Chapter 2
Background

2.1 Application Layer Attacks

Application protocols like HTTP, SMTP, POP3, DNS, IMAP and FTP are well known services. These protocols were defined in individual RFC for communication with each other. Application firewall should realize their features and identities these behaviors in order to inspecting traffic. Deep inspection is kind of mechanism which works as state machine that examines the content in the application layer. The inspect rules are pattern format, string length, or parameter types. If there are any conflicts, firewall can take actions that record as log or signal warnings [10].

Today, more critical mission applications and data pass through the HTTP Protocol. Unfortunately, many of the features make browsers so convenient also make them incredibly insecure. It is often relatively easy for a hacker to find and change hidden fields that indicate a product prices, a hacker can change the parameters of a CGI script to search for a password file instead of a product price. If some components of a web application (such as search functionality) are not integrated and configured correctly, the site could be subject to buffer-overflow attacks that could grant a hacker access to administrative pages. Current web application coding practices largely ignore some of the most basic security measures required to keep a company and its data safe from unauthorized access.

Regarding to well known GPL IDS named as Snort [11]. It features rules based logging to perform content pattern matching and detect a variety of attacks and probes, such as buffer overflows, stealth port scans, CGI attacks, SMB probes, and much

more. According to its rules approximately 1103 rules are collected which prefix have "web" in total 3201 rules. If SQL rules, backdoor rules, and all the other rules directly associated with web protocol are counted together. As a result, more than 2000 rules are under suspicion, which could be threat to server or client through web traffic.

Not only predefined suspicious activity or patterns across around Internet and exploit these weaknesses, several widely spread worms, Code Red [12], Blaster [13], Nimda [14], and SQL Slammer [15], have caused substantial damages.

Lots of organizations are established as helping users understand the typical web security profiles. Open Web Application Security Project (OWASP, www.owasp.org) [16] is an open source reference point for system architects, developers, vendors, consumers and security professionals involved in the security of web applications and web services. Based on [17], the most common vulnerabilities are listed and described in the following.

Buffer Overflow

Buffer overflow vulnerability is one of the most common security flaws. It is an attack technique that overruns the memory. For instance, an application might accept ten-digit numbers, if twenty-digit numbers enters, the server often result in a core dump, which reveals information about the memory of the web server. Even more threatening, a hacker who disables the application could upload code to be executed by the server. Viruses such as Code Red, Slammer are results of buffer overflow attacks.

Cookie Poisoning Attack

Cookies are introduced to allow session management and are heavily used by

most Web applications. Cookie poisoning is a kind of attacks, which alters the value of a cookie on the client side prior to a request to the server. Malicious users could also change cookies by either using an interception proxy or directly modifying a file on a local storage to falsify identity, bypassing server's authentication. It makes possibly that the hacker to impersonate authenticated user and review any and all information that the authenticated user would review. Shoplifting is one of the most well known Cookie poisoning attacks.

Cross Site Scripting (XSS)

XSS is a special form of invalidated input attack. A hacker forces a Web server to serve JavaScript that was not sourced from the Web site administrators. The most common place to execute a cross-site scripting attack is a bulletin board or auction posting, where users can submit comments to be viewed by other users. This malicious JavaScript can be used to steal a user's cookies and even compromise a user's computer. Attacker's goal is to steal cookies from the client, then impersonate the client.

SQL Injection

SQL Injection is a technique where an attacker creates, alters, or inserts existing SQL commands to gain access to unintended data, or to gain the ability to execute system level commands on the host. Since most applications simply translate the form data into a SQL query to the application's database, this activity can expose and cause unintended behavior by the application. The hacker inputs SQL commands into web page forms or parameters. The attacker may be able to run any SQL commands on your database that may lead to compromise of the database server.

Parameter Tampering

Conceptually, parameters are included in the URL strings send client-specific information to the Web service so that a certain remote operation can be executed. Parameter tampering involves manipulating URL strings and modifying parameters to retrieving unauthorized information or changing the data. A classic example of parameter tampering is to change values in form fields. These values can be free-text or numbers in text-box or hidden field. An attack may tamper these values such as user account or prices.

A few above examples of the types of attacks are introduced that the professional hackers may attempt to use. Undoubtedly, application-level attacks would lead to issues of expense, manageability, performance, and even down-time. But, there was no viable alternative and business continuity had to be ensured, so complicated security systems were implemented. The systems continue to evolve and grow increasingly complex as components are added. The network detection systems are suitable and worth to be deployed considering their security benefits.

