Radboud University Nijmegen

Bachelor's Thesis

Logging NFC data on a Google Nexus S

Author:
Hans Harmannij
s3028933

Supervisor:
prof. dr. B.P.F. Jacobs
Co-supervisor:
drs. ing. R. Verdult

August 27, 2012

Abstract

To analyze Android applications using Near Field Communication (NFC) and to assess their security, it is important to be able to log the NFC communication of those apps. In this thesis an application is presented that can log the communication on the phone that contains the NFC controller, i.e. what is communicated between the NFC controller and the software. In the process the Android library providing NFC functionality is dissected and a first analysis is made of the data that is sent by Android to the controller and what is received back. Also some security considerations are brought forward since the methods used to log the data can be used by attackers to eavesdrop and alter the data.

Contents

Chapter 1

Introduction

1.1 Problem description

Lately, more and more smartphones are produced with support for *Near Field Communication (NFC)* (e.g. [1, 2, 3, 4, 5]) Many of these phones are running the Android operating system (e.g. [1, 2, 3, 4])

The NFC structure in phones like these is shown in fig. 1.1. There is communication between the CPU of the phone and a controller that handles the NFC communication. This controller communicates using radio signals to some other NFC device like an RFID card or a payment terminal. The data between CPU and controller is not the same as what is sent through the air. What the controller sends to the other device however does depend on what is sent by the CPU and what the other device sends to the controller influences what is sent to the CPU. So both communication channels are related and parts of the data will be the same.

If it is known what is being sent using NFC through its radio field and what is sent internally to and from the NFC-chip, this can be analyzed, and the security of this technique, its implementation and of the applications using it can be assessed.

NFC is a well researched subject, so ways to intercept what is sent through the air are already known and used (e.g. [6, 7]). Its use in Android is more recent [8] and therefore less researched. I will therefore focus on intercepting what is sent between CPU and controller.

The goal of my research is to find a way to see what is sent to the chip in Android phones that is responsible for Near Field Communication and what is received from the chip.

1.2 Justification

A very important application for NFC in smartphones is mobile payment. There are numerous companies and consortia that are developing a mobile payment system using NFC, like Google Wallet [9], Visa [10], Isis [11], MasterCard Paypass [12] and Six Pack [13].

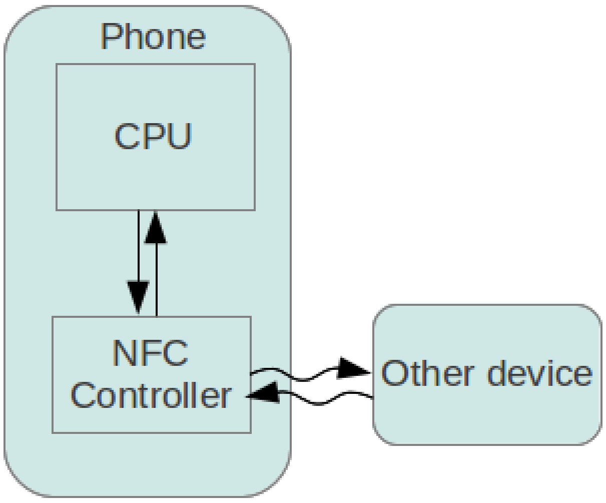


Figure 1.1: NFC structure in phones

In these systems security is an import issue. If the systems are not secure, an attacker might for example be able to tamper with transactions. He might then be able to transfer a different amount of money to and from bank accounts than what is supposed to be transferred. So an attacker can then, for example, pay less than he should or receive more than he should. Another scenario is that the attacker tampers with payments of others and redirects them to his bank account. If the attacker can not tamper with payments, there will still be a problem if he can see information about the payment that should remain private.

The Android operating system is open source ([14]). This means that someone can change the system, including the parts handling NFC on his own phone. If he uses his phone for a mobile payment the changes he made may be able to perform one of the scenarios above. Also if an attacker can gain control of other devices he may be able to do this on other phones than his own.

This is not an exhaustive list of attack scenarios. It does show that there are possible security threats in the use of NFC and mobile payment. It is therefore important to assess those techniques. Publishing about security threats may give attackers new ways for their attacks. However, without research attackers often find vulnerabilities themselves. Research leads to preventing those attacks since it helps the creators of the systems to make their systems more secure and warns people about insecure systems. Where needed, responsible disclosure can be applied.

To analyze mobile payment apps and other NFC enabled apps, you have to know how those apps work and what is sent through NFC. The most straightforward way to get this information is from the creators of the apps. For several reasons however, they often do not disclose this information but use a principle called *security through obscurity*, in which they hope that if attackers don't know how the system is secured, they can't hack it. There are numerous cases where systems like this turned out not to be secure (e.g [15]). The Kerckhoffs' principle expresses this by stating that the security of a cryptographic system should only depend on the secrecy of the key and not on the secrecy of the system itself [16].

Chapter 2

Theoretical framework

2.1 Near Field Communication

NFC is a contactless technology for the short range. It operates at distances of about 4 centimeters or less, with a transfer rates of up to 424kbps. It uses the existing standards for contactless card technology (ISO/IEC 14443 [17, 18, 19, 20], JIS X 6319-4 [21]). It is therefore compatible with cards that conform to those standards. This means that NFC devices can read such cards, but also that the devices can emulate them. NFC extends the contactless card technologies making NFC devices capable of communicating to each other [22].

The NFC Forum defines the NFC Data Exchange Format (NDEF). An NDEF message can contain one or more payloads of an arbitrary type. The standards defines a format for these messages [23], and a protocol for exchanging these messages, for example between two NFC devices or between an NFC reader and an NFC card [24].

2.2 NFC in Google phones

Google has developed two phones with support for NFC, the Google Nexus S [1] and the Google Galaxy Nexus [2]. These phones run the mobile operating system from Google, Android. This operating system has NFC support since the version called Gingerbread, version 2.3 [8]. The source code for Android is freely available [14]. NFC functionality can be found in this source code, and is implemented in the library *libnfc-nxp*.

In both of the Google phones, the PN544 NFC Controller is used [25, 26]. This is the chip that controls the NFC communication. It uses the Host Controller Interface (HCI). This standard describes how the CPU communicates with the NFC Controller.

2.3 Host Controller Interface

HCI defines an interface between hosts in a host network [27]. Hosts are "logical entities that operate one or more service(s)". The host network has a star topology with the host controller in the center, and one or more hosts physically connected to the host controller. HCI consists of three levels:

- Gates for exchanging commands, responses and events
- Host Controller Protocol (HCP) messaging mechanism
- HCP routing mechanism

Hosts exchange messages through pipes. A pipe is a logical communication channel connecting a gate of one host to a gate of another host. There are static pipes which are always available and dynamic gates which can be created and deleted. There are also two types of gates: management gates for managing the network and generic gates.

The messages that are exchanged through these pipes are HCP-messages. These messages consist of a 1 byte header and a variable number of bytes of data. The first two bits of the header describe the instruction type of the message (0 for commands, 1 for events, 2 for responses) and the last 6 bits identify the instruction. See fig. 2.1.

These messages are sent in HCP packets (fig. 2.2). The messages can be fragmented. The packets have a 1 byte header. The first bit is the chaining bit (CB). If $CB = 0$, the following packet contains

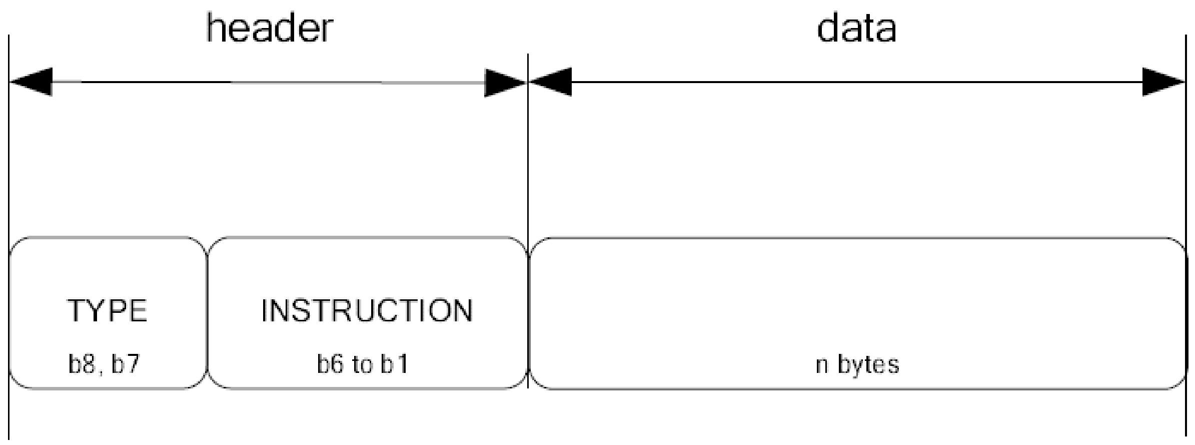


Figure 2.1: HCP Message structure[27]

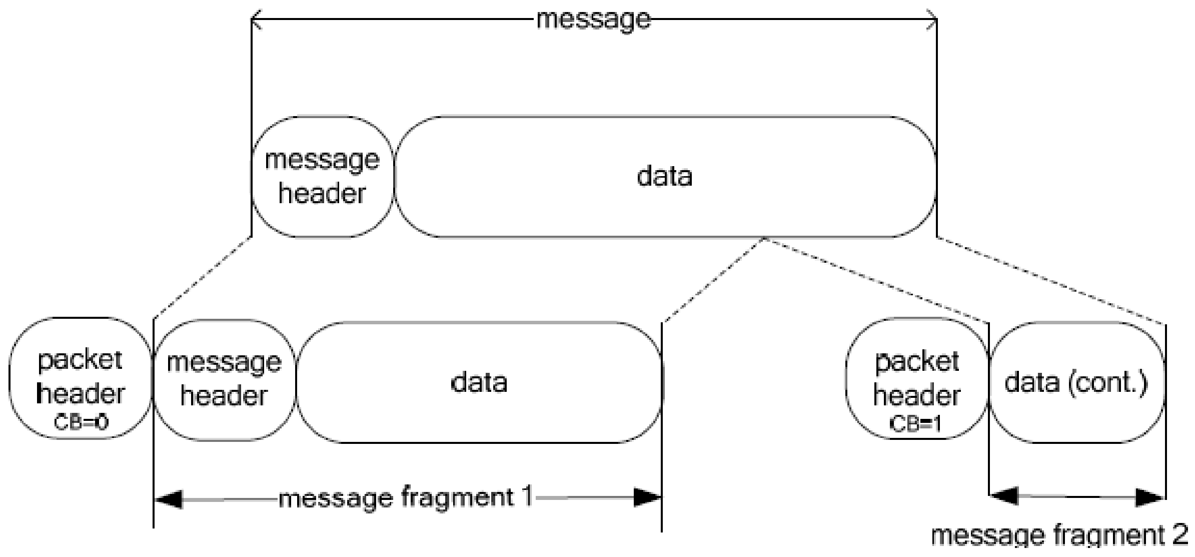


Figure 2.2: Fragmentation of HCP messages in HCP packets[27]

a fragment of the same message, if $CB = 1$, the packet is the last one for this message. The last 7 bits are the pipe identifier (pID) to denote the pipe the packet should be send through.

