcard and a legitimate reader. In [28], timing constraints of NFC link are explored, stating that an NFC communication can be relayed up to a distance three orders of magnitude higher than the operating range. Hancke et al. deeply reviewed relay attacks in [14], showing a practical implementation of RFID hardware devices and discussing relay resistant mechanisms (e.g., timing constraints, distance bounding, and additional verification layers).

Following the technological trend, other works perform relay attacks using Nokia mobile phones with NFC capability [23, 29, 30]. Francis et al. described in [29] a relay scenario composed of Nokia NFC-enabled phones and a Java MIDlet application. Similarly, in [23] they relayed legitimate NFC transactions using Nokia NFC-enabled phones and a JavaCard application. Both works also discussed the feasibility of some countermeasures, such as timing, distance bounding, and GPS-based or network cell-based location. In [30], a vulnerability on how Nokia OS handles NFC Data Exchange Format (NDEF) [34] messages is abused to create a Bluetooth link as a backdoor to the phone, thus being able to install third-party software without user content.

Most recently, researchers have focused on relay attacks with Android accessing to SEs [25, 31–33]. SEs are used in mobile devices to securely store data associated with credit/debit cards. In [31–33], Roland et al. described practical attack scenarios to perform Denial-of-Service and sensitive data disclosure in SE of Android devices. However, these attacks require a non OTS Android device, since root permissions are needed. Similarly, Korak and Hutter [25] compared timing on relay attacks using different communication channels (e.g., Bluetooth, GRPS, or WiFi). They also used custom-made hardware and Android devices running custom firmware to build several attack scenarios. Indeed, a custom Android firmware is needed since an Android version prior to KitKat is used (i.e., HCE mode was not natively supported).

6 Conclusions and Future Work

NFC is a bidirectional contactless communication technology that brings the opportunity to merge smartcards with mobile smartphones. NFC has a lot of applications, from physical authorisation or identification, to cashless payment. However, NFC has been shown vulnerable in several aspects, such as eavesdropping, data modification, or relay attacks. Unlike eavesdropping or data modification, relay attacks are a threat that may bypass security countermeasures, such as identification of communication parties or cryptography schemes.

The rise of mobile phones within an NFC chip inside its hardware put relay attacks in the spotlight: An NFC-enabled phone can be abused to interact with smartcards in its proximity range. This becomes a serious risk when relay attacks in contactless payment cards are feasible, since illegal transactions may be conducted.

In this paper, we study the implementation alternatives offered by Google's Android version 4.4 onward to perform an NFC passive relay attack (i.e., data is transmitted unaltered). We discuss some limitations, and show a practical implementation of an NFC relay attack using off-the-shelf NFC-enabled Android mobile phone devices (i.e., no custom firmware nor root permissions are required). We finally explain threats scenarios with this attack vector, and countermeasures that may apply.

Our contribution in this paper includes a practical demonstration of a relay attack implementation using NFC-enabled Android mobile phone platform. In

our opinion, the combination of high number of potentially exploitable devices, easy development, and fast revenue will cause the *virtual pickpocketing attack* to appear in the wild before long.

As future work, we aim at studying timing constraints of HCE mode in OTS Android devices. Similarly, we aim at analysing whether active relay attacks are feasible in Android.

References

1. International Organization for Standardization: ISO/IEC 18092:2013: Information technology – Telecommunications and information exchange between systems – Near Field Communication – Interface and Protocol (NFCIP-1) (March 2013)
2. Japanese Industrial Standard: JIS X 6319-4:2010: Specification of implementation for integrated circuit(s) cards – Part 4: High speed proximity cards (October 2010)
3. Oak, C.: The year 2014 was a tipping point for NFC payments. [Online] (January 2015) `http://www.finextra.com/blogs/fullblog.aspx?blogid=10382`.
4. de Looper, C.: Mobile Payment Boasts Rosy Future, But Some Obstacles Remain in Play. [Online] (January 2015) `http://www.techtimes.com/articles/24762/20150106/mobile-payments-worth-130-billion-2020.htm`.
5. Fischer, J.: NFC in Cell Phones: The New Paradigm for an Interactive World. IEEE Commun. Mag. **47**(6) (June 2009) 22–28
6. Boysen, A.: NFC is the bridge from cards to the mobile. [Online] (January 2015) `http://www.secureidnews.com/news-item/nfc-is-the-bridge-from-cards-to-the-mobile/`.
7. NFC World: NFC phones: The definitive list. [Online; accessed at January 26, 2015] (January 2015) `http://www.nfcworld.com/nfc-phones-list/`.
8. Reardon, M., Tibken, S.: Apple takes NFC mainstream on iPhone 6; Apple Watch with Apple Pay. [Online] (September 2014) `http://www.cnet.com/news/apple-adds-nfc-to-iphone-6-with-apple-pay/`.
9. Juniper Research Limited: Apple Pay and HCE to Push NFC Payment Users to More Than 500 Million by 2019. [Online] (October 2014) `http://www.juniperresearch.com/viewpressrelease.php?pr=483`.
10. Haselsteiner, E., Breitfuß, K.: Security in Near Field Communication (NFC) – Strengths and Weaknesses. In: Proceedings of the Workshop on RFID Security and Privacy. (2006) 1–11
11. Madlmayr, G., Langer, J., Kantner, C., Scharinger, J.: NFC Devices: Security and Privacy. In: Proceedings of the 3rd International Conference on Availability, Reliability and Security. (2008) 642–647
12. Timalsina, S., Bhusal, R., Moh, S.: NFC and Its Application to Mobile Payment: Overview and Comparison. In: Proceedings of the 8th International Conference on Information Science and Digital Content Technology. Volume 1. (2012) 203–206
13. Damme, G.V., Wouters, K.: Practical Experiences with NFC Security on mobile Phones. In: Proceedings of the 2009 International Workshop on Radio Frequency Identification: Security and Privacy Issues. (2009) 1–13
14. Hancke, G., Mayes, K., Markantonakis, K.: Confidence in smart token proximity: Relay attacks revisited. Computers & Security **28**(7) (2009) 615–627
15. Felt, A.P., Finifter, M., Chin, E., Hanna, S., Wagner, D.: A Survey of Mobile Malware in the Wild. In: Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, ACM (2011) 3–14

