A Comprehensive Survey on SSL/ TLS and their Vulnerabilities

Ashutosh Satapathy
Research Associate
School of Computing
Science and Engineering
VIT University-Chennai Campus
Chennai, India

Jenila Livingston L. M.
Associate Professor
School of Computing
Science and Engineering
VIT University-Chennai Campus
Chennai, India

ABSTRACT

The boom of internet, web technologies bring the whole world under a single roof. Transferring information through e-ways leads security to be an important aspect to deal with. In IP network, SSL/ TLS is the protocol works on the top of the transport layer to secure application traffic and provides end to end secure communication. A security hole in those protocols makes the communication channel vulnerable to be eavesdropped and modified information later. This paper discuses SSL and TLS architectures and presents survey on attacks against SSL/TLS. It also highlights the factors influence on those attacks.

Keywords

Secure Socket Layer, Transport Layer Security, Compression Algorithms, Message Authentication Code, Cipher Block Chaining, SSL/ TLS Attacks

1. INTRODUCTION

Today in business world, World Wide Web (WWW) is the role model behind every action. As the demand increases, it requires transformation from web services to secure web services. Secure Socket Layer (SSL)/ Transport Layer Security (TLS) protocols are used to provide reliable services over transport layer protocol [1]. SSL has gone through several upgradation such as SSLv1.0, SSLv2.0, and SSLv3.0 etc. SSLv3.1 is basically called as TLSv1.0 which provides backward compatible with previous version of SSL.SSL protocol works on two layer of services where first one is SSL connection and second one is SSL session. SSL connection works at transport layer to establish links between clients and servers. Peer to peer associations allow sessions to be built up which is ephemeral. Each SSL session is associated with one SSL connection. SSL/ TLS handshaking protocol is used to create session by exchanging a couple of parameters (e.g., random number, session ID, cipher suite, compression techniques etc.). Each session is maintained by two states mainly. Session state deals with a number of parameters such as session identifier, X509 certificate, compression techniques, cipher specification, master secret etc. connection state parameter includes server and client send MAC secrets, initialization vectors, sequence numbers etc.

2. ARCHITECTURE

2.1 SSL Architecture

SSL is the mixture of four protocols which provide security to upper layer protocols such as HTTP, FTP and any application layer protocol. They are distributed into two layers (see Figure 1).

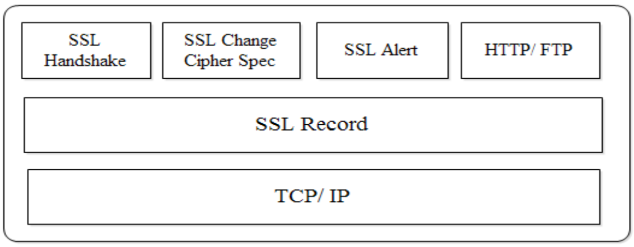

Fig 1: SSL protocol layer structure

2.1.1 SSL Record Protocol

It works as the base for other three protocols and provides confidentiality and integrity to upper layer messages. At sender site, it segments the information into a number of chunks, compresses those, compute MAC and encrypt the chunks with corresponding MAC together. At receiver site, those processes are accomplished in opposite direction before the original messages delivered to receiver. By default compression is disabled in SSLv3.0 and all versions of TLS. Flow structure of SSL packets creation are divided into five parts (see Figure 2).

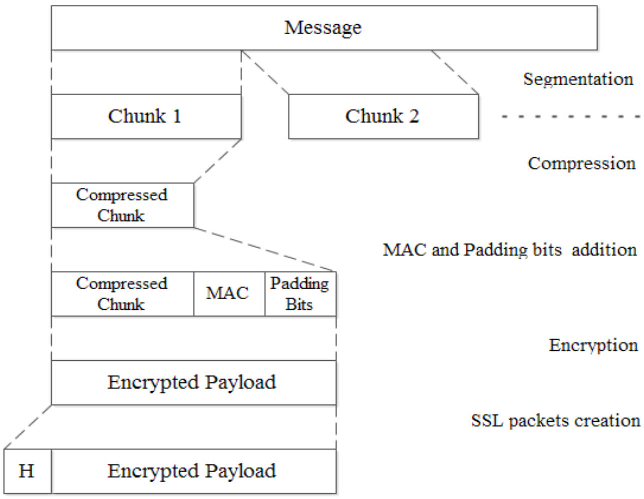

Fig 2: SSL record protocol working principle

i. *Compression Algorithms:* Lossless compression techniques are called by the SSL record protocol to squeeze the data without any loss. (e.g., Huffman codding, LZ77, GZIP etc.)

ii. *Hash Algorithms:* Secure hash functions are played major roles to provide confidentiality to each segment of data. Most popular MD5 and SHA algorithms are used to compute MAC. (e.g., MD5, SHA-1, SHA-224, SHA-256 etc.)

iii. *Encryption Algorithms:* Symmetric stream or block cipher techniques are used to create SSL payload. In case of stream cipher encryption, compressed chunk and MAC are encrypted together. Padding bits are added along MAC to chunk before block cipher encryption. Symmetric algorithms with their key sizes are listed in table 1 and table 2.

