
Exploration and analysis of smartphone Wi-Fi and Bluetooth data

Eirini Sofia Anthi

A thesis presented for the degree of
Bachelor of Science



Cardiff University
Computer Science and Informatics
May 2016

Abstract

In this thesis we examine 96 free mobile applications across 11 categories, in both the Apple *App Store* and Google *Play Store*, to investigate how securely they transmit and handle user data.

For each application, wireless packet sniffing and a series of man-in-the-middle attacks were performed, to try to capture personal or identifying information. Such sensitive information included usernames, passwords, search terms, and location/geo-coordinates data. During the wireless packet sniffing, we monitored the traffic from the device when a specific application was in use, to examine if any sensitive data was transmitted unencrypted. At the same time we revealed and assessed the list of algorithms that each application is using, to establish a secure connection. During the man-in-the-middle attacks, a variety of methods was used in order to try to decrypt the transmitted information. The third party domains to which various applications transmitted sensitive information without the user's permission were also recorded.

The results showed that although all tested applications established a secure connection with the server to transmit data, 85% of them supported weak algorithms to achieve this, which can potentially make the applications vulnerable to attacks. Additionally, 60% of *iOS* and 25% of *Android* applications transmitted unencrypted user data over the Wi-Fi network. Some of this data was also forwarded to third party domains. Finally, the third party domains that received a higher percentage of user data, belonged to Google and Apple.

Acknowledgements

First of all I would like to thank my supervisor, George Theodorakopoulos, for his valuable guidance and support.

Additionally, I would like to thank my family and my boyfriend, for always being there for me and inspiring me to want to become the best I can be.

Finally I would like to further extend my gratitude towards my friends, Shaz and Jason, for their support and friendship the past three years.

Contents

1	Introduction	7
2	Background	9
3	Methodology	10
3.1	Selecting the mobile applications	10
3.2	Using the applications	12
3.3	Wireless packet sniffing	12
3.4	Man-In-The-Middle (MITM) attacks	14
3.4.1	Certificate Validation	15
3.4.2	Man-In-The-Middle attack with <i>Burp Suit</i>	16
3.4.3	Man-In-The-Middle attack with <i>mitmproxy</i>	18
3.4.4	Bypassing Certificate Pinning	19
3.5	Assessing the Cipher Suites	21
3.6	Analysing the captured communications	22
4	Results	23
4.1	Results from the Wireless Packet Sniffing	23
4.1.1	Cipher suites used by <i>iOS</i> applications	23
4.1.2	Cipher suites used by <i>Android</i> Applications	25
4.2	Results from the MITM attack using <i>Burp Suit</i>	26
4.3	Results from the MITM attack using <i>mitmproxy</i>	28
4.4	Results for <i>iOS</i> Applications	28
4.5	<i>Android</i> Applications	33
4.6	Results from the technique used to bypass Certificate Pinning	35
5	Discussion and Evaluation	36
6	Bluetooth	39
7	Future Work	41
8	Conclusion	42
9	Reflection	43
A	Appendix Title	45
A.1	Keywords used throughout the testing.	45
A.2	Cipher Suites Used by iOS applications	48
A.3	Cipher Suites used by Android Applications	49

List of Figures

3.1	Sample of network traffic occurred by the <i>Amazon iOS</i> application. .	12
3.2	Cipher Suit Format	13
3.3	Cipher Suits that <i>Amazon iOS</i> application uses	14
3.4	Man-In-The-Middle Attack	14
3.5	Certificate-based Authentication	15
3.6	Configuring <i>Burp Suit</i>	16
3.7	Configuring the devices to use a proxy.	17
3.8	Captured traffic on <i>Burp Suit</i>	17
3.9	mitmproxy.	18
3.10	Installing custom certificate.	18
3.11	Capturing HTTPS traffic.	19
3.12	iOS SSL Kill on iPhone	20
3.13	Evaluation of cipher suits.	21
4.1	Amount of cipher suites that <i>iOS</i> applications support and how many of these are considered to be weak.	24
4.2	The weak cipher suites are found at the bottom of the list in the <i>ClientHello</i> message.	24
4.3	Amount of cipher suites that <i>Android</i> applications support and how many of these are considered to be weak.	25
4.4	The weak cipher suites are found at the top of the list in the <i>ClientHello</i> message	26
4.5	Applications rejecting self-signed certificate	27
4.6	Warning message on the <i>Android</i> device.	28
4.7	Types of data shared with third parties by <i>iOS</i> applications.	29
4.8	Types of data shared with third parties by <i>iOS</i> applications.	33
6.1	Adafruit BLE sniffer.	39
6.2	Bluetooth traffic from the smartwatch to the phone.	40

List of Tables

- 3.1 List of all tested applications. 11
- 3.2 Types of user data. 22

- 4.1 Sensitive data that we captured for each *iOS* application and the
third party domains that applications forwarded data to. 32
- 4.2 Sensitive data that we captured for each *Android* application and the
third party domains that applications forwarded data to. 35

- A.1 Keywords used throughout the testing. 48
- A.2 Total number of cipher suites used by each application and how many
of these are rated as weak. 49
- A.3 List of all tested applications. 51

Chapter 1

Introduction

In the last decade, the amount of smartphone users has increased dramatically [1]. Smartphones are Internet-enabled devices with an operating system (e.g. iOS, Android, Windows), capable of executing a variety of applications. Most of these devices are also equipped with voice control functionality, a camera, a Wi-Fi antenna, Bluetooth and GPS. Due to their capabilities, smartphone owners not only use their devices to communicate, but they also use them to perform everyday important life activities. Such activities include researching a health condition, accessing education resources, navigating and managing their money [2].

Most of the time users are required to share personal information with the mobile applications they use. However, it is often not clear to smartphone users how exactly these applications handle their personal data. A study by Boyels et al. [3] showed that 54% of smartphone users decided not to install an application once they discovered how much personal information they would need to share. Additionally, 30% of the users uninstalled an application that was already on their mobile phone when they learnt it was collecting personal information they did not wish to share. The same study also showed that users are particularly sensitive about location data, with 19% of the users turning off the location tracking feature on their phone, due to concerns about who could possibly access this information.

The rapid growth of the amount of smartphone users has led to the increase of security threats related to smartphones. According to the ENISA (European Union Agency for Network and Information Security), the number one threat is the leakage of data [4]. This can happen in various ways. When a smartphone gets lost or stolen, its memory or removable media are unprotected, allowing an attacker to access the user's data [4]. Moreover, most of the applications used on a smartphone device will require the user to change their privacy settings in order to allow the application to access sensitive information such as contacts, photographs, etc. Many of these applications have been reported for sharing users personal information with third parties without their consent. A recent study by Zang et al. [5] showed that 73% of Android and 47% of iOS applications shared personal information with third parties, including email addresses and location data. Finally, there is data loss that can occur when a smartphone is connected to Wi-Fi [6].

Even though many smartphone users are aware that the mobile applications they use may share their personal data with third parties, many do not realise how often this happens [7]. Furthermore, a recent survey [8], showed that many users are completely unaware of the risks that come when they share their personal data over

a Wi-Fi connection, and specifically over public Wi-Fi networks. The most severe threat is the unauthorised access to their device which can lead to identity theft and compromised bank accounts [8].

The purpose of this thesis is to examine, in-depth, the data leakage that occurs when users share personal information with various mobile applications over a Wi-Fi connection. Such information includes usernames, passwords, search terms, and location/geo-coordinates data. Additionally, we examine how these applications handle a user's personal information by observing the type of data they share with third parties. Finally, we investigate methods to avoid data leakage. The results of this research will better inform smartphone users as to how mobile applications transmit and handle their data.

We perform tests on both Android and iOS devices, as they have a different operating system and their behaviors as to how they transmit and handle user data differs.

The following chapters are organised as follows: Chapter 3 describes the experiment methodology. Chapter 4 presents the experiment results. Chapter 5 discusses the findings and evaluates the research. Chapter 6 explains the reason why we did not examine the Bluetooth data. Chapter 7 covers the future work and Chapter 8 concludes the thesis. Finally, Chapter 9 contains "The Reflexion".

Chapter 2

Background

Previous studies have mainly focused on investigating the types of sensitive data that various mobile applications share with third parties. The main approach used is dynamic analysis [9].

Dynamic analysis is used to capture mobile application traffic. The only disadvantage of this approach is that requires human intervention, which can limit the scaling of the experiment. There are various methodologies that fall under this approach and have been used successfully in the past.

For instance, Rao et al. [10] used a Virtual Private Network (VPN) to monitor the mobile traffic, involving tools such as *Meddle*. This study showed that a significant amount of Apple *iOS* and Google *Android* applications shared sensitive information such as emails, locations, names, and passwords as plain-text. A different way to observe network traffic is directly from the device. The *Taindroid* application [11] for *Android* platforms allows users to track how private information is obtained and released by mobile applications, in real time. A study by Enck et al. showed that 15 applications sent users location data to third parties and 30 sent the unique phone identifier, phone number, and SIM card serial number. A research study by Zang et al. [5] used a third method to monitor network traffic, during which they performed a man-in-the-middle attack over the Wi-Fi network that the device was connected. They showed that a very large percentage of mobile applications shared personal data with third parties and connected to unknown domains.

Another study which used the same method as [5] was that of Thurm et al. [12]. This study showed that a music *iOS* application shared personal information with eight different domains. Furthermore, the Federal Trade Commission [13], applied the same method, to research the behaviour of 15 fitness applications. The results of this study, showed that 12 of the applications transmitted identifying information to 76 third party domains .

These studies focused on investigating the types of sensitive data that various mobile applications share with third parties. However, how securely these applications transmit this data over Wi-Fi networks had not yet been examined.

In this thesis we build on previous work by testing 96 free applications that require personal information. We investigate how users sensitive information is transmitted and handled, using wireless packet sniffing and dynamic analysis with man-in-the-middle attacks over a Wi-Fi network.

Chapter 3

Methodology

3.1 Selecting the mobile applications

The Google *Play Store* for *Android* and the Apple *App Store* for *iOS* are the two largest distribution channels for mobile applications [14], this is why we chose to examine these two platforms. From a total of 96 applications that were tested, 51 were *iOS* and 45 were *Android*. We looked for the most popular applications as of January/February 2016 that handle sensitive user data, across 10 different categories: Business, Finance, Food and Drink, Games, Health and Fitness, Music, Productivity, Shopping, Social Networking and Travel. The *iOS* applications were tested on an *iPhone 6* and the *Android* applications on a *Motorola Moto e*. The list of the applications that were tested for each platform can be found below in the table 3.1:

Category	Application	iOS	Android
Business	Adobe Reader	✓	✓
	ADP Mobile Solutions	✓	-
	Dropbox	✓	✓
	Facebook Pages	✓	✓
	Indeed Jobs	✓	✓
	Reed.co.uk	✓	✓
	Smart Scan Express	✓	-
Finance	Barclays Mobile Banking	✓	-
	PayPal	✓	-
	Pingit	✓	-
Food and Drink	Burger King	✓	✓
	Domino's Pizza	✓	✓
	Hungry House	✓	✓
	Just Eat	✓	✓
Games	Angry Birds	✓	✓
	Bubble Witch 2	✓	✓
	Candy Crush	-	✓
	Fruit Ninja	✓	-
	Guess the Emoji	✓	✓
	Monsters	-	✓
	Piano Tiles	✓	✓
	Temple Run	✓	✓

Continuation of Table 3.1			
Category	Application	iOS	Android
	Two Dots	✓	-
Health and Fitness	Clue	✓	✓
	iTriage	✓	✓
	Lose it!	✓	✓
	Map My Run	-	✓
	MyFitness Pal	✓	✓
	Period Tracker Lite	✓	✓
	Withings	✓	-
Music	Capitol Fm	✓	✓
	SoundCloud	✓	✓
	Spotify	✓	✓
	Ultimate Guitar	✓	-
Productivity	BlackBoard	✓	✓
	Google Chrome	-	✓
	Safari	✓	-
	Weather	✓	-
Shopping	Amazon	✓	✓
	Ebay	✓	✓
	Groupon	✓	✓
	GumTree	✓	✓
	Wish	✓	✓
Social Networking	Facebook	✓	✓
	Facebook Messenger	✓	✓
	Instagram	✓	✓
	Skype	✓	✓
	Viber	✓	✓
	Whatsapp	✓	✓
Travel	Booking.com	✓	✓
	EasyJet	✓	✓
	Expedia	✓	✓
	Google Earth	✓	✓
	Kayak	✓	✓
	Tripadvisor	-	✓
	Trivago	✓	✓

Table 3.1: List of all tested applications.

3.2 Using the applications

In order to test each application it was necessary to simulate a typical use for 10 to 15 minutes. The time spent on each application varied and exclusively depended on its type. During the simulation, the basic functions of the application were explored. These included: creating a user account, searching using various keywords, performing actions that required personal identifying data, and completing a level of a game. Specific keywords and personal user data that was used during each simulation, was recorded. We then searched for these keywords and personal data in the captured communications. To ensure the integrity of the captured data and to avoid possible interference of other applications, the following measures have been taken: during the testing we made sure that only the tested application was open and no other. For each application all the requested permissions, such as for sharing location data, apart from the push notifications, were allowed. The reason we disabled push notifications is because they keep sending data in the background even when the application is closed [15]. This would result to capturing data not only from the tested application, but also from applications that were tested before and at the time had push notifications on.

