Systematic Study Of Various Possible Attacks Against SSL/TLS Protocol: Counter Measures And Their Implementation

[1]Suresh Prasad Kannojia, [2]Jitendra Kurmi

[1]Assistant Professor, [2]Rsearch Scholar
[1]Department of Computer Science,
[1]University of Lucknow, Lucknow, India

Abstract: In digital world security of information is the major concern though they are secure with various types of security protocols. This paper provides the systematic study of various possible attacks against SSL/TLS protocol, counter measures and their implementation.

Index Terms – SSL, TLS, Crime Attack, BEAST Attack, BREACH Attack

I. INTRODUCTION

In recent years SSL/TLS become an essential technical component to provide privacy and security as it enables individuals to encrypt data for secure communication. Therefore, Transport layer security is subject of interest for the researchers, many issues such as performance of TLS libraries, defending and preventing from various vulnerabilities, attacks, integrity of AEAD ciphers, and impact of TLS overhead in segmented networks. The major problem in the SSL/TLS protocol is information leakage when data is compressed before encryption. First, the attacker injects 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 and using the encrypted packets of the script with the known plaintext packets to obtain encryption keys. In this way, he can steal cookies and hijack the website's session.

This paper consists of three sections: Various types of possible attacks, counter measures and implementation has discussed in Section II. Finally, Section III concludes the paper.
In next section, we discuss some SSL/TLS attacks from the viewpoint of countermeasures and implementations.

II. TYPES OF POSSIBLE ATTACKS, COUNTER MEASURES AND IMPLEMENTATION

1. CRIME ATTACK

The Compression Ratio Info-leak Made Easy is a viable side-channel attack exploiting information leakage vulnerabilities. It works by utilizing HTTP request compression functions and observing the change in the compressed data length. The purpose is that exchanging big data or large amounts of information helps reduce bandwidth usage while preserving integrity, privacy, and even security. We imagine the POST of the browser is as follows:
POST/target HTTP / 1.1
Host: travel.com
Mozilla/5.0 is the user agent (Windows 10 WO W64; rv: 14.0) Firefox/14.0.1 Gecko/20100101Cookie: session id=c73e89dff4119ab2e527cdd648aefb59 session id=a
The attacker wants to get a client's cookie. He knows the session token 'Cookie: session id=?' generated by the target website 'travel.com,' then inject a javascript to initiate requests to the server 'session id=a' and observes the size of the compressed SSL packets responded by the server. If the injected character matches

the cookie value, the size of the response will be smaller. Otherwise, the character is not in cookie value if the size of the reaction is bigger. In this case, the response packet is larger than the initial.
The exact 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.
Countermeasures: The TLS can mitigate CRIME by disabling the SSL compression, either at the client by patching or upgrading the 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)

A ServerHello message is sent in response to a ClientHello message by the server, taking into account the client's compressions. The client selects the 'Compression Method' is 'null' means there is no compression, so the data is not compressed. In this way, a server can control and refuse use of compression.
Implementations: The SSL/TLS compression features were used for CRIME attacks when exchanging data between authenticated parties. The majority of websites support TLS compression. The SSL Pulse project keeps track of the 180,000 most popular HTTPS-enabled websites world wide's SSL/TLS implementations. More than 40% of servers now enable compression thanks to SSL Pulse. There is a low level, nevertheless, as only 10% of client-side applications implement SPDY compression or TLS. In case of CRIME attack to function, both the server and the client must always support compressions. TLS and SPDY compression were never supported by any version of Internet Explorer. The recent versions of Mozilla Firefox eliminate the compression, so these browsers are not vulnerable to CRIME attacks.

2. BEAST Attack

The BEAST (Browser Exploit Against SSL/TLS) attack was proposed by Thai Duong and Juliano Rizzo, which relies on TLS v1.0. However, TLS v1.2 is not susceptible to this type of attack. Whenever we log into a website page, we will see the authenticated page with a "session id" after authentication. All the "session id" is encrypted to prevent session hijacking. A server assigns a "session id" to a client to maintain the current state of the web page, which is a random number or string.

