Abstract

E-commerce and online banking using the Internet infrastructure have considerably increased in the last years. Further, these services are probably the reason why the Internet has gained the importance that it has in our daily lives. In order to provide confidence to use these services in the Internet we need to protect sensitive information travelling through a public network. A number of alternatives are currently used and, in this paper, we focus on the most popular of these mechanisms, namely Secure Socket Layer (SSL). Here, we present a detailed analysis of the specification, understanding its strengths and weaknesses. Further, we enumerate a list of recommendations to be considered when using SSL to secure online transactions. Finally, we discuss the relationship between SSL and Web attacks such as XSS, CSRF and how the former could be potentially used to mitigate these attacks.

Key Words: Public Key Infrastructure, Protocol, Digital Certificate, Symmetric Cryptography, Public Key Cryptography, Cipher, Diffie-Hellman, X.509, HTTP, XSS, CSRF.

1. Introduction

The Internet has become part of our daily lives. We use it for digital communications such as e-mail, messaging, chatting, etc. Further, along with the development of web-based applications, the Internet has provided a convenient alternative to do business across the globe. However, the public nature of the Internet also represents security risks that need to be mitigated. In particular, sensitive information should be protected from disclosure while in transit and it is also important to guarantee that “in the other end” there is the entity which we want to communicate with. Even more, we need to assure that the information received by the target is not modified in transit. These requirements can be summarized as confidentiality, authentication and integrity properties, respectively. Even though, not all the services in the Internet require stringent security requirements, it is important to provide security for the ones that manages sensitive information. In particular, e-commerce is an important economic activity boosted by the Internet in which it is important to protect sensitive information such as credit card numbers and personal customer’s information (i.e. addresses, phone numbers, SSN, etc.). Another common use of the Internet infrastructure is to perform remote log in to corporate information systems (e.g. email, databases, telephone systems, etc.), this also require to validate the identity of the person logging in as well as to protect user credentials from being stolen.

The most common way of achieving these security properties (i.e. confidentiality, authentication and integrity) is by using cryptographic mechanisms. In particular, Secure Socket Layer (SSL) / Transport Layer Security (TLS) are a couple of well known specifications aiming these goals. The current version of SSL is version 3; this version only differs on small details from TLS. Hence, we will focus on SSL and consider that our analysis and recommendations can be applied to TLS as well. The SSLv3 specification itself seems to be completely secure; however, there is evidence that bad configurations and bad assumptions can circumvent the security of SSL. Many people, including “security experts” talk about SSL as the best alternative; however, many of them do not understand the details of it. This fact is risky since a false sense of security can be generated among the internet community. Even more, banks on line and other e-commerce web sites sell themselves as secure because they use SSL to protect customers’ information.
Because of the above, it is important to understand the underpinnings of SSL so that we can use it in the most proper manner. Further, given many of the attacks such as Cross Site Scripting (XSS) and Cross Site Request Forgery (CSRF) aiming web-based applications, in this paper we explore the possibility of using an SSL approach to protect against these attacks. It is important to mention that the success of the future Internet broadly depends on the effectiveness of initiatives such as SSL. As mentioned above, SSL only protects information in transit and hence, it is important to analyze the effectiveness of other alternatives to protect information stored in databases. In this paper we focus on SSL but we believe that protection mechanisms for stored information are equally worth to be analyzed since it is also an important area of research that need to be addressed to achieve a secure Internet.

Our paper is organized as follows: In section 2 we describe SSLv3 in detail with all its variants. In section 3 we include the attacker model and in section 4 we mention some of the weaknesses of the X.509 system. In section 5 we perform an SSL security analysis and enumerate a set of recommendations that can be helpful to protect against SSL attacks. Finally, in section 6 we include some ideas about how to use SSL to protect against Web based attacks. Future work is included in section 7 and we conclude in section 8.