3.3 Wireless packet sniffing

To identify if any of the applications transmitted unencrypted data over the Wi-Fi network, we performed wireless packet sniffing using the packet-capturing tool *Wireshark* [16]. During this process we passively monitored the mobile traffic from the smartphone. This free and open source network analyser tool, tries to capture all the network packets that get transmitted and displays their data in as much detail as possible [17]. After configuring *Wireshark* to monitor mobile traffic from the smartphone, we started using an application. We then observed the mobile traffic from the device, as shown in the figure 3.1:

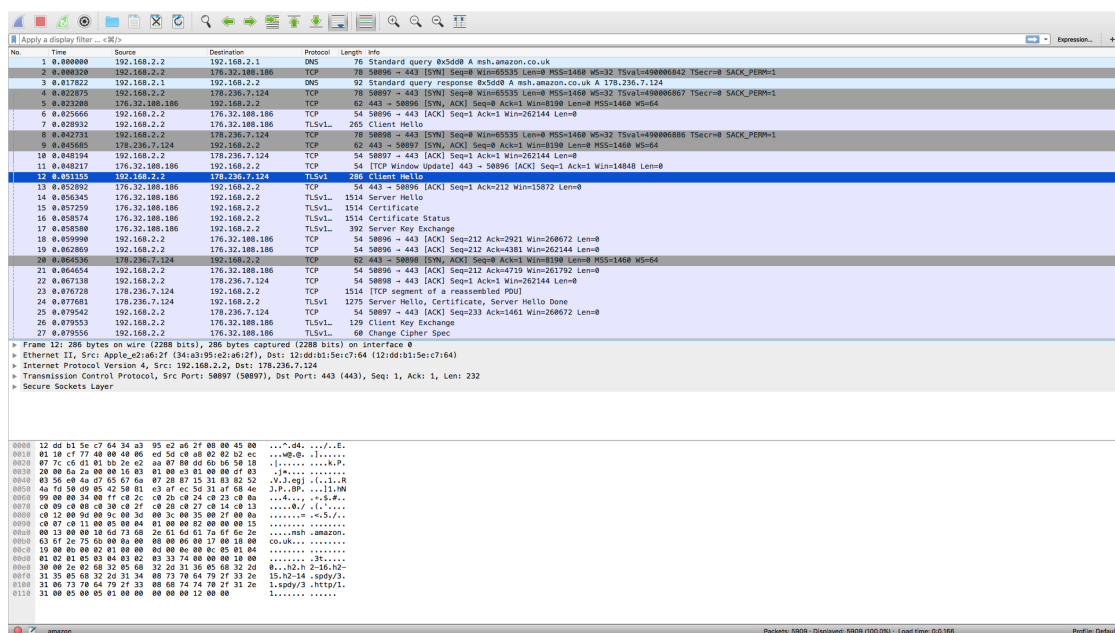


Figure 3.1: Sample of network traffic occurred by the *Amazon iOS* application.

Wireshark's main screen displays the *Internet Protocol (IP)* address of the source and destination device of the transmitted packet, the type of the protocol (e.g. TCP, TLSv1, DNS, etc.), its length, and some information about it. Selecting one of the packets will highlight the row with its details and in the separate window below, more detailed information about this packet will be shown.

For each application we searched all the captured packets for user sensitive data using *Wireshark's* build-in filter functionality. All the intercepted communications were saved for future analysis ¹.

In figure 3.1, we observe that one of the protocols used is the *Transport Layer Security version 1 (TLSv1)*. The TLS and its predecessor *Secure Sockets Layer (SSL)* (we refer to both as SSL), are responsible for establishing a secure channel for communication between the client and the server, which ensures that no third party will eavesdrop or interfere with any of the transmitted messages [18]. Therefore, for any application that employs SSL, we are unable to read or modify any of the transmitted messages. However, the SSL connection can be weakened in a number of ways and hence it is possible to decrypt the transmitted data.

When an SSL connection is established, a handshake known as the *TLS Handshake Protocol* occurs. This handshake, contains the client hello (*ClientHello*) and the server hello (*ServerHello*) messages [18]. The client sends first the *ClientHello* message, which contains a list of supported algorithms (known as cipher suites), in order to establish a secure connection. The server then replies with the *ServerHello* message which contains the selected cipher suit from the client's list [19]. A cipher suit consists of a key exchange algorithm, a signature algorithm, a block cipher algorithm, and a hashing algorithm which computes the authentication key [18]. Usually, it is expressed as a string and has the following format:

```
[SSL/TLS]_[key exchange]_[signature algorithm]_WITH_[block cipher]_[authentication hash]
```

Figure 3.2: Cipher Suit Format

There is a variety of cipher suites available that provide different levels of security. The choice of cipher suites is crucial as they can compromise the security of the communication. It only takes one of the listed cipher suites to be cryptographically insecure, which is enough to break the secure connection between the client and the server and hence intercept the communication. This is possible via the *TLS Protocol Downgrade* attack [20] and it is one of the ways in which the SSL/TLS connection can be weakened.

With *Wireshark* we were able capture the *ClientHello* and *ServerHello* messages, as it is shown in figure 3.1. We were then able to inspect the *ClientHello* message and view its content including the list of the cipher suites the application supported to establish a secure connection with the server, as per figure 3.3. In this specific example, we notice that the tested application contains in its list of cipher suites the TLS_RSA_WITH_RC4_128_MD5, which has known weaknesses as per [21], making the application vulnerable to MITM attacks.

For each application that we tested, we assessed how cryptographically secure are the cipher suites that they use to establish a secure connection with the server, using the method described in section 3.5.

¹These files have been submitted together with the final report.

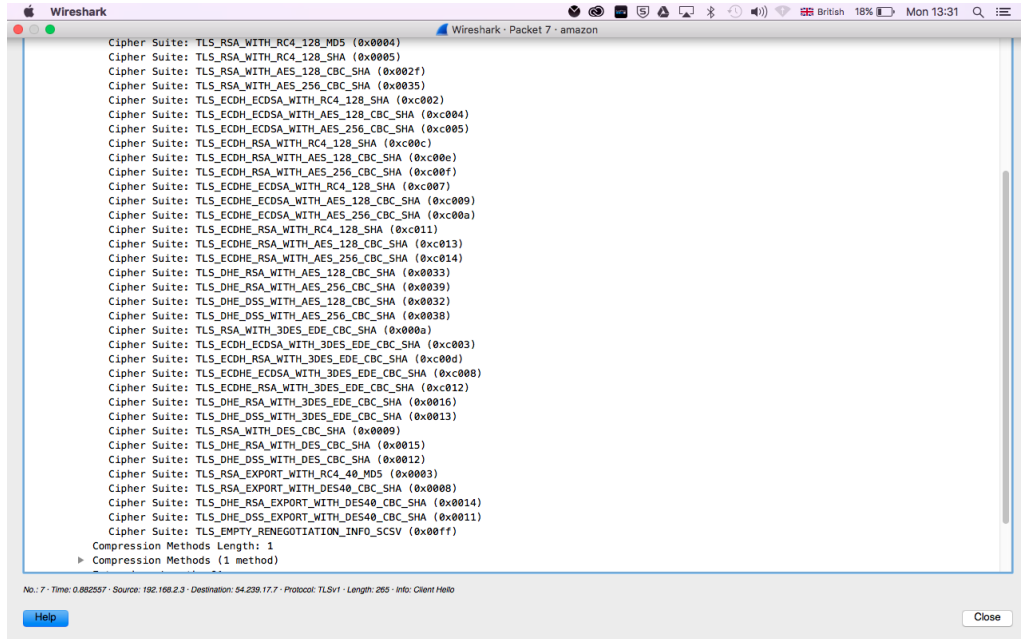


Figure 3.3: Cipher Suites that *Amazon iOS* application uses

3.4 Man-In-The-Middle (MITM) attacks

As mentioned earlier, to examine how various applications transmit and handle user data other than sniffing the packets on the wireless network, we also use dynamic analysis with MITM attacks. The MITM attack is a technique used to intercept the communication between two systems, in this case between the client (application) and the server [19].

During *Hypertext Transfer Protocol (HTTP)* communication, a MITM attack targets the *Transmission Control Protocol (TCP)* layer, which is a protocol that provides reliable, ordered, and error-checked transmission of packets over a network [22]. Throughout a MITM attack the original *TCP* connection gets split into two new ones [19]. One between the client and the attacker and another one between the attacker and the server as shown in the figure 3.4. When the original TCP connection is finally compromised, the attacker is able to act as a proxy and therefore read, insert, and modify data in the intercepted communication [19].

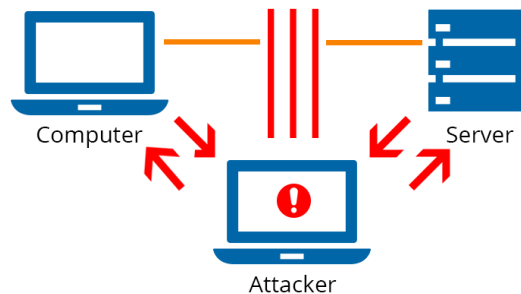


Figure 3.4: Man-In-The-Middle Attack

There are many tools that can be used to help perform such an attack. Specifically, in this project MITM attacks were performed, using the tools *Burp Suit* [23]

and *mitmproxy* [24]. These tools also helped identify HTTP-based traffic only. We note that a recent study by Raora et. al [10] showed that TCP flows (HTTP/HTTPS) are responsible for over 90% of the total traffic volume. In order to perform the attacks described above, we were required to setup a Wi-Fi hot-spot on the computer that ran these tools and then connect the smartphone device to the Internet, via this hot-spot.

3.4.1 Certificate Validation

SSL is built on the fundamental concept of certificate-based authentication, which ensures that the parties involved in the communication, are in fact who they claim to be [25]. This is the main mechanism that helps avoid man-in-the-middle attacks, by preventing the use of fake public keys and impersonation [26]. Certificate-based authentication, is achieved by employing digital certificates. These are special electronic documents that can identify anyone on the Internet, from an individual to a company, and are associated with a public key [26]. They contain information about their owner's identity and also include the digital signature of an entity that has verified that the certificate details are correct [27].

Digital certificates are issued and signed by independent third parties or organisations, that run their own certificate-issuing server, and are able to validate identities, known as Certificate Authorities (CA). Of course, not every CA issued certificate is trustworthy, this is why every client or server that supports certificates, preserves a list of all the trusted CA certificates and therefore CAs [26].

The certificate-based authentication process consists of five main steps as shown in the figure 3.5 [26]:

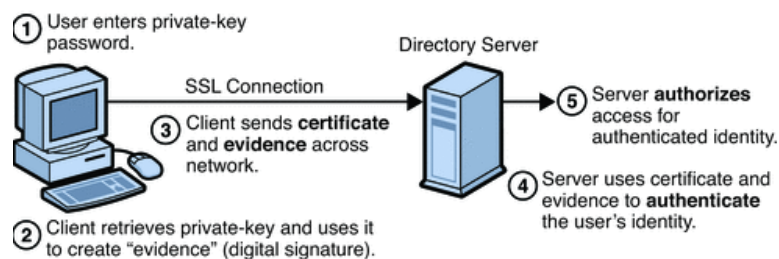


Figure 3.5: Certificate-based Authentication

In order for a server to validate the identity of a client and grant them access, the client is required to digitally sign a randomly generated piece of data (evidence) and send it to the server, together with its digital certificate. The server assesses the client's evidence and the digital certificate and based on these, it authenticates the user's identity.

From all the above, it is made clear that the validation mechanism of digital certificates is crucial and any weakening in this process can lead to severe security issues, making user data vulnerable to MITM attacks. The most dangerous thing that a client or server can do, that could lead to the above issue, is accepting certificates that are not signed from a trusted CAs (self-signed certificates) [25].

In this project we tested the mobile applications using the method described in section 3.4.2, to identify if they do accept self-signed certificates and therefore if they are vulnerable to eavesdropping and tampering attacks.

