UNIVERSITY OF AMSTERDAM

THE NETWORK SECURITY OF CLIENT-SERVER IPHONE APPLICATIONS

Dennis Cortjens
dennis.cortjens@os3.nl

Iwan Hoogendoorn
iwan.hoogendoorn@os3.nl

Abstract

A lot of public places are offering free Wi-Fi networks. These Wi-Fi networks are often not very secure and configured with no or minimal security. Sniffing traffic on these networks is fairly easy. SSL/TLS connection are used by banks for online banking, but also by Facebook, Google and other companies to protect your private data. However, attacks on these secure connection have been succesful and are a serious thread, especially on free Wi-Fi networks. This research performed a SSL decrypt, SSL strip and SSL proxy attack on six populair iOS applications (ABN AMRO Mobiel Bankieren, PayPal, Apple App Store, Apple Facetime, Dropbox and Facebook) and provides prevention measures against the attacks, although the applications weren't vulnerable for the attacks. This research also defined a simple security level system for applications using secure connections.

Contents

1 Introduction

The success of Apple's App Store is remarkable and still growing. Apple launched the App Store on the 11th of July, 2008 with about 500 applications. Almost four years later, with the App Store offering over 500.000 applications in categories like entertainment, education, games, lifestyle, social media and much more, Apple has created a new market for iPad, iPhone and iPod (iOS) owners and developers. [1]

1.1 Problem

The growth of the App Store with applications containing private data, such as online banking and cloud services, has made security on iOS devices a top issue. A lot of companies like Binck (investment), PayPal (online paying), ABN AMRO and ING (online banking) offer applications that make life easier by checking your investment, checking your balance and transferring money where ever you are with your iOS device. Other applications like Dropbox (cloud storage), Facebook (social media) and Facetime (video calling) also bring some other services to your iOS device. But are the companies that provide these applications securing your data in a good way?

A lot of public places are offering free Wi-Fi networks. Whether you're on the train or at the McDonalds, they offer free Wi-Fi. These Wi-Fi networks are often not very secure and configured with no or minimal security. Sniffing traffic on these networks is fairly easy. Using applications like the ones mentioned earlier on public networks could become a high security risk, if the application doesn't have the right level of security.

SSL/TLS connections are used by banks for online banking, but also by Facebook, Google and other companies to protect your private data. However, attacks on these secure connections have been successful and are a serious thread, especially on free Wi-Fi networks.

1.2 Scope

The main research question in this study is:

What are good ways to secure an iOS application on the network and how is the network security currently in some popular iOS apps?

This question is researched with the following sub questions:

1. Which SSL attacks can be used to attack iOS applications?
2. How do these SSL attacks work?
3. How can these SSL attacks be performed?
4. What are the popular applications in the App Store?
5. What is the security level of these applications?
6. Is the security level in relation with the content the application offers?

This research will focus on the network security of iOS applications. A selection will be made of the most popular applications in the App Store, including at least two online banking applications.

2 Background

2.1 Secure Sockets Layer (SSL)

Secure Sockets Layer (SSL), as its successor Transport Layer Security (TLS), is a cryptographic protocol that provides secure communication over the internet. SSL and TLS encrypt the traffic segments at the transport layer of the Open Systems Interconnection (OSI) model. Both use asymmetric cryptography for the key exchange and symmetric cryptography for the actual data communication.

SSL and TLS ensure three important aspects of client/server communication:

Authentication Confirming the identity of the server (and client) to make sure those can be trusted.
Encryption Transforming the communication between the client and server to unreadable data, except for those the data is intended for.
Integrity Confirming the consistency of the data to make sure it isn't altered on the way.

2.1.1 History

The SSL protocol was originally developed by Netscape. SSL version 1.0 was never publicly released. SSL version 2.0 was publicly released in 1995, but because of a number of security flaws it was successed by SSL version 3.0 in 1996. SSL 3.0 was a complete redesign of the protocol by P. Kocher, P. Karlton and A. Freier (Netscape). The draft of version 3.0 was later published by the IETF as a historic document in RFC 6101 [2].

Its successor, the TLS protocol, was developed by T. Dierks and C. Allen (Certicom) in 1999. TLS version 1.0 was an upgrade to SSL 3.0, but according to its RFC 2246 [3]:
...the differences between this protocol and SSL 3.0 are not dramatic, but they are significant enough that TLS 1.0 and SSL 3.0 do not interoperate...
However, TLS 1.0 does include a mechanism to downgrade a connection to SSL. This eventually became a security flaw and was fixed in TLS 1.2. Another security flaw in TLS 1.0 was the Browser Exploit Against SSL/TLS (BEAST) attack, mentioned in section 2.2. TLS version 1.1 was released in 2006 and included fixes for Cipher Block Chaining (CBC) attacks, as well as some notes about new attacks against TLS. It was published by the IETF in RFC 4346 [4]. TLS version 1.2 was released in 2008 and included new Secure Hash Algorithm hashed and the Advanced Encryption Standard (AES) cipher suite. It was published by the IETF in RFC 5246 [5] and was refined in 2011 with RFC 6176 [6], which prohibited to downgrading to SSL 2.0.

2.1.2 Workings

SSL and TLS both work in the same way. There are two types of connections: server-side and mutual certificate authentication.

2.1.2.1 Server-side Certificate Authentication

The server-side certificate authentication process consists of the following steps:

1. client sends a hello message with its SSL version, cipher settings and other session specific data to the server
2. server sends a hello message with its SSL version, cipher settings, other session specific data and server certificate to the client
3. client authenticates the server certificate through the Certificate Authority (CA) chain
4. client generates the pre-master secret
5. client encrypts the pre-master secret with the server's public key from the server certificate and sends it to the server
6. server decrypts the pre-master secret with the server's private key
7. client and server generate the master secret and use it to generate session keys
8. client sends message that its future messages will be encrypted with the session key to the server
9. client sends message that the client's portion of the handshake is finished to the server

10. server sends message that its future messages will be encrypted with the session key to the client
11. server sends message that the server's portion of the handshake is finished to the client
12. client and server send (encrypted) data messages to each other and use the session key to encrypt and decrypt the data

This process is illustrated in figure 1.

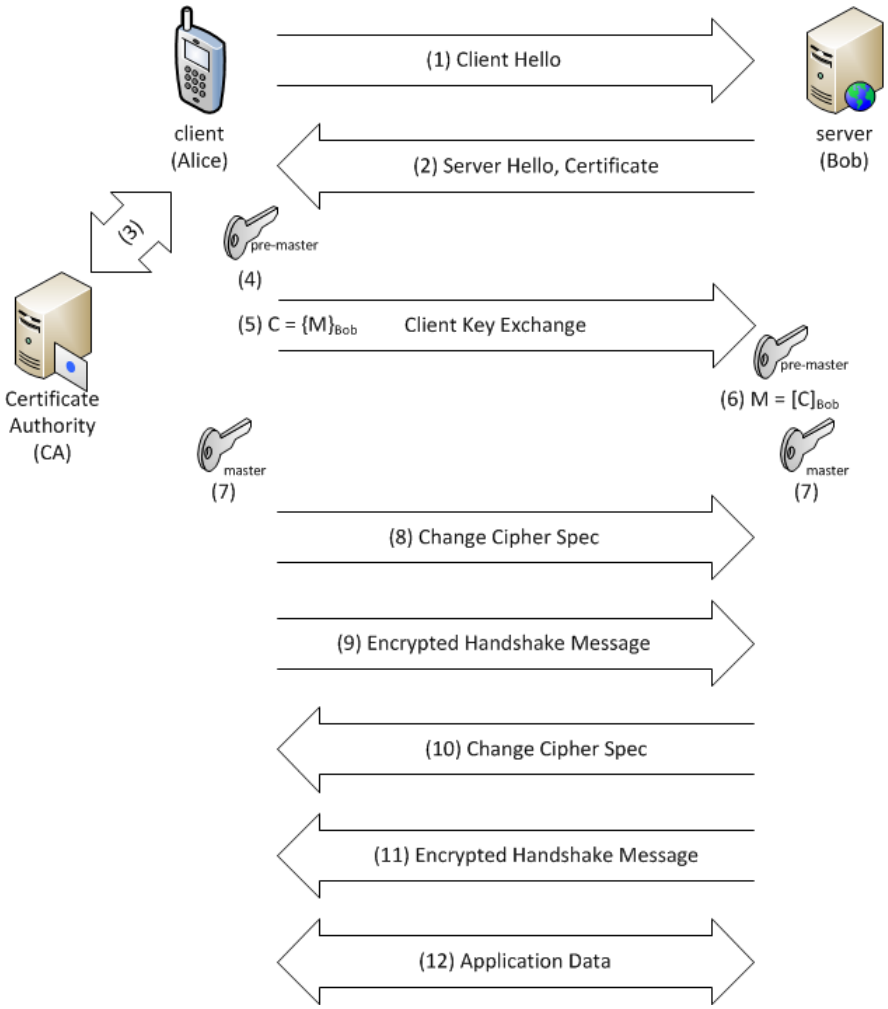

Figure 1: SSL Server-side Certificate Authentication

2.1.2.2 Mutual Certificate Authentication

The mutual certificate authentication process consists of the following steps:

