OTP circumventing in MIFARE ULTRALIGHT: Who says free rides?

bughardy
bughardy@cryptolab.net

Eagle1753
eagle1753@onenetbeyond.org

Introduction

RFID technologies are becoming more and more prevalent in our lives. This motivated us to study them, and in particular to study the MIFARE ULTRALIGHT chips, which are widely used in public/mass transport systems. We focused on multiple-ride tickets, and were surprised that MIFARE ULTRALIGHT chips do not seem to store any type of encrypted data.

NFC chips should help transport company to prevent fraud and fake tickets but the scenario we are going to present shows a very different situation.

In 2012 a security team from U.S.[1] found a bad implementation of transport ticket checking system. Each time a traveler made use of one of his ride, each ultralight card can store several rides, the system write the rides left on the DATA bytes of the ticket. It was enough to erase everything in data area to get a brand new ticket. After that several transport company started to use a security features of Mifare Ultralight ticket, OTP (One-Time-Pad) area. This area is composed o 32 bits. At the beginning each bit is set to zero, after each ride one bit is set to 1. In this area you can't turn bit back to zero.

In this paper, and in the accompanying talk, we will discuss how bypass OTP features to gain again free rides using Mifare Ultralight transport tickets. And we will also show an easy script that displays ticket

informations and help us to get free rides.
[1]Corey Benninger and Max Sobell, security researchers at the Intrepidus Group

MIFARE ULTRALIGHT Sectors

Let's give a look to how a Mifare Ultralight chip is made.
We have 5 sectors:

Sector	Memory
UID – Ticket's serial number	9 bytes
Internal byte, written by manufacturer	1 byte
Lock bytes	2 bytes
OTP bytes	4 bytes
Data bytes	48 bytes

In OTP sector all bits are set to 0 after production; if you want to write some bits in this area, they will be ORed with the existing, thus making impossible to turn a “1” bit into “0”. This security features is used to prevent abuse of NFC chips in transport systems, so that it would be useless to erase the data area in order to get a brand new ticket.

How the system works

As we have already seen, each ride implies that a bit is set to 1 in OTP sector. But there are also others relevant data written on the ticket.
For example transport type (metro, bus, etc) or the time of last stamp. All this data are written on DATA sector.
As soon as you your ticket, you have 90 minutes to use your ride. Then you have to stamp again.
The ticket is valid in any transport system, such as subway, bus or whatever and it doesn't matter where you stamp it: you can do it on a bus and then take subway if you are within 90 minutes; however, you can't get subway more than once for each ride. This means that ticket must store also data to identify metro station.

Vulnerabilities

After much thinking we found two possible vulnerability.

Decoding time data
The first vulnerability we discover involve decoding the time written on the ticket. After acknowledging how it is encoded and written in data sector anyone can write it to the ticket avoiding to write OTP data, so you won't remove rides from your ticket but it looks like a validated one. Unfortunately we are still not able to decode data, we will provide some dumps for anyone who would like help us.

There is also a second way to use this vulnerability: a MiTM attack. Using a NFC emulator, android phone or proxymark, an attacker could sniff the communication between the stamping machine and the chip and write only DATA bytes on the ticket. In that way you can validate the ticket for 90 minutes without knowing how time data is encoded.
We didn't test it cause we don't have a proxymark.

(ab)Using lock bytes
The most interesting vulnerability requires another sector: lock bytes. This sector is made of two bytes, each bit is used to freeze status of others bytes:

First byte

bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7
Lock status of first 4 bytes in DATA sector	Lock status of second 4 bytes in DATA sector	…	…	Lock status of OTP sector	Freeze status of lock bits for second half of data sector	Freeze status of lock bits for first half of data sector	Freeze status of lock bit for OTP sector

Second byte locks only bytes in data sector, so at the moment is useless. Using bit 4 of first byte of lock sector we can lock status of the OTP sector, making impossible to machine turn it's bits to 1. This create an unlimited-

rides ticket.
Our test shows the following lock bytes structure for our ticket:

HEX	F2 03
BIN	11110010 00000011

Analyzing it we saw that there are several bytes in DATA sector which are locked and them can't be unlocked anymore due the bit 6 of first byte. So, by turning bit 4 of first byte to 1, we locked the OTP sector, getting our free ticket for life.

HEX	FA 03
BIN	11111010 00000011

This makes the ticket endless

Tool

We're developing a tool to check ticket status and make it endless. It is really at alpha stage and we wish to rewrite it in C/C++.
It's written in python and requires nfc-tool installed in the computer.
Its functions are:

- Check dumps state
- Edit rides left and endless functionality
- Write dumps on HDD
- Write dumps on ticket

It is not really optimized but it works and it is okay as a first release.

```
bughardy@cryptolab:~$ python nfc.py dump.mfd
Looking for dump...
Printing ticket infos...


SERIAL NUMBER: 04d3e1be32782680ec
INTERNAL BIT: 48
LOCK BYTES: fa03     FREE RIDES ENABLED!
OTP BYTES: 3ffffffe  RIDES LEFT: 1
DATA BYTES:
010400000201 02be4011960000ae10a061040af32cbb244e43671f000
4f8000043671f00000d0004f8ae107a2f12e5f
```

The main function is the dumpParser function which extracts infos from ticket dump:

```
def dumpParser(content):
    SERIAL_NUMBER = content[0:18]
    INTERNAL_BIT = content[18:20]
    LOCK_BYTES = content[20:24]
    OTP_BYTES = content[24:32]
    DATA_BYTES = content[32:-1]
    INFOS_LIST = [SERIAL_NUMBER, INTERNAL_BIT,
LOCK_BYTES, OTP_BYTES, DATA_BYTES]

    return INFOS_LIST
```

The dumpEdit function is used to change number of rides left on the ticket:

```
def dumpEdit(rides,path,content):
    otpData = content[24:32]
    ticketPayLoad1, ticketPayLoad2 = 0,0
    rides = 5 - int(rides)
    while rides != 0:
        ticketPayLoad1 += 10**(2+rides)
        ticketPayLoad2 += 10**(1+rides)
        rides -= 1
```

In ticketPayLoad1 stores values of first 2 bytes of OTP data and ticketPayLoad2 the second half. The function of course can only decrease the rides number.

As you can see code is very simple and we didn't set up everything yet. It is a sort of proof-of-concept.

Fix

Fix the OTP vulnerability should be quite easy; it would be enough to release a firmware update in order to check if lock sector is locking OTP area or not, and if so, refuse to validate the ticket.
The MiTM attack and the decoding attack would be more difficult to be fixed, indeed, as regards our opinion, there is not fix to that. The problem could be fixed by using NFC chips with encrypted communication. But the encryption should change the time data for each ticket, for example using UID as IV, otherwise a man with one ticket could validate several other tickets just copying his time data to their.

bughardy aka Matteo Beccaro
Eagle1753 aka Matteo Collura