3.4.2 Man-In-The-Middle attack with *Burp Suit*

To examine if an application is accepting self-signed certificates, it was necessary to configure the smartphone to use a proxy, in this case *Burp Suit*, which generated and presented to the client a self-signed certificate. We then monitored the behaviour of the application in use and observed if it functioned as expected. Additionally, we checked if we were able to capture any HTTPS traffic on the proxy software. The steps of the procedure are described below [25]:

1. We ensured that the smartphone did not have any existing custom proxy certificates in its trust store.
2. On the computer, we disabled the firewall and started *Burp Suit* proxy. It was necessary to configure it to listen to all external network interfaces by specifying the port (8080) and address (All interfaces) as shown in figure 3.6:

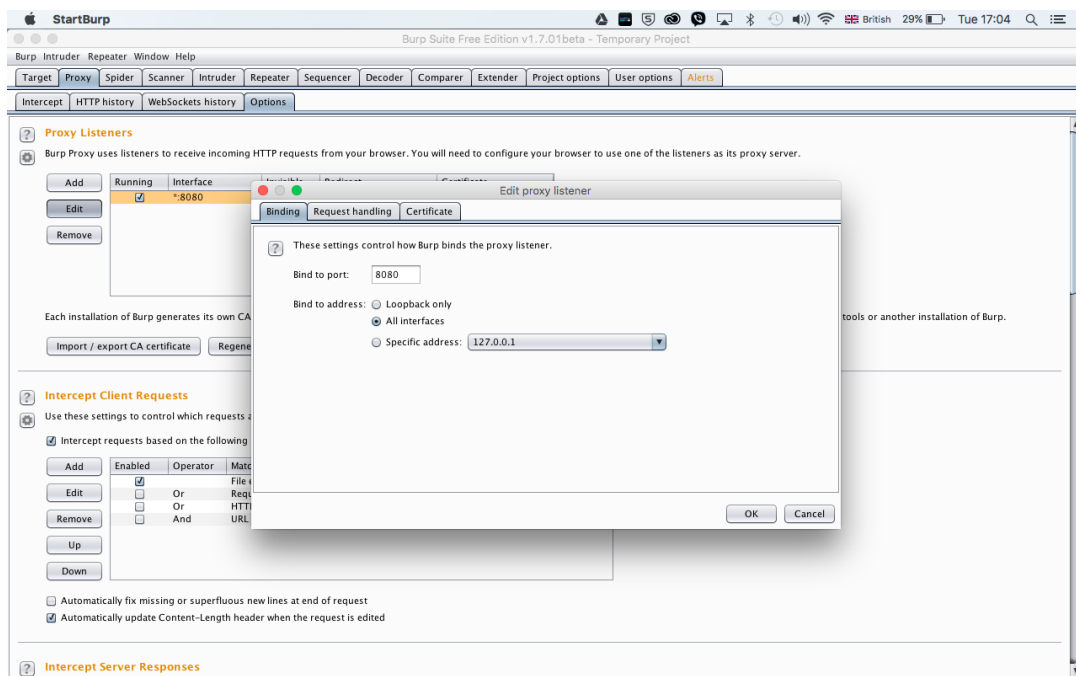
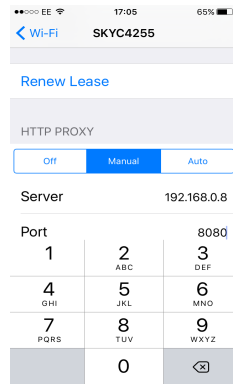
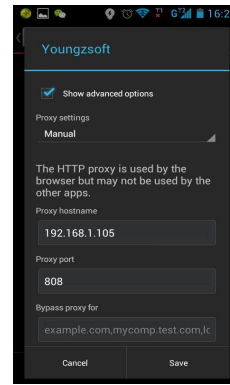


Figure 3.6: Configuring *Burp Suit*.

3. Then we configured the smartphone device to use the proxy. (Settings, Wi-Fi, we chose the desired Wi-Fi network, selected HTTP Proxy Manual). The IP address and port of the proxy were the same to the computer in use, as per 3.7:



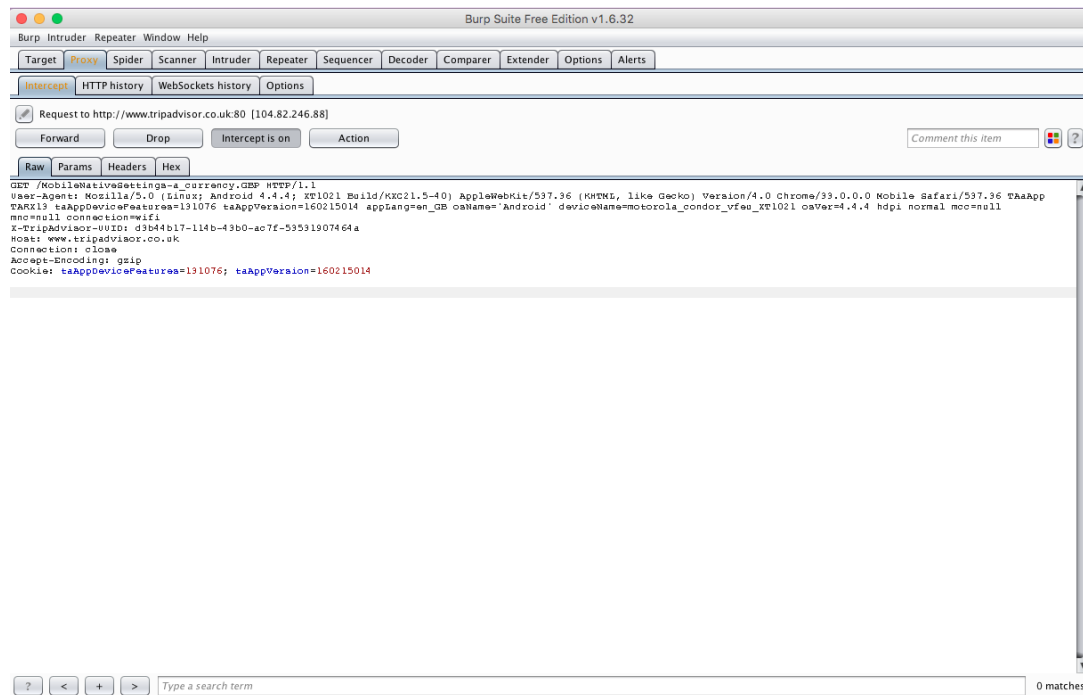
(a) iOS device.



(b) Android device.

Figure 3.7: Configuring the devices to use a proxy.

4. Finally we launched the application we wanted to test and simulated a typical use, while we monitored the proxy to detect if any HTTPS data was being intercepted, as per 3.8

**Figure 3.8:** Captured traffic on *Burp Suit*.

If *Burp Suit* was able to intercept HTTPS traffic from the device without having to install the proxy's certificate on the device's trust store, we know that the application did indeed accept self-signed certificates and was vulnerable to eavesdropping and modification via MITM attacks [25].

3.4.3 Man-In-The-Middle attack with *mitmproxy*

On applications that did not accept self-signed certificates, we were not able to capture the encrypted traffic that occurred from the device using the previous method. In order to overcome this, we performed a MITM attack using *mitmproxy*, as per figure 3.9 [28].

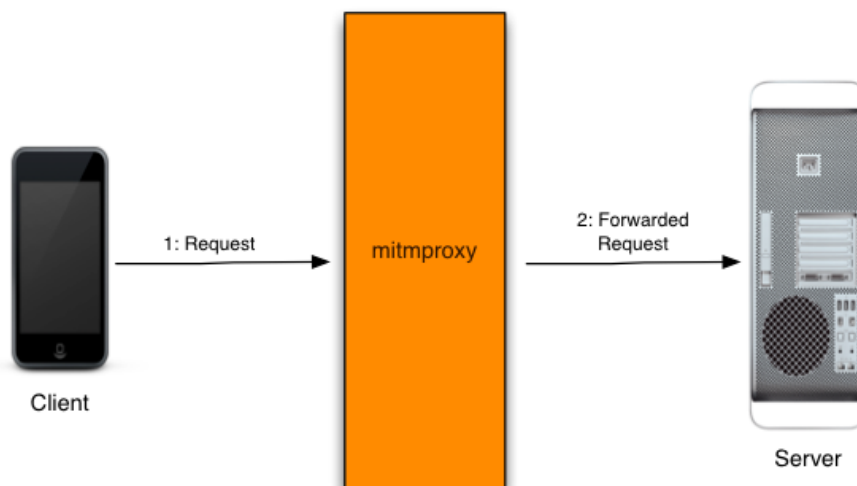


Figure 3.9: mitmproxy.

Once again, we configured the smartphone to use the proxy, however this time, we installed the proxy's certificate in the device's trust store. *mitmproxy* contains a certificate authority implementation and is able to generate digital certificates [28]. Furthermore, to make the client (device) trust certificates issued by *mitmproxy*, we registered it manually on the device as a trusted CA. It is necessary to underline that this method will only work if the application does not employ certificate pinning [29]. More details about this mechanism and how to bypass it will be discussed in section 3.4.4.

To intercept traffic with the *mitmproxy* we followed the steps below [30]:

1. To begin with we started *mitmproxy* and configured the device to use it, by setting the correct proxy details (port and IP address).
2. We then opened the browser on the smartphone and visited `www.mitm.it`. On the screen we were able to view this:

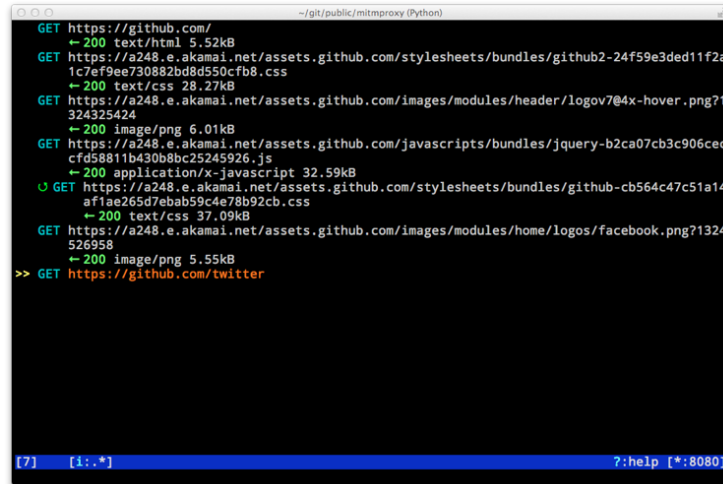


Figure 3.10: Installing custom certificate.

3. We selected the relevant icon and followed the further instructions, as to how to install the proxy's certificate in the device's trust store. When the installation

was done, we opened an application and started observing the *mitmproxy*'s screen for HTTPS traffic.

In the *mitmproxy*'s main screen, we were able to view the mobile traffic that occurred when an application was in use, as per figure 3.11:



```

~/.git/public/mitmproxy (Python)
GET https://github.com/
  ← 200 text/html 5.52kB
GET https://a248.e.akamai.net/assets.github.com/stylesheets/bundles/github2-24f59e3ded11f2a
1c7ef9ee730882bd8d550cfb8.css
  ← 200 text/css 28.27kB
GET https://a248.e.akamai.net/assets.github.com/images/modules/header/logov7@4x-hover.png?1
324325424
  ← 200 image/png 6.01kB
GET https://a248.e.akamai.net/assets.github.com/javascripts/bundles/jquery-b2ca07cb3c906cec
cfd58811b430b8bc25245926.js
  ← 200 application/x-javascript 32.59kB
GET https://a248.e.akamai.net/assets.github.com/stylesheets/bundles/github-cb564c47c51a14
af1ae265d7ebab59c4e78b92cb.css
  ← 200 text/css 37.09kB
GET https://a248.e.akamai.net/assets.github.com/images/modules/home/logos/facebook.png?1324
526958
  ← 200 image/png 5.55kB
>> GET https://github.com/twitter

```

Figure 3.11: Capturing HTTPS traffic.

mitmproxy displays the full flow summary, including the methods used and the full *Uniform Resource Identifiers (URIs)* of the HTTP/HTTPS requests. By selecting one of the requests, the software allows us to inspect and manipulate it [28]. If the application hadn't used any encryption method on the data, we were able to view it as plain-text. Therefore, this method helped us identify if the applications transmitted unencrypted information over the network and examine if they further send any of it to unknown third parties.

To analyse further the captured data, we exported it to a text file and used a *Python* script to help us search this file for any user sensitive data. The exact method is discussed in section 3.6.

3.4.4 Bypassing Certificate Pinning

Certificate pinning is a technique used widely on mobile applications to prevent the possibility of the device's trust store being compromised, by manually installing unverified certificates [29]. Specifically, this technique pins the certificate that the server uses in the application's source code, making it ignore the device's trust store. As a result it will only establish a connection to hosts signed with certificates that are pinned in the application's source code. To applications that employed this mechanism, we used *iOS SSL Kill Switch* to attempt to bypass it.

This process was only applied to *iOS* applications, as it requires *jailbreaking/-rooting* [31] the tested device. The *Android* device used in this project belongs to the University, therefore we weren't allowed to root it. *Jailbreaking* the *iPhone 6*, allowed us to remove all the software restrictions of *Apple's* operating system, and granted us access to the *iOS* file system and manager. As a result we were able to download extra items that are unavailable on the official *Apple App Store* [31].

