OPPOSING FORCE

(Ab)using Smart Cities

The dark age of modern mobility

Authors:

Matteo Beccaro, Founder & CTO at Opposing Force
matteo.beccaro@opposingforce.it

Matteo Collura, Researcher at Politecnico di Torino
eagle1753@onenetbeyond.org

August 20, 2016

Contents

1 Introduction

Since these last few years our world has been getting smarter and smarter. We may ask ourselves: what does *smart* mean? It is the possibility of building systems which are nodes of a more complex network, digitally connected to the internet and to the final users. Our cities are becoming one of those networks and over time more and more elements are getting connected to such network: from traffic lights to information signs, from traffic and surveillance cameras to transport systems. This last element, also called as *Smart Mobility* is the subject of our analysis, divided in three sub-element, each one describing a different method of transport in our city:

- Private transport: for this method we analyze the smart alternatives aimed to make **parking** activity easy, hassle free and more convenient
- Shared transport: we focus our attention on those systems which are sharing transport vehicles. In particular we deal with **bike** sharing which seems to be the most wide spread system in European cities
- Public transport: object of our analysis for this section is the **bus**, **metro** and **tram** network

The aim of our analysis is understanding the ecosystem which each element belongs to and performing a security evaluation of such system. In this way the most plausible attack and fraud scenarios are pointed out and the presence of proper security measures is checked. All the details discussed here are collected from a sample city, but the same methodology and concept can be applied to most of the *smart cities* in the world.

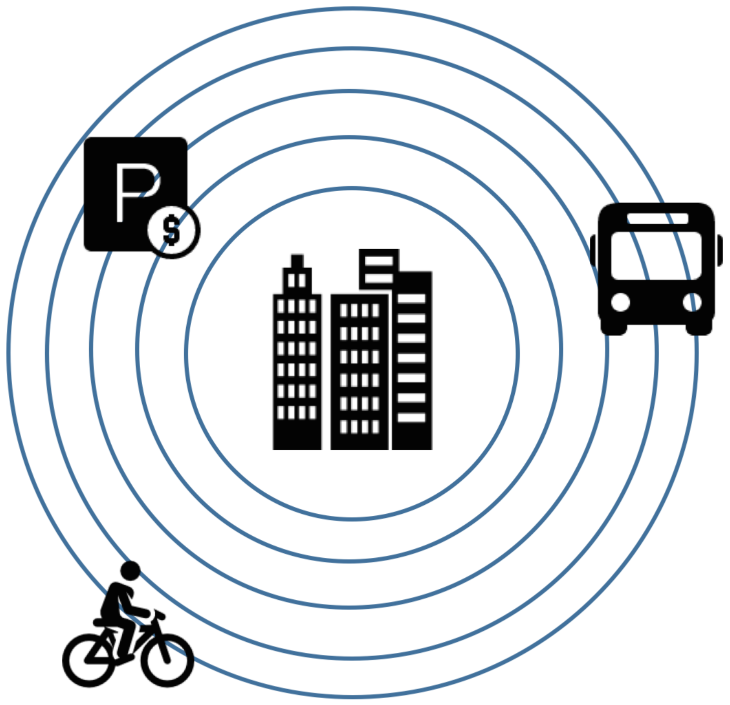


Figure 1: The smart parking meter ecosystem

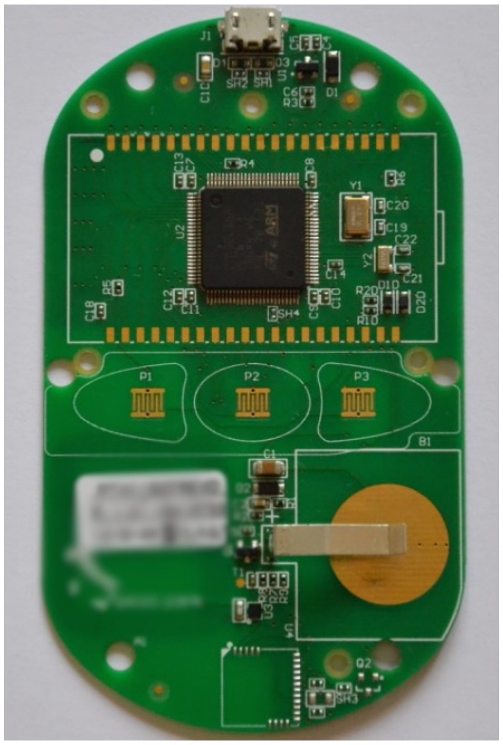
2 Private Transport: Smart Parking

This section is concerning one of the most annoying activities in which we are involved when moving from one place to another by means of our car: finding a park. If it is not a parking lot but just on the border of the street, usually we need to stamp a ticket from the parking meter and leave it under the windshield. This procedure however belongs to the old *analog* world, how to turn it smart? The answer is: **Smart Parking Meter**.