2. SSL analysis

The TCP/IP networking model defines four layers in which communication protocols operate; namely, Link, Internet, Transport and Application layers. If we think of this model to analyze SSL, it is a little bit confusing since we would have to locate SSL operating either at the transport or application layer. Neither of these is true since SSL actually works in between application and transport layers. The OSI networking model defines seven layers and is more suitable to analyze SSL in detail. The OSI model considers the following layers: Physical, Link, Network, Transport, Session, Presentation and Application. Then, SSL can be seen as operating in the session layer. SSL encrypts chunks of application data and then passes on the encrypted data to the transport layer which is in charge of transporting data along the network. It is worth to mention that SSL only works with TCP and hence UDP traffic cannot be secured with SSL. SSL provides three security properties: Confidentiality since it encrypts the application payload. Authentication since it uses Digital Certificates to authenticate the server and can use client Digital Certificates to authenticate clients too. Integrity since it uses Messages Authentication Codes (MAC) to detect corrupted messages. Figure 2.1 shows the basic structure of the SSL specification [1].
It is important to differentiate between two concepts: **Connection** and **Session**. A connection is defined at the transport layer. In SSL, connections are peer-to-peer relationships, these are transient and every connection has to be associated with one session. However, several connections can use the same session and hence there is a place for attacks that we will discuss later on. A session is an association between a client and a server. They define a set of cryptographic security parameters, which can be shared among multiple connections. Sessions are used to avoid the expensive negotiation of new security parameters for each connection.

### 2.1. The SSL handshake protocol

The most important part of the SSL specification is the handshake protocol. This protocol allows server and client to authenticate each other and to negotiate an encryption and MAC algorithm and cryptographic keys to be used to protect the data in SSL sessions. The handshake protocol is depicted in table 2.2. This protocol can be divided in four phases as follows:

Phase 1: Establish security capabilities: Protocol version, session ID, Cipher suite, compression method and initial random numbers.

Phase 2: Server authentication and key exchange: Server might send certificate, key exchange and request certificate. Server also signals end of hello message.

Phase 3: Client authentication and key exchange: If requested, client sends certificate and certificate verification. Client also sends key exchange.

Phase 4: Finish: Change cipher suite and finish handshake protocol.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Messages: Client to Server (C-&gt;S) / Server to Client (S-&gt;C)</th>
<th>Comment</th>
</tr>
</thead>
</table>
| One   | C->S: `Client_hello`  
|       | S->C: `Server_hello` |         |
| Two   | S->C: `Certificate` | Optional. Not needed if the |
Three

<table>
<thead>
<tr>
<th></th>
<th>S-&gt;C: Server_key_exchange</th>
<th>Not necessary if Fixed Diffie-Hellman key exchange mechanism was chosen.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>S-&gt;C: Certificate_request</td>
<td>Optional. In case client authentication is needed.</td>
</tr>
<tr>
<td></td>
<td>S-&gt;C: Server_hello_done</td>
<td></td>
</tr>
</tbody>
</table>

Four

<table>
<thead>
<tr>
<th></th>
<th>C-&gt;S: Certificate</th>
<th>Optional. In case client certificate was required.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C-&gt;S: Client_key_exchange</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C-&gt;S: Certificate_verify</td>
<td>Optional. In case client certificate was sent.</td>
</tr>
</tbody>
</table>

Four

<table>
<thead>
<tr>
<th></th>
<th>S-&gt;C: Change_cipher_spec</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>S-&gt;C: Finished</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.2. The SSL handshake protocol

2.1.1 Phase 1: Establish Security Capabilities

In this phase, client and server interchange hello messages to agree on the protocols that are going to be used. They also interchange a random string including a time stamp to avoid reply attacks. The client hello messages include the following data:

- Highest supported version: The highest version of SSL supported by the client. It might be version 1, version 2, version 3 or TLS. Version 1 was actually never deployed and version 2 is esteemed to be insecure. Therefore, either version 3 or TLS should be used.

- Random string: 32 bit time stamp | | 28 bytes random. These values are used during key exchange to prevent reply attacks.

- Session ID: A variable-length session identifier. A nonzero value indicates that the client wishes to update the parameters of an existing connection or create a new connection on this session. A zero value indicates that the client wishes to establish a new connection on a new session.

- Cipher Suite: List that contains the combination of cryptographic algorithms supported by the client in decreasing order of preference. The cipher suite defines the key exchange mechanism, type of cipher (i.e. stream or block), cipher algorithm (e.g. RC4, DES, IDEA, etc.), MAC algorithm and hash type (MD5 or SHA-1). Cipher suite is analyzed in more detail in the next section.

- Compression method: List of compression methods supported by the client.

If the server supports the highest version of SSL requested by the client, it sends back the following hello messages:
• Random string: 32 bit time stamp | 28 bytes random. These values are used during key exchange to prevent reply attacks.