2.4 Root access in Android

For security reasons, Android denies access to certain, critical parts of the system to untrusted applications. This means that if we want to use those parts we will either have to circumvent the restriction system or we will have to gain elevated privileges. This last option is known as rooting. Smartphone users often root their phones to get more functionality out of their device. For the Google phones there are standard, secure ways to do this. Security vulnerabilities have been exploited to gain root privileges. (e.g. Levitator [28], ZergRush[29]) These exploits have been created for users who are willingly rooting their phone, but vulnerabilities like these are also misused by malicious apps [30].

Chapter 3

Analysis of Libnfc-nxp

3.1 Interaction of modules

In Android, NFC is implemented in the library *libnfc-nxp*. Since the source code of this library is available as part of the Android source, we can analyze this library and find a way to log the data going through it. From the doxygen [31] documentation we can see that there are different parts: DAL, FRI, HAL, HCI, Libnfc, LLC and OSAL. We have to find out how these modules interact with each other.

In order to do so, all the function calls were logged. To do this, the source code was modified to write a message to the Android log at the start of each function and before the function returns. For the DAL and OSAL modules this broke the NFC functionality. The changes to these modules where therefore rolled back. They will be revisited later on.

After modifying the code to log the function calls, an NFC task is performed with the phone. After that the android log is parsed and the results are visualized in a tree view.

For booting the phone a typical part of the visualized log can be seen in fig. 3.1. The beginning of the full log can be found in appendix B.

We observe that the following steps are taken for transceiving data. First the data is sent by going through the modules Libnfc, HAL, HCI and finally LLC. When the writing is finished, a write callback is called which starts the reading. The reading takes place in two steps. Note that this is the case for this example. In other places in the log we see more steps. In the final reading step we go back through the modules LLC, HCI, HAL and Libnfc.

For reading a MIFARE Classic card we see a similar pattern (fig. 3.2), but now it starts with reading. This can take a couple of reading steps, similar to those while booting. The reading goes through the modules LLC, HCI, HAL and Libnfc, and then from Libnfc a command to connect to the MIFARE Classic is send through the modules Libnfc, HAL, HCI and LLC. This is followed by a write callback similar to the one we saw earlier.

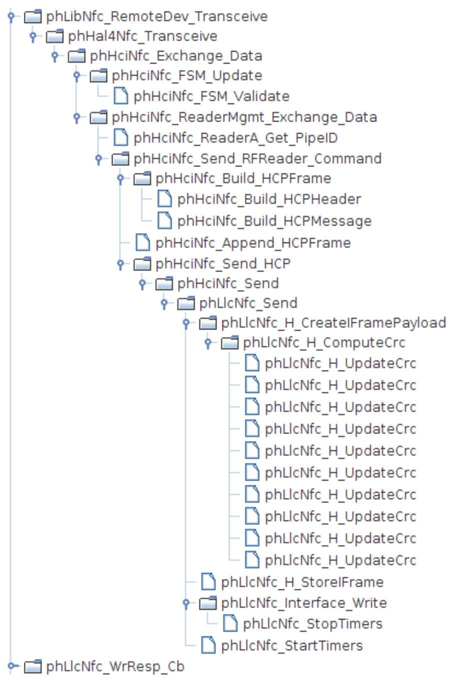
In a later phase of reading the MIFARE Classic, we see that the card is checked for NDEF (fig. 3.3). In this case an extra module is used: FRI. This is called by Libnfc and in turn calls the HAL. The reading step is similar to what we saw before, but again goes through the FRI module. The first part of the full log can be found in Appendix C

We can model the interaction of the different modules as a layered model:

Libnfc	
	FRI
HAL	
HCI	
LLC	

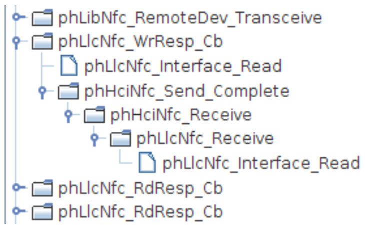
We have two modules left of which we now do not know how they relate to the other modules: OSAL and DAL. According to Wikipedia, an Operating System Abstraction Layer (OSAL) is an API providing wrapper functions to common operating system functionality [32]. When looking at the source and its documentation, we see that the OSAL in Libnfc-nxp does indeed provide such functionality, like freeing and allocating memory and managing timers. A search through the libnfc-nxp source code shows that the OSAL-functions are called from all other modules.

According to Wikipedia, DAL can be a Data Access Layer, which provides functions for persistently storing data [33], or a Database Abstraction Layer, which provides database functionality [34]. The doxygen documentation for the DAL module shows that this is not the functionality it provides. A search through the source shows that the DAL is used by the Libnfc-layer when this layer registers the



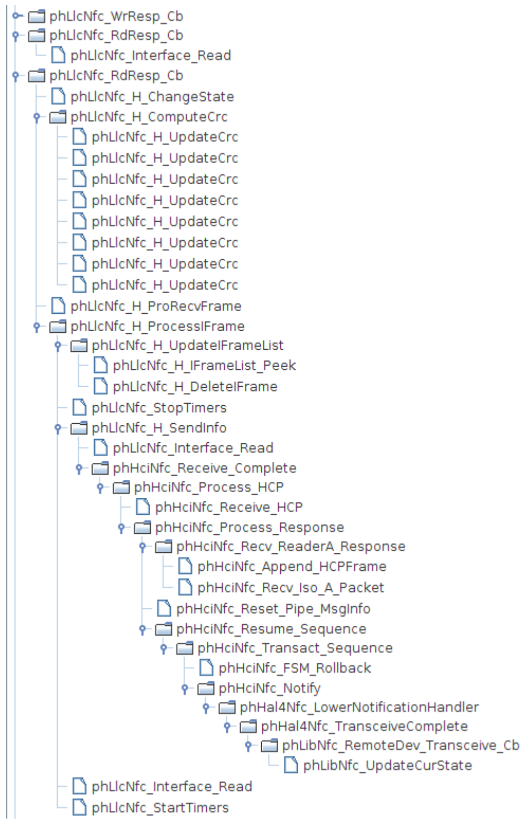
```
phLibNfc_RemoteDev_Transceive
  phHal4Nfc_Transceive
    phHciNfc_Exchange_Data
      phHciNfc_FSM_Update
        phHciNfc_FSM_Validate
      phHciNfc_ReaderMgmt_Exchange_Data
        phHciNfc_ReaderA_Get_PipeID
        phHciNfc_Send_RFReader_Command
          phHciNfc_Build_HCPFrame
            phHciNfc_Build_HCPHeader
            phHciNfc_Build_HCPMessage
          phHciNfc_Append_HCPFrame
          phHciNfc_Send_HCP
            phHciNfc_Send
              phLlcNfc_Send
                phLlcNfc_H_CreateIFramePayload
                  phLlcNfc_H_ComputeCrc
                    phLlcNfc_H_UpdateCrc
                    phLlcNfc_H_UpdateCrc
                    phLlcNfc_H_UpdateCrc
                    phLlcNfc_H_UpdateCrc
                    phLlcNfc_H_UpdateCrc
                    phLlcNfc_H_UpdateCrc
                    phLlcNfc_H_UpdateCrc
                    phLlcNfc_H_UpdateCrc
                    phLlcNfc_H_UpdateCrc
                    phLlcNfc_H_UpdateCrc
                phLlcNfc_H_StoreIFrame
                phLlcNfc_Interface_Write
                  phLlcNfc_StopTimers
                phLlcNfc_StartTimers
phLlcNfc_WrResp_Cb
```

(a) Transceiving data



```
phLibNfc_RemoteDev_Transceive
phLlcNfc_WrResp_Cb
  phLlcNfc_Interface_Read
  phHciNfc_Send_Complete
    phHciNfc_Receive
      phLlcNfc_Receive
        phLlcNfc_Interface_Read
phLlcNfc_RdResp_Cb
phLlcNfc_RdResp_Cb
```