2.1 Token

This device is able to replace all the parking meters in at least 50 big cities of our country. Whenever it is turned on, the user is asked to select the city and the parking zone in which the car will be left. The whole procedure is made by pressing three physical buttons (Enter/Power, Right, Left). Then it is sufficient to place the device under the windshield in such a way it is easily noticeable by an inspector.
It is easy to spot the interfaces with the external world just by giving a quick look at the Smart Parking Meter. There is *NFC* tag, then a micro *USB*. Moreover, when opening the case some well labeled lines are showing up, revealing a *JTAG* and a *SWD* interface.

(a) The smart park meter dressed...



(b) ... and naked

When activated, the display will show the city and zone set in the previous passage, then the date and time and finally the price for the time unit, e.g. 1.50 € per hour. The selected fee will be automatically applied for each second of activity: in this way the user is paying the effective parking time (when using the traditional park meter instead the user is supposed to make a forecast). This device can be ordered online or bought at some specific resellers, who are allowed to recharge money; however, this procedure can also be performed by the user, at home, through a PC, paying by credit card. In order to make the computer properly recognize the device, the user is asked to install a small Java utility.

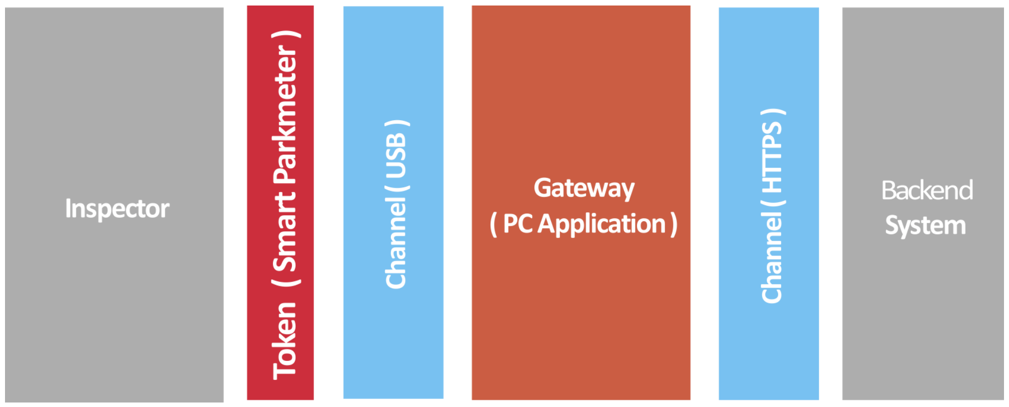


Figure 2: The smart parking meter ecosystem

Summing up, the ecosystem we are analyzing is composed by:

- **Token**: the Smart Parking Meter
- **Gateway**: a PC application connecting the device to the Internet
- **Backend**: web service used to perform several actions on the device during the update process
- **Controller/Inspector**: it is not an element we can actually control, so we are not able to analyze its attack surface and the role he can play during a fraud scenario

2.1.1 Token's Attack Surface: NFC interface

The first step involves the identification of the *NFC* tag implemented in this device. It is sufficient a mobile phone to identify that the chip on board is a *NT3H1101* made by *NXP Semiconductors*. One of the main features of this tag is the possibility to interact both with a I^2C interface according to a master-slave relationship and also with a *SRAM* memory. The user memory is composed by 888 bytes and arranged in 222 pages.
After the examination of the datasheet[1], we decided to inspect these areas:

- **Capability Container** is configured as the *One Time Programmable* sector in MIFARE Ultralight[2]. In this case it doesn't seem to be showing any relevant information, moreover it is not changing while the device is working or afterwards
- **Lock Bytes** are configured according to default values. Their role is again the same as in MIFARE Ultralight[3]; all the pages are by default readable and writable
- **Configuration/Session Registers** are configured according to default values. These registers, two pages each, are used to set a relationship between *NFC*, I^2C and *SRAM* memory. The most useful operations concern the enabling of *SRAM* Memory mirroring on the user memory, and the possibility to write directly on it from *NFC* side. However, the *Session Registers* can be edited only through the I^2C interface. As regards *Configuration Registers*, they depend on the *Session* ones, so, in conclusion, we decided to solve this challenge in our future works