• Session ID: If the client sent a zero value. The server replies with a variable-length session identifier to be used in this session.

• Cipher Suite: Cipher suite chosen by the server.

• Compression method: Compression method chosen by the server.

2.1.1.1 Cipher Suite

SSL uses both symmetric and asymmetric cryptography. Asymmetric cryptography is used to interchange symmetric keys that are going to be used later on to perform message encryption. The SSL specification considers four key exchange mechanisms as follows:

• RSA: Encrypt symmetric key (generated by the client) using server’s public key.
• Fixed Diffie-Hellman (FDH): Server’s public key certificate contains public DH key.
• Ephemeral Diffie-Hellman (EDH): Public key is used to sign temporary DH key.
• Anonymous Diffie-Hellman: DH without authentication. Note that the use of this mechanism is clearly vulnerable to Man in the Middle attacks (MitM).

In appendix A we include all the messages required in the SSL handshake protocol for each of these key exchange mechanisms.

The SSL specification can be used with a variety of cipher and hash algorithms as follows:

• Cipher Algorithm: RC4, RC2, DES, 3DES, IDEA, AES.
• MAC Algorithm: MD5, SHA-1, SHA-256
• Cipher Type: Stream or Block
• Exportable: True or False
• Hash Size: 0 or 16 for MD5, 20 for SHA-1

It is evident that some of the algorithms supported by the SSL specification are vulnerable to brute force attacks. Further, it is worth to notice that a hash size of 0 bytes does not provide any security. Because of that, it is of paramount importance to properly configure server and client to support only robust ciphers and hash algorithms.

2.1.2 Phase 2: Server authentication and key exchange

After phase 1 is completed, both client and server know which cipher and key exchange mechanisms to use. Then, in phase 2, the server is authenticated and, if needed, it requests the client to authenticate with a digital certificate. The messages in this phase are as follows:
S->C: Server’s certificate is sent to the client. The certificate might include a sign-only public key. In this case, an encryption public key would have to be sent later on in the server_key_exchange message.

S->C: Server key exchange. Depending on the key exchange mechanism used, the server can send an anonymous or authenticated DH key (Ys) or an authenticated RSA key using its sign only private key. For your convenience, here we include how a DH key is generated:

\[ Y_s = a^{x_s} \mod q \]

Where: \( q \) is a prime number; \( a < q \) and a primitive root of \( q \); \( x_s \) is the discrete logarithm of \( Y_s \) for the base \( a \).

S->C: If needed, the server request a client certificate, this can be used later on to authenticate the client. Client certificates are usually not used since application level client authentication mechanisms are usually in place. Nevertheless, client certificates are desirable when SSL is used to remote log in from a public network (i.e. Internet) into a corporate private network. This provides an additional level of security. The message contains the type of client digital certificate required (e.g. RSA or DSS for key exchange) and a list of acceptable certification authorities.

The final message on phase 2 is the server_hello_done message. This message is basically a signature on the previous interchanged messages, as follows:

\[ server\_hello\_done = \{\text{hash(Client\_rand||Server\_rand, data)}\}^{PKs^{-1}} \]

Where \( PKs^{-1} \) is the server’s private key.

Given that the messages on phase one were sent in clear text and hence are prompted to be modified by the attacker. The server_hello_done message serves to different purposes: 1. Guarantee that the previous messages were not modified; 2. Authenticate the server and, 3. Avoid reply attacks.

2.1.3 Phase 3: Client authentication and key exchange

In this phase the client sends its digital certificate if it was requested by the server in the previous phase. Depending on the key exchange mechanism, the client sends either a 48 bytes pre-master secret (PS) encrypted with server’s public key or a public DH key.

In case a digital certificate was requested, the client has to send a certificate_verify message to validate the digital certificate authenticity as follows:

\[ certificate\_verify = MD5(pre - master\_secret||pad2)||MD5(Handshake\_messages \|pre - master\_secret||pad1)) \]

2.1.4 Phase 4: Finish
After phase 3, client and server share the pre-master secret and authenticated each other. In phase 4, client and server notify each other that they have set up the cipher specifications and are ready to start interchanging application data. Two messages are sent by both client and server in this phase:

*Change_cipher_spec:* Tells each other that they successfully set up the appropriate ciphers and keys.