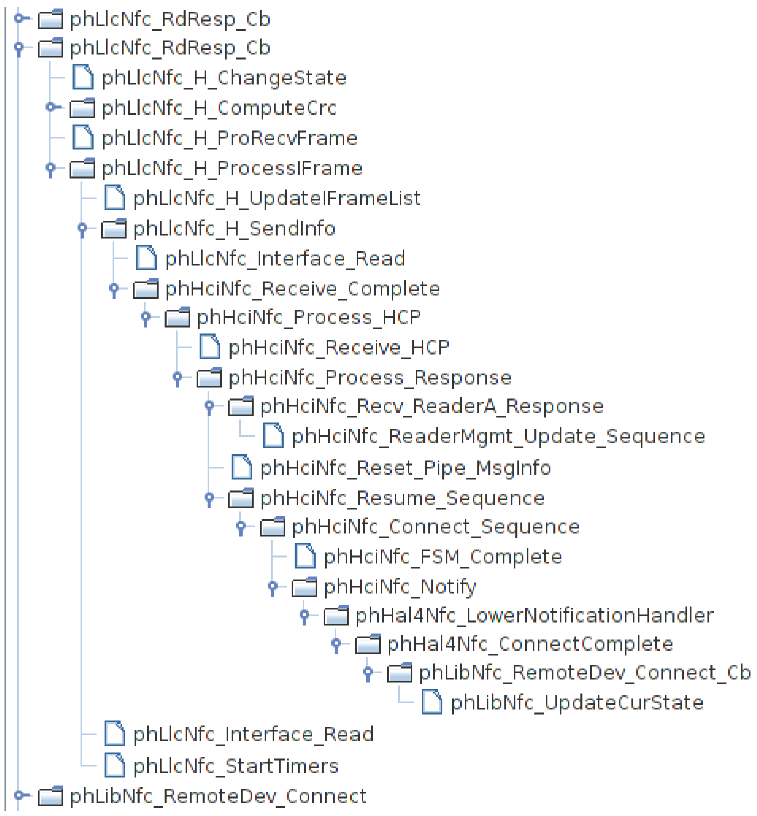
(b) Write callback



```
phLlcNfc_WrResp_Cb
phLlcNfc_RdResp_Cb
  phLlcNfc_Interface_Read
phLlcNfc_RdResp_Cb
  phLlcNfc_H_ChangeState
  phLlcNfc_H_ComputeCrc
    phLlcNfc_H_UpdateCrc
    phLlcNfc_H_UpdateCrc
    phLlcNfc_H_UpdateCrc
    phLlcNfc_H_UpdateCrc
    phLlcNfc_H_UpdateCrc
    phLlcNfc_H_UpdateCrc
    phLlcNfc_H_UpdateCrc
    phLlcNfc_H_UpdateCrc
  phLlcNfc_H_ProRecvFrame
  phLlcNfc_H_ProcessIFrame
    phLlcNfc_H_UpdateIFrameList
      phLlcNfc_H_IFrameList_Peek
      phLlcNfc_H_DeleteIFrame
    phLlcNfc_StopTimers
    phLlcNfc_H_SendInfo
      phLlcNfc_Interface_Read
      phHciNfc_Receive_Complete
        phHciNfc_Process_HCP
          phHciNfc_Receive_HCP
          phHciNfc_Process_Response
            phHciNfc_Recv_ReaderA_Response
              phHciNfc_Append_HCPFrame
              phHciNfc_Recv_Iso_A_Packet
            phHciNfc_Reset_Pipe_MsgInfo
            phHciNfc_Resume_Sequence
              phHciNfc_Transact_Sequence
                phHciNfc_FSM_Rollback
                phHciNfc_Notify
                  phHal4Nfc_LowerNotificationHandler
                    phHal4Nfc_TransceiveComplete
                      phLibNfc_RemoteDev_Transceive_Cb
                        phLibNfc_UpdateCurState
    phLlcNfc_Interface_Read
    phLlcNfc_StartTimers
```

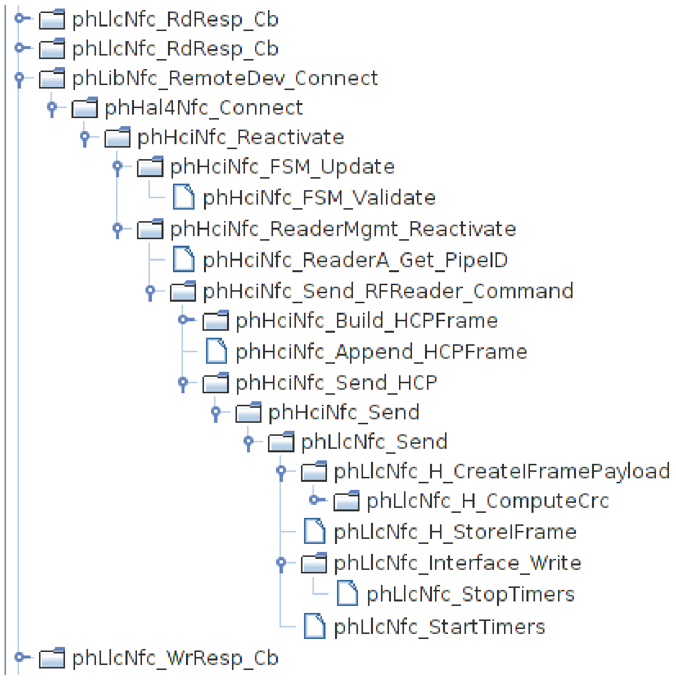
(c) Reading data

Figure 3.1: Typical steps when transceiving data



```
phLlcNfc_RdResp_Cb
phLlcNfc_RdResp_Cb
    phLlcNfc_H_ChangeState
    phLlcNfc_H_ComputeCrc
    phLlcNfc_H_ProRecvFrame
    phLlcNfc_H_ProcessIFrame
        phLlcNfc_H_UpdateIFrameList
        phLlcNfc_H_SendInfo
            phLlcNfc_Interface_Read
            phHciNfc_Receive_Complete
                phHciNfc_Process_HCP
                    phHciNfc_Receive_HCP
                    phHciNfc_Process_Response
                        phHciNfc_Recv_ReaderA_Response
                            phHciNfc_ReaderMgmt_Update_Sequence
                        phHciNfc_Reset_Pipe_MsgInfo
                        phHciNfc_Resume_Sequence
                            phHciNfc_Connect_Sequence
                                phHciNfc_FSM_Complete
                                phHciNfc_Notify
                                    phHal4Nfc_LowerNotificationHandler
                                        phHal4Nfc_ConnectComplete
                                            phLibNfc_RemoteDev_Connect_Cb
                                                phLibNfc_UpdateCurState
        phLlcNfc_Interface_Read
        phLlcNfc_StartTimers
phLibNfc_RemoteDev_Connect
```

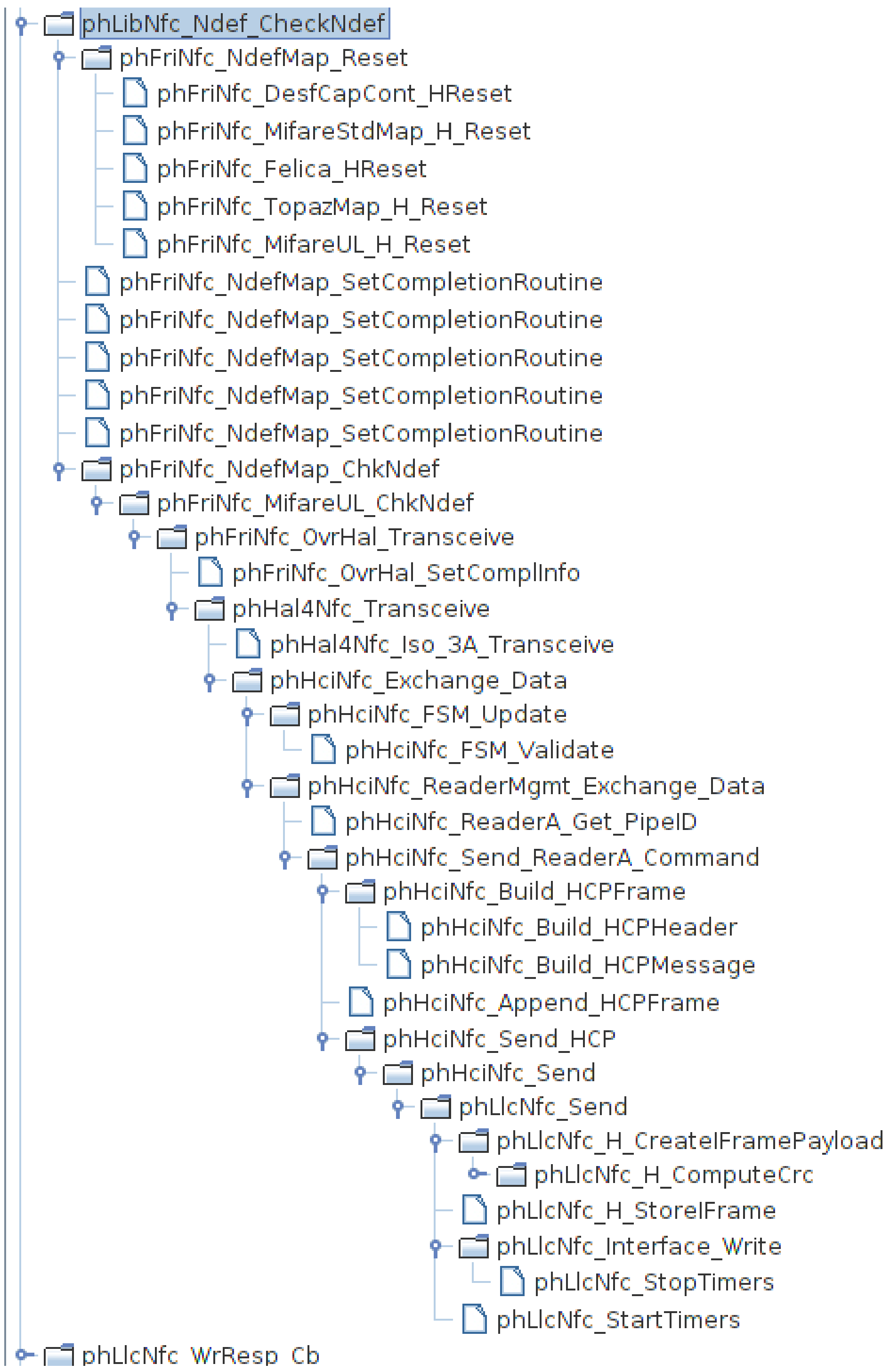
(a) Reading



```
phLlcNfc_RdResp_Cb
phLlcNfc_RdResp_Cb
phLibNfc_RemoteDev_Connect
    phHal4Nfc_Connect
        phHciNfc_Reactivate
            phHciNfc_FSM_Update
                phHciNfc_FSM_Validate
            phHciNfc_ReaderMgmt_Reactivate
                phHciNfc_ReaderA_Get_PipeID
                phHciNfc_Send_RFReader_Command
                    phHciNfc_Build_HCPFrame
                    phHciNfc_Append_HCPFrame
                    phHciNfc_Send_HCP
                        phHciNfc_Send
                            phLlcNfc_Send
                                phLlcNfc_H_CreateIFramePayload
                                    phLlcNfc_H_ComputeCrc
                                phLlcNfc_H_StoreIFrame
                                phLlcNfc_Interface_Write
                                    phLlcNfc_StopTimers
                                phLlcNfc_StartTimers
phLlcNfc_WrResp_Cb
```

