International Journal of
Emerging Research in Management &Technology
ISSN: 2278-9359 (Volume-5, Issue-12)

Research Article | **December 2016**

Vulnerability Analysis in OpenSSL

Malik Nadeem Anwar, Dr. Harleen Kaur
Department of Computer Science and Engineering, Jamia Hamdard,
Delhi, India

Abstract—

The security on the Internet is a major ordeal. In spite of the way that the greater part of the assaults depend on a wrong setup it might happen that it is not client's flaw. Programming may fizzle. The OpenSSL library, executes a considerable measure of figure suites utilized by numerous applications. The Heartbleed powerlessness surprised the Internet in April 2014. The powerlessness, a standout amongst the most weighty since the appearance of the business Internet, permitted aggressors to remotely read shielded memory from an expected 24–55% of well known HTTPS locales. In any case, analysts have uncovered a genuine defenselessness in this standard Web encryption programming known as "Heartbleed", the bug can give programmers access to individual information like charge card numbers, usernames, passwords, and, maybe in particular, cryptographic keys which can permit programmers to imitate or screen a server.

Keywords— openssl, heartbeat bug, cupid bug

I. INTRODUCTION

SSL is separated into two layers, with every layer utilizing administrations gave by a lower layer and giving usefulness to higher layers. The SSL record layer gives classification, validness, and replay assurance over an association arranged dependable transport convention, for example, TCP.SSL 2.0 had numerous security shortcomings which SSL 3.0 settled. OpenSSL has had a few outstanding security issues amid its 16 year history, this defect—the Heartbleed weakness—was a standout amongst the most impactful. Heartbleed permits aggressors to peruse delicate memory from defenseless servers, conceivably including cryptographic keys, login accreditations, and other private information. Compounding its seriousness, the bug is easy to comprehend and abuse. In this work, we investigate the effect of the powerlessness and track the server administrator group's reactions. Utilizing broad dynamic filtering, we evaluate who was defenseless, portraying Heartbleed's extension crosswise over well known HTTPS sites and the full IPv4 address space. We likewise study the scope of conventions and server items influenced. We evaluate that 24–55% of HTTPS servers in the Alexa Top 1 Million were at first defenseless, including 44 of the Alexa Top 100. Two days after divulgence, we watched that 11% of HTTPS locales in the Alexa Top 1 Million stayed defenseless, as did 6% of all HTTPS servers in general society IPv4 address space. We find that helpless hosts were not haphazardly dispersed, with more than half situated in just 10 ASes that don't mirror the ASes with the most HTTPS hosts. In our outputs of the IPv4 address space, we recognize more than 70 models of helpless installed gadgets and programming bundles. We additionally watch that both SMTP+TLS and Tor were vigorously influenced; more than half of all Tor hubs were helpless in the days taking after exposure.

The SSL convention is proposed to give a down to earth, application-layer, generally appropriate association situated instrument for Internet customer/server correspondences security. This point gives a nitty gritty specialized examination of the SSL convention. Various minor defects in the convention and a few new dynamic assaults on SSL are displayed. Our principle goal is to concentrate the absolute most late security issues found in the surely understood OpenSSL cryptographic programming library. There is a genuine defenselessness of the Heartbeat Extension for the vehicle layer security conventions (TLS/DTLS) usage in OpenSSL. This shortcoming permits remote assailants to acquire delicate data (mystery keys utilized for X.509 declarations, client certifications, texts, messages and other basic records and correspondence) for applications, for example, web, email, texting and some virtual private systems, which ought to be secured, under typical conditions, by the SSL/TLS encryption. Further, "Cupid", demonstrates that the bug can be utilized, with the same impact, against any gadget which suggests Extensible Authentication Protocol (EAP) confirmation instruments and a defenseless form of OpenSSL. What's more rules, fixes and techniques for avoiding and overseeing conceivable assaults against helpless frameworks are displayed.

