SSL/TLS Protocol and Implementation Based Attacks

Aditya Shenoy Uppinangady (40216499), Charan Pechhetty (40221337), Gokula Rani Vallabhu (40161606), Pavan Koushik Nellore (40195824), Sai Chandra Sekhar Reddy Dwarampudi (40189233), Manoj Narayana Katragadda (40203239), Lakshmi Narasimha Patsamatla (40217793), and Sudeep Kumar Chamarthi (40184676), *MEng Information System Security*

***Abstract*— Security was not given much consideration when the Internet was first created. Because to this inherent insecurity, the fundamental communication protocols are dependent on the sincerity of all parties involved. It could have worked in the past, when the Internet only had a few nodes—mostly universities—but it absolutely fails now that everyone is online.**

Cryptographic technologies like SSL and TLS are developed to provide safe communication across unreliable infrastructure. This means that if these protocols are implemented correctly, you can open a communication channel to any service on the Internet and be reasonably certain that you're talking to the right server. You can then exchange information in confidence knowing that your data won't end up in the wrong hands and will be received intact. These protocols safeguard the transport layer, or communication connection, from whence the term TLS derives.

***Index Terms*—SSL, TLS, Protocol, Security, Session, Cryptography, Socket, Attack, Library, Process**

I. INTRODUCTION

Secure socket layer (SSL) provides a secure communication channel for two parties that want to communicate with each other. This secure communication channel provides privacy, authentication and data integrity between the two communicating applications. This was first developed by Netscape in the mid-1990s [1]. In 1996, efforts were undertaken to migrate SSL from Netscape to the Internet Engineering Task Force (IETF) by the newly formed TLS working group [2]. This standard version for the internet was now called Transport Layer Security (TLS). The most well know application of SSL/TLS is securing web browser sessions [1]. TLS can also be used to create a Virtual Private Network (VPN) by tunneling an entire network stack to provide authentication and encryption of Session Initiation Protocol (SIP) [1]. For any client-server interaction, TLS can be used. SSL/TLS protect the Transport Layer in the TCP/IP protocol stack. According to their priority, some of the main objectives of SSL/TLS are mentioned below [1]:

- **Cryptographic Security**

The goal of this is to make it possible for two parties to communicate securely.

- **Interoperability**

Using standard cryptographic settings, various SSL/TLS libraries must be able to communicate with one another.

- **Extensibility**

The SSL/TLS framework's implementation needs to be independent of the cryptographic basics. As a result, this implementation can now switch to alternative cryptographic primitives without having to develop new protocols.

- **Efficiency**

The ultimate objective is to successfully complete all the aforementioned objectives.

Aditya Shenoy Uppinangady (40216499) (e-mail: shenoy.aditya796@gmail.com)
Charan Pechhetty (40221337) (e-mail: Charanpechhetty3@gmail.com)
Gokula Rani Vallabhu (40161606) (e-mail: vgr.gokularani@gmail.com)
Pavan Koushik Nellore (40195824) (e-mail: pavankousgik6@gmail.com)
Sai Chandra Sekhar Reddy Dwarampudi (40189233) (e-mail: chandrasekhar3197@gmail.com)
Manoj Narayana Katragadda (40203239) (e-mail: manojnarayanakatragadda@gmail.com)
Lakshmi Narasimha Patsamatla (40217793) (e-mail: vikramadityavarma102@gmail.com)
Sudeep Kumar Chamarthi (40184676) (e-mail: sudeepkumar14628@gmail.com)

II. PROTOCOL HISTORY

Netscape created the initial SSL version, which was never made publicly accessible. Public access to SSL 2, a 1994 release from Netscape, was made possible. Due to significant security flaws in this version, Netscape was obliged to issue SSL 3 in late 1995 [1]. In 1996, the newly formed TLS working group was tasked with migrating SSL from Netscape to the Internet Engineering Task Force (IETF). The protocol was called TLS and the first version was released in 1999 [1]. TLS extensions were introduced in 2003. TLS 1.1 was released with several security fixes. Hard-coded security primitives were removed from the specification and support for authenti- cated encryption was added in TLS 1.2, released in 2008 [1]. TLS 1.3 was released in 2018 which included security and performance improvements. This is used by HTTPS and other network protocols for encryption. In addition to speeding up TLS handshakes, TLS 1.3 removed support for earlier, less secure cryptographic capabilities.

III. PROTOCOL OVERVIEW

The protocol consists of two layers. The Record Protocol is the lower of protocol's two layers, while the Handshake Protocol, Alert Protocol, and Change Protocol are its three subprotocols. Protocol Cipher Specific Data compression, encryption, authentication, and fragmentation are all features of the Record Protocol [1].

The Handshake Protocol enables the negotiation of encryption and Message Authentication Code (MAC) methods as well as cryptographic keys between the two parties (server and client). Alert messages are sent using the Record Protocol and the Alert Protocol [1].

To switch between different cypher specifications, it uses the Change Cipher Spec Protocol [1].

A. TLS Handshake

Every TLS connection starts with a handshake. The two parties will carry out a complete handshake in order to negotiate a TLS session if the client and server have not already established a session [1]. The client and the server will engage in four primary activities during this handshake as illustrated in Fig. 1:

- Swap capabilities and decide on the appropriate connection characteristics
- Verify the provided certificate(s or authenticate in another way
- Decide on a shared master secret that will be used to secure the session
- Confirm that no outside entity has altered the handshake messages

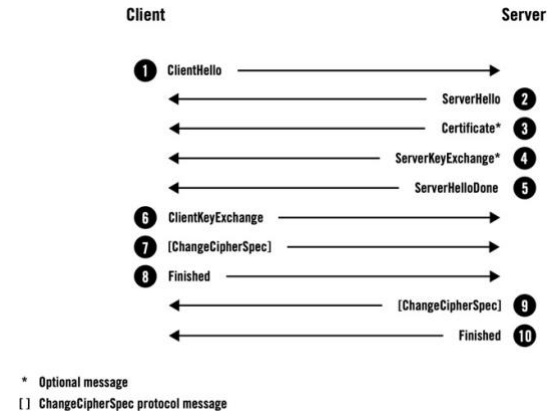

Fig. 1. 1 Complete handshake with server authentication [1]

The client initiates a fresh handshake and offers the server its capabilities. Connection settings are selected by the server. The server transmits its certificate chain (only if server authentication is required). The server delivers any additional data necessary to create the master secret, depending on the key exchange option that is chosen. When the negotiation on the server's end is complete. For the purpose of creating the master secret, the client submits extra information. Informing the server, the client switches to encryption. The handshake messages transmitted and received by the client are hashed using MAC and sent back. Informing the client of the change, the server switches to encryption. The handshake messages the server received and transmitted are sent as a MAC. If there were no issues, the connection is now established, and the parties may start sending application data.

IV. Implementation Issues

A.

1) Heartbleed: The widely used OpenSSL encryption software package contains a security vulnerability known as Heartbleed, which was found in 2014. With the help of this vulnerability, an attacker can steal secret encryption keys, identities, and other confidential data from the memory of the system that is attacked. This attack is caused by an error in the implementation of the TLS (Transport Layer Security) heartbeat extension. This bug's reference is CVE-2014-0160.

CVE-2014-0160:

As shown by reading private keys, the Heartbleed flaw, which affects d1_both.c and t1_lib.c, enables remote attackers to steal private data from process memory via specially crafted packets that result in a buffer over-read. This issue is due to not handling Heartbeat Extension packets properly in TLS and DTLS implementations in OpenSSL 1.0.1 before 1.0.1g.

One line of code contains the mistake that led to the Heartbleed issue. Data is copied using the command memcpy. Payload is the amount of data being transmitted, while bp and pl are the starting and ending points of the copy, respectively. A TLS-encrypted connection between two computers involves the exchanging of heartbeat messages. A Heartbeat response must contain an identical replica of the payload from the request. Attackers may send heartbeat requests with length field values that exceed the actual length of the payload. When responding to Heartbeat queries, the system's OpenSSL processes don't verify that the payload size adheres to the length field specification. The computer then transfers extra information to the heartbeat response from the server memory.

a) :

: Test Environment :: We used a computer running Windows 11 with a Core i7 2.80 GHz processor and 16 GB of RAM for conducting the attack. Kali Linux OS, installed in a virtual box, is used for conducting the attack, which is performed on the Ubuntu OS of version 8.04 with an older openssl version of 0.9.8g, where the heartbleed fix is not yet implemented.

