http://www.jsebs.org
ISSN: 2288-3908

The Journal of Society for e-Business Studies
Vol.22, No.2, May 2017, pp.169-185
https://doi.org/10.7838/jsebs.2017.22.2.169

SSL/TLS 공격에 대한 신규 대응 방안

New Security Approaches for SSL/TLS Attacks Resistance in Practice

짠송닷푹(Tran Song Dat Phuc)*, 이창훈(Changhoon Lee)**

초 록

SSL의 취약점을 이용한 공격 기법인 BEAST를 발표했던 Juliano Rizzo와 Thai Duong이 새로운 공격 기법인 CRIME(Compression Ration Info-leak Made Easy)을 발표하였다. CRIME 공격은 암호화된 데이터에 대한 비밀 정보를 찾아내기 위해 데이터가 압축 및 암호화되는 방법의 취약점을 이용한 공격이다. 공격자는 이 공격법을 반복하여 데이터를 복호화할 수 있고, HTTP 세션의 쿠기 데이터를 복원할 수 있다. 공격자는 SPDY 및 SSL/TLS의 압축 함수를 대상으로 하는 이 보안 취약점을 이용하여 다양한 길이의 입력 데이터를 선택함으로써 암호화된 데이터의 길이를 확인할 수 있다. TLS 프로토콜은 두 통신자(서버 및 클라이언트) 사이에서 발생하는 데이터 통신의 무결성을 보장하고 두 대상에 대한 인증 수단을 제공하고 있으며, 최근 몇 년 동안 이들을 대상으로 TLS 메커니즘의 몇몇 특성들을 이용한 다양한 공격들이 수행되고 연구되었다. 본 논문에서는 CRIME 및 SSL/TLS에 대한 다양한 공격 기법들과 이들에 대한 대응 및 구현 방안에 대하여 논의하며, 실용적인 관점에서 SSL/TLS 공격 대응 방안의 방향을 제시한다.

ABSTRACT

Juliano Rizzo and Thai Duong, the authors of the BEAST attack [11, 12] on SSL, have proposed a new attack named CRIME [13] which is Compression Ratio Info-leak Made Easy. The CRIME exploits how data compression and encryption interact to discover secret information about the underlying encrypted data. Repeating this method allows an attacker to eventually decrypt the data and recover HTTP session cookies. This security weakness targets in SPDY and SSL/TLS compression. The attack becomes effective because the attacker is enable to choose different input data and observe the length of the encrypted data that comes out. Since Transport Layer Security (TLS) ensures integrity of data transmitted between two parties (server and client) and provides strong authentication for both parties, in the last few years, it has a wide range of attacks on SSL/TLS which have exploited various features in the TLS mechanism. In this paper, we will discuss about the CRIME

This study was supported by the Research Program funded by the SeoulTech (Seoul National University of Science and Technology).
The researcher claims no conflicts of interest.

* First Author, Department of Computer Science and Engineering, Seoul National University of Science and Technology(datphuc_89@yahoo.com)
** Corresponding Author, Department of Computer Science and Engineering, Seoul National University of Science and Technology(chlee@seoultech.ac.kr)

Received: 2017-05-08, Review completed: 2017-05-15, Accepted: 2017-05-19

and other versions of SSL/TLS attacks along with countermeasures, implementations. We also present direction for SSL/TLS attacks resistance in practice.

키워드 : SSL/TLS, 핸드쉐이크 프로토콜, CRIME, BEAST, BREACH, SPDY, 압축 함수, DEFLATE, LZ77
SSL/TLS, Handshake Protocol, Record Layer, CRIME, BEAST, BREACH, SPDY, Compression, DEFLATE, LZ77

1. Introduction

Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols which developed by Netscape to secure transaction over network communications, have become the secure standard factor for ensuring authentication, integrity, and data privacy of messages exchanged between parties (server-client) in the Internet. The SSL/TLS protocol can be easily operated with any other services, such as FTP, VPN and VoIP, or any other protocols, such as HTTPS, SIPS and SMTPS. It ensures the authentication for parties in a communication session, the encryption for data when transmission, and the data integrity in transit as well. SSL/TLS is used not only with web browser to browse the Internet more secure but also whenever data confidentiality and protection are necessary, such as SQL access, remote access, email, and transactions with an e-commerce site. The popular protocol is operated with SSL/TLS is HTTP which executes on the application layer and the top layer to provide a trust and secure data communication over Internet. Compression is applied within HTTP to reduce used bandwidth and transmission time before sending out data from web server. The two most common methods are gzip and deflate. In fact, HTTP supports various compression methods within a compression algorithms list so any methods in list can be used to perform compression.