16. International Data Corporation: Smartphone OS Market Share, Q3 2014. [Online] (2014) http://www.idc.com/prodserv/smartphone-os-market-share.jsp.
17. International Organization for Standardization: ISO/IEC 14443-3: Identification cards – Contactless integrated circuit(s) cards – Proximity cards – Part 3: Initialization and anticollision (April 2011)
18. International Organization for Standardization: ISO/IEC 7816-4-2013: Identification cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange (2013)
19. International Organization for Standardization: ISO/IEC 7816-5-2013: Identification cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange (2004)
20. Conway, J.H.: On Numbers and Games. Academic Press (1976)
21. Desmedt, Y.: Major security problems with the "unforgeable" (Feige-)Fiat-Shamir proofs for identity and how to overcome them. In: Proceedings of the 6th Worldwide Congress on Computer and Communications Security and Protection, SEDEP Paris (1988) 147–159
22. GlobalPlatform: GlobalPlatform Card Specification v2.2.1. [Online] (January 2011) http://www.globalplatform.org/specificationform.asp?fid=7512.
23. Francis, L., Hancke, G., Mayes, K., Markantonakis, K.: Practical Relay Attack on Contactless Transactions by Using NFC Mobile Phones. In: Proceedings of the 2012 Workshop on RFID and IoT Security. Volume 8., IOS Press (2012) 21–32
24. NFC Forum: NFC Controller Interface (NCI) Technical Specification – version 1.1. Technical report, NFC Forum, Inc. (July 2014)
25. Korak, T., Hutter, M.: On the Power of Active Relay Attacks using Custom-Made Proxies. In: Proceedings of the 2014 IEEE International Conference on RFID. (2014) 126–133
26. Hancke, G.: A practical relay attack on ISO 14443 proximity cards. Technical report, University of Cambridge (January 2005)
27. International Organization for Standardization: ISO/IEC 14443-4: Identification cards – Contactless integrated circuit(s) cards – Proximity cards – Part 4: Transmission protocol (Juli 2008)
28. Kfir, Z., Wool, A.: Picking Virtual Pockets using Relay Attacks on Contactless Smartcard. In: Proceedings of the 1st International Conference on Security and Privacy for Emerging Areas in Communications Networks. (2005) 47–58
29. Francis, L., Hancke, G., Mayes, K., Markantonakis, K.: Practical NFC Peer-to-Peer Relay Attack Using Mobile Phones. In: Proceedings of the 6th International Workshop on Radio Frequency Identification: Security and Privacy Issues. Volume 6370 of LNCS., Springer Berlin Heidelberg (2010) 35–49
30. Verdult, R., Kooman, F.: Practical Attacks on NFC Enabled Cell Phones. In: Proceedings of the 3rd International Workshop on NFC. (2011) 77–82
31. Roland, M., Langer, J., Scharinger, J.: Practical Attack Scenarios on Secure Element-Enabled Mobile Devices. In: Proceedings of the 4th International Workshop on NFC. (2012) 19–24
32. Roland, M., Langer, J., Scharinger, J.: Relay Attacks on Secure Element-Enabled Mobile Devices. In Gritzalis, D., Furnell, S., Theoharidou, M., eds.: Proceedings of the 27th IFIP TC 11 Information Security and Privacy Conference. Volume 376., Springer Berlin Heidelberg (2012) 1–12
33. Roland, M., Langer, J., Scharinger, J.: Applying Relay Attacks to Google Wallet. In: Proceedings of the 5th International Workshop on NFC. (2013) 1–6
34. NFC Forum: NFC Data Exchange Format (NDEF) Technical Specification – NDEF 1.0. Technical report, NFC Forum, Inc. (July 2006)