INTERNATIONAL JOURNAL OF RESEARCH IN COMPUTER APPLICATIONS AND ROBOTICS
ISSN 2320-7345

A STUDY ON OPEN SSL VULNERABILITIES WITH HEART BLEED WORM DETECTION

[1]Mrs. N.Jayalakshmi, [2]Mrs. Nimmati Satheesh.

[1]Assistant Professor, PSNA College of Engineering and Technology Dindigul, TamilNadu.
[2]Assistant Professor, PSNA College of Engineering and Technology Dindigul, TamilNadu.
E-Mail: - jayapsna@gmail.com, nimmatisatheesh@gmail.com

Abstract

Heart bleed is a security bug disclosed in April 2014 in the Open SSL cryptography library, which is a widely used implementation of the Transport Layer Security (TLS) protocol. Heart bleed may be exploited regardless of whether the party using a vulnerable Open SSL instance for TLS is a server or a client. It results from improper input validation (due to a missing bounds check) in the implementation of the TLS heartbeat extension, thus the bug's name derives from "heartbeat". The vulnerability is classified as a buffer over-read, a situation where more data can be read than should be allowed.[6]Since its introduction in 1994 the Secure Socket Layer (SSL) protocol (later renamed to Transport Layer Security (TLS)) evolved to the de facto standard for securing the transport layer. SSL/TLS can be used for ensuring data confidentiality, integrity and authenticity during transport. A main feature of the protocol is its flexibility. Modes of operation and security aims can easily be configured through different cipher suites. During its evolutionary development process several flaws were found. However, the flexible architecture of SSL/TLS allowed efficient fixes in order to counter the issues. This paper presents an overview on theoretical and practical attacks of the last 20 years.

Keywords: SSL, TLS, BEAST Attack, CRIME Attack, Heart bleed Detection, RC4.

1. Heart bleed Introduction

Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), are cryptographic protocols designed to provide communications security over a computer network.[1] They use X.509 certificates and hence asymmetric to authenticate the counterparty with whom they are communicating, and to exchange a symmetric key. This session key is then used to encrypt data flowing between the parties. This allows for data/message confidentiality,and message authentication codes for message integrity and as a by-product, message authentication. The attacks launched in the last few years have exploited various features in the TLS mechanism. We are going to discuss these attacks one by one.

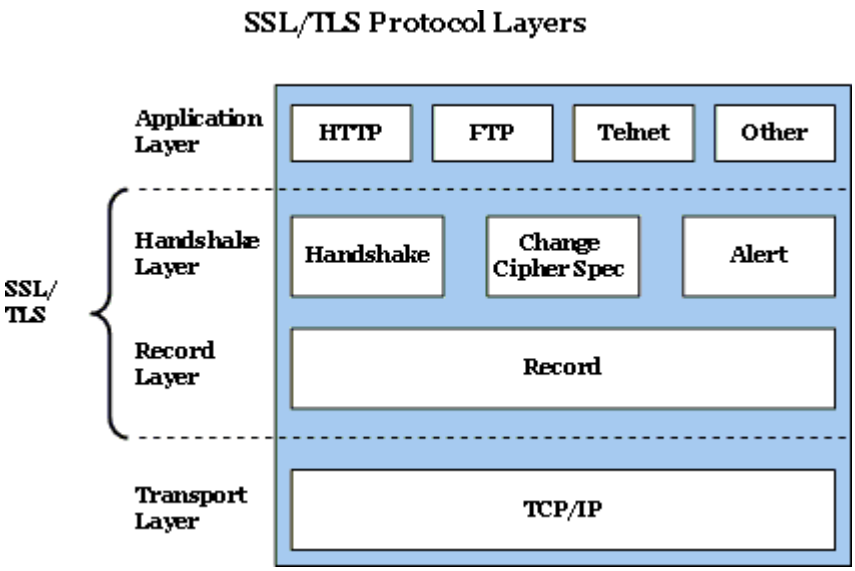
Heart bleed is registered in the Common Vulnerabilities and Exposures system as CVE-2014-

0160.[5] The federal Canadian Cyber Incident Response Centre issued a security bulletin advising system administrators about the bug.[7] A fixed version of Open SSL was released on April 7, 2014, on the same day Heart bleed was publicly disclosed. The Heart bleed bug is a serious vulnerability in the popular Open SSL cryptographic software library. It allows an attacker to read the memory of systems using certain versions of Open SSL, potentially allowing them to access user names, passwords, or even the secret cryptographic keys of the server used for SSL. Obtaining these keys would allow malicious users to observe all communications on that system, allowing further exploit. We will discuss this vulnerability too.

2. TLS, ATTACKS AND ANALYSIS

2.1 SSL Protocol and Types

The SSL protocol has been submitted to the W3O working group on security for consideration as part of a general security approach for the Web, and we are actively working within the W3O and with many of its member entities on establishing open security standards for the net.



This protocol specification was last revised on November 29th, 1994. Recent changes include:

- a fundamental correction to the client-certificate authentication protocol,
- the removal of the username/password messages,
- corrections in some of the cryptographic terminology,
- the addition of a MAC to the messages [see section 1.2],
- The allowance for different kinds of message digest algorithms.

Secure Sockets Layer, more commonly known as SSL, is a protocol that is used to maintain client and server authentication. A site is easily identified as using SSL if it has the small yellow padlock at the bottom of the browser.

In SSL, communication between the server and the client is encrypted using their certificates. This encryption creates virtual information that is not hackable by others. SSL Version 1, a test version, was quickly replaced by SSL version 2, which was the first version, released to the public and was shipped with the Netscape Navigator browser. Today version 2 is still supported despite having some security problems. Later, Microsoft came out with its own version of SSL called PCT. SSL Version 3 is a complete redesign of SSL and fixes the problems found in previous versions as well as having additional features [1].

SSL uses a handshake protocol. Suppose a client wants to make a purchase from a website server, but this server does not know anything about the client.

The first step is for the client to send a message to the server. After the server receives the message, it acknowledges it by sending the client a message in return. The server also sends the client its certificate and asks for the client's certificate. The client exchange message and a certificate verification message. Both the client and server send change cipher spec messages and then send finished messages to end the handshake [2].

A website implements SSL by using HTTPS, which stands for Hypertext Transfer Protocol over Secure Socket Layer. This web protocol was developed by Netscape to encrypt and decrypt page requests as well as the pages that are returned by the web server. HTTPS uses port 443 instead of port 80, which is used for HTTP.

SSL uses a key size of 40-bits for the RC4 stream encryption algorithm. This is considered a sufficient degree of encryption for commercial exchange. Both HTTPS and SSL support the use of X.509 digital certificates from the server. This way, the user can authenticate the sender if needed [3].

The dictionary attack which tends to be more efficient than a brute force attack is where an attack tries every word in a dictionary as a possible password for an encrypted message. This attack is also avoidable because SSL has very large key spaces. The replay attack which reruns messages that were sent earlier is prevented since SSL uses 128-bit nonce value to indicate a unique connection. And as mentioned earlier, the Man-In-the-Middle Attack is prevented by using signed certificates to au public key.

Despite the fact that SSL has the ability to prevent some common attacks, it still has some weaknesses. One of the weaknesses found in SSL is the brute force attack against weak ciphers. This weakness was forced by the US export on Netscape. This weakness still remains one of the most obvious weaknesses of the SSL protocol and it has broken many times [4].

Another weakness in SSL is the renegotiation of the master key. It is known that after a connection has been established, the same master key gets used all the way through the connection. This could be a serious security flaw if SSL are layered underneath a long running connection. One possible solution for this flaw is to force renegotiation of the master key at different times. This way, the difficulty and the cost of the any brute force attack will be multiplied by the number of times that the master key has changed [5].

