UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

epl

A detection system for the sources of information leaks on networked smart devices

Dissertation presented by

Mattieu DETAILLE, Mohammad SYED

to obtain the Master's degree in

Computer Science and Engineering

Supervisor(s)

Dr Ramin SADRE

Reader(s)

Dr Marco CANINI, Quentin DE CONINCK

Academic year 2015-2016

Acknowledgements

First, we would like to express our deepest gratitude to Professor Ramin Sadre for his full support, expert guidance, understanding and encouragement throughout the development of our dissertation.

We would like to thank Dr Olivier Pereira and Dr Naim Qachri for advising us in the development part of our system. Their advice was very helpful and allowed us to attain our implementation goal.

Last but not least, we would like to thank the Catholic University of Louvain-la-Neuve for providing us with the materials we needed to pursue our goal throughout this dissertation.

Abstract

An amazing amount of new technology has been introduced into our homes in the 21^{st} century. Included in this technology, are widely-used smart devices such as smart phones, smart TVs, etc. These smart devices are known for gathering data, which raises concerns about privacy and the risk of information leakage. The purpose of this paper is to bring to light the dangerous practices used by software in these devices. These practices can cause leakage of sensitive data.

We decided to deviate from the approach used by anti-viruses. As connectivity is the main feature of smart devices, we decided to design a detection system that solely investigates their network traffic, in search of vulnerabilities that could indicate the presence of leaks. More than just passive search, the system also performs, when possible, a *Man in the Middle attack* in order to decrypt and analyze the information transiting through the encrypted traffic.

We concentrated our experiments on smart phones and smart TVs because these devices are the most commonly used nowadays. Our experimental results demonstrate that many widely-used applications designed for these smart devices present several sources of leakage. We were also able to isolate some actual information leaks. The system described in this paper can be used by users to assess the risks of using a smart device or any of its applications. It can also be used by companies that buy or create new devices or software for smart devices, as a means of auditing the network traffic of their product in order to verify its quality in term of privacy and risk of leakage.

Contents

Chapter 1

Introduction

Today, according to a Norton study ([1]), one in three, or approximately two million Google Android mobile applications, leak users' personal information, including phone numbers, call histories, phone contacts, PIN numbers, etc. through the network. For example, not long ago, the well-known mobile application *Foursquare*, which indicates good places to go and services such as restaurants located in nearby areas, was discovered to have sent the personal contacts list from every smart device on which it was installed. This is not the only well-known mobile application that has leaked sensitive information. Two years ago, the University of New Haven's Cyber Forensics Research and Education Group ([2]) proved that many frequently-used mobile applications from various domains, such as social media, social meetings, text messaging, etc. were leaking sensitive information such as pictures, videos, locations, etc. Those mobile applications include: *Instagram* (+500 million users), *Tango* (+100 million users), *Nimbuzz* (+10 million users), *MeetMe* (+10 million users), *TextMe* (+ 10 million users), etc. All these findings imply that hundreds of millions of users could be being tracked everyday because of these information leaks.

However, smart phones are not the only smart devices leaking sensitive information. It has also been shown that some smart TVs are sending conversations recorded from their environment ([3]). This is why, in this dissertation, we will not focus solely on smart phones. We will consider smart devices as a whole. Moreover, as information leakage can occur at many different levels (via the device's storage, via the device's network, etc.) and each level would require more than one thesis to be fully explored, we will only focus on data leaked through network of smart devices.

One of the most likely causes of smart device network information leaks is that a lot of users do not even read the authorizations that they give to the various softwares they download onto their smart devices. However, one of the main causes is also the fact that many developers simply do not think about network security or privacy when building their smart device services. For example, they do not use, or misuse, cryptographic algorithms to protect the users' data, they accept certain malicious ads that are combined with their services, etc.

Many researchers are working on warning users about privacy and sensitive data leakage leaked through the network. There are a large number of automated systems that users are already able to use to evaluate the network security of the smart device services they are using. These automated warning systems can be mobile applications (*Norton Mobile Security mobile application* ([4]), *PrivacyHawk* ([5]), etc.), or external services that reroute the tested smart devices' network traffic (Mobilescope ([6])), etc. None of these solutions allow to control different type of smart devices. Even more, they only allow a user to discover its information leaks but not their sources!

As there are already a wide range of automated network information leakage warning systems, we were obliged to define clear and concise objectives in order to stand out. For this reason, we not

only take smart phones into consideration, but smart devices in general. Moreover, we will not solely examine pure information leaks; we will also consider the sources of information leakage. In other words, the factors that lead to information leaks.

The objective of our dissertation is to design a system for detecting the sources of information leaks on networked smart devices. This detection system will allow smart device related developers to test the network security/privacy levels of their work. Additionally, it will also be possible to use this system to warn users about the sources of information leak on their network traffic. Our automated warning system will output the sources of information leaks from which it will be possible to extract pure information leaks.

To achieve this goal, we adopted a specific method. In our system, in order to discover the sources of information leaks, users connect their smart device to a specific WiFi network that we generate. Periodically, we use a reliable method to isolate the network traffic of the specific smart device targeted, and perform several attacks/leak tests on this traffic between the smart device and a certain back-end service. We then generate/update an automated leak sources report for this specific traffic.

This dissertation will follow a specific plan. First, we will provide a non-exhaustive list of the risks afflicting the network of smart devices taken from state-of-the-art literature. Secondly, we will make the link between these network risks and the sources of information leaks. This means proving that these network risks can lead directly or indirectly to information leaks. Thirdly, we will present our detection system for the sources of information leaks. And finally, we will present our experimental results, showing that our detection system for the sources of data leakage actually works on different kinds of smart device (iPhone, Samsung mobile phone, smart TV, etc.).

Chapter 2

Related literature and theoretical focus

In this chapter, we discuss the research and current knowledge that will form the theoretical basis to support our dissertation. In the first section, we talk about our concept of privacy and information leakage in smart devices. In the second section, we discuss the vulnerabilities afflicting smart devices and the applications developed on them. As our dissertation is mainly related to network traffic analysis, we will focus on the risks associated with the network capacities of smart devices. We stress the fact that, as the subject of our dissertation is fairly new, many of our sources of knowledge comes from online resources rather than published works or scientific reviews.

2.1 Privacy and information leakage

In the last few decades, technology has increasingly invaded our daily lives. In this era of easy connectivity, people agree to share information not only on social networks, but also to companies in exchange of services. As a result, and despite the fact that there is some limited agreement on the usage of some of these private data for profiling or advertising, privacy is becoming a growing concern. People feel less and less inclined to trust companies that request private data because of increasing reports of hacking and data theft. For example, the so-called *Panama Papers* incident, which came to light in May 2016, is a prime example of information leakage that violates the privacy of individuals, even though it helps to uncover a certain number of illegal practices([7]).

Before tackling the issue of leakage, we need to define the concept of privacy. In a dictionary, *privacy* is defined as:*"the state of being free from unwanted or undue intrusion or disturbance in one's private life or affairs"*[8]. This definition nicely summarizes the mindset of the users of smart devices, but does not link privacy to the handling of information. Actually, The aspect of privacy that we are really interested in in this dissertation is *information privacy* which is defined as the relationship between the gathering and distribution of data ([9]). The privacy of information must also comply with the expectations of the public with regard to privacy, as well as a number of legal and political aspects.

Information can be gathered at many places (financial, justice case, healthcare, etc.), but we will focus solely on data that is collected through the Internet. We won't linger on the legal aspects of this practice because a detailed description is outside the scope of this dissertation, which is more oriented towards computer science.

Certain legal points are nevertheless worth discussing. Whenever the privacy of an individual is concerned, article 12 of the *Universal Declaration of Human Rights* can be applied. It states the following : "*No one shall be subjected to arbitrary interference with his privacy, family, home or correspondence, nor to attacks upon his honor and reputation. Everyone has the right to the protection of*

the law against such interference or attacks([8])". Unfortunately, this law does not explicitly apply to electronic communications and may eventually be bypassed as, in certain cases, there is some consent from users regarding the collection and sharing of data (companies tend to be rather hazy in term of which data are handled). Moreover, even though this law can be applied in the highest number of countries in terms of privacy (countries that have signed the *Universal Declaration of Human Rights*), there is no other general consensus explicitly concerning *information privacy*. Some nations may have established a few laws on the subject of data protection, but these only apply to their own territory. The issue here is that smart devices are found worldwide and while the handling of data might conform to laws in the county in which it is created, this is not necessarily the case in the other countries where these devices are sold. In certain particular cases, where no law exist, the companies are free to build the systems in whatever way is convenient for them.

Failure to respect data privacy laws is certainly one source of leakage, but even when the laws are applied, some people do not agree that the information being treated is free of leaks as trust in this practice is highly related to each individual's sensitivity. In addition, devices can be subject to external attacks that aim to steal information. As one of the objectives of our dissertation is to find the sources of *information leaks*, we need to agree on a definition. Formally, *information leakage* is defined as a weakness causing systems to reveal sensitive data, such as technical details about the system or user-specific information([10]). Even though this definition is correct from a technical point of view, it does not incorporate the users' point of view. To better take into account what a user should consider as leakage, we will use a broader definition. In the context of this dissertation, *information leakage* is the transmission of sensitive data through unsafe and even safe channels without the explicit confirmation or, at least, understanding of the owner at the time of sending. Two new concepts have been introduced in this definition: safety and users' confirmation or understanding.

The *safety* of communications relies on the security mechanisms used in the network traffic of smart devices. The security issue surrounding communications has been around for quite long time now, since before the creation of smart devices and even computers. When it involves network traffic, a wide range of techniques has been invented to protect the content, but those we are ultimately concerned about in smart devices are *encryption* and *authentication* techniques. If these security mechanisms are used well, no issues should arise. Unfortunately, for various, mostly unjustifiable, reasons (budget, release deadlines, ignorance, etc.), the safety aspect of development is sometimes at least partially overlooked. We will address the risk generated by this negligence in the following section.

The need for confirmation from the user comes from the simple observation that people tend to overlook the users' charter when they use a system. Before using a system, especially software, but this is also true for many smart devices, users must consent to the content of a users' charter. Unfortunately, very few take the time to read these charters and can do nothing but observe the disclosure of private information against their will, even though they have legally agreed to allow this information to be gathered. Of course, some of the data might very well be collected without being mentioned. Eitherway, data collected without confirmation can clearly be considered to be leakage and privacy

violation from the user's point of view.

Finally, as we focus mainly on the network aspect of smart devices, their regulation falls under the domain of the *Internet of Things* (IoT). The *Internet of Things* is defined as the network of physical objects (devices, vehicles, buildings, etc.) embedded with sensors, softwares and connectivity in order to collect and exchange data([11]). The smart devices that we will consider do indeed correspond to this definition. The filed of *IoT* has evolved considerably in the last few years, with new technology introduced at a fast pace. The pace is so fast, in fact, that the laws can not follow while security concerns continue to arise. This field is in high need of standardization and regulation, as highlighted by the *European Commission*([12]). In its current state, *IoT* related companies are free to build devices with the level of security they see fit. The *Internet of Things* needs to universalize data protection systems, but the adopting standards is made difficult because different jurisdictions from different countries must agree to a single way of proceeding.

2.2 Top network risks for smart devices

This section summarizes the state-of-the-art literature we used to identify the leading smart device network risks. The various risks will be given in decreasing order of importance. The literature cited in this section was inspired by the four main, relevant and and best-known sources.

The first, and probably our most important, source is an *OWASP*[1] ([16]) article. It identifies the top ten mobile risks. This list was put together in 2014 to improve smart device security, and was finalized after a 90-day feedback period from the worldwide *OWASP* community (more than 45,000 members).

The second source is an *ENISA*[2] ([17]) article. It identifies the top ten smart phone security risks. This list was put together to assess information security and the privacy risks of using smart phones. Additionally, the ultimate aim of this list was to allow users, businesses and governments to embrace the opportunities offered by smart phones, while minimizing the information security risks to which they are exposed.

The third source is the *Android Hacker's handbook* ([18]). It states several sources of information leaks for android smart phones. The fourth source is the *Secured System Engineering course* by G. Avoine ([19]). This helped us selecting certain weak cryptographic algorithms.

Table 2.1 summarizes the main different smart device risks given by the sources *OWASP*, *ENISA* and the *Android Hacker's handbook*. These main risks are given in decreasing order of importance. From all these risks, we retained only those related to network traffic, as we focus only on the sources of information leaks related to the network traffic of smart devices.

[1]The Open Web Application Security Project is a worldwide not-for-profit charitable organization focused on improving the security of software.

[2]The European Union Agency for Network and Information Security is an expertise center for cyber security in Europe. This agency is located in Greece. It has contributed and still contributes to a high level of network security in Europe.

	OWASP	*ENISA*	*Android Hacker's handbook*
1.	Weak server side controls	Data leakage	Lack of encryption
2.	Insecure data storage	Improper decommissioning	Weak encryption
3.	Insufficient transport layer protection	Unintentional data disclosure	Lack of certificate validation
4.	unintended data leakage	Phishing	Use of plain text after failures
5.	Poor authorization and authentication	Spyware	Inconsistent transport security per network type
6.	Broken cryptography	Network spoofing attacks	-
7.	Client side injection	Surveillance	-
8.	Security decisions via mistrusted inputs	Dialerware	-
9.	Improper session handling	Financial malware	-
10.	Lack of binary protections	Network congestion	-

Table 2.1: Table summarizing the main smart device risks from the different literature sources

After observing and comparing the various smart device risks from each source in Table 2.1, we extracted the main smart device network risks and listed them in decreasing order of importance:

1. **Insufficient transport layer protection**: this corresponds to part of the third risk from *OWASP*, the third risk from *ENISA* and first and fifth risk from the *Android Hacker's handbook*.
2. **Broken cryptography**: this corresponds to the sixth risk from *OWASP* and the second risk from the *Android Hacker's handbook*.
3. **Lack of certificate validation**: this corresponds to part of the third risk from *OWASP*, the sixth risk from *ENISA* and the third risk from the *Android Hacker's handbook*.
4. **Poor authorization and authentication**: this corresponds to the fifth risk from *OWASP*.
5. **Improper session handling**: this corresponds to the ninth risk from *OWASP*.
6. **Client side injection**: this corresponds to the seventh risk from *OWASP*.
7. **Use of plain text after failures**: this corresponds to the fourth risk from the *Android Hacker's handbook*.
8. **Phishing**: this corresponds to the fourth risk from *ENISA*.
9. **Spyware**: this corresponds to the tenth risk from *ENISA*.
10. **Unintended data leakage**: this corresponds to the third risk from *OWASP*, and the third risk from *ENISA*.
11. **Network congestion**: this corresponds to the tenth risk from *ENISA*.

Even if they were related to networks, we did not include the main risks from *Weak Server Side Controls* in the previous list because they were only related to the server side. We did not include other main risks such as *Insecure data storage*, *Security decisions via mistrusted inputs*, etc. because they only concerned hardware risks and were thus not related to the network.

The main goal of this section is thus to summarize the main smart device risks related to network security, as stated by well-known and recognized organizations. We will then, in the following chapter, link these security risks to the sources of information leaks.

2.2.1 Insufficient transport layer protection

According to the *OWASP*, *ENISA* and the *Android Hacker's handbook* sources, *Insufficient transport layer protection* is one of the biggest risks for networked smart devices.

Data is normally exchanged in a client-server way between a mobile application and a server. Here, the problem resides in the transport layer (the fourth level of the OSI model)[3]. The best transport security protocol that can be used today is clearly the SSL/TLS[4] protocol. But even if a mobile application uses it during authentication, this does not mean that it uses it to exchange data after the authentication! This inconsistency can lead to data and session IDs being exposed to an eavesdropper.

This mobile risk can lead to *Privacy information leakage*. In this case, the mobile application transmits some private information to an endpoint via a non-secure channel instead of using the SSL security protocol. The confidentiality of privacy-related data between the mobile application and the endpoint is thus not respected because of this weakness.

In order to see if a mobile application is vulnerable to this kind of risk, it is necessary to answer these questions:

- Are all the connections initiated between the server and the user encrypted?
- Are any sensitive data transmitted via the network without any encryption?
- Is the SSL security protocol used correctly?

This risk can lead to account theft if, for example, the attacker intercepts an admin account.

The *Android Hacker's handbook* ([18]) cites this risk as a source of insecure transmission of sensitive data.

2.2.2 Broken cryptography

According to the *OWASP* and *Android Hacker's handbook*, *Broken cryptography* is a significant risk for networked smart devices.

[3]The Open Systems Interconnection model is a conceptual model that characterizes and standardizes the communication functions of a telecommunication or computing system without regard to their underlying internal structure and technology ([20]).

[4]Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), both often referred to as SSL, are cryptographic security protocols providing security for communications over a computer network.

This risk appears when the network traffic between the mobile application and a server is encrypted with cryptographic algorithms that can possibly be broken. This means that any attacker with access to the network traffic between the mobile application and this server will be able to decrypt it and thus have access to its decrypted content.

It is possible to exploit this weakness because of two main reasons. First, the mobile application uses a process behind encryption/decryption that may be flawed and thus exploited to decrypt sensitive data. Second, the mobile application uses encryption/decryption algorithms that are weak in nature and that can be decrypted directly by the adversary.

Here are different scenarios illustrating this risk:

- *Poor key management processes*: if the keys entered as input for certain cryptographic algorithms are mishandled, even the very best encryption algorithms become useless! This scenario comes from the fact that many people use encryption algorithms correctly, but implement their own protocol when they use them. Here are some examples of this scenario: making the keys available in any manner to the attacker, using hardcoded keys (and thus not regenerating them regularly), etc.

- *Creation and use of custom encryption protocols*: it is common for encryption to be mishandled when trying to create and use a personal encryption algorithm or protocol. It is absolutely necessary to leave that to professionals. It is also necessary to always use modern algorithms that have been accepted as strong by the cryptographic security community.

- *Use of insecure and/or deprecated cryptographic algorithms*: many cryptographic algorithms should not be used because they have been shown to have significant weaknesses and are thus insufficient for modern security requirements. In this context, here are the different cryptographic algorithms/blocks that can be considered: hash algorithms[5], random generators[6], mode of operations[7], and block and stream ciphers[8]. The weak cryptographic algorithms considered by the *OWASP* organization are presented in Table 2.2. The weak cryptographic algorithms considered by *G. Avoine* in his *Secured System Engineering course* are given in Table 2.3.

- *Weak handshake negotiation*: the mobile application and the server negotiate a cipher through the connection handshake. The client (user) manages to negotiate a weak cipher that results in weak encryption of the communication that can thus be easily broken by an adversary. The

[5] A hash function is a function that takes data of variable length as an argument. It outputs a sort of "digest" or "fingerprint" of the argument. This output is shorter than the input.

[6] A random generator generates a sequence of random numbers that cannot reasonably be predicted better than by random chance.

[7] A mode of operation is an algorithm or physical device that uses a block cipher to encrypt messages of arbitrary length to provide a confidentiality property.

[8] Block and stream ciphers are the elementary components of many cryptographic protocols. They are deterministic algorithms that encrypt part of a message at a time (per block of bytes for block cipher and per byte for stream cipher) using a key specified as argument.

Weak hash algorithms	*MD-4*, *MD-5* and *SHA1*
Weak stream ciphers	*RC2*

Table 2.2: Weak cryptographic algorithms considered by *OWASP*

Weak hash algorithms	*MD-5* and *RIPEMD-128*
Weak random generators	*LCG* [9]
Weak mode of operations	*ECB* [10]
Weak block ciphers	*DES*
Weak stream ciphers	*RC4*, *E0*, *A5/1* and *A5/2*

Table 2.3: Weak cryptographic algorithms considered by G. Avoine

confidentiality of the communication between the mobile application and the server is thus not respected because of this weakness.

This risk can lead to privacy violations, information theft, code theft, intellectual property theft, etc.

The *Android Hacker's handbook* ([18]) also cites "weak encryption" as a source of insecure transmission of sensitive data.

2.2.3 Lack of certificate validation

According to the *OWASP*, *ENISA* and *Android Hacker's handbook* sources, *Lack of certificate validation* is an important risk for networked smart devices.

Before defining this smart device risk, it is first necessary to define a certificate and to describe its purpose. According to [21], a certificate is used in asymmetric cryptography. It is an electronic document that is used to prove the ownership of a public key. It contains information about the public key, the owner's identity and the digital signature of the identity that has verified this certificate. The main purpose of a certificate is to prove the identity of a party or simply to securely authenticate a party.

In order to see if a mobile application is vulnerable to this kind of risk, it is necessary to answer these questions:

- Are all the SSL certificates up to date?
- Are all the SSL certificates self-signed?
- Does the mobile application accept user-accepted certificates as authorities.

[9] The *random generator algorithm* LCG (Linear Congruential Generator) is insecure for cryptographic purposes.
[10] The mode of operation ECB (Electronic Codebook) is insecure for long and structured messages.

A scenario illustrating this smart device risk is the following: imagine a mobile application and an endpoint that successfully connect and perform a SSL/TLS handshake to establish a secure channel. However, the mobile application unconditionally accepts any certificate offered by the server. This entirely destroys the mutual authentication between the two parties. The connection between the two parties is indeed vulnerable to a *Man in the Middle attack*[11] through an SSL proxy.

A *Man in the Middle attack* is a kind of network spoofing attack. This attack can be performed thanks to a rogue access point[12] used to tamper with the network traffic of a targeted smart device. In addition, the wireless access point of the targeted smart phone can be changed by a malicious SMS configuration message, which makes the attack incredibly easy!

The ability to decide whether or not a certificate is real (coming from a trusted signing authority or not) is a significant factor for counteracting a *Man in the Middle attack*. To avoid this type of attack, *certificate pinning*[13] can be used. This technique allows users to be absolutely sure that even a valid-looking certificate is indeed the one expected. There are several overheads when applying this technique but it keeps the network connections of smart devices more secure.

Finally, we found an additional source ([22]) stating that the SSL handshake should fail if the certificate provided by a party was not signed by a well-known or widely recognized (= trusted) certificate authority. The main point is that today, mobile applications are performing different levels of certificate validation and using the SSL/TLS security protocol is not enough to preserve data security and integrity. Moreover, mobile applications must absolutely be sure of the identity of the back-end servers they are contacting.

2.2.4 Poor authorization and authentication

According to the *OWASP* source, *Poor authorization and authentication* is a major risk for networked smart devices.

The principle of authentication is extremely important when considering network interactions. It consists in verifying the identity and proof of the identity of a certain party. The principle of authorization derives directly from authentication principle as a user obtains authorization according to who he is and thus how he is authenticated. Nevertheless, some adversaries manage to understand how an authentication scheme is vulnerable and they can fake or bypass the authentication security, thus submitting service requests to the back-end servers, impersonating users. They will then manage to bypass all the interactions with the mobile application. This can also allow the adversary to anonymously execute functions within the mobile applications or back-end servers.

[11] A *Man in the Middle attack* is a cyber attack where an attacker inserts himself between two parties and thus impersonates these two parties.

[12] A rogue access point is a wireless access point installed by a malicious attacker on a secure network without authorization from the administrators.

[13] If you want more information about *certificate pinning*, do not hesitate to visit `https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning`

Mobile application authentication is not the same as that for a traditional web application. A mobile application user is not expected to stay online all the time as in web applications. It is thus necessary to have online and offline authentication.

To detect poor authentication schemes, one possibility is to carry out binary attacks against the mobile application once it is in offline mode. The attacker will then try to execute certain functions in offline mode that need the application to be in online mode. Another possibility is to remove any session tokens from the POST/GET requests and to try and execute any back-end server function. If one of these attacks succeeds, it proves that the authentication scheme of the mobile application is very poor.

Here are other possibilities for controlling authentication schemes:

- Ensuring that all authentication requests are performed server-side.
- Ensuring that no "important" data are loaded before successful authentication.
- Ensuring that the authentication is not infinitely persistent (or not too long).
- Ensuring that no spoofable values are used to authenticate the user and thus verify that no important values for authentication are transmitted unencrypted through the network.