The popularity of SSL/TLS leads to the fact that its security issues has been intensive studied. Some types of attacks were designed to exploit the weakness properties within the SSL/TLS architecture, such as BEAST [11, 12], CRIME [13], BREACH [9], TIME [7] and LUCKY13 [1]. The cipher suites used in SSL/TLS for encryption and key establishment were also proposed along with mitigations and countermeasures to prevent those attacks.

This paper describes numerous SSL/TLS attacks including CRIME and other versions: BEAST, BREACH. We give a general overview on basic concepts of SSL/TLS protocol and SSL/TLS attacks, like countermeasures and implementations. The direction of SSL/TLS attacks resistance is also discussed. Finally, the briefly conclusion is given to summary and complete our paper.

2. SSL/TLS Protocol and its Impact

TLS and SSL are protocols which not only help to authenticate servers and clients but also encrypt data between these parties across an untrusted network. The terms SSL and TLS are often used interchangeably or in conjunction with each other (SSL/TLS), where SSL 3.0 served as the basis for TLS 1.0 which is sometimes referred to as SSL 3.1.

2.1 SSL/TLS Protocol

SSL/TLS is the most widely protocol which provides secure HTTP (HTTPS) transactions between clients (browser) and servers. TLS/SSL can be operated with other protocols, such as Simple Mail Transfer Protocol (SMTP), File Transfer Protocol (FTP), and Lightweight Directory Access Protocol (LDAP). It ensures server-client authentication, data encryption and data integrity over network communications.

A TLS link exists between two peers and is identified by a session that contains information regarding the security mechanisms used. A single session can have multiple connections. TLS provides authentication integrity and confidentiality by supporting several cipher suites but can also support minimal or no security. A cipher suite defines what combination of authentication, encryption and integrity algorithms to be used. Some of the algorithms that provide encryption are AES, 3DES and RC4, and some of the algorithms that provide integrity are MD5, SHA1 and SHA-256. TLS provide authentication by using certificates signed by a trusted Certificate Authority (CA).

The SSL protocol consists of two protocols: the Record Protocol and the Handshake Protocol. The purpose of these protocols is to help client to authenticate a server and establish a secure encrypted SSL connection. The SSL Record Protocol which applies cryptographic cipher and mode of operations for ensuring confidentiality and MAC algorithm for ensuring integrity is used to exchange records between the client and server. The SSL Handshake Protocol uses the SSL Record Protocol to exchange a series of messages between a server and a client when they initially establish an SSL connection. This exchange of messages is enable for authenticating the server to the client, negotiating the used cryptographic algorithms or ciphers, optionally authenticating the client to the server by exchanging digital certificates, using public key encryption to generate and share secret keys, and establish sessions and encrypted SSL connections.

In the authentication process, a client sends a message to a server, and then receives responds from the server with information that the server uses to authenticate itself. The client and server perform an additional exchange of

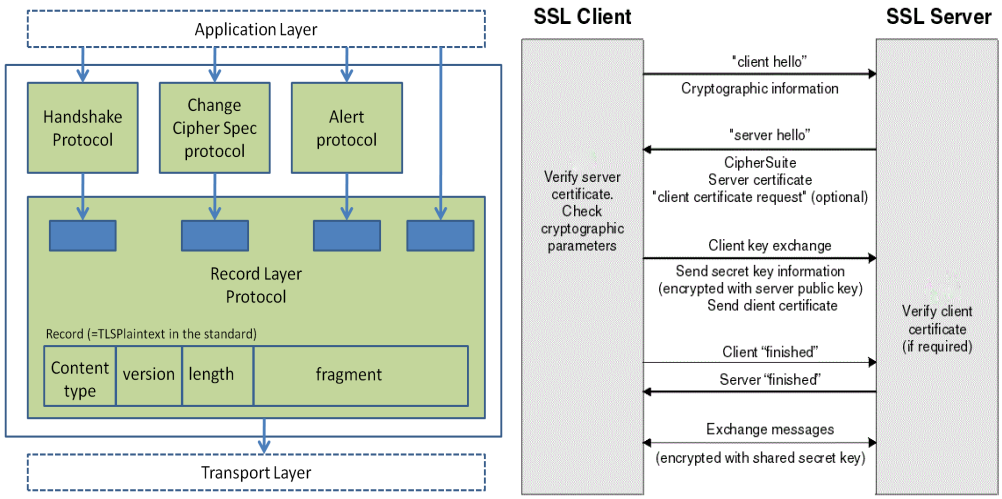
session keys, and finish the authentication. Once authentication step is completed, SSL-secured communication can be established between the server and the client using the symmetric encryption keys generated during the authentication process. This type of authentication called unilateral authentication where only one of the parties is authenticated to the other. In this case, the server authenticates itself to the client, but the client does not.