Steps:

1) In kali linux, Metasploit framework is used for executing the heart bleed attack. Metasploit console can be started by using the following command.(Fig 2)

Fig. 2.

2. Using the Metasploit framework's built-in search function, look up the Heartbleed module. By using this path to the heartbleed module, run the following command to load the module.(Fig 3)

3. Using the info command, the RHOSTS parameter is set to a target website that needs to be attacked. RPORT is set to 8443, and the verbose parameter is set to true.(Fig 4)

Fig. 3.

Fig. 4.

4.Targeted server can be attacked using the exploit command(Fig 5)

Fig. 5.

5. This attack will result in the leak of some data from random memory locations on the targeted server. We do not have control over its location or the ability to select the desired data.

We can repeat this process in order to extract the sensitive data. Each time, a new set of data can be obtained.

Mitigation:

The Heartbleed bug is fixed in version 1.0.1g of the OpenSSL library. The bug is fixed by adding a check before copying the data. From the code snippet attached below, the first part of this code checks to make sure the heartbeat request isn't 0 KB, which could lead to issues. The second portion confirms that the request is, in fact, as lengthy as it claims to be.

Code snippet:(Fig 6)

2) *Gotofail:* Some versions of iOS (iPod, iPad, and iPhone) and OS X (laptop/desktop) are vulnerable due to the gotofail bug.

```
/* Read type and payload length first */

if (1 + 2 + 16 > s->s3->relent)

return 0;

/* silently discard */

hbtype = *p++;

n2s(p, payload);

if (1 + 2 + payload + 16 > s->s3->rrec.length)

return 0;

/* silently discard per RFC 6520 sec. 4 */

pl = p;
```

Fig. 6.

The "gotofail" — failing to verify a digital certificate — is the source of the issue. Because that validation check no longer takes place, this problem is quite significant. Fig. 7 demonstrates how a redundant "goto fail;" (red) phrase was added to the code, omitting a necessary check (in green).

As a result, the user wouldn't see any errors and might connect using a fake certificate, giving the attacker the opportunity to play man-in-the-middle and intercept/inject data [5].

Visit gotofail.com to see if your Apple device is susceptible to this bug Fig.8 and Fig. 9.

Mitigation:

An update patch provided by Apple fixes the bug iOS and OS X operating systems. This can be download from **Settings> General > Software Update** [5].

```
    if ((err = SSLFreeBuffer(&hashCtx)) != 0)
        goto fail;

    if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
        goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;

        err = sslRawVerify(ctx,
                           ctx->peerPubKey,
                           dataToSign,                /* plaintext */
                           dataToSignLen,             /* plaintext length */
                           signature,
                           signatureLen);
        if(err) {
                sslErrorLog("SSLDecodeSignedServerKeyExchange: sslRawVerify "
                    "returned %d\n", (int)err);
                goto fail;
        }

fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;
```

Fig. 7. shows the redundant goto statement present in the code base [5].

3) *Mod_ssl remote code execution:* mod_ssl is a module that provides SSL and TLS support in Apache HTTP server. While trying to cache SSL sessions, mod_ssl prior to 2.8.7

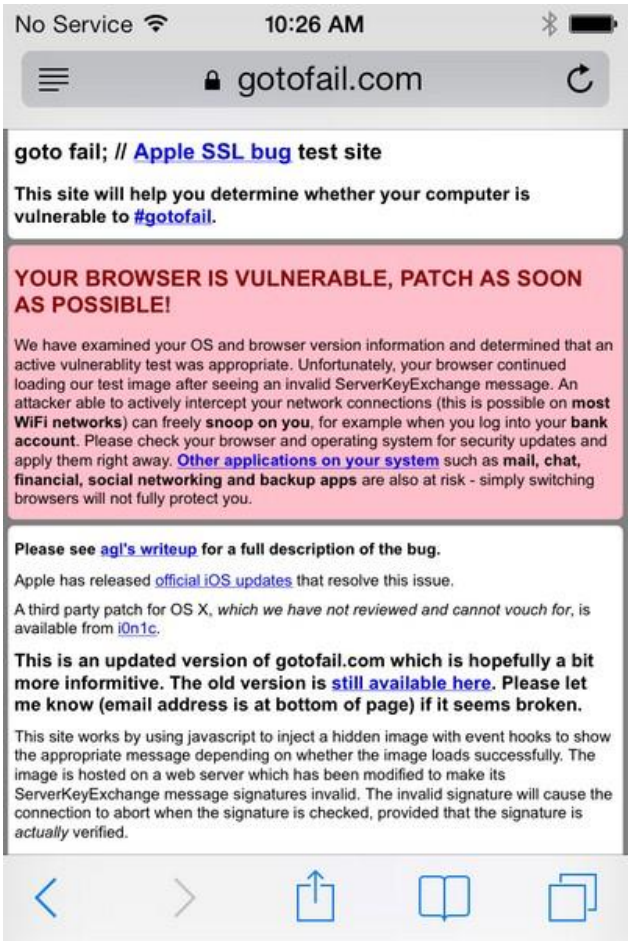

Fig. 8. Visiting gotofail.com on iOS's safari browser [5].

Fig. 9. Visiting gotofail.com on OS X's safari browser [5].

is vulnerable to buffer overflow memory corruption attack [3]. This permits the alteration of any system file as well as enables remote code execution. This happens because the module uses the i2d_SSL_SESSION method incorrectly, which allows remote attackers to use a large client certificate generated by a large, serialised session and issued by a trusted Certificate Authority (CA) to exploit a buffer overflow and execute arbitrary code [4].

The exploit C code is complied using the gcc compiler with the crypto library linked as exploit code requires openssl development library Fig. 10.

Mitigation:

mod_ssl module must to be updated to a version beyond 2.8.7 [4].

```
gcc exploit.c -o exp -lcrypto
```

Fig. 10. shows the how the code is compiled using the gcc compiler.

Fig. 11. Shows a server running mod_ssl 2.8.4

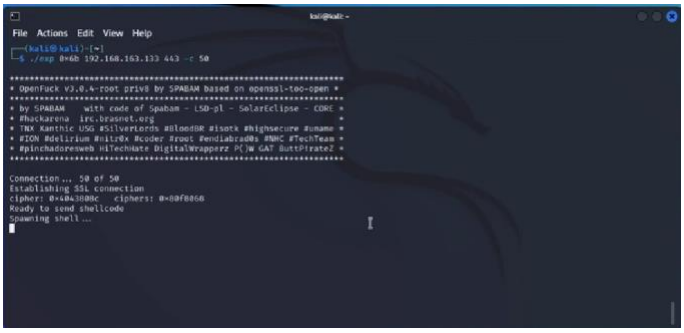

Fig. 12. mod_ssl exploit spawning a shell.

4) *GnuTLS use after free exploit:* gnutls is a library that provides implementation for SSL, TLS and DTLS protocols. It provides simple C APIs to access secure communication. The verify_crt API was found to be vulnerable to memory corruption attack in version 3.6.6 . A rogue server or active network attacker can compromise any client or server that uses gnuTLS to verify X.509 certificates [7]. A vulnerability known as Use-After-Free (UAF) arises from improper use of dynamic memory while a program is running. An attacker can utilize the error to compromise the program if, after releasing

Fig. 13. Shell spawned using mod_ssl exploit

a memory region, a program fails to remove the pointer to that memory [6].

The primary flaw here is that the cleanup path for signature->data in _gnutls_x509_get_signature is not cleared [7].

```
cleanup:
        gnutls_free(signature->data); // <- pointer in datum parameter freed, but not cleared
        return result;
}
```

Fig. 14. 5 Shows thesignature->data not being assigned to NULL after being freed.