(b) Connecting to the card

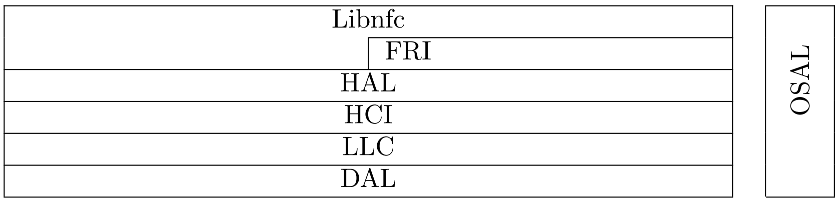
Figure 3.2: Connecting a MIFARE Classic



```
phLibNfc_Ndef_CheckNdef
  phFriNfc_NdefMap_Reset
    phFriNfc_DesfCapCont_HReset
    phFriNfc_MifareStdMap_H_Reset
    phFriNfc_Felica_HReset
    phFriNfc_TopazMap_H_Reset
    phFriNfc_MifareUL_H_Reset
  phFriNfc_NdefMap_SetCompletionRoutine
  phFriNfc_NdefMap_SetCompletionRoutine
  phFriNfc_NdefMap_SetCompletionRoutine
  phFriNfc_NdefMap_SetCompletionRoutine
  phFriNfc_NdefMap_SetCompletionRoutine
  phFriNfc_NdefMap_ChkNdef
    phFriNfc_MifareUL_ChkNdef
      phFriNfc_OvrHal_Transceive
        phFriNfc_OvrHal_SetComplInfo
        phHal4Nfc_Transceive
          phHal4Nfc_Iso_3A_Transceive
          phHciNfc_Exchange_Data
            phHciNfc_FSM_Update
              phHciNfc_FSM_Validate
            phHciNfc_ReaderMgmt_Exchange_Data
              phHciNfc_ReaderA_Get_PipeID
              phHciNfc_Send_ReaderA_Command
                phHciNfc_Build_HCPFrame
                  phHciNfc_Build_HCPHeader
                  phHciNfc_Build_HCPMessage
                phHciNfc_Append_HCPFrame
                phHciNfc_Send_HCP
                  phHciNfc_Send
                    phLlcNfc_Send
                      phLlcNfc_H_CreateIFramePayload
                        phLlcNfc_H_ComputeCrc
                      phLlcNfc_H_StoreIFrame
                      phLlcNfc_Interface_Write
                        phLlcNfc_StopTimers
                      phLlcNfc_StartTimers
phLlcNfc_WrResp_Cb
```

Figure 3.3: NDEF Check while reading a MIFARE Classic

different layers but mainly the DAL is used by the LLC layer. The DAL calls callback functions that are passed through to it. In the function call logs we can see that the callback functions being called are from the LLC Layer. We can update our layer model with this:



3.2 Sent data

We can use what we now know to find suitable functions which we can modify to write the data going through that function to a log. If we are interested in the commands from and responses to apps and are not interested in how this is handled at a lower level, we can log in libnfc functions. The downside of this is that there are a lot of different libnfc-functions which can be called by the application. All these different functions would have to be modified.

If we are interested in the exact data that is sent to and received from the chip we have to log at a lower level. This has the extra advantage that all paths that send data go through the function `phLlcNfc_Interface_Write` and all data is received through the function `phLlcNfc_RdResp_Cb`, as we can see in the function call logs.

Because of that, those two functions were altered, and logging functionality is added to them. In the logs in appendices B and C this extra logging functionality can also be seen. Since we did not log at the lowest level, at the DAL, we can't see any changes this layer makes to the data. To check this, a usb evaluation board (OM5596, [35]) was used. This board was connected to a desktop computer, the data going through the COM-port for the board was logged. Then the PN544 on the board was switched on. We can now compare this log to the log we have for booting the phone. The start of the logs are as following:

	Android	Board
Write	05 F9 04 00 C3 E5	05 F9 02 00 13 B1
Write		05 F9 02 00 13 B1
Read	03	
Read	E6 17 A7	03 E6 17 A7
Write	05 80 81 03 EA 39	05 80 81 03 EA 39
Read	05	
Read	81 81 80 A5 D5	05 81 81 80 A5 D5
Write	06 89 81 02 01 EB 54	06 89 81 02 04 46 03
Write	03 C1 AA F2	
Write		06 92 81 02 01 5F 57
Read		0D 93 81 80 02 03 00 00 45 6C 6C 69 C7 F4
Read	0D	
Read	8A 81 80 61 6E 64 72 6F 69 64 38 48 CC	
Write	05 92 81 14 F9 6D	05 9B 81 14 E7 F1
Write	03C231C0	
Read	03	
Read	C3B8D1	

We see that they are not the same, but similar. We can therefore presume that the DAL does not change the data. In the Android log we see the message "_i2c_read() called to read n bytes". We can find this message being logged in `phDal4Nfc_i2c_read`. This function uses the `read` system call[36] to read data from the file descriptor associated with the NFC controller. This means this is the bottom of the layer model. We also see a `phDal4Nfc_i2c_write` function that uses the `write` system call[37] to write data directly to the controller. We can also add logging functionality here. In appendix C it can be seen that the data logged here is the same as in the LLC layer. This confirms that the DAL does not change the data.

3.2.1 Analyzing the data

Starting from the second write-message we can recognize HCI-packets[27] inside the sent messages. The following is for the android-data:

Write	05 F9 04 00 C3 E5	
Read	03 E6 17 A7	
Write	05 80 **81** **03** EA 39	81: CB=1 (no chaining), pID=1 (administration pipe). 03:Command, ANY_OPEN_PIPE
Read	05 81 **81** **80** A5 D5	81: CB=1, pID=1. 80: Response, ANY_OK
Write	06 89 **81** **02** **01** EB 54	81: CB=1, pID=1. 02: Command, ANY_GET_PARAMETER. 01: Parameter index=1
Write	03 C1 AA F2	(No HCI-packet)
Read	0D 8A **81** **80** **61** **6E** **64** **72** **6F** **69** **64** **38** 48 CC	81: CB=1, pID=1. 80: Response, ANY_OK. 61...38: Parameter value
Write	05 92 **81** **14** F9 6D	81: CB=1, pID=1. 14: Command, ADM_CLEAR_ALL_PIPE
Write	03 C2 31 C0	(No HCI-packet)
Read	03 C3 B8 D1	(No HCI-packet)

So what we see is that in this stage all packets are for the administration pipe, and no chaining is used. The commands being send are:

1. ANY_OPEN_PIPE: The administration pipe is opened. Response: OK
2. ANY_GET_PARAMETER: Parameter 1 is requested. The administration pipe is connected to the administration gate of the host controller. For this gate the requested registry, with ID=1, is SESSION_IDENTITY. This registry contains a session identifier. This can be used to see whether the host configuration changed. Response: OK, SESSION_IDENTITY=61 6E 64 72 6F 69 64 38
3. ADM_CLEAR_ALL_PIPE: Delete all dynamic pipes, close all static pipes and set all registries for the static pipes to their default value. Response: Not received.

What can be seen is that every sent message, after the first two, is built up as follows: two bytes - HCI Packet - two bytes. The first byte always contains the length of the message, excluding this length byte itself. This is a possible explanation for why in Android first one byte is read and then, when the number of bytes to read is known, it reads the rest of the message. In the function call logs it can be seen that the function `phLlcNfc_H_ComputeCrc` is called. We can call this on the sent messages, excluding the last two bytes:

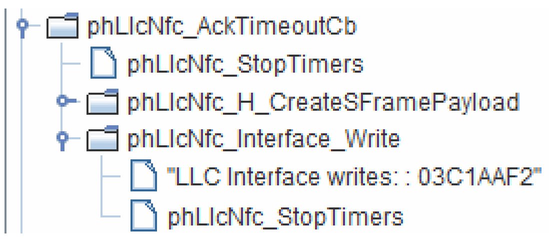
```
uint8_t msg[] = {(uint8_t)0x05, (uint8_t)0xF9, (uint8_t)0x04,
          (uint8_t)0x00, (uint8_t)0, (uint8_t)0};
uint8_t len = msg[0];
phLlcNfc_H_ComputeCrc(msg, len -1, &msg[len -1], &msg[len]);
printArray(msg);
```

This gives as output:

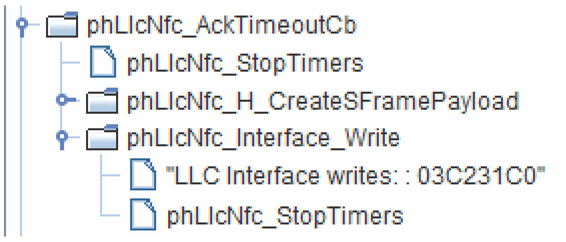
```
05 F9 04 00 C3 E5
```

This matches the message above, so the last two bytes are the CRC. The same applies to the other messages.

There are, in the snippet above, two write-messages without content. When looking to the full log, including function calls we see the following:



```
phLlcNfc_AckTimeoutCb
    phLlcNfc_StopTimers
    phLlcNfc_H_CreateSFramePayload
    phLlcNfc_Interface_Write
        "LLC Interface writes: : 03C1AAF2"
        phLlcNfc_StopTimers
```



```
phLlcNfc_AckTimeoutCb
    phLlcNfc_StopTimers
    phLlcNfc_H_CreateSFramePayload
    phLlcNfc_Interface_Write
        "LLC Interface writes: : 03C231C0"
        phLlcNfc_StopTimers
```

Both messages are sent from an ACK-timeout callback. Also note that the second bit of the second byte is 1 if and only if the message is an ACK-timeout message like these. We don't see any ACK messages being sent, but we can see responses missing in the cases that an ACK-timeout message is sent. So the responses act as acknowledgments. The first bit of the second byte is always 1. For the remaining 6 bits of the second byte these bytes are shown in their binary representation:

Write	11 111 001	Does not comply
Read	11 100 110	Does not comply
Write	10 000 000	
Read	10 000 001	
Write	10 001 001	
Write	11 000 001	ACK Timeout
Read	10 001 010	
Write	10 010 010	
Write	11 000 010	ACK Timeout
Read	11 000 011	ACK Timeout
Write	10 100 100	
Read	11 000 111	ACK Timeout

The byte is split in one pair and two triples of bits. Both bits of this pair we already discussed. The second triple appears to be a message identifier, and the first triple is the identifier of the message it is a response to. For an ACK the first triple is always 000, and the second triple is the identifier of the message that timed out.

Chapter 4

NFC Logging app

4.1 Functionality

The goal of the project is to create an app to log the NFC data going from the software to the NFC Controller. We have seen that the data that is sent internally consists of HCI-packets with a header and trailer added by the LLC-layer, and that it does not contain the exact same data that is sent with radio signals. For example, if the phone is emulating a card, anti-collision and authentication protocols are handled by the NFC controller. However, if a reader reads data from the emulated card, this data does have to be sent from the CPU to the controller.