[1] http://www.nxp.com/documents/data_sheet/NT3H1101_1201.pdf
[2] http://www.nxp.com/documents/data_sheet/MF0ICU1.pdf#page=12
[3] http://www.nxp.com/documents/data_sheet/MF0ICU1.pdf#page=11

- **User Memory** is readable and writable by anyone, no encryption is on. It is surprisingly empty
- ***SRAM* memory** unfortunately is still a mystery to be solved. The only possibility to gather data from here is to access the I^2C and change the proper bits in the *Session Registers*, otherwise it wouldn't be possible to enable the communication channel

2.1.2 Token's Attack Surface: I^2C interface

As stated in the previous subsection, no tests have been performed on this interface by now.

2.1.3 Token's Attack Surface: *USB* interface

The update and recharge processes are performed through this interface so that the device is able to reach the web server and update its configuration files.
As a first analysis, it makes sense trying to eavesdrop the communication between the Smart Parking Meter and the PC. This will possibly reveal the authentication procedure and data flow.

2.1.4 Token's Vulnerabilities: *USB* interface

A malicious user could try to edit the configuration files requested by the device. The operation is possible since there is no signature/key to validate the data: in case there are modifications the Smart Parking Meter is not rejecting them. This opens a window on different scenarios:

- Modification of sensitive data (i.e. city names, prices per hour, working time etc.)
- Inspection of authentication methods towards the web server
- Creation and injection of a custom firmware

Moreover as soon as the device is connected to the PC, it will send out all the parking transactions and funds updates. This will likely update the history of our Smart Parking Meter on the online database.

2.1.5 Attack Scenario: Configuration Files

When the device is working as it is supposed to do, the following information are shown on the display:

- City Name
- City Fare Zone
- Price per time unit
- Date and Time

However, a lot of data is exchanged during the update process of the Smart Parking Meter, like the time window in which it should work (during day, not during night), the faring frequency (per hour, per quarter, etc.), the minimum applicable fee and so on. When the device is stopped, it will show the amount of money it has been charged and the remaining credit. We tried to guess the formula the device applies to charge the right fee:

$$Fee = \frac{(price\ per\ time\ unit) * (faring\ frequency) * (seconds\ elapsed)}{3600\ seconds} + (minimum\ applicable\ fee)$$

As deducible, it would be sufficient to force the first or the second factor at the numerator to be zero together with the minimum applicable fee in order to get a final result of 0 € to be charged. Since however the price per time unit is showed on the display during device activity, it is better to set the faring frequency equal to zero. Needless to say, we tried and the final fare became always zero, no matter how many elapsed seconds!

2.1.6 Attack Scenario: Firmware Upgrade

Another attack scenario involves the firmware upgrade mechanism, which is triggerable from the software installed on the PC (Gateway). Once the process is started, the software will download a *DFU* file, put the device in *DFU* mode and upload it on the device for upgrading. We discovered that no integrity checks are performed on the firmware, therefore, it is possible to install a custom version on the device. An Inspector can verify the correct device activity only by reading the display, which shows all the information regarding the parking payment status. This, combined with the firmware upgrade issue, allows an attacker to load on the device a custom software, able to emulate the data displayed by the original one without taking into account the money on the device.

2.2 Gateway

As stated before, the connection between the device and the Internet is mediated by the Gateway. The user is asked to install a simple and light Java application that will enable a channel of communication with the token. The allowed operations on the device are:

- Update
- Money recharge (provided a user is created on the backend and linked to the device's serial number)

2.2.1 Gateway's Attack Surface

When eavesdropping the incoming and outgoing data packets, it is possible to read clearly each action, from the logs exchange to the update process. We manage to intercept even a firmware upgrade request.

2.2.2 Gateway's Vulnerabilities

As deducible from the above paragraph, the data are not obfuscated, leaving the possibility to an external user to understand how the update process is made. Moreover there is a lack of certificate validation! This means that a modification of the data sent/received will be accepted and will generate possibly new requests. For example it was possible to pretend the device's firmware not to be up to date. As a consequence the server produced a response containing an hardcoded *URL*, which was pointing to a *.dfu* packet. This example shows also how to dump the firmware without even trying to extract it from the device.