After *jailbreaking* the *iPhone 6* following the instructions on [32], we gained access to *Cydia*, the unofficial *iOS App Store*. From there we downloaded and

installed a tool called *iOS SSL Kill Switch*, as described on [33]. This tool, disables the certificate validation process on the device, leaving it exposed to MITM attacks. After we successfully installed this tool we were able to see it within the *System Settings*, as shown in the figure 3.12:

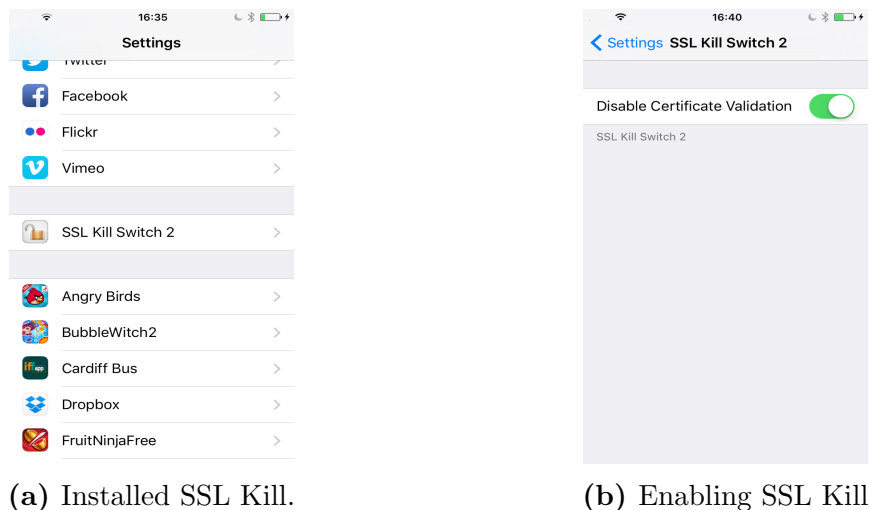


Figure 3.12: iOS SSL Kill on iPhone

Having installed and enabled *iOS SSL Kill Switch*, we used *mitmproxy*, following the method described in the previous section 3.4.3, to observe if we could capture any HTTPS traffic.

medium, and high. Immediately after, is displayed a break down of the each cipher, which explains the algorithms they contain and their key lengths in further detail.

3.6 Analysing the captured communications

To analyse the captured data, we wrote a *Python* script³. This helped us search the saved data in the text files, for any personal user data that might have been transmitted in plain-text. Specifically, the data we looked for included: Personal Identifying Information (PII) such as names and passwords, search terms, and geo-coordinate data, including longitude and latitude values. In the table 3.2, we present all the types of user data that the script looked for in the text files. The complete list of the keywords that were used throughout the simulations and therefore we looked to find in the captured data, can be found in the Appendix A.1. Moreover, in our *Python* script we included regular expressions, in order to identify all the URIs of the requests that the application performed POST requests to. This way we were able to discover if any of the applications transmitted personal user data to unknown domains.

Categories of data	Data types
Behavior	Employment (Job Searches)
	Medical
	Private Messaging (chats, texts, etc.)
	Searching
Location	Latitude
	Longitude
PII	Address
	Age
	Date Of Birth
	Device Information (e.g. Device ID)
	Email Address
	Gender
	Name
	Password
	Post Code
	Telephone Number
	Username

Table 3.2: Types of user data.

In order to ensure that our results were reliable, every time that the script found an occurrence of a keyword within a text file, we manually inspected the findings to confirm that they are correct and identify any further information. For instance, if the script found a match for the “1990”, we manually examined the result to ensure that the finding is indeed the user’s year of birth and not a part of some other information such as long integer. This process was also necessary in order to discover the destination domain, of the data that was transmitted and identified as plain-text.

³This was submitted together with the report

Chapter 4

Results

4.1 Results from the Wireless Packet Sniffing

All the tested mobile applications for both *iOS* and *Android* platforms employed the latest SSL protocol, to establish a secure channel for communication. As a result, although we were able to capture the transmitted data, it was not possible for us to read it because it was encrypted. The only case in which we had the opportunity to capture transmitted data in plain-text, was when we tested the mobile browsers, *Safari* on the *iPhone* and *Google Chrome* on the *Motorola*, and performed requests that did not require a secure connection.

4.1.1 Cipher suites used by *iOS* applications

We examined and assessed the cipher suites in 51 *iOS* applications, 45 of which were found that use the same set of 26 cipher suites. From these 26 suites, 4 are considered to be weak and shouldn't be used. Only 6 of the tested applications used less than 26 suites and didn't support any weak suites. A visual representation of these results is displayed in figure 4.1. A detailed table showing how many cipher suites each application uses and how many of these are considered to be weak, can be found in appendix A.2. The 4 insecure cipher suites that the applications used are: TLS_ECDHE_ECDSA_WITH_RC4_128_SHA, TLS_ECDHE_RSA_WITH_RC4_128_SHA, TLS_RSA_WITH_RC4_128_SHA, and TLS_RSA_WITH_RC4_128_MD5. In the *ClientHello* message, for all *iOS* applications, we observed that these 4 suites were found to be at the bottom of the list, as per figure 4.2. The order in which the suites appear in the *ClientHello* message, denotes the client's preferred suites (with the client's first preference first). Therefore in this case, the four weak cipher suites are the least preferred suites by the client and are unlikely to be used to establish a secure connection [35]. Nevertheless, even in this situation these cipher suites should not be used as a *TLS Downgrade Attack* [20] could be used against them.

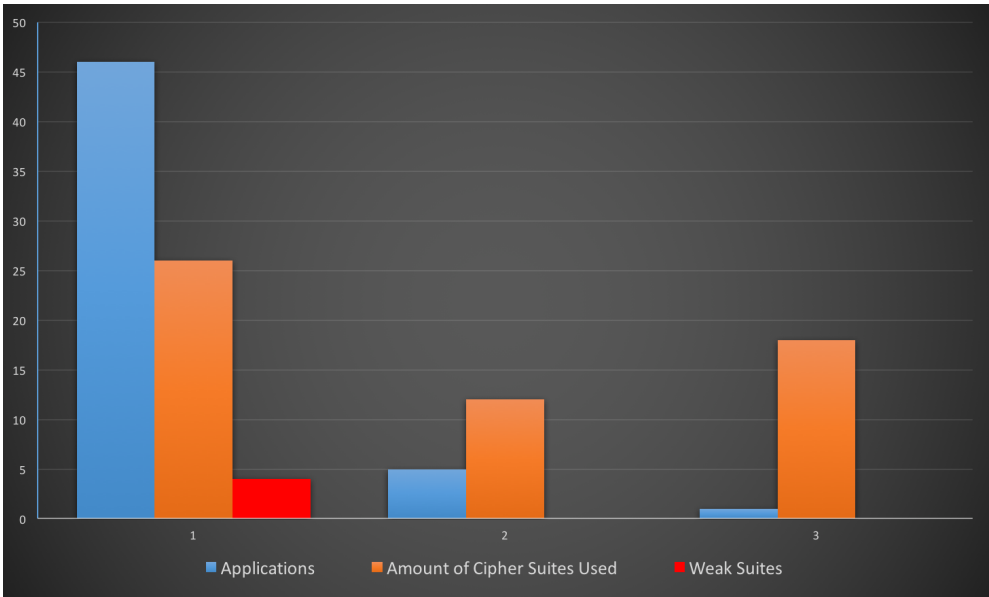


Figure 4.1: Amount of cipher suites that *iOS* applications support and how many of these are considered to be weak.

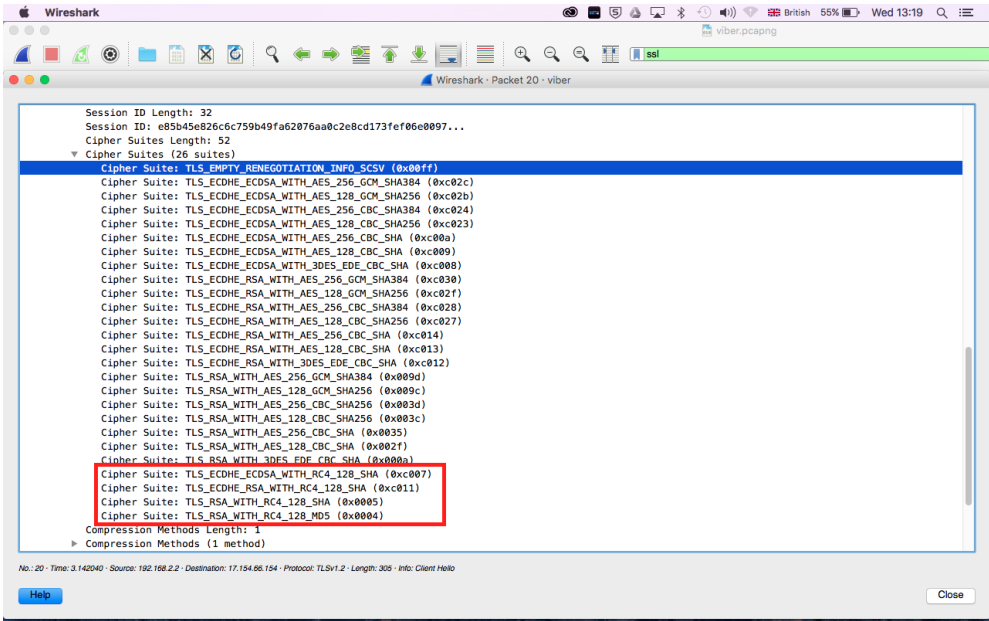


Figure 4.2: The weak cipher suites are found at the bottom of the list in the *ClientHello* message.

4.1.2 Cipher suites used by *Android* Applications

From the 45 *Android* applications that we tested, 27 used the same set of 35 cipher suites, of which 4 are considered insecure. 11 of the applications used less than 35 cipher suites and from these only 6 did not support any insecure suites. 3 of the applications used more than 35 suites and only 1 was found to not support weak cipher suites. Finally, we weren't able to capture the *ClientHello* message for 4 applications and as a result it wasn't possible to examine the cipher suites they use. A visual representation of these results is displayed in figure 4.3. A detailed table showing how many cipher suites each application uses and how many of these are considered to be weak, can be found in appendix A.3.

The insecure cipher suites supported by the *Android* applications are exactly the same ones that were found in *iOS* applications. These suites include the: TLS_ECDHE_ECDSA_WITH_RC4_128_SHA, TLS_ECDHE_RSA_WITH_RC4_128_SHA, TLS_RSA_WITH_RC4_128_SHA, and TLS_RSA_WITH_RC4_128_MD5. In the *ClientHello* message these suites were found to be at the top of the list, which shows that these are the client's most preferred suites, as per figure 4.4. In case the server accepts the client's preferences (the server is free to ignore the client's order and can pick the cipher suit that thinks it is best [35]) a connection will be established using one of these insecure suites, making the application vulnerable to MITM attacks. Conclusively, we can say that 38 out of the 45 tested applications are vulnerable to MITM attacks.

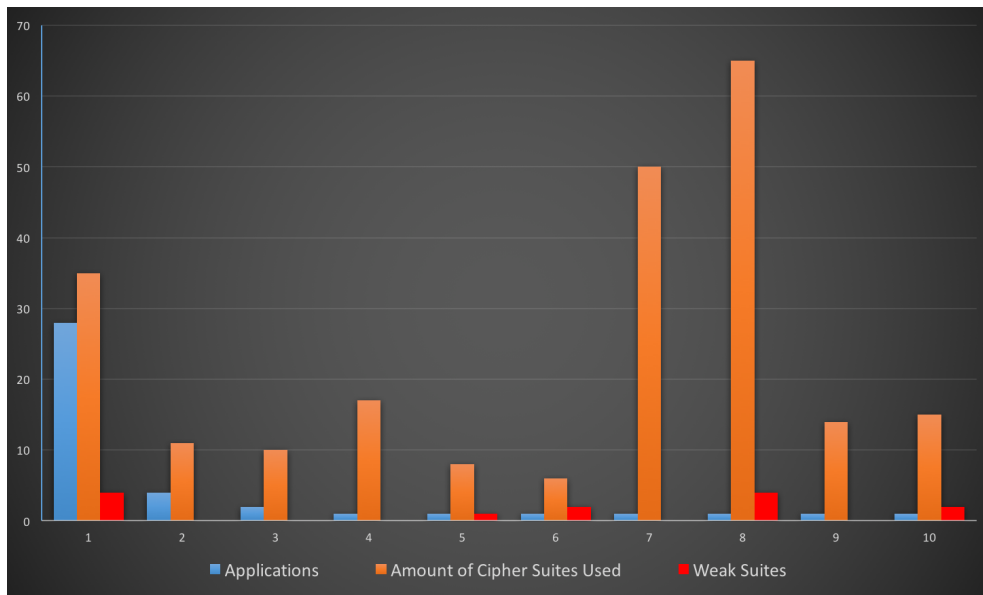


Figure 4.3: Amount of cipher suites that *Android* applications support and how many of these are considered to be weak.