Intrusion detection is a security technology that attempts to identify and isolate intrusions against computer systems. Different IDSes have differing classifications of intrusions. For some IDSes handle URL attacks, they must inspect the HTTP URL field for malicious attacks by using two most popular inspection methodologies, pattern matching and protocol analysis.

2.2 Secure Socket Layer Protocol

The Secure Socket Layer protocol, in short, is intended to provide a practical, application-layer, widely applicable connection oriented mechanism for Internet client/server communications security. The IETF Transport Layer Security working

group is also using SSL 3.0 as a base for their standards efforts [9] [18]. The SSL protocol is designed to prevent eavesdropping, tampering, or message forge. To obtain these objectives it uses a combination of public key and symmetric key cryptography algorithm and digital certificates (X.509). The detailed description of the protocol can be found in [8].

The SSL protocol stack was shown as Figure 2 and fundamentally has two phases of operation: SSL record protocol and SSL handshake protocol. Layered above the record layer is the SSL handshake protocol, a key exchange protocol initializes and synchronizes cryptographic state at the two endpoints. After the key-exchange protocol completes, sensitive application data can be sent via the SSL record layer.

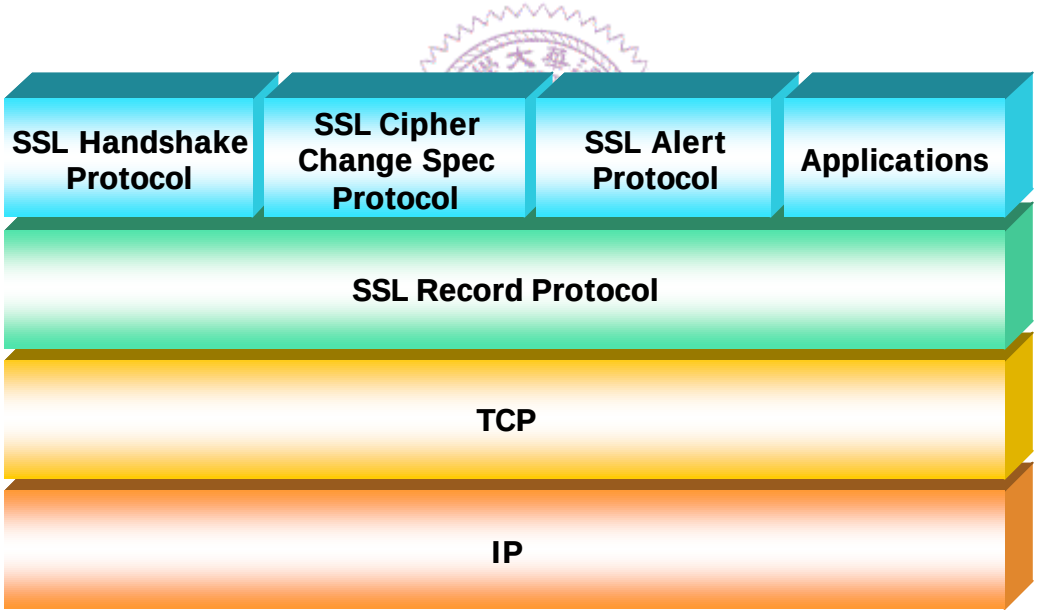

Figure 2. The SSL protocol stack

A. SSL Record Protocol

The SSL record layer provides confidentiality, authenticity, and replay protection over a connection-oriented reliable transport protocol such as TCP. It also

decomposes large messages into fragments of size at most 16 KB in order to facilitate message authentication. All records are compressed using the compression algorithm defined in the current session state and protected using the encryption and MAC (Message Authentication Code) algorithms defined in the current *CipherSpec*. Finally encryption and MAC functions translate compressed units to encrypted data, ready to be sent into TCP packet. This detail process of SSL record protocol is shown in Figure 3.