With mutual authentication, the client also authenticate itself to the server by presenting certificate to the server (signing all messages with the encrypted premaster secret). SSL also provides an authentication process named anonymous key exchange based on the Diffie-Hellman protocol. However, this process is unsecure while it is vulnerable to man-in-the-middle attacks because neither party is authenticated to the other.

2.2 SSL/TLS Benefits in Network Communications

SSL/TLS protocol provides benefits to both two parties (clients and servers) through its support over numerous methods of authentication, integrity, privacy, and interoperability since they are easy to deploy and use.

SSL/TLS helps to authenticate the identities of those parties in a secure communication through encryption algorithms. This protocol also provides integrity todata transmitted in network using check value for preventing data disclosure and resisting common types of attacks, such as man-in-the-middle attack, replay attack and rollback attack. Furthermore, SSL/TLS offers relating addition choice of options support for building mechanisms of authentication, integrity, and algorithms of encryption and hashing that being used during



〈Figure 1〉 SSL Record Protocol (left) and SSL Handshake (right)

the secure session.

SSL/TLS works with most Web browsers, operating systems and Web servers. It is also integrated in a variety of other applications. TLS/SSL is implemented at application layer, that most of its operations are completely invisible to the client. This means the client still is protected from malicious attackers even if they do not have or limit knowledge of network communications security.

2.3 SSL/TLS Drawbacks

The most limitation when implementing SSL/TLS is the increase of processor load. SSL/TLS requires with its cryptography, public key operations and other attributes that are CPU-intensive. It leads to the high cost in performance varies, resources needed. The more and the longer connections are established, the more performance varies and the more resources are lost. The issue of optimization and integration between SSL/TLS and various computing systems should be considered.

Other problems relate to maintenance and management. A TLS/SSL environment is complex and requires maintenance that requires the system administrator must configure the system and manage certificates much.

Besides, SSL/TLS vulnerabilities exist due to the ways to configure it and its multiple implementations. If a TLS connection uses no authentication of either parties, the channel is not protected against man-in-the-middle attacks. Anonymous Diffie-Hellman is therefore discouraged by the IETF because it cannot prevent man-in-the-middle attacks. If TLS is used without Diffie-Hellman key exchange it is often only necessary to acquire a server's private key to decrypt any messages sent to that server. Stolen keys have been used in combination with large-scale monitoring of servers. This can be mitigated by improved protection of the private key or by using forward secrecy. A survey in 2012 showed that many implementations of SSL/TLS often performed certificate validation either incorrectly or not at all. This problem is likely still prevalent today. The most popular attack on this protocol is man-in-the-middle attacks that acquire to improve protection of the private key from server where stolen keys can be used in combination with large-scale servers' monitoring. Many implementations of SSL/TLS often perform certificate validation either incorrect or not at all.

3. Attacks against SSL/TLS Protocol-CRIME and Others

The major problem in SSL/TLS protocol is information leakage that occurs when data is compressed prior to encryption. Attacker firstly inject a javascript or arbitrary content that generates predictable data (known plaintext

packets) into a website. Then he will be able to get the unknown data by sniffing the network of a client that opened this website (or the network of the server) and using the encrypted packets of script with the known plaintext packets to obtain encryption keys. By this way, he can steal cookies and hijack the session of the website. In this section, we present some SSL/TLS attacks with the viewpoint of countermeasures and implementations.

3.1 CRIME Attack

CRIME (Compression Ratio Info-leak Made Easy) is a practical side-channel attack that builds on weakness of information leakage. It works by exploiting compression functions of HTTP requests, and then observing the change in compressed data length. CRIME exploits TLS encryption to discover secret information such as session tokens. The attacker enables to intercept the victims network traffic and to make the victims browser send requests. A request contains a guess of the secret information that the attacker wishes to reveal. Due to the properties of compression, the requests that contain the best guess will have a smaller size. By performing repeatedly improved guesses the attacker can uncover the secret information. The technique uses two data-compression schemes (DEFLATE and gzip) to hijack web sessions protected by SSL/TLS or reduces network congestion and the webpages' loading time. This attack is capable of being effected on most servers and client browsers that support TLS compression.