3 Shared Transport: Bike Sharing

We studied also a new transport system: the shared transport, in particular we focused our attention on the bike sharing service. It is very convenient because there are more than a hundred stations spread all over the city of at least ten bicycles each. It is a low cost option used everyday. The user needs a token device to take a bike: this can be a NFC card or his mobile phone, but, as regards this last case only, it is also required to register an online account.
The token is used to identify the user to the system and then, if a valid subscription is verified, the bicycle is unlocked and ready to be ridden. Once the bicycle is returned to another station, the system evaluates the time elapsed since the unlocking and charges the user of the correct amount of money (Notice that for the first thirty minutes it is free of charge). We mapped the ecosystem of this network in the following elements:

- Token
- Reader/Controller
- Backend
- Web application

We tried to perform an assessment on the complete ecosystem without risking to cause any denial of service, therefore we didn't analyze the last two elements: Backend and Web Application.

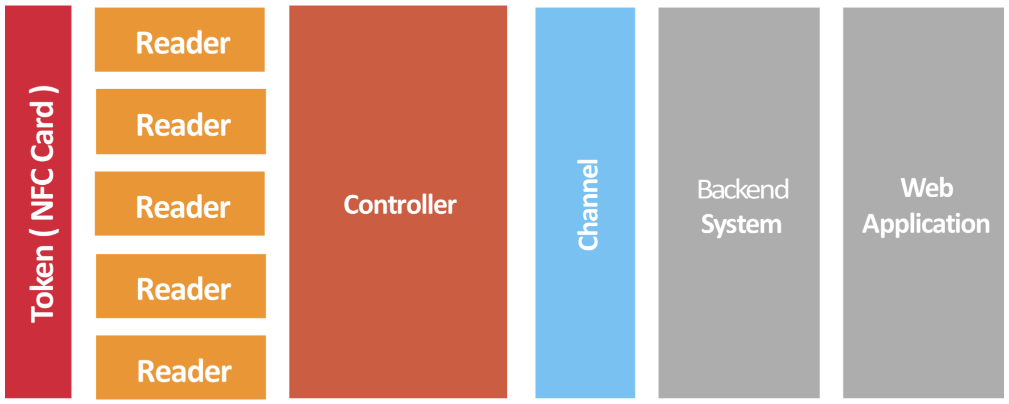


Figure 3: The bikesharing ecosystem

3.1 Token

As seen before the Token device can be either a *NFC* cardor the user's mobile phone with the bike sharing application installed. The *NFC* card is a *MIFARE ULTRALIGHT* tag with 512 bit of *EEPROM* memory, 7 bytes *UID* and very basic security features[4]. The card's UID is used by the controller to identify the user, all the remaining *Data* bytes are not used and all set to 0.

The mobile application instead makes use of exposed *APIs* to speak with the system and send commands, for example when unlocking a bicycle. It also uses *GPS* to localize the user and provide directions to the nearest bike station. Moreover, it allows the releasing of a bike only if the user is close enough to the station.

[4]http://www.nxp.com/documents/data_sheet/MF0ICU1.pdf

```
——————————————— NFC Tag Content ———————————————
[PAGE 0x01] 04:CA:63:25 (UID0–UID2, BCC0)
[PAGE 0x02] 1A:53:36:80 (UID3–UID6)
[PAGE 0x03] FF:48:00:00 (BCC1, INT, LOCK0, LOCK1)
[PAGE 0x04] 00:00:00:00 (OTP0–OTP3)
[PAGE 0x05] FF:FF:FF:FF (DATA)
[PAGE 0x06] 00:00:00:00 (DATA)
[PAGE 0x07] 00:00:00:00 (DATA)
[PAGE 0x08] 00:00:00:00 (DATA)
[PAGE 0x09] 00:00:00:00 (DATA)
[PAGE 0x0A] 00:00:00:00 (DATA)
[PAGE 0x0B] 00:00:00:00 (DATA)
[PAGE 0x0C] 00:00:00:00 (DATA)
[PAGE 0x0D] 00:00:00:00 (DATA)
[PAGE 0x0E] 00:00:00:00 (DATA)
[PAGE 0x0F] 00:00:00:00 (DATA)
————————————————————————————————————————————————
```