This means that the app will not log all data that is sent through the air. To retrieve this data other methods are available[6, 7].

The app will be usable without having to install a modified Android build. It can install itself on a rooted Nexus S device.

The logs are saved to a file. The app does not interpret the data. To know what the HCI-packets mean a separate application could be developed, or one could interpret the data by hand using the available HCI documentation[27].

4.2 Problems to overcome and decisions to be made

4.2.1 Layer to log

As we have seen there are different layers to log. A decision had to be made about where the logging is taking place. At the highest layer the exact commands from the apps using NFC can be seen. At lower layers their encoding in HCI-messages can be seen, and the header and trailer from the LLC-layer can also be logged in those layers. Since there are a lot of different functions that can be called in the higher layers it is very well possible that apps use other functions than what we have seen in the function call logs. At the LLC layer everything is done using the `phLlcNfc_Interface_Write` function for writing to the controller and all data is received using the function `phLlcNfc_RdResp_Cb`. Also since the commands at higher layers are encoded in HCI-messages this layer gives the most information. Therefore this is the layer that will be logged.

4.2.2 Communication between libnfc-nxp and app

The logs should be written to a file. This can be done directly in the NFC-library or in the app. For logging in the app, there should be a communication channel from libnfc-nxp to the app. The app has then control over the data and can decide whether to store it or not, store it to a user specified file, and display it on the screen. If the NFC-library logs directly to a file it will be at a predefined location, or we need a communication channel from the app to the library. We have less control over the data in the app. An advantage of this tactic is that the app does not have to be running while the logging is happening.

In Android we can create a service that runs in the background [38]. We can use this to have our app running in the background while logging. Therefore we will create a communication channel from libnfc-nxp to the app and let the app control the data.

For the communication we will use a *named pipe*. Once created, this pipe can be used as a regular file. The library writes to the pipe and the app reads from it. Such pipes cannot be created in a regular app. We will need root access to do so. With root access we can run shell commands using root privileges. This way we can run the `mkfifo` [39] command for creating a named pipe.

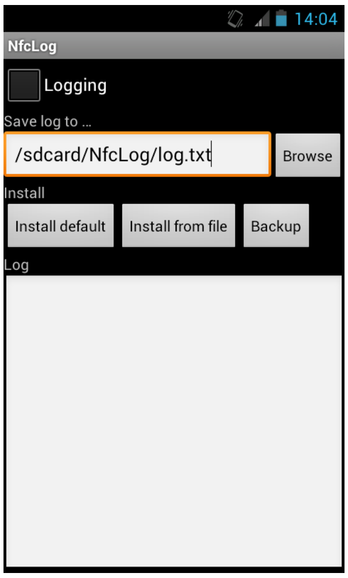


Figure 4.1: Screenshot of the logging app

4.2.3 Using the app with unmodified Android build

Only a small part of Android is changed to make the logging possible. If the user of the phone uses the same version of Android as the one where the changes are made, it is therefore not needed to install an entire Android build on the phone. Only the changed parts have to be overwritten. These files are `/system/lib/libnfc.so`, `/system/lib/libnfc_jni.so` and `/system/app/Nfc.apk`. If the Android build of the user is different from the build in which logging is added, NFC may stop working when these files are overwritten, because the files on which the NFC parts depend are different.

Normal apps do not have access to the files that need to be overwritten, because of a couple of restrictions. First of all, `/system` is *read-only*, so we can't write to it. Also, access to the files to be overwritten is restricted to read-only for non-root users. With root access we can execute the `mount` [40] command to remount `/system` read-write. We can use `chmod` [41] to give all users write-access to the files. This way we can overwrite the system files with our modified versions, and not a full Android installation is needed. After the files are overwritten, a reboot is required for the changes to take effect. This can be achieved with the `reboot` [42] command.

4.3 The app

A screenshot of the app can be seen in fig. 4.1. The button *Install default* installs the system files that are packaged with the app. The *Backup* button copies the current system files to a location chosen by the user. The button *Install from file* installs system files from a user-chosen location. This can be used to restore a backup or to install a newer version of the files than the ones packaged with the app. If the *Logging*-checkbox is checked a service is started. This service creates a named pipe at `/system/nfclogpipe`. It then starts reading this named pipe. Everything that is received on this pipe is written to the file specified in the textbox with the *Save log to...* label. If the app is active it also writes the logs to the *Log*-text area. The service keeps running when the app is closed. If the checkbox is unchecked, the service is given a stop command, on which the service stops reading, removes the named pipe and the service stops running.

The `phLlcNfc_Interface_Write` and `phLlcNfc_RdResp_Cb` functions have been modified to write their data to `/system/nfclogpipe`. If this pipe does not exist the error is ignored, which means that no logging is happening when the reader service has not created the named pipe.

4.4 Security implications

We have seen that it is possible to modify the NFC code in Android. We have access to the data and are able to write this to a Android log, file or application. Since we can change the functions that pass through this data, an attacker should be able to make alterations to this data. This can be a problem for payment applications, for example, if such an app is not secured against this type of attack. An attacker may then be able to change transactions to make it possible for him to, for example, pay less than he should, or receive more than he should or make the money go to another account.

We have also seen that on a rooted phone it is possible to install the modified parts without having to install an entire Android image. This means that an attacker can do the things mentioned above with transactions made by others, if he has root access on their phone. This can be because the phone owner has willingly rooted his phone. But there are also vulnerabilities known that can be used to get root access (e.g. [28], [29]). Using methods like these a malicious app could gain root access without any action from the user and without him noticing it.

Chapter 5

MIFARE Classic example

As an example of how the app can be used to find sensitive information we will log the use of an app that reads an OV-Chipkaart. The OV-Chipkaart is a MIFARE Classic card. Every sector of the memory of these cards is protected using two keys, key A and key B. Different sectors can have different keys. The HCI standard used by the PN544 has built in functionality for working with such cards, including authentication[27]. This means that the keys will probably be sent to the NFC controller, and that they will be logged by our application. The OV-Chipkaart app requires these keys for the specific card to be known. To obtain these keys the MIFARE Offline Cracker GUI (MFOC GUI) [43] is used. This application uses an NFC reader to communicate with a MIFARE Classic card, and one can crack the keys of a card with one press on a button. The cracking will take a few hours, however.

Apart from these cracked keys, there are also some default keys used in MIFARE Classic cards. These include `A0A1A2A3A4A5`, `000000000000`, and `FFFFFFFFFFFF`.

The first time the OV-Chipkaart application is used it will try every key it knows on each sector it wants to read. These keys include both the default keys and the keys retrieved using the MFOC GUI. So when we log this, it will not give any information of the keys that are actually used. However, the application will save the keys it eventually used. For later runs it will only use the keys that worked.

We can activate our logging app while we use the OV-Chipkaart app to read a card for a run with saved keys. We will then have a log of the communication between CPU and NFC controller. We expect to see the keys found obtained by the MFOC GUI in this log. When searching for these keys, the following messages were found:

11 A9 85 21 60 01 7C D0 00 F0 **A0 A1 A2 A3 A4 A5** DE E5	default key
11 A9 85 21 60 01 7C D0 00 F0 **A0 A1 A2 A3 A4 A5** DE E5	default key
11 8D 85 21 60 00 7C D0 00 F0 **00 00 00 00 00 00** AD F5	default key
11 9F 85 21 60 00 7C D0 00 F0 **00 00 00 00 00 00** 1D C7	default key
⋮	
11 A0 85 21 60 58 7C D0 00 F0 **33 28 B4 96 E7 6D** C1 28	cracked key
11 BB 85 21 60 58 7C D0 00 F0 **33 28 B4 96 E7 6D** 29 03	cracked key
⋮	
11 96 85 21 60 F0 7C D0 00 F0 **5E 51 FA 2D F6 AC** 0E 05	cracked key
11 B2 85 21 60 F0 7C D0 00 F0 **5E 51 FA 2D F6 AC** 6E 60	cracked key
⋮	
11 A0 85 21 60 A0 7C D0 00 F0 **FF 0F 97 21 31 D4** 08 B1	cracked key
11 A9 85 21 60 A0 7C D0 00 F0 **FF 0F 97 21 31 D4** 50 A8	cracked key
⋮	
11 84 85 21 60 B0 7C D0 00 F0 **BE 64 7C 70 2A 65** 13 3D	cracked key
11 8D 85 21 60 B0 7C D0 00 F0 **BE 64 7C 70 2A 65** 4B 24	cracked key
⋮	
11 A0 85 21 60 C0 7C D0 00 F0 **D3 8A 0E 54 2C F6** 22 0F	cracked key
11 A9 85 21 60 C0 7C D0 00 F0 **D3 8A 0E 54 2C F6** 7A 16	cracked key
11 B2 85 21 60 F0 7C D0 00 F0 **5E 51 FA 2D F6 AC** 6E 60	cracked key
11 84 85 21 60 F0 7C D0 00 F0 **5E 51 FA 2D F6 AC** BE 37	cracked key
⋮	
11 9F 85 21 60 C0 7C D0 00 F0 **D3 8A 0E 54 2C F6** AA 41	cracked key
11 B2 85 21 60 C0 7C D0 00 F0 **D3 8A 0E 54 2C F6** 92 3D	cracked key
⋮	
11 A0 85 21 60 B0 7C D0 00 F0 **BE 64 7C 70 2A 65** 73 58	cracked key
11 BB 85 21 60 B0 7C D0 00 F0 **BE 64 7C 70 2A 65** 9B 73	cracked key
⋮	
11 B2 85 21 60 C0 7C D0 00 F0 **D3 8A 0E 54 2C F6** 92 3D	cracked key
11 BB 85 21 60 C0 7C D0 00 F0 **D3 8A 0E 54 2C F6** CA 24	cracked key
⋮	
11 BB 85 21 60 80 7C D0 00 F0 **63 8E 15 83 07 6F** C0 C8	cracked key
11 84 85 21 60 80 7C D0 00 F0 **63 8E 15 83 07 6F** 48 86	cracked key
⋮	
11 A9 85 21 60 90 7C D0 00 F0 **AC 8A 5B 4B 97 1B** 8C C4	cracked key
11 96 85 21 60 90 7C D0 00 F0 **AC 8A 5B 4B 97 1B** 04 8A	cracked key
⋮	
11 8D 85 21 60 A0 7C D0 00 F0 **FF 0F 97 21 31 D4** 30 CD	cracked key
11 BB 85 21 60 A0 7C D0 00 F0 **FF 0F 97 21 31 D4** E0 9A	cracked key
⋮	
11 B2 85 21 60 01 7C D0 00 F0 **A0 A1 A2 A3 A4 A5** 36 CE	default key
11 B2 85 21 60 01 7C D0 00 F0 **A0 A1 A2 A3 A4 A5** 36 CE	default key

Some of the messages are repeated many times, with only changes to the LLC part. If there are more than two copies of a message, these copies are left out in the table above. `7C D0 00 F0` is the Unique Identifier (UID) of the card. The parts in bold font are keys of the card. Not all keys are found. All of these keys are of type A. So no keys of type B are found. Also not all keys of type A are found.