Here are a few scenarios illustrating this risk:

- Certain developers assume that only mobile devices containing their mobile applications will be able to access the back-end servers and execute specific functions. This is absolutely not the case! It may be interesting to try to capture certain specific packets emitted from a specific user smart device and to replay them from another device in order to execute certain functions on the back-end servers. This could result in some of this user's private information being changed.
- Due to usability, several mobile applications use 4-digit passwords and store the hash on the back-end servers. It could be interesting to control whether the hash for this password is sent unencrypted via the network. If this is the case, recovering this password would be easy using rainbow hash tables(cf. TMTO attacks[14]).

This risk can lead to fraud, information theft, etc.

2.2.5 Improper session handling

According to the *OWASP* source, *Improper session handling* is a significant risk for networked smart devices.

[14]Time memory tradeoff attacks are mostly used to guess a user's password. They can sometimes be more time- or space-efficient than other password attacks such as brute force attacks (which consist in trying all password possibilities) because it balances "memory" and "time" during the attack. It carries out some pre-computation before the attack.

Mobile applications use session tokens to maintain state over stateless protocols such as HTTP, SOAP, etc. in order to facilitate stateful transactions between a user and a mobile application's back-end server. Back-end servers use session cookies in order to keep users authenticated when they have been successfully authenticated by the mobile application. Improper session handling can occur when the session token is unintentionally shared with the adversary during a transaction between the user and the back-end servers.

An adversary that has access to a session token is able to impersonate a user by submitting this token to the back-end servers. In the worst case, the adversary can then impersonate an administrator and thus cause damage at several levels of the mobile application's infrastructure.

Here are a few scenarios illustrating this risk:

- *Failure to invalidate sessions on the back-end*: a lot of developers invalidate the session on the mobile application but not on the server side. This leaves a major window of opportunity for attackers that can use HTTP manipulation tools.
- *Lack of adequate timeout protection*: any mobile application must have adequate timeout protection on the back-end components. This can help prevent attackers from stealing a session. Typical guidelines are 15 minutes for high security applications, 30 minutes for medium security applications, 1 hour for low security applications.
- *Failure to properly rotate cookies*: it is important to reset cookies during authentication state changes. Authentication state changes include events such as switching from an anonymous user to a logged-in user, switching from one logged-in user to another, switching from a regular user to a privileged user, timeouts, etc.
- *Insecure token creation*: it is important that the tokens be sufficiently long, complex, and pseudo-random in order to be resistant to guessing attacks. It is thus necessary to use well-established industry-standards in order to generate these tokens.

This risk can lead to fraud, information theft, etc.

2.2.6 Client side injection

According to the *OWASP* source, *Client side injection* is an important risk for networked smart devices.

Smart devices receive a large amount of data via the network. This makes them vulnerable to injection attacks. Here is a non-exhaustive list of possible injection attacks:

- *SQL injection attack*: SQLite, which is the default database of a lot of smart devices, can be subject to injection as in web applications. These types of injection become very annoying when the database stores personal information such as payment information, contact information, etc.

- *JavaScript injection attack*: smart device browsers are also subject to JavaScript injection attacks. Mobile browsers sometimes have access to the mobile application's cookies, which can lead to session theft.

- *Binary attack*: Mobile malware or even malicious mobile applications can perform binary attacks against the presentation layer of the OSI model or even the actual binary of the mobile application executable.

Protecting a smart device against this risk requires controlling each input and applying a "control validation" for each of them. This can become complex as the number of inputs from the network is growing. Another way to control this risk is to control the mobile application code by executing code analysis tools to trace the data flow through the application. Certain penetration tools can also be used to control this kind of risk.

Here are a few scenarios illustrating this risk:

- *SQL injection*: data retrieved from back-end servers can contain some malformed data that can contribute to a local SQL injection within the mobile application's local database.

- *Cross-Application Scripting attack*: malicious data transiting between mobile applications can lead to buffer overflows, allowing malicious code execution.

- *Cross-Site Script attack*: malware or other mobile applications can modify HTML files, which can result in the execution of malicious JavaScript code in the presentation layer of the OSI model. This can lead to information theft.

This risk can lead to fraud, privacy violations, information theft, etc.

2.2.7 Use of plain text after failures

According to the *Android Hacker's handbook* source, *Use of plain text after failures* is a major risk for networked smart devices.

This risk may appear in two cases. The first appears when certain errors or unintended actions are performed by the user and an error message (via the network packets) is sent to the user by the back-end server. This error message contains private information sent unencrypted via the network. The second appears when private information is leaked via logged messages.

This is a risk that really should not be underestimated! For example, imagine a mobile application in which a user enters his password. In a first scenario, the user enters a wrong password. The back-end servers, after having processed the user credentials, sends an unencrypted error message to warn the mobile application to display the message 'Try again!' to the user. The error message sent by the back-end servers could contain the user login (email), the wrong user password (which can be close to the real one!), etc., all unencrypted, which is clearly a serious sensitive risk!

In a second scenario, the user enters his real password. By validating the user's credentials, the mobile application displays the password in a log message. This is also a serious sensitive risk!

Taking into account the previous examples, even if they are really simple, we can notice how important it is for developers to be careful when establishing error or log messages.

This risk can lead to privacy violations.

2.2.8 Phishing

As mentioned by *ENISA*, *Phishing* is an important risk for networked smart devices.

Phishing consists in acquiring sensitive information from a user, often credentials, by impersonating an entity that is considered to be trustworthy by the owner of the data ([23]). This kind of attack often relies on making a user accept the data transfer by executing a software that has either been corrupted or specially designed for data extortion and disguised as a reliable one.

Phishing is a well-known practice on traditional computer systems and may represent an even bigger threat for smart devices, especially if the device allows the installation of softwares designed by third parties. For example, smart phones are widely exposed to this threat because of the presence of *app-stores* that allow anyone to publish application without doing a sufficient amount of authenticity checks.

2.2.9 Spyware

According to *ENISA*, *Spywares* are an important risk for networked smart devices.

Much like phishing softwares, spywares are a type of malicious softwares (also called malwares[15]) that target sensitive informations about the device user. However, a much larger range of data is collected by theses softwares. Besides credentials and sensitive files, spywares also target behavioural information about users (web site visited, location, etc.) that can be used for marketing purposes (profiling, targeted ads) among other things.

The work of spywares is made easy by the fact that smart devices provides covert channels and backdoors through which these malicious softwares can leak the data to a receiving endpoint (for example an attacker, a dishonest advertising company, etc.).

2.2.10 Unintended Data Leakage

According to the *OWASP* and *ENISA* sources, *Unintended Data Leakage* is an important risk for networked smart devices.

[15] "Malware" is an umbrella term used to refer to a variety of forms of hostile or intrusive software, including computer viruses, worms, trojan horses, ransomware, spyware, adware, scareware, and other malicious programs. It can take the form of executable code, script, active content, and other software.[24]

Unintentional leakage of information is often due to the collection of data by the smart devices or the softwares installed on it, without the consent of the user. It also results from a lack of protection during the storage of this information, making it accessible to any software installed on the smart device.

Another aspect of this risk is the fact that user are often unaware of the security and privacy mechanisms implemented by the softwares collecting data. The softwares sometimes propose options to limit the disclosure of the information, but users, as they are unaware of these possibilities, expose themselves to this vulnerability and virtually agree to the disclosure.

2.2.11 Network congestion

According to the *ENISA* source, *Network congestion* is an important risk for networked smart devices.

The increasing number of smart device users and thus of the mobile internet services can sometimes lead to network congestion. Network congestion means that network resources become overloaded due to the number of users using the internet. Network congestion leads to network unavailability for the end-user.

Network congestion can happen because the mobile applications are constantly polling the network for updated information. For each bit of data sent, a large number of signaling messages are also sent (e.g.: keep-alive messages). A normal smart phone sends on average 8 times more signaling messages than a laptop with a USB dongle.

To address the network congestion problem, governments and operators continue to work together to explore the possible options. One promising solution is the "quality of service"[16] (QoS) provisions for emergency service levels of mobile data.

This risk can lead to network unavailability for some end-users.

[16] "Quality of service" allows the different network services such as streaming video services, phone services, website services, etc. to set a minimum quality of service necessary for them to work correctly. This solution will then act on the network transport layer of the OSI model to provide the minimum quality of service required by the specific service.

Chapter 3

Information leaks and potential sources

In this chapter, we will explore the potential leak sources we extracted from the state-of-the-art literature described in the previous chapter. In reading this literature, we identified four main sources of potential leaks: *Lack of encryption*, *Weak encryption*, *Weak authentication* and *Covert channels*. We are thus going to describe these sources of potential leaks in a more formal way, but also prove that they lead directly or indirectly to information leaks.

3.1 Lack of encryption

This leak source appears when the network traffic of a smart device is not or only partly encrypted: the SSL/TLS security protocol is not used or not used enough. This means that it is possible for anyone to read the unencrypted information through the network. The information can be represented by any of the following information types: text, user credentials, password hash, image, video, etc. The problem lies in the fact that this kind of information can be sent unencrypted via the network. For example, a WiFi user in a coffee shop using that kind of smart device in such a situation would spray all his sensitive information all over the shop. Anyone present in the shop could thus read his personal information by controlling his network traffic.

If a specific word, sentence, user credential, image, etc. is shown to be present in the network traffic of a mobile application, this could prove that the mobile application did not use enough cryptographic algorithms to protect sensitive data. This leak source, once detected, thus directly leads to information leaks through the network traffic of a user.

3.2 Weak encryption

This leak source appears when an attempt at encryption (for example using the SSL/TLS security protocol) is made but the encryption is easy to bypass. In this case, we have different levels of weak security problem. In reading part of the literature ([19]), we classified them in order of magnitude:

- Use of *weak random generator algorithms*. In reading the literature, we noticed one weak random generator: LCG (Linear Congruential Generator).
- Use of *weak hash algorithms*. In reading the literature, we noticed three weak hash algorithms: MD-4, MD-5 and RIPEMD-128.
- Use of *weak block cipher encryption algorithms*. In reading the literature, we noticed one weak block cipher encryption algorithm: DES.

- Use of *weak stream cipher encryption algorithms*. In reading the literature, we noticed two weak stream cipher encryption algorithms: RC2 and RC4.

When one of these weak cryptographic algorithms is used to encrypt the network traffic, it allows, in some cases, the encrypted traffic to be recovered by attackers. For example, using the DES weak block cipher encryption would expose the DES encrypted network traffic to being recovered in less than one day (cf. COPACOBANA/RIVYERA attack which took place in 2006).

If one of these weak algorithms was shown to have been used by a mobile application, this would prove that the mobile application uses a weak encryption system. This could lead to information leaks if the system was attacked.

In this case, the leak source does not directly lead to information leaks. However, it could do so after a certain amount of work has been done to break the cryptographic system protecting the personal information present in the weak encrypted network traffic. This leak source, once detected, thus indirectly leads to information leaks through the encrypted network traffic of a user.

3.3 Weak authentication

This leak source appears mainly in three different cases: *lack of adequate timeout protection*, *lack of certificate validation* and *lack of replay protection*.

If one of these cases proves applicable to a mobile application, this would prove that the mobile application uses a weak authentication system, which could lead to information leaks if the system is attacked. This leak source, once detected in one of the three different cases, thus indirectly leads to information leaks through the network traffic of a user.

3.3.1 Lack of adequate timeout protection

Adequate timeout protection is required to prevent attackers from stealing a user's session. If there was no timeout protection, this would give an attacker an infinite amount of time to guess the token identifier of a specific user in order to impersonate him. It is thus used to prevent an attacker from impersonating a user. Typical guidelines for adequate timeout protection are 15 minutes for high security applications, 30 minutes for medium security applications and 1 hour for low security applications.

3.3.2 Lack of certificate validation

In the literature about certificate validation, we established three main aspects of certificate validation:

1. verifying the expiration date of the certificate.

2. verifying that the certificate is not self-signed except if it is from a certificate authority (CA). This part thus consists in verifying that the certificate's signer is not the owner of the public key present in the certificate.

3. verifying that the signing authority can be trusted because it is a widely used or a well-known authority. It is also necessary to verify that this signing authority has not been compromised.

The two last certificate validation steps are very important as they prevent an attacker from performing a *Man in the Middle attack*[1] between a smart device and a certain back-end system. In this case, performing this kind of attack would allow the attacker to gain access to the encrypted information that the parties are sending each other.

If a smart device does not verify if a signing authority of a certain certificate can be trusted, anyone can pretend to be any back-end services they want, or even the signing authority itself! This would lead to the possibility of performing a *Man in the Middle attack* as shown in figure 3.1. In this figure, the attacker intercepts all the network traffic between the user's smart device and the back-end services of a specific application. During this attack, the attacker succeeds in impersonating the back-end services/user's smart device because of two main reasons: the smart device does not verify whether the fake certificate sent by the attacker is self-signed, and it does not verify whether the signing authority (the attacker) can be trusted (if it is a certificate authority).

In practice, performing a *Man in the Middle attack* is far from simple. It is necessary to reroute all the network traffic of the smart device targeted to the attacker's device. This can be done in changing some WiFi configurations of the smart device targeted (cf. Manual document of our *leak sources detection system*[2]). For this step, it is not necessary to have physical access to the targeted device as it can be done by sending a malicious text message to this device (as stated in Section 2.2.3).
It is also necessary to install a fake certificate on the smart device targeted for it to trust this certificate. For this step, it is thus necessary to have physical access to the targeted device. However, this can be achieved during a *lunchtime attack*[3]. Uploading a fake certificate to a user's smart device only takes a few minutes (cf. Manual document of our *leak sources detection system*)! Moreover, to extract the user's encrypted network traffic, it is also necessary for him to be connected to the same WiFi as the attacker.

Only a few minutes of physical access to a smart device would thus allow you to gain access to all or part of the encrypted content of a user's network traffic (if the user is on the same WiFi as yours). This becomes really annoying if personal information such as credit card information, passwords or just the personal characteristics of a user are present in its encrypted network traffic. Being able to perform a *Man in the middle attack* can thus lead to serious consequences such as user's personal information leaks!

[1] A *Man in the Middle attack* is a cyber attack in which an attacker inserts himself between two parties and thus impersonates these two parties.

[2] This manual is present beside the source code.

[3] The term "lunchtime attack" refers to the idea that a user's smart device is accessible to an attacker while the user is out for lunch.

Figure 3.1: Man in the Middle attack: example

It is therefore extremely important to validate the certificate as it prevents attackers from forging certificates or using compromised ones to impersonate certificate authorities.

Finally, performing a *Man in the Middle attack* will give us two specific pieces of information. First, it will tell us if the service tested was using a weak authentication mechanism and thus inform us about a specific leak source (*Lack of certificate inspection*). Second, it will allow us to examine the encrypted content of the network traffic produced by the specific service we are testing. We may find personal information about the user that is not needed by the service but that is nevertheless transmitted to its back-end servers. In other words, this can help us find information leaks!

3.3.3 Lack of adequate replay protection

Adequate replay protection such as one-time passwords (OTP) or challenge/response protocols (see the course by Gildas Avoine (Secured System Engineering)[25]) is required to prevent an attacker from replaying an entire user action he has captured. For example, without adequate timeout protection, it is possible to replay a payment performed by a *Paypal* user. This could thus have serious consequences.

3.4 Covert channels

The most difficult information leaks to find are in *covert channels*. To explain the preceding statement, we will first define the meaning of *covert channels*.

For those familiar with the principle of *covert channels* in security: *"A covert channel is a type of computer attack that allows the communication of information by transferring objects through existing information channels or networks using the structure of the existing medium to convey the data in small parts. This makes conveyance through a covert channel virtually undetectable by administrators or users."*[26]; we do not use this exact definition. In the context of this dissertation, a *covert channel* literally represent *hidden* or *unexpected* network traffic between the device and a (distrusted) third party, using conventional or crafted means in order to exchange data (confidential information, files, statistics, etc.). *Covert channels* are the result of several vulnerabilities. These channels expose the devices to *client side injections*, *phishing*[4], *spywares* and *unintentional data leakage* by offering them

[4]Phishing is the attempt to acquire sensitive information such as usernames, passwords, and credit card details (and

an easy entry point.

We note that unlike the potential sources above, where information leaks may be found in the network packets once one of the vulnerabilities has been detected, in the case of *covert channels*, we can merely detect the presence of the channels (the means of detection will be discussed in a further section). We must rely on extraction methods designed for one of the previous sources to possibly find the leaks caused by the behaviour of covert channels. Nevertheless, detecting *covert channels* make it possible to reveal the presence of other threats (phishing, spywares, etc.) and is therefore a sensible feature to consider in our dissertation.

Now that we have defined the kind of network traffic we are targeting, we will highlight the sources of information leakage and the difficulty in exposing these leaks. In the data exchange involving *covert channels*, information leaks could be found in the data itself, but it is not the only source. We will consider the three following means of leakage: statistics, malwares, unwanted softwares and backdoors.

3.4.1 Statistics

A common practice in smart devices (applications) is to gather statistics about the client. Statistics can be information about the client or about his usage of the device. The purpose of the statistics is variable (enhance the user's experience, estimate usage, target the right market, etc.). Nevertheless, in the current age of excessive information gathering, intentionally or not, maliciously or not, statistics can lead to *phishing*.

In the case of statistics, the challenge comes from the fact that one can not simply consider all statistics collectors as malicious or all information leaks as intentional. For example, Google is known to obtain lots of statistics, as well as for providing to private individual the possibility of collecting custom statistics using Google's libraries like *google analytics*[27]. It becomes difficult to differentiate statistics that lead to *phishing* from those that are harmless.

3.4.2 Malware

Covert channels can be used to install malwares[5] on a smart device. There are many varieties of malware and many can lead to information leaks. For example, there is *spyware* which collects information about clients and sends it to third parties[28]. Even without the information leaks, this kind of use of *covert channels* is a major threat because viruses can irrevocably damage smart devices.

The problem with malwares is their hiding schemes. Malwares are hard to detect and their

sometimes, indirectly, money), often for malicious reasons, by masquerading as a trustworthy entity in an electronic communication.[23]

[5]'Malware' is an umbrella term used to refer to a variety of forms of hostile or intrusive software, including computer viruses, worms, trojan horses, ransomware, spyware, adware, scareware, and other malicious programs. It can take the form of executable code, script, active content, and other software.[24]

behaviour is hard to predict. We also need to mention the fact that anti-virus coverage is not available for all kind of smart devices. Smart phones and smart TVs are slowly opening up to anti-virus software, but it is still not a common practice as it is for computers (with a Windows OS). Moreover, anti-virus softwares are not all powerful, they can only detect known malwares and known suspicious behaviours[29]. New malwares are free to roam, helped by the lack of coverage that increases the delay before their detection. One particularity of some malwares is their ability to take control of other softwares and, thereafter, use the legitimate communication means of these softwares to cause the leaks. Information leaks caused by malware are thus at least as hard to prevent as detecting the malware.

3.4.3 Unwanted software

As with malwares, *covert channels* can be used to download *unwanted* softwares. An unwanted software is define as a software whose content or behaviour is not what is intended (not as advertised) without necessarily being a malware[30]. For example, the software may be installed without the customer's approval, gathers data about the client, install other softwares or advertisements, etc. These softwares often do not disclose how they handle processed information.

In the case of unwanted softwares, the lack of knowledge on the behaviour of these softwares and how the information it gains access to is handled, is the main obstacle. Once installed, the software is considered to be legitimate by the device. It is hard to tell upfront if these software will cause leaks. The concern here is a matter of trust.

3.4.4 Backdoor

One form of *covert channels* is backdoors. Backdoors are means of bypassing the security mechanisms of a computer program. These entry points in the system are either installed by malicious softwares or by the programmer for troubleshooting purposes ([31]). In both case, the presence of backdoors is a major threat as they offer any attacker an opportunity to access (sometimes take control of) the device. Any compromised device may be subject to data theft as well as data or software injection.

Detecting backdoors proves to be quite difficult as their main feature is to avoid security mechanisms. Often backdoors hijack computer services (ftp, ssh, ...) and protocols (DNS, SCTP, ...) in order to carry the data. In the case of backdoors, we must mainly look for excessive use of services or protocols that are commonly used less. Nevertheless, identifying such suspicious behaviour does not necessarily indicate the presence of backdoors. An additional step would require to analyse the content of the data transiting trough these suspicious connections. Unfortunately, this step cannot necessarily be automated because the data in hijacked services or protocols are often hidden (e.g.: the data may be put in headers rather than payloads).

Chapter 4

A detection system for the sources of information leaks

The main goal of our dissertation was to build an automated system to warn users about the sources of information leaks present in their smart devices' network traffic.

In this chapter, we are going to present our automated warning system. An overview of the system's architecture will be presented first. We will then study the different modules of the system architecture in detail and explain their methodology and how they are implemented.

4.1 System architecture

In Figure 4.1, we see an overview of our entire system architecture. The system architecture is divided into six main modules: *Traffic capture*, *Lack of encryption module*, *Weak encryption module*, *Weak authentication module*, *Covert channels module* and *Automated report generation*.

The goal of the first module, *Traffic capture*, is to capture the network traffic of a specific user in a reliable and isolated manner. Reliable means capturing all the user's network traffic and thus minimize packet loss. Isolated means that we do not want to capture traffic from other users. In this module, we have two types of traffic capture: basic traffic capture and the traffic captured by performing a *Man in the Middle attack*. Our system allows the network traffic to be captured and processed live or to be first captured, and then processed later (cf. Manual document of our *leak sources detection system*[1]). The live capture means simply that our system allows a user to test certain mobile applications and wait directly for an automated report of its traffic leak sources without performing any additional tasks. The automated report will then be updated every five minutes thanks to the network traffic captured during this period of time.

The second module, is the *Lack of encryption module*. It deals with the *Lack of encryption leak source* and has two goals. The first is to find unencrypted key words in the network traffic of the user. The second is to find unencrypted files in the network traffic of the user. This module counts one global parameter that must be set by the user. This global parameter is referenced by the blue circle "**Search key words**" in Figure 4.1. The user must enter as the global parameter in our system personal search key words including his names, address(es), phone number(s), credit card number(s), password(s), etc. These key words will only be used by this module and the *Weak authentication module*. It will not be accessed by anyone or any other modules.

The third module, the *Weak encryption module*, deals with the *Weak encryption leak source*. Its goal is to find whether any weak cryptographic algorithms were used to secure the network traffic

[1] This manual is present beside the source code.