3.1.1 Token's Attack Surface

We started analyzing the Token's attack surface starting from the NFC card.
Since it is a *MIFARE ULTRALIGHT* card, the following interesting features are worth to be explored:

- One Time Programmable (OTP) sector
- Lock bytes sector
- UID
- No encryption nor authentication for reading and writing data

It was trivial to see that both *OTP* and *Lock bytes* sectors are not used in this implementation because all *Data* sectors are set to 0 both before and after using the system. Therefore the only sector in use is the *UID*, which is used to identify the user. Figure 4 is showing in a schematic way the authentication process. For what concerns the mobile app, we downloaded and decompiled the *Android APK*, doing a quick static code review. Then the communication between the application and the backend during a bike unlock is inspected.

3.1.2 Token's Vulnerabilities

During our analysis we find several issues regarding the token, both in the *NFC* card and the mobile application: those would allow a malicious user to bypass completely the security of the system and get free bikes.

1. *NFC* Card
   - It is possible to read other people's cards without authentication
   - The *UID* is stored and transmitted in clear text, therefore its usage as *ID* for the authentication is not a wise thing.
   - It is possible to exploit the anti-collision protocol to read the *UID* during a legit transaction from an longer distance

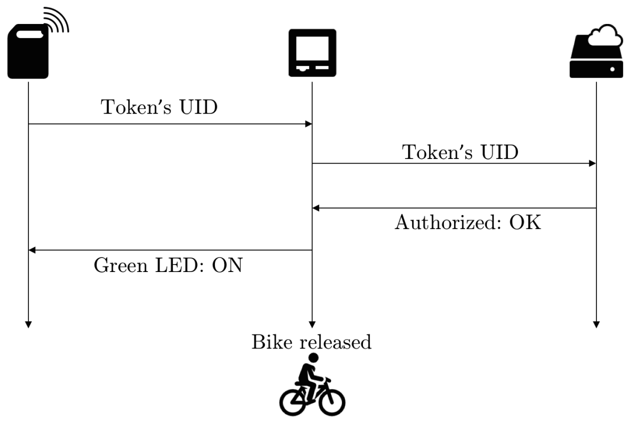
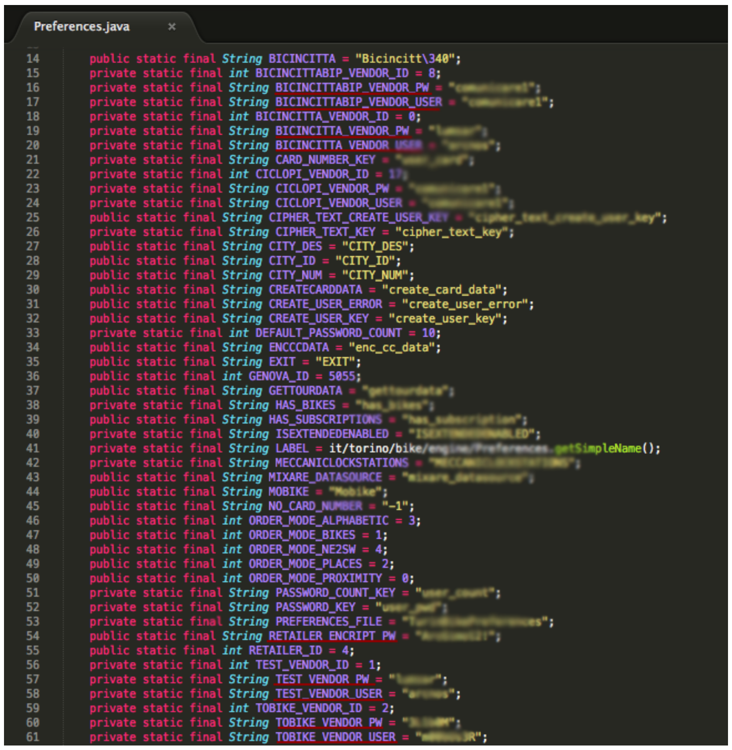


Figure 4: Authentication scheme

2. Mobile application

   - During the static code analyze several hardcoded credentials were found: they are used to administrate the backend Webservice. We suppose it would be possible to use those credentials to take complete control of the backend.
   - Other vulnerabilities were found during the user registration process: they would allow an attacker to forge valid accounts with even positive money balance



