Lucky Thirteen

TLS/SSL Vunerability

Seminarpaper

Ausgewählte Kapitel der IT-Security

Vorgelegt von:
Jasmin Wellbrock

Personenkennzeichen
1910475028

Abgabe am:
23.11.2022

List of abbreviations

TLS	Transport Layer Security
SSL	Secure Sockets Layer
BEAST	Browser Exploit Against SSL/TLS
CRIME	Compression Ratio Info-leak Made Easy
TIME	Timing Info-leak Made Easy
CBC	Cipher Block Chaining

Keywords

Lucky Thirteen
BEAST
CRIME
TIME

Contents

Chapter 1

Introduction

This paper was created as part of the company's presentation on the Lucky Thirteen vulnerability and is intended to increase awareness of exploits in SSL/TLS. 2.1

1.1 SSL/TLS in General

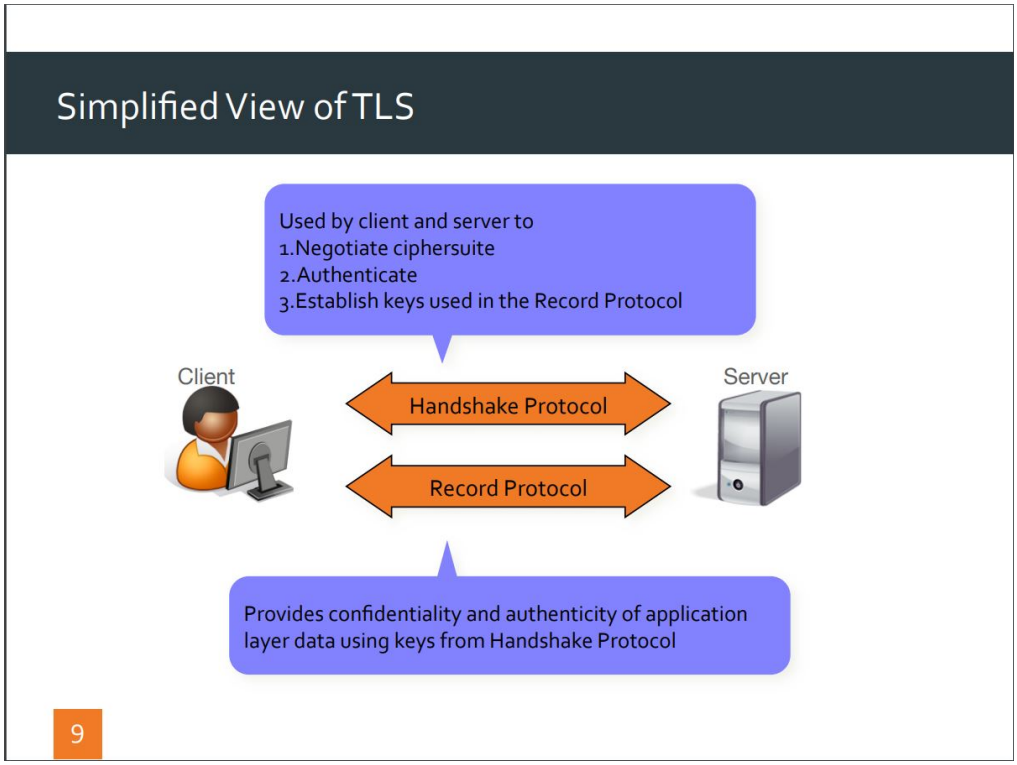

Figure 1.1: TLS
[Pat22]

The TLS protocol, which stands for Transport Layer Security, aims to offer cryptographic security, interoperability, extension, and relative performance. TLS is based on

the now-deprecated SSL (Secure Sockets Layer) specifications (1994, 1995, 1996) developed by Netscape Communications[FKK11]. Although not glaring, the differences between this protocol and SSL 3.0 are enough to prevent TLS 1.0 and SSL 3.0 from working together. TLS is a stateful connection by using a TLS Handshake. When a TLS client and server first start communicating, they agree on a protocol version, select cryptographic algorithms, optionally authenticate each other, and use public-key encryption techniques to generate shared secrets. The Client sends a hello message to the server and the server response to the hello request. The hello phase messages are used to exchange security enhancement capabilities between the client and server. Before a client and server can begin to exchange information protected by TLS, they must securely exchange or agree upon an encryption key and a cipher to use when encrypting data. [SHSA15]