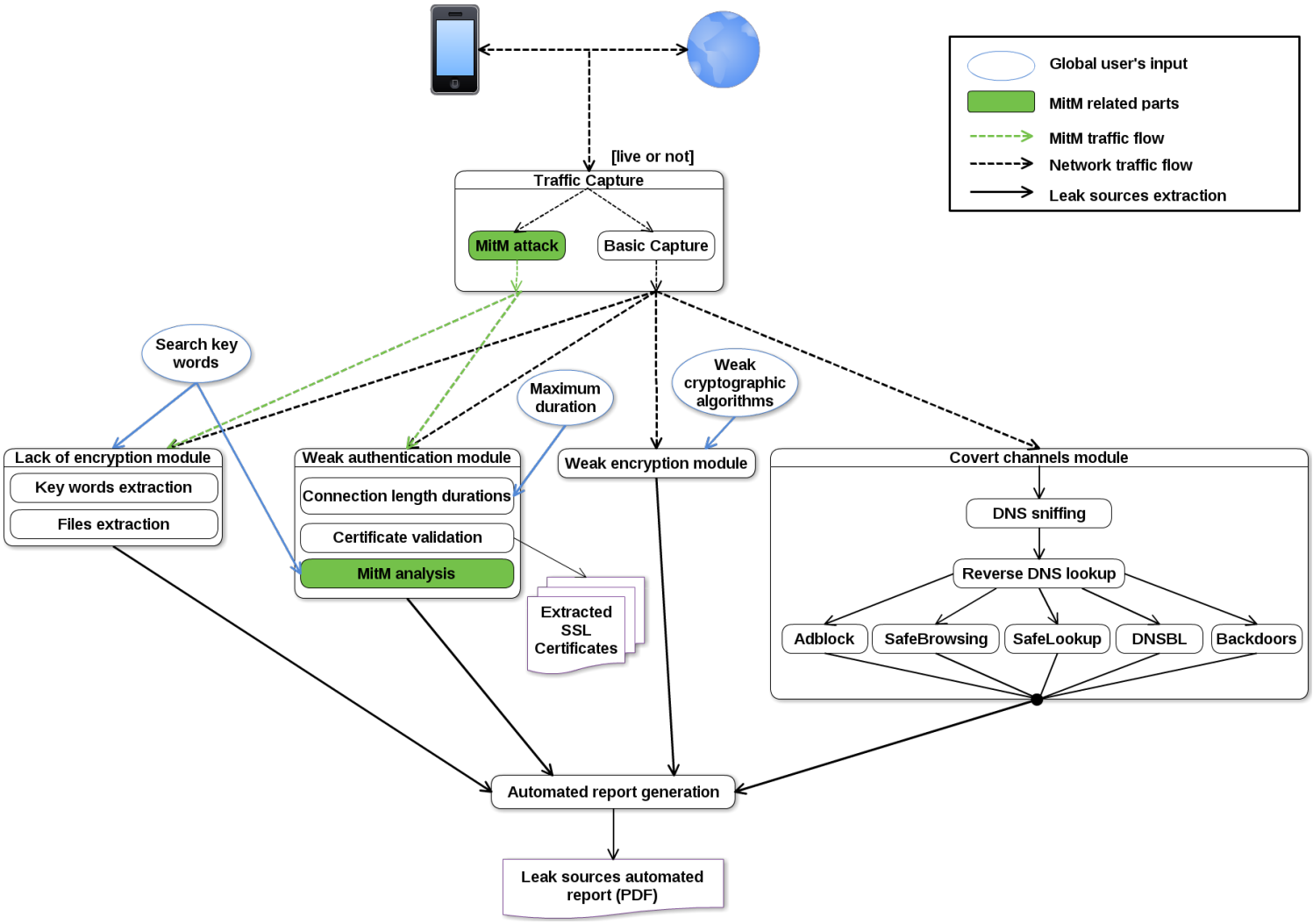


Figure 4.1: System architecture

considered. It has one global parameter represented by the blue circle "**Weak cryptographic algorithms**". This global parameter already contains some weak cryptographic algorithms that we found to be weak. It can also be completed by the user. Users can effectively enter, as input for this module, the weak cryptographic algorithms (used by the SSL/TLS security protocol) they consider to be weak.

The fourth module, the *Weak authentication module*, deals with the *Weak authentication leak source*. It has three goals. The first is to verify that the duration of the connections of the network traffic doesn't exceed a specific global parameter represented by the blue circle "**Maximum duration**". As stated in Section 3.3.1, it is recommended that this parameter be set to one of three possible values: 15 minutes for high security applications, 30 minutes for medium security applications, and 1 hour for low security applications. The default value is 15 minutes. However, if needed, this global parameter can be set to other values. The second goal of this module is to control whether the certificates presented by the back-end servers contacted by the smart device are valid or not. It assesses the validity of the SSL certificates present in the traffic capture. A certificate is valid if its date of use does not exceed its expiration date and if the certificate is neither self-signed nor signed by a distrusted certifi-

cation authority (see Section 3.3.2). The third goal of the module is to perform an "*MITM (Man in the Middle attack) analysis*". This analysis consists in controlling whether or not the *MitM attack* was successful, but also to control whether any user personal key words were found in the deciphered traffic obtained thanks to this attack. These key words must be entered by the user as the global parameter for the system (the same global parameter as for the *Lack of encryption module*).

The fifth module, *Covert channels*, deals with the *Covert channels leak source*. The goal of this module is to identify connections between the smart device and malicious entities. Malicious entities are ip addresses and domain names that lead to phishing, malware or unwanted software injection and backdoors (see Section 3.4). The module is divided into several sub-modules. The first two sub-modules (DNS sniffing and reverse DNS lookup) are used to extract ips and domains contacted in a certain network traffic, while the four remaining sub-modules (safebrowsing, safelookup, adblock, dnsbl) are used to check the extracted ips and domains contacted in a certain network traffic against databases of known malicious entities. Finally, the last sub-module gathers information about potential backdoors.

The sixth and last module, *Automated report generation*, deals with the generation of the leak sources automated report. Its goal is to generate a leak sources report thanks to four different inputs obtained from the four different leak source modules. Thanks to the various types of data collected from the different leak source modules, its job is to summarize these data and generate the automated report.

The following sections will describe the method applied and provide details about the implementation behind each of the modules. We will also justify the choices we made when more than one option was available. At the end of this chapter, we will present the complexity of each module and then the complexity of the entire system.

4.2 Traffic capture

In this section, we describe the hardware and software needed to perform the *basic traffic capture* and the *MitM traffic capture*. We thus describe the steps we took before processing any network traffic to discover any sources of information leaks. Moreover, we outline the different possible approaches we considered and the approach we chose to perform these two types of traffic capture.

At the end of this section, we summarize our entire method for capturing the two types of network traffic and introduce our leak source analysis method.

4.2.1 Possible approaches

To choose the best approach (one that is both reliable and isolated) for capturing network traffic from a specific smart device, we describe the four different steps necessary when capturing the two types of network traffic. We will also describe the various choices possible at each step.

1. Choose the environment for performing the tests. We had two different possibilities: performing the tests in the university computer science labs or at home.

2. Choose the hardware (the device) we should use for capturing the network traffic in an efficient and isolated way. The goal was to minimize network packet loss but also to isolate a specific network flow between a specific user and a specific smart device. We had several possibilities for choosing our hardware: a computer, a Raspberry pi (model 1) or an ODROID server.

3. Choose the operating system best suited to this task. We had three main choices: Windows, Mac or Ubuntu.

4. Choose the software to capture the *basic network traffic*. For this choice, we found an incredible number of software choices ([32]). We nevertheless retained three: *Wireshark*, *NetworkMiner* and *Tcpdump*.

5. Choose the software to capture the *MitM network traffic*. We found two possible software packages to perform this type of capture on smart phones: *Mitmproxy* ([37]) and *Fiddler* ([33]). However, we did not find any software packages to perform a *Man in the Middle attack* on other smart devices such as smart TVs.

4.2.2 Choices and justifications

For the first step, we chose to work from home because we had to have access to the internet modem in order to reroute the network traffic of a specific networked smart device.

For the second step, we chose to use the ODROID server as the hardware for our project because it contains more memory (1 GB) than the Raspberry pi (256 MB) and has a more powerful processor. Moreover, we detected a non-negligible loss of traffic network captured when using the Raspberry pi. We indeed tested the bandwidth of the WiFi that could be generated by the servers thanks to the *iperf* network performance tool ([34]) and obtained a bandwidth of 16.5 Mbits/sec. for the ODROID server and a bandwidth of 10.3 Mbits/sec. for the Raspberry pi server. The ODROID server was thus definitely the best solution.

We chose to work with a server and not our own computer because the goal was to work in the most isolated environment possible in order to deal only with the network traffic of the targeted smart device. For example, using a computer, we did not want to filter network traffic by source addresses or destination addresses of the targeted smart device. It is indeed possible for a smart device application to send packets without source addresses (these packets would be filtered out with the filtering method...). Our goal was thus to redirect the network traffic of the smart device to the ODROID server which should capture it and process it. Moreover, we chose to work with a server in order to provide our leak source detection system as a black box that could be installed and plugged to an internet modem. This black box could then be used to detect information leak sources in the network traffic of any user, not especially computer scientists.

For the third step, we chose to use the Ubuntu operating system because many of the tools we wanted to use were available only on Ubuntu. Moreover, it was easier to build this operating system on the ODROID server.

For the fourth step, we chose to use *Wireshark* as our tool for capturing network traffic for several reasons. The first was that it could be called from the Shell terminal using software directly derived from this tool: *tshark*. The second reason was that it was possible to install it on Ubuntu. The third reason was that it also provided a graphic user interface that allowed us to have a global view of network traffic. The fourth and final reason was that we had already used this tool and we thus did not need to spend time learning how to use it. We did not choose *NetworkMiner* because it was only available on Windows. We did not choose *tcpdump* because it provided limited protocol decoding and had no graphic user interface.

For the fifth and final step, although *Fiddler* had an attractive graphic interface, we nevertheless chose to use *Mitmproxy*. The reason was that *Mitmproxy* could be called from the Shell terminal while *Fiddler* could not. This almost allowed us to automate the *Man in the Middle attack*!

4.2.3 Requirements

For the traffic capture, we used the following devices:

- A smart device (smart phone, smart TV, etc).
- A USB stick able to generate WiFi.
- An ODROID server running on Ubuntu.
- A hard drive with several hundred GBs.
- An ethernet cable
- A physically-accessible modem able to access the internet.

4.2.4 Summary of the traffic capture method

On Figure 4.2, you can see a summary of our method for capturing the two types of network traffic in an efficient and isolated manner. In this figure, the *black box* receives the network traffic from the smart device, saves it to a hard drive and outputs it to the internet. This *black box* can also receive network traffic from the internet and send it to the smart device.

The only thing needed to change the type of traffic capture is to change the destination address contacted by the smart device. Nothing need to be done to capture the *basic network traffic*. However, to capture the *MitM network traffic*, it is necessary to set the proxy address to be contacted by the smart device. For example, it was *192.168.42.1:8888* in our case. The software *Mitmproxy*, running on the ODROID server, then captured the network traffic at this address.

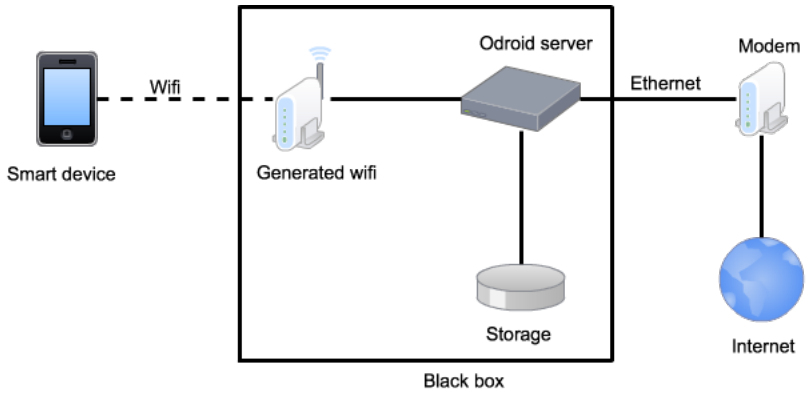


Figure 4.2: Network traffic capture model

In order to explain this model as simply as possible, we describe the path followed by the network traffic from the smart device to the back-end servers of the service tested (to the internet).

First, the network traffic generated by the smart device is sent to a generated WiFi. This generated WiFi is maintained by a WiFi USB stick[2]. We configured the ODROID server ([35]) such that it generates a WiFi called *thesis*. The ODROID server is itself connected to a modem via an ethernet cable, thus giving it access to the internet.

Secondly, the network traffic of the smart device is received on the ODROID server. It is then captured either by the *tshark* software for the *basic capture* or by the *Mitmproxy* software for the *MitM capture*. After having been captured, the network traffic is saved on a hard drive (*Storage* in Figure 4.2) connected via USB to the ODROID server. It is finally transferred to the internet. This thus means that the network traffic of the smart device is finally transferred by the modem to the back-end servers of the service tested.

The network traffic that comes from the internet to the smart device tested follows the exact reverse path.

If you want more information about the setup of the system or how we configure a smart phone to perform the *Man in the Middle attack*, do not hesitate to have a look in the *Setup* and *Manual* documents of our system (present beside the source code).

As you have probably understood since Section 3, the goal of our dissertation is to find some sources of information leaks. For each type of leak source, we found some tools or wrote some scripts which helped us to detect any packets that could directly contain leaked information, or help detect any security weaknesses which could lead to information leakage.

Our analysis method consisted in controlling each of the leak source we described in Section 3.

[2]Model D-Link N300

4.3 Lack of encryption module

To control the *Lack of encryption leak source*, the goal was to determine whether it was possible to find a word, a sentence, a user identifier, a password hash or even a file (image, video, etc) unencrypted in the network traffic of a smart device.

4.3.1 Method

To control this leak source, our method had three steps.

1. Control each unencrypted traffic network packet in order to know whether a certain key word is included in it or not.
2. Use a tool called *Foremost* ([36]), capable of extracting any files lying unencrypted or partly unencrypted in network traffic.
3. Use a tool called *Net-creds* ([37]), capable of extracting user identifiers and password hashes.

For the first step, the user will have to give his personal key word information in order to have a pertinent key word search in the network traffic. Those personal key words will be divided into categories such as *name*, *address*, *payment*, etc. We have also added some general key words that will be searched in addition to those personal key words.

For the second, as it is more general, we do not need any user interactions.

For the third step, we only used this tool manually as we did not have time to adapt it for an automated check. This step thus will not appear in the implementation part of this leak source.

4.3.2 Implementation

In order to control this leak source, we used the python libraries *dpkt* and *socket*.

To find a given String in the network traffic of a user, we built a script able to check whether a certain String was present in the payload of each packet in the network traffic of his smart device.

To extract files from an unencrypted or partly unencrypted network traffic, we ran the tool *Foremost* ([36]). This tool was able, given a pcap[3] file, to extract all the unencrypted or partly unencrypted files such as images and videos. Moreover, this tool could be called from Shell script.

In order to control this leak source, we designed the following steps to be performed automatically for each network traffic tested:

- Run a script we built to indicate the unencrypted network packets containing one or several key words such as an email address, contacts, phone numbers, etc.

[3] A pcap file is a file generated by the tool *Wireshark*. This type of file contains network traffic saved by the same tool.

- Run the tool *Foremost* in order to extract all the unencrypted files including pictures, videos, text files, etc.

4.4 Weak encryption module

To control the *Weak encryption leak source*, the goal was to determine whether any weak cryptographic algorithms were used to encrypt the network traffic of a smart device.

4.4.1 Method

To establish this method, we had to dive a little into the documentation on the SSL/TLS security protocol. We discovered that during the SSL/TLS handshake, the source party (which initializes the connection between the two parties) sent all the cryptographic algorithms understood to the destination party. Afterwards, the destination party replied with the cryptographic algorithms it preferred to the source party. The cryptographic algorithms adopted for this connection were thus included in the reply of the destination party.

The method is composed of the following steps:

1. Find the cryptographic algorithms agreed on and used by both parties to secure their mutual connection.
2. Control whether any weak cryptographic algorithms were present in the ones used by both parties.

In order to find weak cryptographic algorithms, we are going to search for them among three main cryptographic algorithm types: *hash algorithms*, *block cipher encryption algorithms* and *stream cipher algorithms*. The weak cryptographic algorithms we extracted from the literature ([19]) are summarized in Table 4.1.

We did not include the weak random generator algorithms type in Table 4.1 because we could not find a way to check whether this kind of algorithm is used as a basis for encrypting specific network traffic.

Weak hash algorithms	*MD-4, MD-5, RIPEMD-128*
Weak block cipher algorithms	*DES*
Weak stream cipher algorithms	*RC2, RC4*

Table 4.1: Weak cryptographic algorithms

At this stage, it is necessary to understand that our weak cryptographic algorithms will only be searched among SSL/TLS connections (security protocol the most used around the world), as we treated only this security protocol.

We found a way to completely automate this process, and will explain the detail in the implementation section of this leak source.

4.4.2 Implementation

In order to control this leak source, we used the python libraries *scapy, layer SSL of scapy, dpkt* and *socket*.

Thanks to the SSL layer of the *scapy* python library, we succeeded in retrieving all the cryptographic algorithms used for a SSL/TLS connection between two parties.

Our script thus controls that the cryptographic algoritms cited in Section *4.4.1* are not used during the reply from the destination party during the SSL/TLS handshake.

In order to control this leak source, we designed the following step to perform automatically for each network traffic tested:

- Run a script we built to indicate SSL/TLS packets using weak/insecure cryptographic algorithms. Thanks to this script, it is also possible to extract all the cryptographic algorithms used for encrypting a specific network traffic SSL/TLS connection. It additionally makes it possible to control whether unknown (homemade) cryptographic algorithms were used to encrypt a specific network traffic SSL/TLS connection.

4.5 Weak authentication module

To control the *Weak authentication leak source*, the goal was to determine whether three specific characteristics of a weak authentication system were used in any connection in the specific network traffic tested. Here are the three characteristics of a weak authentication system:

1. The duration of the connections of the system are too long for the service type provided (*Lack of adequate timeout protection*).
2. The certificate used to authenticate the system expired or was self-signed by an authority which was not a real certificate authority (*Lack of certificate validation*).
3. It was possible to perform a *Man in the Middle attack* between the user's smart device and the back-end services (*Lack of certificate validation*) of the system, meaning that it was possible to impersonate both parties.

4.5.1 Method

To establish the method for controlling the duration of the connections, we had to study in greater detail the TCP and SSL/TLS protocols. The goal was to detect the connection initialization and

finalization packets in order to extract the exact duration of each TCP and SSL/TLS connection. Once all the durations had been obtained, we simply had to control whether the durations were greater than the typical duration guidelines given in Section *3.3*.

To establish the method for controlling certificate validity, we divided the certificate validation check into three main parts: *expiration date check*, *self-signed check* and *trusted signing-authority check*.

In order to control this leak source, we designed the following step to perform automatically for each network traffic tested:

1. Extract all the SSL certificates used in the captured traffic.
2. Control their expiration date (*expiration date check*).
3. Control whether those certificates were self-signed (*self-signed check*).
4. Try to perform a *Man in the Middle attack* between the user's smart device and the back-end services (*trusted signing-authority check*).

Unfortunately, we did not find a way to automate a replay attack check. We thus did not manage to control the *Lack of adequate replay protection* check we introduced in Section 3.3. This is why we have not introduced it in this section and do not describe it in the implementation section of this leak source.

We are going to describe the implementation of the *Lack of adequate timeout protection* and *Lack of certificate validation* checks in the implementation section of this leak source.

4.5.2 Implementation

In order to control this leak source, we used the python libraries *scapy*, *layer SSL of scapy*, *dpkt* and *socket* to check the connection durations. For the certificate validation check, we used the python libraries *pyopenssl* (mainly its *crypto* module) and *certifi*; we also used the shell command of the *ssldump* library.

Concerning connection durations and thanks to the *scapy* python library and its SSL layer, we managed to retrieve them in order to control whether or not they respected typical duration guidelines.

Concerning certificate extraction, we relied on the *ssldump* library to parse the content of the certificates in hexadecimal Strings. We then isolated the hexadecimal format and converted it to plain bytes. The plain bytes could then be registered in a binary file using the *.der* extension (a standard for certificates with the *.pem* extension).

Concerning the expiration date check, we simply needed to check two fields on the certificates: the creation and expiration date. If the date at the time of capture did not fall between these two

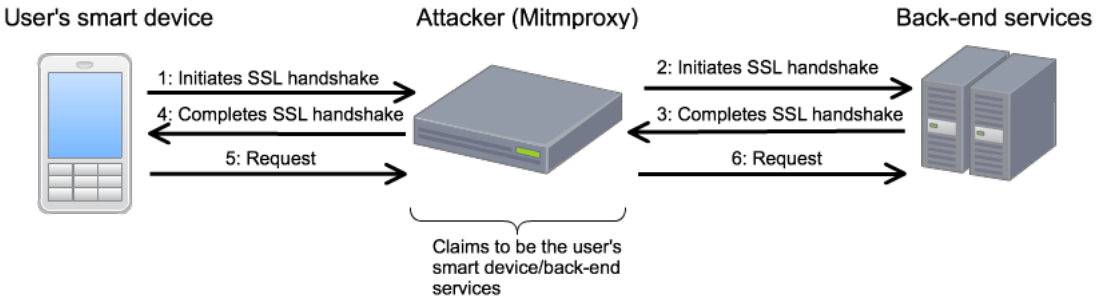


Figure 4.3: Mitmproxy: workflow diagram

dates, the certificate was considered invalid. Luckily, the *pyopenssl* library allowed us to make this check quite easily with a function *has_expired()* executed on the certificate.

To verify if a certificate was self-signed, we checked whether the signing authority was the same as the issuer of the certificate. The library *pyopenssl* was able to make this check easily. Nevertheless, we wanted to differentiate as much as possible the self-signed root certificate issued by CAs (eg: GlobalSign, Verisign, etc.) from certificates self-signed by individuals. The library *certifi* gave us access to a list of certificates trusted by Mozilla that we used as a comparison point. This method of validation allowed us to identify self-signed certificates from individual ones, but we still obtained a large number of false positives. Some of the CAs used in smart devices are not necessarily recognized by Mozilla's database. Unfortunately, these databases of certificates trusted by the smart devices are not necessarily public and, therefore, hard to obtain.

For the *trusted signing-authority check*, this consisted in verifying that the signing-authority was controlled by the user's smart device. This mainly means verifying that the user's smart device verified that the certificates presented by the back-end servers were not self-signed and could thus be trusted (not compromised). We controlled it by performing a *Man in the Middle attack*. As stated earlier, we used the software *Mitmproxy* ([37]) to perform this attack. *Mitmproxy* software generates a proxy and uses an interception self-signed certificate in order to simulate the back-end services that the user's smart device tries to contact.

In Figure 4.3, you can observe the workflow diagram of the *Man in the Middle attack* performed by *Mitmproxy*. Here are the comments regarding its workflow:

1. The user's smart device first tries to initiate an SSL handshake in order to have an encrypted connection. The attacker (*Mitmproxy*) receives the different packets concerning this SSL handshake and extracts the appropriate information to initiate its own SSL handshake with the back-end services.

2. Thanks to the information collected in the previous step, *Mitmproxy* initiates an SSL handshake with the back-end services.

3. The back-end services complete the SSL handshake believing that it has established a secured connection with the user's smart device. In receiving the SSL handshake completion packets,

Mitmproxy extracts the information needed to build an interception certificate.

4. *Mitmproxy* completes the SSL handshake with the user's smart device thanks to an interception self-signed certificate completed with the information extracted in the previous step.

5. The different SSL requests from the user's smart device are then sent to the attacker (*Mitmproxy*) believing that it is sending them to the back-end services.

6. The same SSL requests are received and modified by the attacker before being sent to the back-end services, which believe they are receiving the requests directly from the user's smart device.

When observing the *Mitmproxy* workflow, we can clearly see that the attack succeeds because the user's smart device, in step 4, does not verify whether the signing authority of the interception self-signed certificate presented by *Mitmproxy* can be trusted. Receiving a self-signed certificate, if it does not come from a genuine and trusted certificate authority, is thus a sign of a *Man in the Middle attack*.

In order to control this leak source, we elaborated the following steps to perform automatically for each network traffic tested:

- Run a script we built to ensure that user authentication was not infinitely persistent and thus to verify that the mobile application had appropriate timeout protection.
- Run a script we built in order to control the certificates expiration date and the fact that the certificates were self-signed or not.
- Run the tool *Mitmproxy* in order to know whether a *Man in the Middle attack* was possible to perform and to know whether the user's smart device accepted an interception self-signed and non-trusted certificate as authority.

4.6 Covert channels module

The aim of the *covert channels detection module* is to identify the communications between a smart device and malicious entities. As mentioned in Section 3.4, these malicious entities can perform *phishing*, inject *malwares* or *unwanted softwares* and take advantage of *backdoors*. To detect the presence of covert channels, there is more than one approach, each with their own pros and cons.

A first approach would be to analyze ourselves all the IP addresses and domains, meaning the servers behind these addresses, as well as the packets exchanged between the device and these entities in order to identify the malicious ones. We would build a database from the knowledge acquired on these entities. This method is very rigorous, but very time expensive, as well as being difficult. The complexity comes from the fact that it requires deep understanding of all the means and manifestations of malicious behaviour while analysing the servers and data exchange.

Another approach is to look for the IP addresses and domains, found in the packet exchange, in the databases of known malicious entities provided by third parties. Here, we rely on third parties to whom we delegate all the analysis work. The advantage of this approach is that the databases on which we rely were created by experts. On the other hand, these databases are not complete.