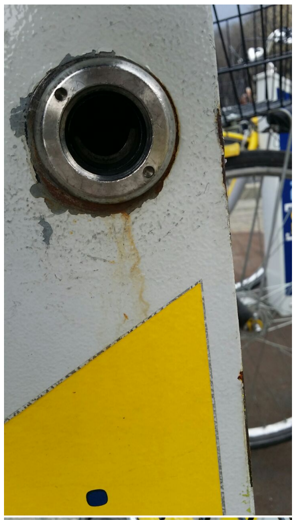
Preferences.java

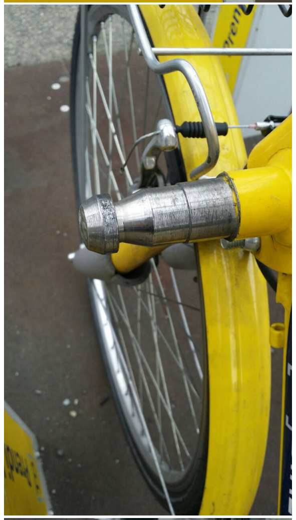
```
public static final String BICINCITTA = "Bicincitt\340";
private static final int BICINCITTABIP_VENDOR_ID = 8;
private static final String BICINCITTABIP_VENDOR_PW = [redacted];
private static final String BICINCITTABIP_VENDOR_USER = [redacted];
private static final int BICINCITTA_VENDOR_ID = 0;
private static final String BICINCITTA_VENDOR_PW = [redacted];
private static final String BICINCITTA_VENDOR_[redacted];
private static final String CARD_NUMBER_KEY = [redacted];
private static final int CICLOPI_VENDOR_ID = [redacted];
private static final String CICLOPI_VENDOR_PW = [redacted];
private static final String CICLOPI_VENDOR_USER = [redacted];
public static final String CIPHER_TEXT_CREATE_USER_[redacted]";
private static final String CIPHER_TEXT_KEY = "cipher_text_key";
public static final String CITY_DES = "CITY_DES";
public static final String CITY_ID = "CITY_ID";
public static final String CITY_NUM = "CITY_NUM";
public static final String CREATECARDDATA = "create_card_data";
public static final String CREATE_USER_ERROR = "create_user_error";
public static final String CREATE_USER_KEY = "create_user_key";
private static final int DEFAULT_PASSWORD_COUNT = 10;
public static final String ENCCCDATA = "enc_cc_data";
public static final String EXIT = "EXIT";
public static final int GENOVA_ID = 5055;
public static final String GETTOURDATA = [redacted];
private static final String HAS_BIKES = [redacted];
public static final String HAS_SUBSCRIPTIONS = [redacted];
private static final String ISEXTENDEDENABLED = [redacted]";
private static final String LABEL = it/torino/bike[redacted]();
private static final String MECCANICLOCKSTATIONS = [redacted];
public static final String MIXARE_DATASOURCE = [redacted];
public static final String MOBIKE = [redacted];
public static final String NO_CARD_[redacted] = "-1";
public static final int ORDER_MODE_ALPHABETIC = 3;
public static final int ORDER_MODE_BIKES = 1;
public static final int ORDER_MODE_NE2SW = 4;
public static final int ORDER_MODE_PLACES = 2;
public static final int ORDER_MODE_PROXIMITY = 0;
private static final String PASSWORD_COUNT_KEY = "[redacted]";
private static final String PASSWORD_KEY = "[redacted];
private static final String PREFERENCES_FILE = [redacted]";
public static final String RETAILER_ENCRIPT_PW = [redacted];
public static final int RETAILER_ID = 4;
private static final int TEST_VENDOR_ID = 1;
private static final String TEST_VENDOR_PW = "[redacted]";
private static final String TEST_VENDOR_USER = [redacted]";
private static final int TOBIKE_VENDOR_ID = 2;
private static final String TOBIKE_VENDOR_PW = [redacted];
private static final String TOBIKE_VENDOR_USER = [redacted]";
```

Figure 5: Hardcoded login credentials

3.2 Reader/Controller

In the system we have analyzed the reader and the controller are in the same physical structure. In order to avoid any service disruption we mapped the attack surface but never performed any case scenario which could cause a denial of service.





3.2.1 Reader/Controller's Attack Surface

We pointed out several entry points a malicious user can exploit to attack the Reader/Controller element:

- NFC Interface
- Reader/Controller to Backend communication
- Physical Hook

We tested only the station hook to find out how it is realizing whether a bike has been correctly locked or not.