1.2 SSL/TLS Vulnerabilities

This general overview of TLS attacks is intended to apply only to those mentioned in RFC 7457 of February 2015[SHSA15] and not to more recent attacks, as the attacks are intended to be compared to the Lucky Thirteen exploit. Known attacks on TLS/SSL are:[SHSA15]

- STARTTLS Command Injection Attack (CVE-2011-0411)
- BEAST (CVE-2011-3389)
- Attacks on RC4, CRIME attack [CRIME] (CVE-2012-4929)
- the TIME attack later BREACH attack (CVE-2013-3587)
- Certificate and RSA-Related Attacks, such as Brumley03 (CVE-2003-0147)
- Renegotiation (CVE-2009-3555)
- Triple Handshake (CVE-2014-1295)
- Virtual Host Confusion, Denial of Service or Padding Oracle Attacks, including the Lucky Thirteen attack

1.2.1 BEAST (CVE-2011-3389)

Browser Exploit Against SSL/TLS (BEAST CVE-2011-3389) is an attack that exploits a vulnerability in Transport-Layer Security (TLS) 1.0 and older SSL protocols that use Cipher Block Chaining (CBC) mode for encryption. It allows attackers to intercept and decrypt HTTPS client-server sessions and obtain authentication tokens. It does this by combining a man-in-the-middle (MITM) attack with a record-splitting and selected boundary attack. [MC] BEAST relies on the predictable way initialization vectors are generated as part of the encryption process in CBC mode. Due to the predictability of this process and the fixed size of ciphers (i.e. blocks), attackers can manipulate

the boundaries of cipher blocks (the part of the BEAST attack that relates to the chosen boundaries) and slowly reveal the plaintext without damaging it by obtaining the having to decrypt the key.[oST11]

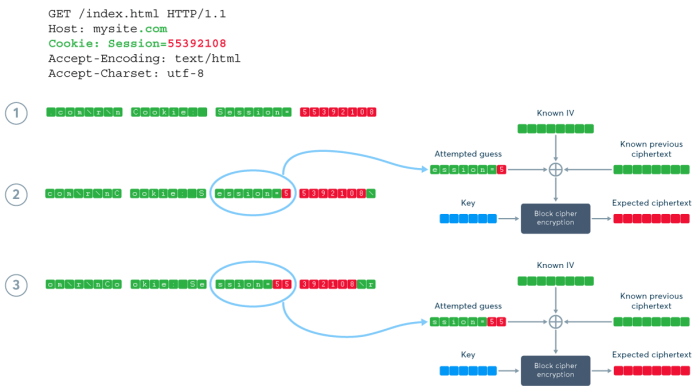

Figure 1.2: BEAST
Source by: https://cdn.invicti.com/statics/img/blogposts/beast-attack-2.png last accessed on: 2022-11-19

1.2.2 CRIME (CVE-2012-4929)

The researchers Thai Duong and Juliano Rizzo created the SSL/TLS exploit known as Compression Ratio Info-leak Made Easy (CRIME). CRIME is a side-channel attack that uses the compressed size of HTTP requests to find session tokens or other confidential information. The method takes use of SSL/TLS-protected online connections that employ one of the two built-in data-compression techniques (DEFLATE or gzip) for lowering network load or speeding up web page loading. Other encrypted and compressed protocols are probably also at risk from CRIME, which is known to attack SPDY and SSL/TLS compression. It is possible to send or store the same amount of data using compression, which uses fewer bits. DEFLATE is the primary data compression technique used in TLS. Lempel-Ziv coding, often known as LZ77, and Huffman coding, are the two subalgorithms that make up DEFLATE. Huffman coding is used to get rid of repetition in symbols, whereas LZ77 is used to get rid of repetition in repeated sequences. The program reads the input, scans for repeated strings, replaces them with back references to the previous occurrence as (distance, length), and then wraps the material in a stream using zlib formatting. The window size is one of the crucial variables in this compression method. The bigger the window size, the greater the compression ratio; it ranges from 1 to 15. Sequence lengths are restricted to 258 bytes, while distances are limited to 32K bytes. A string is output as a sequence of literal bytes if it does not appear anywhere in the previous 32K bytes.