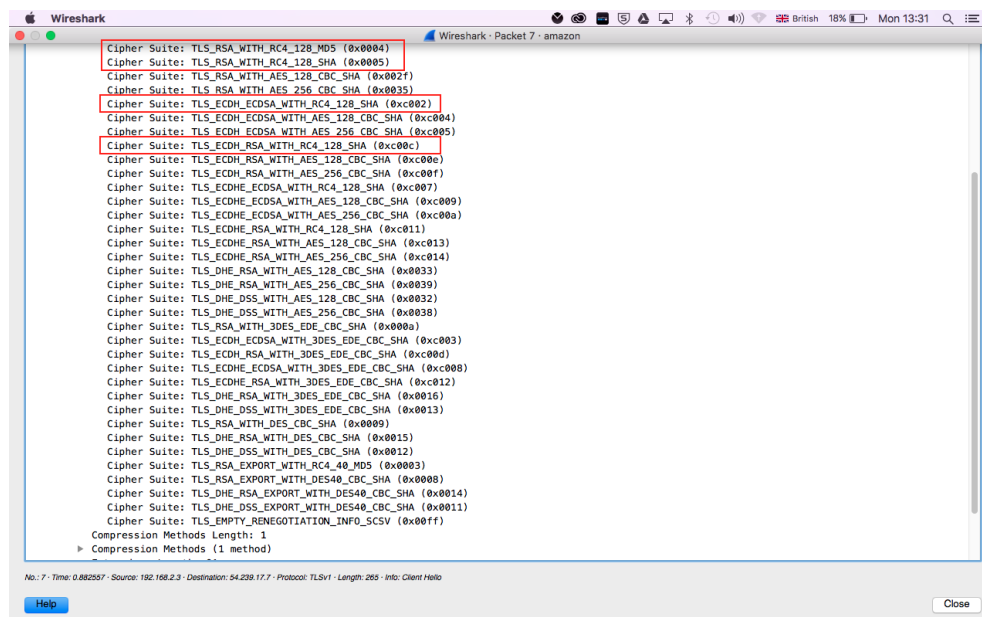


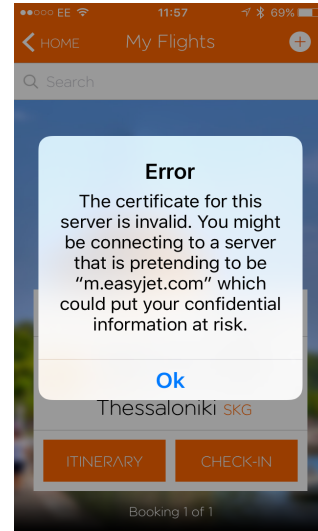
Figure 4.4: The weak cipher suites are found at the top of the list in the *ClientHello* message

4.2 Results from the MITM attack using *Burp Suit*

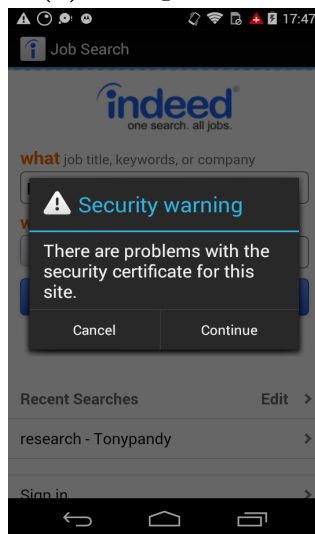
We performed a MITM attack using *Burp Suit* to 51 *iOS* and 45 *Android* applications, in order to check if they would accept self-signed certificates that were not installed in the device's trust store. We found that none of the applications for both platforms accepted the unverified certificate, prompting us with a message as shown in figure 4.5. As a result, we were not able to capture any of the HTTPS traffic that occurred during the simulation of a typical use for each application.



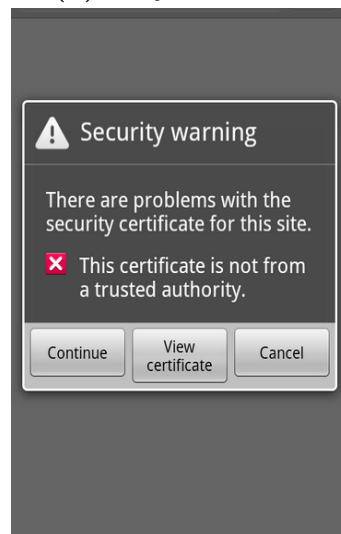
(a) Trivago on *iOS*



(b) EasyJet on *iOS*



(c) Indeed on *Android*



(d) Google Chrome on *Android*

Figure 4.5: Applications rejecting self-signed certificate

4.3 Results from the MITM attack using *mitmproxy*

In order to perform this MITM attack we had to install the certificate that *mitmproxy* generated in the trust store of each device. After we completed this procedure, we observed that the *Android* device displayed a warning message as per figure 4.6, to inform us that an unauthenticated certificate is currently being used. On the contrary, on the *iOS* device we did not get any warnings about the fake certificate. Nevertheless, at this point we were able to capture HTTPS traffic from both devices, hence we started testing the applications, the results of which are presented in the following sections.

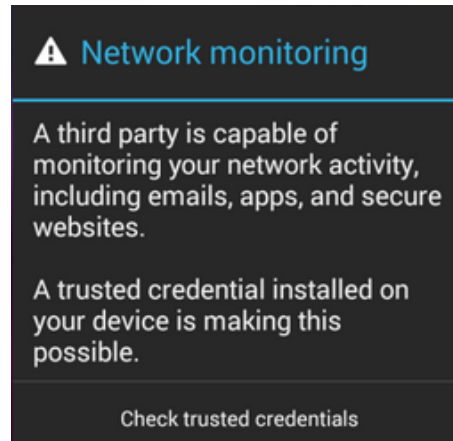


Figure 4.6: Warning message on the *Android* device.

4.4 Results for iOS Applications

We performed a MITM attack using *mitmproxy* to 51 *iOS* applications to investigate if any sensitive user data gets transmitted unencrypted over the Wi-Fi network and also to examine if any of these applications sent sensitive data to third party domains. From the 51 applications we found that 30 transmitted the data unencrypted over the network, of which 20 forward it to third party domains. 8 of the applications used encryption on the actual user data, therefore although we captured the transmitted data, we were unable to read it. 12 applications used certificate pinning and did not function at all, claiming that there is a problem with the network. The table 4.1, shows the sensitive data that we captured for each application and the domains that each one was forwarded to. In the table we marked applications that employed certificate pinning with an xmark and used the abbreviation form of non applicable (n/a) where data was not forwarded to any third party domains.

The Burger King, Indeed Jobs, Lose it!, and Ebay applications transmitted the most unencrypted user data, which included: usernames, passwords, emails, location, gender, and search terms. Additionally we managed to capture usernames and passwords for Spotify, Blackboard, Instagram and EasyJet. The applications that forwarded the most data to third party domains was Indeed Jobs and Burger King. Gaming applications mainly transmitted and shared information about the device such as: phone model, screen size, etc. Moreover, the third party

domains that received the most sensitive user data were `googleanalytics.com`, `googleservices.com`, and `apple.com`. Figure 4.7, shows the types of data that the 20 *iOS* applications shared with third parties.

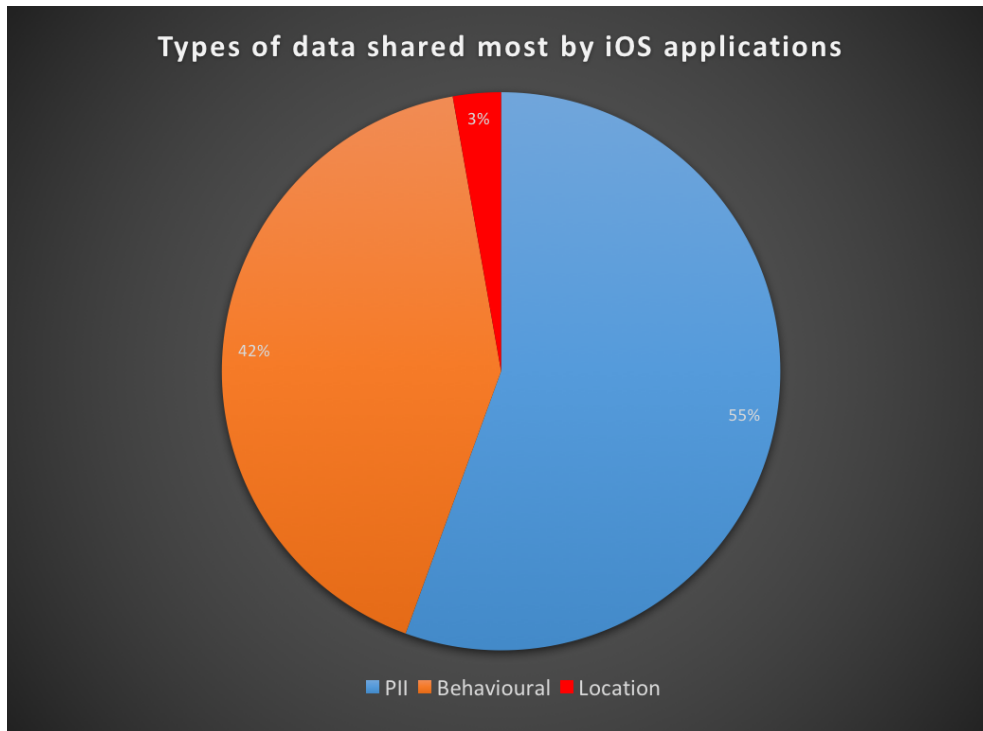


Figure 4.7: Types of data shared with third parties by *iOS* applications.

The fact that we were able to capture the username, password, and email for Instagram, EasyJet, Blackboard, Ebay, and Spotify, made us think that this might actually be a security issue. If an unauthorised person logged into these applications using these credentials, they would have access to much more sensitive information such as PayPal, bank accounts, home address, passport details, etc. Therefore we decided to report our observations to each of the application’s development teams, using the *Responsible Disclosure*¹ procedure. Facebook (for Instagram), Spotify, and Blackboard replied to us thanking us for reporting this issue, confirming that it is indeed a security flaw.

Category	Application	Transmitted data that was unencrypted	Shared with 3rd party domains
Business	Adobe	none	n/a
	ADP Mobile Solutions	none	n/a
	Facebook Pages	✗	✗
	Dropox	✗	✗
	Indeed Jobs	password	n/a
		email	googleadservices.com

¹This procedure involves privately notifying affected software vendors of vulnerabilities. The vendors then typically address the vulnerability at some later date, and the researcher reveals full details publicly at or after this time [36].

Continuation of Table 4.1			
Category	Application	Transmitted data that was unencrypted	Shared with 3rd party domains
		search terms	googleanalytics.com
	Reed	none	n/a
	Smart Scan Express	none	n/a
Finance	Barclays Mobile Banking	✗	✗
	Paypal	✗	✗
	Pingit	✗	✗
Food and Drink	Burger King	username	n/a
		email	googleapis.com googleanalytics.com facebook.com
		search terms	googleanalytics.com
		password	n/a
		telephone	n/a
		post code	n/a
		location	n/a
	Domino's Pizza	device info	crashlitics.com apple.com
	Hungry House	device info	apple.com
	Just Eat	location	stats.ge
Games	Angry Birds	device info	rovio.com toons.tv apple.com
	Bubble Witch	device info	adtrack.com
	Fruit Ninja	device info	apple.com facebook.com amazon.com
	Guess the Emoji	device info	apple.com google.com googleads.com twitter.com
	Piano Tiles	device info	apple.com googleads.com
	Temple Run	device info	apple.com
	Two Dots	device info	apple.com

Continuation of Table 4.1			
Category	Application	Transmitted data that was unencrypted	Shared with 3rd party domains
Health and Fitness	Clue	none	n/a
	iTriage	search terms	googleads.com
	Lose it!	gender email username device info	n/a
	Period Tracker	none	n/a
	MyFitness Pal	name	googleads.com
		username	n/a
	Withings	location	n/a
Music	Capitol Fm	email device info	iech.ch youtube.com
	Soundcloud	device info	n/a
	Spotify	username	n/a
		password	
	Ultimate Guitar	search terms	n/a
Productivity	Blackboard	username password	n/a
	Safari	none	n/a
	Weather	none	none
	Safari	✗	✗
Shopping	Amazon	search terms	n/a
	Ebay	email username password location	n/a
	Gumtree	username search terms	googleads.com
	Wish	gender date of birth	yahoo.com
Social Network	Facebook	✗	✗
	Facebook Messenger	✗	✗
	Instagram	username password	n/a
	Skype	✗	✗

Continuation of Table 4.1			
Category	Application	Transmitted data that was unencrypted	Shared with 3rd party domains
	Viber	none	n/a
	Whatsapp	✗	✗
Travel	Booking.com	email search terms	googleads.com
	EasyJet	username password	twitter.com
	Expedia	search terms	apple.com
	Google Earth	none	none
	Kayak	✗	✗
	Trivago	✗	✗

Table 4.1: Sensitive data that we captured for each *iOS* application and the third party domains that applications forwarded data to.