3.2.2 Reader/Controller's Vulnerabilities

It is possible to trick the bike station pretending that a bike has not been unlocked. It is sufficient to pass the *NFC* card on the reader and wait until the LED starts blinking. Then, we need to extract the bike slowly for just a centimeter. After a short amount of time the process goes in timeout and the lock mechanism is activated again, even if the bike has been released. This way an attacker can unlock an unlimited number of bikes. Unfortunately, if the system is tricked in such way but, after some time, the bike is properly locked, the backend system recognizes that the same bike has been locked twice. This will produce a ban on the used *NFC* card and therefore the associated user after a week or so.

4 Public Transport: Bus, Tram, Metro

Public transport has been renewed in its architecture, in order to look smarter. That's why in our sample city the old paper tickets have been replaced with *NFC* tags. A user can buy a five-ride or fifteen-ride ticket. Each ride is valid 90 minutes from the stamping time, except for the metro, which takes out one ride each time we use it. As shown in Figure 5 the ecosystem we are analyzing is now composed of:

- Token (*NFC* tickets)
- Validating Machine
- Backend

We tried to understand more about the backend but it would have required invasive techniques. As a consequence it was just checked the presence of any fraud detection mechanisms and, if so, how they work.

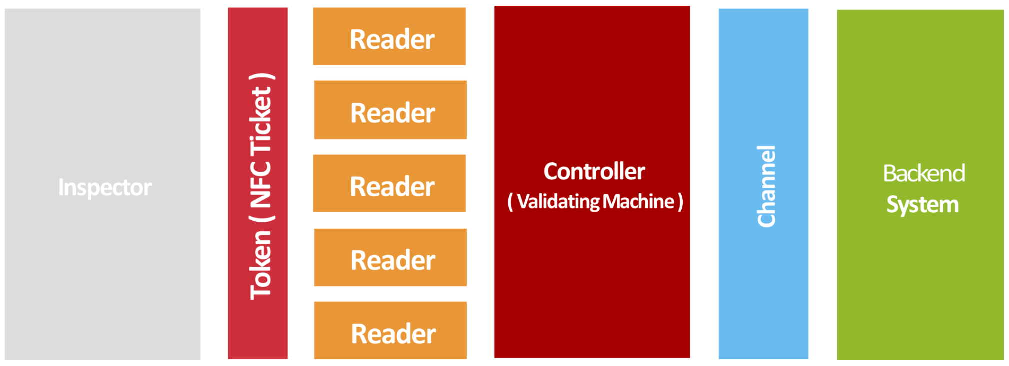


Figure 6: Public transport system

4.1 Token

The ticket is a well known *MIFARE ULTRALIGHT* tag, with 512 bit of *EEPROM* memory, 7 bytes *UID* and very basic security features[5].

4.1.1 Token's Attack Surface

The following interesting features are worth to be explored:

- One Time Programmable (*OTP*) sector
- Lock bytes sector
- UID
- No encryption nor authentication for reading and writing data

After a quick analysis we figured out that the ticket rides are stored in the *OTP* sector, as suggested by reference design. Other relevant data are stored in the *Data* sector, such as date/time of last stamp, type of transportation vehicle and its ID number, an integrity signature, and some other data we have not decoded yet. The UID is used to calculate the signature stored in the data sector; unfortunately it will be saved in the backend, as we will see in the following paragraphs.

[5]http://www.nxp.com/documents/data_sheet/MF0ICU1.pdf

4.1.2 Token's Vulnerabilities

On the token's side we find that the lock bytes are not all secured: the most of the block locking bits are in facts set to 0, including the *OTP* one. As a consequence, at least the ticket is vulnerable to the *Lock Attack*.

4.2 Validating Machine

We don't know very much of the physical specifications of the Validating Machine, however, during the working period they are working offline. We supposed that when the vehicles are sent to deposit, about once per week, they are connected to a backend service, to upload their logs and update the blacklist. We will speak about that later.

4.2.1 Validating Machine's Attack Surface

Whenever a ticket is supposed to be stamped, the validating machine is checking first:

1. Last stamp date/time greater than 90 minutes
2. *OTP* sector writable
3. Signature integrity

As regards point #1 it is a nice try to exploit the *Time Attack*, by editing the timestamp however we like. Unfortunately, the presence of a Signature (#3) is making this impossible (as long as that string of bytes is not deciphered). For what concerns point #2 there is a check on the *OTP* sector lock bit. This is avoiding the usage of *Lock Attack*. The last possibility to check is the *Replay Attack*. Finally, since the *NFC* interface is linked to an embedded system, it would be interesting to check how it will respond to a fuzzing test. We didn't try any similar attack in order not to cause any denial of service.