Table 1. Stream encryption algorithms and key sizes

Algorithms	Key Sizes (bits)
RC4	40 or 128

Table 2. Block encryption algorithms and key sizes

Algorithms	Key Sizes (bits)
RC2	40
DES	40 or 56
Fortezza	80
IDEA	128
3DES	168
AES	128 or 256

As described above, hash function with a shared secret key is used to calculate Hashed based Message Authentication Code (HMAC) where '+' stands for concatenation operation. Its evaluation is given below.

HASH (MAC_secret_key + pad_2 +

HASH (MAC_secret_key + pad_1 + seq_no + compression_type + compressed_chunk_length + compressed_chunk) (1)

Padding bits (pad_1 and pad_2) length for MD5 and SHA-1 are 384 bits and 320 bits respectively. Segmentation allows maximum size of chunk is 2^{14} bytes. If compression is applied, the length of chunk after compression is not more than 1024 bytes. So, it permits the maximum size of MAC is 1024 bytes. 2^{14} + 2048 bytes is the maximum length of SSL payload which forces encryption algorithm to restrict the incremental length not more than 1024 bytes. At last SSL header is appended to SSL payload before packets send to lower layer. SSL record protocol header consists of four major fields such as Content Type, Major Version, Minor Version and Compressed length. Content Type defines *handshake*, *change_cipher_spec*, *application_data* and *alert* which supply information to upper layer protocol to process the chunks/ segments at receiver side. Major and Minor Version specifies used major and minor version of SSL in use. Compressed length indicate size of SSL payload in bytes.

2.1.2 SSL Change Cipher Spec Protocol

It is one of the simplest protocol utilizes SSL record protocol and deals with single byte. Byte with value 1 indicates current state is updated with remaining state causes new cipher suite to be activated for current link. Normally new SSL handshaking is followed by change cipher spec message.

2.1.3 SSL Alert Protocol

It propagates faults to peer devices happens during SSL negotiation and connection. It deals with two bytes which are compressed and encrypted alike other messages. First byte indicates level of alert and carries two values such as '1' stands for warning or '2' stands for fatal. If the alert message is fatal, link must be aborted and no new link can be established on that particular session by SSL. Second byte indicates the degree of severity specified by code related to different alert messages. Alert messages with corresponding codes and types are listed in table 3 [2].

Table 3. Alert messages of SSL

Codes	Alerts	Representations	Types
0	close_notify	No more messages on this link to receiver	Warning
10	unexpected_message	Inappropriate message to receiver	Fatal
20	bad_record_mac	Incorrect MAC record to receiver	Fatal
21	decryption_failed	Invalid decryption due to improper chunk size	Fatal
30	decompression_failure	Decompression fail due to improper input	Fatal
40	handshake_failure	Negotiation fail due to improper security parameters set	Fatal
41	no_certificate	Reply to no proper certificate is available	Warning
42	bad_certificate	Corrupted certificate or contains invalid signature	Warning
43	unsupported_certificate	Sender certificate is unsupported	Warning
44	certificate_revoked	Certificate was withdrawn by signer	Warning
45	certificate_expired	Issued certificate is no longer valid	Warning
46	certificate_unknown	An uncertain problem causes certificate to be inappropriate while handling	Warning
47	illegal_parameter	Security parameter are inconsistent w.r.t. their field in handshake	Fatal

2.1.4 SSL Handshake Protocol

It is the first protocol come to action after the connection is established by transport layer protocol. Client and server validate each other and exchange necessary security parameters such as cipher suite, compression techniques, random number etc. before sending application data to each other (see Figure 3). Handshake protocol packet consists of three fields. 'Type' deals with 1 byte represents type of the packet, 'Length' of 3 bytes indicates length of the packet and 'Content' (≥ 0 bytes) carries necessary security parameters to be set during negotiation. Handshake messages with corresponding codes and security parameters are listed in table 4 and table 5.