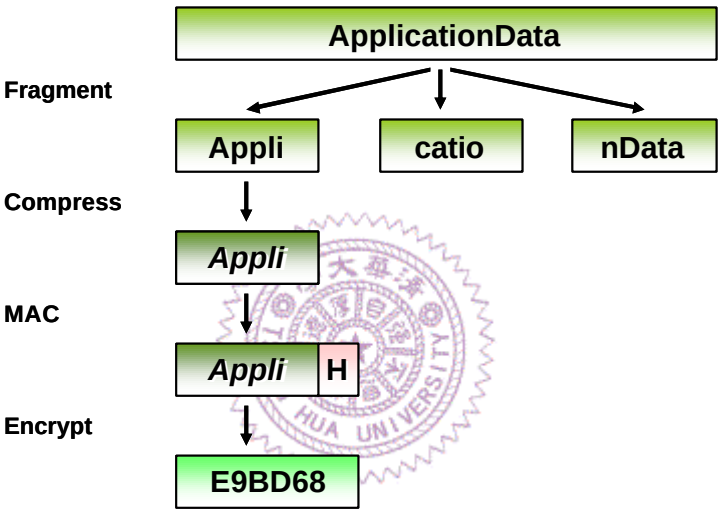

Figure 3. SSL Record Protocol.

B. SSL Handshake Protocol

The handshake allows the server to authenticate itself to the client using public-key techniques like RSA, and allows the client and the server to cooperate in the creation of symmetric keys used for rapid encryption, decryption, and tamper detection during the session that follows. Optionally, the handshake also allows the client to authenticate itself to the server.

SSL negotiation is a nine-step process. They are described as follow.

1. The client sends the server a *Client Hello* message. This hello message contains the SSL version and the cipher suites the client can talk. The client sends its maximum key length details at this time.
2. The server returns the *Server Hello* message with one of its own in which it nominates the version of SSL and the ciphers and key lengths to be used in the conversation, chosen from the choice offered in the *Client Hello* message.
3. The server sends its digital certificate to the client for inspection. Most modern browsers automatically check the certificate (depending on configuration) and warn the user if it's not valid. By valid we mean if it does not point to a certification authority that is explicitly trusted or is out of date, etc.
4. The server sends a *Server Done* message as it has concluded the initial part of the setup sequence. If the server claims to authenticate the client, then sends a *Certificate Request* after sending its own certificate optionally.
5. The client generates asymmetric key and encrypts it using the server's public key (cert). It then sends this message to the server.
6. The client sends a *Change Cipher Spec* message telling the server all future communication should be with the new key. Optionally, it provides its certificate to the server. On the other hand, the client sends a *Certificate Verify* message in which it encrypts a known piece of plain text using its private key. The server uses the client certificate to perform decryption operation. Then ascertaining the client has the private key.
7. The client now sends a *Finished* message using the new key to determine if the server is able to decrypt the message and the negotiation was successful.
8. The server sends a *Change Cipher Spec* message telling the client that all future communications will be encrypted.

9. The server sends its own *Finished* message encrypted using the key. If the client can read this message then the negotiation is successfully completed.

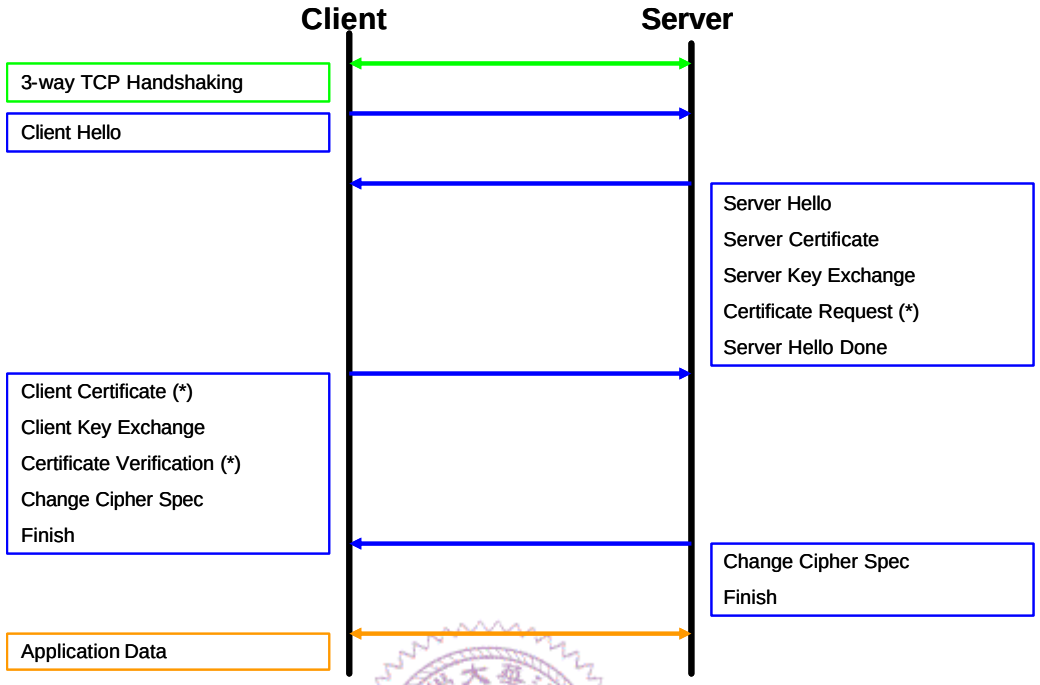

Figure 4. SSL Full Handshake.