All messages of which the HCI part has the form `85 21 60 xx 7C D0 00 F0 xx xx xx xx xx xx` contain a key in the last 6 bytes. In this example we already knew the keys. But if we found a pattern like this in a payment application we would be able to obtain the keys for this application, even if they were secret. Without knowing the keys beforehand other ways have to be used to find such patterns.

Chapter 6

Concluding

6.1 Conclusion

An app has been created to log the data sent to and received from the NFC controller. Logging can be done at different layers. Logging at the LLC layer logs the HCI packets, which contain the commands for the controller, and the LLC header and trailer, which give information about ACK timeouts, and which message is a response to which.

The logging is done by modifying the source code for libnfc-nxp. Installation of the modified library can happen on a rooted phone. If the Android build on the phone is the same as the one to which the logging is added, it is not necessary to install an entire custom Android build. Installation of such a modified library can also be done by malicious apps if they have root access, either granted by the owner of the phone or through a security exploit. A modified version of libnfc-nxp could pose a security threat for insecure apps.

6.2 Future work

The security implications described in here are not vulnerabilities themselves, but they are a property of Android that has to be taken into account when developing an application in which security is important. Therefore apps like that should be assessed on their security. The app presented here can provide a tool in doing so.

The application does not interpret any data, it just writes it to a file. This data has to be interpreted to be of use. This can be done manually, but also by using software. Developing this software is something that still has to be done.

Bibliography

[1] Google. Nexus s. Retrieved June 4, 2012, from http://www.android.com/devices/detail/nexus-s, 2012.

[2] Google. Galaxy nexus. Retrieved June 4, 2012, from http://www.android.com/devices/detail/galaxy-nexus, 2012.

[3] HTC. Htc one x. Retrieved June 11, 2012, from http://www.htc.com/www/smartphones/htc-one-x/#specs, 2012.

[4] phoneslimited. Looking into the connectivity on the samsung galaxy s3. http://www.newsbroadcast.co.uk/31552/looking-into-the-connectivity-on-the-samsung-galaxy-s3, June 2012.

[5] Nokia. Detailed specifications for the nokia 808 pureview. Retrieved June 11, 2012, from http://www.nokia.com/global/products/phone/808pureview/specifications/, 2012.

[6] G.P. Hancke. Practical eavesdropping and skimming attacks on high-frequency rfid tokens. *Journal of Computer Security*, 19(2):259–288, 2011.

[7] Lishoy Francis, Gerhard Hancke, Keith Mayes, and Konstantinos Markantonakis. Practical nfc peer-to-peer relay attack using mobile phones. In Siddika Ors Yalcin, editor, *Radio Frequency Identification: Security and Privacy Issues*, volume 6370 of *Lecture Notes in Computer Science*, pages 35–49. Springer Berlin / Heidelberg, 2010. 10.1007/978-3-642-16822-2_4.

[8] Xavier Ducrohet. Android 2.3 platform and updated sdk tools. http://android-developers.blogspot.nl/2010/12/android-23-platform-and-updated-sdk.html, December 2010.

[9] Google. Google wallet. Retrieved June 11, 2012, from http://www.google.com/wallet/, 2012.

[10] Visa. Visa Certifies Smartphones for Use as Visa Mobile Payment Devices. Press Release, January 2012. http://investor.visa.com/phoenix.zhtml?c=215693&p=irol-newsArticle&id=1646350.

[11] Isis. Isis. Retrieved June 11, 2012, from http://www.paywithisis.com/, 2012.

[12] MasterCard. Paypass cell phone trial. Retrieved June 11, 2012, from http://www.mastercard.com/us/paypass/phonetrial/index.html, 2012.

[13] Balaban, Dan. Dutch banks and telcos to move forward on m-payment project. http://nfctimes.com/news/dutch-banks-and-telcos-move-forward-m-payment-project, July 2010.

[14] Google. Android open source project. Retrieved June 4, 2012, from http://source.android.com/.

[15] F.D. Garcia, G. de Koning Gans, R. Muijrers, P. Van Rossum, R. Verdult, R. Schreur, and B. Jacobs. Dismantling mifare classic. In *13th European Symposium on Research in Computer Security (ESORICS 2008)*, pages 97–114. Springer Berlin / Heidelberg, 2008.

[16] A. Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, 9(1):5–38, 1883.

[17] ISO. Iso/iec 14443-3: Identification cards contactless integrated circuit(s) cards - proximity cards part 1: Physical characteristics, 2000.

[18] ISO. Iso/iec 14443-3: Identification cards contactless integrated circuit(s) cards - proximity cards part 2: Radio frequency power and signal interface, 2001.

[19] ISO. Iso/iec 14443-3: Identification cards contactless integrated circuit(s) cards - proximity cards part 3: Initialization and anticollision, 2001.

[20] ISO. Iso/iec 14443-4: Identification cards contactless integrated circuit(s) cards - proximity cards part 4: Transmission protocol, 2001.

[21] JSA. Specification of implmentation for integrated circuit(s) cards - part 4 : High speed proximity cards, July 2005.

[22] NFC Forum. About nfc, 2011.

[23] NFC Forum. Nfc data exchange format (ndef), technical specification, 2006. http://www.nfc-forum.org/specs/spec_list/.

[24] NFC Forum. Simple ndef exchange protocol, technical specification, 2011. http://www.nfc-forum.org/specs/spec_list/.

[25] NXP. Google and NXP integrate NFC in Android 2.3. Press Release, December 2010. http://www.nxp.com/news/press-releases/2010/12/google-and-nxp-integrate-nfc-in-android-2-3.html.

[26] NXP. NXPs NFC solution implemented in Galaxy Nexus from Google. Press Release, November 2011. http://www.nxp.com/news/press-releases/2011/11/nxp-nfc-solution-implemented-in-galaxy-nexus-from-google.html.

[27] ETSI. Smart cards; uicc - contactless front-end (clf) interface; host controller interface (hci)(release 7). http://www.etsi.org/deliver/etsi_ts/102600_102699/102622/07.05.00_60/ts_102622v070500p.pdf.

[28] Jon Larimer and Jon Oberheide. Levitator. Retrieved June 11, 2012, from http://jon.oberheide.org/files/levitator.c, 2011.

[29] ieftm. Revolutionary - zergrush local root 2.2/2.3. Retrieved June 11, 2012, from http://forum.xda-developers.com/showthread.php?t=1296916, 2011.

[30] X. Jiang. Gingermaster: First android malware utilizing a root exploit on android 2.3 (gingerbread). Retreived June 11, 2012, from http://www.csc.ncsu.edu/faculty/jiang/GingerMaster/, August 2011.

[31] Dimitri van Heesch. Doxygen. Retrieved June 4, 2012, from http://www.stack.nl/~dimitri/doxygen/, May 2012.

[32] Wikipedia. Operating system abstraction layer — wikipedia, the free encyclopedia. Retrieved June 7, 2012, from http://en.wikipedia.org/w/index.php?title=Operating_system_abstraction_layer&oldid=485346908, 2012.

[33] Wikipedia. Data access layer — wikipedia, the free encyclopedia. Retrieved June 7, 2012, from http://en.wikipedia.org/w/index.php?title=Data_access_layer&oldid=481262985, 2012.

[34] Wikipedia. Database abstraction layer — wikipedia, the free encyclopedia. Retrieved June 7, 2012, from http://en.wikipedia.org/w/index.php?title=Database_abstraction_layer&oldid=468680557, 2012.

[35] NXP. Nxp nfc controller pn544 for mobile phones and portable equipment. Data sheet, February 2010. http://www.nxp.com/documents/leaflet/75016890.pdf.

[36] read(2) - linux programmer's manual. Linux man page, February 2009. Retrieved June 11, 2012, from http://www.kernel.org/doc/man-pages/online/pages/man2/read.2.html.

[37] write(2) - linux programmer's manual. Linux man page, August 2010. Retrieved June 11, 2012, from http://www.kernel.org/doc/man-pages/online/pages/man2/write.2.html.

[38] Google. Services. Android developer's guide, June 2009. Retrieved June 11, 2012, from http://developer.android.com/guide/topics/fundamentals/services.html.

[39] David MacKenzie. mkfifo(1) - linux man page. Linux man page, 2010. Retrieved June 11, 2012, from http://linux.die.net/man/1/mkfifo.

[40] mount(8) - linux man page. Linux man page. Retrieved June 11, 2012, from `http://linux.die.net/man/8/mount`.

[41] David MacKenzie and Jim Meyering. chmod(1) - linux man page. Linux man page, 2010. Retrieved June 11, 2012, from `http://linux.die.net/man/1/chmod`.

[42] Scott James Remnant. Reboot(8) - linux man page. Linux man page, 2009. Retrieved June 11, 2012, from `http://linux.die.net/man/8/reboot`.

[43] Mifare offline cracker gui. [Software] Retreived June 23, 2012, from `http://www.huuf.info/OV/`.

Appendix A

Step by step instructions to log NFC communication

To enable logging on a Google Nexus S the following steps have to be followed:

1. Android has to be built from source. The Android Source website [14] explains how to download the source and build Android.
2. Change the library found in `external/libnfc-nxp/` of the source to write the information that has to be logged to `/system/nfclogpipe`.
3. Build Android and install it on the phone
4. Install the application `NFCLogApp.apk` on the phone
5. In the app, set the log file location to the preferred location
6. Enable logging
7. Execute the NFC actions you want to log
8. Disable logging
9. The logs can now be found in the previously specified file

Note that once Android has been built and installed on the phone, only the files `out/target/product/crespo/` `out/target/product/crespo/system/lib/libnfc_jni.so` and `out/target/product/crespo/system/app/Nfc.apk` have to be copied to the phone, when changes are made to the NFC library and no other parts have been changed. To do so, follow these steps:

1. Copy the mentioned files to the USB-storage of the phone
2. Use the "Install from file" button in the app to select the location of the files and install them to the right location on the phone.