Running certtool with a malicious certificate causes the program to crash.

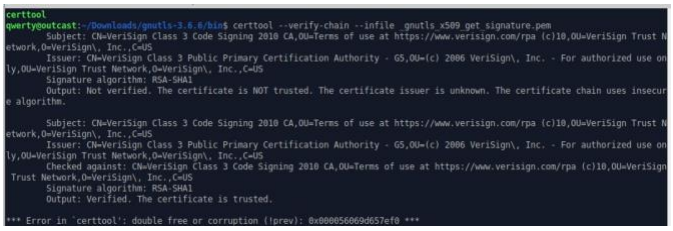

Fig. 15. 6certtool crashes when malicious certificate is processed for verification.

Mitigation:

Updating gnuTLS from 3.6.6 will fix the issue as in subsequent versions the signature->data points to NULL after being freed.

V. Protocol Attacks

A. BEAST

BEAST stands for Browser Exploit Against SSL/TLS. Beast attack targets network flaws in SSL 1.0 and earlier TLS protocols. The attack was initially carried out in 2011 by security researchers Thai Duong and Juliano Rizzo, but Phillip Rogaway first identified the potential vulnerability in 2002.

According to the 2020 Acunetix Web Application Vulnerability Report, weak TLS 1.0 was still enabled on 30.7% of scanned web servers, leaving them open to the BEAST attack.

This demonstrates how, despite the introduction of numerous new features in software enhancing security, old threats remain a significant challenge for enterprises. This situation also applies to SSL/TLS flaws like OpenSSL Heartbleed, BEAST, BREACH, or POODLE.[15]

How Does the BEAST Attack Work:

The Secure Sockets Layer (SSL) protocol is replaced by the Transport Layer Security (TLS) protocol. These are cryptographic protocols that enable you to encrypt communication between a web browser and a web server using various cipher suites. This makes it hard for someone to overhear the conversation and take secret information. Using man-in-the-middle attack strategies, attackers may be able to listen in on the communication between a web server and a web browser. If they do and there is no encryption, they can see all the data sent back and forth between the web server and web browser, including passwords, credit card numbers, and other personal information.

Encryption could also have flaws and be broken. TLS 1.0 (and ear- lier) encryption can be swiftly compromised, the researchers discovered, allowing the attacker an opportunity to overhear the discussion. The attacker can trick the client into thinking that it can only utilize TLS 1.0 if your server supports that protocol.

An attack to downgrade a protocol is what this entails. The attacker can then eavesdrop using the BEAST attack.[15]

Technical Details of BEAST:

Block ciphers are used for symmetric encryption in the TLS protocol. The same key is required to encrypt and decode the message in symmetric encryption. Information is encrypted in blocks of a defined length using block ciphers. The final block is padded when there is insufficient data for it. DES, 3DES, and AES are three common block ciphers. Any encryption could be easily broken if the same data and key always generated the same encrypted information. TLS employs initialization vectors for this reason. This indicates that random information is used to seed encryption. By doing this, even if you repeatedly employ the same data and key, the encrypted content you get each time will be unique. Therefore, it would not be practical to seed each block of a block cipher with random data. This is why cipher block chaining is also used with SSL/TLS (CBC). A logical XOR operation is used to bind blocks together. In actuality, this means that the value of every block is influenced by the value of the block before it. An encrypted value for some original data consequently depends on the block of data before it.[15]

The Attack Technique

Everything can be broken, it simply depends on how long it takes, is the fundamental tenet of code breaking. For SSL/TLS ciphers, the same idea remains true. Even a strong cypher can be broken. Simply expressed, it is impractical to break and impossible to break with existing computing power in a reasonable amount of time. By testing several combinations to check if they produce the same outcome using the same initialization vector, the attacker could crack a block cipher (which they know). They can only check it for an entire block at once, and a block may include, for instance, 16 bytes. Because of this, each block would require the attacker to test 25616 combinations (3.4028237e+38) in order to check it.

The BEAST attack makes this considerably easier by requiring the attacker to predict a single byte at a time. If the

attacker only need one piece of secret information, such as a password, and can guess the majority of the data (such as HTML code), it is possible to accomplish that. The attacker can then carefully test the encryption by choosing the proper data length such that they only have one byte of information in a block that they are unfamiliar with. The block can then be tested using only 256 different permutations of this byte. They then carry out the same procedure for the following byte, eventually obtaining the complete password.[15]

What is the BEAST vulnerability?

BEAST can exploit a server employing block cyphers and TLS 1.0. This is as a result of CBC's inherent flaw, which enables attackers to act as man-in-the-middle and carry out the remaining attacks. Many prerequisites must be satisfied for BEAST to be feasible:

An attacker must be able to watch the network connection and the client and server sessions, or "sniff" them. A block cipher must be used with a vulnerable version, such as TLS 1.0 or an older SSL protocol, or a downgrade must be possible. It must be likely for an applet or JavaScript injection to circumvent a website's same-origin policy.[16]

Recovering Information Without Decrypting It:

Consider a man-in-the-middle attacker who is capable of inserting data into TLS 1.0 transmission by sniffing it. An attacker can insert a specially constructed data block into a message and then check to see if the encrypted block this produces matches the equivalent block in the actual message stream if they are aware of the type of data being sent and where it is in the message. If so, the attacker has found the plaintext block and the guess that originally injected was accurate. If not, they can attempt once more using several likely combinations. It's referred to as a "record splitting attack."TLS 1.0 uses XOR to encrypt a data block in CBC mode by combining the plaintext block, the previous ciphertext block, and the initialization vector. A targeted plaintext attack can be launched by an attacker by choosing a likely block of data, XORing it with the IV and the block of ciphertext preceding it, and injecting the result into the session. This is possible because XOR can be reversed. What is known is:

- All Ciphertext is known.
- Next message IV is previous block.
- Attacker can inject some plain text.

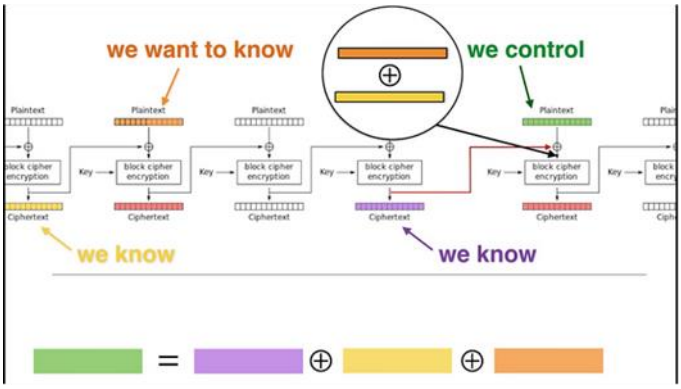

Fig. 16. Overview of CBC attack on TLS. [19]

Purple and yellow in the above diagram are recognized as encrypted text. In order for both red cipher texts to match, orange must be changed. It will be difficult to guess those many bytes because AES has a block size of 16 bytes, or 128 bits. Instead, we design the URL so that we only need to guess one byte. Similar to what we did in the POODLE attack.Orange part would become: 15 byte of known value || 1 byte of unknown. After the initial message is encrypted, we already know the cipher text that corresponds to it. Then, when we get to the "control" phase, we already know the values of 15 bytes, and after 28 iterations, we can guess the single byte.The attacker might also make another attempt with the remaining bytes. [23]

The Aftermath of the BEAST Attack:

The BEAST attack left behind the lesson that even the smallest gap in security can become a wide-open floodgate if left untreated for an extended period of time. It highlighted how crucial it is to identify vulnerabilities as soon as possible and make sure that the solutions are used in practice rather than simply in theory. It certainly affected individuals who work in IT security. Even now, hackers frequently take advantage of vulnerabilities in out-of-date software.[18]

Mitigation:

- Support TLS 1.1 onward on server.
- 0/n or 1/n-1 approach taken by Chrome.