In both approaches, the question of trust must be considered. The reliability of the content of a database is as good as the trust a user can put in the expertise of the maker. The expertise of big companies (Google, Mozilla, etc.) is thus more trustworthy than ours. Therefore, we decided to use the second approach for our detection module. Nevertheless, we must still keep in mind that these databases are biased. For example, we mentioned *google analytics* as a potential source of phishing in Section 3.4, but we can be sure to never find any entry concerning malicious IP addresses or domains related to Google's libraries in databases provided by Google. We thus stress the fact that the result provided by our detection code will be biased and not completely reliable because of the third parties we rely on.

The following section will describe the steps required to apply the second approach to the *covert channels detection module*: extracting IP addresses and domains from device network traffic, querying chosen databases (Safe Browsing, Adblock and DNS [4] Blacklists) and identifying potential *backdoors*.

4.6.1 IP and domain extraction

The first step in covert channels detection is to extract all the IPs and domains from the capture traffic. The extraction is done in two stages : *DNS packet sniffing* and *reverse DNS lookup*.

Method

In *DNS packet sniffing*, we need to isolate all the DNS packets containing the answer to the DNS queries made by the smart device. Then, we run through the packets to identify the domain name queried as well as the domain name and IP addresses that were resolved by the DNS. Normally, all the IPs accessed by a device should be those resolved by DNS queries, but in the case of covert channel, we can expect some IP addresses to have been hard coded and thus not require a DNS lookup. That is where the next part comes in useful.

In the *reverse DNS lookup* stage, we run through all the packets to obtain all the source and destination IP addresses, as well as some other pieces of information like the ports and protocols used (these data will be used in other modules). For all these addresses, we perform a reverse DNS lookup (convert an IP address in a domain name) to find any known domain name for these addresses.

[4] The Domain Name System (DNS) is a hierarchical decentralized naming system for computers, services, or any resource connected to the Internet or a private network. It translates domain names to the numerical IP addresses needed for the purpose of locating and identifying computer services and devices with the underlying network protocols.[38]

Implementation

To do the *DNS packet sniffing*, we used the *dpkt* python library to parse the packets. To identify DNS packets, we look for packets using the **UDP**[5] protocol and port **53**. Then, among these DNS packets, we isolate those corresponding to DNS answers to previously made queries. In these packets, we can find the resolved IP addresses, but also the domain names queried as well as alternative domain names (for example, alternative names can correspond to the public address of the Amazon server of the application).

To perform the *reverse DNS lookup*, we rely on the *socket* python library. This library has a *gethostbyaddr(IP_address)* method that performs the reverse lookup given a valid IP address in its standard representation (e.g: 216.239.32.77). This action requires access to the Internet as the reverse lookup actually corresponds to a DNS query.

At the end of the first step, we are able to associate all the IPs with their known domains (some IP addresses do not correspond to any domain; e.g: IP address of the smart device) and all the packets with the domains they originate from or are destined for. The next step consists in checking the data we extracted against the content of database provided by third parties. In the following sections, we will describe the characteristics of the databases we chose to use : *Safe Browsing*, *Adblock* and *DNSBL*.

4.6.2 Safe Browsing

"Safe Browsing is a Google service that enables applications to check URLs against Google's constantly updated lists of suspected phishing, malware, and unwanted software pages."[40]

Method

These lists provided by Google allow us to verify the presence all the sources of leakage mentioned in section 3.4. However, the service is primarily designed to be used in browsers (Google Chrome and Mozilla Firefox [40]) to check *URLs* [6]. The first step provides only domain names and IP addresses. Nevertheless, domain names are actually partial URLs. Therefore, we first convert all the domains names into URLs accepted by the Safe Browsing API.

Safe Browsing API provides two ways of verifying the URLs against their lists. The first and easiest way is to query a web application made by Google with an http request, using *Safe Browsing Lookup API*. Unfortunately, the number of URLs that can be queried has a daily limit of 10000.

The second method consists in downloading parts of Google's lists and making local queries,

[5]The User Datagram Protocol (UDP) uses a simple connectionless transmission model with a minimum of protocol mechanism. It has no handshaking dialogues, and thus exposes the user's program to any unreliability of the underlying network protocol. There is no guarantee of delivery, ordering, or duplicate protection. UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram.[39]

[6]URL is the abbreviation of Uniform Resource Locator. URL is the global address of documents and other resources on the World Wide Web.[41]

using *Safe Browsing API v3*. Unlike the lookup api which has access to updated lists, the second method requires regular and manual updates of the local lists (which can take several hours). In our detection code, we decided to use both methods in order to mitigate the weakness of each.

Implementation

Using a python script provided by Google (*expression.py* [7]), we convert the domains in URLs in the format used in Safe Browsing API.

The python code corresponding to the *Safe Browsing Lookup API* use the python module *google-safe-browsing-lookup-python* [42] developed by a third party. As for *Safe Browsing API v3*, we also used a third party's python module *gglsbl* [43]. Both modules provide a small API, based on the Safe Browsing API, that makes it possible to query online database on one hand and local database on the other. With the help of these modules, we can detect and isolate the packets originating from or destined to known malicious URLs. We want to stress the fact that Google's lists are originally designed for web browsers, and even if a smart device can access the URLs from these lists, it is probable that many addresses specific to smart devices are not listed.

4.6.3 Adblock

The *Adblock* service was originally designed to block intrusive advertisements on web browsers and was later extended to detect tracking ads and malwares[44]. The particular design of the *Adblock* service will allow us to identify the domain names that are considered undesirable by Adblock's community.

Method

To find compromising domains, *Adblock* uses a set of rules. The rules are actually *regular expressions*[8] that are checked against domain names. For a domain, matching one of the rules means that it contains in its name a set of characters that identify it as a compromising domain (for example, the domain might contain *analytics* or *tracking* in its names). All domain names that match any of these rules are thus blocked. Basically, *Adblock* works as a pattern matching module.

Implementation

This module has a two-part implementation process: the list of rules and a parser for these rules. The rules are provided by the *EasyList* community[45]. This community maintains several lists of rules that can be used with *Adblock*. The standard list used by *Adblock* to remove intrusive

[7] Available at: `google-safe-browsing.googlecode.com/svn/trunk/python/expression.py`

[8] A regular expression is a sequence of symbols and characters expressing a string or pattern to be searched for within a longer piece of text.

advertisements is named *easylist*. Nevertheless, the list that we are interested in is named *easyprivacy*. The *easyprivacy*list[9] contains rules for blocking tracking ads, scripts and malwares.

One concern that is raised by this approach is the credibility of the list as it is open source and maintained by an unsupervised community. However, the fame and proven usefulness of the *Adblock* system seemed to us a strong enough argument to include this module in our detection system.

The rules are parsed thanks to a python library (conveniently) named *adblockparser*[46]. Once the rules have been parsed, the domains extracted from the traffic capture are checked and every match is reported.

4.6.4 DNSBL

DNS blacklists are mainly lists of IP addresses, sometimes domain names, which need to be blocked because the system behind the address is known for producing *email spam*[10]. Some *DNS blacklists* also contain the addresses of servers known for *phishing* and *malware injection*. These are the lists we will use in the *DNSBL module*.

Nevertheless, we must keep in mind one major issue while using *DNS blacklists*: the credibility of the lists. These lists are maintained by private companies with different policies regarding the choice of IPs that should be blocked ([47]; for example, while some lists only block the undesirable IP addresses, other block entire ISPs[11]). In the end, it is only a matter of trust in the companies that maintain these blacklists. Therefore, the reliability of the results depends on which blacklists contains the IP addresses found in the traffic capture.

Method

In this case, the method is very simple: we just need to send a query to *DNSBL*. All the blacklists that we will use are online and can be accessed by sending a DNS query with the IP address we need to check, to the server maintaining the blacklists.

Implementation

The implementation of the *DNSBL module* relies on the python library *pydns* that makes it possible to manipulate DNS messages and make queries. Using the IP addresses extracted from the traffic capture, we send queries to a set of DNS blacklists known to contain the addresses of server responsible for *phishing* or *malware injection*. Therefore, this module requires access to the Internet.

[9] Available at: `https://easylist.github.io/easylist/easyprivacy.txt`

[10] **Email spam**, also known as junk email or unsolicited bulk email (UBE), is a subset of electronic spam involving nearly identical messages sent to numerous recipients by email. Clicking on links in spam email may send users to phishing web sites or sites that are hosting malware. [48]

[11] An Internet service provider (ISP) is a company that provides customers with Internet access. Data may be transmitted using several technologies, including dial-up, DSL, cable modem, wireless or dedicated high-speed interconnects.[49]

Moreover, one query can take quite a long time and only check one IP address in one blacklist. In order to speed up slightly the *DNSBL module*, we used two techniques. First, we enforced a timeout of *one second* on each query, given the high number of IPs that might be extracted and the number of blacklists. Secondly, we used a form of multi-threading from the python library *multiprocessing*. A pool of processes (typically five processes) are created and all the queries are divided between them.

4.6.5 Backdoor

We discussed the matter of the presence of *backdoors* and the challenge of identifying this kind of covert channel in Section 3.4.4. Here, we will outline the method we devised to acquire information regarding the potential presence of backdoors. This information will not necessarily be enough to identify a backdoor with any certainty. The information given by this sub-module will need to be correlated with those of the previous sub-modules and even other leak source detection modules in order to find clues to any unusual behaviour in the traffic of smart devices.

Method

In this sub-module, we isolated three pieces of information: the open ports on the smart device, the services (ports) accessed by applications during the session and the proportion of each protocol in the captured traffic.

Information about the *open ports* on the smart device can be used to identify ports that always remain open, even when there is no activity. These open ports are a great threat because they give attackers an entry point into the device. The higher the number of open ports, the larger the attack surface on the smart device [17]. Therefore, before starting the capture session, we scan all the ports of the device to identify open ports. The scan is carried out before the capture in order to identify any port activity that was not initiated by the user.

The proportion of each protocol during a session mainly makes it possible to observe any abnormal usage of protocol other than TCP. For example, even if UDP is the usual alternative to TCP, a high proportion of UDP packets, in particular DNS packets, might indicate the presence of DNS tunnelling[12]. Verifying the proportion of each protocol is again a fairly simple task which only requires running through all the packets of the captured traffic and count the occurrences of each protocol (already done by the *reverse DNS lookup* sub-module).

Identifying the ports (services) accessed by the external domains may allow to observe any unusual use of a service. On every connected device, the ports are mapped to a service; ports 0 to 1024 are called *well-known ports* because the services corresponding to these ports are predefined in *RFC 1700* ([50]). Domains accessing ports other than port 80 (http) or 443 (https) may indicate an attempt

[12] DNS tunnels are commonly used to carry out covert file transfers, server traffic and web browsing. File transfer via DNS is likely to use the DNS traffic aggressively considering the DNS protocol and the encapsulation overhead for transferring data over the tunnel. [51]

to bypass conventional communication means. For example, an external domain establishing an ssh session (port 22) with the smart device will gain access to the content of the device and may be able to extract or inject data. Lets note that the example on *DNS tunnelling* given in the previous paragraph is also relevant for service usage (dns being a service) and it highlights the relation and the need for correlation between these two types of data. Obtaining information about the ports that are accessed is a fairly easy task and can be accomplished simply by looking at the information extracted along side the IP addresses and domains by the *reverse DNS lookup* sub-module.

Another piece of information that might be useful when looking for backdoors is obviously the content of the packets of communications verifying the criteria given in the paragraphs above. Nevertheless, we will not isolate the content of these packets because it is either not readable (encrypted) or already given in the *Lack of encryption* and *MitM* module if the content had any private pieces of information.

Implementation

In order to scan for open ports, we used the tool *nmap*. This tool scans an IP address and gives the ports open when the scan was performed. In the results from *nmap*, we are interested in ports that are not commonly open on smart devices (most of the ports except port 80 and 443). The results from this tool are not enough to identify ports that are always open. The smart device may be looking for or installing updates which will appear as open connections during the scan, but will close later. Nevertheless, this piece of information is not entirely useless as it allows to identify connections from activities not initiated by a user.

As we already mentioned in the previous section, most of the information concerning the use of services (ports) and the proportion of protocols was retrieved by the *reverse DNS lookup* sub-module. We only filter these data to isolate useful ones and present the data in a graphical way(histogram). The mapping between the ports and the service they host is done thanks to the python library *socket*.

4.7 System complexity

Concerning the complexity of our entire leak sources detection system, we evaluate it as linear according to the number of packets controlled by the system (n), the number of personal key words to control in the encrypted and unencrypted packets (m), the number of weak cryptographic algorithms to control (c) and the number of domains[13] to control for each packet (d). In the worst case, we have indeed a complexity of $O(n * m)$ for the *Lack of encryption module*, a complexity of $O(n * c)$ for the *Weak encryption module*, a complexity of $O(n + n * m)$ for the *Weak authentication module* and a complexity of $O(n + n * d)$ for the *Covert channels module*. As all the modules are independent, we sum their complexities and obtain a final complexity: $O(n * m + n * c + n + n * d)$.

[13] It is indeed possible to have several domains to control for a specific IP address contacted by a network packet.

Chapter 5

Experimental results

As explained earlier, we chose to verify two types of smart device: smart phones and smart TVs. For the smart phones, we chose to control two different brands: Samsung and Apple. We verified a *Samsung S4* and an *iPhone 6*. For the smart TV, we verified a Samsung Smart TV[1]. This chapter will thus be divided into several parts dedicated to smart phone and to smart TV tests.

The different leak sources we found will be summarized in a number of tables. Each table will thus be divided into four main parts: *Lack of encryption*, *Weak encryption*, *Weak authentication* and *Covert channels*.

The *Lack of encryption* part of the table will contain the key words (listed by category) and details of the files found in the unencrypted network traffic of the smart device service tested.

The *Weak encryption* part of the table will contain the list of weak cryptographic algorithms used to encrypt part of the network traffic of the smart device service tested.

The *Weak authentication* part of the table will be divided into three sub-parts: *Lack of adequate timeout protection*, *Lack of certificate validation* and *MitM* (*Man in the Middle attack*). The *Lack of adequate timeout protection* sub-part will state the timeout guideline[2], the number of connections opened, the number of connections not terminated and the maximum duration of the connections terminated and not-terminated. It is indeed important to have an idea about the number of connections not terminated as their duration can be potentially infinite. The *Lack of certificate validation* part will indicate the number of legitimate expired and self-signed certificates. The *MitM* part will contain the key words (listed by category) found in the encrypted network traffic of the smart device service tested.

The *Covert channels* part of the table will be divided in five parts, one for each of the sub-modules: *Adblock*, *DNSBL*, *SafeBrowsing*, *SafeLookup* and *Backdoors*. In each part dedicated to a specific sub-module, we will mainly give the domains that were considered suspicious. Only the part corresponding to *Backdoors* will contain more information, ports and services, depending on the findings. Also, for each device, we will give, once at the beginning of the dedicated section, the result of the *nmap tool* that will show the open ports when the device is unused.

During this chapter, the *Lack of encryption* part and the *MitM* sub-part will be supported by some excerpts from the automated report we generated for the services tested. These excerpts will be composed of network traffic packet parts or even of some of the files extracted from this network traffic. The goal is to show interesting files and packet parts that contained some of the interesting key words found in the unencrypted and encrypted network traffic of those services. Additional comments will be provided in order to explain the level of importance of these findings.

[1] Samsung 2013 SMART LED-TV 46”

[2] We tested the smart device service each time 10 minutes more than the timeout guideline we chose for this service.

5.1 Smart phones

For each smart phone, we decided to audit several mobile applications in several domains: social media, finance, transport and housing and a Belgian mobile application. We chose these domains because they represent the main services that mobile applications provide for users today. We chose to dedicate a domain to Belgian mobile applications for two main reasons: our dissertation was written in Belgium and we needed to audit mobile applications with a smaller number of users in order to observe the difference in leak sources between well-known and less-known mobile applications. Table 5.1 summarizes the different mobile applications we chose to audit and their respective fields.

Fields	Mobile applications
Social media	*Facebook*, *Snapchat* and *MeetMe*
Finance	*Paypal*
Transport and Housing	*Airbnb* and *Uber*
Belgian mobile application	*Airbsit*

Table 5.1: Mobile applications experimented

Concerning the *Social media* field, we chose these mobile applications because they were, with more than one billion Play Store downloads for *Facebook*, and more than a hundred million Play Store downloads for *Snapchat* and more than 10 million Play Store downloads for *MeetMe*, probably the best-known mobile applications in this field. Moreover, *MeetMe* was already tested and proved to leak some information two years ago by the University of New Haven's Cyber Forensics Research and Education Group ([2]).

For the *Finance* field, we chose to audit the mobile application *Paypal* because it was, for us, the best-known, the most international and also probably the most used mobile application in this field. It effectively has more than ten million Play Store downloads.

For the *Transport and Housing* field, we chose *Airbnb* and *Uber* because they were for us, with more than ten million Play Store downloads, the best-known, the most international and also probably the most used mobile applications in their field. Moreover, concerning the transport aspect, we chose *Uber* because, this company has become really popular in a short period of time, so its mobile application could be prone to security breaches and sources of information leaks.

Finally, in the *Belgian mobile application* field, we chose to audit the mobile application *Airbsit* as it did not have an enormous number of Play Store downloads (10,000 - 50,000). This mobile application helps users find babysitters or find a job as a babysitter.

Apple	Samsung
• 62078 (iphone-sync)	• all ports closed

Table 5.2: Open ports on Apple and Samsung device

The table 5.2 reports the list of ports that were open while the devices where unused. We can

see that the Apple device was running some synchronization service.

5.1.1 

We tested the *Facebook* mobile application for Apple and Samsung smart devices. Table 5.3 summarizes the leak sources we found for this mobile application.

Leak sources		Apple (*iPhone 6*)	Samsung (*Samsung S4*)
Lack of encryption		Nothing found	Nothing found
Weak encryption		• 56 % of IP traffic SSL encrypted • No weak cryptographic algorithms used	• 73 % of IP traffic SSL encrypted • No weak cryptographic algorithms used
Weak authentication	Lack of adequate timeout protection	• **Timeout guideline**: 30 minutes • 160 TCP connections opened • Maximum TCP terminated connection duration: 1 minute • 1 TCP connection not terminated • Maximum TCP non-terminated connection duration: 25 minutes	• **Timeout guideline**: 30 minutes • 252 TCP connections opened • Maximum TCP terminated connection duration: 1 minute • 2 TCP connections not terminated • Maximum TCP non-terminated connection duration: 28 minutes
	Lack of certificate validation	• no expired certificates • no self-signed certificates	• no expired certificates • no self-signed certificates
	MitM	Not succeeded	Not succeeded
Covert channels	Adblock	• cx.atdmt.com	• csi.gstatic.com
	DNS blacklists	• 191.254-4-62.akamai.com • a3.da1.akamai.net • a3.mzstatic.com • a3.mzstatic.com.edgesuite.net • a2.mzstatic.itunes-apple.com.akadns.net	• 208.253-4-62.akamai.com • googleapis.l.google.com • pagead.l.doubleclick.net • www.googleadservices.com • csi.gstatic.com • play.googleapis.com • settings.crashlytics.com
	SafeBrowsing	No Match	No Match
	SafeLookup	No Match	No Match
	Backdoors	• port 33000 (service unknown) accessed by edge-star-shv-01-amt2.facebook.com	Nothing noteworthy

Table 5.3: Sources of leaks in the *Facebook* mobile application

Among the different timeout guidelines given in the section 3.3.1, we chose 30 minutes for the mobile application *Facebook*. We chose this value because we considered *Facebook* as a medium security application. In other words, *Facebook* mobile services transmit some sensitive private information, not related to finance (payment).

As stated in the table 5.3, the *Facebook* mobile application is one of the rare mobile application tested to be resistant against the *Man in the Middle Attack*. Indeed, the figure 5.1 shows the error message we obtained from the *Facebook* mobile application when performing the *Man in the Middle*

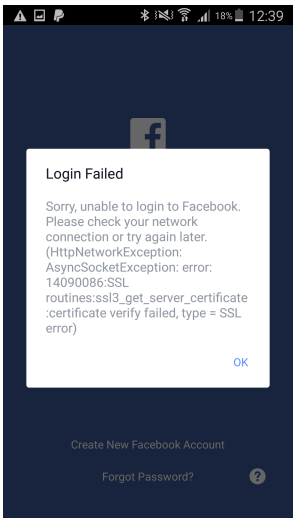


Figure 5.1: Samsung MitM on Facebook

attack against the Samsung device.

The *covert channels* module found analytics and tracking domains. For example, the domain *c.atdmt.com* is known to be used by *Facebook* for cookie tracking. Besides the analytics services of Google, we also found the domain *pagead.l.doubleclick.net*. The later hostname is a variation of *doubleclick.net*, a subsidiary of Google proposing advertisement services. The domains with *doubleclick.net* are known for proposing intrusive advertisements and even adwares that are used for cookie tracking. The module also found a connection to the port 33000 by a domain of *Facebook*, but we weren't able to find the purpose of this connection nor the service used.

Facebook evaluation

The *Facebook* mobile application passed the *Lack of encryption*, *Weak encryption* and *Weak authentication* tests. We indeed didn't find any of our personal key words in the unencrypted network traffic of this service. We didn't find any weak cryptographic algorithms used by this service. Moreover, we can strongly affirm that the *Facebook* authentication service is strong as it passed all the tests of this leak source.

Moreover, as you can see in Figures 5.2 and 5.3, the network traffic of the application *Facebook* is composed almost entirely of SSL encrypted packets. It is an example of good practise that other applications should follow.

It comes at no surprise that the *covert channels* module found analytics and tracking domains since there is a known and ongoing controversy around *Facebook* for its intrusive behaviour ([52]).

Ultimately, the mobile application *Facebook* succeeded almost all of our tests.

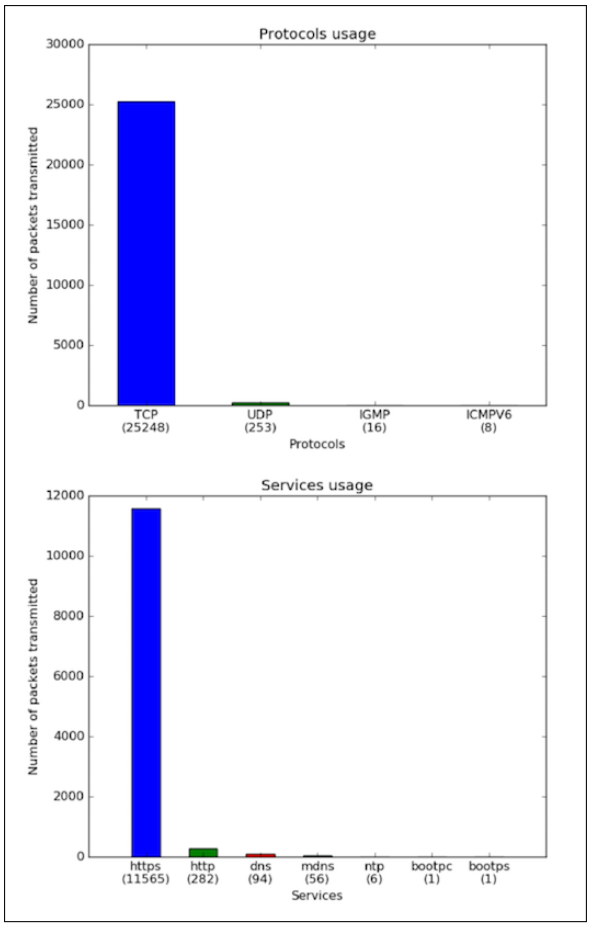


Figure 5.2: Apple protocol and service usage

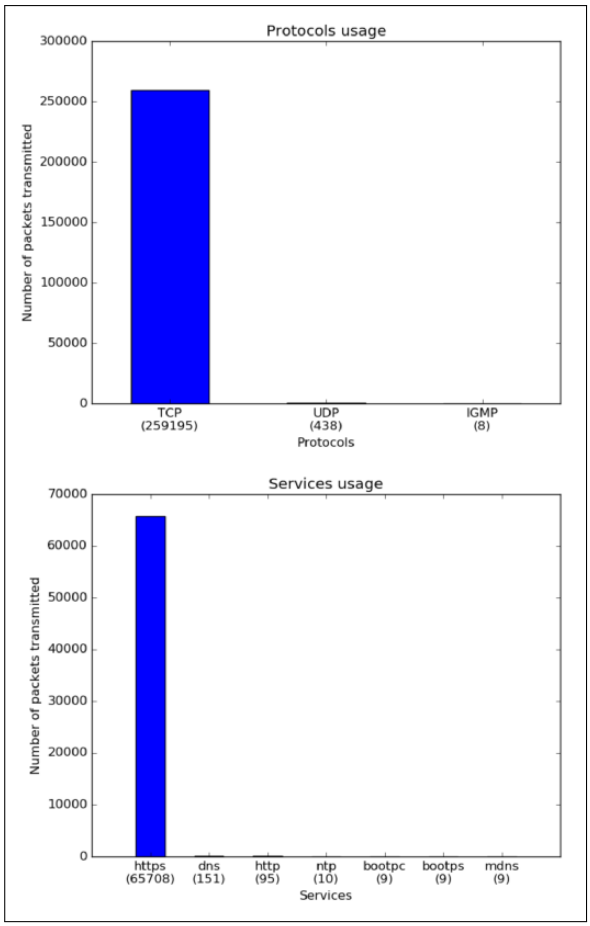


Figure 5.3: Samsung protocol and service usage

5.1.2

We tested the *Snapchat* mobile application for Apple and Samsung smart devices. Table 5.4 summarizes the sources of leaks we found for this mobile application.