1. client sends a hello message with its SSL version, cipher settings and other session specific data to the server
2. server sends a hello message with its SSL version, cipher settings, other session specific data and server certificate to the client
3. client authenticates the server certificate through the Certificate Authority (CA) chain
4. client generates the pre-master secret
5. client encrypts the pre-master secret with the server's public key from the server certificate
6. client signs the encrypted pre-master message and sends it with the client certificate to the server
7. server authenticates the client certificate through the Certificate Authority (CA) chain
8. server decrypts the pre-master secret with the server's private key

9. client and server generate the master secret and use it to generate session keys
10. client sends message that its future messages will be encrypted with the session key to the server
11. client sends message that the client's portion of the handshake is finished to the server
12. server sends message that its future messages will be encrypted with the session key to the client
13. server sends message that the server's portion of the handshake is finished to the client
14. client and server send (encrypted) data messages to each other and use the session key to encrypt and decrypt the data

This process is illustrated in figure 2.

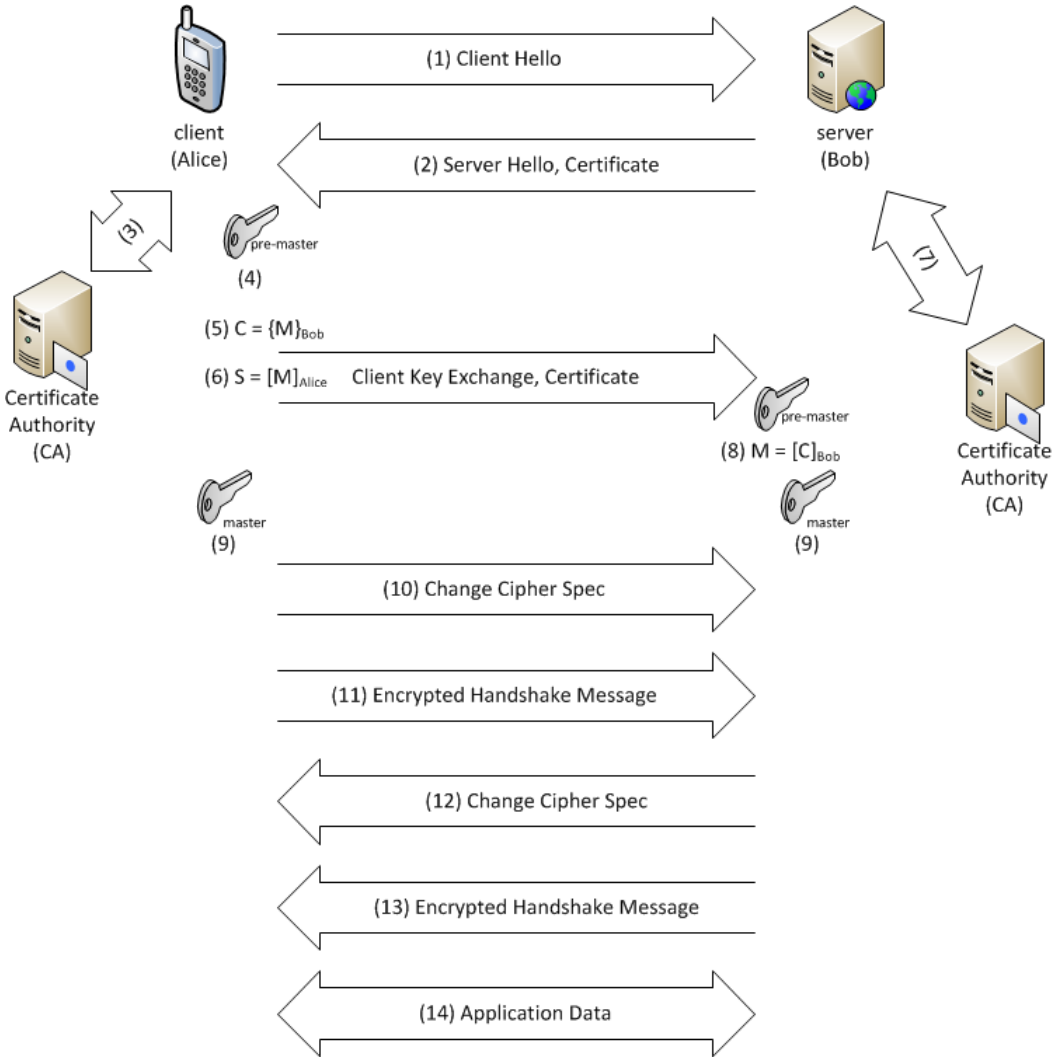

Figure 2: SSL Mutual Certificate Authentication

[7]

2.2 SSL Attacks

As mentioned in section 2.1.1 there are some security flaws in SSL and TLS, but most of them are fixed in the latest version of TLS (1.2). However, some developers still implement old versions of SSL and TLS in their applications and/or TLS in the wrong way. This makes attacks on SSL and TLS possible.

One of the most famous SSL attacks is the BEAST attack. It was discovered by Juliano Rizzo and Thai Duong in 2011. They presented the attack at the Ekoparty security conference in Buenos Aires, Argentina. The attack was based on an older exploit not using a random Initialization Vector (IV) in CBC mode for every TLS message, discovered by Wei Dai and Gregory Bard. Rizzo and Duong found a way to make this attack work against web browsers by generating cookies. A lot of web browser developers fixed the BEAST attack vulnerability within their TLS 1.0 implementation. Although it was fixed in TLS 1.1, this version of the protocol wasn't widely implemented yet. [8]

There are other SSL attacks better known as:

- SSL Decrypt
- SSL Strip
- SSL Proxy (with Self-signed Certificate)

2.2.1 SSL Decrypt

It is possible to decrypt SSL traffic with Wireshark [9] and Wireshark's SSL Dissector function. However, this can't be done on live network traffic. The decryption can only take place on captured SSL traffic using RSA key exchange. The data within the SSL traffic is only readable later on. This could be enough to find useful information, like usernames and passwords. Capturing SSL traffic can be done by creating a SPAN/monitor port on a switch or by a man-in-the-middle attack.
The following steps describe SSL decryption with Wireshark:

1. capture SSL traffic and save it as a *.pcap* file
2. open the Wireshark preferences by going to *Edit* > *Preferences...* or by pressing *Shift* + *Ctrl* + *P*
3. open the SSL preferences by going to *Protocols* > *SSL*
4. obtain the RSA private key from the server in a *.key* or *.pem* file
5. insert the Pre-Shared-Key in the field using the sequence <*ip*>,<*protocol*>,<*path-to-file*>,<*optional:password*>
6. click *Apply* and *OK*

[10] [11]

2.2.2 SSL Strip

It is also possible to strip SSL traffic with a man-in-the middle attack using arpspoof [13] and sslstrip [17]. With arpspoof the victim's computer is spoofed to send all traffic to the attacker's computer. When the victim is using a SSL connection, sslstrip creates a secure connection (HTTPS) between the attacker's computer and the server. On the other side sslstrip creates an insecure connection (HTTP) between the attacker's computer and the victim's computer. The server isn't aware of this intervention, because a server-side authentication only requires the server to be authenticated. The server doesn't know the real identity of the client. So this could be every (spoofed) computer. Because of the insecure connection between the attacker and the victim, usefull information is send in plain back and forth from the victim's computers. So the attacker is able to read the data. The victim is able to detect such an attack. Normally, when using a secure connection in a web browser the victim will see 'https://' and a green field with the website's certificate. But because the connection between the attacker and victim is HTTP and insecure, it will be showed as 'http://' and without the website's certificate. However, not many people check the connection before navigating to a secure website.