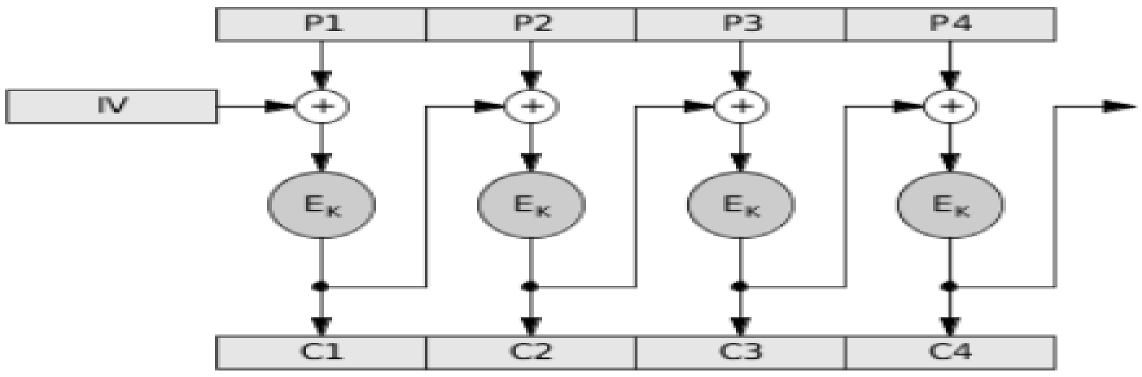

Figure 1. CBC Mode of Operation

It is placed in the site cookie or the browser's URL. The BEAST attack is a chosen-plaintext attack target 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 a plain text message. By sniffing the network, he may acquire the IVs for each record. He can then guess the session cookie and compare it to the encrypted text, revealing the whole, correct cookie. When transferring large amounts of data with many packets, the

TLS 1.0 protocol uses the last ciphertext block of the predecessor packet as an initialization vector for the subsequent. The first block employs an IV in CBC mode to ensure that each message is unique. In reality, the IV adds unpredictability to the output, but this is not classified data.
Having the message:
session id=Gxs36NepewqeMI763Hej31pkl
We assume that an attacker initially knows the IV value. In this case, the string 'session id= Gxs36NepewqeMI763Hej31pkl' in CBC mode is a plaintext that will be XORed with the IV ciphertext the preceding block. By guessing a single character at the moment, the attacker can predict the plaintext value, which can then be XORed with the IV to see if it matches the ciphertext value. He executes the method by injecting a random string "session id=a," watching the results, and repeats the process until he can successfully obtain a cookie character. Once the attacker gets the first correct character, he can apply the attack to the next character to exploit the entire cookie value.
Countermeasures: There exists a vulnerability when anyone uses CBC mode in block cipher. TLS 1.0 was the first to notice it. TLS 1.1 and 1.2, on the other hand, addressed the problem by demanding "specific IVs" for each block of data. TLS 1.1 and 1.2 also run in GCM (authentication-encryption) mode, protecting them against this attack. Some browsers have tried to work around the problem while still supporting the SSL3.0 / TLS1.0 protocol. We can use RC4 stream cipher instead of a block cipher with modes of operation if we're utilizing an earlier version of TLS or the server is still using SSL.
Implementations: The software supports the vulnerability, which recommends the client to notice and consider. If we use Google, it should be upgraded to Chrome 16 or later with browsers. Similarly, we need to ensure that MS12-006 has been applied with Microsoft. For Mozilla, it should be Firefox 10 or later. And libraries must support TLS 1.1 or higher.

3. BREACH Attack

The BREACH (Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext) attack makes intelligent guesses about the secret data before sending queries through the victims' browser. However, it depends on the responses to represent both user input and the private data in the body. The best guess is evident in the response with the most petite size. Because BREACH focuses on HTTP compression, all versions of the TLS protocol, regardless of the cipher suite, are vulnerable. TLS can even carry out the BREACH attack in under a minute with only a few thousand requests. There are no known TLS mitigations for this exploit at this time. BREACH exploits the compression and encryption combination between clients and servers. The CRIME attack focuses on TLS compression, whereas the BREACH attack focuses on HTTP compression. Except for header information, the HTTP compressed response generated by the server compresses the whole response. The DEFLATE algorithm is made up of two parts. It employs LZ77 instead of occurrences of characters with pointer values to save space. In addition, Huffman coding substitutes characters with symbols to reduce size of data description with shortest possible amount. The BREACH exploits LZ77 compression while minimizing Huffman coding impacts.
We consider a GET request to an SSL enabled web server and observe the response:
GET/stuff/page1.php?id=786345
<a href="page2.php?token=csvfdfcrvet343v">Searching</a>
<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:
<a href="page2.php?token=csvfdfcrvet343v">Searching</a>
<form target="https://www.travel.com:443/stuff/search.php?id=token=a">...