B. Lucky 13

LUCKY13 is a cryptographic timing exploit that can be used against Datagram Transport Layer Security (DTLS) and Transport Layer Security (TLS) protocol versions that use the Cipher Block Chaining (CBC) method of operation. This can be mentioned as a man-in-the-middle attack. The TLS protocol, which acts as replacement to the safe Sockets Layer (SSL) protocol, ensures anonymity, data integrity, and safe communication between networks or apps. The vulnerability that allows the SSL LUCKY 13 to occur impacts TLS 1.1 and 1.2, as well as DTLS 1.0 or 1.2 versions.

Vulnerability in SSL Lucky 13:

The Vulnerability in this attack is the flaw in specifications of TLS and also this attack can be said as padding oracle attack . The attack is a more sophisticated form of padding oracle attack that takes advantage of varying computation periods based on whether the plaintext is padded with one or two bytes. The attack provides for complete plaintext recovery in OpenSSL, but only a partial plaintext recovery attack in GnuTLS. An attacker can retrieve up to four bits of the last byte of plaintext chunks. The final output of this attack is that the attacker can be able to read the sensitive information.

Triggering of SSL Lucky 13 attack:

This attack depends on the variation in the processing times between 2 Byte of correct padding in TLS Messages with 1 byte of correct padding in TLS messages. The TLS messages with 2 byte are somewhat faster in processing so that the difference can be detected during the TLS messages arrival. Ciphertexts which are generated to attack can be replaced in the place of plain text so as to establish the difference during some TLS Sessions. This difference will creates an error

message in network and it gives this as a result. By performing this attack repeatedly and by calculating the responses the different padding conditions can be separated from each other

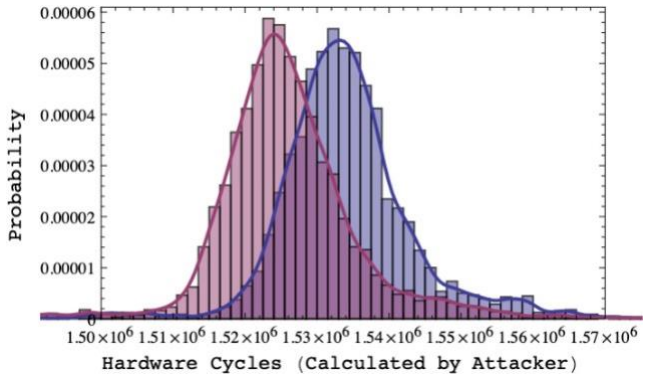

Fig. 17. Attacker calculation of cycles[30]

To collect the whole block of plaintext which is TLS encrypted we need 223 TLS sessions. An attacker needs 2 13 TLS sessions to recover single plaintext byte. If attacker can collect the amount of information mentioned above the target is closer to the attacker and they can get the plain text. Attacker can generate the sessions via the malware inserted in user system.

1) :

a) :

: Mitigation for Lucky13:: Patches is one of the solution to mitigate this kind of attack but there are many solutions present to defend from this lucky13 attack. Below are few defenses[30].

Addition of time delays:

We can say adding random time delays so that it become complex for CBC mode decryption to the attacker but attacker can overcome this by adding more sessions but it can be used in some cases.

Using of RC4 Cipher Suites:

Instead of using a block cipher we can use RC4 stream Cipher to avoid the usage of padding of plaintext, but this cannot be applied under DTLS which is datagram transport layer security. By shifting to RC4 cipher there are less possibilities for attack as RC4 cipher suites are mostly used in TLS. There are certain Cryptographic weakness in RC4 which makes attacker can be take control of the system remotely and we can consider this as a temporary fix.

Encryption which is authenticated:

To eliminate this kind of attacks some encryption algorithms such as AES- GCM should be used which is a block cipher that provides data integrity and speedy authenticated encryption.

Implementation of MEE- TLS- CBC decryption :

Another mitigation step can be done by modifying the CBC – mode decryption process. The goal is to eliminate the clock side-channel by providing a consistent processing time for all ciphertexts in this mode and the working time must be proportional to the amount of the ciphertext rather than the plaintext. This can be done by assuring that the MAC processing amount remains same and independent of the message duration indicated by the underlying plaintext.

The above method is complex sometimes and consumes lot of time. As a result, the best long-term mitigation approach for preventing SSL LUCKY 13 exploits is to avoid using TLS in CBC mode and to use AEAD cipher suites.

C. Triple Handshake attack

It was discovered in 2009 that the TLS renegotiation process was vulnerable. The protocols were corrected by developing a new technique for secure renegotiation, but the correction was not entirely effective as a new attack called the Triple Handshake Attack was discovered in 2014.

Prerequisites:

The weaponization of Triple Handshake Attack is quite complex and requires very specific conditions to be satisfied. However, there are two primary requirements which when satisfied leads to successful exploitation.

- The attack depends on the site's ability to use client certification.
- The client must be tricked into using their client certification on sites where they are normally not used. This can be achieved through sophisticated phishing techniques.

Since only a handful sites on the internet use client certificates, this exploit has not received widespread usage, unlike the original insecure renegotiation problem.

Step 1: Unknown key share weakness

The attack requires the adversary to host a malicious server as the master secret is generated using pre-master secret and random value selected by the client (victim). The attacker tricks the victim into visiting the website hosted by the attacker's malicious server. The client (victim) sends the pre- master secret and random value to this malicious server, this is then forwarded to a legitimate server. The attacker obtains the responses from the legitimate server and these responses from that legitimate server are sent to the victim. In this attack, there are 3 parties and 2 TLS connections as shpown in Fig 18. This weakness observed is called the Unknown key share weakness. The attack in its current step is not very dangerous but with phishing (a malicious server pretending to be the legitimate server), the attacker gains the capability to do real damage.

Step 2: Full Synchronization

Every connection between two principals will have a different verify_data value as every server will have their own certificate. In this step, the attacker takes advantage of "the session resumption mechanism and its abbreviated handshake". During a session resumption, there are no steps to ensure authentication as the protocol assumes that the master key is sufficient to authenticate two principals. Therefore, when the session resumes, the elements that were different (certificates) during the first connection are not considered. Hence, when the handshake is completed both SSL connections will have the same Finished message as show in Fig 19.

Step 3: Impersonation

The attacker now impersonates the client by using the client's certificate. This is achieved by forcing the client into

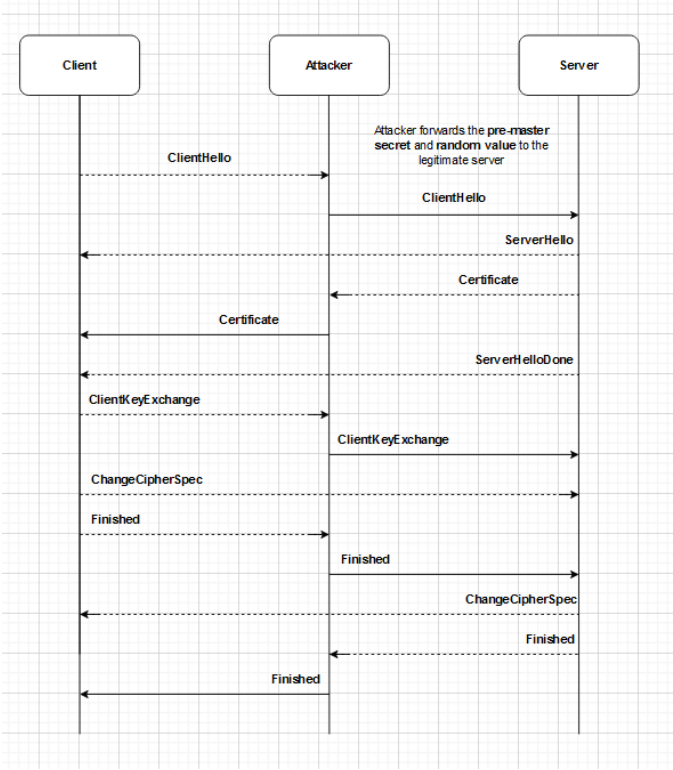

Fig. 18. 2 Attacker communicates both with the client and the legitimate server [1].

renegotiation. This renegotiation occurs when the attacker requests a resource on the legitimate server that requires authentication, the legitimate server requests a renegotiation which is forwarded to the client by the attacker. The attacker now authenticates with the server using the client's certificate. Since the connection parameters are the same on both sides (client and legitimate server), the attacker can just mirror the protocol messages. The attacker now has full control over both the connections and can now send arbitrary application data on either side. One disadvantage of the attack is that the attacker loses traffic visibility after the renegotiation. The attacker still stays in the middle and continues to mirror the encrypted traffic between the two principals.