The ClientHello message exchanged during a TLS handshake contains a list of supported compression techniques (by compression, we are discussing only TLS compression and SPDY, HTTP compression is not considered in the scope of this section). The compression technique will be employed, the server replies in the ServerHello message. One-byte IDs serve as the specification of compression techniques. TLS compression is utilized, and it is applied to the whole stream of data being sent. Particularly when

HTTP is utilized, compression is performed to the whole stream of subsequent HTTP requests, including the header. CRIME is a brute-force attack that takes use of a feature of compression routines and tracks changes in the length of compressed data. Despite the complexity of the compression function's internals, this straightforward illustration may demonstrate how the information leak can be taken advantage of.

TLS records can be up to 16 kilobytes in size. When a record is larger than 16 Kbytes, TLS divides it into many records and independently compresses each record. If an attacker is aware of the secret's location, he or she can force TLS to divide the request such that the first record contains just one unknown byte by adding the necessary padding to the request route. After identifying a potential match, the attacker removes 1 byte of padding, and TLS divides the record once again such that the initial record has just one unknown byte. This process is repeated periodically until the attacker discovers the entire secret. The assault can also be improved. The attacker can send a request containing 32 copies of Cookie if the secret value is in base64, meaning that there are 64 different possibilities for each unknown character: secret cookie=X (for 32 variations of the X character). The overall compressed length will be reduced if one of them corresponds to the real cookie. The attacker can attempt again with a 16/16 split once they know which half of the provided alphabet the unknown byte belongs to, and so on. Because

$$2^6 = 64 \tag{1.1}$$

, this focuses on the unknowable byte value after 6 requests.[SF13]

In a single session the same secret/cookie is sent with each request from the browser. TLS has an optional compression feature that allows you to compress data before encrypting it. TLS encrypts the content under her TLS layer, but an attacker can see the length of the encrypted request sent over the network, and that length depends directly on the compressed plaintext data. Finally, an attacker can trick a client into generating a compressed request containing attacker-controlled data in the same secret data stream. These features of browser-based SSL are taken advantage of by the CRIME attack. As SSL/TLS encrypts the data, when the request is transmitted to the server, the attacker who is listening sees an opaque blob, but the compressed plaintext's length is apparent. The request route is the only item the attacker has complete control over. Attack code must be loaded into the victim's browser to carry out the attack. This may be done either by fooling the victim into visiting a hacked or malicious website or by injecting attack code into the victim's legitimate HTTP traffic while connected to an open wireless network. Once this is complete, the attacker has the following methods for forcing the victim's browser to make repeated requests to the target HTTPS website:

- Requests that cross domains
- Placing the payload in the query string of a GET request
- Making use of image tags (Rizzo and Duong's technique)

The attacker then can examine each request that the attack code makes to the server. Every request that successfully guesses the secret will be quicker than ones

that incorrectly guess the secret. Therefore, the attacker can discover the values of the accurate guess and, consequently, the entire secret by comparing the content length. All browsers and servers that implement TLS compression are vulnerable to this attack. When the attack was made public, 42% of servers and 45% of browsers enabled TLS compression, according to test results from the Qualys SSL Lab's SSL Pulse. Because they did not support TLS compression, Internet Explorer, Safari, and Opera were unaffected. Google Chrome (NSS), Mozilla Firefox, and Amazon Silk, three of the most popular web browsers, all enabled TLS compression as they implemented DEFLATE. The attack also succeeded against several well-known Web services, including Gmail, Twitter, Dropbox, and Yahoo Mail, which enable TLS compression on the server side. The attack was successful with all TLS versions, all cipher suites (including AES and RC4), and even when HSTS was turned on and preloaded by the browser manufacturer. Since CRIME uses a brute-force approach, it may be optimized to make O(log(W)) queries, where W is the cookie charset. One prerequisite of the attack may be met by launching a man-in-the-middle assault on a public network using Moxie Marlinspike's modified version of SSL Strip. By posing as a proxy, this program steals the SSL/TLS session that is currently in use and executes a man-in-the-middle attack. To replicate the attack, Krzysztof Kotowicz's proof of concept code is very helpful. Although the pseudocode of Duong and Rizzo functions flawlessly in practice, it lacks a method to synchronize JavaScript with the program monitoring network lengths. HTTP compression is still supported by browsers, and sessions that use HTTP compression are vulnerable to this attack. This attack has been expanded upon by Timing Infoleak Made Simple (TIME). Recently, a novel targeted approach called Browser Reconnaissance and Exfiltration through Adaptive Compression of Hypertext (BREACH) was presented to successfully extract encrypted secrets. There are several potential mitigations, with varying degrees of difficulty, viability for implementation, and functional loss. It is simple to advise defenses to avoid utilizing TLS compression since CRIME attacks the compression technique. Given the complexity of modern websites and the vast array of user-controlled attack vectors, it will be challenging to even disable compressing any content containing secrets or to disable compressing secret data and attacker-controlled data in the same compression stream without flushing the compression state between the two.