The response length 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:
<a href="page2.php?token=csvfdfcrvet343v">Searching</a>
<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 the 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 until he has successfully guessed the entire 'token' value. TLS can operate for the BREACH attack in under a minute based on many thousand requests and secret size. The method is effective because it allows guessing a secret one character at a time.
Countermeasures: Cryptographic techniques can use some methods to mitigate the BREACH attacks. It involves deactivating HTTP compression, disguising secrets (by XORing with a random secret per request), and segregating secrets from user input, using the same-site cookies flag, rate-limiting requests, and eliminating HTTP compression.
Implementations: To conduct a meet-in-the-middle assault on the client's browser and examine the HTTP traffic that talks with a server, the BREACH attack's practical implementation, Rupture, requires HTTP injection. However, this method is still at low risk for organizations using mailboxes like Gmail, Yahoo Mail, or social networks like Facebook. There are various faster methods attackers can use to break or steal secret information compared to it. The rules are setup in the intrusion detection system or a host based detection system based on the number of connections needed for a BREACH attack alerted when several connections occur from an individual system.

4. Other Versions of the Perfect SSL/TLS Attacks

We can consider that significant problems addressed in SSL/TLS protocol come from the client (browser) level. A MITM attacker can steal non-encrypted requests, tamper with them, and mislead browsers into sending arbitrary material to sites. This interaction allows for various attacks: BEAST, CRIME, BREACH, RC4, TIME, and Lucky13 are all words. In some cases, the mitigation methods or security countermeasures can help protect the transmission against known or unknown attacks. However, it would take a lot of discussions, considerations, and politics to get it effectively implemented. The solution is that a server (website) can control and manage which other websites can initiate requests to access it.

5. Fixing DEFLATE against CRIME/BREACH Attacks

HTTP compression uses DEFLATE, which implements LZ77 matching and Huffman coding. The BREACH attack targets HTTP compression and observes how the server leaks information in a 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 the secret by observing the page size decrease based on the DEFLATE algorithm. Unfortunately, even if the compressor did no LZ77 matching, the attacker might still be 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 encode the document. Since the significant 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 user response times. The solution for all those issues is that DEFLATE compressor fixing to ignore the unique, human-oriented information in individual files and takes advantage of the significant redundancy in HTML, CSS, and JavaScript due to the standard strings inherent to each language. The method produces consistent compressed data and uses different properties to avoid leaking secret information.

6. LZ77 Matching

Secrets encoded in the data stream should never be part of a match candidate. The matchings are only produced from unlikely byte numbers to compose the majority of a secret (ASCII symbols) or a white string list typical 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 the majority of a secret (whitespace, special characters: < >, /, =, ", #, ( ), { }), format tag and attribute names longer than one character, or any global, user-provided safe strings. The matchings do not occur with undetected mode if the user does not provide a list of secure lines. Furthermore, with allowed strings, for all formats, they begin with an alphanumeric character in a match if the character immediately previous to the string is non-alphanumeric. The 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 good match source. The pool of match sources that the attacker may query does not increase.

7. Huffman Coding

Modifying the Huffman coding is to leak as little information as possible about character frequencies in any secrets. The Huffman code uses only probability models determined by the user, consisting of a non-dynamic, default frequency table, as determined by the primary 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 start-up. 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 times database table has been updated since server start-up. The method has a more significant size expansion in the worst case, which depends on the most extended code in the Huffman tree. It helps when an attacker cannot effectively control the decision to gain information about the page content.

8. LZ77 Replacement in DEFLATE Compression

A CRIME or BREACH attack targets a secret transmitted over HTTP, SPDY, TLS, or any other protocol that provides secure communication, often at the application layer. The protocol must also use a DEFLATE-based compression mechanism to compress the requests. Gzip and Zlib are two examples of such algorithms. These prerequisites are met in the challenge by providing the attacker with access to a compression oracle. The attacker employs a characteristic of one of DEFLATE's two algorithms, LZ77. When LZ77 encounters the exact substring a second time, it replaces it with a distance and length reference to the first. The attacker can predict the first byte of the secret session ID by sending queries with his estimated payload. If he's right, DEFLATE will notice that it finds two identical bytes and creates a compressed byte that's shorter than usual. He then repeats the process for the next byte, attaching it after the bytes he's already identified. He must know how long the secrets are before he stops guessing.