*Finished*

\[
= \text{MD5}(\text{master secret} \ || \text{pad2} \ || \text{MD5}(\text{handshake messages} || \text{Sender} || \text{master secret} || \text{pad1})) || \text{SHA} - 1(\text{master secret} || \text{pad2} || \text{SHA} - 1(\text{handshake messages} || \text{Sender} || \text{master secret} || \text{pad1}))
\]

Where:

\[
\text{pad1} = 0x363636 \ldots
\]
\[
\text{pad2} = 0x5C5C5C \ldots
\]

The *finished message* verifies that the key exchange and authentication process were successful. Each party verifies the received message with the generated message to guarantee that client and server can securely communicate each other.

The integrity of all the previous messages is also verified since the hash includes all the handshake messages previously exchanged.

### 2.1.4.1 The master secret and key generation

The master secret (MS) is generated by using the premaster secret (PS) and the client and server random strings (CR and SR), as follows:

\[
\text{MS} = \text{MD5}(\text{PS} || \text{SHA-1}(\text{‘A’} || \text{PS} || \text{CR} || \text{SR})) \ || \text{MD5}(\text{PS} || \text{SHA-1}(\text{‘BB’} || \text{PS} || \text{CR} || \text{SR})) \ || \text{MD5}(\text{PS} || \text{SHA-1}(\text{‘CCC’} || \text{PS} || \text{CR} || \text{SR}))
\]

This construction of the Master Secret protects against collision attacks and is cryptographically secure.

Once the master secret is generated it is used to further generate client and server’s MAC keys, encryption keys and Initialization Vectors (IV) for block ciphers. These values are derived by taking the appropriate number of bits out of the string showed below:

\[
\text{MD5}(\text{MS} || \text{SHA-1}(\text{‘A’} || \text{MS} || \text{SR} || \text{CR})) \ || \text{MD5}(\text{MS} || \text{SHA-1}(\text{‘BB’} || \text{MS} || \text{SR} || \text{CR})) \ || \text{MD5}(\text{MS} || \text{SHA-1}(\text{‘ CCC’} || \text{MS} || \text{SR} || \text{CR})) \ || \text{MD5}(\text{MS} || \text{SHA-1}(\text{‘DDDD’} || \text{MS} || \text{SR} || \text{CR})) \ || \ldots
\]

It is important to mention that MAC and encryption keys are used depending on the application messages’ direction. That is, when the client sends application data to the server, a MAC and encryption key tuple is used and, when the server sends application data to the client, a different MAC and encryption keys are used.
2.1.5 The SSL resource record

Once the handshake protocol is finished, client and server are ready to exchange application data. The SSL records carrying application data looks like:

\[
SSL \text{ Record} = \text{Content type}||\text{Major version}||\text{Minor version}||\text{Length}||\{\text{Data}||\text{MAC}(K',\text{Data})\}_K
\]

Where:

\[\text{MAC}(K',\text{Data}) = \text{MD5}\left(K'||\text{pad2}||\text{MD5}(K'||\text{pad1}||\text{seq_num}||\text{compressed_type}||\text{length}||\text{Data})\right)\]

3. The attacker model

When analyzing the security of any scheme it is important to consider the skills and information that an attacker trying to attack it can have or acquire. In particular we consider a strong attacker who controls the communication channel. That is, she is able to sniff the network, drop and modify messages in transit as well as to craft and inject malicious messages. Therefore, the attacker has all the skills and information to launch a man in the middle attack (MitM).

Furthermore, the attacker has enough computational power to perform brute force attacks against weak ciphers. Even more, the attacker can inject malicious code on the victim’s system. Malicious code includes spyware and malicious javascripts. The attacker can also launch MD5 collision attacks and can ask to a Certification Authority (CA) for a digital certificate. On the other hand, the attacker cannot force a user to accept an invalid certificate but can certainly try to deceive users. We also assume that CAs are responsible to validate the identity of the entities asking for a digital certificate.

4. The X.509 system

We need to validate the authenticity of domain names by using digital certificates. If the digital certificate of a domain name (e.g. google.com, citibank.com, etc.) is signed by a trusted certification authority (CA), we believe that the public key in the digital certificate really belongs to the domain name.

Before asking for a digital certificate, a domain name should be registered in the DNS, once we have a public identity (domain name), we can ask (and pay) to the CA to sign our certificate and attest for our identity.