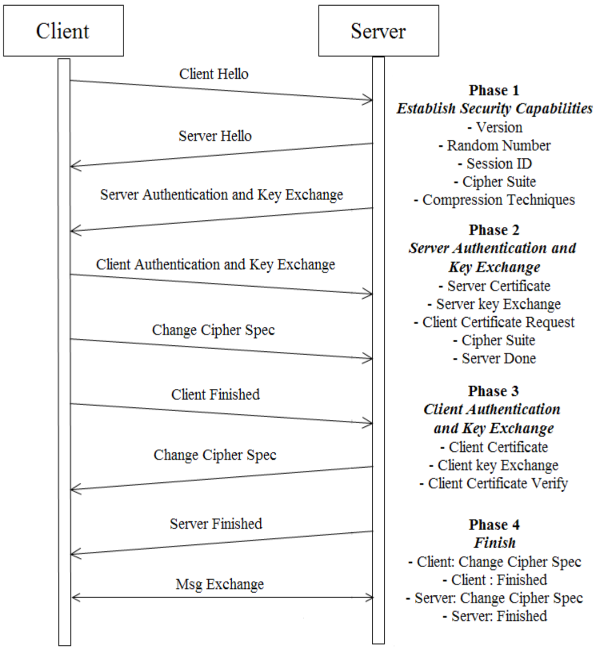

Fig 3: SSL/TLS handshaking protocol operation

Table 4. Handshake messages of SSL

Codes	Messages	Parameters
0	MT_hello_request	Void
1	MT_client_hello	version,random_no, session_id, cipher_suite, compression_tech
2	MT_sever_hello	version,random_no, session_id, cipher_suite, compression_tech
11	MT_certificate	X.509 certificates chain
12	MT_server_key_exchange	msg_signature, public_parameters
13	MT_certificate_request	cert_authorities, cert_type
14	MT_server_done	Void
	MT_client_key_exchange	msg_signature, public_parameters
15	MT_certificate_verify	cert_signature
20	MT_finished	MD5_hash + SHA_hash

Table 5. SSL cipher suite

Parameters	Values
Key exchange algorithms	RSA, Diffie-Hellman, Fortezza
Cipher algorithm	RC4, RC2, DES, 3DES or IDEA, Fortezza
MAC algorithm	MD5 or SHA
Cipher type	Stream or Block
MAC size	MD5(0 or 16 bytes) or SHA (20 bytes)
IV size	Initialization vector size used in CBC

CHR: MT_client_hello.random_no

SHR: MT_server_hello.random_no

SPM: secret_pre_master

SM: secret_master

HSM: Handshake_messages upto current message

CVSM: MT_certificate_verify.cert_signature.MD5_hash

CVSS: MT_certificate_verify.cert_signature.SHA_hash

KB: Key_block

To maintain authenticity of server key exchange messages, signature is taken by encrypting hash with private key of sender. Public parameters contains information regarding different cryptographic algorithms are listed in table 6. SHA-1 is used for creation of Digital Signature Standard (DSS) signature and both MD5 and SHA-1 (36 bytes) are used for RSA signature.

Table 6. Algorithms and its parameters from server

Algorithms	Public Parameters
Ephemeral Diffie-Hellman	A prime no. and its primitive root
RSA	Public key (exponent and Modulo)

The computation of hash is given below.

$$HASH\ (CHR+SHR+public\_parameters) \tag{2}$$

After server certificate and key exchange, it requests for client certificate through *certificate request* message. In the response client sends own certificate, key exchange parameters and ended with *certificate_verify* message. Client key exchange parameters are listed in table 7.

Table 7. Algorithms and parameters from client

Algorithms	Public Parameters
Ephemeral Diffie-Hellman	A prime no. and its primitive root
RSA	48 bits encrypted *secret_pre_master*

certificate_verify contains signature of client certificate and calculated as follow.

$$CVSM= MD5\ (SM+pad\_2+MD5\ (HSM+SM+pad\_1)) \tag{3}$$

$$CVSS=SHA(SM+pad\_2+SHA\ (HSM+SM+pad\_1)) \tag{4}$$

As mention above SHA-1 is used for DSS signature and both MD5 and SHA-1 are used for RSA signature. Handshake messages contain all the message from *MT_client_hello* to *MT_client_key_exchange*. Finished message validates key exchanges are successful or not under new cipher suite which is immediately followed by change cipher spec message. It is computed as given below.

$$MD5\ (SM+pad\_2+MD5\ (HSM+sender\_id+SM+pad\_1) +$$
$$SHA\ (SM+pad\_2+SHA\ (HSM+sender\_id+SM+pad\_1) \tag{5}$$

sender_id represent whether current sender is client or server. Diffie-Hellman server/client exchange parameters are used to calculate respective public keys which are exchanged to compute *SPM* at both sides. RSA client key exchange parameters contain encrypted *SPM* which is decrypted by server key to compute *SM*.

$$SM =MD5\ (SPM+SHA\ (\text{'A'}+SPM+CHR+SHR)) +$$
$$MD5\ (SPM+SHA\ (\text{'BB'}+SPM+CHR+SHR)) +$$
$$MD5\ (SPM+SHA\ (\text{'CCC'}+SPM+CHR+SHR)) \tag{6}$$

In case of 3DES_ECE_CBC_SHA, SSL change cipher spec message requires *server_send_key* (168 bits key + 24 control bits), *client_send_key* (24 bytes), *server_send_MACsecret* (20 bytes), *client_send_MACsecret* (20 bytes), *server_send_IV* (8 bytes), *client_send_IV* (8 bytes) to change pending state to current state. So, it requires a key block of 104 bytes to be generated from *SM* which contains above parameters in sequential order is evaluated as follow.