Among the different timeout guidelines given in the section 3.3.1, we chose 30 minutes for the mobile application *Snapchat*. We chose this value because we considered *Snapchat* as a medium security application. In other words, *Snapchat* mobile services transmit some sensitive private information, not related to finance (payment).

As stated in the table 5.4, the *Snapchat* mobile application is one of the rare mobile application tested to be resistant against the *Man in the Middle Attack*. Indeed, Figure 5.4 shows the error message we obtained from the *Snapchat* mobile application when performing the *Man in the Middle attack* against the Apple device.

The entry for DNS blacklists for the Apple devices shows some hostnames of IP addresses that were blocked. After examination, we concluded that the IP addresses are probably not harmful. Actually, we found that these IP addresses were blocked because they are dynamically allocated. In some cases, dynamic allocation is a hiding scheme used by malicious entities and some DNSBLs preemp-

Leak sources		Apple (*iPhone 6*)	Samsung (*Samsung S4*)
Lack of encryption		Nothing found	Nothing found
Weak encryption		• 66 % of IP traffic SSL encrypted • No weak cryptographic algorithms used	• 40 % of IP traffic SSL encrypted • No weak cryptographic algorithms used
Weak authentication	Lack of adequate timeout protection	• **Timeout guideline**: 30 minutes • 20 TCP connections opened • Maximum TCP terminated connection duration: 1 minute • 11 TCP connections not terminated • Maximum TCP non-terminated connection duration: 4 minutes	• **Timeout guideline**: 30 minutes • 84 TCP connections opened • Maximum TCP terminated connection duration: 11 minutes • 14 TCP connections not terminated • Maximum TCP non-terminated connection duration: 9 minutes
	Lack of certificate validation	• no expired certificates • no self-signed certificates	• no expired certificates • no self-signed certificates
	MitM	Unsuccessful	Unsuccessful
Covert channels	Adblock	No Match	No Match
	DNS blacklists	• chat-gateway250-prod.chat.snapchat.com • storage.googleapis.com • storage.l.googleusercontent.com • ams15s21-in-f144.1e100.net • geofilter.storage.googleapis.com • ams15s22-in-f176.1e100.net	No Match
	SafeBrowsing	No Match	No Match
	SafeLookup	No Match	No Match
	Backdoors	Nothing noteworthy	Nothing noteworthy

Table 5.4: Sources of leaks in the *Snapchat* mobile application

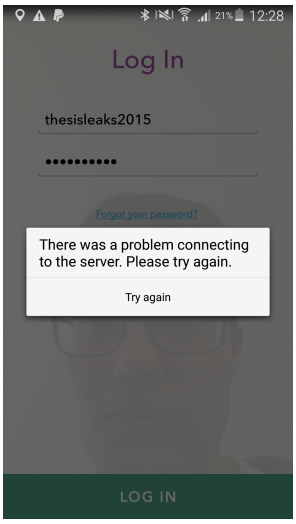


Figure 5.4: Apple MitM on Snapchat

tively block such IP addresses.

Snapchat evaluation

Despite its catastrophic past history of security leaks ([53], [54]), the *Snapchat* company has adopted an extremely firm stance on security this year. We think that this could be the reason why the *Snapchat* mobile application passed the *Lack of encryption*, *Weak encryption* and *MitM* tests. We effectively did not find any of our personal key words unencrypted in the Snapchat network traffic. We did not find any weak cryptographic algorithms used by this service either. We ultimately did not manage to perform a *Man in the Middle attack* against *Snapchat services* and they don't show any sign of actual *covert channels*.

With regards to the *Lack of timeout protection*, no connection terminated exceeded the timeout duration guideline (30 minutes). However, we can see, especially for the Apple smart device, that there were a large number of non-terminated connections (11) compared to the total number of connections opened (20). This could indicate that some connections may be infinitely persistent and thus give attackers a long time to impersonate users...

Ultimately, except for the *Lack of timeout protection* test, the *Snapchat* mobile application passed our leak source tests relatively well.

5.1.3 

We tested the *MeetMe* mobile application for Apple and Samsung smart devices. Table 5.5 summarizes the sources of leaks we found for this mobile application.

Among the different timeout guidelines given in the section 3.3.1, we chose 30 minutes for the mobile application *MeetMe*. We chose this value because we considered *MeetMe* as a medium security application. In other words, *MeetMe* mobile services transmit some sensitive private information, not related to finance (payment).

MeetMe does not encrypt the conversation that happens between its users. Figure 5.5 is an excerpt of an unencrypted network packet containing the message we received from another user: *Thesis message 1*. Figure 5.6 is an excerpt of an unencrypted network packet containing the message we sent to another user: *Thesis message 2*. These packets are definitely linked to the *MeetMe* service as they were issued by the domains *stream.meetme.com* and *messages.meetme.com*.

When *MeetMe* asks to verify a user authentication, it leaks its phone number (without any encryption). Figure 5.7 is an excerpt of an unencrypted network packet that illustrates it. This packet is definitely linked to the *MeetMe* service as it is issued by the domain *fr.ssl.meetme.com*.

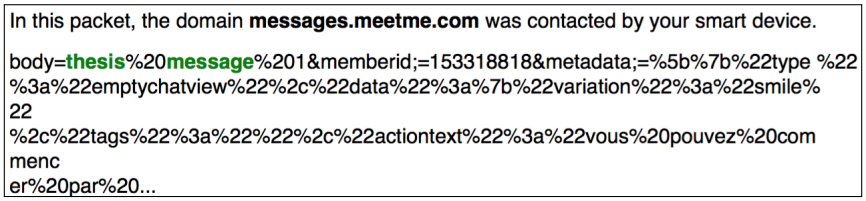
MeetMe tracks its users' location, gender, language, device model and device id. It transmits it without any encryption. The figure 5.8 is an excerpt of an unencrypted network packet that illustrates it. If we enter the longitude and latitude in a String location converter, we obtain: *Rue Charlemagne, 1348 Ottignies-Louvain-la-Neuve, Belgium*, our exact localization when we performed the *MeetMe* tests! Those information leaks are even more surprising because if we have a look at the domain con-

Leak sources		Apple (*iPhone 6*)	Samsung (*Samsung S4*)
Lack of encryption		• **Text conversation**: thesis message (Fig. 5.5 and 5.6) • **Phone number**: 477522662 (Fig. 5.7) • **iPhone name**: iPhone de mattieu • Some jpg files representing part of pictures transmitted to other users could be extracted	• **Samsung model number**: gt-i9505 (5.8) • **Latitude**: lat, latitude, 50.6697268 (Fig. 5.8) • **Longitude**: lng, longitude, 4.613926 (Fig. 5.8) • **Gender**: gender, 0 (Fig. 5.8) • **Language**: en (Fig. 5.8) • **Conversation**: thesis message • **Tracking**: device_id (Fig. 5.8) • Some jpg files representing part of pictures transmitted to other users could be extracted
Weak encryption		• 29 % of IP traffic SSL encrypted • No weak cryptographic algorithms used	• 27 % of IP traffic SSL encrypted • No weak cryptographic algorithms used
Weak authentication	Lack of adequate timeout protection	• **Timeout guideline**: 30 minutes • 295 TCP connections opened • Maximum TCP terminated connection duration: 1 minute • 4 TCP connections not terminated • Maximum TCP non-terminated connection duration: 32 minutes (> 30 minutes!) • The non-terminated connection initiated by the domain **noticeprd1.cloudapp.net** lasted 32 minutes	• **Timeout guideline**: 30 minutes • 461 TCP connections opened • Maximum TCP terminated connection duration: 1 minute • 16 TCP connections not terminated • Maximum TCP non-terminated connection duration: 37 minutes (> 30 minutes!) • The non-terminated connection initiated by the domain **map-pb.quantserve.com.akadns.net** lasted 37 minutes • The non-terminated connection initiated by the domain **pixel.adsafeprotected.com** lasted 31 minutes
	Lack of certificate validation	• no expired certificates • no self-signed certificates	• no expired certificates • no self-signed certificates
	MitM	• **Name**: thesisf, thesisl (Fig. 5.9) • **Gender**: female (Fig. 5.9) • **Birth date year**: 1993 (Fig. 5.9) • **Phone number**: 477522662 (Fig. 5.9) • **Credential**: email, thesisleaks2016@gmail.com, password, thesis2015 (Fig. 5.9)	Unsuccessful
Covert channels	Adblock	• analytics.localytics.com	• analytics.localytics.com • beap-bc.yahoo.com • bid.g.doubleclick.net
	DNS blacklists	No Match	No Match
	SafeBrowsing	No Match	No Match
	SafeLookup	No Match	No Match
	Backdoors	• port 1883 (mqtt) accessed by stream.meetme.com • port 8080 (http-alt) accessed by h-sdk.online-metrix.net	• port 1883 (mqtt) accessed by stream.meetme.com

Table 5.5: Sources of leaks in the *MeetMe* mobile application

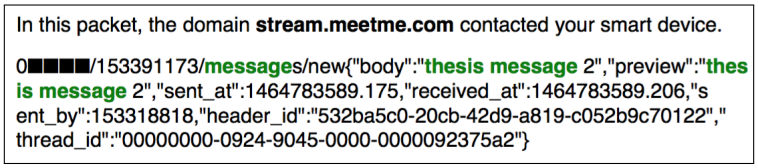
tacted (*ib.adnxs.com*), it appears that it is the eighth-biggest name in tracking analysis ([55]). This thus means that *MeetMe* is selling and sending very personal information of its users to some advertisers.

All the profile information we entered as a user of the *MeetMe* mobile application are contained in the encrypted packet excerpt of the figure 5.9. This packet was obtained thanks to *Apple MitM*. It is definitely linked to the *MeetMe* service as it contacted the domain **ssl.meetme.com**.


In this packet, the domain **messages.meetme.com** was contacted by your smart device.

body=thesis%20message%201&memberid;=153318818&metadata;=%5b%7b%22type %22
%3a%22emptychatview%22%2c%22data%22%3a%7b%22variation%22%3a%22smile%
22
%2c%22tags%22%3a%22%22%2c%22actiontext%22%3a%22vous%20pouvez%20com
menc
er%20par%20...

Figure 5.5: Apple lack of encryption: report excerpt


In this packet, the domain **stream.meetme.com** contacted your smart device.

0■■■■/153391173/messages/new{"body":"thesis message 2","preview":"thes
is message 2","sent_at":1464783589.175,"received_at":1464783589.206,"s
ent_by":153318818,"header_id":"532ba5c0-20cb-42d9-a819-c052b9c70122","
thread_id":"00000000-0924-9045-0000-0000092375a2"}

Figure 5.6: Apple lack of encryption: report excerpt

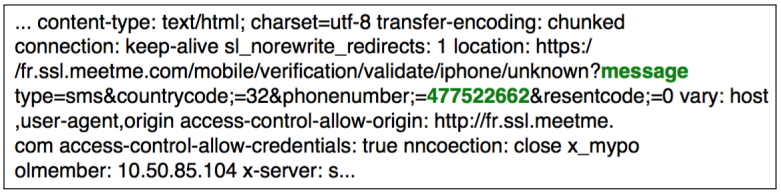
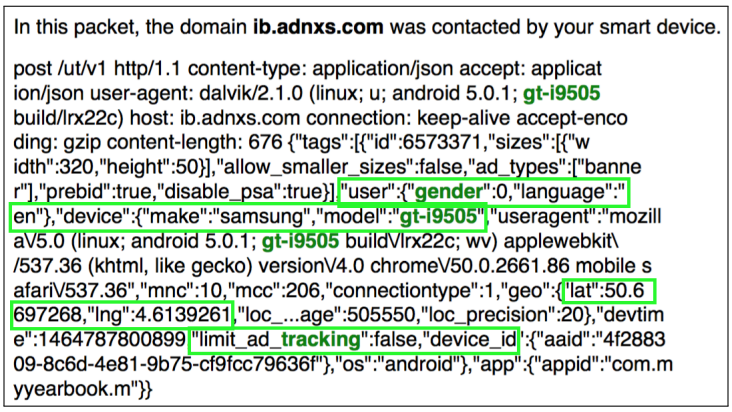

... content-type: text/html; charset=utf-8 transfer-encoding: chunked
connection: keep-alive sl_norewrite_redirects: 1 location: https:/
/fr.ssl.meetme.com/mobile/verification/validate/iphone/unknown?message
type=sms&countrycode;=32&phonenumber;=477522662&resentcode;=0 vary: host
,user-agent,origin access-control-allow-origin: http://fr.ssl.meetme.
com access-control-allow-credentials: true nncoection: close x_mypo
olmember: 10.50.85.104 x-server: s...

Figure 5.7: Apple lack of encryption: report excerpt


In this packet, the domain **ib.adnxs.com** was contacted by your smart device.

post /ut/v1 http/1.1 content-type: application/json accept: applicat
ion/json user-agent: dalvik/2.1.0 (linux; u; android 5.0.1; gt-i9505
build/lrx22c) host: ib.adnxs.com connection: keep-alive accept-enco
ding: gzip content-length: 676 {"tags":[{"id":6573371,"sizes":[{"w
idth":320,"height":50}],"allow_smaller_sizes":false,"ad_types":["banne
r"],"prebid":true,"disable_psa":true}],"user":{"gender":0,"language":"
en"},"device":{"make":"samsung","model":"gt-i9505","useragent":"mozill
a\/5.0 (linux; android 5.0.1; gt-i9505 build\/lrx22c; wv) applewebkit\
/537.36 (khtml, like gecko) version\/4.0 chrome\/50.0.2661.86 mobile s
afari\/537.36","mnc":10,"mcc":206,"connectiontype":1,"geo":{"lat":50.6
697268,"lng":4.6139261,"loc_...age":505550,"loc_precision":20},"devtim
e":1464787800899,"limit_ad_tracking":false,"device_id":{"aaid":"4f2883
09-8c6d-4e81-9b75-cf9fcc79636f"},"os":"android"},"app":{"appid":"com.m
yyearbook.m"}}

Figure 5.8: Samsung lack of encryption: report excerpt

Concerning *covert channels*, we have an analytics service called *analytics.localytics.com* and another tracking service *bid.g.doubleclick.net*. The first one was certainly used to locate *Meetme* users.

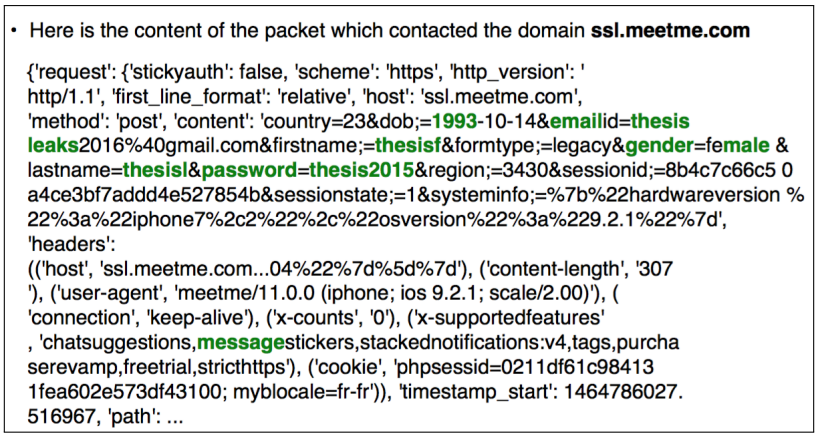


Figure 5.9: Apple MitM: report excerpt

Concerning the second one, domains with *doubleclick.net* are known for proposing intrusive advertisements and even adwares that are used for cookie tracking. The module also found a connection to the port 8080 (unencrypted http service) by the domain *h-sdk.online-metrix.net*. From the few information we could gather, it seems its another analytics service. The domain `stream.meetme.com` is also accessing port 1883 who runs a service called *mqtt* (MQ Telemetry Transport), which is a message broker[3]; we thus suppose that this service was used by *MeetMe* for its own messaging service.

MeetMe evaluation

The *MeetMe* mobile application passed the *Weak encryption* test. We indeed didn't detect any weak cryptographic algorithms used by the *MeetMe services* to encrypt part of its traffic.

With regards to the *Lack of encryption* test, it has been shown and proved that *MeetMe* was leaking very personal information about its users such as their conversation, location, phone number, gender, language, profile pictures, iPhone name, Samsung model number and device id. This is a serious case of information leakage! It is even more problematic as *MeetMe* was proved and warned to leak its user conversations two years ago (as stated in the introduction).

Moreover, in Figure 5.11, we can clearly see that *http* usage is on par with *https* usage in the Samsung device. In figure 5.10, the amount of *http* traffic is not negligible since it represents more than a half of the total traffic accessing some Apple services. There is clearly a weakness in term of encryption.

Concerning the *Lack of timeout protection* test, some non-terminated connections lasted more than the recommended duration guideline (30 minutes). Indeed, the non-terminated connections initiated by the domains `noticeprd1.cloudapp.net`, `map-pb.quantserve.com.akadns.net` and

[3]Message broker is an intermediary program module that translates a message from the formal messaging protocol of the sender to the formal messaging protocol of the receiver. Message brokers are elements in telecommunication networks where software applications communicate by exchanging formally-defined messages. Message brokers are a building block of Message oriented middleware.

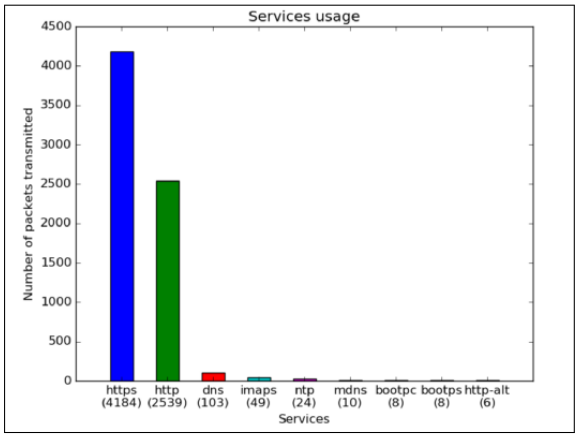


Figure 5.10: Apple service usage

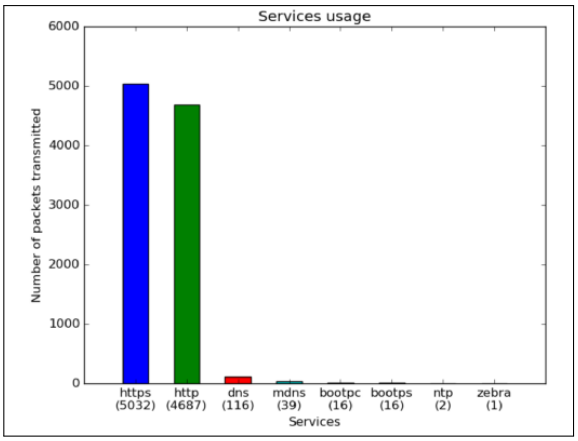


Figure 5.11: Samsung service usage

`pixel.adsafeprotected.com` lasted more than 30 minutes. We can not argue about these domains because they are not directly linked to *MeetMe* user navigation.

Concerning the *MitM* test, Figure 5.9 shows that the *Man in the Middle attack* succeeded, at least for the Apple device!

The *covert channels* module mainly shows the presence of analytics services, which indicate a risk of phishing, especially in a so poorly secured application.

In the end, the *MeetMe* mobile application was one of the blackbird of our leak source tests. It indeed only succeeded to pass the *weak encryption* leak test. However, it didn't use so much encryption for encrypting its network traffic... Moreover, except for the *Lack of timeout protection* test, the other leak sources detected are really serious!

5.1.4 

We tested the *Paypal* mobile application for Apple and Samsung smart devices. Table 5.6 summarizes the sources of leaks we found for this mobile application.

Leak sources		Apple (*iPhone 6*)	Samsung (*Samsung S4*)
Lack of encryption		Nothing found	Nothing found
Weak encryption		• 56 % of IP traffic SSL encrypted • **MD5** weak hash algorithm was used by the domain **altfarm.mediaplex.com** • **RC4** weak stream cipher algorithm was used by the domains **b.stats.paypal.com** and **altfarm.mediaplex.com**	• 49 % of IP traffic SSL encrypted • No weak cryptographic algorithms used
Weak authentication	Lack of adequate timeout protection	• **Timeout guideline**: 15 minutes • 67 TCP connections opened • Maximum TCP terminated connection duration: 1 minute • 1 TCP connection not terminated • Maximum TCP non-terminated connection duration: 1 minute	• **Timeout guideline**: 15 minutes • 82 TCP connections opened • Maximum TCP terminated connection duration: 4 minutes • 4 TCP connection not terminated • Maximum TCP non-terminated connection duration: 4 minutes
	Lack of certificate validation	• no expired certificates • no self-signed certificates	• no expired certificates • no self-signed certificates
	MitM	• **Name**: thesisfirstname, thesislastname, mattieu, detaille • **Birth date year**: 1993 • **Home address**: taille au vivier, namur • **Phone number**: 477522662 • **Credit card number**: 4891090129709262 • **Credential**: mattieu.detaille@gmail.com, thesisleaks2016apple@gmail.com, password, thesis2015	• **Samsung model number**: gt-i9505 (Fig. 5.12) • **Name**: thesisfirstname, thesislastname, mattieu, detaille (Fig. 5.13, 5.15 and 5.16) • **Birth date year**: 1993 (Fig. 5.13) • **Home address**: taille au vivier, namur (Fig. 5.13) • **Phone number**: 477522662 (Fig. 5.13) • **Credit card number**: 4891090129709262 (Fig. 5.14) • **Credential**: mattieu.detaille@gmail.com, thesisleaks2015@gmail.com, password, thesis2015 (Fig. 5.13, 5.15 and 5.16)
Covert channels	Adblock	• t.paypal.com	No Match
	DNS blacklists	No Match	No Match
	SafeBrowsing	No Match	No Match
	SafeLookup	No Match	No Match
	Backdoors	Nothing noteworthy	Nothing noteworthy

Table 5.6: Leak sources of *Paypal* mobile application

Among the different timeout guidelines given in the section 3.3.1, we chose 15 minutes for the mobile application *Paypal*. We chose this value because we considered *Paypal* as a high security application. In other words, *Paypal* mobile services transmit some sensitive private information related to finance (payment).

Figures 5.12, 5.13, 5.14, 5.15 and 5.16 are excerpts from the encrypted packets obtained thanks to the *Samsung MitM*. They all contacted the domain **api-m.paypal.com**.