The Transaction Layer Security protocol, commonly known as TLS, is based on SSL and became its successor. TLS has some changes in its MAC, has clearer and more precise specifications, cleaner handling because of not having a client certificate, and more flexibility.

Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), are cryptographic protocols designed to provide communications security over a computer network.[1] They use X.509 certificates and hence asymmetric to authenticate the counterparty with whom they are communicating,[2] and to exchange a symmetric key. This session key is then used to encrypt data flowing between the parties. This allows for data/message confidentiality, and message authentication codes for message integrity and as a by-product, message authentication.[clarification needed] Several versions of the protocols are in widespread use in applications such as browsing, electronic, Internet faxing, instant messaging,

and voice-over-IP (VoIP). An important property in this context is forward secrecy, so the short-term session key cannot be derived from the long-term asymmetric secret key.[3]

As a consequence of choosing X.509 certificates, certificate authorities and public are necessary to verify the relation between a certificate and its owner, as well as to generate, sign, and administer the validity of certificates. While this can be more beneficial than verifying the identities via a web of trust, the 2013 mass surveillance disclosures made it more widely known that certificate authorities are a weak point from a security standpoint, allowing man-in-the-middle attacks (MITM).[4][5] In the Internet Protocol Suite, TLS and SSL encrypt the data of network connections in the application layer. In model equivalences, TLS/SSL is initialized at layer 5 (session layer) and works at layer 6 (the presentation layer).[citation needed] The session layer has a handshake using an asymmetric cipher in order to establish cipher settings and a shared key for that session; then the presentation layer encrypts the rest of the communication using a symmetric cipher and that session key. In both models, TLS and SSL work on behalf of the underlying transport layer, whose segments carry encrypted data.

TLS is an Internet Engineering Task Force (IETF) standards track protocol, first defined in 1999 and updated in RFC 5246 (August 2008) and RFC 6176 (March 2011). It is based on the earlier SSL specifications (1994, 1995, 1996) developed by Netscape Communications [6] for adding the HTTPS protocol to their Navigator web browser.

TLS is an Internet Engineering Task Force (IETF) standards track protocol, first defined in 1999 and last updated in RFC 5246 (August 2008) and RFC 6176 (March 2011). It is based on the earlier SSL specifications (1994, 1995, 1996) developed by Netscape Communications [1] for adding the HTTPS protocol to their Navigator web browser.

Early research efforts towards transport layer security included the Secure Network Programming (SNP) application programming interface (API), which in 1993 explored the approach of having a secure transport layer API closely resembling Berkeley sockets, to facilitate retrofitting preexisting network applications with security measures [1].

The SSL protocol was originally developed by Netscape. Version 1.0 was never publicly released; version 2.0 was released in February 1995 but "contained a number of security flaws which ultimately led to the design of SSL version 3.0."[2] SSL version 3.0, released in 1996, was a complete redesign of the protocol produced by Paul Kocher working with Netscape engineers Phil Karlton and Alan Freier. Newer versions of SSL/TLS are based on SSL 3.0. The 1996 draft of SSL 3.0 was published by IETF as a historical document in RFC 6101.

The basic algorithm was written by Dr. Taher Elgamal. As the Chief Scientist of Netscape, Taher was recognized as the "father of SSL" [2].

TLS 1.0 was first defined in RFC 2246 in January 1999 as an upgrade of SSL Version 3.0. As stated in the RFC, "the differences between this protocol and SSL 3.0 are not dramatic, but they are significant to preclude interoperability between TLS 1.0 and SSL 3.0." TLS 1.0 does include a means by which a TLS implementation can downgrade the connection to SSL 3.0, thus weakening security. TLS 1.1 was defined in RFC 4346 in April 2006 [3]. It is an update from TLS version 1.0. Significant differences in this version include:

⇒ Added protection against Cipher block chaining (CBC) attacks.
⇒ The implicit Initialization Vector (IV) was replaced with an explicit IV.
⇒ Change in handling of padding errors.

⇒ Support for IANA registration of parameters.

TLS 1.2 was defined in RFC 5246 in August 2008. It is based on the earlier TLS 1.1 specification. Major differences include:

SSL 3.0

⇒ This protocol was released in 1996, but first began with the creation of SSL 1.0 developed by Netscape. Version 1.0 wasn't released, and version 2.0 had a number of security flaws, thus leading to the release of SSL 3.0. Some major improvements of SSL 3.0 over SSL 2.0 are:
⇒ -Separation of the transport of data from the message layer
⇒ -Use of a full 128 bits of keying material even when using the Export cipher
⇒ -Ability of the client and server to send chains of certificates, thus allowing organizations to use certificate hierarchy which is more than two certificates deep.
⇒ -Implementing a generalized key exchange protocol, allowing Diffie-Hellman and Fortezza key exchanges as well as non-RSA certificates.
⇒ -Allowing for record compression and decompression
⇒ -Ability to fall back to SSL 2.0 when a 2.0 client is encountered

TLS 1.0

⇒ This protocol was first defined in RFC 2246 in January of 1999. This was an upgrade from SSL 3.0 and the differences were not dramatic, but they are significant enough that SSL 3.0 and TLS 1.0 don't interoperate. Some of the major differences between SSL 3.0 and TLS 1.0 are:
⇒ -Key derivation functions are different
⇒ -MACs are different - SSL 3.0 uses a modification of an early HMAC while TLS 1.0 uses HMAC.
⇒ -The Finished messages are different
⇒ -TLS has more alerts
⇒ -TLS requires DSS/DH support
⇒ Expansion of support for authenticated encryption ciphers, used mainly for Galois/Counter Mode (GCM) and CCM mode of Advanced Encryption Standard encryption.
⇒ TLS Extensions definition and Advanced Encryption Standard cipher suites were added [4].

All TLS versions were further refined in RFC 6176 in March 2011 removing their backward compatibility with SSL such that TLS sessions will never negotiate the use of Secure Sockets Layer (SSL) version 2.0.

2.2 Study of the Internet Protocol TLS

The TLS protocol allows client-server applications to communicate across a network in a way designed to prevent eavesdropping and tampering.

Since protocols can operate either with or without TLS (or SSL), it is necessary for the client to indicate to the server the setup of a TLS connection. There are two main ways of achieving this. One option is to use a different port number for TLS connections (for example port 443 for HTTPS). The other is for the client to request that the server switch the connection to TLS using a protocol-specific mechanism (for example STARTTLS for mail and news protocols).

Once the client and server have agreed to use TLS, they negotiate a connection by using a

handshaking procedure [5]. During this handshake, the client and server agree on various parameters used to establish the connection's security:

1. The client sends the server the client's SSL version number, cipher settings, session-specific data, and other information that the server needs to communicate with the client using SSL.
2. The server sends the client the server's SSL version number, cipher settings, session-specific data, and other information that the client needs to communicate with the server over SSL. The server also sends its own certificate, and if the client is requesting a server resource that requires client authentication, the server requests the client's certificate.
3. The client uses the information sent by the server to authenticate the server e.g., in the case of a web browser connecting to a web server, the browser checks whether the received certificate's subject name actually matches the name of the server being contacted, whether the issuer of the certificate is a trusted certificate authority, whether the certificate has expired, and, ideally, whether the certificate has been revoked. If the server cannot be authenticated, the user is warned of the problem and informed that an encrypted and authenticated connection cannot be established. If the server can be successfully authenticated, the client proceeds to the next step.
4. Using all data generated in the handshake thus far, the client (with the cooperation of the server, depending on the cipher in use) creates the pre-master secret for the session, encrypts it with the server's public key (obtained from the server's certificate, sent in step 2), and then sends the encrypted pre-master secret to the server.
5. If the server has requested client authentication (an optional step in the handshake), the client also signs another piece of data that is unique to this handshake and known by both the client and server. In this case, the client sends both the signed data and the client's own certificate to the server along with the encrypted pre-master secret.
6. If the server has requested client authentication, the server attempts to authenticate the client. If the client cannot be authenticated, the session ends. If the client can be successfully authenticated, the server uses its private key to decrypt the pre-master secret, and then performs a series of steps (which the client also performs, starting from the same pre-master secret) to generate the master secret.
7. Both the client and the server use the master secret to generate the session keys, which are symmetric keys used to encrypt and decrypt information exchanged during the SSL session and to verify its integrity (that is, to detect any changes in the data between the time it was sent and the time it is received over the SSL connection).
8. The client sends a message to the server informing it that future messages from the client will be encrypted with the session key. It then sends a separate (encrypted) message indicating that the client portion of the handshake is finished.
9. The server sends a message to the client informing it that future messages from the server will be encrypted with the session key. It then sends a separate (encrypted) message indicating that the server portion of the handshake is finished.
10. The SSL handshake is now complete and the session begins. The client and the server use the session keys to encrypt and decrypt the data they send to each other and to validate its integrity [5].

This is the normal operation condition of the secure channel. At any time, due to internal or external stimulus (either automation or user intervention), either side may renegotiate the connection, in which case, the process repeats itself [2].

This concludes the handshake and begins the secured connection, which is encrypted and decrypted with the key material until the connection closes.

If any one of the above steps fails, the TLS handshake fails and the connection is not created.

In step 3, the client must check a chain of "signatures" from a "root of trust" built into, or added to, the client. The client must also check that none of these have been revoked; this is not often implemented correctly, but is a requirement of any public-key authentication system. If the particular signer beginning this server's chain is trusted, and all signatures in the chain remain trusted, then the Certificate (thus the server) is trusted.

The TLS protocol exchanges records which encapsulate the data to be exchanged in a specific format. Each record can be compressed, padded, appended with a message authentication code (MAC), or encrypted, all depending on the state of the connection. Each record has a content type field that designates the type of data encapsulated, a length field and a TLS version field. The data encapsulated may be control or procedural messages of the TLS itself, or simply the application data needed to be transferred by TLS. The specifications (cipher suite, keys etc.) required to exchange application data by TLS, are agreed upon in the "TLS handshake" between the client requesting the data and the server responding to requests. The protocol therefore defines both the structure of payloads transferred in TLS and the procedure to establish and monitor the transfer.

When the connection starts, the record encapsulates a "control" protocol the handshake messaging protocol (content type 22). This protocol is used to exchange all the information required by both sides for the exchange of the actual application data by TLS. It defines the messages formatting or containing this information and the order of their exchange. These may vary according to the demands of the client and server— i.e., there are several possible procedures to set up the connection. This initial exchange results in a successful TLS connection (both parties ready to transfer application data with TLS) or an alert message (as specified below).

A simple connection example follows, illustrating a handshake where the server (but not the client) is authenticated by its certificate:

1. Negotiation phase:
    - A client sends a Client Hello message specifying the highest TLS protocol version it supports, a random number, a list of suggested Cipher Suites and suggested compression methods. If the client is attempting to perform a resumed handshake, it may send a session ID.
    - The server responds with a Server Hello message, containing the chosen protocol version, a random number, Cipher Suite and compression method from the choices offered by the client. To confirm or allow resumed handshakes the server may send a session ID. The chosen protocol version should be the highest that both the client and server support. For example, if the client supports TLS1.1 and the server supports TLS1.2, TLS1.1 should be selected; SSL 3.0 should not be selected.

⇒ The server sends its Certificate message (depending on the selected cipher suite, this may be omitted by the server).

⇒ The server sends a ServerHelloDone message, indicating it is done with handshake negotiation.

⇒ The client responds with a ClientKeyExchange message, which may contain a PreMasterSecret, public key, or nothing. (Again, this depends on the selected cipher.) This Pre-Master Secret is encrypted using the public key of the server certificate.

- The client and server then use the random numbers and Pre-Master Secret to compute a common secret, called the "master secret". All other key data for this connection is derived from this master secret (and the client- and server-generated random values), which is passed through a carefully designed pseudorandom function.

1. The client now sends a Change Cipher Spec record, essentially telling the server, "Everything I tell you from now on will be authenticated (and encrypted if encryption parameters were present in the server certificate)." The Change Cipher Spec is itself a record-level protocol with content type of 20.
   - Finally, the client sends an authenticated and encrypted finished message, containing a hash and MAC over the previous handshake messages.
   - The server will attempt to decrypt the client's finished message and verify the hash and MAC. If the decryption or verification fails, the handshake is considered to have failed and the connection should be torn down.
2. Finally, the server sends a Change Cipher Spec, telling the client, "Everything I tell you from now on will be authenticated (and encrypted, if encryption was negotiated)."
   - The server sends its authenticated and encrypted finished message.
   - The client performs the same decryption and verification.
3. Application phase: at this point, the "handshake" is complete and the application protocol is enabled, with content type of 23. Application messages exchanged between client and server will also be authenticated and optionally encrypted exactly like in their Finished message. Otherwise, the content type will return 25 and the client will not authenticate.

Public key operations (e.g., RSA) are relatively expensive in terms of computational power. TLS provides a secure shortcut in the handshake mechanism to avoid these operations: resumed sessions. Resumed sessions are implemented using session IDs or session tickets.

Apart from the performance benefit, resumed sessions can also be used for single sign-on as it is guaranteed that both the original session as well as any resumed session originate from the same client. This is of particular importance for the FTP over TLS/SSL protocol which would otherwise suffer from a man-in-the-middle attack in which an attacker could intercept the contents of the secondary data connections.

In an ordinary full handshake, the server sends a session id as part of the Server Hello message. The client associates this session id with the server's IP address and TCP port, so that when the client connects again to that server, it can use the session id to shortcut the handshake. In the server, the session id maps to the cryptographic parameters previously negotiated, specifically the "master secret". Both sides must have the same "master secret" or the resumed handshake will fail (this prevents an eavesdropper from using a session id). The random data in the Client Hello and Server Hello messages virtually guarantee that the generated connection keys will be different from in the previous connection. In the RFCs, this type of handshake is called an abbreviated handshake. It is also described in the literature as a restart handshake.

RFC 5077 extends TLS via use of session tickets, instead of session IDs. It defines a way to resume a TLS session without requiring that session-specific state is stored at the TLS server.

When using session tickets, the TLS server stores its session-specific state in a session ticket and sends the session ticket to the TLS client for storing. The client resumes a TLS session by sending the session ticket to the server, and the server resumes the TLS session according to the

Session-specific state in the ticket. The session ticket is encrypted and authenticated by the

server, and the server verifies its validity before using its contents.

One particular weakness of this method is that it always limits encryption and authentication security of the transmitted TLS session ticket to AES128-CBC-SHA256, no matter what other TLS parameters were negotiated for the actual TLS session. This means that the state information (the TLS session ticket) is not as well protected as the TLS session itself. Of particular concern is OpenSSL's storage of the keys in an application-wide context (SSL_CTX), i.e. for the life of the application, and not allowing for re-keying of the AES128-CBC-SHA256 TLS session tickets without resetting the application-wide OpenSSL context (which is uncommon, error-prone and often requires manual administrative intervention) [5].