Compression is a method to help reduce the size of data that necessary to quickly transmit between parties over a network or to save storage space in a database. In TLS, the technique used to compress data is DEFLATE. DEFLATE consists of two sub algorithms: Lempel-Ziv coding (LZ77) and Huffman coding. LZ77 is used to eliminate the redundancy of repeating sequences, while Huff man coding is used to eliminate the redundancy of repeating symbols.

During a TLS handshake, the client states in the ClientHello message the list of compression algorithms that it supports. The server responds with the compression algorithm in the Server Hello message. Compression algorithms are specified by one-byte identifiers. When TLS compression is used, it is applied as a long stream on all the transferred data. The purpose is that when exchanging big data or large amount of information, it helps reduce usage of bandwidth while preserving integrity and privacy, even security.

We imagine the POST of browser is as following:

POST/target HTTP/1.1
Host: travel.com
User-Agent: Mozilla/5.0 (Windows 10 WOW64; rv: 14.0) Gecko/20100101 Firefox/14.0.1
Cookie: sessionid=c73e89dff4119ab2e527cdd648aefb59

sessionid=a

The attacker want to get a client's cookie. He knows the session token '*Cookie: sessionid=?*' generated by the target website '*travel.com*', then he will inject a javascript to initiate requests to the server '*sessionid=a*', and observe the size of the compressed SSL packets responsed by server. If the injected character is match with the cookie value, the size of the response will be smaller and otherwise, the character is not in cookie value if the size of the response is bigger. In this case, the response packet has larger size than initial.

The same process is repeated since the attacker makes a brute force attack on the cookie value through the server responses until the entire cookie value is retrieved.

3.1.1 Countermeasures

CRIME can be mitigated by disabling the SSL compression, either at the client by patching or upgrading browser to prevent the use of the SSL compression (Internet Explorer, Google Chrome v.21 or later, Firefox v.15 or later, Opera v.12 or later, Safari v.5 or later), or at the server by using SSL Labs service or SSL scanning tools to look for the compression in the miscellaneous section and manage the protocol features of the TLS protocol, then disable it.

Version: TLS 1.2 (0×0303)
Random
Session ID Length: 0
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0×c02f)
Compression Method: null (0)

The server response to a ClientHello message by a Server Hello message which based on the compressions offered by the client. The client selects the '*Compression Method*' is '*null*' means there is no compression, and the data will not be compressed. By this way, a server can control and refuse the use of compression.

3.1.2 Implementations

The CRIME attack takes advantage on SSL/TLS compression features when exchanging data between authenticated parties. In addition, SPDY which is a networking protocol that uses a similar compression scheme is also vulnerable. The SPDY protocol was developed by Google and uses techniques like compression, multiplexing and prioritization to reduce the latency of web pages. SPDY has been implemented in popular browsers like Google Chrome or Firefox, and be supported by several websites such as Google search, Gmail and Twitter.

TLS compression supported among websites is widespread. SSL Pulse is a project which monitors SSL/TLS implementations on the world's top of 180,000 HTTPS-enabled websites. Due to SSL Pulse, it has more than

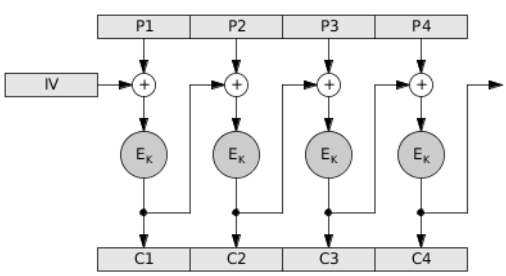
forty-percent of servers supports compression. However, there is a low level since just about ten-percent of client-side support TLS or SPDY compression. The acquirement for CRIME attack to apply is that both the server and the client need to support the compression feature.

In practice, Internet Explorer never supported TLS or SPDY compression with all its versions. Mozilla Firefox v.15 or latest versions removes the compression, so these browsers are not vulnerable to CRIME attack.

3.2 BEAST Attack

Thai Duong and Juliano Rizzo proposed the BEAST (browser exploit against SSL/TLS) which carries out on TLS v1.0. However, TLS v1.2 is not susceptible to thistype of attack.