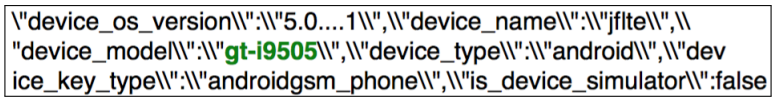


Figure 5.12: Samsung MitM: report excerpt

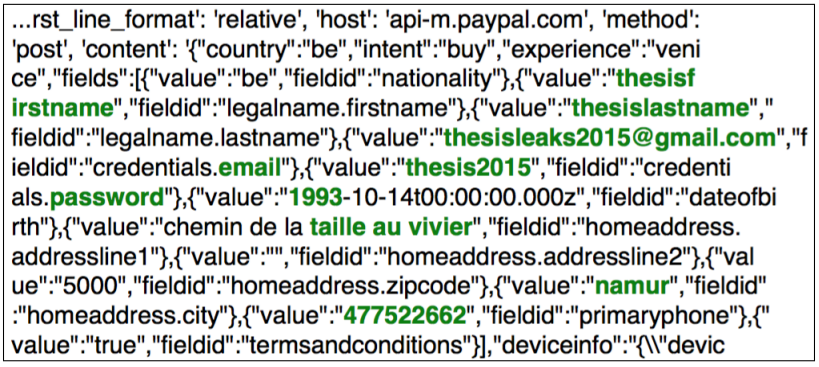


Figure 5.13: Samsung MitM: report excerpt

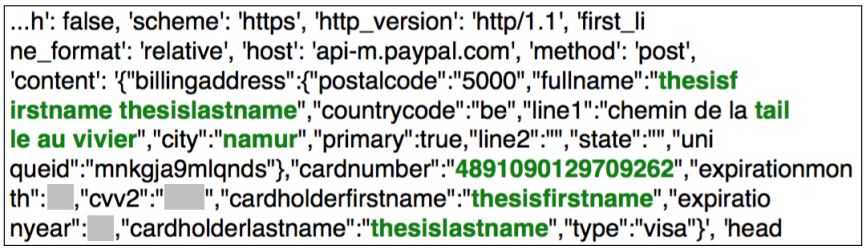


Figure 5.14: Samsung MitM: report excerpt

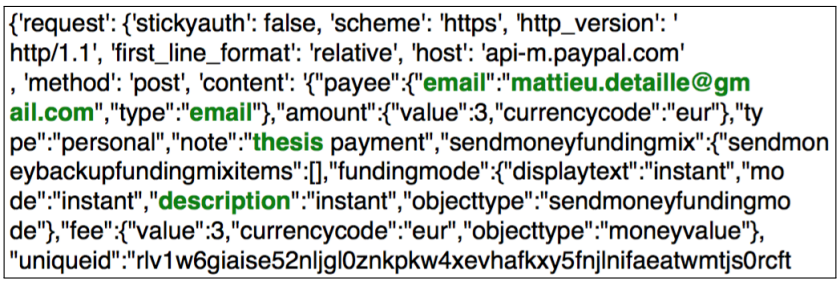


Figure 5.15: Samsung MitM: report excerpt

Figure 5.16: Samsung MitM: report excerpt

Paypal services are tracking information related to the smart devices that contact them. Figure 5.12 proves it. It is indeed possible to see that the model, type, device key type, etc. are sent to *Paypal* back-end services.

The packet excerpt associated with Figure 5.13 contains almost all the profile information we entered as a user of the *Paypal* mobile application. It effectively contains the first name, last name, login, password, birth date, home address, etc. of the user who created the account.

All the credit card information we entered as a user of the *Paypal* mobile application is contained in the packet excerpt associated with Figure 5.14, even the three-digit secure code and the expiration date of the card[4].

Figures 5.15 and 5.16 contain respectively the information from a *Paypal* transaction and a payment request (both for three cents) intended for the account *mattieu.detaille@gmail.com*.

Concerning *Paypal* transactions, we succeeded, thanks to the *Mitmproxy tool*, during a *MitM* on the *Apple device*, to modify a live *Paypal transaction*. We indeed managed to change the amount of the transaction from 1 cent to 10 cents. In Figure 5.17, we can observe the information we altered thanks to the *Mitmproxy* tool. We can also see the different information we could possibly alter such as the identifier of the user who will receive the money, the fees we pay, etc. In the response of this attack, in Figure 5.18, it is possible to observe that *Paypal* is considering that we are paying one cent (upper part on the figure), but lower, we can see: *You are going to pay 20 cents*; meaning the attack succeeded! The 20 cents come from the fact that the Paypal transaction is actually including a 10 cents fee.


```
{"amount":{"currencyCode":"EUR","value":10},"payee":{"firstName":"Mattieu","email":"mattieu.deta
ille@gmail.com","lastName":"Detaille","type":"Email"},"type":"Personal","sendMoneyFundingMix":{"
fundingMode":{"displayText":"Instant","description":"Instant","objectType":"SendMoneyFundingMode
","mode":"Instant"},"schemaVersion":"1.0.0","sendMoneyBackupFundingMixItems":[],"sendMoneyFundin
gMixItems":[{"amount":{"currencyCode":"EUR","value":2,"objectType":"MoneyValue"},"objectType":"S
endMoneyFundingMixItem","fundingSource":{"schemaVersion":"1.0.0","usable":false,"userOfflinePref
erable":false,"expired":false,"largeImage":{"front":{"url":"https:\/\/www.paypalobjects.com\/web
static\/wallet\/bankscards\/img-fi-mastercard-large.png","objectType":"Image"},"objectType":"Two
SidedImage","back":{"url":"https:\/\/www.paypalobjects.com\/webstatic\/wallet\/bankscards\/img-f
i-cardgeneric-large.png","objectType":"Image"}},"objectType":"SendMoneyCredebitCard","uniqueId":
"CC7QB499UH5GDYJ","userOfflinePreferred":false,"cardNumberPartial":"9606","cardType":{"shortName
":"MasterCard","name":"MasterCard","schemaVersion":"1.0.0","uniqueId":"MASTER_CARD","objectType"
:"CredebitCardType","type":"MASTER_CARD"},"cardImages":{"objectType":"CardImages","frontUrl":"ht
tps:\/\/www.paypalobjects.com\/webstatic\/wallet\/bankscards\/img-fi-mastercard-large.png"},"sma
llImage":{"front":{"url":"https:\/\/www.paypalobjects.com\/webstatic\/wallet\/bankscards\/img-fi
-mastercard-small.png","objectType":"Image"},"objectType":"TwoSidedImage","back":{"url":"https:\
/\/www.paypalobjects.com\/webstatic\/wallet\/bankscards\/img-fi-cardgeneric-small.png","objectTy
pe":"Image"}}}}],"objectType":"SendMoneyFundingMix","currencyConversionAvailable":false,"totalAm
ount":{"currencyCode":"EUR","value":2,"objectType":"MoneyValue"},"fee":{"currencyCode":"EUR","va
lue":1,"objectType":"MoneyValue"},"uniqueId":"tj5YAyU3B5YOaUKFTCWMbA_PcjacIZzS_u332_Ude7VLNobfNb
C2BnkeU-y2iMEeLyJR2dh1x366M4QuvD7dnlUS4ry","totalRecipientAmount":{"currencyCode":"EUR","value":
1,"objectType":"MoneyValue"}}}
```


Figure 5.17: Apple MitM: Paypal transaction parameters

As the figures in the *Apple MitM* contain the same scenarios as Figures 5.13, 5.14, 5.15 and 5.16, these data are not shown.

The *Adblock* module blocked the hostname **t.paypal.com**. Unfortunately, we couldn't find any information about this host. Nevertheless, we have verified the existence of the server corresponding to this hostname, but we don't know anything of its usage since it leads to an empty web site.

[4]Obviously, this information is hidden, as it is clearly personal.

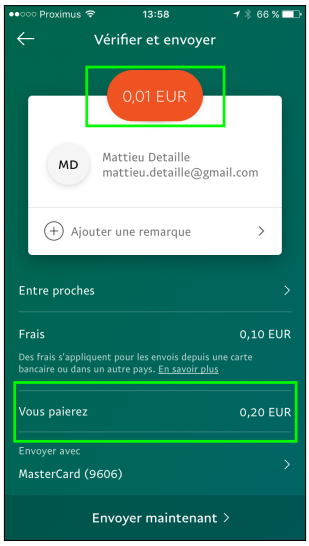


Figure 5.18: Apple MitM: strange Paypal behavior

Paypal evaluation

The *Paypal* mobile application passed the *Lack of encryption* and *Lack of timeout protection* tests. We did not find any of our personal key words in the unencrypted network traffic of this service. In addition, most of the connections opened by this service were closed before the timeout guideline (15 minutes).

With regards to the *Weak encryption* leak source, we found it originally when testing a specific payment part of the mobile application *Uber*. Part of the network traffic between our smart device and the back-end servers of Paypal was indeed encrypted using the RC4 Stream Cipher encryption algorithm. This encryption algorithm is considered weak by the cryptographical community ([56]). Afterwards, when testing the mobile application *Paypal*, we noticed that this weak Stream Cipher encryption algorithm was again used to encrypt part of its network traffic. By digging a little deeper, we found the domain initializing the connections using this weak algorithm: b.stats.paypal.com. We can thus suppose that the data sent via these insecure connections were statistics about users' payments, user profiles, etc. obtained from the *Paypal* mobile application. As this weakness was very surprising, we did some research and found a forum on the *Paypal website*[5] where people were discussing this weakness, meaning it has already been discovered and *Paypal* knew certainly about its existence.

Regarding the *MitM* test, Figures 5.13, 5.14, 5.15 and 5.16 show that the *Man in the Middle attack* succeeded! This is clearly a bad point for *Paypal*, knowing that we did not manage to perform such an attack against the *Facebook* and *Snapchat* mobile applications. In addition, it was even possible to modify a *Paypal* transaction in live (see figure 5.17).

Besides, the single domain **t.paypal.com** blocked on the Apple device, the application Paypal didn't show any other sign indicating the presence of *covert channels*.

[5] *https://www.paypal-community.com/t5/About-Protections/Why-is-PayPal-still-preferring-a-RC4-cipher-with-TLS-1-2-Is-RC4/td-p/959252*

In the end, the *Paypal* mobile application was surprisingly one of the blackbirds in our leak source tests. First, because it uses weak encryption algorithms and second, because of its vulnerability to the *Man in the Middle attack*.

5.1.5 

We tested the *Airbnb* mobile application for Apple and Samsung smart devices. Table 5.7 summarizes the sources of leaks we found for this mobile application.

Among the different timeout guidelines given in the section 3.3.1, we chose 30 minutes for the mobile application *Airbnb*. We chose this value because we considered *Airbnb* as a medium security application. In other words, *Airbnb* mobile services transmit some sensitive private information, not related to finance (payment). All the payment information have indeed to be completed on the *Airbnb* website.

The name of the iPhone tested (*iPhone de Mattieu*) was found unencrypted in the network packet given Figure 5.19. We did not find information about the domains that requested this information. However, this is still an information leak as the name of an iPhone is often personal. For example, in this case, the iPhone name contains the name of its owner: *Mattieu*.

Some addresses were found unencrypted in the network packets in Figures 5.20 and 5.21. These addresses were transmitted unencrypted during the *Airbnb* tests. These addresses are close to (Walibi) or even in (golf course of Louvain-la-Neuve, esplanade of Louvain-la-Neuve) Louvain-la-Neuve. The most surprising part of this finding was that we were in Louvain-la-Neuve when we tested the *Airbnb* mobile application. However, even after some deeper research, we did not manage to link the domain that contacted our smart device (**a442.w45.akamai.net**) to the *Airbnb* services. We nevertheless strongly believe that these packets are linked to the *Airbnb* services because of two mains reasons. First, they were transmitted during the *Airbnb* tests. Second, *Airbnb* is a mobile application that deals with home addresses as its goal is to provide people with accommodation.

Figures 5.22, 5.23 and 5.24 are encrypted packet excerpts obtained thanks to the *Samsung MitM*. They all contacted the domain **api.airbnb.com**.

The packet excerpt associated with Figure 5.22 contains almost all the profile information we entered as a user of the *Airbnb* mobile application. It contains the first name, last name, login, password, birth date, etc. of the user who created the account. Moreover, it contains information about the model number of our *Samsung* mobile phone (*gt-i9505*).

An example of *Airbnb* geo-localization is illustrated by the packet excerpt in Figure 5.23. This packet was emitted when we performed the *Airbnb* tests, in Louvain-la-Neuve.

The packet excerpt in Figure 5.24 contains an example of a San Francisco accommodation search we performed using the *Airbnb* services. We can easily observe that it is an accommodation search thanks to the key words *min_bedrooms*, *min_bathrooms*, *checkin*, *checkout*, etc.

Leak sources		Apple (*iPhone 6*)	Samsung (*Samsung S4*)
Lack of encryption		• **iPhone name**: iPhone de mattieu (Fig. 5.19) • **Address**: louvain, louvain-la-neuve, belgium (Fig. 5.20 and 5.21)	Nothing found
Weak encryption		• 45 % of IP traffic SSL encrypted • No weak cryptographic algorithms used	• 57 % of IP traffic SSL encrypted • No weak cryptographic algorithms used
Weak authentication	Lack of adequate timeout protection	• **Timeout guideline**: 30 minutes • 181 TCP connections opened • Maximum TCP terminated connection duration: 1 minute • 3 TCP connections not terminated • Maximum TCP non-terminated connection duration: 26 minute	• **Timeout guideline**: 30 minutes • 82 TCP connections opened • Maximum TCP terminated connection duration: 4 minutes • 3 TCP connections not terminated • Maximum TCP non-terminated connection duration: 15 minutes
	Lack of certificate validation	• no expired certificates • no self-signed certificates	• no expired certificates • no self-signed certificates
	MitM	• **Name**: thesisfirstname, thesislastname • **Birth date year**: 1993 • **Home address**: taille, vivier • **Phone number**: 477522662 • **Credential**: thesisleaks2016@gmail.com, password, thesis2015 • **Geo-localization**: namur, louvain-la-neuve • **City research**: san francisco, paris	• **Samsung model number**: gt-i9505 (Fig. 5.22) • **Name**: thesisfirstname, thesislastname (Fig. 5.22) • **Birth date year**: 1993 (Fig. 5.22) • **Home address**: taille, vivier, belgium • **Phone number**: 477522662 • **Credential**: thesisleaks2015@gmail.com, password, thesis2015 (Fig. 5.22) • **Geo-localization**: namur, louvain-la-neuve (Fig. 5.23) • **City research**: san francisco (Fig. 5.24)
Covert channels	Adblock	• www-google-analytics.l.google.com • g.msn.com	• www-google-analytics.l.google.com • g.msn.com • stats.g.doubleclick.net • csi.gstatic.com • ssl-google-analytics.l.google.com
	DNS blacklists	No Match	No Match
	SafeBrowsing	No Match	No Match
	SafeLookup	No Match	No Match
	Backdoors	Nothing noteworthy	Nothing noteworthy

Table 5.7: Sources of leaks in the *Airbnb* mobile application

As the figures for the *Apple MitM* contain the same scenarios as in Figures 5.22, 5.23 and 5.24, these data are not shown.

The *Adblock* module blocked the hostnames `www-google-analytics.l.google.com` and `g.msn.com` on both devices. The host **g.msn.com** simply correspond to the *MSN* company and its not clear for which reason it was blocked since there is no mention of harmful behaviour. Besides `www-google-analytics.l.google.com`, the Samsung device also blocked `csi.gstatic.com` and `ssl-google-analytics.l.google.com`; all these hostnames are from *Google's* statistics services.

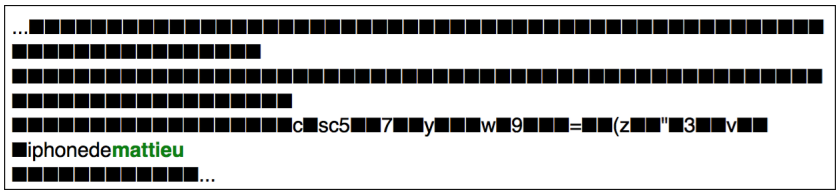


Figure 5.19: Apple (lack of encryption): report excerpt

Figure 5.20: Apple (lack of encryption): report excerpt

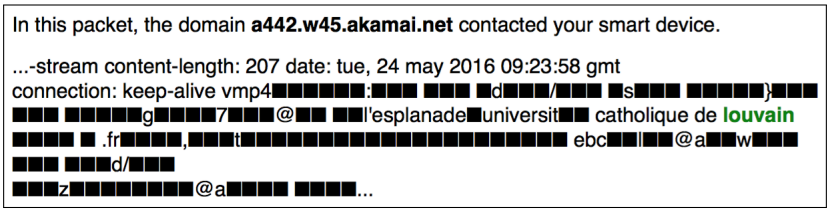


Figure 5.21: Apple (lack of encryption): report excerpt

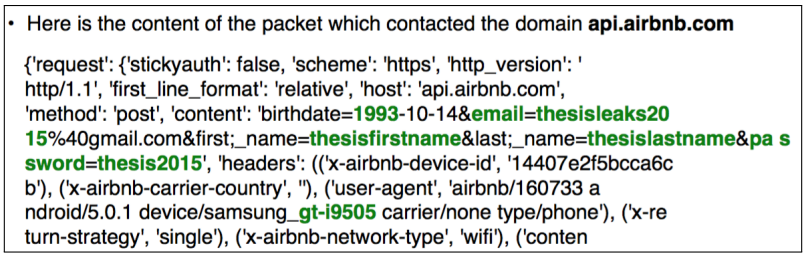


Figure 5.22: Samung MitM: report excerpt

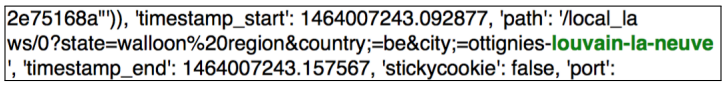


Figure 5.23: Samsung MitM: report excerpt

Since it is unclear how *Google* handles the information they gather and also because these hosts might indicate the usage of the analytics services offered by Google(cf. section 3.4), these domains are potential threat to a user's privacy. Finally, the most interesting finding is **stats.g.doubleclick.net**. The later hostname is a variation of the *doubleclick.net*, a subsidiary of Google proposing advertisement services. The domains with *doubleclick.net* are known for proposing intrusive advertisements and even adwares that are used for cookie tracking.

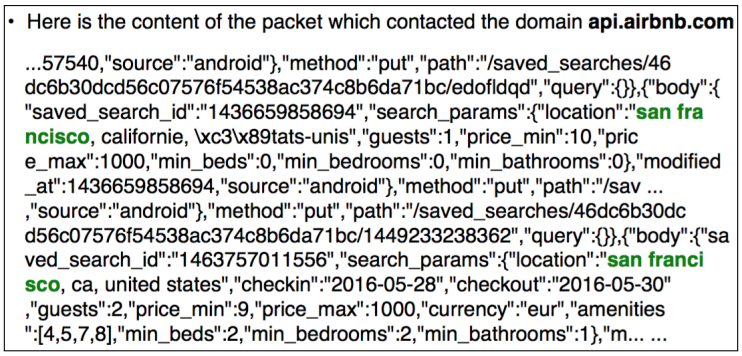


Figure 5.24: Samsung MitM: report excerpt

Airbnb evaluation

The *Airbnb* mobile application passed the *Weak encryption* and *Lack of timeout protection* tests. We did not find any weak cryptographic algorithms used by this service. In addition, most of the connections opened by this service were closed before the timeout guideline (30 minutes).

Concerning the *Lack of encryption* test, we can see that the key words from Figures 5.19, 5.20 and 5.21 are, even if we did not manage to link them properly to the *Airbnb* services, personal key words related to people's names or even people's localizations. This is thus clearly an information leak, even if it was not linked to the *Airbnb* services.

Concerning the *MitM* test, Figures 5.22, 5.23 and 5.24 show that the *Man in the Middle attack* succeeded! This is clearly a bad point for *Airbnb,* knowing that we did not manage to perform such an attack against the *Facebook* and *Snapchat* mobile applications.

The application doesn't have much issues related to *covert channels.* Nevertheless, the analytics service of Google seems to be used by both Apple and Samsung devices.

In the end, except for the *MitM* test and because we cannot clearly state that it failed the *Lack of encryption* test, the *Airbnb* mobile application passed our leak sources test relatively well.

5.1.6 

We tested the *Uber* mobile application for Apple and Samsung smart devices. Table 5.8 summarizes the sources of leaks we found for this mobile application.

Leak sources		Apple (*iPhone 6*)	Samsung (*Samsung S4*)
Lack of encryption		Nothing found	Nothing found
Weak encryption		• 50 % of IP traffic SSL encrypted • **RC4** weak stream cipher algorithm was used by the domains **b.stats.paypal.com**, **slc.stats.paypal.com** and **phx.stats.paypal.com**	• 41 % of IP traffic SSL encrypted • **RC4** weak stream cipher algorithm was used by the domains **b.stats.paypal.com** and **slc.stats.paypal.com**
Weak authentication	Lack of adequate timeout protection	• **Timeout guideline**: 15 minutes • 144 TCP connections opened • Maximum TCP terminated connection duration: 1 minute • 1 TCP connection not terminated • Maximum TCP non-terminated connection duration: 27 minutes	• **Timeout guideline**: 15 minutes • 104 TCP connections opened • Maximum TCP terminated connection duration: 1 minute • 1 TCP connection not terminated • Maximum TCP non-terminated connection duration: 20 minutes
	Lack of certificate validation	• no expired certificates • no self-signed certificates	• no expired certificates • no self-signed certificates
	MitM	• **Name**: thesisfirstname, thesislastname, mattieu (Fig. 5.25 and 5.27) • **Phone number**: 477522662 (Fig. 5.27) **Tracking**: device id, battery status, battery level, os, rooted, altitude (Fig. 5.26) • **Credential**: thesisleaks2016apple@gmail.com, password, thesis2015 (Fig. 5.27)	• **Samsung model number**: gt-i9505 (Fig. 5.12) • **Credential**: thesisleaks2016@gmail.com
Covert channels	Adblock	No Match	• csi.gstatic.com
	DNS blacklists	• e.crashlytics.com • clients.l.google.com • fra15s11-in-f14.1e100.net • googleapis.l.google.com	• clients.l.google.com • csi.gstatic.com • ams15s21-in-f3.1e100.net • android.l.google.com • e.crashlytics.com
	SafeBrowsing	No Match	No Match
	SafeLookup	No Match	No Match
	Backdoors	Nothing noteworthy	Nothing noteworthy

Table 5.8: Sources of leaks in the *Uber* mobile application

Among the different timeout guidelines given in the section 3.3.1, we chose 15 minutes for the mobile application *Uber*. We chose this value because we considered *Uber* as a high security application. In other words, *Uber* mobile services transmit some sensitive private information related to finance (payment).

Figures 5.25, 5.26 and 5.27 are encrypted packet excerpts obtained thanks to the *Apple MitM*. They all contacted the domain **cn-dc1.uber.com**.

Uber tracks the name of our iPhones. The figure 5.25 contains the name of the iPhone which contacted the *Uber* services: *iPhone de Mattieu*. This is an information leak as the name of an iPhone

is often personal. For example, in this case, the iPhone name contains the name of its owner: *Mattieu*.

Uber tracks its user device id. It also tracks surprisingly its user battery status, battery level, os, altitude and the fact that its user devices are rooted or not. Figure 5.26 contains these information.

The packet excerpt associated to the figure 5.27 contains almost all the profile information we entered as a user of the *Uber* mobile application. It indeed contains the first name, last name, login, password, etc. of the user which created his account. This packet was captured during one of our *sign up* in the *Uber* mobile application.