2.3 Transport Layer Security (TLS)

TLS has a variety of security measures:

- ⇒ Protection against a downgrade of the protocol to a previous (less secure) version or a weaker cipher suite.
- ⇒ Numbering subsequent Application records with a sequence number and using this sequence number in the message authentication codes (MACs).
- ⇒ Using a message digest enhanced with a key (so only a key-holder can check the MAC). The HMAC construction used by most TLS cipher suites is specified in RFC 2104 (SSL 3.0 used a different hash-based MAC).
- ⇒ The message that ends the handshake ("Finished") sends a hash of all the exchanged handshake messages seen by both parties.
- ⇒ The pseudorandom function splits the input data in half and processes each one with a different hashing algorithm (MD5 and SHA-1), then XORs them together to create the MAC. This provides protection even if one of these algorithms is found to be vulnerable [5].

2.4 Heartbleed Open SSL amenability

The Heartbleed Bug is a serious vulnerability in the popular OpenSSL cryptographic software library. This weakness allows stealing the information protected, under normal conditions, by the SSL/TLS encryption used to secure the Internet. SSL/TLS provides communication security and privacy over the Internet for applications such as web, email, instant messaging (IM) and some virtual private networks (VPNs).

The Heartbleed bug allows anyone on the Internet to read the memory of the systems protected by the vulnerable versions of the OpenSSL software. This compromises the secret keys used to identify the service providers and to encrypt the traffic, the names and passwords of the users and the actual content. This allows attackers to eavesdrop on communications, steal data directly from the services and users and to impersonate services and users.

The Heartbleed bug is a serious vulnerability in the popular OpenSSL cryptographic software library, affecting versions 1.0.1 to 1.0.1f. This weakness allows stealing the information protected, under normal conditions, by the SSL/TLS encryption used to secure the data payloads. SSL/TLS provides communication security and privacy over the Internet for applications such as web, email, instant messaging (IM) and some virtual private networks (VPNs).

The Heartbleed bug allows anyone on the Internet to read the memory of the systems protected by the vulnerable versions of the OpenSSL software. This compromises the secret keys used to identify the service providers and to encrypt the traffic, the names and passwords of the users and the actual content. This allows attackers to eavesdrop on communications, steal data directly from the services and users and to impersonate services and users.

On April 7 th 2014 OpenSSL and a team of security engineers published advisories regarding a severe vulnerability that "allows anyone on the Internet to read the by vulnerable versions of the Open SSL software". They have dubbed this vulnerability "Heart bleed" as it refers to a memory function leak used by in Open SSL a heart beat .SSL and TLS are cryptographic protocols designed to secure communications over the internet by way of certificates and asymmetric cryptography. This is implemented in conjunction with Certificate Authorities (CA) and Public Key Infrastructure (PKI). Collectively this forms the basis upon which

Extremely wide reaching and dangerous. Open SSL is deployed with open source web servers such as Apache and NGINX which account for over 50% of active sites on the internet [6].

The Heartbleed attack relies on what is considered a relatively simple programming error that performing correct error bounds checking. This allows an attacker to craft conversations with an OpenSSL Client or Server that reads outside of properly allocated memory. Due to the fact that OpenSSL handles account, certificate, and key information, reading in to memory can reveal extremely sensitive data. However, one of the is coverers of the vulnerability recently tweeted in an apparent attempt to allay concerns terns make stating private "heap alloc key exposure unlikely for #heart bleed #dont panic" this I is Although highly dependent correct, on how committed an attacker is in their efforts. Prolonged and recurring exploitation of this vulnerability against a system radically increases the likelihood of exposing private key information. The heartbeat function does serve a legitimate purpose in that it allows both parties in a communication channel to maintain a session while no longer actively exchanging data. There are multiple proof-of-concepts available in the wild demonstrating exploitation techniques against this vulnerability [7].

Researchers have classified the type of information being leaked into four categories:

1. Primary Key Material – encryption keys are leaked allowing attackers to inspect confidential traffic and impersonate the service.
2. Secondary Key Material – user account and password information can be stolen
3. Protected Content – actual confidential information contained within a previously assumed secure communication channel is exposed.
4. Collateral – incidental information gleaned during the attack with regards to OpenSSL implementation specifics and architecture [8].

Unfortunately, exploitation of this vulnerability does not record log evidence that can be used as an indicator of attack. However, IDS/IPS systems may be able to detect malicious heartbeat request/response communications based on the record type (and size) contained within the protocol. As described in the software vulnerability section above, detection is possible by comparing the size of a request against its reply.

To elaborate, systems with packet inspection capabilities (IDS/IPS, Analytics, Proxy) can look for request and response packets containing matches to specific hexadecimal values for different TLS versions. One must also factor in the size of the packet in order to reduce false positives and avoid simply identifying legitimate heartbeat communication [9].

Some operating system distributions that have shipped with potentially vulnerable OpenSSL version:

1. Debian Wheezy (stable), OpenSSL 1.0.1e-2+deb7u4.
2. Ubuntu 12.04.4 LTS, OpenSSL 1.0.1-4ubuntu5.11.

3. CentOS 6.5, OpenSSL 1.0.1e-15.
4. Fedora 18, OpenSSL 1.0.1e-4.
5. OpenBSD 5.3 (OpenSSL 1.0.1c 10 May 2012) and 5.4 (OpenSSL 1.0.1c 10 May 2012).
6. FreeBSD 10.0 – OpenSSL 1.0.1e 11 Feb 2013.
7. NetBSD 5.0.2 (OpenSSL 1.0.1e).
8. OpenSUSE 12.2 (OpenSSL 1.0.1c).

Operating system distribution with versions that are not vulnerable:

1. Debian Squeeze (oldstable), OpenSSL 0.9.8o-4squeeze14.
2. SUSE Linux Enterprise Server.
3. FreeBSD 8.4 – OpenSSL 0.9.8y 5 Feb 2013.
4. FreeBSD 9.2 – OpenSSL 0.9.8y 5 Feb 2013.
5. FreeBSD 10.0p1 – OpenSSL 1.0.1g (At 8 Apr 18:27:46 2014 UTC).
6. FreeBSD Ports – OpenSSL 1.0.1g (At 7 Apr 21:46:40 2014 UTC) [10].

That's a lot of system's that are vulnerable. We all thought Linux being the bearer of security flag, apparently not anymore!

There are several versions of the Heartbleed exploit actively in the wild, some are simply being used to test if systems are vulnerable, as well as more robust versions available in Metasploit and other frameworks. To watch potential exploits come through I have left a honeypot website purposely vulnerable to the Heartbleed bug, with a script that loads fake password and other random seemingly juicy data files into RAM [14].

To get a better picture of the Heartbleed vulnerability in our environment, we can use the full Tripwire suite. Tripwire IP360 provides reporting on the state of the vulnerability in your environment. Tripwire Log Center provides a guard dog on your network looking for indicators of Heartbleed exploits in real-time from IDS and other systems.

If we bring the two products together as well, when a Heartbleed exploit against a host is detected targeting a host, Tripwire Log Center can lookup vulnerability data on that host to better understand the risk. If the system attacked is vulnerable you can fire off alerts to your team, or activate scripts to automate remediation and counter measures in real-time (Fig.2).

Also there is another technique to detect Heartbleed vulnerability. This technique uses a BPF packet filter to automatically flag larger-than-typical TLS heartbeat responses from the server, and can be used with Wireshark and tcpdump as well as with the Riverbed AppResponse, Shark, and Pilot products. (AppResponse and Shark support many terabytes of stored packets, coupled with the ability to quickly analyze those packets; more Riverbed product specific hints will follow in a separate blog post) [15].

In addition, for the majority of published Heartbleed exploits so far (which are moving the compromised data in the clear on the wire), this technique also identifies what exact data was compromised (e.g., the set of user passwords exposed, vs. the "crown jewels" of a server's Private keys).