Whenever we log into a websitepage, after authentication we will see the authenticated page with a "*session id*". All the "*session id*" is encrypted to prevent session hijacking. A "*session id*" is assigned by a server to a client to maintain the current state of the web page



〈Figure 2〉 CBC Mode of Operation

which is a random number or string. It is placed in the site cookie or in the browser's URL.

The BEAST attack is a chosen-plaintext attack targets on TLS 1.0. Since the cookie place is predictable, the attacker can monitor the encrypted traffic and use the session cookie value as an initialization vector (IV) to guess plain text message. He can gather the IVs for each record just by sniffing the network. He then can make a guess at the session cookie and see if the cipher text matches to reveal entire correct cookie. It is noting again that in TLS 1.0 protocol, when exchanging large data with multiple packets, it uses the last cipher text block of the previous packet as an IV for the next packet (like CBC mode).

In CBC mode, an IV is used in the first block aiming to make each message unique. In fact, the IV only adds randomness to the output, but it is not a secret information.

Having the message:

sessionid=Gxs36NepewqeMI763Hej31pkl

We assume that an attacker initially knows the IV value. In this case, the string '*sessionid=Gxs36NepewqeMI763Hej31pkl*' is a plaintext and will be XORed with the IV which is ciphertext of the previous block in CBC mode. The attacker can predict the plaintext value by guessing single character at a time, then XOR with the IV to check whether it corresponds to the ciphertext value. He executes

the method by injecting a random string "*sessionid=a*", watching the results, and repeats the process until he can successfully obtain a cookie character. Once the attack gets the first correct character, he can apply the attack to the next character to exploit the entire cookie value. Although showing the effectiveness in execution and performance, the attack still exists limitations, such as the requirement in the same network of attacker to apply a MITM attack, the attacker just can make a guess on only one block at a time, and attacker should does monitor a modified traffic to see the matching results with multiple requests.

3.2.1 Countermeasures

It exists a vulnerability when using CBC mode in block cipher. It was identified in TLS 1.0. However, it was addressed in TLS 1.1 and TLS 1.2 with the use of "explicit IVs" for each block. In addition, TLS 1.1 and TLS 1.2 choose using GCM (authentication-encryption) mode to operate, so they are not susceptible to this attack. Some of the browsers have attempted to implement a solution to address the vulnerability while remaining compatible with the SSL 3.0/TLS 1.0 protocol. If we use a lower version of TLS or if the server is still using SSL, we can use a stream cipher such as RC4 instead of block cipher with modes of operation.

3.2.2 Implementations

The software support against the vulnerability which recommends client to notice and consider. With browsers, if we are using Google, it should be upgraded to Chrome 16 or later. Similarly, with Microsoft, we need to ensure that MS12-006 has been applied. For Mozilla, it should be Firefox 10 or later. And with libraries, they must support TLS 1.1 or higher.

3.3 BREACH Attack

BREACH (Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext) also targets HTTP compression and uses the same approach as CRIME. This attack also uses the victims' browser to send requests containing a guess of the secret information but relies on that the responses reflect the users input as well as containing the secret information in the body. Obviously, the response with smallest size contains the best guess. Since BREACH targets HTTP compression, all versions of TLS protocol are vulnerable and they do not matter which cipher suite is used. The BREACH attack can even be operated with only a few thousand request in under a minute. There are currently no known TLS mitigations towards this attack.

BREACH exploits the compression and encryption combination between clients and servers. While the CRIME attack targets the TLS compression, the BREACH attack targets HTTP compression. HTTP compressed response generated by server compresses whole parts of the response, except header in-

formation. The DEFLATE algorithm consists of two components. It uses LZ77 instead of occurrences of characters with pointer values to reduce space. And, Huffman coding replaces characters with symbols to optimize the description of the data to the smallest size possible. The BREACH attacks the LZ77 compression while minimize the effects of Huffman coding.

We consider a GET request to an SSL enabled web server and observe the response:

GET/stuff/page1.php?id=786345
Searching
<form target="https://www.travel.com:443/stuff/search.php?id=786345">…

The attacker requests with string '*id=786345*' to guess the '*token=…*' value character by character. Similar the method in CRIME, he injects token values in the '*id*' and monitor how the length size has changed in the compressed response. If the injected token matches the actual token, the length of the response will be smaller than initial duplicate due to HTTP compression. In this case, the actual token value is the length of the string '*token=csvfdfcrvet343v.*'

The attacker only knows that the '*token=…*' is a part of the string. He logs into the application and initiates the attack by sending a token '*token=a*'.