4.5 *Android* Applications

From the 45 applications that we examined, 11 transmitted data unencrypted over the Wi-Fi network. 9 applications used encryption on the actual user data so although we were able to capture the traffic we were not able to read it. Furthermore, 25 applications employed certificate pinning and did not function throughout this process. The table 4.2, shows the transmitted sensitive data that we captured for each *Android* application and also the third party domains to which it was sent.

Ebay, Gumtree, and Booking.com, were the only applications that transmitted unencrypted usernames and passwords. Domino's Pizza, Gumtree, and Booking.com shared with third parties all the terms that were searched for in the application, location data was only shared by Just Eat and gaming applications mainly transmitted and shared device information. The third party domains that received the most user sensitive data were `googleads.com` and `apple.com`. Figure 4.8, shows the types of data that the 11 *Android* applications shared with third parties.

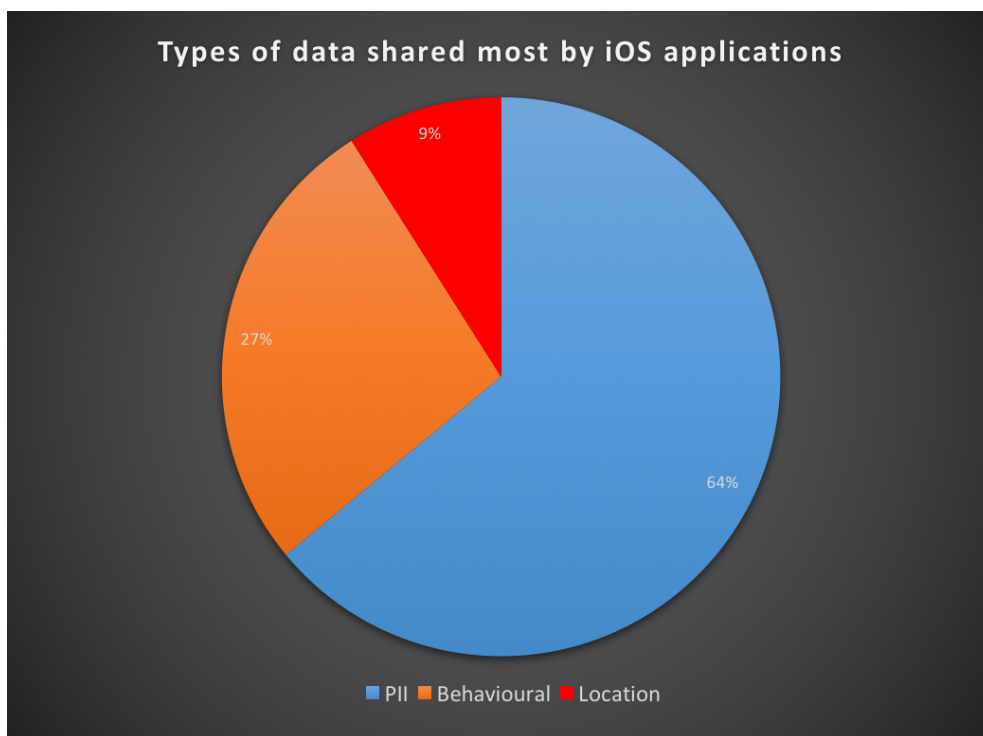


Figure 4.8: Types of data shared with third parties by *iOS* applications.

Category	Application	Transmitted data that was unencrypted	Sent to 3rd party domains
Business	Adobe	✗	✗
	Facebook Pages	✗	✗
	Dropox	✗	✗
	Indeed Jobs	none	n/a
	Reed	none	n/a

Continuation of Table 4.2			
Category	Application	Transmitted data that was unencrypted	Sent to 3rd party domains
Food	Burger King	none	n/a
	Domino's Pizza	search terms	googleads.com
	Hungry House	device info	apple.com
	Just Eat	location	stats.ge
Games	Angry Birds	device info	rovio.com cloudads.net googleads.com
	Bubble Witch	device info	adtrack.com
	Guess the Emoji	device info	apple.com google.com googleads.com twitter.com
	Don't tap the white tile	device info	apple.com googleads.com
	Temple Run	device info	apple.com
	Two Dots	device info	apple.com
Health & Fitness	Clue	none	none
	iTriage	✗	✗
	Lose it!	✗	✗
	Period Tracker	✗	✗
	MyFitness Pal	✗	✗
Music	Capitol Fm	✗	✗
	Soundcloud	✗	✗
	Spotify	✗	✗
Prod.	Blackboard	✗	✗
	Google Chrome	✗	✗
Shopping	Amazon	search terms	n/a
	Ebay	email username password location	n/a
	Gumtree	username	n/a
		search terms	googleads.com
	Wish	none	n/a

Continuation of Table 4.2			
Category	Application	Transmitted data that was unencrypted	Sent to 3rd party domains
Social Networking	Facebook	X	X
	Facebook Messenger	X	X
	Instagram	X	X
	Skype	X	X
	Viber	X	X
	Whatsapp	X	X
Travel	Booking.com	email	n/a
		search terms	googleads.com
	EasyJet	X	X
	Expedia	X	X
	Google Earth	X	X
	Kayak	X	X
	Tripadvisor	X	X
	Trivago	X	X

Table 4.2: Sensitive data that we captured for each *Android* application and the third party domains that applications forwarded data to.

4.6 Results from the technique used to bypass Certificate Pinning

We *jailbroke* the iPhone 6 and used *SSL Kill Switch*, on the *iOS* applications that employed certificate pinning, in order to investigate if it is possible to bypass this mechanism and capture the transmitted data. We found that this tool was effective on 75% of the applications. As a result we managed to capture the traffic that occurred while we were testing them. The other 25% was able to detect that the device was *jailbroken* and did not operate. Additionally noticed that none of these applications encrypt the users data in order to transmit it.

Chapter 5

Discussion and Evaluation

We performed wireless packet sniffing to investigate if any of the applications transmitted data unencrypted over the Wi-Fi network. Our results showed that all the applications for both *iOS* and *Android* platforms used SSL protocol to establish a secure channel for communication with the server. This protocol is fairly employed by developers, as it provides protection against passive eavesdropping [24]. Anyone performing wireless packet sniffing over the network will be able to capture the traffic, but they won't be able to read it as it is encrypted. SSL may provide privacy and data integrity between a client and a server, however it can be weakened and the cipher suites that applications use to establish this connection have a great role in this. We examined all the cipher suites that applications support in order to establish a secure connection, and we found that the majority of them for both platforms and specifically 90% of the *iOS* and 80% of the *Android* applications, supported four insecure cipher suites. These suites were the same for both operating systems and included the: TLS_ECDHE_ECDSA_WITH_RC4_128_SHA, TLS_ECDHE_RSA_WITH_RC4_128_SHA, TLS_RSA_WITH_RC4_128_SHA, and TLS_RSA_WITH_RC4_128_MD5.

These cipher suites are considered to be weak mainly because they use the RC4 stream cipher. Even though RC4 is widely supported and preferred by most servers, it has been known to have a variety of cryptographic weaknesses, making it unable to provide a sufficient level of security [21, 37]. For this reason, according to the Internet Engineering Task Force (IETF), the RC4 algorithm is prohibited and clients must not include RC4 ciphers in their *ClientHello* message. Additionally, the MD5 hash algorithm is also known to have cryptographic weaknesses and ciphers that employ it should not be used [18, 38]. A few of the reasons that applications support these suites although they are considered to be insecure and have been prohibited, include: that are compatible with most servers, have simple design, and are fast due to the reduced amount of operations they need to perform [39]. Nevertheless, 85% of all the tested *iOS* and *Android* applications that support these suites, even though they use SSL, are considered to potentially be vulnerable to MITM attacks.

We also tested the applications in order to investigate if they accept self-signed certificates. We found that none of the applications, for both *iOS* and *Android*, accepted the self-signed certificate that *Burp Suit* proxy generated. This is an indication that accepting self-signed certificates is indeed a severe security issue that developers are aware of, making the certificate validation processes as robust as possible [25].

Using *mitmproxy* we established that approximately 60% of the *iOS* and 25% of

the *Android* applications, transmitted and forwarded sensitive unencrypted data to third party domains. The most common data that was forwarded by applications to third party domains was Personal Identifying Information (PII) and Behavioural including: device information, email, name and search terms. For both platforms, gaming applications mainly transmitted and forwarded information about the device. A reason why PII and behavioural types of data are shared with third parties, could be that this information is used by these organisations to develop targeted advertising [40]. The percentage of *Android* applications that shared user data with third party domains seemed to be significantly less than the percentage of the *iOS* applications. This was due to the fact that 20% of *Android* applications encrypted the actual user data and 56% employed certificate pinning. On the other hand only 15% of the *iOS* applications encrypted the user data and only 23% employed certificate pinning. Therefore for the applications that encrypted the data and used certificate pinning we were unable to investigate if they shared sensitive information with third parties.

Comparing our results with a recent study by Zang et al.[5], which also investigated data sharing by applications, we can observe some differences. In the previous study, more applications shared location and other sensitive user data and very few employed certificate pinning. On the contrary our results showed that fewer applications shared location and other sensitive user data with third parties. Additionally, the amount of applications that used certificate pinning, specifically for *Android* applications has increased dramatically. The overall increase in applications employing certificate pinning may be because without it, data can be intercepted by installing fake certificates in the device's trust store [29]. Additionally penetration testing recently performed on various mobile applications [5, 41] could also explain why more of them started using certificate pinning. The fact that significantly more *Android* applications employ certificate pinning compared to *iOS*, is because certificate pinning was one of the many security enhancements introduced in the new firmware version, Android 4.2 [29].

The domains to which applications from both platforms sent the most user sensitive data were: `googleanalytics.com`, `googleservices.com`, `googleads.com`, and `apple.com`. Previous studies [5, 10] have also found these domains to be dominant. This may be due to Google and Apple owing a variety of mobile advertisement networks and services such as AdMob, Google Analytics, Double Click and iAds [42, 43].

Finally we used *SSL Kill Switch* on the *jailbroken* iPhone, in order to attempt to bypass certificate pinning on applications that employed it, and we successfully managed to do so on 75% of the applications. The finance applications (Barclays, PayPal, Pingit) detected that the device was *jailbroken* and did not operate. Conclusively, *jailbreaking* or *rooting* the smartphone comes introduces security issues and unless the applications are designed to not operate in such a device, the users data is in danger of being stolen.

Overall the methods we chose to evaluate how securely mobile applications transmitted and handled user data over a Wi-Fi network were effective, but had limitations. To begin with, all the methods we used required human intervention which limited significantly the amount of applications that we were able to test. The MITM attacks we performed to both platforms, although they were able to provide us with valuable information about the applications certificate validation process and data

sharing behaviour, they required physical access to the device in order to install fake certificates. Therefore even though we were able to intercept any transmitted sensitive data, these methods would be very difficult to apply in real life. Additionally, the tools we used to perform these attacks focused only on HTTP/HTTPS traffic, limiting the scope of the research. The *SSL Kill Switch*, allowed us to successfully bypass the certificate pinning mechanism however we were required to *jailbreak* the iPhone. This was a very time consuming and insecure process. To analyse the captured data we wrote a Python script to searched for sensitive data in the captured communications text files. The script was very effective in analysing our data, however if these files were larger in size, Python would run very slow and would not be the most appropriate language to use to implement it.

Chapter 6

Bluetooth

Part of the objectives of this project was to investigate also the data leakage that can happen to data that gets transmitted via the smartphone's Bluetooth known as Bluetooth Low Energy (BLE). We would try to capture the transmitted data between a smartphone and a Pebble smartwatch and try to decrypt it. Sensitive data such as text messages, reminders, emails, social media notifications, etc. get transmitted to the smartwatch via bluetooth. In order to explore this issue we needed to use an external USB Bluetooth sniffer such as *Ubertooth* [44]. Unfortunately, we did not have access to it and we could not conduct the experiment properly. Consequently, this is something that a future research could focus on investigating. Nevertheless, we decided to use a low cost Bluetooth sniffer for the purposes of this project, the Adafruit BLE sniffer, as shown in figure 6.1.

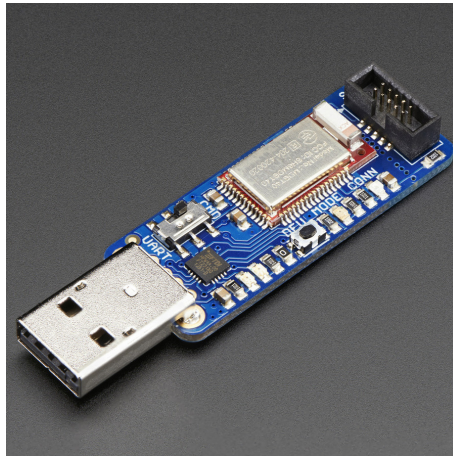


Figure 6.1: Adafruit BLE sniffer.