Figure 1.3: CRIME(CVE-2012-4929)
source by: https://www.sslprobe.com/images/crime-checks.png last accessed on: 2022-11-19

Therefore, the strategy of turning off TLS compression to address the problem has the most proponents. This solution is quite simple to implement and can be patched through vendor updates with little manual work. Although browsers still support HTTP compression, Chrome and Firefox have both disabled TLS compression (and SPDY compression, if enabled) in their respective browsers. Several other server software packages have also done the same. Only HTTP requests and response headers will be compressed by TLS compression, which is a far less amount of traffic than the body of web application pages that are compressed when HTTP compression is enabled.[SF13]

1.2.3 TIME

Timing Info-leak Made Easy (TIME) is a selected plaintext attack on HTTP answers that was created by Tal Be'ery and Amichai Shulman of Imperva. The CRIME attack paradigm uses the length of encrypted and compressed data to determine the plaintext. To determine the amount of the compressed payload, TIME use this model and timing information differential analysis. To violate SSL/TLS encryption and/or the Same Origin Policy, the attacker simply must have control of the plaintext under the TIME attack paradigm. This theoretically enables any bad website to conduct a TIME attack against its users (SOP).[SF13]

Two practical problems exist with the CRIME assault.

1. The CRIME attack targets HTTP requests, specifically obtaining the cookie value of a victim. The CRIME attack is no longer applicable in those situations since TLS compression has been deactivated in both most browsers and server software.

2. CRIME, the victim must be subjected to a man-in-the-middle attack by the attacker.

These two restrictions were addressed by TIME. TIME changed the attack target from HTTP requests to HTTP replies since the Internet now uses HTTP response compression and advises it as a best practice for performance and bandwidth advantages. Before going into further detail about the TIME assault, go over a few elements that are crucial to its effective implementation. JavaScript and other scripting languages can access Document Object Model (DOM) properties and methods across domain names under the control of the Same Origin Policy (SOP) mechanism. Browsers implement SOP to stop malicious scripts supplied from an attacker site from gathering data from another site. But img and other multimedia tags are not covered by SOP. As a result, a weird scenario arises in which some of a page's contents can be controlled programmatically, but not all of them. The embedding page has access to well-documented SOP information leaks in this scenario, including the success or failure of the load of the embedded content and the load duration of the content. Due to this violation of SOP, certain data can leak from one domain to another.