Because of the above, a digital certificate should not be issued if the domain name was not previously registered in the DNS. Once the CA validates the existence of the domain name in DNS by querying the WHOIS database, it links domain name, and public key in a digital certificate. In sum, the DNS provides a public identity (domain name) and the certificate helps other parties to validate your identity in the Internet (authentication).

The X.509 is a standard for Public Key Infrastructures that defines format for Digital Certificates, certification path validation algorithms and methods for certificate revocation list (CRL) implementations.

---

1 Or SHA-1
When someone (e.g. a server) wants to be authenticated in the internet (i.e. wants his identity to be validated), he requests a digital certificate (DC) to a Certification Authority (CA). The DC links user (server) identity (i.e. canonical name) with the server public key. The CA signs with his private key the DC, this signature can later be verified by any other party.

Then, he distributes his DC to the parties that want to authenticate him. These parties can authenticate him by verifying the digital signature of the certificate using the public key of the CA. The magic here is that public keys of CAs (a.k.a. root keys) are implicitly trusted because are supposed to be retrieved by the client through a secure mechanism.

Under certain circumstances, a user with a Digital Certificate can delegate his authority by issuing further digital certificates (child certificates). Therefore, a hierarchy can be created and X.509 also defines the mechanism to verify the authenticity of child certificates by traversing the certification hierarchy from the child to the root.

When a private key is compromised and a Digital Certificate has to be revoked, X.509 also defines the mechanism to invalidate these certificates.

### 4.1 X.509 drawbacks and possible fixes

X.509 cannot exclude untrusted CAs: Given the fact that there are many Certification Authorities, it becomes difficult to guarantee that actually all of them follow adequate procedures to issue certificates. If one of them does issue certificates without validating that the name really exists in the DNS, all the security is compromised. That is, any user could request a digital certificate for a name that does not exists or, even worst, for a name that belongs to another party. Therefore, the security of the whole X.509 system is compromised because the security of the system is equal to the security of the weakest link.

X.509 Certificates are not designed to allow delegation: Only under particular circumstances a digital certificate can be issued to issue child certificates; however, this is usually the exception and not the rule. This prevents the system from escalating and limits scope of authentication.

To deal with the exclusion problem and avoid trusting unknown Certification authorities, we can run our private CA. However, this is expensive and difficult to maintain. Further, cross authentication between different organizations is painful and might cause disclosure of confidential information. Another, solution is to reduce the number of CAs, regulate them and validate that security procedures are implemented and followed.

To be able to delegate self-signed certificates can be used; however, a self-signed certificate can sign any other certificate thwarting the purpose of digital certificates. Name constrains are also supposed to be useful to allow delegation of certificates. For example, when getting a certificate for “mycompany.com” this certificate was supposed to be useful to sign sub domains such as “department.mycompany.com”; however this is usually not the case.

One of the main problems of the X.509 system is that there are out there many certification authorities that can issue digital certificates to almost anyone that can pay for them. Further,
many browsers have installed a bunch of root certificates that users cannot verify to be valid. The exclusion problem mentioned by Dan Kaminsky [2] is a reality and should be addressed in the X.509 system. The fact that DNSEC propose a single root (inherited from DNS) generates confidence in the system, it brings out other security issues, though.

Regarding the delegation problem, the design of DNSEC is better to allow delegation, X.509 really limits the delegation of certificates. There are many weaknesses associated with the use of X.509 certificates, in particular its design allows attacker to generate bogus certificates.

5. SSL Security Analysis

In the previous sections we have presented all the details of the SSL specification, including the X.509 system, which is an important and sensitive part of SSL. Also, in appendix A, we show in detail all the variants of the handshake protocol. In this section we discuss some important points associated with the security of SSL.

In Section 2 we showed how SSL can work with different cipher suites. An important security consideration is that some of the ciphers that can be used in SSL are known to be currently broken. Therefore, it is important to configure server and client to use a strong cipher (i.e. 3DES or AES). If an attacker successfully inject malicious code in the client enabling the use of ciphers such as RC2, RC4, DES, etc; she could force the use of these ciphers between client and server and hence rise her chances to break the security by performing brute force attacks. In a similar manner, the attacker could force the use of a non secure version of SSL (i.e. SSLv2) or the use of insecure hash functions such as MD5. In order to mitigate these attacks; it is important to configure and make sure that the server only accepts the use of strong ciphers such as 3DES or AES and SSLv3 or TLS.