Weaponization:

To weaponize this attack, the attacker first has to find suitable entry points in an application and design payloads for each of these. After renegotiation, the attacker loses all traffic visibility and hence will not be able to verify certain interactions between the client and the legitimate server. The attack can however send arbitrary data to both the client and legitimate server before renegotiation. The attacker achieves this by placing JavaScript malware on their site. After renego- tiation and authentication, the attacker's malware has unlimited capability of issues HTTP requests in the background. The success of the attack also depends on using sophisticated phishing attacks to lure the client into visiting the attacker's site.

1) :

a) :

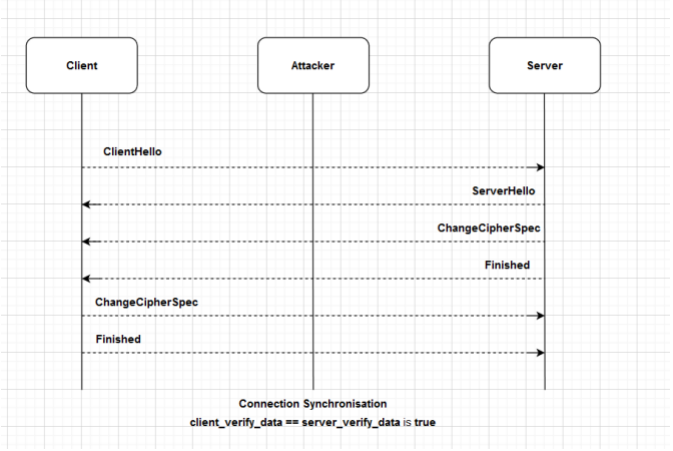

Fig. 19. 3 Connection Synchronization [1]

: Mitigation:: Triple Handshake primarily takes advantage of how the renegotiation is designed in the TLS protocol. Browser vendors quickly reacted by modifying their browser source code to abort connections when the browser sees a different certificate after renegotiation and degenerate DH public keys are no longer accepted. Older browsers like Internet Explorer are relatively safe as Microsoft patches system libraries rather than core browser codebase.

Following are the recommendation by security researchers to protect end users from this attack [1]:

- Require client certificates for all access

If this policy is implemented, then the attacker would need their own certificate to carry out the first step of the attack as client certificate is required for all TLS connections to a site.

- Disable renegotiation

The whole attack depends on renegotiation. This is used in combination with client certificates. If renegotiation is disabled, then the whole attack vector can be avoided.

- Enable only ECDHE suites

The attack is not vulnerable to ECDHE suites. Almost all modern browsers support ECHDE suites, even modern smartphones (Android and iOS). If the browsers enforce the usage of ECHDE suites, then the whole attack vector can be avoided as only DHE and RSA are vulnerable.

D. POODLE

Padding Oracle on Downgraded Legacy Encryption. The SSL 3.0 protocol's weakness is exploited by the Poodle attack. This flaw enables an attacker to intercept SSLv3-encrypted traffic. The Transport Layer Security proto- col (TLS), the replacement for SSL, no longer has the flaw (Secure Socket Layer). According to the most current Acunetix 2020 Web Application Vulnerability Assessment, up to 3.9% of web servers are still POODLE susceptible, meaning they continue to support the SSL 3.0 protocol even though the TLS standard was established in 1999. Given that they support TLS 1.0, even more servers (more than 30%) are susceptible to the BEAST attack. [8].

Simple cryptographic protocols like SSL and TLS can enable you safely validate and transfer data over the internet.

The SSL and TLS protocols, for instance, can aid to secure your payment processing if you're using your credit card to make purchases on a website. This will prevent fraudsters from obtaining your credit card information. [10]

What Can an Attacker Do with POODLE?

The POODLE security flaw enables a man-in-the-middle attacker to eavesdrop on supposedly secure communications. This means that attackers can utilise POODLE to steal users' confidential information and possibly pass themselves off as the user, robbing them of control over the vulnerable web application. The following three stages must be completed successfully for a POODLE attack to be effective:

1. A successful MiTM attack allows an attacker to access victims' sensitive data at the initial stage. In a MiTM attack, an attacker sneakily places themself between two parties who are communicating over the internet in order to intercept and relay communications to and from them. This is a type of active eavesdropping. Hence, the attacker is actually in charge of the entire communication even though the victims think they are speaking to each other directly over a private connection.
2. The second part of the attack involves forcing the server to use the SSL 3.0 protocol. If it is not possible to migrate to a newer protocol like TLS 1.2, they accomplish this by repeatedly dropping connections throughout the MiTM attack until the service switches to an earlier protocol. The downgrade assault is the name of this POODLE attack stage.
3. When the server changes to SSL 3.0, the attacker utilises POODLE to decrypt packets and extract data. This entails that they can access the information in plaintext in unencrypted data and intercept the session of the susceptible client. [9]

How to prevent and repair POODLE attacks?

A POODLE attack can be conducted against any server that accepts SSL 3.0 and earlier versions of TLS. Current TLS versions are secure, and current browsers prevent websites using outdated TLS versions (1.0, 1.1). A server that is set up to only support more recent protocols (TLS 1.2, 1.3) guards against POODLE attacks. [9]

How Does POODLE Work?

Many aspects of the SSL/TLS protocol make the POODLE attack conceivable. For now, all we need to know is that SSL/TLS enables the server and the browser to use cipher suites, which are collections of several methods that can be used to encrypt communication. The POODLE vulnerability affects cipher suites that combine block ciphers with symmet- ric encryption, including the AES or DES algorithms. In these situations, asymmetric encryption is used to create a secret key once the client and server have agreed upon it (a private key and a public key). Then, using this key, all communication will be symmetrically encrypted. Moreover, POODLE-vulnerable cipher suites employ cipher-block chaining (CBC mode). This indicates that because the logical operation XOR is used to generate values, each block's value is determined by the value of the block before it. Also, a random data block known as an initialization vector is added at the beginning. This is essential so that the appearance of encrypted data changes each time it is used (and therefore the attacker cannot figure out the data based on similarities). [8]

What Is Padding?

All of the data in a block cipher must be multiples of the block size. For instance, data must be 64, 80, or 336 bytes if the block size is 8. (a multiple of 8). It must be padded with unnecessary information if it is not a multiple of 8 in order to be the appropriate length. The padding method used by the majority of web servers is:

- The padding length must always be contained in the last byte of the last block. The amount of padding in the preceding bytes is represented by that value. For instance, the value of the block is xx-xx-xx-yy-yy-yy-yy-04, here 4 bytes contain padding (yy represents padding). This is a prerequisite for SSL.
- The values of all padding bytes are often the same as the length value in implementations. For instance, the value of the block would be xx-xx-xx-04-04-04-04 if 4 out of 8 bytes were padded.
- There must be an additional block added with merely padding if the length of the data is greater than the block size, for instance 336 in this case: 07-07-07-07-07-07-07. If the last byte did not indicate padding, the algorithm would not be able to distinguish padding from actual data.

In other words, as long as the last byte is between 00 and 07, the block will be allowed because SSL does not verify padding bytes (aside from the padding length). For instance, a block with the numbers xx-xx-xx-12-34-56-78-04 is acceptable. [8]

What Is a Padding Oracle?

The padding oracle is a circumstance when the attacker knows or may infer the reason why the data they submitted to the server was refused, such as whether the padding or MAC is incorrect. Imagine the following situation:

- The attacker receives information from the browser and is aware that it includes a password. The attacker is aware that this is an HTTP POST request and is also aware of the precise location of the password within this request.
- The attacker modifies the encrypted data and sends it to the server.
- The server informs the attacker that the data is inaccurate in response. It can, however, respond with one of two possible forms of errors: either it can inform the attacker that the padding or MAC was incorrect. This enables the POODLE attack. [8]

How does CBC work?