The request is:

'GET/stuff/page1.php?id=token=a'

and the response is:

Searching
<form target="https://www.travel.com:443/stuff/search.php?id=token=a">...

The length of the response will decrease by 6 via the length value of the string '*token=…*'. The attacker observes the response length and knows that the injected token is incorrect. He will repeat the process with different values of the '*token*'. When the attacker tries '*token=c*', the request is:

'GET/stuff/page1.php?id=token=c'

The response is:

Searching
<form target="https://www.travel.com:443/stuff/search.php?id=token=c">...

The token response length will decrease by 7 since the first character of token guess matches the actual token character. The attacker determines that the injected token is correct. He then varies the next character and repeats the entire process again until he has successfully guessed the entire '*token*' value. In practice, the BREACH attack can be

operated under a minute based on number of thousand requests and secret size. The method is effective because the fact that it allows guessing a secret one character at a time.

3.3.1 Countermeasures

Some methods can be used to mitigate the BREACH attack, include disabling HTTP compression, hiding the response length (by adding garbage data of random length to the response), separating secrets from user input, using same-site cookies flag, masking secrets (by XORing with a random secret per request), and rate-limiting the requests.

3.3.2 Implementations

One of a practical implementation of the BREACH attack, Rupture, is using HTTP injection to execute a meet-in-the-middle attack on client's browser to analyze the HTTP traffic which communicates to a server. However, this method is still at low risk for organizations using mailbox like Gmail, Yahoo mail or social network like Facebook. Compared to it, there is various faster methods attacker can use to break or steal secret information. The rules could be set up in the intrusion-detection system or a host-based detection system based on number of connections needed for a BREACH attack that is alerted when a number of connections occursfrom an individual system.

4. CRIME, BEAST, BREACH and Other Versions—The Perfect SSL/TLS Attacks

We can consider obviously that major problems addressed in SSL/TLS protocol come from the client (browser) level. A MITM attacker can intercept non-encrypted requests, mess with them, and trick browser into sending requests with arbitrary content to the sites that we care about. It is this interaction that makes several attacks possible: BEAST, CRIME, BREACH, RC4, TIME, Lucky13.

In some cases, the mitigation methods or security countermeasures can be applied help to protect the transmission against known or unknown attacks. However, it would take a lot of discussions, considerations and politics to get it effective implemented. The solution is that a server (web site) can control and manage which other web sites can initiate requests to access into it.

4.1 Fixing on DEFLATE against CRIME/BREACH Attacks

HTTP compression uses DEFLATE which is an implementation of LZ77 matching and Huffman coding. The BREACH attack targets HTTP compression and observes how the server leaks information in compressed encrypted response. The attacker issues requests, varying the input until the input

matches some secret elsewhere on the page. Then, he will make a correct guess of secret by observing the page size decrease based through the DEFLATE algorithm. Unfortunately, even if the compressor did no LZ77 matching, the attacker might be still able to vary his chosen plaintext to gain information about the character frequencies, as the attacker's injection of characters can change the Huffman tree used to encode the document.

Since the major problem is compression, the most efficient way to mitigate the attacks is to disable compression. However, HTTP compression typically offers space savings (around 60%) which translates into faster response times for users. The solution for all those issues is that DEFLATE compressor fixing to ignore the unique, human-oriented information in individual files and take advantage of the large redundancy in HTML, CSS, and Javascript due to the common strings inherent to each language. The method not only produces compatible compressed data, but also avoids leaking secret information by using some extra properties.

4.1.1 LZ77 Matching

Secrets already encoded in the data-stream should never be part of a match candidate. The matchings are only produced from unlikely byte numbers to compose majority of a secret (ASCII symbols), or a string white list common

〈Table 1〉 SSL/TLS Attacks

	Target	Example	Based on	Timeline
Padding Oracle	Steal request payload	Session cookie	Padding Oracle model	Somewhere in the 1990's
Browser Exploit Against SSL/TLS (BEAST)	Steel request payload	Session cookie	HSR model	Described in 2002 (led to TLS1.1), demonstrated in 2011
Compression Ratio Info-leak Made Easy (CRIME)	Steel request payload	Session cookie	HSR model	Described in 2002, demonstrated in 2011
Time info-leak Made Easy (TIME)	Steel response payload	CSRF token Session Cookie	HSR model	Demonstrated in 2012
LUCKY13	Steel request payload	Session cookie	Padding Oracle model	Demonstrated in 2012
RC4	Steel request payload	Session cookie	Cryptographic weakness	Demonstrated in 2013
Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext (BREACH)	Steel response payload	CSRF token		Demonstrated in 2013

to the data format. The supported formats include HTML, CSS, Javascript, and undetected (auto-detected based on the first 100 bytes of uncompressed input).