In previous sections, we also showed the four key exchange mechanisms supported by the SSL specification; namely, RSA key exchange, Fixed Diffie-Hellman, Ephemeral Diffie-Hellman and Anonymous Diffie-Hellman. Each of these schemes has strengths and weaknesses as mentioned below.

RSA: In this method the client randomly generates the pre-master secret, encrypts it with the server’s public key and then client and servers use it to generate the master secret. The risk involved in this scheme is that an attacker could have been compromised the client and hence force it to generate a weak pre-master secret (e.g. a stream of 0’s). If this is the case the security of the SSL is completely broken.

Fixed Diffie-Hellman: In this method server and client have a digital certificate with their own public Diffie-Hellman keys. Then, they interchange their DC containing the Diffie-Hellman keys; which can be validated. This scheme is secure as long as the private part of the Diffie-Hellman keys are kept secret. However, the problem with this scheme is the fact that keys are never changed. If the attacker can figure out the private part of the keys, for example by launching a brute force attack; she will be able to decrypt further and previous (assuming she captured the encrypted packets when they were originally sent) communications being encrypted with those keys.
Anonymous Diffie-Hellman: This scheme is completely insecure since it is vulnerable to man in the middle (MitM) attacks. Since the Diffie-Hellman public keys are not authenticated. An attacker can set up secure communications with the client and server, locating herself in the middle. For details about this attack, see appendix A.

Ephemeral Diffie-Hellman: We consider this scheme to be the most secure. Public Diffie-Hellman keys are randomly generated by client and server each time a new session is set up. Further, the server DH key is signed with the server’s private key and hence can be verified by the client. Even more, if the client also uses a digital certificate, the server can also validate the authenticity of the client’s DH key. Even if the client does not use a digital certificate this scheme is secure since a MitM attack cannot be successfully performed (i.e. an attacker cannot share a different pre-master with the client and server without the client noticing it).

Besides the security considerations associated with the key exchange mechanism; there are many other issues that we need to consider when evaluating the security of SSL. In particular, SSL security relies on a secure X.509 system as well as user behavior. These assumptions are mentioned in the following section.

5.1 SSL assumptions that need to be hold to guarantee the scheme remains secure

Certification authority assumptions:

- Certification Authority root key is secure
- CAs do not to issue bogus digital certificates
- All CAs verify the identities of domain’s owner previous to issue a digital certificate
- CAs use strong encryption and hash algorithms

Cryptographic assumptions:

- All crypto algorithms (i.e. hash functions and ciphers) are secure

Browser assumptions:

- All root certificates in browser are correct
  - The browser can update the list of valid certificates upon certificate’s revocation
  - The attacker has not added a bogus certificate to the victim’s browser
  - The browser was securely downloaded and installed
  - Root certificate in browser has not been altered by malicious software
- Browser does not contain any remotely exploitable vulnerability
- Malicious site cannot overwrite lock and https with bogus images
- No Unicode or homograph vulnerabilities can be exploited

User assumptions:

- User verifies the yellow pad, https and lock closed when using SSL
- User cannot be tricked into bogus SSL
- User does not accept bogus certificate
- User does not ignore warnings about bogus / mismatches of digital certificates
Because of the above, it is of paramount importance to properly configure servers to use SSL and take appropriate measures to guarantee the authenticity of digital certificates in which the browser rely on. Furthermore, users should be educated to not accept bogus certificates, validate certificates and prevent their clients from being infected with malicious software that can alter the list of trusted certificates or inject malicious java script that can steal users credentials or redirect users to malicious sites.

6. Web Security and SSL

Given the stateless nature of the http protocol, attacks such as Cross Site Scripting (XSS) and Cross Site Request Forgery (CSRF) can introduce security holes by circumventing the same origin policy. For example, a XSS attack can execute a malicious script on the clients to steal authentication cookies, then, use that cookie to authenticate with the server using the victim’s identity. On the other hand, a CSRF attack send a malicious crafted request through a client’s browser by using an already set up session between that client and a sensitive web service.

These kinds of attacks rely on the stateless nature of the http protocol. The main problem is that in order to maintain the state of different sessions, servers use cookies. Therefore, once a client has successfully logged in, the following messages are supposed to be valid as long as the packets include the corresponding cookie or session ID.