It separates the data into blocks of a certain length after taking the complete set of data. Each block is encrypted before being used to encrypt the following block. With the use of an XOR operation, the newly encrypted value of a block is added to, or simply used for, the encryption of the following block. The final block is encrypted using this technique. The process of decryption is carried out in reverse order, starting with the last block and continuing until the first block is decrypted. The padding data is eliminated when the decryption procedure is finished, and the MAC value is checked using the data that came before it. If the MAC value cannot be verified, the sender terminates the connection under the assumption that malicious behaviour has taken place in the communication network. First, the attacker successfully performs a man-in- the-middle (MITM) attack to eavesdrop on the client-server connection. The server is then compelled to switch from TLS

to SSLv3 by the attacker. If that effort is unsuccessful, the attacker forces the server to use an earlier TLS version, such as TLS 1.1 or TLS 1.2. This is known as a Protocol downgrade attack. As the attacker had been watching the conversation for a while, it was easy for him to identify the HTTP headers and the encrypted contents because of their predictable nature. An attacker will find it simpler to spoof the session cookies as a result.

If the padding is the perfect length, SSLv3 does not verify the padding's integrity. This gives the attacker the opportunity to guess the session cookie's last byte by replacing the padding with the session cookie block and sending it to the server. The attacker takes use of the padding validation algorithm's weakness after lowering the protocol. The attacker regularly queries the server to determine whether the biscuit block's final byte is accurate. If the block is correct, the receiver accepts it. The block will be declined in every other case. The last byte of the block will be changed and transmitted again if the server denies it. [11]

Impact Of Poodle Attack:

On the network, sensitive information can be transmitted and taken by a intruder. They can exploit this information to access database content by pretending to be an actual user. [11]

How to Prevent POODLE Attack?

Configure the server to support only TLS 1.2 and above. Disable all SSLv2 and SSLv3. [12]

The TLS_FALLBACK_SCSV cypher should be used by the client. This informs the server that even if the client is downgrading to a lower SSL version, it is still capable of using a higher version. This alerts the server that there may be a man in the middle tampering with the connection, and the connection is terminated as a result. [11]

Implementation :

1) We identified few websites that still runs with SSLV3, and tried to implement POODLE attack on one of the servers that are vulnerable.
2) Now we are trying to check the vulnerabilities that this following website has

Website- portal.westermann.de

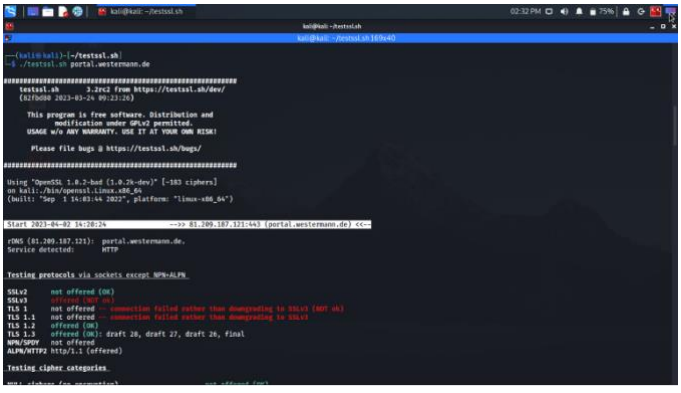

Fig. 20. 3. Now we are trying to check the vulnerabilities that this following website has Website- portal.westermann.de

As we can see it is vulnerable to POODLE attack as it offers SSLV3

Now we are trying to exploit with the following command nmap –script ssl-enum-ciphers -p 443 example.com

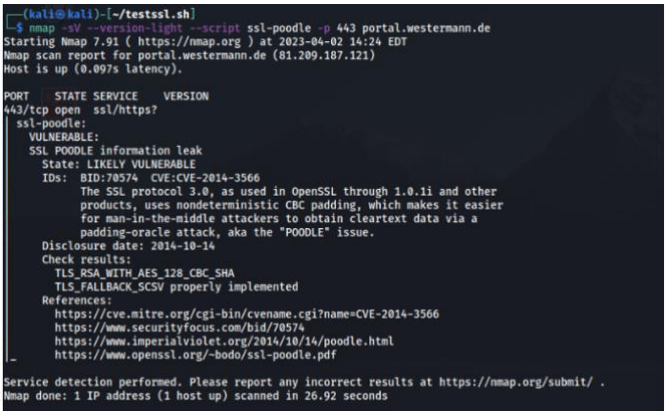

Fig. 21. 4. We can check the vulnerability by using the following command nmap -sV –version-light –script ssl-poodle -p 443 example.com

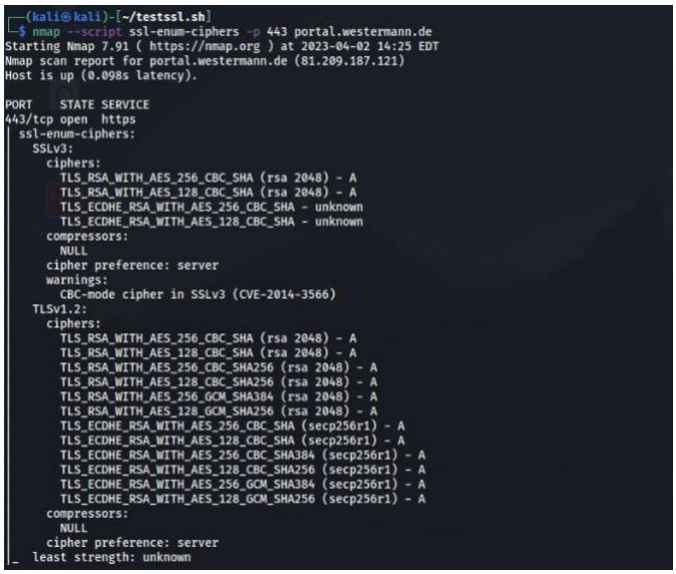

Fig. 22. 5. Now we are trying to exploit with the following command nmap

–script ssl-enum-ciphers -p 443 example.com

E. DROWN ATTACK:

Drown is a TLS to SSLv2 cross-protocol attack that gives an attacker access to all client and server communication by decrypting the pre-master secret of a TLS handshake. Any server using SSLv2 and TLS 1.2 is under a major and real threat from it.

Drown attack, which stands for "Decrypting RSA with Obsolete and Weakened encryption ", was created by academics at universities all around the world in conjunction with the OpenSSL Project. It was initially reported to OpenSSL on December 29th, 2015.The HTTPS servers used by DROWN, which accounted for 16% of all HTTPS servers, still allow SSLv2 connections. A further 17% of HTTPS servers, for a total of 33% of servers vulnerable to the attack in March 2016,

were also prone to it when the private key is used on a separate server that accepts SSLv2 connections. [1]

Attack Process:

The DROWN attack works because SSLv2 employs 40-bit export cyphers, is vulnerable to a Bleichenbacher attack, and shares a certificate and private key with TLS. By using the ciphertext and exploiting SSLv2 vulnerabilities, an attacker is now able to decrypt RSA key exchange messages sent across secure TLS connections.