This is illustrated in figure 3.

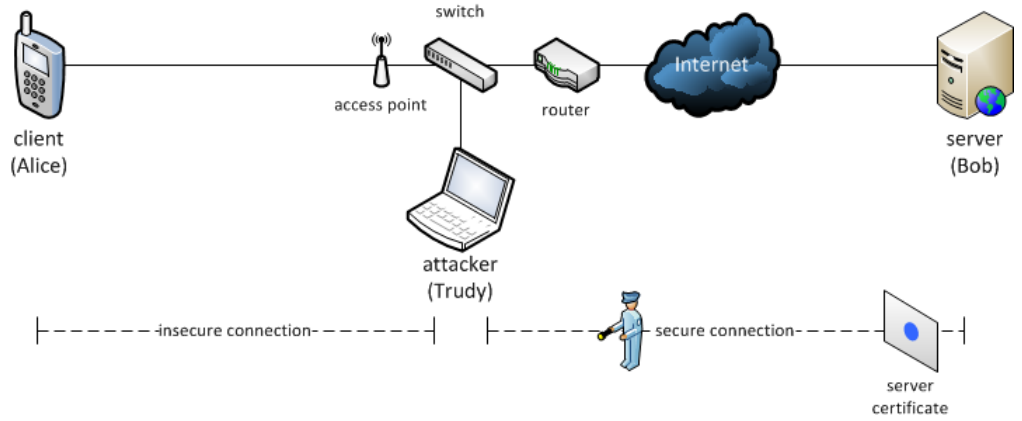

Figure 3: SSL Strip attack

The following steps describe a SSL strip attack with BackTrack 5 R2:

1. open *Terminal*
2. enable IPv4 forwarding by using the
   echo 1 > /proc/sys/net/ipv4/ip_forward command
3. redirect HTTP (port 80) traffic to another port (10000) by using the
   iptables -t nat -A PREROUTING -p tcp –destination-port 80 -j REDIRECT –to-port 10000 command
4. open another *Terminal*
5. run *arpspoof* on the network interface and the victim's IP address by using the
   arpspoof -i eth0 -t <victim-ip> <gateway> command
6. run *sslstrip* by going to *BackTrack > Exploitation Tools > Web Exploitation Tools > sslstrip*
7. run *sslstrip* to log all traffic to and from the server by using the
   python sslstrip.py -a command
8. open another *Terminal*
9. run *ettercap* to quietly output text from the network interface by using the
   ettercap -T -q -i eth0 command
10. wait for the victim to login

[14] [15] [16]

2.2.3 SSL Proxy

The SSL proxy creates an invisible/transparent proxy for SSL traffic on (a part of) the network and makes sure all network traffic is send through that proxy. In most cases this will be the attacker's computer. A self-signed certificate will make sure the connection to the attacker's computer is also secured, creating a 'secure' connection from client to server. In this case the victim isn't able to detect such an attack by looking for 'http://' and the website's certificate, because the connection at both sides of the attacker is secured. However, this attack should give the user a certificate warning, because the certificate between the attacker's computer and victim's computer is self-signed and not trusted. Unfortunately, not many people check this warning and just click continue to navigate further. This attack is possible by using dnsspoof [13] and Burp Suite [18].

This is illustrated in figure 4.

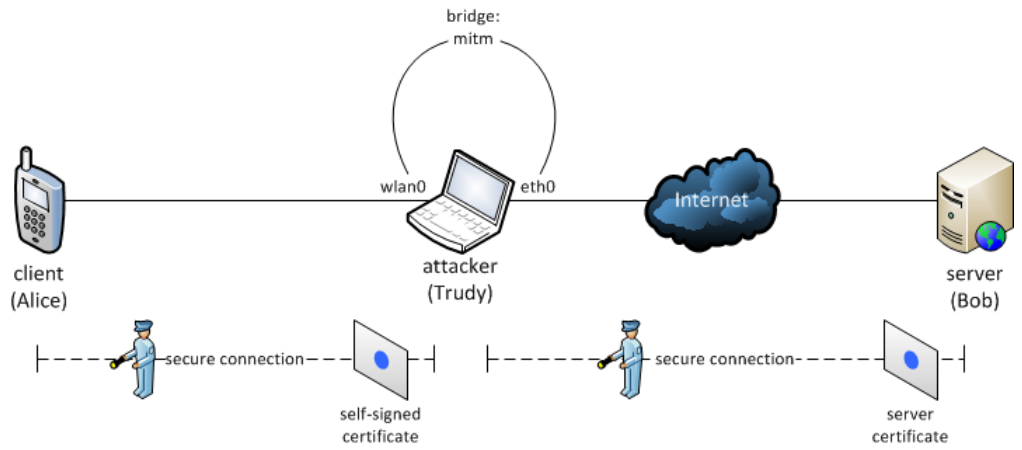

Figure 4: SSL Proxy attack

The following steps describe a SSL proxy with self-signed certificate attack with BackTrack 5 R2:

1. open *Terminal*
2. run *dnsspoof* on the network interface by using the *dnsspoof -i mitm* command
3. go to the *Burp Suite* folder by using the *cd /pentest/web/burpsuite/* command
4. run *Burp Suite* to start the proxy server by using the *java -jar -Xmx2g burpsuite v1.4.01.jar &* command
5. open the proxy options by going to *proxy* > *options*
6. add HTTP port 80 traffic with *support invisible proxying for non-proxy-aware clients* by entering/selecting the data and clicking *add*
7. add HTTPS port 443 traffic with *support invisible proxying for non-proxy-aware clients* and *use a self-signed certificate* by entering/selecting the data and clicking *add*
8. open the intercept console by going to *proxy* > *intercept*
9. disable interception to stop the need for confirming every packet by clicking *intercept is on*
10. wait for the victim to use SSL secured websites

[19] [20]

2.3 Previous research

A lot of research has been done on the security of the iPhone itself. This research generally focuses on the security of the hardware and operating system. And not on the security of a specific application or the security of an application from a network perspective. SSL attacks has been researched more often and also on an academic level. A lot of practical tutorials on performing these attacks can be found on ethical hacking blogs and YouTube. A combination of these fields has been done, but not on an academic level.

In 2002 Peter Burkholder researched man-in-the-middle attacks on SSL. He described a SSL strip like attack with arpspoof and dnsspoof on secure web browser traffic. [21] Although this isn't exactly like the SSL strip attack performed in this research, this paper helped in understanding the workings of man-in-the-middle attacks with arpspoof and dnsspoof on SSL connections.
In 2004 Kristof Boeynaems extensively researched this subject too. He described an in depth background of SSL and possible/impossible attacks on the SSL protocol, including timing and man-in-the-middle attacks. [22] Although this is a theoretical research, this in depth paper helped in understanding the workings of the SSL protocol and the possible/impossible attacks on SSL connections.
In both the SSL protocols up to TLS 1.0 are described.

In 2003 some research was done with timing attacks on SSL. Brice Canvel, Alain Hiltgen, Serge Vaudenay and Martin Vuagnoux researched the weakly implemented CBC mode [23]. David Brumley and Dan Boneh researched the unprotected OpenSSL implementations [24].
In 2005 this last research was improved by Onur Aciicmez, Werner Schindler and Cetin Koc. They improved the efficiency of the attack by a factor of more than 10. [25]
Although this isn't useful for our research, timing attacks are a possible attack on SSL.

These findings are somewhat outdated, but they are a good basis and reference for this research.