Therefore, when using SSL to protect communication, we suggest two things: use non-persistent http connections and establish a new session (i.e. run the handshake protocol) each time a new sensitive connection is established. This also means that a single session is associated with a single connection. Clearly, the problem with this approach is that it deteriorates the communication performance.

The second approach suggests the use of SSL and TCP to enforce the same origin policy. Each connection between a browser and a server is uniquely identified by a tuple of source port and destination port. Further, each single connection can be identified by an SSL session ID. Hence we propose a slightly modification of SSL such that it can use the src/dst port tuple and session ID to filter requests coming from a different TCP connection / SSL session. This scheme is similar to the one used by traditional operating systems. In this case, access to sensitive resources is controlled by the kernel based on user’s privileges and many other configurable policies.

We believe that the control of http request performed at the session layer by SSL can effectively mitigate attacks aiming web applications such as CSRF or XSS.

7. Conclusions

In this paper we have presented a detailed description of the SSL specification. We did a deep analysis of the different schemes that can be used by SSL. Further, we included a security analysis of it and highlighted the most important assumptions that need to be hold to guarantee that SSL remains secure. In addition, along the document we mentioned several recommendations about how to effectively use SSL to secure communications through the
Internet. We also discussed the X.509 scheme and highlighted some of the vulnerabilities of this system.

Finally, we suggested two approaches to use SSL in order to protect from attacks launched against web applications. In particular, the first alternative suggests the use of non-persistent http and unique session ID to avoid session ridings. The second alternative proposes a “security kernel” like scheme in which SSL uses TCP source/destination ports as well as session IDs to prevent unsecure connections to issue requests over already established secure connections.

We consider that our analysis is helpful to understand the risks associated with the use of SSL and motivate people and experts to be more cautious when using this specification. Therefore, we arose awareness among the Internet community. Furthermore, our proposal to mitigate XSS and CSRF attacks by using SSL is valuable since it does not require a lot of changes in the SSL specification and can help to solve these kinds of problems.

Addressing these security issues in the Internet is of paramount importance to guarantee the success of the future Internet.

References

[3] ICANN Proposal to DNSEC-sign the Root Zone

Disclaimer

Some ideas of this paper were taken or complemented from the Network Security Course taught by Adrian Perrig at Carnegie Mellon University. Our main contributions are the detailed analysis of all the SSL variants, security recommendations, the X.509 considerations and Web-based applications analysis.
APPENDIX A: SSL handshake protocol for different key exchange mechanisms

A.1. Handshake protocol with RSA Key exchange mechanism. (Server uses a DC with a sign-only key.

1. \( C \rightarrow S: \text{Client}_\text{hello} \)
2. \( S \rightarrow C: \text{Server}_\text{hello}: \)
   - RSA key exchange
   - AES Encryption
   - SHA-1 based MAC
3. \( S \rightarrow C: \{\text{Server name, } PK1_s\}_{Kca^{-1}}, \text{Server name, } PK1_s \)
4. \( S \rightarrow C: \{PK2_s\}_{PK15^{-1}}, PK2_s \)
5. \( S \rightarrow C: \text{Server}_\text{hello done} \)
6. \( C \rightarrow S: \{\text{Pre}_\text{master secret}\}_{PK2s} \)
7. \( C \rightarrow S: \text{change}_\text{cipher}_\text{spec} \)
8. \( C \rightarrow S: \text{finished} \)
9. \( S \rightarrow C: \text{change}_\text{cipher}_\text{spec} \)
10. \( S \rightarrow C: \text{finished} \)

Where:

- \( PK1_s \): Server Public Key used only to verify server signatures.
- \( Kca^{-1} \): Certification Authority private key.
- \( PK2_s \): Server Public Key used to perform encryption.
- \( PK15^{-1} \): Server private key used to sign.

Note: In case the server use a DC with an encryption key the message 4 is not needed and the client can simply use \( PK1_s \) to encrypt the pre-master secret.

A.2. Handshake protocol with Ephemeral DH Key exchange mechanism and client certificate required.

1. \( C \rightarrow S: \text{Client}_\text{hello} \)
2. \( S \rightarrow C: \text{Server}_\text{hello}: \)
   - Ephemeral DH key exchange
   - AES Encryption
   - SHA-1 based MAC
3. \( S \rightarrow C: \{\text{Server name, } K_s\}_{Kca^{-1}}, \text{Server name, } K_s \)
   
   \text{Note: Upon receiving message 3, the client can verify the integrity of Server name and } PK_s \text{ by using the Certification Authority Public Key.}
4. \( S \rightarrow C: \text{Server Key exchange: } g, p, g^x, \{H(g,p,g^x)\}Ks^{-1} \)
Note: In ephemeral DH, the DH key \( (g^s) \) is randomly generated each time a new session needs to be established.