One of the final transmissions in the TLS handshake is the key exchange. The handshake starts with a ClientHello, to which the server replies with a ServerHello along with a certificate containing the public key. The client encrypts a pre-master secret concatenated with the public key using the public key, which will be used to build the session key (which serves to ensure the authenticity of the encryption). The client declares they are finished, the server declares it has all it requires, and both the client and the server are now free to transfer any type of information with the security of encryption. For DROWN, the true problem is not with this handshake. Nevertheless, there is a weakness in the handshake of SSLv2. The process begins with a ClientHello, is followed by a ServerHello, the MasterKey is then sent, and the MasterKey is then verified by the server. This makes it simple to determine whether or not your attack was successful and enables the attacker to determine the effectiveness of their own strike. [1]

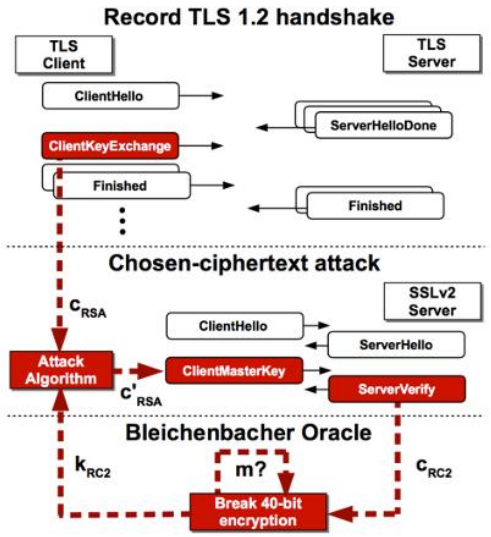

Fig. 23. DROWN attack implementation [1]

The first phase of DROWN, known as the Bleichenbacher step, is releasing thousands of different multiples of the ciphertext to see what sticks. The attacker learns the plaintext if the Bleichenbacher oracle replies that it might be decrypted. The assailant keeps doing this until they can crack the pre-master secret's encryption. This works because RSA keys are used by default in SSLv2 and TLS.

SSLv2 might be the key to this exploit. It replies with an obfuscated version of the pre-master secret if the attacker's ciphertext can be broken. It replies with a new pre-master secret that has been obscured if it isn't. The attacker will have an oracle if they send the identical Bleichenbacher ciphertext again and check to see if the pre-master secrets they receive back are different or the same, indicating that the ciphertext they supplied may be broken. The pre-master secrets are obscured by SSLv2 using a key with just forty bits of secrecy, making it possible for an attacker to tell whether they are the same or different. Forty bits are simple to brute force. [2]

As SSLv2 and TLS share the same RSA private key, the attacker can obtain the pre-master secret for TLS on the same server by decrypting the pre-master secret in the SSLv2 implementation. While this is an expensive process, Bleichenbacher's attack is known as the "million-message attack" for a reason since it involves numerous requests to the server. Any server running TLS 1.2 and SSLv2 is susceptible.

DROWN Prevention and Future Steps:

Disabling SSLv2 on all web servers is a simple way to prevent the DROWN attack. Unfortunately, the user cannot defend themselves against the attack because DROWN is a server-side vulnerability. The user's communications are susceptible as long as the web server they are accessing supports SSLv2 connections. It is crucial to remember that even if a web server disallows SSLv2, it is still susceptible if it shares a private key with a server that supports SSLv2. [1]

The DROWN attack is shielded from using the newest TLS standard (TLS 1.3). The DROWN attack is now blocked by the recently developed protocol, which is accessible. Prior to TLS 1.3, several attacks, including DROWN, FREAK, and POODLE, occurred. Every element of previous protocols was examined for this version to determine its need and security.

Static RSA key exchanges, which enable Bleichenbacher-style attacks, are not supported by TLS 1.3 in contrast to its forerunners. The usage of an outdated certificate from TLS 1.2 by the server, however, voids TLS 1.3's assurance of protection against DROWN. As a result, server administrators shouldn't combine TLS 1.3 with outdated certificates. Most browsers do not yet support TLS 1.3, even though it is not susceptible to DROWN. [2]

F. CRIME ATTACK

The Secure Sockets Layer (SSL)/Transport Layer Security (TLS) and SPDY protocols encoding is flawed, and this is how the CRIME attack takes advantage of it. The acronym means Compression Ratio Info-leak Made Simple. This sort of risk is an assault on secret web cookies transmitted via compressed HTTPS for SSL/TLS protocols or SPDY, Google's HTTP-like protocol. The exploit may expose cookie data to session theft.

TLS 1.0 apps that use TLS compression, Google's SPDY protocol, older versions of Mozilla Firefox that support SPDY, and older versions of Google Chrome that support TLS and SPDY were recognised as vulnerable by security experts. In 2012, approximately 42% of servers enabled the optional feature of SSL compression, possibly affecting many famous websites. Only 0.8% of servers enabled the SPDY that was expressly embedded. Approximately 7% of computers enabled compression. [2]

Adam Langley proposed the CRIME attack, which was first demonstrated by security experts Juliano Rizzo and Thai

Duong, who also developed the BEAST exploit.The complete scope of the exploit was to be disclosed at the 2012 ekoparty security symposium. Rizzo and Duong described CRIME as a universal attack that can be used against a wide range of protocols, which is not restricted to SPDY, TLS and HTTP. [1]

The proof of concept for the CRIME attack is described in this report using RC4 and CBC ciphers.

Attack Process:

In the CRIME attack, adversaries can learn about the ciphertext amount sent by the client browser. They can observe how the compressed request payload — the browser's private cookie and the inserted malicious content — changes in size. When the compressed content shrinks in size, the injected content probably matched some of the confidential content they want to access. The value of the user's session cookie can possibly be found by observing the change in length.

CRIME Attack using RC4 stream cipher:

The attacker sends multiple requests to the server and checks the length of the encrypted data.

We first submit a request with the character we want to discover, followed by numerous characters that were not detected in the original request, such as some special characters: chr(x) + "*:/[$/&]. Then we submit a second request, but this time we reverse the payload: "*:/[$/&" + chr(x) and compare the two lengths. If len(enc(request1)) = len(enc(request2)), we have discovered a character. This method is known as 'two_tries'.

In this demonstration, we selected the secret cookie=quokkalight and if we run the algorithm two_tries we will find the following result:

Result 1: cookie=quokie=quokie=quokie=quokie=quokie=

Result 2:

cookie=quokkalight

The algorithm needs to take all the paths in the tree to find all the possible solutions. We can see all the results as a tree that can be represented like this:

CRIME Attack using CBC cipher:

The CBC cipher method divides everything into blocks when using AES or DES. For example, if we use AES with CBC cipher mode, the block will be divided into 16 segments, with padding inserted at the end if len(data)%16! = 0. It is important to observe that in the CBC mode, the payload=rand will create a length of 16 rather than 12 because padding will be added at the end of the data before encryption. Given that the attacker controls the GET and POST data, the attacker will set the padding length to 1 by putting an arbitrary value into the GARB variable.

Next, we will use the same method explained in the RC4 part, but we will add a step before the main call

We call the function adjust_padding(so we can have a padding of length 1

We call the function two_tries_recursive(and we can find the secret FLAG

CRIME Prevention and Future Steps:

CRIME can be prevented by stopping the use of compression, either on the client end, by disabling compression of SPDY requests in the browser, or by the website prohibiting the use of data compression on these kinds of interactions using the TLS protocol negotiation features. The Transport

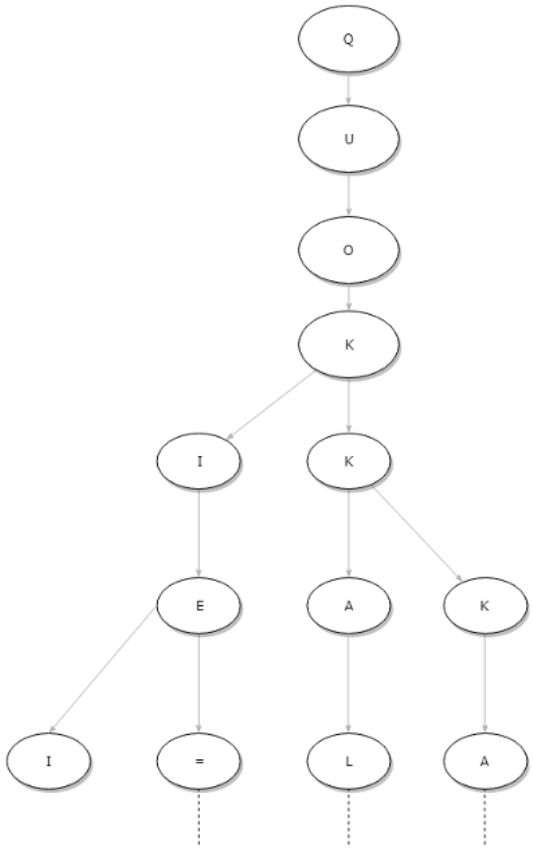

Fig. 24. Tree Representation [1]