A recent study by Ryan M. [45] proved that the encryption used by Bluetooth Low Energy (BLE) can be broken. In a Bluetooth connection, there are always two parties that communicate with each other. The master, which is the central device (e.g. smartphone) and a slave (e.g. smartwatch, Bluetooth speaker, etc.). These two parties in order to establish a secure channel for communication they are required to use a key exchange protocol. This key exchange protocol is not based on any well-known and robust key exchange protocols such as Diffie Hellman (DH) [46], but it was invented by Bluetooth SIG and is known to have significant weaknesses [47]. The attack performed by in [45] targets the key exchange protocol rather than the encryption itself.

The procedure during which a master and a slave establish a secure connection is called pairing and has three different modes [48]: 1) Just Works, 2) 6-digit pin, and 3) OOB: a 128 bit value exchanged out-of-band. Due to weakness of the key exchange protocol, if the master and slave use the Just Work or 6-digit pin modes to pair, all the required values of the elements needed to decrypt the communication apart from one (which has a value from 0-999,999) are known. Therefore using a simple brute force attack the last value can be calculated and the communication can be decrypted [45].

A passive eavesdropper could actually capture the key exchange process and the encrypted traffic between the two devices. Then a tool can be used to analyse the key exchange process and find the value of the unknown element required to break the encryption [45]. An open source tool that can be used to achieve this is *Crackle* [49].

After many unsuccessful attempts, trying to configure this low cost sniffer to listen to the BLE traffic from the smartphone device, Adafruit engineers informed us that this sniffer can only listen to peripheral devices and not central ones. As a result it was possible for us to capture only the traffic that occurred from the smartwatch to the smartphone. Although we managed to do so as per figure 6.2, we were unable to capture the key exchange procedure and therefore we could not continue any further.

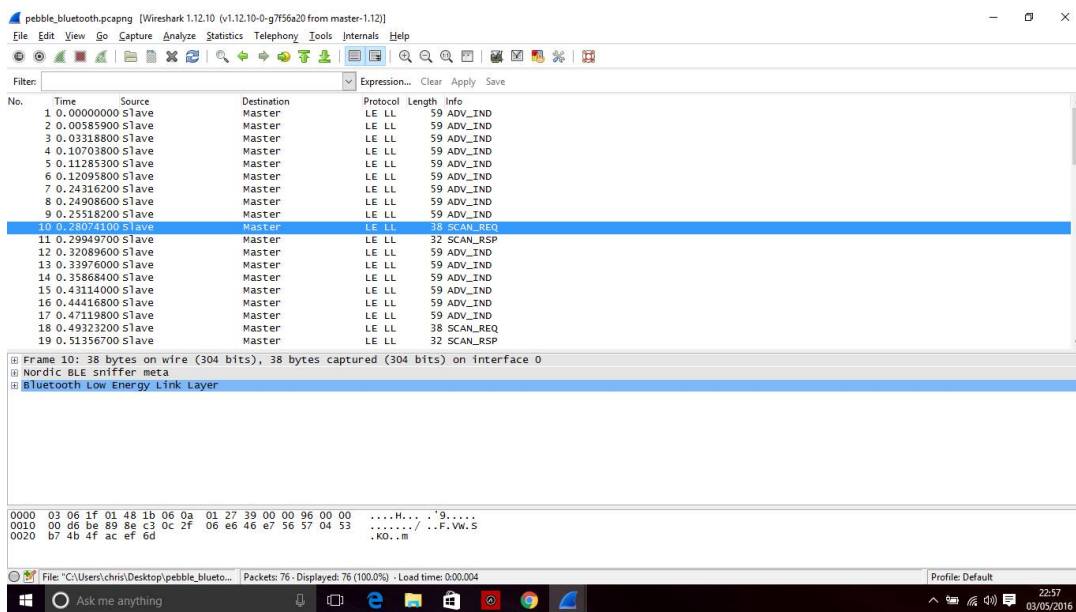


Figure 6.2: Bluetooth traffic from the smartwatch to the phone.

From left to right we can see the number of the transmitted packet, then the time, the source which is the slave (Pebble smartwatch), the destination which is the master (iPhone), the protocol used, the size of the transmitted packet and finally some information about the type of packet.

Chapter 7

Future Work

To expand on the results of this research, future study could focus on testing more applications from each category, for both operating systems. Non-TCP traffic could also be investigated for sensitive data leakage using *tcpdump*, which monitors traffic that is not on the TCP. To the applications that supported weak cipher suites *TLS Downgrade Attack* could be performed, to explore if the SSL can indeed be compromised this way. In this project we managed to apply tools to bypass certificate pinning only to *iOS* devices. Future studies could also *root* an *Android* device and then use *Android-SSL-TrustKiller* [50] to try to bypass certificate pinning in this operating system as well. Furthermore, tools that track the data sharing behaviour of applications directly from the smartphone device such as *Taindroid* could be used to monitor both the operating system and the application. As a result it would be possible to clearly distinguish any leakage that happens due to the application's activity and the background system processes [5, 11].

Additionally, paid applications could also be tested for data leakage. The results could then be compared with the ones from the free applications in order to review any difference in the data sharing behaviour. Tools that limit the data sharing such as *Limit ad Tracking* and *Opt out of interest based ads* can be employed to examine any differences in the activity of the applications. Finally, regarding the data that gets transmitted via Bluetooth Low Energy (BLE), future research could use *Ubertooth* BLE sniffer [44] to capture the traffic between smartphones and smart devices (e.g. smartwatches) and try to decrypt it using *Crackle*.

Chapter 8

Conclusion

The study was set out to explore and analyse how user data gets transmitted and handled by various mobile applications. We selected 51 *iOS* and 45 *Android* mobile applications and carried out 4 different experiments, while we simulated a typical use for each application. The results showed that all applications use **SSL** protocol to establish a secure channel for communication with the server, which protects data from passive eavesdropping, specifically when transmitted over public networks. However, this does not mean that user data is secure, as our findings showed that a very small percentage of these applications encrypted the actual user data and approximately 85% of these applications supported 4 weak cipher suites which make them vulnerable to MITM attacks. Moreover, our results showed that 60% of the *iOS* and 15% of *Android* applications forwarded sensitive user data, mostly PII and Behavioural, to third party domains mainly owned by Google and Apple. Background research regarding Bluetooth Low Energy indicated that there are severe security issues with its encryption model. However due to lack of necessary equipment, we were unable to conduct an experiment and research it any further.

Although our research methodology had its limitations, we still managed to arrive to significant conclusions as to how securely user data gets transmitted and handled by various applications, over a Wi-Fi network. Additionally, two of the methods we used, were designed in order to break or bypass the basic encryption mechanisms that developers have employed, such as **SSL** and certificate pinning. This is proof that these security measures are not invulnerable. As a result, users need to become fully aware that their personal information can never be 100% secure and the only way to protect their privacy, is to understand these security risks.

Our hope is that this project will educate its readers, making them aware of the data leakage that occurs when applications transmit and handle their data. Therefore, we aspire that they will be more cautious when sharing sensitive data with mobile applications and at the same time we hope that they will always remember to take measures to protect their privacy when using public networks.

Chapter 9

Reflection

For my thesis I had decided to pursue a research based project in order to improve my research skills and gain valuable experience as a researcher. The main reasons why I chose this project were the topic, of which I found to be current and interesting. Taking into consideration that I did not have any previous experience in the field of mobile security, I started reading on and experimenting with mobile and network security before I began working on my thesis.

The initial project goals were to investigate the data leakage that can happen when data is transmitted from smartphones (for *iOS* and *Android* device) over a Wi-Fi network and Bluetooth, but also the leakage that happens when applications share user data with third parties. In order to be able to select the appropriate methodology to achieve this, I was required to study the fundamentals of data transmission procedure over a network and Bluetooth, the protocols that are used in each case, and their encryption mechanisms. After I had a clear understanding of these, I was able to research which methods/experiments were the most suitable to fulfil my goals. Having in mind the experiments I would carry out and having selected the mobile applications and peripheral device to test the data leakage over a network and Bluetooth respectively, I had to set a time plan to help me organise myself and ensure that I would complete the project on time. Creating my own time plan and having to evaluate the approximate time that I would need to complete each sub part of this project was very difficult, however I did manage to follow it as much as possible. As it is natural, I feel like I underestimated the time that some procedures take, such as data collection and analysis and I did fall back slightly on these.

Managing and representing results from my data analysis, was another challenging part of this project. Sometimes I felt that the way I chose to represent my findings could have been clearer. After many attempts trying to organise them in different ways, I believe I have managed to represent them as simply and clearly as possible.

As the sample of applications that I chose to test was quite large, I had planned to carry out only one experiment, in order to evaluate if I could capture any sensitive unencrypted data over Wi-Fi and also to enable me to investigate if applications forward any user data to third parties. Nevertheless, although this experiment required a lot of time and a complex method to analyse the results, I felt like this was not enough and I had to push myself harder. As a result I decided to carry out three more experiments to investigate this issue and to have a more spherical opinion

about it. I took this decision almost one and a half months prior the submission date of the dissertation, which was very risky because I had very little time and I had to re-examine 96 applications in three more experiments. I had to work very hard in order to complete all the testing and data analysis, however I feel as though my risk paid off.

Throughout my investigations I found that for some major mobile applications I was able to capture the user's credentials and get access to very sensitive data. I thought that this did not seem right and I reported this issue to these companies. When I received a message from these companies thanking me and confirming that this was indeed an issue I felt very happy, surprised and satisfied because my final year project contributed to making these applications safer.

Regarding the Bluetooth data although I did not have access to the necessary hardware I was determined to find a way to examine this issue. I purchased a low cost Bluetooth sniffer and I used it to try to capture the traffic from the smartphone to a smartwatch. Even though the company that produces this sniffer claims that it can capture the Bluetooth traffic from a smartphone I found that this was not the case. I was very disappointed at this point. I didn't give up and I researched every tutorial I could find. Finally, I contacted the engineers within this company and they confirmed that this device can only capture BLE data from peripheral devices. Even though I felt disappointment, at this point I felt also relieved because there was nothing else I could have done.

Overall I feel satisfied with my performance on this project, with what I managed to achieve, the knowledge I gained and not giving up regarding the Bluetooth aspect of the project. I know that I have tried my absolute best to fulfil the goals of this project and this makes me feel very happy. It was an experience through which I learnt so many things about mobile security, I experienced (on a small scale) how it feels to carry out research and it also helped me improve my organisational/research skills.

In the future, with the experience I have now, I would make sure that to allocate more time to data collection and analysis, better plan the methodology and ensure that it is sufficient, ensure from the beginning that I have access to all the necessary equipment that I need to carry out the experiments, and I would certainly organise the collected data better so that it is easier to represent afterwards. Additionally, I feel as though I have to learn to control my emotions as they can effect my performance.

Appendix A

Appendix Title

A.1 Keywords used throughout the testing.

Category	Type	Term Searched
Behavior	Employment	analyst
Behavior	Employment	assistant
Behavior	Employment	chef
Behavior	Employment	developer
Behavior	Employment	education
Behavior	Employment	fulltime
Behavior	Employment	full-time
Behavior	Employment	graduate
Behavior	Employment	IT
Behavior	Employment	research
Behavior	Employment	security
Behavior	Employment	teacher
Behavior	Employment	£21000
Behavior	Medical	chest pain
Behavior	Medical	cough
Behavior	Medical	fever
Behavior	Medical	headache
Behavior	Medical	medication
Behavior	Medical	mycrogynon
Behavior	Medical	pneumonia
Behavior	Medical	sinusitis
Behavior	Private Messaging	ciao
Behavior	Private Messaging	cinema at nine
Behavior	Private Messaging	hello

Continuation of Table A.1		
Category	Type	Term Searched
Behavior	Private Messaging	hey
Behavior	Private Messaging	holla
Behavior	Private Messaging	how are you?
Behavior	Private Messaging	meet me at seven
Behavior	Searching	beer
Behavior	Searching	boat cruise
Behavior	Searching	cavalieri hotel
Behavior	Searching	fish
Behavior	Searching	game of thrones
Behavior	Searching	indian
Behavior	Searching	kickboxing
Behavior	Searching	laptop
Behavior	Searching	mani club
Behavior	Searching	nintendo
Behavior	Searching	pancacke accessories
Behavior	Searching	rocksmith
Behavior	Searching	weights
Location	Latitude	51.5
Location	Longitude	-3.0
Location	Latitude	latitude
Location	Longitude	longitude
PII	Address	athens
PII	Address	cardiff
PII	Address	corfu
PII	Address	newport
PII	Address	risca
PII	Address	thessaloniki
PII	Address	united kingdom
PII	Address	
PII	Age	23
PII	Age	27
PII	DOB	23/07/1962
PII	DOB	23-07-1990
PII	DOB	17/09/1990
PII	DOB	17-09-1990

Continuation of Table A.1		
Category	Type	Term Searched
PII	DOB	July 62
PII	DOB	1962
PII	DOB	Sept 90
PII	DOB	1990
PII	Device Info	iphone
PII	Device Info	motorola
PII	Device Info	MEID: 89*****
PII	Device Info	MEID: 67*****
PII	Email	irini@yahoo.gr
PII	Email	irinianthi90@gmail.com
PII	Email	chris-2@live.co.uk
PII	Email	c1417801@gmail.com
PII	Gender	Female
PII	Gender	female
PII	Name	chris northfield
PII	Name	irene anthi
PII	Name	nenitsa tsoukala
PII	Password	*****
PII	Password	*****
PII	Password	*****
PII	Password	*****
PII	Password	*****
PII	Password	*****
PII	Post Code	np108fl
PII	Post Code	np10 8fl
PII	Telephone Number	07745971980
PII	Telephone Number	00447745971980
PII	Telephone Number	077-459-71980
PII	Telephone Number	077-459-71980
PII	Username	chrisnorthfield
PII	Username	ireneanth
PII	Username	ireneanthi
PII	Username	irinaki90
PII	Username	irini90
PII	Username	lina

Continuation of Table A.1		
Category	Type	Term Searched
PII	Username	ninoula
Location	Latitude	51.5
Location	Longitude	-3.0
Location	Latitude	latitude
Location	Longitude	longitude

Table A.1: Keywords used throughout the testing.