II. LITERATURE REVIEW

A. Background

These two protocols are cryptographic protocols. Originally the protocol was named SSL (secure socket layer) and the latest version is named TLS (trans- port layer security). This protocol is used by many others such as HTTPS, POP3S, STMPS, SMTPS, IMPAS. It brings to the user of the library several cipher suites. A cipher suite is composed of a key exchange algorithm(for example Diffie Hellman), a bulk encryption algorithm to exchange data once the connection is established(for example AES), a message authentication code(for example SHA) and a pseudo random function to create the session key. It allows authentication thanks to asymmetric cryptography with the usage of certificates, confidentiality with symmetric cryptographic key, and integrity with message authentication code(MAC). To

allow all these properties, a initialization of the connection in required. Afterward both parties are able to ex- change records in a safe way.To establish a reliable connection which granted the security described above, several steps are necessary. Let's illustrate it with a connection between a client which has no certificate and a server which got one assuming a TCP connection has been opened. This is the typical case when a user connect on a web site with HTTPS. First the client sends a Client Hello message with 5 fields : the higher version of SSL/TlS which can be used, a random number, a sessionID, a cipher suite and a compression method. The random number is used as a challenge to authenticate the server and also as unencrypted salt. The session ID is used to change the parameters of an existing connection. The cipher suite contains an ordered list of the cipher suites wished by the client. The compression method contains also an ordered list with the preference methods. This last list can be empty. Data are not always compressed.

B. Overview

The Heartbeat Extension for the Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) conventions was archived and proposed as a standard in mid 2012 by IETF, under heading of R. Seggelmann, M. Tuexen and M. Williams. The Heartbeat Extension gives another convention to TLS/DTLS permitting the utilization of keep-alive usefulness without performing a renegotiation and a premise for way MTU (PMTU) disclosure for DTLS. Essentially the session is kept open notwithstanding when any official information is not traded over the scrambled association. The essential objective of the TLS convention is to give protection and information respectability between two imparting applications, yet there is not as a matter of course an element accessible to keep the association alive without consistent information exchange. The DTLS is essentially intended to secure movement running on top of questionable transport conventions. Typically, such conventions have no session administration. The main component accessible at the DTLS layer to make sense of if an associate is still alive is an exorbitant renegotiation, especially when the application utilizes unidirectional movement [1]. Besides, DTLS requirements to perform way MTU (PMTU) disclosure however has no particular message sort to acknowledge it without influencing the exchange of client [2]. The Heartbeat Extension is intended to conquer these restrictions.

OpenSSL is an open-source usage of SSL and TLS convention [3]. SSL and TLS are transport layer convention which are mostly required in end-to-end security over the Internet. A vehicle layer convention gives end-to-end security administrations for application that utilization solid transport convention, for example, TCP (Transmission Control Protocol) [4]. The Idea is to give security administrations to exchanges on Internet. For instance, when client shops on the web, the accompanying security administrations are coveted:

1. The client should make sure that server (site) has a place with the real merchant, not a faker. The client wouldn't like to give an imposer his/her Mastercard numbers. In like manner, the merchant needs to validate the client.
2. The client and merchant should make sure that the substance of the message are not adjusted amid move (message respectability)
3. The client and merchant should make sure that faker doesn't capture delicate data, for example, Mastercard numbers [5] and to give this kind of security two conventions are overwhelming today are Secure Socket Layer (SSL) Protocol and The Transport Layer Security (TLS). In fact SSL that started things out and TLS is more similar to successor to SSL. TLS is IETF(Internet Engineering Task Force) standard rendition of SSL. For instance when you see letters https://in your web program alongside a lock symbol as appeared in fig 2.1

Fig. 2.1 use of *https://* in google.com

That means the web page your seeing is in encrypted form, in particular that means you are using SSL/TLS protocol to safeguard your information while using that site. Any information transmitting through this website that might be password or any information is in encrypted form and OpenSSL is just an implementation of these protocols. To transmute and receive data from this secure site OpenSSL uses session, in this session user and server is involved. In particular there is an extension to TLS protocol known as "Heartbeat" [6]. And what heartbeat extension is allows you to do is keep TLS session up and running even though no data has gone through in a while by sending special request message known as heartbeat request. So, heartbeat request sends from one computer to another, and basically this request includes some data as "payload" and "size" size of payload as shown in fig 2.2.