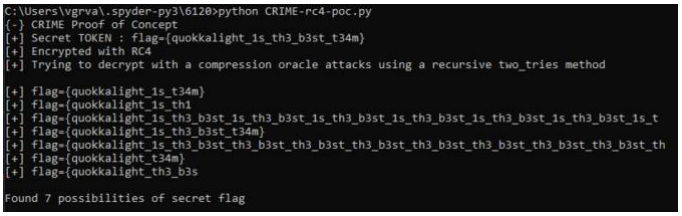

Fig. 25. CRIME Attack implemented using RC4 [1]

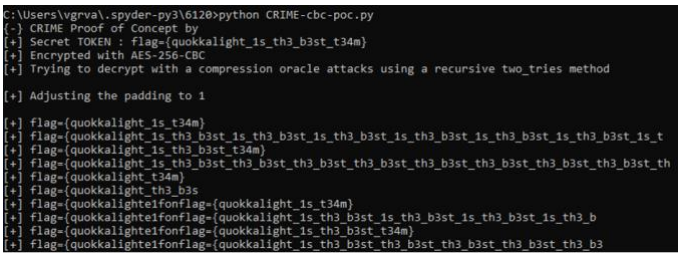

Fig. 26.

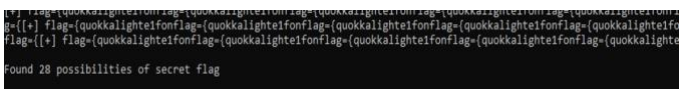

Fig. 27. CRIME Attack implemented using CBC [1]

Layer Security (TLS) Protocol Version 1.2 [7] requires the client to transmit a list of algorithms for compression in its 'ClientHello' message, and the server selects one of them and sends it back in its 'ServerHello' message. Because the server can only select one of the compression methods provided by the client if the client only provides 'none' (no compression), the information will not be compressed. Similarly, because all TLS clients must enable 'no compression,' a server may always decline to use compression.

The CRIME attack against SPDY and TLS-level compression was characterized as fixed in the most recent versions of the Chrome and Firefox web browsers as of September 2012. Some companies have implemented defences. Since 1.0.9/1.1.6 (October/November 2011) using OpenSSL 1.0.0+, and since 1.2.2/1.3.2 (June/July 2012) using all versions of OpenSSL, the nginx web server has not been susceptible to CRIME.

VI. Overview of existing libraries

The table 1 shows an overview of existing SSL/TLS libraries along with version of SSL/TLS they support.

A. OpenSSL

A software library called OpenSSL is used by programs that need to identify the party on the other end of a communication or offer secure communications across computer networks against eavesdropping. The vast majority of HTTPS websites utilize it, as do many Internet servers.

Both the SSL and TLS protocols are open-source and included in OpenSSL. The C programming language's core library supports fundamental cryptography operations and offers a number of utility operations. A number of programming languages provide wrappers that make it possible to utilize the OpenSSL library.

B. WolfSSL

The embedded systems developers are the primary audience for the compact, portable, and embedded SSL/TLS library known as wolfSSL. It is a C programming language-based open source implementation of TLS (SSL 3.0, TLS 1.0, 1.1, 1.2, and 1.3), which also includes DTLS 1.0, 1.2, and 1.3. Support for several APIs, including those outlined by SSL and TLS, is provided, along with client libraries and a server implementation for SSL/TLS. A compatibility interface for OpenSSL with the most popular OpenSSL functions is also included in wolfSSL.

C. GnuTLS

A free software programme that implements the TLS, SSL, and DTLS protocols is called gnuTLS. It provides access to X.509, PKCS #12, OpenPGP, and other structures through interfaces provided by its application programming interface (API), which enables secure communication over network transport layers. is strongly recommended that these files be saved in PDF Sizing of Graphics

Overview of existing SSL/TLS libraries

D. BoringSSL

In order to serve Google's requirements, BoringSSL was forked off of OpenSSL.

Even though BoringSSL is an open-source project, it is not meant for widespread use. Applications that use BoringSSL ship their own versions of it, and if we decide to make API changes, we update everything as necessary. Under the sake of compatibility, sacrifices can be avoided thanks to this. We find it to be effective, but you might not. Because Google used OpenSSL extensively for many years in a variety of methods, it amassed a sizable number of patches over time that were kept up to date while following upstream OpenSSL. This led to the creation of BoringSSL.

E. MatrixSSL

An open-source TLS/SSL implementation called MatrixSSL is made specifically for specialized software in embedded hardware settings. Full cryptographic software is included in the MatrixSSL library, along with symmetric and public key methods that are accepted in the market. The Inside Secure TLS Toolkit is the name given to it now.

S.No	Library	SSL 2.0	SSL 3.0	TLS 1.0	TLS 1.1	TLS 1.2	TLS 1.3	DTLS 1.1	DTLS 1.2
1	OpenSSL	No	Disabled	Yes	Yes	Yes	Yes	Yes	Yes
2	WolfSSL	No	Disabled	Disabled	Yes	Yes	Yes	Yes	Yes
3	GnuTLS	No	Disabled	Yes	Yes	Yes	Yes	Yes	Yes
4	BoringSSL	-	-	Yes	Yes	Yes	Yes	Yes	Yes
5	MatrixSSL	No	Disabled at compilation	Yes	Yes	Yes	Yes	Yes	Yes

Table 1. Comparision between the libraries

VII. Conclusion

In the past several years, numerous SSL/TLS attacks have been found. These attacks focus on weaknesses in the design or execution of protocols. The most significant of these attacks and their defenses were studied for this essay. Some attacks are just marginally practicable, but with certain improvements, they might soon become realistic.

Because TLS 1.3 does not use less secure cryptographic primitives, it offers the best protection against a large number of attacks. There are effective defenses against various protocol attacks, like Lucky 13, BEAST, Triple Handshake and CRIME.

The best practices for SSL/TLS deployment must be known by the system administrator, and all servers and clients within the company must be set up and updated on a regular basis in accordance with these principles.

References

[1] I. Ristić, . . Bulletproof, Tls-Understanding, S. Deploying, P. Tls, and To, *Secure Servers and Web Applications*, pp. 230–236, 2014.

[2] https://en.wikipedia.org/wiki/Heartbleed

[3] https://cipcyber.com/2020/02/how-to-exploit- heartbleed-using-metasploit-in-kali-linux.html

[4] https://www.csoonline.com/article/3223203/the- heartbleed-bug-how-a-flaw-in-openssl-caused-a-security-crisis.html

[5] https://www.synopsys. com/blogs/software-security/heartbleed-bug/\#:\textasciitilde:text=OpenSSL\%20processes\%20in\%20the\%20machine

[6] https://www.malwarebytes.com/blog/news/2014/02/ apple-in-hot-water-after-ssltls-validation-fiasco
[7] https://gitlab.com/gnutls/gnutls/-/issues/694
[8] https://www.acunetix.com/blog/web-security-zone/ what-is-poodle-attack/
[9] https://www.techtarget.com/whatis/definition/ POODLE-attack
[10] https://www.makeuseof.com/what-is-the-poodle- attack/
[11] https://beaglesecurity.com/blog/vulnerability/ poodle-attack.html
[12] https://medium.com/@c0D3M/poodle-attack- explained-ed6a1cd0667d
[13] https://www.acunetix.com/blog/web-security-zone/ what-is-beast-attack/
[14] https://crashtest-security.com/ssl-beast-attack-tls/
[15] https://www.invicti.com/blog/web-security/how- the-beast-attack-works/
[16] https://cwatch.comodo.com/website-security/beast- attack.php
[17] https://medium.com/@c0D3M/beast-attack- explained-f272acd7996ePOODLE-attack
[18] https://crashtest-security.com/prevent-ssl-lucky13/
[19] https://www.youtube.com/watch?v=-_8-2pDFvmg
[20] fromhttps://drownattack.com/drown-attack- paper.pdf