3 Preparation

3.1 Security Levels

There isn't a standardized system that defines the security levels of applications based upon the data they process or store. Such a system would give application developers a guideline in the level of security they need for their application. With hundreds of programming languages and thousands of developers in the world, it is hard to create such a system. Not all programming languages provide the same security features and not all developers will have their own best practices for security. This research tries to define a simple security level system for applications which are based on the data they process or stores and the most common and well-known SSL ciphers.
This report defines the following security levels:

Highest An application that uses a TLS 1.x connection with an AES cipher. The AES cipher is considered to be sufficient to protect classified information up to secret (AES 128-bit and up) and even top secret (AES 192-bit and up) [26]. So this should be sufficient to protect applications that provide online banking.

High An application that uses a TLS 1.x connection with a 3DES cipher. The 3DES cipher is considered a weaker cipher, because of a lesser maximum block length of 168-bit (3x DES 56-bit) and an efficiency loss, against AES [27]. However, this should provide sufficient security for payment transactions, like an online store.

Medium An application that uses a TLS 1.x connection with a RC4 cipher. The RC4 cipher is considered a safe cipher, but there are some implementation that have proven to be weak and crackable [28]. It should be sufficient to protect private data, like documents. But is insufficient for any payment transactions or online banking.

Weak An application that uses a TLS 1.x connection with a DES cipher. The DES cipher is considered an insecure cipher, because the maximum block length of 56-bits is easily crackable [29]. There even is special hardware available to crack DES encrypted data. It should not be used and must be replaced by 3DES or AES.

Weakest An application that uses a SSL x.0 connection. As mentioned in section 2.1 the old SSL versions contain security flaws and are considered insecure. They should not be used anymore and must be replaced by TLS.

None An application that uses no secure connection at all.

These security levels are schematically shown in figure 5.

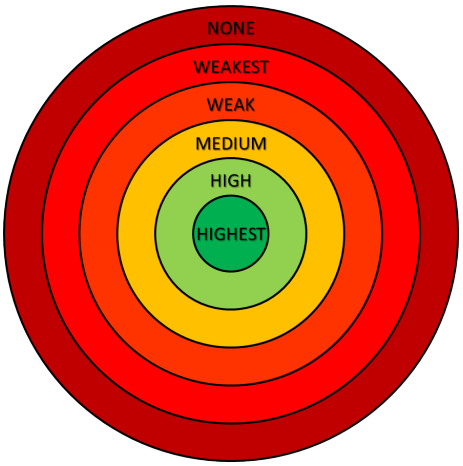

Figure 5: Security levels

3.2 Test Environment

The test environment for this research was set up with different hardware and software components.

Hardware:

Apple iPhone 4 The Apple iPhone 4 smartphone is the victim.
Notebook The notebook computer is the attacker.
Switch The switch with SPAN/monitor port is the core component for connecting the wired devices.
Wireless Access Point The wireless access point is the core component for connecting the wireless devices.

Software

arpspoof A command line application to send spoofed Address Resolution Protocol (ARP) messages onto the network to associate the attacker's MAC address with another IP address on the network [13].
BackTrack A custom Ubuntu distribution, available as boot CD, installation and virtual machine, with a large collection of security-related tools for penetration testing purposes [30].
Burp Suite A collection of different security-related tools, like a proxy server and web spider, for penetration testing purposes [18].
dnsspoof A command line application to send spoofed Domain Name System (DNS) messages onto the network to reroute traffic to the attacker's IP address [13].
iptables A Linux kernel firewall.
sslstrip A command line application to strip the secure part of a connection and capture usernames and passwords on the network [17].
Wireshark A tool to capture and browse network traffic and to analyse network protocols [9].

The SSL attacks require different environments. Therefore we build two environments. One for the SSL decrypt and strip attacks and one for the SSL proxy attack.

3.2.1 SSL Decrypt / Strip

The environment for the SSL decrypt and strip attacks consists of a switch with a port connected to the internet by the Dynamic Host Configuration Protocol (DHCP). A port set as SPAN/monitor connected to the attacker's notebook computer and a port connected to the wireless access point. This wireless access point is the gateway to the network for the Apple iPhone 4 which is the victim. This is schematically shown earlier in figure 3.

3.2.2 SSL Proxy

The environment for the SSL proxy attack consists of the attacker's notebook computer with a wired and wireless network interface card. The wired side is connected to the internet by DHCP. The wired and wireless interfaces are bridged, so the wireless side can use the internet connection of the wired side, also by DHCP. The wireless side is configured in master/access point mode to act as an access point for the Apple iPhone 4 which is again the victim. This is schematically shown earlier in figure 4.

3.3 Test Specification

The client-server applications chosen for this research are two online banking applications and four other popular applications.

Online banking applications:

ABN AMRO Mobiel Bankieren The online banking application of one of biggest banks in the Netherlands.
PayPal The mobile application of the world's biggest online money transfer service.

Other applications:

Apple App Store The application for installing new application on a iOS device.
Apple FaceTime The mobile application for Apple's video calling service.
Dropbox The mobile application of Dropbox's cloud storage service.
Facebook The mobile application of the world's biggest social media service.

4 Testing and Analysing

4.1 SSL Decrypt

For the SSL decrypt attack step 1 as described in section 2.2.1 was used for each application. Before each test, all applications on the Apple iPhone 4 were closed to make sure there is no other traffic within the capture. The RSA private key could be obtained by creating a physical image of the iPhone with the UFED Physical Analyser forensic software. And then browse or carve this image for the possible key. This is a study in itself and couldn't be performed, because of the limited time for this research. To analyse the findings the .pcap file was opened with Wireshark and the most common IP address in the capture was checked to belong to the company of the application by querying the RIPE and ARIN databases. Then a filter was placed on the victim's IP address and the checked IP address. In that way the traffic back and forth from the application was distinguished. In a case of multiple IP addresses for the connection, the filter was extended with those IP addresses. This attack resulted in some interesting information about the SSL connection in the applications.

ABN AMRO Mobiel Bankieren The ABN AMRO Mobiel Bankieren application showed a TLS version 1.0 connection with a TLS_RSA_WITH_AES_256_CBC_SHA cipher suite. This connection is used for both authentication and data transmission. Although they do not use the latest version of TLS, they did add the AES cipher suite (RFC3268 [31]) which is officially added in TLS 1.2. The AES cipher suite increases the efficiency and security. This increase is provided by the way AES is build (effenciency) and is provided by bigger block lengths of 192-bit and 256-bit (security). An attack with the known RSA private key should succeed, because of the use of RSA key exchange. Another interesting discovery is that ABN AMRO has registered its IP addresses with ARIN (America) and not RIPE (Europe).

Paypal The PayPal application showed a TLS version 1.2 connection with a TLS_RSA_WITH_3DES_EDE_CBC_SHA cipher suite. This connection is used for both authentication and data transmission. Although they use the latest version of TLS, they do not use the AES cipher suite. Triple Data Encryption Standard (3DES) has a maximum block length of 168-bit (3x DES 56-bit), were AES has block lengths of 192-bit and 256-bit. An attack with the known RSA private key should succeed.

Apple App Store The Apple App Store application showed a TLS version 1.0 connection with a TLS_RSA_WITH_RC4_128_MD5 cipher suite. This connection is used only for authentication. Rivest Cipher 4 (RC4) has known weaknesses that argue against it in using it in new systems, especially when it's weakly implemented [28]. This could make the Apple App Store a security risk, especially when using it for authenticating banking information for paid applications. An interesting discovery with the data transmission is that they use a balancing service for downloading applications and that this connection isn't secured. The balancing service is provided by Akamai Technologies [32] and was discovered by a third party IP address in the captured traffic. An attack with the known RSA private key should succeed.

Apple FaceTime The Apple FaceTime application showed a TLS version 1.0 connection with a TLS_RSA_WITH_RC4_128_MD5 cipher suite. This connection is again used only for authentication. Apple apparently uses this for all their iOS applications. So it comes with the same possible security risks as the Apple App Store. However, Apple FaceTime isn't using your banking information and so the risk is lower. The data transmission part was a connection over the User Datagram Protocol (UDP) with the Session Initiation Protocol (SIP) which is used for multimedia communication, including Voice over IP (VoIP). Although it is be possible to use SSL over UDP [7], this would be very unreliable. UDP has no guarantee of packets arriving and therefore encrypted data would not be decipherable. An attack with the known RSA private key should succeed.