KB= MD5 (SM+SHA (‘A’+SPM+CHR+SHR)) +
MD5 (SM +SHA (‘BB’+SPM+CHR+SHR)) +
MD5 (SM+SHA (‘CCC’+SPM+CHR+SHR)) + [...] (7)

2.2 TLS Architecture

As TLS is the upgraded version of SSL, it has same architecture and protocols except there are some changes in security parameters and computation of MAC, digital signature and key block. It also introduces some new alert messages and pseudorandom function to strengthen the security compare to SSL are described below.

2.2.1 TLS Record Protocol

i. *Version:* It indicates the major and minor version reinforced by client for current TLS. The major and minor version for different version of TLS are listed in table 8 contradict to major and minor version for SSL are 3 and 0 respectively.

Table.8.Versions of TLS

Major Version	Minor Version	Class
3	1	TLS 1.0
3	2	TLS 2.0
3	3	TLS 3.0

ii. MAC: A cryptographic hash function with shared secret key is used to calculate MAC. Here compression_version is concatenated with other fields which are same as SSL chunk fields shown in Eq.1.

(MAC_secret_key, seq_no +TLS_compression_type + TLS_compression_version + TLS_compressed_chunk_length + TLS_compressed_chunk) (8)

iii. Pseudorandom function (PRF): It takes shared secret, tag and data as inputs to PRF. It is calculated by taking XOR over two hash value (MD5 and SHA). shared_secret_left and shared_secret_right indicate left half and right half of shared secret. Pseudo_MD5 and Pseudo_SHA are called for three times to produces (3x 16 bytes) and (3x 20 bytes) for 48 bytes final output. In case of Pseudo_SHA from 60 bytes last 12 bytes are truncated to produce 48 bytes.

PRF (shared_secret, tag, data) =
(Pseudo_MD5 (shared_secret_left, tag +data)) XOR
(Pseudo_SHA (shared_secret_right, tag+ data)) (9)

Here pseudo random function is called two times by *pseudo_MD5* and *pseudo_SHA* functions where *data’* is the concatenation of *tag* and *data* supply to PRF.

Pseudo_hash (key, data’) =
HMAC_hash (key, pMAC (1) +data’) +
HMAC_hash (key, pMAC (2) + data’) +
HMAC_hash (key, pMAC (3) + data’) + ... (10)

pMAC (0) = data’
pMAC (k) = MAC_hash (key, pMAC(k-1)) (11)

HMAC_hash is *HMAC_MD5* for *Pseudo_MD5* and *HMAC_SHA for Pseudo_SHA.* Both *pad_2* (512 bits) and *pad_1* (512 bits) carries binary values of X5C and X36 respectively which are repeated 64 times.

HMAC_hash (key, data’) = hash ((key’ XOR pad_2) +
hash ((key XOR pad_1) + data’)) (12)

2.2.2 TLS Handshake Protocol

i. *Cipher Suite:* Key exchange and encryption algorithms are supported by TLS are same as SSL listed in table V except Fortezza. Apart from fixed and ephemeral Diffie-Hellman algorithms, it also supports Elliptic Curve Diffie-Hellman.

ii. *Certificate Types:* Response to certificate request message by TLS, RSA or DSS signed certificate is issued where the key exchange parameters are RSA or Diffie-Hellman public parameters listed in table VI and VII.

iii. *Certificate Verify Message:* It carries signature of client certificate calculated over previous handshake messages by hashing these. It eliminates concatenation of master secret and padding bits with handshake messages before hash calculation as those will not add extra security to certificate.

MT_certificate_verify.cert_signature.MD5_hash =
MD5 (handshake_messages) (13)
MT_certificate_verify.cert_signature.SHA_hash =
MD5 (handshake_messages) (14)

iv. *Finished Message:* Contrast to SSL, hash (MD5 and SHA) are computed over handshake messages and concatenation of these are given as input to PRF.