Appendix B

Log of booting the phone

The following log is recorded during the booting of the phone and shows functions being called in libnfc-nxp and data being logged in the `phLlcNfc_Interface_Write` and `phLlcNfc_RdResp_Cb` functions. Only the start is included here because of the length of the log.

```
D/NFC     (  398): Entering phLibNfc_Mgt_ConfigureDriver.
D/NFC     (  398): Exit function
D/NFC     (  398): Entering phLibNfc_Mgt_Initialize.
D/NFC     (  398): Entering phLibNfc_UpdateNextState.
D/NFC     (  398): Exit function
D/NFC-HCI (  398): Entering phHal4Nfc_Open.
D/NFC-HCI (  398): Entering dlopen_firmware.
D/NFC-HCI (  398): Exit function
D/        (  398): Entering phHal4Nfc_Configure_Layers.
D/        (  398): Exit function
D/NFC-HCI (  398): Entering phHciNfc_Initialise.
D/        (  398): Entering phLlcNfc_Register.
D/NFC-LLC (  398): Entering phLlcNfc_Interface_Register.
D/NFC-LLC (  398): Exit function
D/        (  398): Exit function
D/NFC-HCI (  398): Entering phHciNfc_FSM_Update.
D/NFC-HCI (  398): Entering phHciNfc_FSM_Validate.
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/        (  398): Entering phLlcNfc_Init.
D/        (  398): Entering phLlcNfc_H_Frame_Init.
D/        (  398): Exit function
D/NFC-LLC (  398): Entering phLlcNfc_Interface_Init.
D/NFC-LLC (  398): Exit function
D/        (  398): Entering phLlcNfc_TimerInit.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_CreateTimers.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_CreateUFramePayload.
D/        (  398): Entering phLlcNfc_H_ComputeCrc.
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Exit function
D/        (  398): Exit function
D/NFC-LLC (  398): Entering phLlcNfc_Interface_Write.
D/libnfc-nxp-log(  398): NFC-Data LLC Interface writes: : 05F90400C3E5
D/        (  398): Entering phLlcNfc_StopTimers.
```

```
D/        (  398): Exit function
D/NFC-LLC (  398): Exit function
D/        (  398): Entering phLlcNfc_StartTimers.
D/        (  398): Exit function
D/        (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/        (  398): Entering phLibNfc_Ndef_Init.
D/        (  398): Exit function
D/NFC     (  398): Exit function
D/NFC-LLC (  398): Entering phLlcNfc_WrResp_Cb.
D/NFC-LLC (  398): Entering phLlcNfc_Interface_Read.
D/NFC-LLC (  398): Exit function
D/NFC-LLC (  398): Exit function
D/NFC-LLC (  398): Entering phLlcNfc_RdResp_Cb.
D/libnfc-nxp-log(  398): NFC-Data Receive callback: : 03
D/NFC-LLC (  398): Entering phLlcNfc_Interface_Read.
D/NFC-LLC (  398): Exit function
D/NFC-LLC (  398): Exit function
D/NFC-LLC (  398): Entering phLlcNfc_RdResp_Cb.
D/libnfc-nxp-log(  398): NFC-Data Receive callback: : E617A7
D/        (  398): Entering phLlcNfc_H_ChangeState.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_ComputeCrc.
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_ProRecvFrame.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_ProcessUFrame.
D/        (  398): Entering phLlcNfc_StopTimers.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_ResetFrameInfo.
D/        (  398): Exit function
D/NFC-HCI (  398): Entering phHciNfc_Notify_Event.
D/NFC-HCI (  398): Entering phHciNfc_Resume_Sequence.
D/NFC-HCI (  398): Entering phHciNfc_Initialise_Sequence.
D/NFC-HCI (  398): Entering phHciNfc_Admin_Initialise.
D/NFC-HCI (  398): Entering phHciNfc_Allocate_Resource.
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Entering phHciNfc_Allocate_Resource.
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Entering phHciNfc_Open_Pipe.
D/NFC-HCI (  398): Entering phHciNfc_Send_Generic_Cmd.
D/NFC-HCI (  398): Entering phHciNfc_Build_HCPFrame.
D/NFC-HCI (  398): Entering phHciNfc_Build_HCPHeader.
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Entering phHciNfc_Build_HCPMessage.
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Entering phHciNfc_Send_HCP.
D/NFC-HCI (  398): Entering phHciNfc_Send.
D/        (  398): Entering phLlcNfc_Send.
D/        (  398): Entering phLlcNfc_H_CreateIFramePayload.
D/        (  398): Entering phLlcNfc_H_ComputeCrc.
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
```

```
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Exit function
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_StoreIFrame.
D/        (  398): Exit function
D/NFC-LLC (  398): Entering phLlcNfc_Interface_Write.
D/libnfc-nxp-log(  398): NFC-Data LLC Interface writes: : 05808103EA39
D/        (  398): Entering phLlcNfc_StopTimers.
D/        (  398): Exit function
D/NFC-LLC (  398): Exit function
D/        (  398): Entering phLlcNfc_StartTimers.
D/        (  398): Exit function
D/        (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/        (  398): Exit function
D/        (  398): Exit function
D/NFC-LLC (  398): Exit function
D/NFC-LLC (  398): Entering phLlcNfc_WrResp_Cb.
D/NFC-LLC (  398): Entering phLlcNfc_Interface_Read.
D/NFC-LLC (  398): Exit function
D/NFC-HCI (  398): Entering phHciNfc_Send_Complete.
D/NFC-HCI (  398): Entering phHciNfc_Receive.
D/        (  398): Entering phLlcNfc_Receive.
D/NFC-LLC (  398): Entering phLlcNfc_Interface_Read.
D/NFC-LLC (  398): Exit function
D/        (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-LLC (  398): Exit function
D/NFC-LLC (  398): Entering phLlcNfc_RdResp_Cb.
D/libnfc-nxp-log(  398): NFC-Data Receive callback: : 05
D/NFC-LLC (  398): Entering phLlcNfc_Interface_Read.
D/NFC-LLC (  398): Exit function
D/NFC-LLC (  398): Exit function
D/NFC-LLC (  398): Entering phLlcNfc_RdResp_Cb.
D/libnfc-nxp-log(  398): NFC-Data Receive callback: : 818180A5D5
D/        (  398): Entering phLlcNfc_H_ChangeState.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_ComputeCrc.
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_ProRecvFrame.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_ProcessIFrame.
```

```
D/        (  398): Entering phLlcNfc_H_UpdateIFrameList.
D/        (  398): Entering phLlcNfc_H_IFrameList_Peek.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_DeleteIFrame.
D/        (  398): Exit function
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_StopTimers.
D/        (  398): Exit function
D/NFC-LLC (  398): Entering phLlcNfc_H_SendInfo.
D/NFC-LLC (  398): Entering phLlcNfc_Interface_Read.
D/NFC-LLC (  398): Exit function
D/NFC-HCI (  398): Entering phHciNfc_Receive_Complete.
D/NFC-HCI (  398): Entering phHciNfc_Process_HCP.
D/NFC-HCI (  398): Entering phHciNfc_Receive_HCP.
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Entering phHciNfc_Process_Response.
D/NFC-HCI (  398): Entering phHciNfc_Recv_Admin_Response.
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Entering phHciNfc_Reset_Pipe_MsgInfo.
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Entering phHciNfc_Resume_Sequence.
D/NFC-HCI (  398): Entering phHciNfc_Initialise_Sequence.
D/NFC-HCI (  398): Entering phHciNfc_Admin_Initialise.
D/NFC-HCI (  398): Entering phHciNfc_Send_Generic_Cmd.
D/NFC-HCI (  398): Entering phHciNfc_Build_HCPFrame.
D/NFC-HCI (  398): Entering phHciNfc_Build_HCPHeader.
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Entering phHciNfc_Build_HCPMessage.
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Entering phHciNfc_Send_HCP.
D/NFC-HCI (  398): Entering phHciNfc_Send.
D/        (  398): Entering phLlcNfc_Send.
D/        (  398): Entering phLlcNfc_H_CreateIFramePayload.
D/        (  398): Entering phLlcNfc_H_ComputeCrc.
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Exit function
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_StoreIFrame.
D/        (  398): Exit function
D/NFC-LLC (  398): Entering phLlcNfc_Interface_Write.
D/libnfc-nxp-log(  398): NFC-Data LLC Interface writes: : 0689810201EB54
D/        (  398): Entering phLlcNfc_StopTimers.
D/        (  398): Exit function
D/NFC-LLC (  398): Exit function
D/        (  398): Entering phLlcNfc_StartTimers.
D/        (  398): Exit function
D/        (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
```

```
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-LLC (  398): Exit function
D/NFC-LLC (  398): Entering phLlcNfc_Interface_Read.
D/NFC-LLC (  398): Exit function
D/        (  398): Entering phLlcNfc_StartTimers.
D/        (  398): Exit function
D/        (  398): Exit function
D/        (  398): Exit function
D/NFC-LLC (  398): Exit function
D/NFC-LLC (  398): Entering phLlcNfc_WrResp_Cb.
D/NFC-LLC (  398): Entering phLlcNfc_Interface_Read.
D/NFC-LLC (  398): Exit function
D/NFC-HCI (  398): Entering phHciNfc_Send_Complete.
D/NFC-HCI (  398): Entering phHciNfc_Receive.
D/        (  398): Entering phLlcNfc_Receive.
D/NFC-LLC (  398): Entering phLlcNfc_Interface_Read.
D/NFC-LLC (  398): Exit function
D/        (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-LLC (  398): Exit function
```