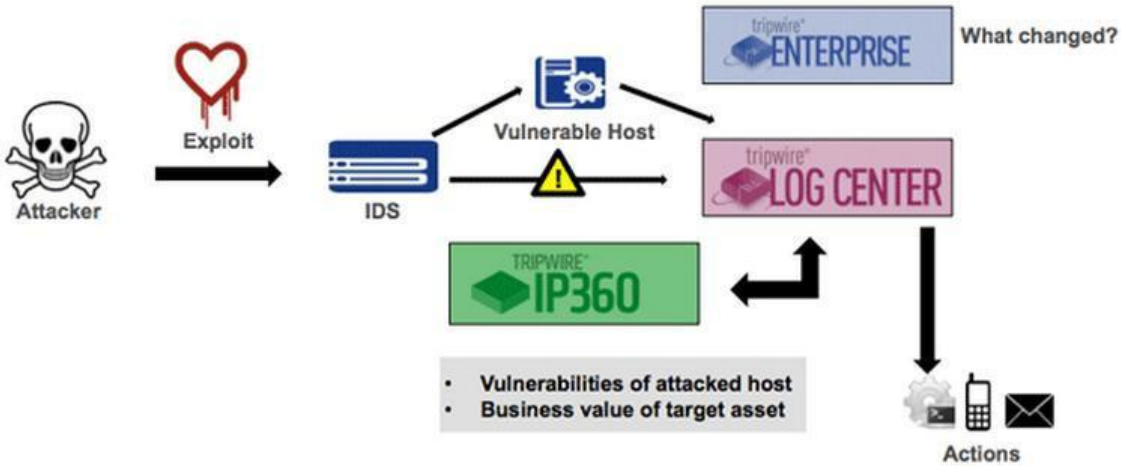


FIGURE 2: Heartbleed Vulnerability [14].

2.5 BEAST Attack

The ability to mount an adaptive chosen plaintext attack with predictable initialization vectors (IVs) against SSL/TLS using cipher block chaining (CBC) was known in 2004, but until late 2011 was thought to be largely theoretical.

Researchers Thai Doung and Juliano Rizzo found a way to exploit the vulnerability and demonstrated a live attack against Paypal at the Ekoparty security conference in September of 2011. Doung and Rizzo had notified and had been working with major software vendors including

Mozilla and Google to release a patch (CVE-2011-3389). Although the vulnerability is cryptographic in nature, it requires certain conditions to be successful. The proof of concept code presented at the conference also required a Java-based Same Origin Policy (SOP) bypass that they had found during their research and which has been patched by Oracle [16].

BEAST leverages a type of cryptographic attack called a chosen-plaintext attack. The attacker mounts the attack by choosing a guess for the plaintext that is associated with a known ciphertext. To check if a guess is correct, the attacker needs access to an encryption oracle [19] to see if the encryption of the plaintext guess matches the known ciphertext. To defeat a chosen-plaintext attack, popular configurations of TLS use two common mechanisms: an initialization vector (IV) and a cipher block chaining mode (CBC). An IV is a random string that is XORed with the plaintext message prior to encryption — even if you encrypt the same message twice, the ciphertext will be different, because the messages were each encrypted with a different random IV. The IV is not secret; it just adds randomness to messages, and is sent along with the message in the clear. It would be cumbersome to use and track a new IV for every encryption block (AES operates on 16-byte blocks), so for longer messages CBC mode simply uses the previous ciphertext block as the IV for the following plaintext block. The use of IVs and CBC is not perfect: a chosen-plaintext attack can occur if the attacker is able to predict the IV that will be used for encryption of a message under their control and the attacker knows the IV that was used for the relevant message they are trying to guess. 8 This new research demonstrated that the above attack can be mounted against TLS under certain conditions. When a SSL 3.0 or TLS 1.0 session uses multiple packets, subsequent packets use an IV that is the last ciphertext block of the previous packet, essentially treating the session as one long message. This allows an attacker

who can see encrypted messages sent by the victim to see the IV used for the session cookie, the because cookie's location is predictable, and also know the IV that will be used at the beginning of the next message packet (the last ciphertext block from the current message packet). If the attacker can also ``choose'' a plaintext message sent on behalf of the victim, they can make a guess at the session cookie and see if the ciphertext matches.

A successful implementation of the attack requires browser or web technologies to meet the above criteria. For clarity, we walk through the example as follows: The network attacker (who we will call Mallory) has the ability to eavesdrop on the network (e.g., over a wireless network) and coerces Alice to visit http://mallory.com perhaps through phishing, advertising or another attack. The malicious website contains an attack script that forces Alice's browser to make a request to http://bob.com and Mallory records the encrypted cookie. Using a technology that allows Mallory to adapt the attack through a SOP bypass or technology that allows multi-origin communication, Mallory now tries to guess the session cookie as the first block of subsequent requests and sees if the resulting ciphertext matches the previously recorded session cookie ciphertext [20].

This class of attack is well known enough that it was mitigated in TLS version 1.1; however, due primarily to client compatibility reasons neither TLS 1.1 or 1.2 are widely supported on the web and most vendors still require support (i.e. fall back) for SSL v3.0 and TLS v1.0.

Browser vendors have attempted to implement a workaround to address the vulnerability at the implementation level while still remaining compatible with the SSL 3.0/TLS 1.0 protocol. These initially included inserting empty fragments into the message in order to randomize the IV as in the case of OpenSSL, and when that proved problematic to reliably implement, 1/n-1 record splitting where a single byte of the plaintext is injected in each record.

The resulting padding added to complete the block (16 or 15 bytes) is random, and its search space is too high for an attacker to guess.

2.6 SSL/TLS CRIME and BREACH Attacks

The authors of the BEAST attack are also the creators of the later CRIME attack, which can allow an attacker to recover the content of web cookies when data compression is used along with TLS

[21]. When used to recover the content of secret authentication cookies, it allows an attacker to perform session hijacking on an authenticated web session.

While the CRIME attack was presented as a general attack that could work effectively against a large number of protocols, only exploits against SPDY request compression and TLS compression were demonstrated and largely mitigated in browsers and servers. The CRIME exploit against HTTP compression has not been mitigated at all, even though the authors of CRIME have warned that this vulnerability might be even more widespread than SPDY and TLS compression combined.
BREACH is an instance of the CRIME attack against HTTP compression - the use by many web browser and web servers of zip or DEFLATE data compression algorithms via the content-encoding option within HTTP.[1] Given this compression oracle, the rest of the BREACH attack follows the same general lines as the CRIME exploit, by performing an initial blind brute-force search to guess a few bytes, followed by divide-and-conquer search to expand a correct guess to an arbitrarily large amount of content.

While the CRIME attack was presented as a general attack that could work effectively against a large number of protocols, including but not limited to TLS, and application-layer protocols such as SPDY or HTTP, only exploits against TLS and SPDY were demonstrated and largely mitigated in browsers and servers. The CRIME exploit against HTTP compression has not been mitigated at all, even though the authors of CRIME have warned that this vulnerability might be even more widespread than SPDY and TLS compression combined. In 2013 a new instance of the CRIME attack against HTTP compression, dubbed BREACH, was announced. Built based on the CRIME attack a BREACH attack can extract login tokens, email addresses or other sensitive information from TLS encrypted web traffic in as little as 30 seconds (depending on the number of bytes to be extracted), provided the attacker tricks the victim into visiting a malicious web link or is able to inject content into valid pages the user is visiting (ex: a wireless network under the control of the attacker). All versions of TLS and SSL are at risk from BREACH regardless of the encryption algorithm or cipher used.[20] Unlike previous instances of CRIME, which can be successfully defended against by turning off TLS compression or SPDY header compression, BREACH exploits HTTP compression which cannot realistically be turned off, as virtually all web servers rely upon it to improve data transmission speeds for users.[24] This is a known limitation of TLS as it is susceptible to chosen-plaintext attack against the application-layer data it was meant to protect.