In HTML, CSS and Javascript format, the matchings are only produced from bytes unlikely to compose majority of a secret (whitespace, special characters: < >, /, =, “, #, ( ), { }), format tag and attribute names longer than one character, or any global, user-provided safe strings. With undetected mode, the matchings do not occur if the user does not provide a list of safe strings. Furthermore, with whitelisted strings, for all formats, they begin with an alphanumeric character in a match if the character immediately previous to the string is non-alphanumeric, and end in an alphanumeric character in a match if the character immediately following the string is non-alphanumeric. Consequently, we must ensure that secrets are never a match source. It does not matter if a valid match source is used to create output that is not a valid match source. When this happens, the pool of match sources that the attacker may query does not increase.

4.1.2 Huffman Coding

The purpose to modify the Huffman coding is to leak as less information as possible about the frequencies of character in any secrets. The Huffman code is using only probability models determined by the user, consisting of a non-dynamic, default frequency table, as determined by the main file type (HTML, JS and CSS), fixed length Huffman codes, and a user-provided frequency table. The Huffman coding strategy must not change on every request. Otherwise, an attacker might be able to determine a secret's character frequencies by figuring out which injected characters tip the compressor to use a different strategy.

The user might calculate the frequency table based only on the first n requests for a resource after server startup, and use that table only after all n requests (n = 8), or choose to update the frequency table with information from a request only when a random number is less than 1/4^n, where n is the number of previous times the table has been updated since server startup.

The method has a greater size expansion in the worst case which depends on the longest code in the Huffman tree. It helps in the case that an attacker cannot effectively control the decision to gain information about the page content.

4.2 LZ77 Replacement in DEFLATE Compression

In the CRIME or BREACH attack, the target are secrets exchanged through HTTPs, SPDY, TLS or any protocol offering secure communication between a server and a client, usually at the application layer. The protocol also needs to compress the requests using a

compression algorithm based on DEFLATE. Common such algorithms are gzip and zlib. In the challenge, these requirements are implemented by giving the attacker access to a compression oracle.

The attacker exploit characteristic of LZ77, one of the two algorithms that forms DEFLATE. LZ77 scans the input, and when it encounters the same substring for the second time, it replaces the second occurrence with a reference to the first one, in terms of distance and length. The attacker guess the first byte of the secret session ID by producing requests whose payload is his guess of the byte. If he got it right, he can observe that DEFLATE detects two identical bytes, and produces a shorter compressed than usual byte. To proceed, he repeats the same for the next byte, appending it after the bytes he have already discovered. Obviously, he knows the length of the secrets before stopping guessing.

The Burrows-Wheeler Transform (BWT) [10] is a reversible algorithm that is used in the bzip2 compression algorithm. In fact, this method only does formats the data but not reduce the size of them, so it will be more effective when compressed by other algorithms. When a string of characters is transformed using BWT, the size of the characters remains the same, and the algorithm just calculates the order that the characters appear in. BWT helps compression algorithms become easier to procedure due to the input format which contains sets of repeated characters. Because of characteristics of move-to-front coding, the data will be more compressible if they are in a specific format which consists of statistical encoders such as Huffman and other arithmetic coding.

Although simple Huffman and arithmetic coders combine well with the BWT in algorithm process, there is still need a more complex and better coding scheme. This is because the probability of a certain value at a given point in vector depends to a certain extent on the immediately preceding value. The most important effect to procedure in practice is that zeroes exist in the vector. By using Huffman codes following a run of zeroes, it can compress the output of the move-to-front coder with a modified Huffman coder. In addition, we do a replacement of the Huffman trees for whole input by a new one for just each 16 kbyte input block.