Dropbox The Dropbox application showed a TLS version 1.0 connection with a TLS_RSA_WITH_RC4_128_MD5 cipher suite. This connection is used for both authentication and data transmission. The cipher suite is the same as with Apple, but Dropbox isn't using your banking information and so the security risk is lower. An attack with the known RSA private key should succeed.

Facebook The Facebook application showed a TLS version 1.2 connection with a TLS_RSA_WITH_RC4_128_SHA cipher suite. This connection is used for both authentication and data

transmission. Again, the cipher suite is the same as with Apple. Although they use the latest version of TLS, they do not use the AES cipher suite which would increases the efficiency and security of the cipher, especially pertaining to RC4. Facebook isn't using your banking information in the application and so the security risk is lower. An attack with the known RSA private key should succeed.

The information is summarized per application in table 1.

Application	SSL	Cipher suite	Authentication	Data
ABN AMRO	TLSv1	TLS_RSA_WITH_AES_256_CBC_SHA	X	X
PayPal	TLSv1.2	TLS_RSA_WITH_3DES_EDE_CBC_SHA	X	X
Apple App Store	TLSv1	TLS_RSA_WITH_RC4_128_MD5	X	-
Apple FaceTime	TLSv1	TLS_RSA_WITH_RC4_128_MD5	X	-
Dropbox	TLSv1	TLS_RSA_WITH_RC4_128_MD5	X	X
Facebook	TLSv1.2	TLS_RSA_WITH_RC4_128_SHA	X	X

Table 1: SSL connection information per application

4.2 SSL Strip

For the SSL strip attack the steps as described in section 2.2.2 were used for each application. Wireshark was ran simultaneously to see what happened on the network and the captures from each application were saved. Before each test, all applications on the Apple iPhone 4 were closed to make sure there is no accidental hit from other traffic. To make sure the attack should work, a browser login was made after each application. To analyse the findings the .pcap file was used and the IP address checking and filtering was done as mentioned in section 4.1.

ABN AMRO Mobiel Bankieren The ABN AMRO Mobiel Bankieren application didn't show any login information with sslstrip. Instead it showed an error message saying '*No connection Unable to establish a connection. Check your Internet connection.*' as showed in figure 6. The attack was unsuccessful on this application.

PayPal The PayPal application didn't show any login information with sslstrip. Instead it showed an error message saying '*Error System error. Please try again later.*' as showed in figure 7. The attack was unsuccessful.

Apple App Store The Paypal application didn't show any login information with sslstrip. Instead it showed an error message saying '*Cannot connect to iTunes Store*' as showed in figure 8. The attack was unsuccessful.

Apple FaceTime The Apple FaceTime application didn't show any login information with sslstrip, but it didn't show any error message either. The application was even able to establish a streaming video connection. However, the attack was still unsuccessful on the application. This is probably caused by the way Apple FaceTime works. As mentioned in section 4.1 the secure connection is only used for authentication and not for the data transmission. The application is able to establish a video streaming connection without authentication, so authentication isn't mandatory. The attack isn't successful and doesn't show any login information, because the authentication fails and no login information is send.

Dropbox The Dropbox application didn't show any login information with sslstrip. Instead it showed an error message saying '*Unable to Connect to Dropbox There may be a problem with your iPhone's Internet connection.*' as showed in figure 9. The attack was unsuccessful.

Facebook The Facebook application didn't show any login information with sslstrip. Instead it showed an error message saying '*Login Failed Sorry, an unexpected error occurred. Please try again later. (-1200)*' as showed in figure 10. The attack was unsuccessful on this application.