PRF (secret_master, tag, MD5 (handshake_messages)
+SHA (handshake_messages)) (15)

v. *Master Secret and Key Block Computation:* Master secret is calculated by calling PRF function over three input such as pre master secret, tag and concatenation of client and server random number which is much simpler than SSL.

secret_master = PRF (secret_pre_master,
“master secret”,
MT_client_hello.random_no +
MT_server_hello.random_no) (16)

Key block is computed over three inputs such as master secret,tag and concatenation of client and server random no by passing as a parameters to PRF.

key_block = PRF (secret_master, “key expansion”,
MT_client_hello.random_no +
MT_server_hello.random_no) (17)

vi. *Padding:* Unlike SSL, before encryption an arbitrary length of padding bits, length between 0 to 2^8-1are concatenated after MAC to make the chunk size multiple of cipher block size. In SSL, minimum amount of padding bits are added to make multiple of cipher block.

2.2.3 TLS Alert Protocol

In TLS, additional alert messages are introduced to make communication reliable are listed in table 9 [2]. It supports all SSL alert messages except alert code 41.

Table 9. Alert messages of TLS

Codes	Alerts	Representations	Types
22	Record_overflow	Payload size exceeded more than 2^{14} + 2048 bytes	Fatal
48	unknown_ca	CA certificate cannot be trusted or discovered	Fatal
49	accessed_ denied	Negotiation failed due to access control provided by receiver	Fatal
50	decode_ error	Information could not be decoded properly due to incorrect message length	Fatal
51	decrypt_ error	Unable to decrypt the secret key, verify digital signature or authenticity of finished message	Warning/ Fatal
60	export_ restriction	Negotiation against export restriction are detected and terminated	Fatal
70	protocol_ version	Protocol version is not supported by server	Fatal
71	insufficient_ security	Handshaking fail due to stronger cipher suite required by server	Fatal
80	internal_error	Error associated to local system and not related to SSL.	Fatal
90	User_cancelled	Abnormal termination of session by user	Fatal
100	no_renegotiation	Client or server response w.r.t hello request is not suitable for renegotiation	Warning

2.3 Cipher Block Chaining

Cipher Block Chaining is one of the block cipher mode operations which is used to encrypt a series of plaintext chunks. For creation of first cipher text chunk, it gets initialization vector (IV) from *KB* represents in Eq.7. Plaintext chunk from 2 to onwards required IVs are the previous cipher text chunk respectively (see Figure 3).

Cipher (k) = E [key, (Plain (k) XOR cipher (k-1))]

Cipher (0) = IV *(18)*

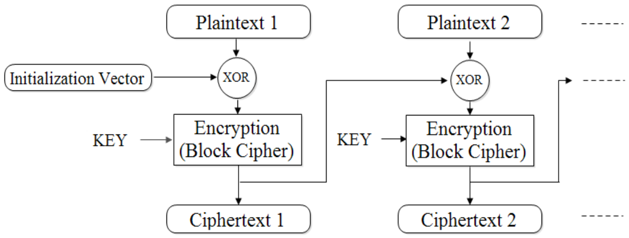

Fig 3: CBC for encryption

In CBC decryption, XOR operation is taken placed after cipher text decrypted by key where IV for first plaintext is from *KB.* Previous cipher text acts as IV for next plain text (see Figure 4).

Plain (k) = D [k, Cipher (k)] XOR Cipher (k-1) *(19)*

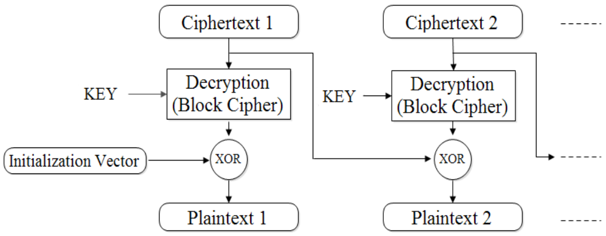

Fig.4. CBC for decryption

3. TYPES OF ATTACKS

One of the biggest threats to transport level security due to flaws in SSL/ TLS, which is used to secure the communication between sender and receiver. Vulnerabilities in SSL/TLS triggers both active and passive attacks such as BEAST, CRIME, TIME, BREACH, LUCKY 13, RC4 BIASES, SSL Renegotiation, POODLE, Truncation, Bar Mitzvah etc.[3] and their fixes are listed in table 10.

3.1 BEAST attack

It is the short form of Browser Exploit Against SSL/ TLS attack occurs by exploit TLS 1.0 and was developed by T. Duong and J. Riazo. It takes the advantages of symmetric encryption and cipher block chaining (CBC) technique to guess secret key which is used to encrypt the plaintext. In TLS 1.0, last cipher text block is the initialization vector for current plaintext. XOR operation between initialization vector and plaintext is encrypted by symmetric key to produce corresponding cipher text. If the hacker can guess a plaintext block, he can guess the symmetric key and check whether cipher text is matched or not [4, 5]. It is one type of brute force attack fixed by the corresponding TLS 1.1 and TLS 1.2.