Appendix C

Log of reading a MIFARE Classic card

The following log shows the start process of Reading a MIFARE Classic card. In addition to the function calls and the data logged in phLlcNfc_Interface_Write and phLlcNfc_RdResp_Cb it also shows the data that is logged in phDal4Nfc_i2c_read and phDal4Nfc_i2c_write.

```
D/libnfc-nxp-log(  398): NFC-Data i2c_read: 06
D/NFC-LLC (  398): Entering phLlcNfc_RdResp_Cb.
D/libnfc-nxp-log(  398): NFC-Data Receive callback: : 06
D/NFC-LLC (  398): Entering phLlcNfc_Interface_Read.
D/NFC-LLC (  398): Exit function
D/NFC-LLC (  398): Exit function
D/libnfc-nxp-log(  398): NFC-Data i2c_read: 97855000A7AB
D/NFC-LLC (  398): Entering phLlcNfc_RdResp_Cb.
D/libnfc-nxp-log(  398): NFC-Data Receive callback: : 97855000A7AB
D/        (  398): Entering phLlcNfc_H_ChangeState.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_ComputeCrc.
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_ProRecvFrame.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_ProcessIFrame.
D/        (  398): Entering phLlcNfc_H_UpdateIFrameList.
D/        (  398): Exit function
D/NFC-LLC (  398): Entering phLlcNfc_H_SendInfo.
D/NFC-LLC (  398): Entering phLlcNfc_Interface_Read.
D/NFC-LLC (  398): Exit function
D/NFC-HCI (  398): Entering phHciNfc_Notify_Event.
D/NFC-HCI (  398): Entering phHciNfc_Receive_Complete.
D/NFC-HCI (  398): Entering phHciNfc_Process_HCP.
D/NFC-HCI (  398): Entering phHciNfc_Receive_HCP.
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Entering phHciNfc_Process_Event.
D/NFC-HCI (  398): Entering phHciNfc_Recv_ReaderA_Event.
D/NFC-HCI (  398): Entering phHciNfc_Notify_Event.
D/NFC-HCI (  398): Entering phHciNfc_FSM_Update.
```

```
D/NFC-HCI (  398): Entering phHciNfc_FSM_Validate.
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Entering phHciNfc_Resume_Sequence.
D/NFC-HCI (  398): Entering phHciNfc_Info_Sequence.
D/NFC-HCI (  398): Entering phHciNfc_ReaderMgmt_Info_Sequence.
D/NFC-HCI (  398): Entering phHciNfc_ReaderA_Info_Sequence.
D/NFC-HCI (  398): Entering phHciNfc_Send_Generic_Cmd.
D/NFC-HCI (  398): Entering phHciNfc_Build_HCPFrame.
D/NFC-HCI (  398): Entering phHciNfc_Build_HCPHeader.
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Entering phHciNfc_Build_HCPMessage.
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Entering phHciNfc_Send_HCP.
D/NFC-HCI (  398): Entering phHciNfc_Send.
D/        (  398): Entering phLlcNfc_Send.
D/        (  398): Entering phLlcNfc_H_CreateIFramePayload.
D/        (  398): Entering phLlcNfc_H_ComputeCrc.
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Exit function
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_StoreIFrame.
D/        (  398): Exit function
D/NFC-LLC (  398): Entering phLlcNfc_Interface_Write.
D/libnfc-nxp-log(  398): NFC-Data LLC Interface writes: : 06BB8502029570
D/        (  398): Entering phLlcNfc_StopTimers.
D/        (  398): Exit function
D/NFC-LLC (  398): Exit function
D/        (  398): Entering phLlcNfc_StartTimers.
D/        (  398): Exit function
D/        (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-LLC (  398): Exit function
D/NFC-LLC (  398): Entering phLlcNfc_Interface_Read.
D/NFC-LLC (  398): Exit function
D/        (  398): Entering phLlcNfc_StartTimers.
D/        (  398): Exit function
D/        (  398): Exit function
D/        (  398): Exit function
```

```
D/NFC-LLC (  398): Exit function
D/libnfc-nxp-log(  398): NFC-Data i2c_write: 06BB8502029570
D/libnfc-nxp-log(  398): NFC-Data i2c_read: 0C
D/NFC-LLC (  398): Entering phLlcNfc_WrResp_Cb.
D/NFC-LLC (  398): Entering phLlcNfc_Interface_Read.
D/NFC-LLC (  398): Exit function
D/NFC-HCI (  398): Entering phHciNfc_Send_Complete.
D/NFC-HCI (  398): Entering phHciNfc_Receive.
D/        (  398): Entering phLlcNfc_Receive.
D/NFC-LLC (  398): Entering phLlcNfc_Interface_Read.
D/NFC-LLC (  398): Exit function
D/        (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-LLC (  398): Exit function
D/        (  398): Entering phLlcNfc_AckTimeoutCb.
D/        (  398): Entering phLlcNfc_StopTimers.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_CreateSFramePayload.
D/        (  398): Entering phLlcNfc_H_ComputeCrc.
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Exit function
D/        (  398): Exit function
D/NFC-LLC (  398): Entering phLlcNfc_Interface_Write.
D/libnfc-nxp-log(  398): NFC-Data LLC Interface writes: : 03C3B8D1
D/        (  398): Entering phLlcNfc_StopTimers.
D/        (  398): Exit function
D/NFC-LLC (  398): Exit function
D/        (  398): Exit function
D/NFC-LLC (  398): Entering phLlcNfc_RdResp_Cb.
D/libnfc-nxp-log(  398): NFC-Data Receive callback: : 0C
D/NFC-LLC (  398): Entering phLlcNfc_Interface_Read.
D/libnfc-nxp-log(  398): NFC-Data i2c_read: 98858004A55929B125809329
D/NFC-LLC (  398): Exit function
D/NFC-LLC (  398): Exit function
D/libnfc-nxp-log(  398): NFC-Data i2c_write: 03C3B8D1
D/NFC-LLC (  398): Entering phLlcNfc_WrResp_Cb.
D/NFC-LLC (  398): Exit function
D/NFC-LLC (  398): Entering phLlcNfc_RdResp_Cb.
D/libnfc-nxp-log(  398): NFC-Data Receive callback: : 98858004A55929B125809329
D/        (  398): Entering phLlcNfc_H_ChangeState.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_ComputeCrc.
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
D/        (  398): Exit function
D/        (  398): Entering phLlcNfc_H_UpdateCrc.
```

```
D/         (  398): Exit function
D/         (  398): Entering phLlcNfc_H_UpdateCrc.
D/         (  398): Exit function
D/         (  398): Entering phLlcNfc_H_UpdateCrc.
D/         (  398): Exit function
D/         (  398): Entering phLlcNfc_H_UpdateCrc.
D/         (  398): Exit function
D/         (  398): Exit function
D/         (  398): Entering phLlcNfc_H_ProRecvFrame.
D/         (  398): Exit function
D/         (  398): Entering phLlcNfc_H_ProcessIFrame.
D/         (  398): Entering phLlcNfc_H_UpdateIFrameList.
D/         (  398): Entering phLlcNfc_H_IFrameList_Peek.
D/         (  398): Exit function
D/         (  398): Entering phLlcNfc_H_DeleteIFrame.
D/         (  398): Exit function
D/         (  398): Exit function
D/         (  398): Entering phLlcNfc_StopTimers.
D/         (  398): Exit function
D/NFC-LLC (  398): Entering phLlcNfc_H_SendInfo.
D/NFC-LLC (  398): Entering phLlcNfc_Interface_Read.
D/NFC-LLC (  398): Exit function
D/NFC-HCI (  398): Entering phHciNfc_Receive_Complete.
D/NFC-HCI (  398): Entering phHciNfc_Process_HCP.
D/NFC-HCI (  398): Entering phHciNfc_Receive_HCP.
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Entering phHciNfc_Process_Response.
D/NFC-HCI (  398): Entering phHciNfc_Recv_ReaderA_Response.
D/NFC-HCI (  398): Entering phHciNfc_ReaderA_InfoUpdate.
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Entering phHciNfc_Reset_Pipe_MsgInfo.
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Entering phHciNfc_Resume_Sequence.
D/NFC-HCI (  398): Entering phHciNfc_Info_Sequence.
D/NFC-HCI (  398): Entering phHciNfc_ReaderMgmt_Info_Sequence.
D/NFC-HCI (  398): Entering phHciNfc_ReaderA_Info_Sequence.
D/NFC-HCI (  398): Entering phHciNfc_Send_Generic_Cmd.
D/NFC-HCI (  398): Entering phHciNfc_Build_HCPFrame.
D/NFC-HCI (  398): Entering phHciNfc_Build_HCPHeader.
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Entering phHciNfc_Build_HCPMessage.
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Entering phHciNfc_Send_HCP.
D/NFC-HCI (  398): Entering phHciNfc_Send.
D/         (  398): Entering phLlcNfc_Send.
D/         (  398): Entering phLlcNfc_H_CreateIFramePayload.
D/         (  398): Entering phLlcNfc_H_ComputeCrc.
D/         (  398): Entering phLlcNfc_H_UpdateCrc.
D/         (  398): Exit function
D/         (  398): Entering phLlcNfc_H_UpdateCrc.
D/         (  398): Exit function
D/         (  398): Entering phLlcNfc_H_UpdateCrc.
D/         (  398): Exit function
D/         (  398): Entering phLlcNfc_H_UpdateCrc.
D/         (  398): Exit function
D/         (  398): Entering phLlcNfc_H_UpdateCrc.
D/         (  398): Exit function
D/         (  398): Exit function
D/         (  398): Exit function
```

```
D/        (  398): Entering phLlcNfc_H_StoreIFrame.
D/        (  398): Exit function
D/NFC-LLC (  398): Entering phLlcNfc_Interface_Write.
D/libnfc-nxp-log(  398): NFC-Data LLC Interface writes: : 0684850203179F
D/        (  398): Entering phLlcNfc_StopTimers.
D/        (  398): Exit function
D/NFC-LLC (  398): Exit function
D/        (  398): Entering phLlcNfc_StartTimers.
D/        (  398): Exit function
D/        (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-LLC (  398): Exit function
D/NFC-LLC (  398): Entering phLlcNfc_Interface_Read.
D/NFC-LLC (  398): Exit function
D/        (  398): Entering phLlcNfc_StartTimers.
D/        (  398): Exit function
D/        (  398): Exit function
D/        (  398): Exit function
D/NFC-LLC (  398): Exit function
D/libnfc-nxp-log(  398): NFC-Data i2c_write: 0684850203179F
D/NFC-LLC (  398): Entering phLlcNfc_WrResp_Cb.
D/NFC-LLC (  398): Entering phLlcNfc_Interface_Read.
D/NFC-LLC (  398): Exit function
D/NFC-HCI (  398): Entering phHciNfc_Send_Complete.
D/NFC-HCI (  398): Entering phHciNfc_Receive.
D/        (  398): Entering phLlcNfc_Receive.
D/NFC-LLC (  398): Entering phLlcNfc_Interface_Read.
D/NFC-LLC (  398): Exit function
D/        (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-HCI (  398): Exit function
D/NFC-LLC (  398): Exit function
```