A.2 Cipher Suites Used by iOS applications

Category	Application	Total Ciphers	Weak Ciphers
Business	Adobe Reader	12	0
	ADP Mobile Solutions	26	4
	Dropbox	26	4
	Facebook Pages	26	4
	Indeed Jobs	26	4
	Reed.co.uk	26	4
	Smart Scan Express	26	4
Finance	Barclays Mobile Banking	26	4
	PayPal	26	4
	Pingit	26	4
Food and Drink	Burger King	26	4
	Domino's Pizza	26	4
	Hungry House	26	4
	Just Eat	26	4
Games	Angry Birds	26	4
	Bubble Witch 2	26	4
	Fruit Ninja	26	4
	Guess the Emoji	26	4
	Piano Tiles	26	4
	Temple Run	26	4
	Two Dots	26	4
Health and Fitness	Clue	18	0
	iTriage	26	4
	Lose it!	26	4
	Period Tracker Lite	26	4

Continuation of Table A.2			
Category	Application	Total Ciphers	Weak Ciphers
	MyFitness Pal	26	4
	Withings	24	0
Music	Capitol Fm	26	4
	SoundCloud	12	0
	Spotify	12	0
	Ultimate Guitar	26	4
Productivity	BlackBoard	26	4
	Safari	-	-
	Weather	26	4
Shopping	Amazon	26	4
	Ebay	26	4
	Groupon	26	4
	GumTree	26	4
	Wish	26	4
Social Networking	Facebook	26	4
	Facebook Messenger	26	4
	Instagram	12	0
	Skype	26	4
	Viber	26	4
	Whatsapp	26	4
Travel	Booking.com	26	4
	EasyJet	26	4
	Expedia	26	4
	Google Earth	26	4
	Kayak	26	4
	Trivago	26	4

Table A.2: Total number of cipher suites used by each application and how many of these are rated as weak.

A.3 Cipher Suites used by Android Applications

Category	Application	Total Ciphers	Weak Ciphers
Business	Adobe Reader	35	4
	Dropbox	8	1
	Facebook Pages	35	4

Continuation of Table A.3			
Category	Application	Total Ciphers	Weak Ciphers
	Indeed Jobs	35	4
	Reed.co.uk	6	2
Food and Drink	Burger King	17	0
	Domino's Pizza	35	4
	Hungry House	35	4
	Just Eat	35	4
Games	Angry Birds	50	0
	Bubble Witch 2	-	-
	Candy Crush	65	4
	Guess the Emoji	35	4
	Piano Tile	35	4
	Monsters	35	4
	Temple Run	-	-
Health and Fitness	Clue	11	0
	iTriage	35	4
	Lose it!	-	-
	Map My Run	11	0
	MyFitness Pal	11	0
	Period Tracker Lite	35	4
Music	Capitol Fm	35	4
	SoundCloud	35	4
	Spotify	10	0
Productivity	BlackBoard	35	4
	Google Chrome	-	-
Shopping	Amazon	35	4
	Ebay	53	4
	Groupon	35	4
	GumTree	35	4
	Wish	35	4
Social Networking	Facebook	35	4
	Facebook Messenger	35	4
	Instagram	14	0
	Skype	35	4
	Viber	11	0
	Whatsapp	35	4

Continuation of Table A.3			
Category	Application	Total Ciphers	Weak Ciphers
Travel	Booking.com	35	4
	EasyJet	15	2
	Expedia	35	4
	Google Earth	35	4
	Kayak	35	4
	Tripadvisor	10	0
	Trivago	35	4

Table A.3: List of all tested applications.

Bibliography

- [1] Statista. *Number of smartphone users worldwide from 2014 to 2019*. <http://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/>. Accessed: 1/04/2016. 2016.
- [2] Aaron Smith. “US smartphone use in 2015”. In: *Pew Research Center* (2015). Accessed: 1/04/2016, pp. 18–29.
- [3] Jan Lauren Boyles, Aaron Smith, and Mary Madden. “Privacy and data management on mobile devices”. In: *Pew Internet & American Life Project 4* (2012).
- [4] ENISA. *Top Ten Smartphone Risks*. <https://www.enisa.europa.eu/activities/Resilience-and-CIIP/critical-applications/smartphone-security-1/top-ten-risks>. Accessed: 4/04/2016. 2016.
- [5] Jinyan Zang, Krysta Dummit, James Graves, Paul Lisker, and Latanya Sweeney. *Who Knows What About Me? A Survey of Behind the Scenes Personal Data Sharing to Third Parties by Mobile Apps*. <http://techscience.org/a/2015103001/>. Accessed: 14/02/2016. 2015.
- [6] Michael Cooney. *10 Common Mobile Security Problems to Attack*. <http://www.pcworld.com/article/2010278/10-common-mobile-security-problems-to-attack.html>. Accessed: 4/04/2016. 2012.
- [7] Carnegie Mellon University. *Knowledge of location sharing by apps prompts privacy action*. <https://www.sciencedaily.com/releases/2015/03/150323132846.html>. Accessed: 4/04/2016. 2015.
- [8] Statista. *The Hidden Dangers of Public WiFi*. http://www.privatewifi.com/wp-content/uploads/2015/01/PWF_whitepaper_v6.pdf. Accessed: 5/04/2016. 2016.
- [9] Thomas Ball. “The concept of dynamic analysis”. In: *Software Engineering—ESEC/FSE’99*. Springer. 1999, pp. 216–234.
- [10] Ashwin Raoa et al. “Using the Middle to Meddle with Mobile”. In: (2013).
- [11] Appanalysis. *Realtime Privacy Monitoring on Smartphones*. <http://www.appanalysis.org/index.html>. Accessed: 9/04/2016. 2016.
- [12] Scott Thurm and Yukari Iwatani Kane. “Your apps are watching you”. In: *The Wall Street Journal* 17 (2010), p. 1.
- [13] ENISA. *Federal Trade Commission*. <https://www.ftc.gov/search/site/fitness%20app>. Accessed: 9/04/2016. 2016.
- [14] H Victor. “Android’s Google Play beats App Store with over 1 million apps, now officially largest”. In: *Retrieved January 16* (2013), p. 2014.

- [15] Mark A Fox, Peter F King, and Seetharaman Ramasubramani. *Method and apparatus for maintaining security in a push server*. US Patent 6,421,781. July 2002.
- [16] Angela Orebaugh, Gilbert Ramirez, and Jay Beale. *Wireshark & Ethereal network protocol analyzer toolkit*. Syngress, 2006.
- [17] Wireshark. *How does it work?* <https://www.wireshark.org/#learnWS/>. Accessed: 18/04/2016. 2016.
- [18] OWASP. *Transport Layer Protection Cheat Sheet*. https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet. Accessed: 18/04/2016. 2016.
- [19] OWASP. *Man-in-the-middle attack*. https://www.owasp.org/index.php/Man-in-the-middle_attack/. Accessed: 18/04/2016. 2016.
- [20] Bodo Moeller and Adam Langley. “TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks”. In: (2015).
- [21] Internet Engineering Task Force (IETF). *RFC 7465 - Prohibiting RC4 Cipher Suites*. <https://tools.ietf.org/html/rfc7465#section-1>. Accessed on 25/04/2016.
- [22] Christensson P. *TCP/IP Definition*. <http://techterms.com/definition/tcpip>. Accessed: 18/04/2016. 2006.
- [23] Dafydd Stuttard. *Burp Suite, 2007*.
- [24] Dan Boneh, Srinivas Inguva, and Ian Baker. “SSL, MITM Proxy”. In: *http://crypto.stanford.edu/ssl-mitm* (2007).
- [25] Ollie Whitehouse Tyrone Erasmus Shaun Colley. *The Mobile Application Hacker's Handbook*. John Wiley & Sons; 1 edition (3 April 2015), 2015.
- [26] Oracle Corporation. *Certificates and Certificate Authorities (CA)*. <https://docs.oracle.com/cd/E19316-01/820-2765/gdzen/index.html/>. Accessed: 19/04/2016. 2016.
- [27] OpenSSL. *Certificate Authorities (CA) and Digital Signatures*. <https://www.openssl.org/docs/manmaster/apps/ca.html/>. Accessed: 19/04/2016. 2016.
- [28] mitmproxy. *How mitmproxy works*. Accessed: 20/04/2016. 2016. URL: <http://docs.mitmproxy.org/en/latest/howmitmproxy.html>.
- [29] Nikolay Elenkov. *Certificate pinning in Android 4.2*. 2012.
- [30] mitmproxy. *About Certificates*. Accessed: 20/04/2016. 2016. URL: <http://docs.mitmproxy.org/en/latest/certinstall.html>.
- [31] A Cohen. “The iPhone Jailbreak: A Win Against Copyright Creep”. In: *Time.com* (2010).
- [32] Pangu. *Pangu Jailbreak*. <http://en.pangu.io>. Accessed: 20/04/2016. 2016.
- [33] Alban Diquet. *iOS SSL Kill Switch*. <https://github.com/iSECPartners/ios-ssl-kill-switch>. Accessed: 20/04/2016. 2016.
- [34] OWASP. *O-Saft*. <https://www.owasp.org/index.php/O-Saft/>. Accessed: 20/04/2016. 2016.

- [35] *RFC 5246 - The Transport Layer Security (TLS) Protocol Version 1.2*. <https://tools.ietf.org/html/rfc5246>. Accessed on 05/01/2016.
- [36] Google. *Rebooting Responsible Disclosure: a focus on protecting end users*. <https://security.googleblog.com/2010/07/rebooting-responsible-disclosure-focus.html>. (Accessed on 30/04/2016).
- [37] Nadhem AlFardan. *On the Security of RC4 in TLS*. <http://www.isg.rhul.ac.uk/tls/>. Accessed on 25/04/2016.
- [38] MIT Laboratory for Computer Science and RSA Data Security. *RFC 1321 - The MD5 Message-Digest Algorithm*. <https://tools.ietf.org/html/rfc1321>. (Accessed on 25/04/2016).
- [39] Souradyuti Paul and Bart Preneel. "On the (in) security of stream ciphers based on arrays and modular addition". In: *Advances in Cryptology—ASIACRYPT 2006*. Springer, 2006, pp. 69–83.
- [40] Upkar Varshney and Ron Vetter. "Mobile commerce: framework, applications and networking support". In: *Mobile networks and Applications* 7.3 (2002), pp. 185–198.
- [41] Alexander Mense et al. "Analyzing Privacy Risks of mHealth Applications." In: *Studies in health technology and informatics* 221 (2016), p. 41.
- [42] Google. *Monetize and promote with Google Ads*. Google Developers. <https://developers.google.com/ads/?hl=en>. (Accessed on 03/05/2016).
- [43] Apple. *Ad for Developers*. Apple Developer. <https://developer.apple.com/iad/>. (Accessed on 03/05/2016).
- [44] M Ossmann. "Project ubertooth". In: *Retrieved* 18 (2012), p. 23.
- [45] Mike Ryan. "Bluetooth: With Low Energy Comes Low Security." In: *WOOT*. 2013.
- [46] Elaine B Barker, Don Johnson, and Miles E Smid. "Recommendation for pair-wise key establishment schemes using discrete logarithm cryptography (revised)". In: (2007).
- [47] Markus Jakobsson and Susanne Wetzel. "Security weaknesses in Bluetooth". In: *Topics in Cryptology—CT-RSA 2001*. Springer, 2001, pp. 176–191.
- [48] Mark A Kurisko and Philip D Mooney. *Security apparatus and method during BLUETOOTH pairing*. US Patent 7,174,130. Feb. 2007.
- [49] Mike Ossmann. *Crackle, crack Bluetooth Smart (BLE) encryption*. <https://lacklustre.net/projects/crackle/>. (Accessed on 03/05/2016).
- [50] Marc Blanchou. *iSECPartners/Android-SSL-TrustKiller. Bypass SSL certificate pinning for most applications*. <https://github.com/iSECPartners/Android-SSL-TrustKiller>. (Accessed on 03/05/2016).