3.2 CRIME attack

It is the short form of Compression Ratio Info Leak Mass Exploitation attack occurs by hijacking the session by decrypting the session cookies in TLS 1.0 and was developed by J. Riazo and T. Duong [6]. It takes the advantages of TLS and SPDY header compression. SPDY is an open networking protocol and control HTTP traffic developed by Google. Both TLS and SPDY compression techniques use DEFLATE algorithm, which eliminates duplicate string by compression then encrypt it. The key is obtained by cheating the browser and sending encrypted compressed request to genuine website, waiting for the HTTP response size and increasing attack with respect to HTTP responses [7]. Hacker repeats the techniques with different values until the key will be obtained. It is one type of brute force attack fixed by disabling the compression mechanism in TLS 1.1 and TLS 1.2.

3.3 Time attack

Timing Info-Leak Made Easy (TIME) attack by which attacker extracts secret information without eavesdropping into the network and was developed by T. Be'ery and A. Shulman of Imperva. To perform this attack, hacker wants to know cookies location, prefix/suffix and location to insert plaintext. Information about the session cookies is obtained by time taken to get the response from server/ receiver [8]. Due to noise over the network, a single process will be repeated for certain integral number of time and minimal response time is taken as the final response time for that particular request.

Suppose client inputs contain “secret element = unknown data” which is the payload and secret element and its value is reflected in the response. In first iteration for arbitrary user input the response size is 1028 bytes. If in the second iteration the user input is “secret element = a” and the response size is 1008 bytes. So it is taken less time compare to first iteration. With several requests the shortest response time for every character for each position in the payload is computed which is happened to be the correct guess and specific value of the secret element.

3.4 BREACH attack

Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext is the crime attack against the response body and it was developed by A. Prado, N. Harris and Y. Gluck [9]. Attacker exploits the HTTP compression technique (LZ77 algorithm) by guessing character and symbol without downgrading or tampering SSL to launch this attack and its guess will be reflected in response body [8]. It has taken less than 30 seconds for fairly stable pages to obtain the secret like CSRF token, view state etc... It is vulnerable to any version of SSL or TLS. To launch breach attack, both attacker and victim must be in the same network. The command and control center has web server driver called iframe streamer which is going to inject HTTP request in the victim, callback listener whose work is to call back when response come to victim and traffic monitor observes the length of the cipher text coming back. Basic oracle logic is the collection of algorithms is used to guess the secrets. For fighting against Huffman coding, character set pool plus random padding is used and for fighting against block cipher, window technique is used. It is one of the most vulnerable attacks on SSL which is yet to be patched.

3.5 LUCKY 13 attack

It is one of the most vulnerable attacks in SSL till now and was developed by N. A. Fardan and K. Paterson at Royal Holloway, University of London in February 2013. It uses padding oracle technique is a side channel attack which is affected on padding of a cipher text. Attacker exploits TLS’s cipher block chaining by replacing the last some bytes with chosen bytes and watch amount of time taken by server to respond [10]. TLS packets those contain true padding takes less time to process. If TLS generates transaction to fail, it produces a message that carries errors which helps the attacker to send malicious packets in a new session repeatedly backing every foregoing failure [6, 11]. Result shows that 2^{23} sessions required extracting information about cookies and 2^{19} sessions required if 64 bit encoding scheme is used by TLS. Overall LUCKY 13 attack requires 2^{13} sessions; if a byte of information regarding MAC tag or padding is known.

3.6 RC4 BIASES attack

It is also known as ARC4 or ARCFOUR attack discovered by Alfardan, Bernstein, Paterson, Poettering and Schuldt by exploit all versions of SSL/ TLS. RC4-128 encryption algorithm is used to encrypt the payload. It takes 128 keys and generates string of random keys. These keys are XORed with the different block of plaintexts to produce block of cipher texts. The problem is that the random keys generated by RC4 are not quite random which helpful to recover some part of plaintext with large number of TLS encryptions [12, 13]. If same message is encrypted with different RC4 keys, then random cipher texts will be generated. As keys are not quite random or there are tiny biases, the cipher texts will be not quite random or very small biases exist. Attackers tally up these deviations from random by doing statistical analysis of individual locations of the cipher texts. Experimental results show that approximately 2^{32} cipher texts give nearly all plaintexts. Around 2^{30} sessions required to extract plain texts from cipher texts.