5. Conclusion

In this paper, we have presented SSL/TLS vulnerabilities with specific SSL/TLS attacks, including CRIME and other versions: BEAST, BREACH. For each attack, we also gave implementation in practice and countermeasures for mitigating the attack effect. Since the SSL/TLS attacks have increased rapidly because of its weaknesses, we must protect our transmission carefully as much as possible. User should always be

concerned and aware of security as the landscape changes constantly with the need to upgrade and to implement software fixes. If we run a server, it is better to immediately disable support for TLS export cipher suites. Weshould also disable other insecure suites and enable forward secrecy as well. If we use a browser, we must ensure that the latest version of browser installed, and check for upgrades frequently. And if we are a system admin or developer, we should ensure that any TLS libraries we use are up to date.

References

[1] AlFardan, N. and Paterson, K., "Lucky Thirteen: Breaking the TLS and DTLS Record Protocols," IEEE Symposium on Security and Privacy, http://www.ieee-security.org/TC/SP2013/papers/4977a526.pdf, 2013.

[2] AlFardan, N., Bernstein, D., Paterson, K., Poettering, B., and Schuldt, J., "On the Security of RC4 in TLS and WPA," http://www.isg.rhul.ac.uk/tls/RC4biases.pdf, 2013.

[3] Bellare, M. and Rogaway, P., "Entity authentication and key distribution," pp. 232-249, 1994.

[4] Dierks, T. and Allen, C., "The TLS Protocol Version 1.0," RFC 2246, Internet Engineering Task Force, 1999. Available at: http://www.ietf.org/rfc/rfc2246.txt.

[5] Hwang, S. J. and Lee, C. H., "Padding Oracle Attack on Block Cipher with CBC|CBC-Double Mode of Operation using the BOZ-PAD," The Journal of Society for e-Business Studies, Vol. 20, No. 1, pp. 89-97, 2015.

[6] Jin, C. Y., Kim, A. C., and Lim, J. I., "Correlation Analysis in Information Security Checklist Based on Knowledge Network," The Journal of Society for e-Business Studies, Vol. 19, No. 2, pp. 89-97, 2014.

[7] Mavrogiannopoulos, N., Vercauteren, F., Velichkov, V., and Preneel, B., "A cross-protocol attack on the TLS protocol," Proceedings of the 2012 ACM Conference in Computer and Communications Security, pp. 62-72, http://doi.acm.org/10.1145/2382196.23 82206, 2012.

[8] Popov, A., "Prohibiting RC4 Cipher Suites," Work in Progress, draft-ietf-tls-prohibiting-rc4-01, 2014.

[9] Prado, A., Harris, N., and Gluck, Y., "The BREACH Attack," http://breachattack.com, 2013.

[10] Rescorla, E., "SSL and TLS: Designing and Building Secure Systems," Addison-Wesley, 2001.

[11] Rizzo, J. and Duong, T., "Browser Exploit Against SSL/TLS," http://packetstormsecurity.com/files/105499/Browser-Exploit-Against-SSL-TLS.html, 2011.

[12] Rizzo, J. and Duong, T., "Here Come The

Ninjas," Ekoparty Security Conference, 2012.

[13] Rizzo, J. and Duong, T., "The CRIME Attack," EKOparty Security Conference, 2012.

[14] Rosenfeld, M., "Internet Explorer SSL Vulnerability," 2008. Available at: http://www.thoughtcrime.org/ie-ssl-chain.txt.

[15] Seok, O. N., Han, Y. S., Eom, C. W., Oh, K. S., and Lee, B. K., "Developing the Assessment Method for Information Security Levels," The Journal of Society for e-Business Studies, Vol. 16, No. 2, pp. 159-169, 2011.

저 자 소 개

Tran Song Dat Phuc (E-mail: datphuc_89@yahoo.com)

2011	HCMC University of Technology, Vietnam (Bachelor Degree)
2015	Seoul National University of Science and Technology, Department of Computer Science and Engineering (Master Degree)
2015~current	Seoul National University of Science and Technology, Department of Computer Science and Engineering (Doctoral Candidate)
Research Interests	Information Security, Cryptography, Network Security

Changhoon Lee (E-mail: chlee@seoultech.ac.kr)

2001	Hanyang University, Department of Mathematics (Bachelor Degree)
2003	Korea University, Department of Information Management and Security (Master Degree)
2008	Korea University, Department of Information Management and Security (Doctoral Degree)
2009~2012	Hanshin University, Department of Computer Engineering Assistant Professor
2012~2015	Seoul National University of Science and Technology, Department of Computer Science and Engineering Assistant Professor
2015~current	Seoul National University of Science and Technology, Department of Computer Science and Engineering Associated Professor
Research Interests	Information Security, Cryptography, Digital Forensics, Computer Theory