Two different handshake types can be distinguished: The full handshake and the resume handshake. The full handshake is negotiated when a client establishes a new SSL connection with the server, and requires the negotiation of the SSL handshake. The resume handshake is negotiated when a client establishes a new HTTP connection with the server but using an existing SSL connection. As the SSL session ID is reused, the part of the SSL handshake negotiation can be avoided.

B.1. Full Handshake

The client sends a *Client Hello* message to which the server must respond with a *Server Hello* message. The *Client Hello* and *Server Hello* messages establish the following attributes: protocol version, session ID, cipher suite, and compression method. Additionally, two random values are generated and exchanged. Following the

hello messages, the server will send its certificate. If the server is authenticated, it may request a certificate from the client. Now the server will send the *Server Hello Done* message, indicating that the hello message phase of the handshake is complete. The *Client Key Exchange* message is now sent. At this point, a *Change Cipher Spec* message is sent by the client and then immediately sends the *Finished* message. In response, the server will send its own *Change Cipher Spec* and *Finished* message. At this point, the handshake is complete and the client and server may begin to exchange application layer data. Figure 4 shows its full state and the star mark indicates optional parameters.

B.2. Resume Handshake

The client sends a *Client Hello* message using the session ID of the session to be resumed. The server then checks its session cache for a match. If a match is found, and the server is willing to re-establish the connection under the specified session state, it will send a *Server Hello* message with the same session ID value. At this point, both client and server have to send *Change Cipher Spec* messages and proceed directly to *Finished* messages. Once the re-establishment is complete, the client and server may begin to exchange application layer data. If a session ID match is not found, the server generates new session ID and the SSL client and server still perform a full handshake. In [19], it shows that reusing cached SSL session keys can significantly reduce client response time. Figure 5 shows the state of resume handshake.

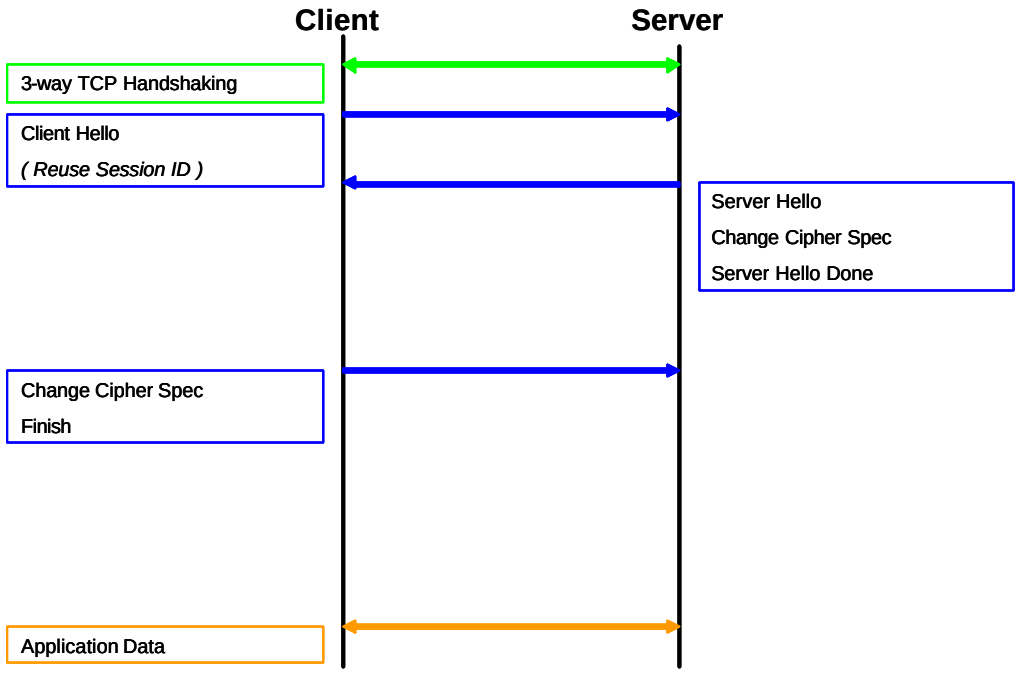

Figure 5. SSL Resume Handshake.