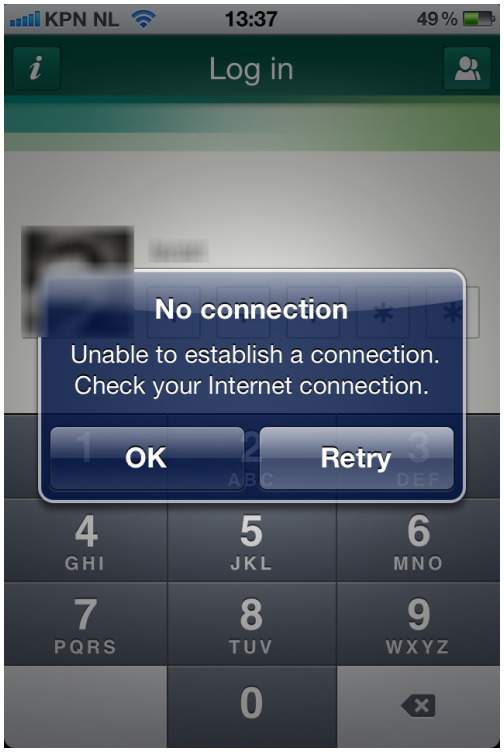

Figure 6: SSL Strip ABN AMRO error message

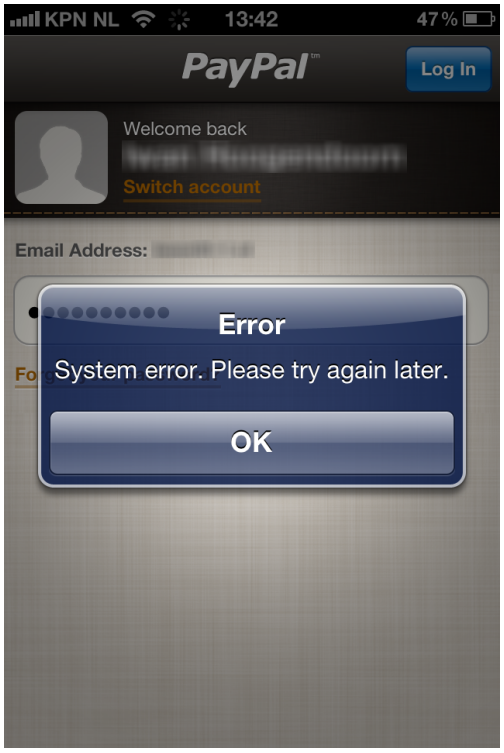

Figure 7: SSL Strip PayPal error message

Figure 8: SSL Strip Apple App Store error message

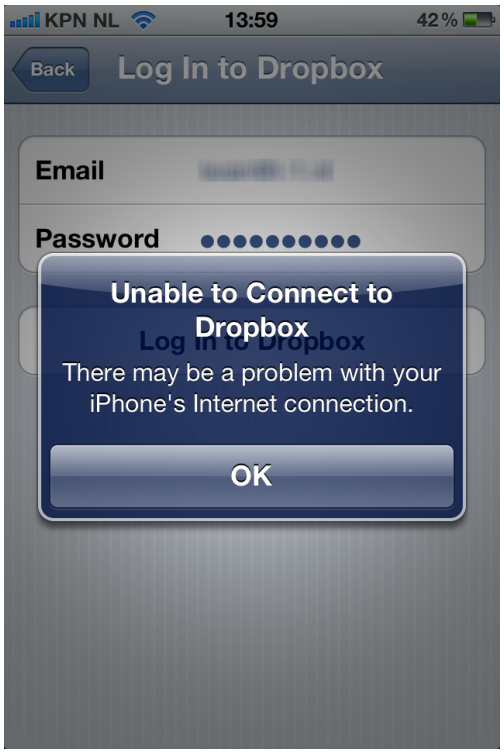

Figure 9: SSL Strip Dropbox error message

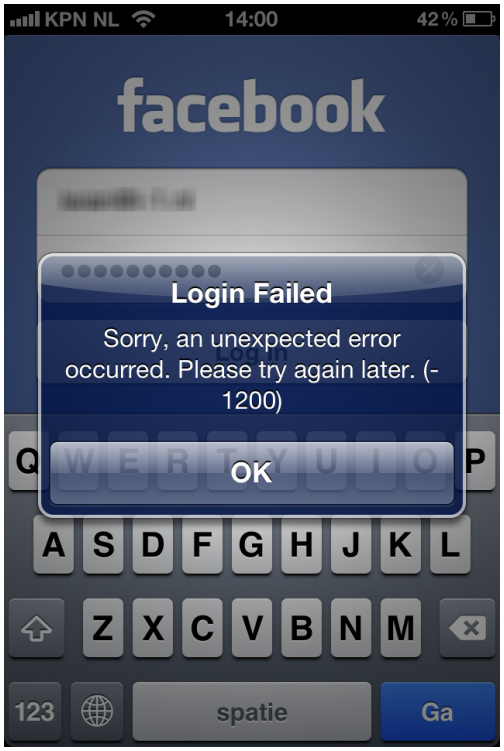

Figure 10: SSL Strip Facebook error message

Analysing the captured network traffic showed a SSL handshake failure as showed in figure 11. This failure is probably caused by the application checking for a secure connection or using mutual authentication between the server and client. Because sslstrip establishes a secure connection (HTTPS) between the attacker and server and an insecure connection (HTTP) between the attacker and client, the secure connection can't be established and therefore the SSL handshake fails. ABN AMRO Mobiel Bankieren, PayPal, Apple App Store, Dropbox and Facebook showed a connection error message for the SSL handshake failure. Apple FaceTime showed no error message and the streaming video connection was established between the two Apple FaceTime users. Wireshark confirmed this by showing the UDP stream.

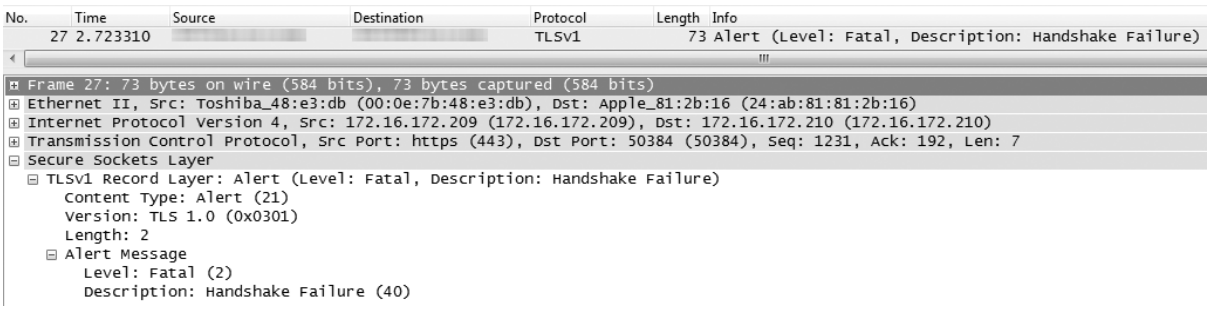

Figure 11: SSL handshake failure in Wireshark

4.3 SSL Proxy

For the SSL proxy attack the steps as described in section 2.2.3 were used for each application. Wireshark was ran simultaneously to see what happened on the network and the captures from each application were saved. Before each test, all applications on the Apple iPhone 4 were closed to make sure there is no other

traffic within the capture. To make sure the attack should work, a HTTPS website was visited after each application. To analyse the findings the .pcap file was used and the IP address checking and filtering was done as mentioned in section 4.1.

ABN AMRO Mobiel Bankieren The ABN AMRO Mobiel Bankieren application couldn't connect to its server. Instead it showed an error message saying '*Service unavailable ABN AMRO Mobiel Bankieren is temporarily unavailable. Please try again later.*' as showed in figure 12. The attack was unsuccessful on this application.

PayPal The PayPal application couldn't connect to its server. Instead it showed an error message saying '*Error System error. Please try again later.*' as showed earlier with the SSL strip attack in figure 7. The attack was unsuccessful.

Apple App Store The Apple App Store application couldn't connect to its server. Instead it showed an error message saying '*Cannot connect to the Store A secure connection could not be established. Please check your Date & Time settings.*' as showed in figure 13. The attack was unsuccessful.

Apple FaceTime The Apple FaceTime application couldn't connect to its server, but it didn't show any error message either. The application was again able to establish a streaming video connection. However, the attack was still unsuccessful on the application. This is caused by the same issue as with the SSL strip attack mentioned in section 4.1.

Dropbox The Dropbox application couldn't connect to its server. Instead it showed an error message saying '*Unable to Connect to Dropbox There may be a problem with your iPhone's Internet connection.*' as showed earlier with the SSL strip attack in figure 9. The attack was unsuccessful.

Facebook The Facebook application couldn't connect to its server. Instead it showed an error message saying '*Login Failed Sorry, an unexpected error occurred. Please try again later. (-1202)*' as showed in figure 14. The attack was unsuccessful.

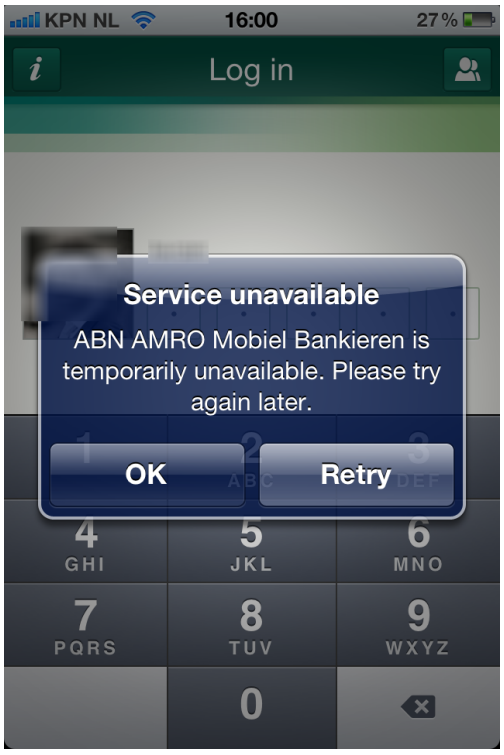

Figure 12: SSL Proxy ABN AMRO error message

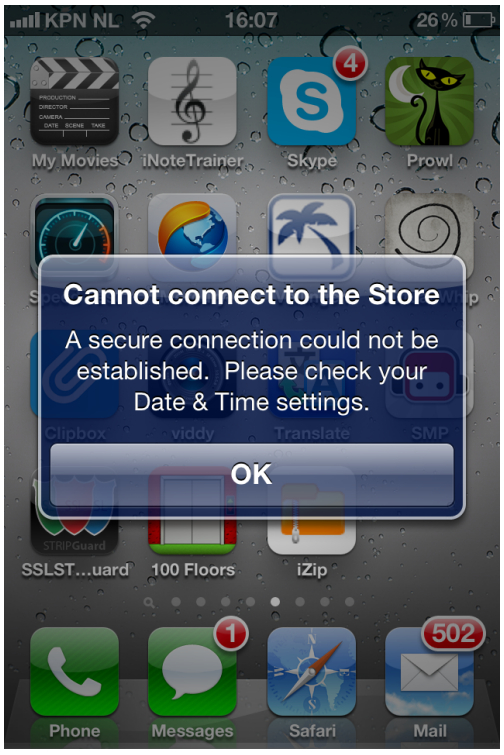

Figure 13: SSL Proxy Apple App Store error message

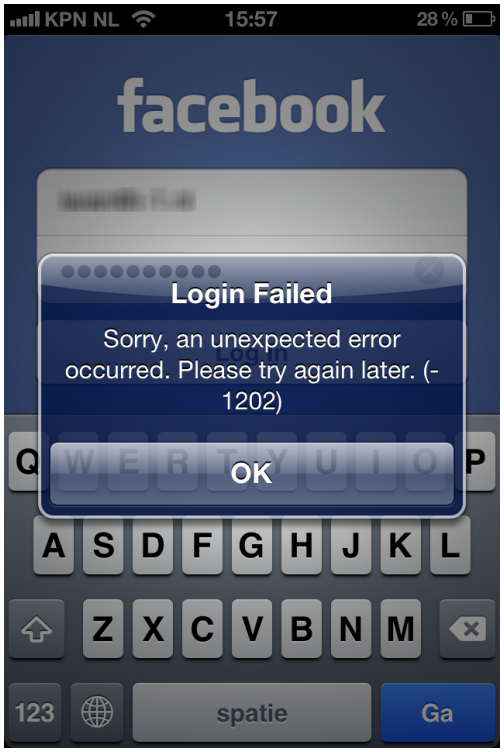

Figure 14: SSL Proxy Facebook error message

Analysing the captured network traffic showed a SSL handshake failure as showed earlier in figure 11. This failure is caused by the self-signed certificate which isn't trusted and therefore can't be used for establishing a secure connection. The interesting part is that the handshake terminates at the attacker's computer, because the self-signed certificate resides there. This is probably the reason why ABN AMRO Mobiel Bankieren, PayPal, Dropbox and Facebook showed a connection error message. The Apple App Store is the only application that explicitly said it can't establish a secure connection. Apple FaceTime again showed no error message and the streaming video connection was established between the two Apple FaceTime users. Wireshark confirmed this by showing the UDP stream.