Compression Ratio Info-leak Made Easy (CRIME) is an attack on SSL/TLS that was developed by researchers Juliano Rizzo and Thai Duong. CRIME is a side-channel attack that can be used to discover session tokens or other secret information based on the compressed size of HTTP requests. The technique exploits web sessions protected by SSL/TLS when they use one of two data-compression schemes (DEFLATE and gzip) which are built into the protocol and used for reducing network congestion or the loading time of web-pages. Rizzo and Doung demonstrated it at the Ekoparty security conference in September 2012 after notifying major affected software vendors, including Mozilla and Google (CVE-2012-4929 18). CRIME is known to work against SSL/TLS compression and SPDY, although other encrypted and compressed protocols are also likely vulnerable [24].

In a single session the same secret/cookie is sent with every request by the browser. TLS has an optional compression feature where data can be compressed before it is encrypted. Even though TLS encrypts the content in the TLS layer, an attacker can see the length of the encrypted request passing over the wire, and this length directly depends on the plaintext data which is being compressed. Finally, an attacker can make the client generate compressed requests that contain attacker-controlled data in the same stream with secret data. The CRIME attack exploits these properties of browser-based SSL. To leverage these properties and successfully implement the CRIME attack, the following conditions must be met:

- The attacker can intercept the victim's network traffic. (e.g. the attacker shares the victim's (W)LAN or compromises victim's router).
- Victim authenticates to a website over HTTPS and negotiates TLS compression with the server.
- Victim accesses a website that runs the attackers code [25].

This attack is feasible on all browsers and servers that support TLS compression. According to the Qualys SSL Lab's SSL Pulse test data showed 42% of servers and 45% of the browsers supported TLS compression when the attack was released. Internet Explorer, Safari, and Opera were not affected, as they did not support TLS compression.

Among the widely used web browsers, Google Chrome (NSS) and Mozilla Firefox, as well as

Amazon Silk supported TLS compression as they implement DEFLATE. The attack also worked against several popular Web services that support TLS compression on the server side, such as Gmail, Twitter, Dropbox and Yahoo Mail. This attack worked for all TLS versions and all cipher suites (AES and RC4) and even if HSTS is active and preloaded by the browser vendor.

CRIME is a the brute-force attack, so it requires O(W) requests where W is cookie charset, with the possibility to optimize to O(log(W)). The modified version of SSL Strip by Moxie Marlinspike can be used in a public network to launch a man-in-the-middle attack which will satisfy one requirement of the attack. This tool strips the ongoing SSL/TLS session and performs a man-in-the-middle attack by acting as a proxy. The proof of concept code by Krzysztof Kotowicz is also useful to simulate the attack. Duong and Rizzo's pseudo code works well in practice, but does not include a mechanism to sync the JavaScript with the program observing lengths on the network.

Browsers still support HTTP compression, and this attack is possible on HTTP compressed sessions. Timing Info-leak Made Easy (TIME) is an extension of this attack. Recently Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext (BREACH) introduced a new targeted techniques to reliably retrieve encrypted secrets [26].

BREACH exploits the compression in the underlying HTTP protocol. Therefore, turning off TLS compression makes no difference to BREACH, which can still perform a chosen-plaintext attack against the HTTP payload.[2]

As a result, clients and servers are either forced to disable HTTP compression completely, reducing performance, or to adopt workarounds to try to foil BREACH in individual attack scenarios, such as using cross-site request forgery (CSRF) protection.[3]

Another suggested approach is to disable HTTP compression whenever the referrer header indicates a cross-site request, or when the header is not present.[4] This approach allows effective mitigation of the attack without losing functionality, only incurring a performance penalty on affected requests.

As BREACH focuses on the HTTP compression of the response body, it is possible to mount on all versions of SSL/TLS, and does not require TLS-layer compression. The cipher suite used during the session negotiation does not affect this attack. The number of requests required are proportional to the size of secret, but in general BREACH attack can be exploited with just a few thousand requests, and under a minute. In short, the scope of this attack includes a considerable portion of the HTTP traffic in the Internet as a large portion of enterprise applications and online websites use HTTP compression to optimize bandwidth.

The three main requirements for exploitation of the vulnerability to be effective are:

1. The application supports HTTP compression.
2. The response should reflect back user's input.
3. The response should have some sensitive/ secret information embedded in the body.

If the user's input is not reflected, there is no possible way to mount a chosen plaintext attack and measure the size of the responses. This attack targets the secret information in the response body (e.g. CSRF tokens), not the session cookie in the request header. So this is useful only if the the response of this attack contains sensitive information.

Like CRIME and TIME, the oracle needs to be aware of Huffman coding scheme and overcome the false positives generated due to the same. In their research paper, Gluck, Harris, and Prado gave a detailed explanation on methods to overcome the aberrations caused by the subtle inner working of the DEFLATE and how they were able to optimize the attack [27].

2.7 Crypt analysis of RC4

In cryptography, RC4 (also known as ARC4 or ARCFOUR meaning Alleged RC4, see below) is the most widely used software stream cipher and is used in popular protocols such as Transport Layer Security (TLS) (to protect Internet traffic) and WEP (to secure wireless networks). While remarkable for its simplicity and speed in software, RC4 has weaknesses that argue against its use in new systems. It is especially vulnerable when the beginning of the output keystream is not discarded, or when nonrandom or related keys are used; some ways of using RC4 can lead to very insecure cryptosystems such as WEP.

RC4, a stream cipher designed by Rivest for RSA Data Security Inc., has found several commercial applications, but little public analysis has been done to date. In this paper, alleged RC4 (hereafter called RC4) is described and existing analysis outlined. The properties of RC4, and in particular its cycle structure, are discussed. Several variants of a basic "tracking" attack are described, and we provide experimental results on their success for scaled-down versions of RC4. TRC4 generates a pseudorandom stream of bits (a keystream). As with any stream cipher, these can be used for encryption by combining it with the plaintext using bit-wise exclusive-or; decryption is performed the same way (since exclusive-or with given data is an involution). (This is similar to the Vernam cipher except that generated pseudorandom bits, rather than a prepared stream, are used.) To generate the keystream, the cipher makes use of a secret internal state which consists of two parts:

- A permutation of all 256 possible bytes (denoted "S" below).
- Two 8-bit index-pointers (denoted "i" and "j").

The permutation is initialized with a variable length key, typically between 40 and 256 bits, using the key-scheduling algorithm (KSA). Once this has been completed, the stream of bits is generated using the pseudo-random generation algorithm (PRGA) [29].

Many stream ciphers are based on linear feedback shift registers (LFSRs), which, while efficient in hardware, are less so in software. The design of RC4 avoids the use of LFSRs, and is ideal for software implementation, as it requires only byte manipulations. It uses 256 bytes of memory for the state array, S[0] through S[255], k bytes of memory for the key, key[0] through key[k-1], and integer variables, i, j, and K. Performing a modular reduction of some value modulo 256 can be done with a bitwise AND with 255 (which is equivalent to taking the low-order byte of the value in question) [31].

Unlike a modern stream cipher (such as those in eSTREAM), RC4 does not take a separate nonce alongside the key. This means that if a single long-term key is to be used to securely encrypt multiple streams, the cryptosystem must specify how to combine the nonce and the long-term key to generate the stream key for RC4. One approach to addressing this is to generate a "fresh" RC4 key by hashing a long-term key with a nonce. However, many applications that use RC4 simply concatenate key and nonce; RC4's weak key schedule then gives rise to related key attacks, like the Fluhrer, Mantin and Shamir attack (which is famous for breaking the WEP standard) [33].

