International Journal of Applied Engineering Research
ISSN 0973-4562 Volume 10, Number 4 (2015) pp. 9947-9956

http://www.ripublication.com

SSL-Heartbleed Bug

MythiliBoopathi

Vellore Institute of Technology, Vellore

nmythili@vit.ac.in

Mohit Panwar

Vellore Institute of Technology, Vellore

Mohitpanwar10@yahoo.com

Mamatva Goel

Vellore Institute of Technology, Vellore
Mamatva@Yahoo.com

Abstract

This paper describes the Heartbeat Extension for the Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) protocols and heatbleed vulnerability in later versions of OpenSSL(Secure Socket Layer).

Keywords: TLS, HeartBeat Extension, HeartBleed, openSSL, Buffer Overflow, Security.

Introduction

The Secure Sockets Layer (SSL) is a specification used to protect sensitive data that passes from computer to computer on the internet. An internet user actually uses this technology all the time. When using google.com, one can use http:\\google.com, or https:\\google.com. The https:\\ signifies that the communication between client and server is secure and no one else can see what is being sent back and forth (another way one can tell that a site is using it is by seeing a lock icon right beside the url of the website you visit). This specification doesn't have a problem but a mistake has

been found in an implementation. Open SSL's implementation is actually open source which means source code is available for everyone in the world to read. This bug actually affects any web server that uses this Open SSL.

The primary goal of the TLS Protocol is to provide privacy and data integrity between two communicating applications. The protocol is composed of two layers: the TLS Record Protocol and the TLS Handshake Protocol. TLS is based on reliable protocols, but there is not necessarily a feature available to keep the connection alive without continuous data transfer.

The Heartbeat Extension provides a new protocol for TLS/DTLS allowing to keep alive the connection without performing a renegotiation.

Heartbeat Extension

The user can use the new HeartbeatRequest message, which has to be answered by the peer with a HeartbeartResponse immediately.

The support of Heartbeats is indicated with Hello Extensions. A peer cannot only indicate that its implementation supports Heartbeats, it can also choose whether it is willing to receive HeartbeatRequest messages and respond with HeartbeatResponse messages or only willing to send Heartbeat Request messages. The former is indicated by using peer_allowed_to_send as the HeartbeatMode; the latter is indicated by using peer_not_allowed_to_send as the Heartbeat mode. This decision can be changed with every renegotiation. Heartbeat Request messages must not be sent to a peer indicating peer_not_allowed_to_send. If an endpoint that has indicated peer_not_allowed_to_ send receives a Heartbeat Request message, the endpoint should drop the message silently and may send an unexpected_message Alert message.

The format of the Heartbeat Hello Extension is defined by:

```
enum {
peer_allowed_to_send,
peer_not_allowed_to_send,
} HeartbeatMode;
struct {
HeartbeatMode mode;
} HeartbeatExtension;
```

Upon reception of an unknown mode, an error Alert message using illegal_parameter as its AlertDescription must be sent in response.

Heartbeat Protocol

The Heartbeat protocol is a new protocol running on top of the Record Layer. The protocol itself consists of two message types:

HeartbeatRequest and HeartbeatResponse.

enum {

heartbeat_request,

heartbeat_response,

} HeartbeatMessageType;

A Heartbeat Request message can arrive almost at any time during the lifetime of a connection. Whenever a Heartbeat Request message is received, it should be answered with a corresponding HeartbeatResponse message.

However, a HeartbeatRequest message should not be sent during handshakes. If a handshake is initiated while a HeartbeatRequest is still in flight, the sending peer must stop the TLS retransmission timer for it. The receiving peer should discard the message silently, if it arrives during the handshake. In case of DTLS, HeartbeatRequest messages from older epochs should be discarded.

There must not be more than one HeartbeatRequest message in flight at a time. A Heartbeat Request message is considered to be in flight until the corresponding HeartbeatResponse message is received, or until the TLS retransmit timer expires.