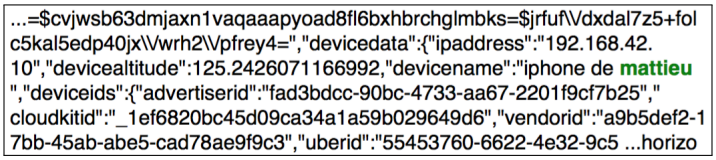


Figure 5.25: Apple MitM: report excerpt

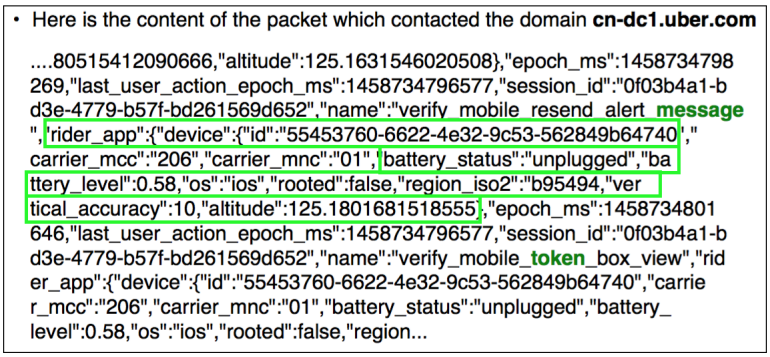


Figure 5.26: Apple MitM: report excerpt

Figure 5.27: Apple MitM: report excerpt

Uber evaluation

The *Uber* mobile application passed the *Lack of encryption* and *Lack of adequate timeout protection* tests. We indeed didn't find any of our personal key words in the unencrypted network traffic of this service. Moreover, the majority of the connections opened by this service were closed before the timeout guideline (15 minutes).

Concerning the *Weak encryption* leak source, we found it originally in testing a specific payment part of the mobile application *Uber*. This payment part was definitely related to *Paypal* as the domains initializing these weak encrypted connections were `b.stats.paypal.com`, `slc.stats.paypal.com` and `phx.stats.paypal.com`. In observing these domains, we suppose that the data sent via these insecure connections were some statistics about users' payments, users' profile, etc. More information about this leak source can be found in the section 5.1.4.

Concerning the *MitM* test, Figures 5.25 and 5.27 show that the *Man in the Middle attack* succeeded for the *Apple smart device*! In observing the key words discovered during the *Samsung MitM*, we can suppose that it didn't succeed for the *Samsung* smart device. The fact that it succeeded for the *Apple smart device* is a bad point for *Uber*, knowing that we didn't succeed to perform such an attack against the *Snapchat* and *Facebook* mobile applications.

The *covert channels* module found domains known for analitycs, in which most of them are related to Google.

Ultimately, the *Uber* mobile application had surprisingly some serious leak sources. First, because of its use of a payment system using a weak encryption algorithm and second, because of its vulnerability to the *Man in the Middle attack*.

5.1.7 

We tested the *Airbsit* mobile application for Apple and Samsung smart devices. Table 5.9 summarizes the sources of leaks we found for this mobile application.

Among the different timeout guidelines given in the section 3.3.1, we chose 15 minutes for the mobile application *Airbsit*. We chose this value because we considered *Airbsit* as a medium security application. In other words, *Airbsit* mobile services transmit some sensitive private information related to finance (payment).

Although the packet excerpt in Figure 5.28 contacts the domain **map.googleapis.com**, we are sure that this kind of packet is issued by the *Airbsit* services when a user confirms his address as a parent or even as a babysitter on the mobile application *Airbsit*. *Airbsit* transmits thus its user home addresses without any encryption. This is an important leak of information as anyone on the same network as an *Airbsit* user can guess his home address by simply capturing its network traffic, at the moment he fills its profile information. In this figure, it is also possible to see that the model number of the smart device (*gt-i9505*) has been tracked.

Leak sources		Apple (*iPhone 6*)	Samsung (*Samsung S4*)
Lack of encryption		• **Home address**: taille, vivier, namur • Some jpg files representing part of our profile pictures could be extracted (Fig. 5.31)	• **Samsung model number**: gt-i9505 (Fig. 5.28) • **Home address**: taille, vivier, namur, belgium (Fig. 5.28) • Some jpg files representing part of our profile pictures could be extracted (Fig. 5.29 and 5.30)
Weak encryption		• 39 % of IP traffic SSL encrypted • No weak cryptographic algorithms used	• 17 % of IP traffic SSL encrypted • No weak cryptographic algorithms used
Weak authentication	Lack of adequate timeout protection	• **Timeout guideline**: 15 minutes • 70 TCP connections opened • Maximum TCP terminated connection duration: 11 minutes • No TCP connection not terminated	• **Timeout guideline**: 15 minutes • 58 TCP connections opened • Maximum TCP terminated connection duration: 4 minutes • 2 TCP connections not terminated • Maximum TCP non-terminated connection duration: 1 minute
	Lack of certificate validation	• no expired certificates • no self-signed certificates	• no expired certificates • no self-signed certificates
	MitM	• **Name**: thesisfirstname, thesislastname • **Gender**: male • **Birth date year**: 1993 • **Home address**: taille au vivier, namur, belgium • **Phone number**: 477522662 • **Credit/Debit card number**: 67732012409987, 4891090129709262 • **Credential**: thesisleaks2016@gmail.com	• **Samsung model number**: gt-i9505 • **Name**: thesisf, thesisl, thesisfi, thesisla (Fig. 5.32, 5.33 and 5.34) • **Gender**: male (Fig. 5.34) • **Birth date year**: 1993 (Fig. 5.32, 5.33 and 5.34) • **Home address**: taille au vivier, namur, belgium (Fig. 5.33 and 5.35) • **Phone number**: 477522662 (Fig. 5.34) • **Credit/Debit card number**: 67732012409987 (Fig. 5.33), 4891090129709262 (Fig. 5.32) • **Email**: thesisleaks2016@gmail.com
Covert channels	Adblock	• ssl-google-analytics.l.google.com • csi.gstatic.com	• ssl-google-analytics.l.google.com • csi.gstatic.com
	DNS blacklists	No Match	No Match
	SafeBrowsing	No Match	No Match
	SafeLookup	No Match	No Match
	Backdoors	Nothing noteworthy	Nothing noteworthy

Table 5.9: Sources of leaks in the *Airbsit* mobile application

We managed to extract part of certain profile pictures from the same *Airbsit* user from the network traffic of the *Airbsit* services. These picture parts are shown in Figures 5.29, 5.30 and 5.31. The first two are pictures extracted from the Samsung smart device network traffic and the third one was extracted from the Apple smart device network traffic. It is possible to see, especially for the pictures extracted from the Samsung smart device network traffic, part of a face in the picture. In the first and second pictures, it is even possible to see this person's eyes. However, we can see that the third image is clearly much more hidden/protected than the others. We did several tests for Samsung and

Apple smart devices and obtained more or less the same result every time. We thus think this could be linked to the type of smart device.

Figures 5.32, 5.33, 5.34 and 5.35 are excerpts from encrypted packets obtained thanks to the *Samsung MitM*. They all contacted the domain **babysit-eu-api.herokuapp.com**.

All the credit and debit card information we entered as a user of the *Airbsit* mobile application are contained respectively in the packet excerpts associated with Figures 5.32 and 5.33. In Figure 5.32, it is even possible to observe the three-digit secure code and the expiration date of the credit card[6].

All the profile information we entered as a babysitter in the *Airbsit* mobile application is contained in the packet excerpt in Figure 5.34.

All the babysitting order information we entered as a parent in the *Airbsit* mobile application is contained in the packet excerpt in Figure 5.35.

As the figures from the *Apple MitM* contain the same scenarios as in Figures 5.32, 5.33, 5.34 and 5.35, these data are not shown.

In this packet, the domain **maps.googleapis.com** was contacted by your smart device.

get /maps/api/js/geocodeservice.search?4s1%20chemin%20de%20la%20**taille**
%20au%20**vivier**%2c%205000%20**namur**%2c%20**belgium**&7sus&9sen-us&key;=aizas ya
6teyjmytlrrd3gbndg2psmmhqkltddgy&callback;=_xdc_._aw4v3b&**token**=121518 h
ttp/1.1 host: maps.googleapis.com connection: keep-alive accept: */
* user-agent: mozilla/5.0 (linux; android 5.0.1; **gt-i9505** build/lrx22

Figure 5.28: Samsung (lack of encryption): report excerpt

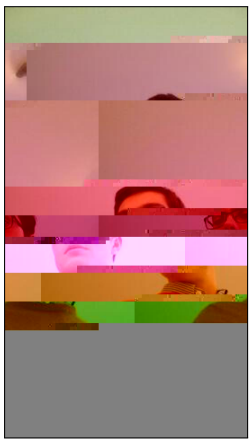


Figure 5.29: Samsung (lack of encryption): report excerpt

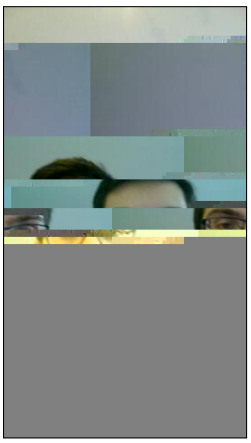


Figure 5.30: Samsung (lack of encryption): report excerpt

Figure 5.31: Apple (lack of encryption): report excerpt

[6]Obviously, this information is hidden, as it is clearly personal.

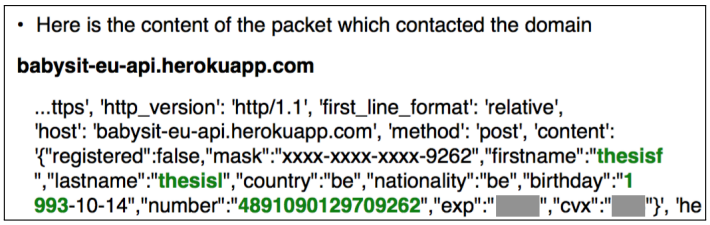


Figure 5.32: Samsung MitM: report excerpt

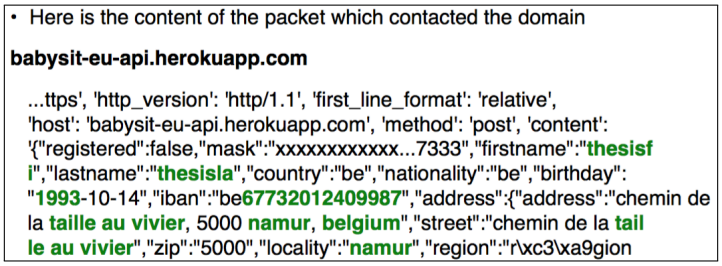


Figure 5.33: Samsung MitM: report excerpt

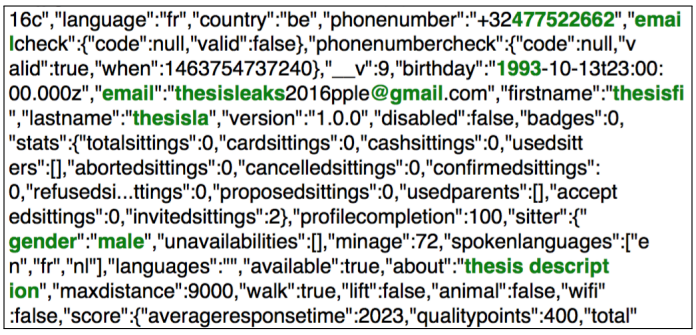


Figure 5.34: Samsung MitM: report excerpt

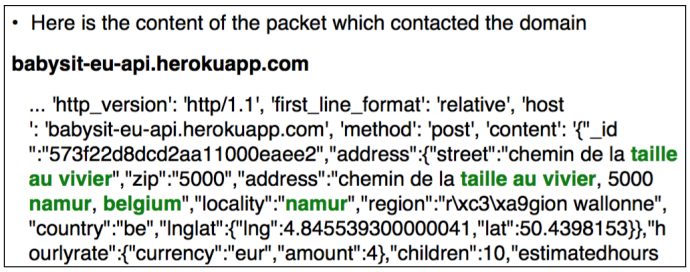


Figure 5.35: Samsung MitM: report excerpt

Airbsit evaluation

The *Airbsit* mobile application passed the *Weak encryption* and *Lack of timeout protection* tests. We did not find any weak cryptographic algorithms used by this service. In addition, most of the connections opened by this service were closed before the timeout guideline (15 minutes).

With regards to the *Lack of encryption* test, we see that the key words from Figure 5.28 are personal key words related to the localization of people's homes. Moreover, we can see that the images

extracted from the *Airbsit* network traffic (Fig. 5.29, 5.30 and 5.31) are related to user profile pictures. These are thus clearly information leaks that we can associate with the mobile application *Airbsit*[7].

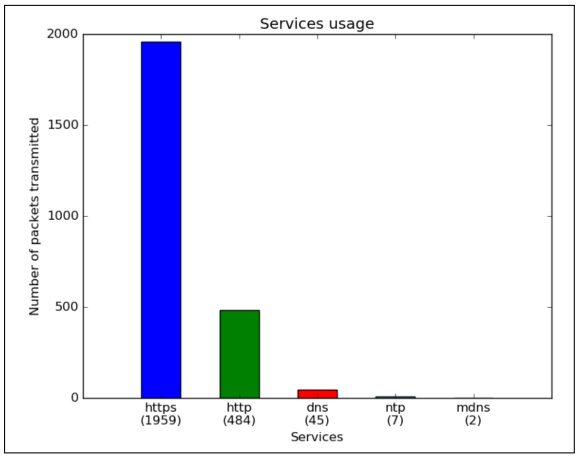


Figure 5.36: Apple services usage

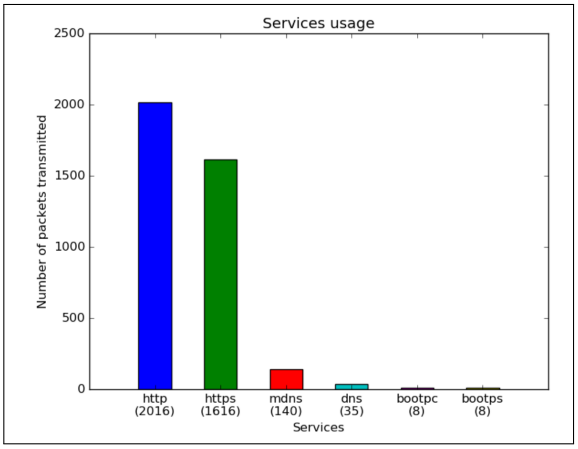


Figure 5.37: Samsung services usage

Moreover, in the figure 5.37, we can clearly see that *http* is used more than *https* in the Samsung device. In figure 5.36, the amount of *http* traffic is not negligible since it represents almost a fourth of the total traffic accessing some services. There is clearly a weakness in term of encryption.

Regarding the *MitM* test, Figures 5.32, 5.33, 5.34 and 5.35 show that the *Man in the Middle attack* succeeded!

The module for *covert channels* only noted some Google's services related to analytics. There is a minor threat of phishing.

Ultimately, the mobile application *Airbsit* performed fairly badly in our leak sources test. It did not pass the *Lack of encryption* test, which is one of the most important. Moreover, the *Man in the Middle attack* was possible to perform against the *Airbsit* services.

5.1.8 Weak encryption Apple leak source

As this leak source concerns only the Apple device for almost every mobile application tested, we decided to describe it in a different section to that dedicated to each mobile application tested.

Regarding the Apple mobile applications *Facebook*, *Paypal*, *Airbnb*, *Uber*, *Airbsit* and *MeetMe*, we found that part of the network traffic exchanged with these services was encrypted using the *RC4* weak stream cipher encryption algorithm. We also found that part of this network traffic was using the *MD5* hash algorithm, which is considered weak by the cryptographic community. Surprisingly, when tracing the source domains which were initializing these connections[8], we found that they were all linked directly to the Apple industry. We then dug a little deeper and read the content of the pack-

[7] Each *Airbsit* user is indeed not authorized to see each profile (picture) of each of the other *Airbsit* users.

[8] `mt-ingestion-service-mr22.itunes-apple.com.akadns.net`, `mt-ingestion-service-st11.itunes.apple.com`, `p24-buy.itunes.apple.com`, `pd-nk.itunes-apple.com.akadns.net`, `lcdn-locator-usuqo.apple.com.akadns.net`, `gsp-ssl.ls-apple.com.akadns.net` and `keyvalueservice.icloud.com.akadns.net`

ets. We found some interesting key words such as *"Symantec"*, *"Certification Authority"*, *"VeriSign"*, etc. When researching the words *"Symantec VeriSign"* on Google, we found that Symantec offered Business SSL certificate solutions. We thus concluded, as this weakness was found only in the Apple device's network traffic, that Apple was using this service to distribute certificates and that this service was using these weak cryptographic algorithms to distribute the certificates.

We also discovered that some of the domains related to *iCloud* (Apple software providing cloud storage solutions) were using the *RC4* weak stream cipher encryption algorithm and the *MD5* weak hash algorithm. Here are the domains:

- `caldav.icloud.com.akadns.net`
- `setup.icloud.com.akadns.net`
- `p25-availability.icloud.com.akadns.net`
- `p25-btmmdns.icloud.com.akadns.net`

5.2 Smart TVs

To control our *Samsung smart TV*[9], we chose to perform two types of test. First, we recorded its network traffic during one entire day of the week (24 hours). Second, we recorded its network traffic during one entire week-end (48 hours). During the weekday test, we watched the news at 1 PM and 8 PM. During the week-end test, the television was not watched.

Smart TV
• 80 (http)
• 443 (https)
• 4443 (pharos)
• 6000 (X11)
• 7676 (imqbrokerd)

Table 5.10: Open ports on smart TV

Table 5.10 reports the list of ports that were open while the devices were unused. We can see that the smart TV has several active services. We could not find any information about the purpose of the service *pharos*. The service *X11* is a version of the *X Window System protocol* that uses a client-server model to produce a graphic display; the service is known to be vulnerable, allowing external entities to connect to the display ([57]). The service *imqbrokerd* is simply a *Message Queue broker*.

Table 5.11 summarizes the leak sources we found for the *Samsung smart TV*.

Among the different timeout guidelines given in the section 3.3.1, we chose 30 minutes for the smart TV. We chose this value because we considered smart TV as a medium security application. In

[9] Samsung 2013 SMART LED-TV 46"

Leak sources		Samsung smart TV (weekday test)	Samsung smart TV (week-end test)
Lack of encryption		**Nothing found**	**Nothing found**
Weak encryption		• 48 % of the IP traffic SSL encrypted • **RC4** weak stream cipher algorithm was used by the domain **fkp.samsungcloudsolution.com**	• 40 % of the IP traffic SSL encrypted • **RC4** weak stream cipher algorithm was used by the domain **fkp.samsungcloudsolution.com**
Weak authentication	Lack of adequate timeout protection	• 2753 TCP connections opened • Maximum TCP terminated connection duration: 1 minute • 19 TCP connections not terminated • Maximum TCP non-terminated connection duration: 1007 minutes (> 30 minutes!) • The non-terminated connection initiated by the domain **noticeprd1.cloudapp.net** lasted 1007 minutes • The non-terminated connection initiated by the domain **time.samsungcloudsolution.com** lasted 841 minutes • The non-terminated connection initiated by the domain **noticeprd.cloudapp.net** lasted 443 minutes	• 193 TCP connections opened • Maximum TCP terminated connection duration: 1 minute • 22 TCP connections not terminated • Maximum TCP non-terminated connection duration: 528 minutes (> 30 minutes!) • The non-terminated connection initiated by the domain **prd-snap-broker-elb-17377633.eu-west-1.elb.amazonaws.com** lasted 528 minutes • The non-terminated connection initiated by the domain **noticeprd1.cloudapp.net** lasted 386 minutes • The non-terminated connection initiated by the domain **noticeprd.cloudapp.net** lasted 386 minutes • The non-terminated connection initiated by the domain **time.samsungcloudsolution.com** lasted 120 minutes • The non-terminated connection initiated by the domain **game.internetat.tv** lasted 120 minutes • The non-terminated connection initiated by the domain **googleapis.com** lasted 120 minutes • The non-terminated connection initiated by the domain **PRD-ONTV-OPENAPI-ELB-Oregon-337487521.us-west-2.elb.amazonaws.com** lasted 120 minutes
	Lack of certificate validation	• no expired certificates • 1 self-signed certificate issued by self-signed.ueiwsp.com	• no expired certificates • no self-signed certificates
	MitM	Unsuccessful	Unsuccessful
Covert channels	Adblock	No Match	No Match
	DNS blacklists	• ns11.whois.co.kr • ytimg.l.google.com • i.ytimg.com • media.internetat.tv • selfsigned.ueiwsp.com • googleapis.l.google.com • ams15s21-in-f142.1e100.net	No Match
	SafeBrowsing	No Match	No Match
	SafeLookup	No Match	No Match
	Backdoors	Nothing Noteworthy	Nothing Noteworthy

Table 5.11: Leak sources for the *Samsung smart TV*

other words, smart TV services transmit some sensitive private information, not related to finance

(payment).

Figures 5.38 and 5.39 represent the packet number that was transmitted per hour during the weekday and week-end experiments. The experiment in Figure 5.38 began one day at 9 AM and lasted 24 hours. The experiment in Figure 5.38 began on a Friday at midnight and lasted 48 hours.

In Figure 5.38, during the weekday test, we can easily see the two peaks of data transmission located during news time (1 PM and 8 PM (13 and 20 on this graph)). We can also see that some data was transmitted during the night, which is rather strange... In Figure 5.39, we can see that network transmission was regular during the week-end: two packets per hour.

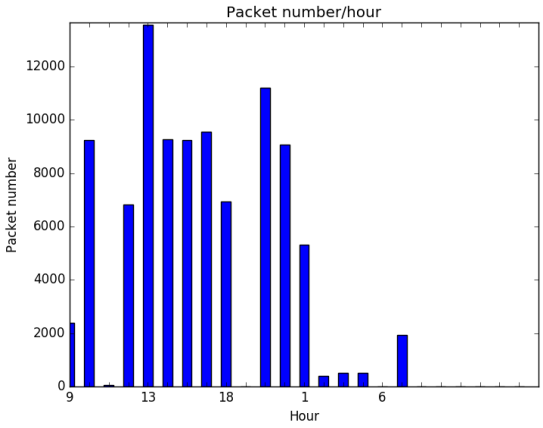


Figure 5.38: Smart TV: weekday experiment

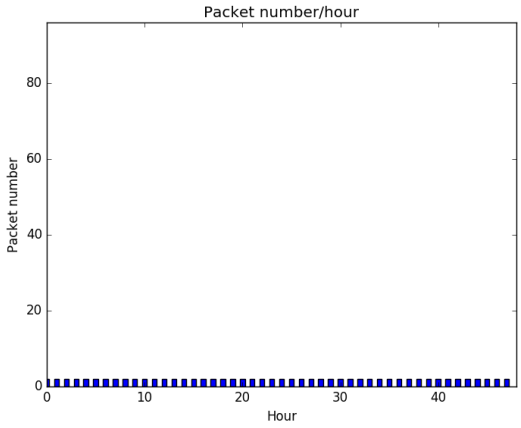


Figure 5.39: Smart TV: week-end experiment

Samsung TV evaluation

The *Samsung TV* passed the *Lack of encryption* test. We did not find any of our personal key words in the unencrypted network traffic of this service.

Regarding the *Weak encryption* test, the *RC4* weak stream cipher encryption algorithm was used to encrypt connections initialized by the domain **fkp.samsungcloudsolution.com**. As this domain is clearly related to the Samsung cloud service, and as this cryptographic algorithm is considered weak by the cryptographic community, this is a serious leak source.

For the *Lack of timeout protection* test, some non-terminated connections lasted more than the recommended duration guideline (30 minutes). The non-terminated connections initiated by the domains `noticeprd1.cloudapp.net`, `noticeprd.cloudapp.net`, `prd-snap-broker-elb-17377633.eu-west-1.elb.amazonaws.com`, `game.internetat.tv`, and `time.samsungcloudsolution.com`, `googleapis.com` and `PRD-ONTV-OPENAPI-ELB-Oregon-337487521.us-west-2.elb.amazonaws.com` lasted considerably more than 30 minutes. This is a leak source as it could provide attackers with unlimited time to impersonate users' smart TVs.