Round-Trip Time (RTT) is the sum of the time it takes an IP datagram to travel from the sender to the destination and the time it takes the sender to get the acknowledgement from the receiver. The transmission timings between the two places account for this time lag. A datagram with a smaller MTU (Maximum Transmission Unit) size will have a shorter RTT than one with a greater MTU size. The TIME attack's fundamental strategy relies on RTT timing discrepancies and HTTP compression to deduce the contents of an HTTP request sent by the browser that includes sensitive data.
The greatest size of an IP datagram that can be sent using a particular data link connection is called the Maximum Transmission Unit (MTU). For most WAN links, the MTU value, which is a LAN design parameter, is a number that both ends of a link mutually agree to use. Various links may have quite varying MTU sizes (e.g., typically from 128 B up to 10 kB). 1500 bytes are now the most common Path MTU on the Internet. IP Fragmentation occurs when a packet is bigger than the MTU limit. The difference in RTT between the two packets—one inside the MTU range and the other at least one byte larger—is thus large enough to be measured. The maximum number of data octets in a TCP segment, excluding the TCP (or IP) header, that can be transferred in a single IP datagram across a connection is known as the Maximum Segment Size (MSS). Although MSS can theoretically be 65495 bytes in size, in practice it is typically less than the outgoing interface MTU by the combined amount of TCP and IP header bytes. The formula for calculating MSS is as follows: MSS = MTU - sizeof(TCPHDR) - sizeof(IPHDR) [TCPHDR stands for TCP header and IPHDR for IP header].[SF13]
By enabling a transmitting device to deliver all packets inside the specified window size before receiving an ACK, the TCP Sliding Window System protocol optimizes the byte stream. The attacker wants to make the compressed data's length beyond the boundaries of one TCP packet. The attacker then increases the quantity of compressed data to match the number of TCP packets permitted by the TCP window (i.e., sliding window). Any extra packet generated because of inaccurate assumptions while the TCP window is maxed out results in a full round trip and a large delay since it must first wait for the ACK from previous packets before sliding the TCP window and allowing for another packet to be sent. Consider this example of the assault method now. Consider a user input where the payload is "secret element = unknown data." The answer contains the secret element and its value, and whatever the user enters is likewise mirrored in the response. The user input is anything at all in the first iteration, and the return size is 1024 bytes. Now, due to compression, the response size will be 1008 bytes if the user input is "secret element = a". As a result, it will go faster than the initial iteration. Like this, while making repeated queries, the attacker will determine which character in which position in the payload has the quickest response time, which will only occur in the event of a successful guess. The secret element's value will be those precise values. An attacker can submit numerous queries to a target website with data they control while also monitoring the response time by injecting malicious code or JavaScript. By taking advantage of the SOP weakness, JavaScript can operate. The attacker can identify 1 byte discrepancies if the payload length is exactly on the window boundary since it will result in an extra RTT with a noticeable delay. The response time does not include additional RTT when the attacker-controlled

data agrees with the accurate guess. The padding is changed to retain 1 byte excess beyond the boundary for each byte of correctly guessed secret data in the response. The attacker can therefore effectively discover the entire secret simply figuring out the time difference. The attacker can entirely learn the secret without using eavesdropping by making repeated queries. This secret applies to all sensitive information contained in the response, including the username, CSRF token, bank account number, etc. It is not just the session cookie that is covered by this secret. The attacker must be aware of details about the HTTP response to carry out the TIME attack. The location of the secret element, the prefix and suffix of the secret element (secret/cookie are frequently structured so they have a fixed prefix or suffix), and a location to insert a chosen plaintext are all necessary for the attacker to know (many applications embed user input expressed by HTTP parameters within their response). The attacker can get this information by looking at a few of the target website's answers. The attacker wants to determine the secret element value (also known as the cookie value). The attacker uses JavaScript to make HTTP requests, and the response time exposes the request size. The minimal reaction time can be collected from the range of response time values to reduce network noise if the operation is conducted again for a certain amount of time. By making a few calls to the targeted application using this technique, the attacker may even utilize this timing information to determine if the victim is still logged in or not. The answer will include further information pertinent to the user account if the user is signed in. The response will be the application's login screen, which will include less information than in the preceding scenario if the user is not signed in. As a result, both the response time and the response size will probably be increased.

The TIME attack can take use of these components' positions and known prefixes and suffixes as hidden elements. While the precise details of these pieces vary depending on the application, they are all present. Users' input must be mirrored back in the response for this attack to be effective (not to be confused with the reflected cross-site script attack). If the other components of the answer are kept constant, changing the reflected user input changes the response's size and content. There will be no apparent changes in the HTTP response's compression size and, hence, no apparent changes in response time if the user's input, which contains the guess of the secret element, is not reflected. Random network noises can impact timing measurements. Random delay can be introduced by congestion and packet loss in any of the components (client, router, and server) along the routing path. It has been discovered that transmitting the payload a few times (suppose ten times) and using the smallest timing value will accommodate for such unpredictable delay impacts. Prior to being embedded into the response, user input is encoded to thwart injection attempts. The assault target is therefore mostly restricted to alphanumeric characters.

The integrity and confidentiality of SSL/TLS are violated by the time information leaks and the existence of the TIME attack, even if this timing attack does not directly exploit any SSL/TLS flaw. The countermeasures listed below focus on modifying how the application is created rather than how the protocol operates.[SF13]

- It can be acceptable to obstruct statistical analysis by adding random time delays to the decryption for any timing attack, although it is not entirely successful. If an attacker can collect enough samples, they can average many observations,

which eliminates randomness. Random delay can merely lengthen the attack's duration but not make it impossible.