It is noteworthy, however, that RC4, being a stream cipher, was for a period of time the only common cipher that was immune[26] to the 2011 BEAST attack on TLS 1.0. The attack exploits a known weakness in the way cipher block chaining mode is used with all of the other ciphers supported by TLS 1.0, which are all block ciphers.

In 2013 there was a new attack scenario proposed by AlFardan, Bernstein, Paterson, Poettering and Schuldt that uses new statistical biases in RC4 key table[30] to recover plaintext with large number of TLS encryptions [32].

2.8 Attacks on RC4

In spite of existing attacks on RC4 that break it, the cipher suites based on RC4 in SSL and TLS were at one time considered secure because of the way the cipher was used in these protocols defeated the attacks that broke RC4 until new attacks disclosed in March 2013 allowed RC4 in TLS to be feasibly completely broken. In 2011 the RC4 suite was actually recommended as a work around for the BEAST attack. In 2013 a vulnerability was discovered in RC4 suggesting it was not a good workaround for BEAST. An attack scenario was proposed by AlFardan, Bernstein, Paterson, Poettering and Schuldt that used newly discovered statistical biases in the RC4 key table to recover parts of the plaintext with a large number of TLS encryptions. A double-byte bias attack on RC4 in TLS and SSL that requires 13×2^{20}encryptions to break RC4 was unveiled on 8 July 2013, and it was described as "feasible" in the accompanying presentation at the 22nd USENIX Security Symposium on August 15, 2013.

However, many modern browsers have been designed to defeat BEAST attacks (except Safari for Mac OS X 10.8 or earlier, for iOS 6 or earlier, and for Windows. As a result, RC4 is not the best choice for TLS 1.0 anymore. The CBC ciphers which were affected by the BEAST attack in the past are becoming a more popular choice for protection.

Microsoft recommends disabling RC4 where possible.

RC4 is an extremely simple and elegant algorithm. The first phase is the key scheduling algorithm (KSA). This algorithm takes an initial array and initializes it to values 0 to 255. For each index of the array a shuffling occurs that mixes in the key. Once this algorithm runs, the output of the KSA is input to the pseudo-random generation algorithm (PRGA) that continually shuffles the array. The output of the PRGA is exclusive-ored with the plaintext to produce a cipher text.

These recent attacks have found strong biases in the first 257 bytes of encryption which will allow recovery of roughly the first 200 bytes of plaintext in approximately 2 28 to 2 32 encryptions of the same plaintext under unique keys (referred to here as the broadcast attack). The attack recovers the plaintext at each position by gathering the set of observed ciphertexts (each encrypted with a different key) for the corresponding position. It tries each of the 256 candidate plaintexts and computes the PRGA output byte by exclusive-oring the candidate with each ciphertext. It then calculates which plaintext candidate resulted in PRGA outputs which most closely matches the known PRGA bias for that position. To take an extremely simplified example, consider when the PRGA output always outputs the value one. The correct plaintext will be the one which, when exclusive-ored with each of the ciphertexts, always results in one [34].

Additionally, using previously discovered long-term biases in RC4, 50% of a 16-byte secret value can be extracted after analysis of $6 * 2^{30}$ encryptions of the same plaintext message using a single key. This attack is different to the broadcast attack since it can recover plaintext from a one or more encryption streams in which the same plaintext is sent repeatedly. The attack recovers plaintext using transitional biases in the PRGA stream that recur at fixed positions in the stream. For example, if the PRGA output is zero at any offset that is a multiple of 256-bytes, then the next output is more likely to also be a zero. The attack works by starting with a known plaintext byte that repeats at a fixed

position and finding a candidate for the plaintext byte that repeats at the next position. The candidate that most closely matches the transition bias of the PRGA is selected. This process is then repeated to find the next unknown plaintext byte.

In both attacks the request structure such as the URL, which may be known or the location and beginning of the cookie as well as the plaintext structure, such as the language or HTML can be used to further optimize the attack.

2.9 TLS Attacks in practice

Examples of attacks:

Attacks on the Handshake Protocol

1. *Cipher suite rollback*
   The cipher-suite rollback attack presented by Wagner and Schneier aims at limiting the offered cipher-suite list provided by the client to weaker ones or NULL-ciphers. An Man-in-the-middle (Mitm) attacker may alter the ClientHello message sent by the initiator of the connection, strips of the undesirable cipher-suites or completely replaces the cipher-suite list with a weak one and passes the manipulated message to the desired recipient.
2. *ChangeCipherSpec message drop*
   This simple but effective attack described by Wagner and Schneier was feasible in SSL 2.0 only. During the handshake phase the cryptographic primitives and algorithms are determined. For activation of the new state it is necessary for both parties to send a ChangeCipherSpec message. This message informs the other party that the following communication will be secured by the previously agreed parameters. The pending state is activated immediately after the ChangeCipherSpec message is received.
   An attacker located as Mitm could simply drop the ChangeCipherSpec messages and cause both parties to never activate the pending states.
3. *Key exchange algorithm confusion*
   Another flaw pointed out by Wagner, Schneier is related to a feature concerning temporary key material. SSL 3.0 supports the use of temporary key material during the handshake phase (RSA public keys or DH public parameters) signed with a long term key. A problem arises from a missing type definition of the transferred material. Each party implicitly decides, based on the context, which key material is expected and decodes accordingly. More precise, there is no information on the type of the encoded key material. This creates a surface for a type confusion attack.
4. *Version rollback*
   Wagner and Schneier described an attack where a ClientHello message of SSL 3.0 is modified to look like aClientHello message of SSL 2.0. This would force a server to switch back to the more vulnerable SSL 2.0.
5. *Bleichenbacher Attack on PKCS#1*
   In 1998 Daniel Bleichenbacher presented an attack on RSA based SSL cipher suites. Bleichenbacher utilized the strict structure of the PKCS#1 v1.5 format and showed that it is possible to decrypt thePreMasterSecret in an reasonable amount of time. The PreMasterSecret in a RSA based cipher suite is a random value generated by the client and sent (encrypted and PKCS #1 formatted) within the ClientKeyExchange. An attacker eavesdropping this (encrypted) message can decrypt it later on by abusing the server as a decryption oracle.
6. *Timing based attacks*
   Brumley and Boneh outlined a timing attack on RSA based SSL/TLS. The attack extracts the private key from a target server by observing the timing differences between sending a specially crafted ClientKeyExchangemessage and receiving an Alert message inducing an invalid formatted PreMasterSecret. Even a relatively small difference in time allows to draw conclusions on the used RSA parameters.
   Brumley's and Boneh's attack is only applicable in case of RSA based cipher-suites. Additionally, the attack requires the presence of a high-resolution clock on

the attacker's side.

7. *Improvements on Bleichenbacher's attack*
   The researchers Klíma, Pokorny and Rosa not only improved Bleichenbacher's attack, but were able to defeat a countermeasure against Bleichenbacher's attack.

   Breaking the countermeasure A countermeasure against Bleichenbacher's attack is to generate a randomPreMasterSecret in any kind of failure and continue with the handshake until the verification and decryption of the Finished message fails due to different key material (the PreMasterSecret differs at client and server side). Additionally, the implementations are encouraged to send no distinguishable error messages. This countermeasure is regarded as best-practice. Moreover, because of a different countermeasure concerning version rollback attacks the encrypted data includes not only the PreMasterSecret, but also the major and minor version number of the negotiated SSL/TLS version.