For the *MitM* test, we did not find a way to perform it because we did not find the right tool to do so. Smart TVs use protocols that are different to normal smart devices such as smart phones. This

is why the *Mitmproxy* tool ([58]) we used for the *smart phone MitM* did not work on the smart TVs.

One interesting finding in the *Covert Channels* module is the domain `selfsigned.ueiwsp.com`, blocked by the *DNSBL* sub-module. This domain is also the issuer of a self-signed certificate that was found in the *Weak Authentication* module. Unfortunately, we were unable to find much information on this domain beyond the fact that it is related to *Universal Electronics*.

Ultimately, except for the *Weak encryption* test, the *lack of timeout protection* test and because we were unable to perform a *Man in the Middle attack*, the *Samsung smart TV* passed our leak source tests reasonably well. We did not find any traces of conversations transmitted by this smart TV. Of course, this does not mean that such things are not happening...

5.3 Performance summary

In this section, we will describe the performances of our detection system for sources of information leaks. This means providing a summary of each leak source type we managed to detect. In addition, we will state the type of leak source in Section 3 that we were never able to detect because of this system.

First, the *Lack of encryption* leak source type could be detected by our system thanks to personal key words we entered as input for the system. It detected several *Lack of encryption* leak source problems in the network traffic of the mobile applications *MeetMe*, *Airbsit*, etc.

Second, the *Weak encryption* leak source type could be detected by our system thanks to weak cryptographic algorithm names we entered as input for the system. It detected several *Weak encryption* leak source problems in the mobile application *Paypal* and in the network traffic of the smart TV.

Third, concerning the *Weak authentication* leak source type, we will analyze each of its sub leak source types. The *Lack of adequate timeout protection* leak source type could be detected by our system thanks to predefined TCP connection timeout guidelines we entered as input for the system. It detected several *Lack of adequate timeout protection* leak source problems in the network traffic of the smart TV. The *Lack of certificate validation* leak source type did not find many certificates that were actually invalid. In fact, the detection system found a large number of self-signed certificates, but, as expected, they turned out to be false positives because we did not have access to a complete database of the CAs trusted by smart devices (cf. Section 4.5). Of course, we also captured our self-signed certificate used for the *MitM*, but decided not to report it in the results of the experiments as this is normal behavior. The *MitM* leak source type could be detected by our system. It detected that it was possible to perform this attack against several mobile applications such as *Paypal*, *MeetMe*, *Airbnb*, *Uber*, etc.

Fourth, regarding the *Covert channels* module, we first noticed that *SafeBrowsing* and *SafeLookup* never found any matches in the services we tested. We can thus assume that either the risks covered

by Google's databases are not commonly present in smart devices, or these databases are far too incomplete. On the other hand, the services that we tested are maybe not representative enough of the malicious behaviors that can be found on networked smart devices. We can make a similar assumption concerning the *Backdoors* sub-module as very few instances of suspicious behavior were found. Of the domains blocked by the *Adblock* and *DNSBL* sub-modules, we noted that most of them were related either to analytics or tracking services, in other words statistics with a risk of phishing. We did not find any actual proof of malware or unwanted software injection.

Finally, in order to give an overview of the leak source automated report our system generates, we added the automated reports generated for the mobile applications *Paypal* (including the *MitM part*), *Airbsit* and *MeetMe* in the appendix.

Chapter 6

Conclusion

Nowadays, an increasing amount of services for smart device require large network infrastructures to sustain the load created by millions or even billions of users, such as *Facebook*, *Instagram*, *Uber*, etc. As these services are becoming more and more popular and as their infrastructures are getting bigger, it is becoming very important that users and especially developers be warned about the sources of information leaks induced by these network services.

Therefore, we oriented the subject of our dissertation towards the design of a detection system for the sources of information leaks on networked smart devices. We did not only provide a system capable of detecting information leaks from the smart device's network traffic, but we also provided a warning system capable of detecting some of these sources of information leaks. In other words, our system is able to output certain information leaks as well as the causes of these leaks. Our detection system is able to find four different types of sources of information leakage: *Lack of encryption*, *Weak encryption*, *Weak authentication* and *Covert channels*. Some of theses sources cover more than one risk leading to leaks (cf. section 4.1).

The results of our experiments show that we attained our objectives. They prove that our detection system is able to detect almost all the sources of leaks we defined. We tested our system on several smart device services (mobile applications such as *Facebook*, *Paypal*, *Airbnb*, etc. for smart phones; casual TV watching for smart TVs) in order to assess them.

The following list gives some of our most interesting findings:

- *MeetMe* mobile application sends information about the localization, the gender and device information about its users to some tracking ads.
- *Airbsit* mobile application sends the home addresses of its users unencrypted via the network (*Lack of encryption*).
- During the *Airbnb* mobile application tests, some unencrypted addresses near our localization were retrieved via the network by our smart phones (*Lack of encryption*).
- *Paypal* mobile application uses the *RC4* weak stream cipher encryption algorithm to send statistics about its users (*Weak encryption*).
- The *smart TV* uses the *RC4* weak stream cipher encryption algorithm to connect to its cloud service (*Weak encryption*).
- *Paypal*, *Airbnb*, *Airbsit*, *MeetMe* and *Uber* are vulnerable to a Man in the Middle attack (*Weak authentication*).

- *Paypal* is particularly vulnerable to the *Man in the Middle attack*, as we succeeded to modify the amount in a live transaction we made (*Weak authentication*).
- *Snapchat*, for the Apple devices, opens very few connections in which many are not terminated (*Weak authentication*).
- The smart TV initializes abnormally long connections (*Weak authentication*).
- The smart TV uses a self-signed certificate issued by the domain *selfsigned.ueiwsp.com*; this domain was also blocked by the DNSBL module (*Weak authentication* and *Covert Channels*).

Concerning the *Lack of certificate validation*, we noticed that, in general, the devices and applications tested didn't used expired or self-signed certificates, despite some being vulnerable to the *MitM* attack.

Concerning the *Covert channels* as source of information leaks, our detection system brought to light that this source was not a very common feature of the tested softwares and smart devices, compared to the other sources of data leakage. Nevertheless, we noticed that among the findings of the *covert channels* module, most of them were related to analytics (statistics), which comforted us in the idea that information are being gathered excessively at the expense of the users.

After analysing our findings from the various services for smart devices, we can see that our detection system propose a completely new approach to evaluate the risk of information leaks. It not only detects pure information leaks in the network of smart device, like many other existing systems do, but it is also able to detect sources of information leaks such as the use of weak cryptographic algorithms, the use of a weak authentication system, malicious domains contacted, etc. Moreover, It gives the possibility to perform a *Man in the Middle attack* on the service tested, which means that the user/developer plays a part in the detection system for the sources of information leaks. Our system thus has real potential for helping developers enhance their network security techniques. Additionally, our system can help users to assess the risk of network information leaks, implied by the usage of some devices or softwares.

To conclude, we managed to build a working detection system for the sources of information leakage on networked smart devices. We have proved that our detection system was able to detect sources as well as concrete leaks from various smart devices and services running on them (*MeetMe*, *Airbsit*, *Airbnb*, etc.), even from very popular ones (*Paypal*). However, the subject of information leakage progresses constantly and we are aware that there is not a single solution for detecting information leaks and their sources. We hope that our work has played and will continue to play a part in discovering new information leaks as well as sensitizing application developers about network security. We additionally hope our approach will inspire other works on this topic.

Limitations and future work

Concerning the limitations of our detection system for the sources of information leaks, it is obvious that many additional sources of information leakage could be controlled by our system. For example, in the test related to *Weak encryption* and *Weak authentication*, we only control TCP connections. It could have been interesting to control other protocol types. Moreover, we only perform one type of network attack against smart device services, the *Man in the middle attack*. Also, we cannot deny that the field of information leaks is in constant evolution. Our system, despite being powerful, still needs to be updated regularly. However, we tried to control the main sources of leaks with this detection system within the limits of our knowledge and the time available.

In the state-of-the-art literature we consulted, we progressively understood that there were two ways of addressing the issue created by sources of information leaks. First, discovering only information leaks and making them public in order to sensitize a large number of people and even certain developers. Second, discovering *sources* of information leaks in a more in-depth way, making them public and thus sensitizing developers first. We chose the second way for this dissertation and it turned out to be a very informative approach. Therefore, for future work, we would recommend to proceed with this approach in order to warn developers, in particular, by exposing *sources* of information leak and not just information leaks.

Moreover, our system is designed to work in a closed, and thus local, environment. Developers could easily build our system on small servers connected to the internet modem and control the information leak sources for the smart service they are developing. However, our system requires knowledge of computer science for its setup. It could effectively be difficult for someone unfamiliar with the subject to install our detection system on a server. We could still imagine, for the future, building a black box that ordinary users could plug into their internet modem in order to discover information leaks and information leak sources in their network traffic.

List of abbreviations and symbols

- **api**: application programming interface
- **CA**: Certification Authority
- **cf**: confer
- **DES**: Data Encryption Standard
- **DNS**: Domain Name System
- **DSL**: Digital Subscriber Line
- **DNSBL**: Domain Name system Black Listing
- **ECB**: Electronic Code Book
- **ENISA**: European Union Agency for Network and Information Security
- **ftp**: file transfer protocol
- **Fig**: Figure
- **GBs**: Gigabytes
- **HTML**: HyperText Mark-Up Language
- **HTTP**: Hypertext Transfer Protocol
- **HTTPS**: HyperText Transfer Protocol Secure
- **IoT**: Internet of Things
- **IP**: Internet Protocol
- **ISP**: Internet Service Provider
- **lat**: latitude
- **LCG**: Linear Congruential Generator
- **lng**: longitude
- **Mbits**: Megabits
- **MD-4**: Message Digest 4
- **MD-5**: Message Digest 5
- **MitM**: Man in the Middle attack
- **modem**: modulator-demodulator

- **OSI**: Open Systems Interconnection
- **OTP**: One-Time Password
- **OWASP**: Open Web Application Security Project
- **RC2**: Rivest Cipher 2
- **RC4**: Rivest Cipher 4
- **RIPEMD-128**: RACE Integrity Primitives Evaluation Message Digest-128
- **SCTP**: Stream Control Transmission Protocol
- **SHA1**: Secure Hash Algorithm(1)
- **smart TV**: smart television
- **SMS**: Short Message Service
- **SOAP**: Simple Object Access Protocol
- **ssh**: secure shell
- **SSL**: Secure Sockets Layer
- **TCP**: Transmission Control Protocol
- **TLS**: Transport Layer Security
- **UBE**: Unsolicited Bulk Email
- **UDP**: User Datagram Protocol
- **URL**: Uniform Resource Locator
- **USB**: Universal Serial Bus
- **WiFi**: Wireless Fidelity

Bibliography

[1] **Jennifer Dudley-Nicholson**. *One in three Google Android apps 'leak' information but many Aussie users remain unaware of the risk* [online]. news.com.au, published in 2015, [consulted 20 November 2015]. Available at: http://www.news.com.au/technology/online/security/one-in-three-google-android-apps-leak-information-but-many-aussie-users-remain-unaware-of-the-risk/news-story/06d65f0635e8ce18c26a0602bf501c12

[2] **Brian Donohue**. *Dozens of Popular Android Apps Leak Sensitive User Data* [online]. Kaspersky Daily lab, published in 2014 [consulted 15 May 2016]. Available at: https://blog.kaspersky.com/privacy_holes_in_popular_android_apps/6047/

[3] **Wall Street Journal**. *Is Your Samsung Smart TV Spying on You?* [online]. Wall Street Journal, published in 2015, [consulted 20 November 2015]. Available at : http://www.wsj.com/video/is-your-samsung-smart-tv-spying-on-you/19F3D7D2-1F18-464C-8884-7EA250000F8C.html

[4] **Norton**. *Using the App Advisor feature in Norton Mobile Security* [online]. Norton support, published in 2016 [consulted 29 May 2016]. Available at: https://support.norton.com/sp/en/us/home/current/solutions/v97499944_EndUserProfile_en_us

[5] **Ms. Smith**. *What apps sell or steal your data or take over your phone? PrivacyHawk can tell you* [online]. NETWORKWORLD, published in 2015 [consulted 29 May 2016]. Available at: http://www.networkworld.com/article/2930791/microsoft-subnet/what-apps-sell-or-steal-your-data-or-take-over-your-phone-privacyhawk-can-tell-you.html

[6] **Tom Simonite**. *How to Detect Apps Leaking Your Data* [online]. MIT Technology Review, published in 2012 [consulted 29 May 2016]. Available at: https://www.technologyreview.com/s/428772/how-to-detect-apps-leaking-your-data/

[7] *Panama Papers*. In : *Wikipedia* [online]. Wikimedia Foundation[consulted 25 May 2016]. Available at: https://en.wikipedia.org/wiki/Panama_Papers

[8] *Privacy*. Dictionary.com [online] [consulted 27 May 2016]. Available at : http://www.dictionary.com/browse/privacy

[9] *Information privacy*. In : *Wikipedia* [online].[consulted 5 October 2015]. Available at: https://en.wikipedia.org/wiki/Information_privacy

[10] *Information leakage*[online]. The Web Application Security Consortium [consulted 5 October 2015]. Available at: http://projects.webappsec.org/w/page/13246936/Information%20Leakage

[11] Internet of Things. In *Wikipedia* [online]. Fondation Wikimedia [consulted 5 October 2015]. Available at : https://en.wikipedia.org/wiki/Internet_of_Things

[12] **Newsroom Editor**. Conclusions of the Internet of Things public consultation . In : *Digital Agenda for Europe* [online]. European Commission, published in February 2013 [consulted 10 October 2015]. Available at: http://ec.europa.eu/digital-agenda/en/news/conclusions-internet-things-public-consultation

[13] *Privacy in the Age of the Smartphone* [online]. PRC, published in August 2005, updated in May 2015 [consulted 21 October 2015]. Available at: https://www.privacyrights.org/content/privacy-age-smartphone

[14] The Internet of Things and privacy in Europe and the USA. In : *TaylorWessing* [online]. TaylorWessing, March 2015 [consulted 10 October 2015]. Available at: http://united-kingdom.taylorwessing.com/globaldatahub/article_wp29_iot.html

[15] **Kim Walker**.The legal considerations of the internet of things. In *ComputerWeekly* [online]. TechTarget[consulted 15 October 2015]. Available at : http://www.computerweekly.com/opinion/The-legal-considerations-of-the-internet-of-things

[16] **OWASP**. *OWASP Mobile Security Project* [online]. OWASP, published in 2014, [consulted 25 November 2015]. Available at: https://www.owasp.org/index.php/OWASP_Mobile_Security_Project#tab=Top_10_Mobile_Risks

[17] **Dr. Giles Hogben, Dr. Marnix Dekker, ENISA**. *Smartphones: Information security risks, opportunities and recommendations for users* [online]. ENISA, published in 2010, [consulted 10 May 2016]. Available at: https://www.enisa.europa.eu/publications/smartphones-information-security-risks-opportunities-and-recommendations-for-users

[18] **Joshua J. Drake, Pau Oliva Fora, Zach Lanier, Collin Mulliner, Stephen A. Ridley, Georg Wicherski**. *Android Hacker's Handbook*. John Wiley & Sons, 2014. 576 p. ISBN 111860864X

[19] **G. Avoine**. *Secured System Engineering(LINGI-2144 course)*. UCL. Chapter 4: Basics of Cryptography, chapter 8: Generating Randomness, chapter 9: Implementation of Hash Functions, chapter 10: Implementation of Block Ciphers, chapter 11: Implementation of Stream Ciphers.

[20] **Wikipedia**. *OSI model* [online]. Wikipedia, published in 2016 [consulted 15 May 2016]. Available at: https://en.wikipedia.org/wiki/OSI_model

[21] **F. Koeune – O. Pereira**. *Introduction to Cryptography(MAT2450 course)*. UCL. Slides 09.

[22] **Priyank Gupta**. *Validating SSL certificates in mobile apps* [online]. tumblr, published in 2014, [consulted 20 March 2016]. Available at: http://priyaaank.tumblr.com/post/81172916565/validating-ssl-certificates-in-mobile-apps

[23] *Phishing*. In : *Wikipedia* [online]. Wikimedia Foundation[consulted 6 April 2016]. Available at: https://en.wikipedia.org/wiki/Phishing

[24] *Malware*. In : *Wikipedia* [online]. Wikimedia Foundation[consulted 6 April 2016]. Available at: https://en.wikipedia.org/wiki/Malware

[25] **G. Avoine**. *Secured System Engineering(LINGI-2144 course)*. UCL. Chapter 5: Authentication protocols.

[26] *Covert Channel* [online]. Technopedia[consulted 6 April 2016]. Available at: https://www.techopedia.com/definition/10255/covert-channel

[27] *Google analytics* [online]. Google[consulted 20 February 2016]. Available at: https://analytics.google.com

[28] *Spyware*. In : *Wikipedia* [online]. Wikimedia Foundation[consulted 6 April 2016]. Available at: https://en.wikipedia.org/wiki/Spyware

[29] *How does anti-virus software work?* [online]. AntivirusWorld[consulted 6 April 2016]. Available at: http://www.antivirusworld.com/articles/antivirus.php

[30] *Unwanted Software Policy* [online]. Google[consulted 25 February 2016]. Available at: https://www.google.com/about/company/unwanted-software-policy.html

[31] **Margaret Rouse**.*back door* [online]. TechTarget[consulted 22 May 2016]. Available at: http://searchsecurity.techtarget.com/definition/back-door

[32] **Andrew Tabona**. *The Top 20 Free Network Monitoring and Analysis Tools for Sys Admins* [online]. TalkTechToMe, published in 2015, [consulted 20 October 2015]. Available at: http://www.gfi.com/blog/the-top-20-free-network-monitoring-and-analysis-tools-for-sys-admins/

[33] **Eric Lawrence, Telerik company**. *Fiddler*. Published in 2012. Available at: http://www.telerik.com/fiddler

[34] **Jon Dugan, Seth Elliott, Bruce A. Mah, Jeff Poskanzer, Kaustubh Prabhu**. *iperf*. Published in 2014. Available at: https://iperf.fr

[35] **lady Ada**. *Setting up a Raspberry Pi as a WiFi access point* [online]. adafruit, published in 2015, [consulted 23 October 2015] Available at: https://learn.adafruit.com/downloads/pdf/setting-up-a-raspberry-pi-as-a-wifi-access-point.pdf

[36] **United States Air Force Office of Special Investigations - Center for Information Systems Security Studies and Research**. *Foremost*. Published in 2006. Available at: http://foremost.sourceforge.net

[37] **Laurent Gaffie - Psychomario**. *Net-creds*. Published in 2015. Available at: https://github.com/DanMcInerney/net-creds

[38] *Domain Name System* [online]. Wikimedia Foundation[consulted 7 April 2016]. Available at: https://en.wikipedia.org/wiki/Domain_Name_System

[39] *User Datagram Protocol*. In : *Wikipedia* [online]. Wikimedia Foundation[consulted 14 May 2016]. Available at: https://en.wikipedia.org/wiki/User_Datagram_Protocol

[40] *Safe Browsing API* [online]. Google[consulted 22 November 2015]. Available at: https://developers.google.com/safe-browsing/

[41] *URL - Uniform Resource Locator*. In : *Webopedia* [online]. Quinstreet Enterprise,[consulted 7 April 2016]. Available at: http://www.webopedia.com/TERM/U/URL.html

[42] **Julien Sobrier**. *Python library for Google Safe Browsing v3 Lookup API* [online]. Ghithub,[consulted 23 November 2015]. Available at: https://github.com/juliensobrier/google-safe-browsing-lookup-python

[43] **Aleh Filipovich**. *Python client library for Google Safe Browsing API* [online]. Ghithub,[consulted 23 November 2015]. Available at: https://github.com/afilipovich/gglsbl

[44] *About Adblock Plus* [online]. Pelican[consulted 27 November 2015]. Available at: https://adblockplus.org/en/about

[45] *EasyList* [online]. Eyeo[consulted 27 November 2015]. Available at: https://easylist.github.io/

[46] **Mikhail Korobov**.*adblockparser* [online]. Ghithub,[consulted 28 November 2015]. Available at: https://github.com/scrapinghub/adblockparser

[47] *What is a DNSBL?* [online]. CGP Holdings[consulted 24 November 2015]. Available at: http://www.dnsbl.info/

[48] *Email spam* [online]. Wikimedia Foundation[consulted 15 April 2016]. Available at: https://en.wikipedia.org/wiki/Email_spam

[49] *Internet Service Provider (ISP)* [online]. Technopedia[consulted 21 April 2016]. Available at: https://www.techopedia.com/definition/2510/internet-service-provider-isp

[50] *Well-Known TCP Port Numbers*. In : *Webopedia* [online]. Quinstreet Enterprise,[consulted 23 May 2016]. Available at: http://www.webopedia.com/quick_ref/portnumbers.asp

[51] **Ryan Mazerik**.*DNS tunnelling* [online]. Infosec Institute[consulted 22 May 2016]. Available at: http://resources.infosecinstitute.com/dns-tunnelling/

[52] **Jane Wakefield**.*What is Facebook doing with my data?*. In : *BBC News Service* [online]. BBC, published in November 2015 [consulted 1 June 2016]. Available at: http://www.bbc.com/news/magazine-34776191

[53] **GIBSONSEC**. *Snapchat security advisory* [online]. GIBSONSEC, published in 2013, [consulted 25 May 2016]. Available at: http://gibsonsec.org/snapchat/

[54] **Cale Guthrie Weissman**. *Snapchat's drastic security measures shut down entire 'Internet neighborhoods' on the service* [online]. Business Insider UK, published in 2015, [consulted 25 May 2016]. Available at: http://uk.businessinsider.com/snapchat-takes-drastic-security-measures-2015-4?r=US&IR=T

[55] **Joanna Geary and Nicola Hughes**. *Adnxs (AppNexus): What is it and what does it do?* [online]. Theguardian, published in 2012 [consulted 29 May 2016]. Available at: https://www.theguardian.com/technology/2012/apr/23/adnxs-tracking-trackers-cookies-web-monitoring

[56] **G. Avoine**. *Secured System Engineering(LINGI-2144 course)*. UCL. Chapter 11: Implementation of Stream Ciphers.

[57] **Joe Barrett**.*X11 Hacking* [online]. Joe's Journal, published in April 2014 [consulted 2 June 2016]. Available at: http://winterspite.com/security/x11-hacking/

[58] **Aldo Cortesi - Maximilian Hils**. *Mitmproxy*. Published in 2014. Available at: https://mitmproxy.org

[59] **Rene Millman**. *Too many apps leak personal data to third parties, report finds* [online]. SC UK, published in 2015, [consulted 04 December 2015]. Available at: http://www.scmagazineuk.com/too-many-apps-leak-personal-data-to-third-parties-report-finds/article/452249/

[60] **Rohit Tamma, Donnie Tindall**. *Learning Android Forensics*. Packt Publishing Ltd., 2015. 337 p. ISBN 1782174575

[61] **Mattia Epifani,Pasquale Stirparo**. *Learning iOS Forensics*. Packt Publishing Ltd., 2015. 164 p. ISBN 1783553510

[62] **Charlie Miller, Dionysus Blazakis, Dino Dai Zovi, Stefan Esser, Vincenzo Iozzo, Ralf-Philipp Weinmann**. *IOS Hacker's Handbook*. John Wiley & Sons, 2012. 408 p. ISBN 1118204123

[63] **Dominic Chell, Tyrone Erasmus, Shaun Colley, Ollie Whitehouse**. *The Mobile Application Hacker's Handbook*. John Wiley & Sons, 2015. 816 p. ISBN 1118958500

[64] **Michael Collins**. *Network Security Through Data Analysis*. O'Reilly, 2014. 348 p. ISBN 1449357903

[65] **Luuk Smit**. *What does your television know about you ?*. University of Twente, Faculty of Electrical Engineering, Mathematics and Computer Science, 2015. 5 p.

[66] **R. Braden**. *RFC1122* [online]. Internet Engineering Task Force [consulted 10 May 2016]. Available at: https://tools.ietf.org/html/rfc1122#page-87

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**