- For an application to restrict the display of basic multimedia material, including images on other apps, the browser should support and follow the "X-Frame-Options"34header for any content inclusion (not only IFRAME) enabling apps to take charge of how their content is presented on other domains.
- Applications must strictly limit how much user input appears in the returned information. Additionally, the application must be able to accept unknown variable input (such as an additional variable in the URL) and avoid reflection in the response.
- To prevent repeating requests from an attacker, it might be helpful to enable anti-automation measures like CAPTCHA and CSRF tokens.[SF13]

Chapter 2

Lucky Thirteen

This chapter deals with the TLS/SSL vulnerability Lucky 13 and its countermeasures.

Figure 2.1: LuckyThirteen
Source by: https://news.softpedia.com/news/Experts-Explain-the-Risks-Posed-by-the-Lucky-13-Attack-326791.shtml last accessed on: 2022-11-09

2.1 The Vulnerability in General

In TLS (TLS 1.0) and SSL 3.0 CBC-MAC encryption was used to provide integrity of the message. The MAC is added to the plaintext and then it is filled with bytes (Padding) up to 255 bytes, but only the message block is CBC-encrypted, not the padding. So, the padding is not protected, which allows an attacker to tamper with the padding and perform a padding oracle attack. In the CBC mode, an initial key and a key are used to encrypt the plain text. When decrypting the message, the padding is checked first. If there is a valid padding, only then is the MAC checked. Otherwise, the server throws an error indicating whether invalid padding or MAC error occurred. The padding oracle attack uses CBC decryption to discover the plaintext by modifying the previous block of ciphertext. An attacker can modify the encrypted message based on these error messages, and after repeated requests, the message can eventually be decrypted by the server without the encryption key. This padding oracle issue was addressed by removing any explicit error messages that could indicate to the

attacker whether the padding check or the MAC check caused a decryption error. This solution left the implementation vulnerable to the possibility of a timing attack since the attacker can see the time differences in the Server replies on incorrect padding.

In TLS 1.1 and later, whenever a record cannot be decrypted (due to a bad MAC or padding error), the TLS server terminates the session. This was implemented to prevent an attacker from repeatedly sending requests to decrypt the encrypted message but padding oracle attacks can work across different sessions provided the victim reinitiates the session after aborting and the secret appears in every stream at the same position. Browsers and HTTPS are designed to meet both requirements. The TLS standard 1.2 states that the MAC must be checked even if the padding fails, considering that the value of the padding is zero, to address the time difference issue. If the padding check fails, there is no way of finding out the size of the actual message and the number of padding bytes to remove. Therefore, no way to calculate the correct MAC. All according to specification Blob is used to calculate MAC. As a result, the MAC calculation can take a little longer if the padding is damaged. And this subtle timing error is exploited by the Lucky Thirteen attack to decrypt the encrypted message. TLS typically uses HMAC with MD5, SHA1, or SHA256 as the hash function. Each of these hash functions processes messages in 64-byte chunks. The hash functions contain an 8-byte field plus some special hash function padding, which means that a one-block message can only contain about 55 bytes of real data (which includes the 13-byte record headers included). The attack tries different values in the penultimate ciphertext block, this time in the last two bytes, to force the last plaintext block to contain the correct two padding bytes. The attacker wants 55 bytes to be authenticated using HMAC-SHA1 as this can be mathematically distinguished from 56 or 57 bytes, as if it goes a single byte past 55 the hash function must run a whole extra round resulting in a tiny (500 -1000 hardware cycles) delay.[SF13]

The ability to intercept the client-server connection in order to read the clear text TLS handshake messages and inject changed ciphertext is a need for the attack. The most typical setting for this is an open Wi-Fi network. The attacker can supply the target browser with custom JavaScript (which does not have to come from the target webserver; it could even be provided on a page that is not HTTPS) or client-side malware to force it to open several connections. Each of these connections will have cookies at a known place in the HTTP stream because of the way the HTTP protocol is designed. To ensure that there is only one unknown byte in the target block at each stage of the attack, the malicious JavaScript can also manage the cookie's position.

The goal of the attack is to intercept a message, change it, and add TLS padding so that it exceeds that 55-byte limit. The same statement, nevertheless, would appear below it if the padding had been properly removed. The decryption procedure will execute MAC operations on the lengthier version of the message when an attacker tampers with it (damages the padding), which will take noticeably longer to compute than when the padding is intact. It is feasible to measure clearly if the decryption was successful or not by repeating this procedure thousands of times to remove noise and network jitter. Once the attacker has the knowledge, all that is left to do is execute a typical padding oracle assault.