A newer version of SSL protocol is named as Transport Layer Security Protocol (TLS) [20]. The differences between SSL 3.0 and TLS are negligible. The SSL 3.0 protocol was slightly modified by an IETF workgroup and finally standardized in 1999 as TLS 1.0. Although TLS and SSL have subtle implementation differences, application developers usually notice very little difference, and end users should see no difference at all. TLS 1.0 and SSL 3.0 are not, however, interoperable. The most significant difference is that TLS requires certain encryption algorithms that SSL does not. A TLS server must “back down” to SSL 3.0 to interoperate with SSL-3.0 clients. For the purpose of simplicity refer to both SSL and TLS as SSL in this study. A more complete treatment of these protocols can be found in [20] [21] [22].

2.3 Attacks behind HTTPS

The most common services that put on the secure socket layers are HTTP and

HTTPS. The HTTPS protocol performs all HTTP secure communications between a browser and a web server.

HTTPS is the secure version of HTTP based on SSL protocol, the communication protocol of the WWW [23]. The default TCP/IP port of HTTPS is 443. However, HTTPS protocol produces more overheads than HTTP protocol because of the computational costs associated with using SSL. A much more time consuming activity is the public key encryption and decryption needed for protecting the exchange of the private key. The private key encryption/decryption and secure hashing functions are also quite expensive operations, and they all tend to make secure transactions far slower than the unsecured ones. However, instead of using plain text socket communication, HTTPS encrypts the session data using either a version of the SSL/TLS protocol, thus ensuring reasonable protection from eavesdroppers, and man in the middle attacks [24][25]. The level of protection depends on the correctness of the implementation by the web browser and the server software and the actual cryptographic algorithms supported.

In the area of Web security, despite strong encryption on the browser-server channel, Web users still have no assurance about what happens at the other end. New vulnerabilities and exploits are discovered continuously. Therefore, it is impossible to be 100% certain that all weaknesses in a system had been addressed. As is often pointed out within the security discipline, what we need is defense in depth and each layer added must mean a stronger defense. According to a report published by Infonetics Research [26], SSL traffic as a percentage of total network traffic will grow from 41 percent in 2003 to 49 percent in 2004 in large corporate and government organizations where SSL is used. Yet intrusion detection systems only examine clear-text HTTP traffic and ignore the encrypted traffic. These threats will be invisible by application firewalls as shown in Figure 6. The transaction and business operation

were unable to be analyzed by detection engine and they look safety without impacting network security. This leaves the most important and risk level traffic on the network invisible to the primary security tools designed to identify and prevent attacks.

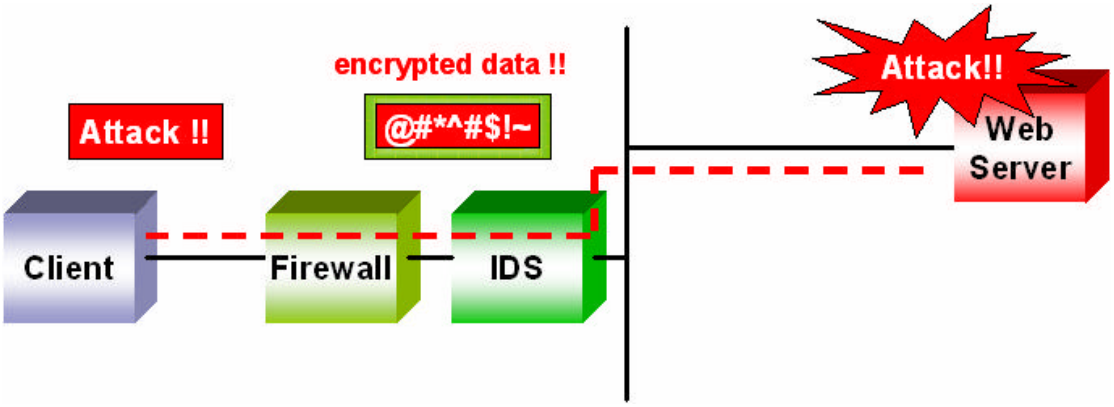

Figure 6. Attack is invisible under encryption.