When using an unreliable transport protocol like the Datagram Congestion Control Protocol (DCCP) or UDP, HeartbeatRequest messages must be retransmitted using the simple timeout and retransmission scheme DTLS uses for flights. In particular, after a number of retransmissions without receiving a corresponding HeartbeatResponse message having the expected payload, the DTLS connection should be terminated. The threshold used for this should be the same as for DTLS handshake messages. Please note that after the timer supervising a HeartbeatRequest messages expires, this message is no longer considered in flight. Therefore, the Heartbeat Request message is eligible for retransmission. The retransmission scheme, in combination with the restriction that only one HeartbeatRequest is allowed to be in flight, ensures that congestion control is handled appropriately in case of the transport protocol not providing one, like in the case of DTLS over UDP.

When using a reliable transport protocol like the Stream Control Transmission Protocol (SCTP) or TCP, HeartbeatRequest messages only need to be sent once. The transport layer will handle retransmissions. If no corresponding HeartbeatResponse message has been received after some amount of time, the DTLS/TLS connection may be terminated by the application that initiated the sending of the HeartbeatRequest message.

Heartbeat Request and Response Messages

The Heartbeat protocol messages consist of their type and an arbitrary payload and padding.

struct {

HeartbeatMessageType type;

uint16 payload_length;

opaque payload[HeartbeatMessage.payload_length];

opaque padding[padding_length];

} HeartbeatMessage;

The total length of a HeartbeatMessage must not exceed 2^14 or max_fragment_ length when negotiated.

Type: The message type, either heartbeat_request or

heartbeat_response.

Payload_length: The length of the payload.

Payload: The payload consists of arbitrary content.

Padding: The padding is random content that must be ignored by the receiver. The length of a HeartbeatMessage is TLSPlaintext.length for TLS and DTLSPlaintext.length for DTLS. Furthermore, the length of the type field is 1 byte, and the length of the payload_length is 2. Therefore, the padding_length is TLSPlaintext.length - payload_length - 3 for TLS and DTLSPlaintext.length - payload_length - 3 for DTLS. The padding_length MUST be at least 16.

The sender of a HeartbeatMessage MUST use a random padding of at least 16 bytes. The padding of a received HeartbeatMessage message MUST be ignored.

If the payload_length of a received HeartbeatMessage is too large, the received HeartbeatMessage must be discarded silently.

When a HeartbeatRequest message is received and sending a

HeartbeatResponse is not prohibited as described elsewhere in this document, the receiver must send a corresponding HeartbeatResponse message carrying an exact copy of the payload of the received HeartbeatRequest.

If a received HeartbeatResponse message does not contain the expected payload, the message must be discarded silently. If it does contain the expected payload, the retransmission timer must be stopped.

Use Case

Each endpoint sends HeartbeatRequest messages at a rate and with the padding required for the particular use case. The endpoint should not expect its peer to send HeartbeatRequests. The directions are independent.

Liveliness Check: Sending HeartbeatRequest messages allows the sender to make sure that it can reach the peer and the peer is alive. Even in the case of TLS/TCP, this allows a check at a much higher rate than the TCP keep-alive feature would allow. Besides making sure that the peer is still reachable, sending HeartbeatRequest messages refresh the NAT state of all involved NATs.

HeartbeatRequest messages should only be sent after an idle period that is at least multiple round-trip times long. This idle period should be configurable up to a period of multiple minutes and down to a period of one second. A default value for the idle period should be configurable, but it should also be tuneable on a per-peer basis.

Heartbleed Bug

The type of bug that Heart Bleed is categorized as is called a buffer overflow, where more data is read from memory then what should have been. Buffer overflow bugs are actually pretty common and have been around for more than 25 years. Software companies are constantly fixing buffer overflow bugs and they can be pretty easy to miss when reviewing code.

Example how the Heartbleed bug works:

Client1: Server, are you still there? If so, reply "POTATO" (6 letters). #the server's memory is shown.

Server: POTATO # server shows the same memory content but POTATO is highlighted.

Client1: Server, are you still there? If so, reply "BIRD" (4 letters). #The server's memory is shown.

Server: BIRD # Server shows the same memory content but now with BIRD highlighted.

Client1: Server, are you still there? If so, reply "HAT" (500 letters). # The server's memory is shown.

Server: “HAT. Client2 requests the "missed connections" page. Administrator wants to set server's key to "14835038534". Client3 wants pages about "snakes but not too long". Client4 wants to change account password to ‘CoHoBaSt’. “ # Server shows the same memory content, highlighting the first 500 letters of the memory beginning at HAT.