5 Conclusion

5.1 General

In general this research provided some interesting information about iOS application security. All the applications use a SSL connection. The online banking applications use TLS 1.x and use a strong cipher suite. The ABN AMRO Mobiel Bankieren application has the **highest** level of security, closely followed by the Paypal application with a **high** security level. This is expected from banking applications that have your banking information to do money transfers, although the security level of PayPal was expected to be higher. Apple FaceTime, Dropbox and Facebook use a weaker cipher suite and have a **medium** level of security, but provide enough security for the data they process or store. However, the Apple App Store uses this cipher suite too and uses your banking information to do money transfers. This is a security risk. An application that uses banking information should be at least secured with a **high** security level. This is summarized in table 2 which shows the expected level of security, the tested level of security as described earlier and the balance between those two. This shows that two out of six applications lack the security level that they should have according to the data they process or store.

Application	**Expected Security Level**	**Tested Security Level**	**Balance**
ABN AMRO Mobiel Bankieren	highest	highest	$=$
PayPal	highest	high	$<$
Apple App Store	high	medium	$<$
Apple FaceTime	medium	medium	$=$
Dropbox	medium	medium	$=$
Facebook	medium	medium	$=$

Table 2: Security levels per application

The applications per security level are shown in figure 15.

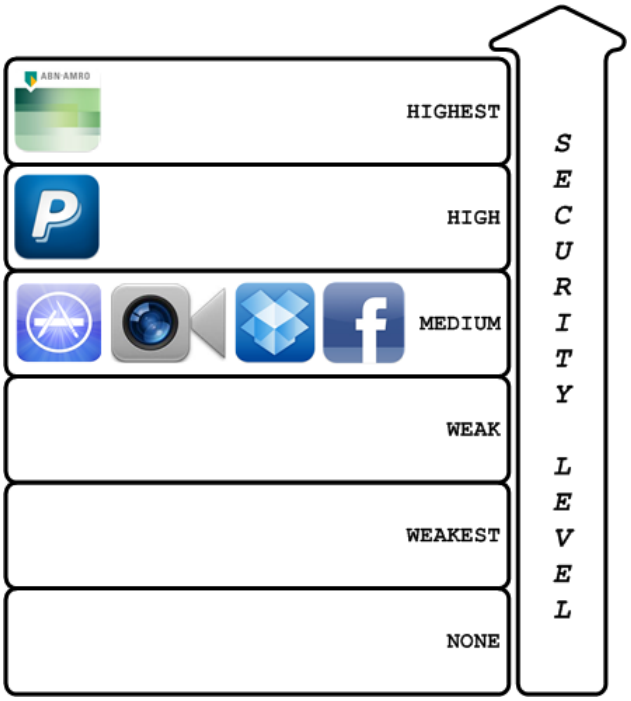

Figure 15: Applications per security level

The attacks performed on the applications are summarized in table 3. The conclusion of each attack will be described in the following sections.

Application	SSL Decryption	SSL Strip	SSL Proxy
ABN AMRO Mobiel Bankieren	-[1]	-	-
PayPal	-[1]	-	-
Apple App Store	-[1]	-	-
Apple FaceTime	-[1]	-	-
Dropbox	-[1]	-	-
Facebook	-[1]	-	-

[1] theoretically possible, but not tested

Table 3: SSL attacks per application

5.2 SSL Decrypt

Although the full and actual attack couldn't be performed, the information that this attack provided showed the possibility of the attack. All the applications use a RSA key exchange, so a SSL decrypt attack should succeed. A SSL decrypt attack can provide fully decrypted SSL traffic and could be seriously harmful, when an attacker has access to your wired network or is in range of your wireless network. On free Wi-Fi networks this is a serious risk, especially when doing money transfers on such a network.

5.2.1 Prevention

Preventing SSL decrypt attacks can be done from a user, application developer and system administrator perspective. From a user perspective you should never do private communication on a free Wi-Fi network or even not use such a network at all. Every telecom provider can offer you an internet connection and those connections are considered a lot safer then free Wi-Fi networks.

From an application developer perspective two things can be done to protect an application from a SSL decrypt attack:

protect server's private key The private key of the server should be protected. A private key in a separate file is vulnerable for theft and should be set with the right access permissions. Some operating systems will protect the private key on your behalf by marking them as not exportable. However, the operating system has to access the key to be able to use it and this is exploitable. Make sure to use the latest security patches that will fix such exploits and keep your private key safe.

don't use RSA key exchange The RSA key exchange is susceptible to eavesdropping when the private key of the server is compromised. The Diffie-Hellman key exchange can be used instead and is by design resistant to eavesdropping, but can be susceptible to a man-in-the-middle attack. Make sure to choose the key exchange algorithm wisely and subsidiary to the environment and purpose of the connection.

[33]

From a system administrator perspective the access to the network, both wired and wireless, should be restricted. Switches should use port security to prevent an attacker from plugging into your network. Wireless access points should have at least WPA2 security and MAC filtering enabled. Although MAC addresses can be spoofed and wireless does not have the ability to shutdown ports in a case of violation, these measures make it a lot harder for an attacker.

5.3 SSL Strip

The test results show that the SSL strip attack isn't effective on the applications, because they use a connection that checks for being secure or has mutual authentication. So both sides of the connection have to be trusted for a connection to be established. This means the security of the applications against a SSL strip attack

is sufficient. Although the attack isn't effective against the applications, a SSL strip attack is considered seriously harmful. It can provide login information even when using a secure connection, if an attacker has access to your wired network or is in range of your wireless network. On free Wi-Fi networks this is a high security risk, especially when doing money transfers on such a network.

5.3.1 Prevention

Preventing SSL strip attacks can be done from a user, application developer and system administrator perspective. From a user perspective four things can be done to protect you from a SSL strip attack:

don't use free Wi-Fi You should never do private communication on a free Wi-Fi network or even not use such a network at all. Every telecom provider can offer you an internet connection and those connections are considered a lot safer then free Wi-Fi networks.

check 'https://' and the certificate You should check whether or not you're using a HTTPS connection by looking in the address bar for 'https://' in front of your URL. Besides that you should also look for a green field with the certificate. This certificate shows you the company name of the website or the company name of one of the Certificate Authorities, like VeriSign, Thawte, Geotrust, GoDaddy and Comodo. If you can't find both or the application is showing a message saying it can't verify the identity of the site, don't use the site! Be aware of false favo icons (the icon next to the address bar) that show a lock icon and pretend the connection to be safe.

use StripGuard StripGuard is a free mobile application for Android and iOS devices by ACIS Professional Center. ACIS Professional Center is a Thai IT security company. The application checks a connection for still using HTTPS. If it is, the network is safe and the user could use it (but be aware of the things pointed out earlier). If it isn't, a hacker is active on that network and the user should avoid it. This application has been used in this research and works really well. [34] [35]

use HTTPS Everywhere HTTPS Everywhere is a free browser plugin for Google Chrome and Mozilla Firefox by the Electronic Frontier Foundation and The TOR Project. The Electronic Frontier Foundation and The TOR Project are organizations that fight for a free and anonymous internet. The application rewrites all requests from the browser to HTTPS, so a HTTPS connection can't drop back to HTTP. [36]

From an application developer perspective an application should check the connection for being secure or use mutual authentication to make sure both sides of the connection can be trusted and no one can eavesdrop in the middle. Make sure that authentication is mandatory for any live streaming connection to establish, otherwise it is established anyway (like with Apple FaceTime).