3. CONCLUSIONS

We presented our analysis for SSL/TLS attacks. We found serious logic flaws in advanced attacks mechanisms. We discussed the weaknesses and ways of its protection.

SSL/TSL has been around for many years without any major modifications. This protocol was considered to be secure. The CRIME, BREACH,BEAST, Hartbleed attacks proved that in one very specific use case it can be compromised. While this use case can be avoided and SSL/TSL re-secured, will this have an effect on the thoughts of SSL/TSL security as a whole. People tend to lose faith in security protocols as soon as the simplest attack is successful. Will this be the end to SSL/TSL, or will users still have faith in the non-compressed version, that has yet to be broken, or will they run to a new protocol to be positive that they are secure? This will only be answered in time.

We believe that our study takes some steps in the security problem space that SSL protocols have brought. Also our study suggests and analyses Heartbleed exploit detection. We believe that our study brings some new chain of trust between the client and the protocol security. In future work we are considering the security challenges that come with other advanced SSL attacks. Fundamentally, we believe that vulnerabilities of SSL/TSL demands new research efforts on ensuring the security quality of the protocols.

4. REFERENCES

[1] “The Secure Sockets Layer Protocol”. Internet: http://www.cs.bris.ac.uk/~bradley/publish/SSLP/chapter4.html [Nov. 22, 2013].

[2] “SSL: Intercepted today, decrypted tomorrow”. Netcraft, pp. 10-12, May 25, 2013.

[3] “SSL/TLS in Detail“. Microsoft TechNet, July 31, 2003.

[4] “Description of the Secure Sockets Layer (SSL) Handshake“. Internet: http://www.support.microsoft.com [Dec. 1, 2013].

[5]Secure electronic transaction Internet: http://en.wikipedia.org/wiki/Secure_Electronic_Transaction [Dec. 12, 2013].

[6] OpenSSL TLS/DTLS Heartbeat Read Overrun Vulnerability. Internet: http://herjavecgroup.com/admin/pdf/THG_TAB_Heartbleed.pdf [May, 2014].

[7] “The HeartbleedCodenomicon, Bug”. Internet: http://heartbleed.com/ [Apr, 2014].

[8] “April 2014 Web Server NetcraftSurvey”..April 2 204. Internet:

http://news.netcraft.com/archives/2014/04/02/april-2014-web-server-survey.html [Apr,2014].

[9] OpenSSL Security Advisory OpenSSL.Internet: https://www.openssl.org/news/secadv_20140407.txt [Apr, 2014].

[10] "Wild at Heart: Were Intelligence Agencies Using Heartbleed in Electronic Frontier Foundation. Internet: https://www.eff.org/deeplinks/2014/04/wild-heart were- intelligence-agencies-using-heartbleed-november-2013 [Apr, 2014].

[11] "Daily Ruleset Update Summary". Emerging ThreatsInternet: Snort Rul http://www.emergingthreats.net/2014/04/09/daily-ruleset-update-summary-04092014/ [Apr, 2014].

[12] "Detecting SSLOpenHeartbleed with Suricata"Inliniac. . Internet: http://blog.inliniac.net/2014/04/08/detecting-openssl-heartbleed-with-suricata/ [Apr, 2014].

[13] Detect Exploit openSSL Heartbleed vulnerability using Nmap and Metasploit on Kali Linux". Internet:http://www.blackmoreops.com/2014/05/03/detect-exploit-openssl-heartbleed-vulnerability-using-nmap-metasploit-kali-linux/ [June, 2014].

[14] Heart attack: detecting heartbleed exploits in real-time". Internet: http://www.tripwire.com/state-of-security/incident-detection/heart-attack-detect-heartbleed-exploits-in-real-time-with-active-defense/ [June, 2014].

[15] How to Detect a Prior Heartbleed Exploit". Internet: http://www.riverbed.com/blogs/Retroactively-detecting-a-prior-Heartbleed-exploitation-from-stored-packets-using-a-BPF-expression.html [June, 2014].

[16] Ivan Ristić.SSL/TLS,, Deployment Best Practices". Internet: https://www.ssllabs.com/downloads/SSL_TLS_Deployment_Best_Practices_1.3.pdf [June, 2014].

[17] Nadhem AlFardan, Dan Bernstein, Kenny Paterson, Bertram Poettering, Jacob Schuldt.,,On the Security of RC4 in TLS". Internet: http://www.isg.rhul.ac.uk/tls/ [June,

[18] Lars Nybom, Alexander Wall. SSL/TLS,, and MITM attacks". Internet: http://www.it.uu.se/edu/course/homepage/distrinfo/ht09/presentations/Group7.pdf [June, 2014].

[19] Pratik Guha Sarkar. Attacks,, on ssl a comprehensive study of beast, crime, time, breach, lucky 13 & rc4 biases". Internet: https://www.isecpartners.com/media/106031/ssl_attacks_survey.pdf [June, 2014].

[20] ,,List of browsers support for different TLS version". Internet: https://en.wikipedia.org/wiki/Transport_Layer_Security#Web_browsers [June, 2014].

[21] Hong lei Zhang. ,,Three attacks in SSL protocol and their solutions". Internet: https://www.cs.auckland.ac.nz/courses/compsci725s2c/archive/termpapers/725zhang.pdf [June, 2014].

[22] ,,Vulnerability Summary for CVE-2012-4929". Internet: http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2012-4929 [June, 2014].

[23] Software >> sslstrip“. Internet:http://www.thoughtcrime.org/software/sslstrip/ [June, 2014].

[24] SSL, GONE IN 30 SECONDS“. Internet:http://breachattack.com/ [June, 2014].

[25] Christopher Meyer. SoK:„ Lessons Learned From SSL/TLS Attacks“. Internet: http://www.nds.rub.de/media/nds/veroeffentlichungen/2013/08/19/paper.pdf[June, 2014].

[26] Scott C. Johnson. CRIME„ Attack on SSL/TSL“. Internet: http://www.cs.rit.edu/~sxj4236/crypto2_paper2.pdf [June, 2014].

[27] Be'ery, Tal and Amichai Shulman. "TIME Prefect CRIME." Internet: https://media.blackhat.com/eu-13/briefings/Beery/bh-eu-13-a-perfect-crime-beery-wp.pdf [June, 2014].

[28] Constantin Lucian. „Researchers resurrect and improve CRIME attack against SSL“. Internet: http://www.networkworld.com/news/2013/031413-researchers-resurrect-and- improve-crime-267698.html?page=1 [June, 2014].

[29] zoompf. Explaining the Crime weakness in SPDY and SSL“. Internet: http://zoompf.com/2012/09/explaining-the-crime-weakness-in-spdy-and-ssl [June, 2014].

[30] Vlastimil Klíma. „Attacking RSA-based Sessions in SSL/TLS“. Internet: http://eprint.iacr.org/2003/052.pdf [June, 2014].

[31] Canvel, B.“Password Interception in a SSL/TLS Channel“net:. Inte http://lasecwww.epfl.ch/memo_ssl.shtml [February, 2003].

[32] Jonsson J. „On the Security of RSA Encryption in TLS“.n ProcI. of CRYPTO '02, pp. 127 -142, 2002.

[34] Kurt Seifried. As„ with marriage, SSL security success is in the details Attacks against SSL“. Internet:http://www.linux-magazine.com/Issues/2010/112/Security-Lessons-Secure-Programming [June, 2014].

[35] Dan Goodin. Two„ new attacks on SSL decrypt authentication cookies“. Internet: http://arstechnica.com/security/2013/03/new-attacks-on-ssl-decrypt-authentication-cookies/ [June, 2014].