3.7 SSL Renegotiation attack

It is happened by exploit SSL 3.0 and all versions of TLS and was discovered by M. Ray and S. Dispensa in August 2009. Attacker hijacks HTTPs connection to add plaintext into the conversion [14]. He/she doesn’t decrypt the client server communication. During secure online transaction, client initiates SSL handshaking process. Hacker blocks the request and captures those packets. Then he initiates new session and complete the handshaking process. After completion of handshaking process, attacker asks the server to credit money to his account during banking transaction. Server asks for renegotiation. Those block packets of victim will be sent to server which will be the new SSL handshake over the session that previously established. Two sessions are enough to lunch attacks against victim. It can be fixed either by disable renegotiation on server side or client-server has to verify about previous handshaking.

3.8 POODLE attack

Padding Oracle On Downgraded Legacy Encryption attack is one of the man in the middle attack where attacker exploit SSL 3.0 vulnerabilities to decrypt HTTP cookies [11]. It was discovered by B. Moller, T. Doung and K. Kotowicz on 14th October 2014. Attacker is sitting between client and server downgrade TLS v1.0 or latter version handshake attempted between them for secure transmission to SSL v3.0. Padding technique is used in SSL v3.0 which is random in nature i.e. padding 1 to L bytes which are not deterministic to obtain integral number of chunks to perform cipher block chaining operation. Those bytes are not covered by MAC and not validated while decrypting. Last byte of the padding indicates number of padding bytes is used which is helpful for hacker to trigger the attack [15]. Attacker copy intermediate byte(s) to last bytes and try to exploit. If the modified last byte is same as previous byte then after decryption correct number of padding bytes will be trunked without affecting MAC bytes. Now the message will be accepted by the server which will be helpful to hacker to recover plaintext byte by byte but one byte at a time by performing XOR operation. 1 out of 256 times the message will be accepted; worst case 255 times out of 256 results error message and session will be aborted but at last time it will be normal.

3.9 FREAK attack

Factoring RSA Export keys (FREAK) is one of the TLS vulnerabilities found in several well-known browsers (e.g. Safari, Android browser, Cisco, Opera). It is also called server spoof attack against browsers. A group of weak export cipher suites used by TLS are targeted by the attacker. These algorithm packages are implemented within several TLS client libraries such as Open SSL, Boring SSL, LibReSSL, IBM JSSE, SChannel etc. Implementation of above libraries in browser makes use of export cipher suite incorrectly, even if non export cipher suite is negotiated between server and client for information exchange. Negotiation of export cipher suite between server and client allows attacker to trick client’s browser to use weak export key by performing MITM attack [16]. FREAK attack downgrades cipher suite that uses RSA key exchange algorithm where key size is lesser than 512 bits. Thus it will take less than 12 hours for factorization [17]. Like FREAK, Logjam vulnerabilities of SSL/TLS allows attacker to downgrade the export cipher suite that uses Diffie-Hellman key exchange algorithm [18]. It can be prevented by disabling export cipher suite in browsers.

3.10 Bar Mitzvah attack

Exploit RC4 steam cipher algorithm supported by SSL/TLS helps to extract information over encrypted communication [12, 19]. Attacker tries to extract weak keys by targeting first 100 bytes of encrypted information out of which 36 bytes belongs to SSL/ TLS finished message. As finished message carries most predictable information, plain finished messages are XORed with encrypted finished messages to extract part of Pseudo Random Number Generator Sequences (PRNGS). After Discarding PRNGS which do not follow the pattern of weak keys generated PRNGS, all the keys of selected PRNGS are used to decrypt cipher text captured by attacker using RC4 algorithm. Keys with 0.5 probabilities are successfully determined which minimizes the number of trials taken by brute force attack as a difference of $2^{11.2}$. This attack unable to extract full plaintexts from cipher texts.

3.11 TLS Truncation Attack

Abnormal termination of TLS connection performed by adversary to keep alive victim session using multiple browser connection [20]. It was developed by B. Smyth and A. Pironti in July 2013. To increase performance, web browser load content through multiple connection. As TLS provides integrity and confidentiality over a single connection, client browser's multiple connections to single server are ordered over TLS single connection. Prior to perform this attack, attacker has full control over network which help to inject/ drop packets into different connections. It is triggered at the time of client logout request by injecting TCP FIN or RESET message for that connection prior to it causes request message unavailable to server due to abnormal termination of connection. As logout conformation come before logout request received by server, attacker launch this attack to keep alive the session without victim knowledge. At last, other connection of browser is used to access victim account and modify it.

Table 10. Attacks and their fixes

Attacks	Fixes
BEAST	Use RC4, 3DES, AES 256
CRIME	Disable TLS compression
TIME	Encrypt then MAC, use AES-GCM ciphers
LUCKY 13	Add random time delays, use authenticated encryption, use RC4
BREACH	Disable HTTP compression
RC4 BIASES	Disable RC4 in SSL/TLS
SSL Renegotiation	Client and server verify previous hand shake
POODLE	disable SSL 3.0 in web browser
FREAK	Configure SSL/TLS with higher version of cipher
Bar Mitzvah	Disable RC4 in SSL/TLS
TLS Truncation	Centralized authentication and chain sign outs

4. CONCLUSION

SSL/ TLS, the two isolated protocols are used to secure the communication channels between two ends by providing two layers of security such as authentication and encryption to user data. A logical or operational error in these protocols gives a way to attacker to exploit it. This paper outlines architecture and operational flow of these protocols and summarizes different types of attacks and their fixes. At last more research on this field has to be done to increase the degree of safety of SSL/ TLS by reducing bugs.