Client1 writes this all details down.

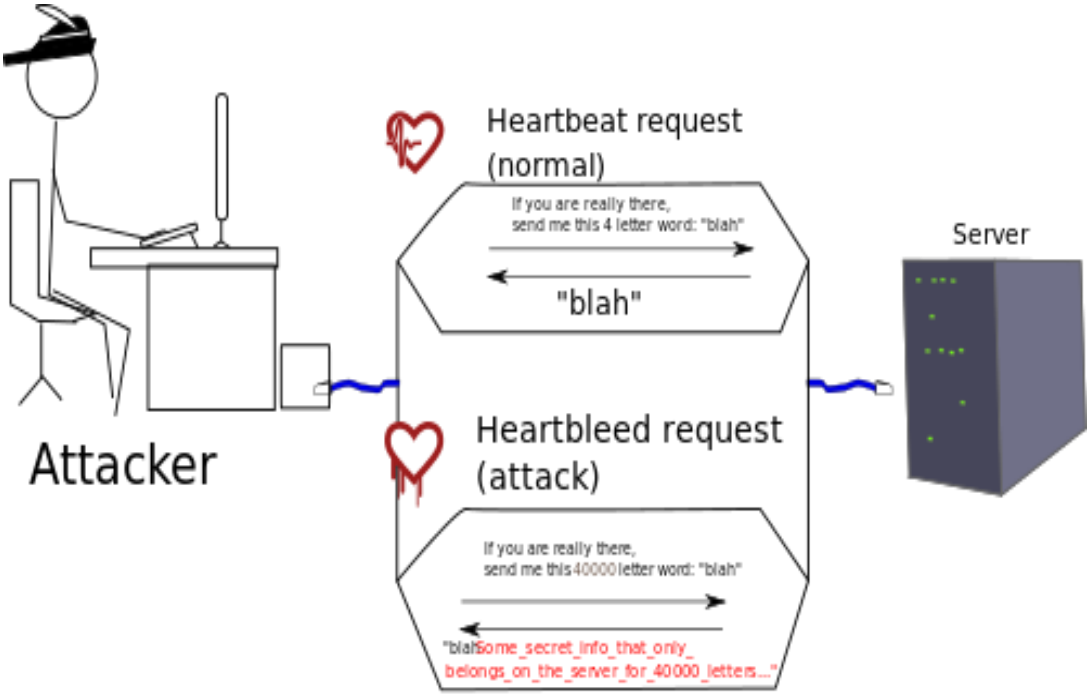

Figure 1: Example demonstration of the attack

The TLS and DTLS implementations in OpenSSL 1.0.1 before 1.0.1g do not properly handle Heartbeat Extension packets, which allows remote attackers to obtain sensitive information from process memory via crafted packets that trigger a buffer over-read, as demonstrated by reading private keys, related to d1_both.c and t1_lib.c, aka the Heartbleed bug.

In order to coordinate recovery from this bug I have classified the compromised secrets to three categories:

1) primary key data- Leaked secret keys allows the attacker to decrypt any past and future traffic to the protected services and to impersonate the service at will. Any protection given by the encryption and the signatures in the X.509 certificates can be bypassed. Recovery from this leak requires patching the vulnerability, revocation of the compromised keys and reissuing and redistributing new keys. Even doing all this will still leave any traffic intercepted by the attacker in the past still vulnerable to decryption. All this has to be done by the owners of the services.
2) secondary key data – This includes the user credentials (user names and passwords) used in the vulnerable services. Recovery from this leaks requires owners of the service first to restore trust to the service according to steps

described above. After this users can start changing their passwords and possible encryption keys according to the instructions from the owners of the services that have been compromised. All session keys and session cookies should be invalidated and considered compromised.

3) protected data . This is the actual content handled by the vulnerable services. It may be personal or financial details, private communication such as emails or instant messages, documents or anything seen worth protecting by encryption. Only owners of the services will be able to estimate the likelihood what has been leaked and they should notify their users accordingly. Most important thing is to restore trust to the primary and secondary key material as described above.