5. \( S \rightarrow C: \) Certificate request
6. \( S \rightarrow C: \) Server_hello done
7. \( C \rightarrow S: \) \{\( \text{client name}, K_c \}_{K_{ca}^{-1}}, \text{Client name, } K_c \)

Note: Upon receiving message 6, the server can verify the integrity of Client name and \( PK_c \) by using the Certification Authority Public Key.

8. \( C \rightarrow S: \) client key exchange: \( g, p, g^c, \{H(g,p,g^s)\}_{K_{c^{-1}}} \)
9. \( C \rightarrow S: \) Certificate verify

Therefore, both parties can calculate the premaster secret \( g^s = g^c \) as follows:

Symmetric key calculated by the server: \( (g^c)^s \)
Symmetric key calculated by the client: \( (g^s)^c \)

10. \( C \rightarrow S: \) change_cipher_spec
11. \( C \rightarrow S: \) finished
12. \( S \rightarrow C: \) change_cipher_spec
13. \( S \rightarrow C: \) finished

A.3. Handshake protocol with Fixed DH Key exchange mechanism and client certificate required.

1. \( C \rightarrow S: \) Client_hello
2. \( S \rightarrow C: \) Server_hello:
   - Fixed DH key exchange
   - AES Encryption
   - SHA-1 based MAC

3. \( S \rightarrow C: \) \{\( g^s, p, \text{Server name, } PK_s \)\}_{K_{ca}^{-1}}, g^c, g, p, \text{Server name, } PK_s \)

Note: Upon receiving message 3, the client can verify the integrity of \( g^c, g, p, \text{Server name, } PK_s \) by using the Certification Authority Public Key.

4. \( S \rightarrow C: \) Certificate request
5. \( S \rightarrow C: \) Server_hello done
6. \( C \rightarrow S: \) \{\( g^c, g, p, \text{client name, } PK_c \)\}_{K_{ca}^{-1}}, g^c, g, p, \text{Client name, } PK_c \)

Note: Upon receiving message 6, the server can verify the integrity of \( g^c, g, p, \text{Client name, } PK_c \) by using the Certification Authority Public Key.

7. \( C \rightarrow S: \) client key exchange: NULL
8. \( C \rightarrow S: \) Certificate verify
9. \( C \rightarrow S: \) change_cipher_spec
And both parties share the pre-master secret calculated as follows:

By the server: \((g^c)^s \mod p\)
By the client: \((g^s)^c \mod p\)

A.4. Handshake protocol with Anonymous DH Key exchange mechanism.

1. C \(\rightarrow\) S: Client_hello
2. S \(\rightarrow\) C: Server_hello:
   - Anonymous DH key exchange
   - AES Encryption
   - SHA-1 based MAC
3. S \(\rightarrow\) C: \(\{\text{Server name}, K_s\}_{K_{ca}} , \text{Server name}, K_s\)
   
   Note: Upon receiving message 3, the client can verify the integrity of Server name and PK_s by using the Certification Authority Public Key.
4. S \(\rightarrow\) C: Server Key exchange: g, p, g^s
   
   Note: Because DH key is not authenticated, an attacker can catch message 4 and send a fake message g, p, g^A
5. S \(\rightarrow\) C: Certificate request
6. S \(\rightarrow\) C: Server_hello done
7. C \(\rightarrow\) S: \(\{\text{client name}, K_c\}_{K_{ca}} , \text{Client name}, K_c\)
   
   Note: The attacker only hast to pass these messages without altering them. Upon receiving message 6, the server can verify the integrity of Client name and PK_c by using the Certification Authority Public Key.
8. C \(\rightarrow\) S: client key exchange: g, p, g^c
   
   Note: Because DH key is not authenticated, an attacker could have catch message 8 and send a fake message g, p, g^A
9. C \(\rightarrow\) S: Certificate verify
Therefore, the attacker could have performed a man in the middle attack sharing different symmetric keys with each party as follows:

Symmetric key shared with the client: \((g^c)^A\)
Symmetric key shared with the server: \((g^s)^A\)