5. REFERENCES

[1] Stalling, W. (2011). Transport-Level Security. In Cryptography and Network Security (5th ed., pp. 485-520). Upper Saddle River, NJ: Pearson.

[2] Panday, K. K. SSL/ TLS Alert Protocol and the Alert Codes. Retrieved October 10, 2014, from https://blogs.msdn.microsoft. com/kaushal/2012/10/05/ssltls-alert-protocol-the-alert-codes/

[3] Sarkar, P. G., and Fitzgerald, S. (2013). Attack on SSL: A Comprehensive study of BEAST, CRIME, TIME, BREACH, LUCKY 13 and RC4 BIASES [PDF]. San Francisco, CA: ISECPartners.

[4] Newman, R. Taming the B.E.A.S.T. - owasp.org. Retrieved October 21, 2014, from https://www.owasp.org/images/1/10/ Taming _the_B.E.A.S.T..pdf

[5] Luedtke, D. (2012, April 18). BEAST attack on SSL/TLS explained-SlideShare [PPT]. Munich: University of German Federal Armed Forces.

[6] BEAST vs. CRIME Attack. (13, October 14). Retrieved November 01, 2014, from http://resources.infosecinstitute.com/ beast-vs-crime-attack/

[7] Beery, T. and Shulman, A. (2013, March). A Perfect Crime? Only Time Will Tell [PDF]. Amsterdam, Netherlands: Blackhat.

[8] Beery, T., and Shulman, A. (2013, October 10). Black Hat EU 2013 – A Perfect CRIME? Only TIME Will Tell. Retrieved November 15, 2014, from http://www.youtube.com/watch?v= rTlpFfTp3-w

[9] GLUCK, Y., HARRIS, N., and PRADO, A. (2013, July 12). BREACH: Reviving the CRIME Attack [PDF].

[10] Fardan, N. J., and Paterson, K. G. (May 2013). 2013 IEEE Symposium on Security and Privacy (pp. 526-540). IEEE.

[11] Franke, D. F. (2014, October 14). How POODLE Happened. Retrieved December 21, 2014, from https://www.dfranke.us /posts/2014-10-14-how-poodle-happened.html.

[12] Bar Mitzvah Attack. Retrieved December 25, 2014, from https://www.blackhat.com/docs/asia-15/materials/asia-15-Mantin-Bar-Mitzvah-Attack-Breaking-SSL-With-13-Year-Old-RC4-Weakness-wp.pdf

[13] Paterson, K. (2013, August 28). On the security of RC4 in TLS and WPA. Retrieved December 25, 2014, from http://www.isg. rhul.ac.uk/tls/

[14] Zoller, T. (2005). TLS/ SSLv3 renegotiation vulnerability explained – G-SEC. Retrieved December 28, 2014, from http://www.g-sec.lu/practicaltls.pdf.

[15] Bowes, R. (2013, January 2). Padding oracle attacks: In depth. Retrieved July 10, 2015, from https://blog.skullsecurity.org/2013/ padding-oracle-attacks-in-depth

[16] Vulnerability Notice: FREAK – Factoring attack on RSA-Export keys. (2015, March 20). Retrieved April 12, 2015, from http://learn .extremenetworks.com/rs/extreme/images/VN-2015-003_FREAK.pdf.

[17] Understanding Common Factor Attacks: An RSA-Cracking Puzzle. Retrieved April 30, 2015, from http://www.loyalty.org/ ~schoen/rsa/

[18] Kerner, S. M. (2015, May 20). Logjam SSL/TLS Vulnerability Exposes Cryptographic Weakness Retrieved August 10, 2015, from http://www.eweek.com/security/logjam-ssltls-vulnerability-exposes-cryptographic-weakness.html

[19] Roos, A. (1995, September 22). Weaks in RC4. Retrieved July 13, 2014, from https://netfuture.ch/1995/09/weak-keys-in-rc4/

[20] Smyth, B., and Pironti, A. (2013, July). Truncating TLS Connections to Violate Beliefs in Web Applications. Retrieved May 10, 2015, from https://media.blackhat.com/us-13/US-13-Smyth-Truncating-TLS-Connections-to-Violate-Beliefs-in-Web-Applications-WP.pdf.