4.2.2 Validating Machine's Vulnerabilities

Actually there is just one vulnerability we are able to exploit which is the aforementioned *Replay Attack*: the Validating Machine is not able to retrieve on the fly the history of the ticket to be stamped. For that reason, a clone *MIFARE ULTRALIGHT* tag can be used to make the stamping machine believe that there is one ride more than expected. This will produce a valid ticket, with the same number of rides of the real one but with a verified data sector.

4.3 Backend

As stated before, there is a lack of knowledge about the backend since we skipped intrusive tests in order to avoid denials of service. However, its existence is confirmed since we experienced during our tests that if we try to validate a five-ride ticket more than five times, one week after the 5th stamp, the stamping machine refuses the ticket. This is a clear evidence of a blacklist check.
The Validating Machine contains a ticket blacklist, updated through the backend approximately once every week. Here each record is possibly including the ticket's *UID* and rides left. If the ticket to be stamped is blacklisted, the Stamping Machine will write *0xFF* all over the ticket, making it forever useless.

4.3.1 Backend's Attack Surface

The communication between the Validating Machine and the Backend is occasional, so it is hard even to define the channel. For sure the exchanged data are stored in a database, indexed by the ticket *UIDs*. In order to avoid any denial of service or being too intrusive, we refrained from performing any test.

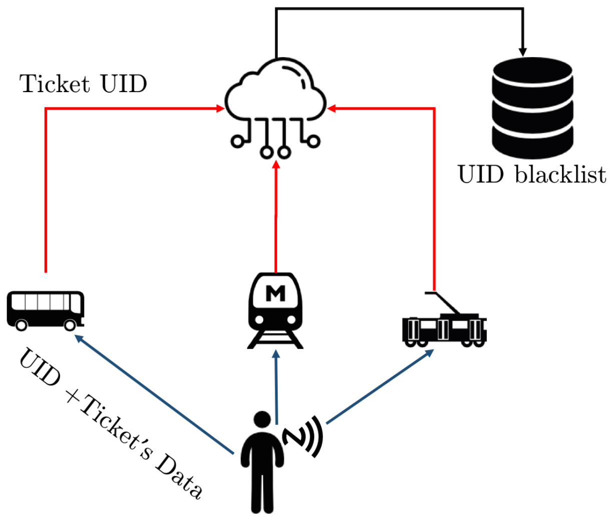


Figure 7: Public transport system

4.3.2 Backend's Vulnerabilities

This system is vulnerable to the injection of virgin multiple-ride tickets. A whitelist cannot exist as long as the Validating Machine and the ticket-vending machines are not always connected to the Internet nor the backend.

5 Impact

We would like to point out what could be the impact of a large scale abuse of the above issues. Clearly the aim of this research is not try to understand what a state-wide attack could do on the transportation system but instead what are the possible financial losses derived from the exploitation of the above vulnerabilities from, for example, a criminal organization. That organization could sell ticket which could be valid for life for a fraction of the price, sell modified parking device or discounted recharge for bike sharing account and this, for our example city, would be a multi hundred milion euros loss each year.
We are also interested in continuing our research to understand what a more advanced criminal operation could do to create a disruption of the transportation services.

6 Future works

In the future we plan to extend our area of research to other elements of smart cities[6], such as:

1. Power Management
2. Water Management
3. Surveillance

We will also try to extend our methodology to cover those additional elements.

[6]http://securingsmartcities.org/wp-content/uploads/2015/05/CitiesWideOpenToCyberAttacks.pdf

7 Conclusions

By means of this paper we hope the readers will be aware of the possible risks concerning all the applications that are labeled as “smart”. This document’s aim is to give an overview about how our cities are getting smarter and more connected, how such ecosystems are composed and how to perform a security assessment on them. Moreover we hope the reader will be curious and willing to look into other environments, now that he has a deeper knowledge of the inner working of such systems.

8 Links

Opposing Force - www.opposingforce.it - @_opposingforce
SecuringSmartCities - www.securingsmartcities.org
Matteo Beccaro - @_bughardy_
Matteo Collura - @eagle1753