Affected Devices and Softwares

- OpenSSL 1.0.1 through 1.0.1f (inclusive) are vulnerable
- Debian Wheezy (stable), OpenSSL 1.0.1e-2+deb7u4
- Ubuntu 12.04.4 LTS, OpenSSL 1.0.1-4ubuntu5.11
- CentOS 6.5, OpenSSL 1.0.1e-15
- Fedora 18, OpenSSL 1.0.1e-4
- OpenBSD 5.3 (OpenSSL 1.0.1c 10 May 2012) and 5.4 (OpenSSL 1.0.1c 10 May 2012)
- FreeBSD 10.0 - OpenSSL 1.0.1e 11 Feb 2013
- OpenSUSE 12.2 (OpenSSL 1.0.1c)
- Android 4.1.1 (OpenSSL 1.0.1c)
- 3.1.10 Ruby (when compiled with OpenSSL)

Limitations of the Attack

- IDS/IPS can detect the attack but cannot block the attack unless heartbeatrequests are blocked altogether.
- Attacker can only obtain 64k of OpenSSL memory.

Mitigation of The Attack

- Upgrade the OpenSSL version to 1.0.1g
- Request revocation of the current SSL certificate to your software vendor.

- Regenerate the private key.
- Use two-factor authentication
- Recompile OpenSSL package in Ruby with the handshake removed from the code by compile time option -DOPENSSL_NO_HEARTBEATS
- Perfect Forward Secrecy- When used in the memo Perfect Forward Secrecy (PFS) refers to the notion that compromise of a single key will permit access to only data protected by a single key. For PFS to exist the key used to protect transmission of data MUST NOT be used to derive any additional keys, and if the key used to protect transmission of data was derived from some other keying material, that material MUST NOT be used to derive any more keys. We need to disable compression at the HTTP level. In the apache configuration file(httpd.conf) edit the following section
- *<Location />*
- *SetEnvIfExpr "%{HTTPS} == 'on'" no-gzip*
- *</Location>*

Summary

Information Security over Internet is the most vital component in information security because it is responsible for securing all information passed through Internet nodes. SSL is the secure communications protocol of choice for a large part of the Internet community. The Transport Layer Security (TLS) protocol was released to create a standard for private communications. The protocol "allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering or message forgery." The Heartbeat Extension provides a new protocol for TLS/DTLS allowing to keep alive the connection without performing a renegotiation.

In the Heartbeat Extension, the client sends a request to the server comprising of a echo message and the size of the echo message, to keep-alive the connection between the client and the server. Then the server sends a response comprising of the message of length specified in the request from client.

The attacker(authenticated client) can send a request forged with an echo message and size larger than the echo message. The client does not verify the size specified in the request against the message actual size and sends a response with sensitive information in the current memory.

This probably can lead to the leak of the private key using which attacker can decipher all the traffic between the client and the server.

Conclusion

The SSL/TLS handshake protocol has a vulnerability and worrisome feature, especially in versions which have been recently revised. Heartbeat Extension must be irradiated from the OpenSSL or the above mentioned solutions should be obliged for the secure connection to server. Furthermore, these are not universal weaknesses: different implementations may or may not be vulnerable. Nonetheless, if the specification does not explicitly warn of an attack (or prevent it directly), it seems reasonable to over destructive criticism. SSL and TLS are evoked as not effective methods of securing sensitive communications, as the aggregate of larger amounts of information should be properly secured.

References

[1] CVE-2014-0160

[2] "THE SSL PROTOCOL". Netscape Corporation. 2007. Archived from the original on 14 June 1997.

[3] OpenSSL Security Advisory (published 7th of April 2014, ~17:30 UTC)

[4] HEARTBLEED.COM

[5] Yan Zh~Why the Web Needs Perfect Forward Secrecy

[6] https://www.eff.org/deeplinks/2014/04/why-web-needs-perfect-forward-secrecy

[7] https://www.ietf.org/rfc/rfc2246.txt

[8] http://www.ietf.org/rfc/rfc2409.txt

[9] David Wagner~Analysis of the SSL 3.0 protocol

[10] University of California, Berkeley

[11] https://www.schneier.com/paper-ssl-revised.pdf

[12] Demonstration of CVE-2014-0160 by Jared Stafford (jspenguin@jspenguin.org)

[13] A. Freier, P. Karlton, P. Kocher (August 2011). "The Secure Sockets Layer (SSL) Protocol Version 3.0".

[14] [RFC6066].