III. CONCLUSION

In this paper we have discussed various type of possible attacks with SSL/TLS and their countermeasures and way of implementation to minimize the attacks and how to provide the secure communication.

REFERENCES

[1] Dr. Taher A. Elgamal Father of SSL https://www.sec.gov/Archives/edgar/data/855612/000119312515162501/d918 759d10ka.htm

[2] Dang, K. Encrypted Cyber Attacks: Real Data Unveils Hidden Danger within SSL, TLS Traffic [online]. 2018.

[3] Desai, D. February 2018 Zscaler SSL Threat Report [online]2018.

[4] Velan, P. et al. A Survey of Methods for Encrypted Traffic Classification and Analysis. In: International Journal of Network Management. John Wiley & Sons, Ltd., 2015, vol. 25, pp. 355–374. No. 5. ISSN 1055-7148.

[5] Salowey, J. et al. Transport Layer Security (TLS) Session Resumption without Server-Side State. Request for Comments: 5077 [online]. 2008.

[6] Turner, S. and Polk, T., 2011. Prohibiting secure sockets layer (SSL) version 2.0. Request for Comments, 6176.

[7] Barnes, R., Thomson, M., Pironti, A. and Langley, A., 2015, June. Deprecating secure sockets layer version 3.0. In IETF RFC 7568.

[8] Dierks, T. and Allen, C., 1999. Rfc2246: The TLS protocol version 1.0.

[9] Dierks, T. and Rescorla, E., 2006. IETF RFC 4346,“. The Transport Layer Security (TLS) Protocol Version, 1.1

[10] A. Freier, P. Karlton, and P. Kocher. The Secure Sockets Layer (SSL) Protocol Version 3.0. RFC 6101 (Historic), August 2011.

[11] Bodo M¨oller. Security of CBC ciphersuites in SSL/TLS: Problems and countermeasures, 2004. Unpublished manuscript. Available at https://www.openssl.org/bodo/tls-cbc.txt.

[12] Gregory V. Bard. A challenging but feasible blockwise-adaptive chosenplaintext attack on SSL. In SECRYPT, pages 99–109, 2006.

[13] Pratik Guha Sarkar and Shawn Fitzgerald. Attacks on SSL – a comprehensive study of BEAST, CRIME, TIME, BREACH, Lucky 13 and RC4 biases, August 2013. White paper. Available at https://www.isecpartners.com/media/106031/ssl_attacks_survey.pdf.

[14] John Kelsey. Compression and information leakage of plaintext. In Joan Daemen and Vincent Rijmen, editors, Fast Software Encryption – FSE 2002, References 164 volume 2365 of Lecture Notes in Computer Science, pages 263–276, Leuven, Belgium, February 4–6, 2002. Springer, Heidelberg, Germany.

[15] David Wagner and Bruce Schneier. Analysis of the SSL 3.0 protocol. In Proceedings of the 2nd Conference on Proceedings of the Second USENIX Workshop on Electronic Commerce - Volume 2, WOEC’96, pages 4–4, Berkeley, CA, USA, 1996. USENIX Association.

[16] D. Brumley and D. Boneh, “Remote timing attacks are practical,” in Proceedings of the 12th conference on USENIX Security Symposium - Volume 12, ser. SSYM’03. Berkeley, CA, USA: USENIX Association, Jun. 2003, pp. 1–1.

[17] I. Ristic. Is BEAST still a threat? https://blog.qualys.com/ssllabs/2013/09/10/ is-beast-still-a-threat, 2013

[18] Hoffman, P., and McManus, P., 2018. DNS queries over HTTPS (doh). Internet Requests for Comments, IETF, RFC, 8484.

[19] Rizzo, J., & Duong, T. (2012, September). The CRIME attack. In ekoparty security conference (Vol. 2012).

[20] Prado, A., Harris, N., & Gluck, Y. (2013). The BREACH Attack.