Any implementation that complies with TLS version 1.1 or 1.2 or DTLS version 1.0

or 1.1 is susceptible to the attack. It also applies to SSL 3.0 and TLS 1.0 implementations that have safeguards put in place to thwart an earlier padding oracle attack found several years ago. The CBC-mode encryption used by all TLS and DTLS ciphersuites is possibly weak.

The assault is likely to be utterly infeasible due to the delay produced by the different Internet infrastructures. Due to the minimal or nonexistent latency, it may, nonetheless, be useful against fast internal networks. The server's time to re-initiate all these connections is the only practical constraint on such a cookie attack. TLS handshakes take a long time, and this exploit can need thousands of connections per byte. As a result, in real life, a TLS assault would probably take days.[SF13]

2.2 Mitigation/Countermeasures

The following mitigating strategies were also presented by Nadhelm and Kenny, however they also have limitations. This Lucky 13 attack can be reduced by carefully implementing all MAC-then-Encode-then-Encrypt (MEE) techniques. The processing times for good and invalid input will be the same if uniform processing time is applied to decrypt ciphertexts of a given size. To do this, it is necessary to eliminate significant temporal variations during MAC processing of the ciphertext, regardless of the characteristics of the underlying plaintext. Any time attack can be mitigated by including random timing delays in the decryption process. However, this countermeasure will not totally prevent the attack; rather, it will merely lengthen the decryption procedure by requiring more samples, which may render the assault exceedingly improbable when combined with the usual network delay. The optimum solution is to use an authorized encryption technique, such AES-GCM or AES-CCM, however this is only supported by TLS 1.2, which is not yet commonly used. [SF13]

Chapter 3

Conclusion

As shown in the table below, for every TLS vulnerability there is a countermeasure, it is also important to keep browsers and operating systems up to date. Thus, the security benefits of TLS outweigh its weaknesses.

Overview of the vulnerabilities and their countermeasures:

Name	Vulnerability	Countermeasure
Lucky 13	Recover plaintext from TLS connection when CBC-mode encryption is used	Implement uniform processing time to decrypt ciphertext of a given size, so the processing time of valid and invalid input will be the same.
BEAST	Mounts the attack by choosing a guess for the plaintext that is associated with a known ciphertext	Injecting empty fragments into the message to randomize it. 1/n-1 record splitting where a single byte of the plaintext is injected in each record.
CRIME	Can be used to discover session tokens or other secret information based on the compression size of HTTP request	Disable compressing any content containing secrets or to disable compressing secret data.
TIME	Uses CRIME attack model and timing information differential analysis to infer the compression payload's size	Add random time delays to the decryption. Do not reflect inputs of users in the response. Handle unknown variable inputs and prevent it from reflecting in the response.

Table 3.1: Overview Vulnerabilities and Countermeasures

Appendix A

Directories

List of Figures

List of Tables

Bibliography

[FKK11] Alan O. Freier, Philip L. Karlton, and Paul C. Kocher. The secure sockets layer (ssl) protocol version 3.0. *RFC*, 6101:1–67, 2011. 2

[MC] https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2011-3389 MITRE Corporation. Cve-2011-3389. last accessed on: 2022-10-08. 2

[oST11] National Institute of Standards and Technology. Cve-2011-3389 detail, 2011. [On-Line], Accessed October 8, 2022. `https://nvd.nist.gov/vuln/detail/CVE-2011-3389`. 3

[Pat22] Kenny Paterson. Lucky 13, beast, crime, ... is tls dead, or just resting?, 2022. https://www.ietf.org/proceedings/89/slides/slides-89-irtfopen-1.pdf. 1

[SF13] Pratik Guha Sarkar and Shawn Fitzgerald. Attacks on ssl a comprehensive study of beast, crime, time, breach, lucky 13 & rc4 biases. *Internet: https://www. isecpartners. com/media/106031/ssl_attacks_survey. pdf [June, 2014]*, 2013. 4, 6, 7, 8, 9, 11, 12

[SHSA15] Yaron Sheffer, Ralph Holz, and Peter Saint-Andre. Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS). RFC 7457, February 2015. 2