From a system administrator perspective the measures mentioned in section 5.2.1 should be taken.

5.4 SSL Proxy

The test results show that the SSL proxy attack isn't effective on the applications, because they don't trust the self-signed certificate. This means the security of the applications against a SSL strip attack is sufficient. The SSL proxy attack is considered less harmful then a SSL strip attack, because most applications including web browsers will show a certificate warning. However, it can still provide login information, when a user decides to manually continue with the untrusted certificate. On free Wi-Fi networks this is a security risk, especially when doing money transfers on such a network.

5.4.1 Prevention

Preventing SSL proxy attacks can be done from a user, application developer and system administrator perspective. From a user perspective two things can be done to protect you from a SSL strip attack:

don't use free Wi-Fi You should never do private communication on a free Wi-Fi network or even not use such a network at all. Every telecom provider can offer you an internet connection and those connections are considered a lot safer then free Wi-Fi networks.

check the certificate You should check whether or not you're using a trusted certificate. This can be done by looking for the green field with the certificate. This shows you the company name of the website or the company name of one of the Certificate Authorities. If you can't find this or the application is showing a message saying it can't verify the identity of the site, don't use the site! Be aware of false favo icons (the icon next to the address bar) that show a lock icon and pretend the connection to be safe.

From an application developer perspective the measures mentioned in section 5.3.1 should be taken.

From a system administrator perspective again the measures mentioned in section 5.2.1 should be taken.

5.5 Achievements

With this research project some important things are achieved:

- defined a simple security level system for application using secure connections
- showed six popular iOS applications that aren't vulnerable to a SSL strip or proxy attack
- showed prevention solutions for the SSL attacks
- set a basis for further research

5.6 Future research

Other applications Looking at the Apple App Store, there are so many different applications that use client-server architecture. These applications all have different security levels that will fit in one of the security levels described in section 3.1. With so many applications available more research can be done on these other applications.

Extracting RSA private key What may complete the SSL decrypt attack, is trying to extract the RSA private key from the server. It should be impossible to get this from the servers of the tested applications, especially ABN AMRO Mobiel Bankieren and PayPal. However, the private key may be found and extracted from the client. So browsing or carving the physical image of the Apple iPhone 4, may provide the information needed for a successful SSL decrypt attack.

Trusting self-signed certificate This research showed that some applications do not trust the self-signed certificate and therefore can't establish the secure connection. If the self-signed certificate is to be trusted by the Apple iPhone 4, then the application should trust the certificate as well and the SSL proxy attack could eventually be successful. There is a YouTube video on this involving Twitter [37].

Developing own application Another way to capture the secure traffic is by creating some kind of malware application for the Apple iPhone that sits between the application and network stack, captures the data and sends it to the attacker's computer or server. In order to do this the iPhone needs to be rooted or so called jailbroken. Developing such an application will be interesting from both an ethical hacking and anti-malware developing perspective.

References

[1] Wikipedia: App Store (iOS), http://en.wikipedia.org/wiki/App_Store_(iOS).

[2] RFC 6101: The Secure Sockets Layer (SSL) Protocol Version 3.0, http://tools.ietf.org/html/rfc6101/, 2011.

[3] RFC 2246: The TLS Protocol Version 1.0, http://tools.ietf.org/html/rfc2246/, 1999.

[4] RFC 4346: The Transport Layer Security (TLS) Protocol Version 1.1, http://tools.ietf.org/html/rfc4346/, 2006.

[5] RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2, http://tools.ietf.org/html/rfc5246/, 2008.

[6] RFC 6176: Prohibiting Secure Sockets Layer (SSL) Version 2.0, http://tools.ietf.org/html/rfc6176/, 2011.

[7] Wikipedia: Transport Layer Security, http://en.wikipedia.org/wiki/Transport_Layer_Security.

[8] The Tor Blog: Tor and the BEAST SSL attack, https://blog.torproject.org/blog/tor-and-beast-ssl-attack/, 2011.

[9] Wireshark, http://www.wireshark.org.

[10] The Wireshark Wiki: SSL, http://wiki.wireshark.org/SSL/, 2012.

[11] Citrix: How to decrypt SSL and TLS traffic using Wireshark, http://support.citrix.com/article/CTX116557/, 2008.

[12] sslstrip, http://www.thoughtcrime.org/software/sslstrip/.

[13] dsniff, http://www.monkey.org/~dugsong/dsniff/.

[14] Jeff Beard-Shouse's blog: An introduction to SSL Strip and building a better browser, http://clarkehackworth.com/content/introduction-ssl-strip-and-building-better-browser/, 2010.

[15] Vishnu Valentino Ethical Hacking Tutorial, Security Tips and Trick: Break SSL Protection Using SSLStrip and Backtrack 5, http://vishnuvalentino.com/computer/break-ssl-protection-using-sslstrip-and-backtrack-5/, 2011.

[16] YouTube: SSL Strip, http://www.youtube.com/watch?v=XtaAuhQWvcg, 2009.

[17] sslstrip, http://www.thoughtcrime.org/software/sslstrip/.

[18] Burp Suite, http://portswigger.net/burp/.

[19] 41J Blog: SecurityTube, Wireless Lan Security Megaprimer notes part 13 - SSL Man-In-The-Middle Attacks, http://vishnuvalentino.com/computer/break-ssl-protection-using-sslstrip-and-backtrack-5/, 2011.

[20] YouTube: Wireless LAN Security Part 13 - SSL Man-In-The-Middle Attacks, http://www.youtube.com/watch?v=aB0qdZXfujk, 2012.

[21] Peter Burkholder, *SSL Man-in-the-Middle Attacks*, 2002.

[22] Kristof Boeynaems, *Man-in-the-Middle aanval op het SSL protocol*, 2004.

[23] Brice Canvel, Alain Hiltgen, Serge Vaudenay and Martin Vuagnoux, *Password Interception in a SSL/TLS Channel*, 2003.

[24] David Brumley and Dan Boneh, *Remote timing attacks are practical*, 2003.

[25] Onur Aciicmez, Werner Schindler and Cetin Koc, *Improving Brumley and Boneh Timing Attack on Unprotected SSL Implementations*, 2005.

[26] CNSS Policy No. 15 Fact Sheet No. 1: National Policy on the Use of the Advanced Encryption Standard (AES) to Protect National Security Systems and National Security Information, http://csrc.nist.gov/groups/ST/toolkit/documents/aes/CNSS15FS.pdf, 2003.

[27] Wikipedia: Triple DES, http://en.wikipedia.org/wiki/Triple_DES.

[28] Wikipedia: RC4, http://en.wikipedia.org/wiki/Rc4.

[29] Wikipedia: Data Encryption Standard http://en.wikipedia.org/wiki/Data_Encryption_Standard.

[30] BackTrack Linux, http://www.backtrack-linux.org.

[31] RFC 3268: Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS), http://tools.ietf.org/html/rfc3268/, 2008.

[32] Akamai Technologies, http://www.akamai.com.

[33] wirewatcher: Decrypting SSL traffic with Wireshark and ways to prevent it, http://wirewatcher.wordpress.com/2010/07/20/decrypting-ssl-traffic-with-wireshark-and-ways-to-prevent-it/, 2010.

[34] Google Play: StripGuard, https://play.google.com/store/apps/details?id=com.acis.

[35] Apple App Store: StripGuard, http://itunes.apple.com/us/app/sslstripguard/id510891106?mt=8.

[36] HTTPS Everywhere, https://www.eff.org/https-everywhere/.

[37] YouTube: iPhone Penetration Testing - Man-in-the-Middle of iphone https traffic, http://www.youtube.com/watch?v=q6ShMaUba5Y, 2012.

List of Figures

List of Tables