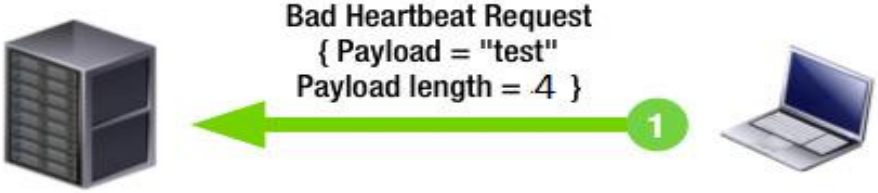

Fig. 2.2 Heartbeat request from client to server.

The Computer that responding to heartbeat request will actually contents the same payload information and also little bit of padding as shown in fig 2.3

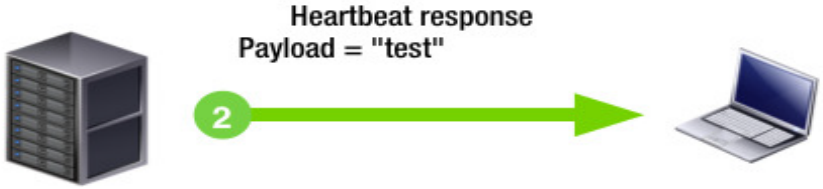

Fig. 2.3 Heartbeat Respond from server to client

Along these lines a safe session amongst client and server keeps alive and secure clients information utilizing OpenSSL. From this segment I expected that you comprehend the essential system of OpenSSL convention and how it functions, now in next segment I will be defy you with real issue i.e.

OpenSSL weakness "Heartbleed" and how can it works. Building up an association : handshake To set up a solid association which allowed the security depicted over, a few stages are essential. We should delineate it with an association between a customer which has no declaration and a server which made them expect a TCP association has been opened. This is the run of the mill situation when a client associate on a site with HTTPS. In the first place the customer sends a Client Hello message with 5 handle: the higher form of SSL/TlS which can be utilized, an arbitrary number, a sessionID, a figure suite and a pressure technique. The irregular number is utilized as a test to validate the server furthermore as decoded salt. The session ID is utilized to change the parameters of a current association. The figure suite contains a requested rundown of the figure suites wished by the customer. The pressure technique contains additionally a requested rundown with the inclination strategies. This last rundown can be vacant. Information are not generally packed.

C. Heartbeat Extension

The Heartbeat Extension for the Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) protocols was documented and proposed as a standard in early 2012 by IETF, under direction of R. Seggelmann, M. Tuexen and M. Williams. The Heartbeat Extension provides a new protocol for TLS/DTLS allowing the usage of keep-alive functionality without performing a renegotiation and a basis for path MTU (PMTU) discovery for DTLS. Basically the session is kept open even when any official data is not exchanged over the encrypted connection.

The primary goal of the TLS protocol is to provide privacy and data integrity between two communicating applications, but there is not necessarily a feature available to keep the connection alive without continuous data transfer. The DTLS is basically designed to secure traffic running on top of unreliable transport protocols. Usually, such protocols have no session management. The only mechanism available at the DTLS layer to figure out if a peer is still alive is a costly renegotiation, particularly when the application uses unidirectional traffic. Furthermore, DTLS needs to perform path MTU (PMTU) discovery but has no specific message type to realize it without affecting the transfer of user. The Heartbeat Extension is designed to overcome these limitations.

D. The protocol

The Heartbeat convention is another convention running on top of the Record Layer. The convention itself comprises of two message sorts:

1. HeartbeatRequest;
2. HeartbeatResponse.

These messages comprise of their sort, payload_length, a discretionary payload and cushioning as found in Figure 2.4

```
struct
{
  HeartbeatMessageType type;
  uint16 payload_length;
  opaque payload[HeartbeatMessage.payload_length];
  opaque padding[padding_length];
} HeartbeatMessage;
```

Figure 2.4 Heartbeat message

The client can utilize the HeartbeatRequest message practically whenever amid the lifetime of an association. The associate ought to answer quickly with a relating HeartbeatResponse message conveying a precise of the subjective payload contained by HeartbeatRequest. As indicated by [3] the most extreme size of the message is limited to 2^14 (16Kbytes) or max_fragment_length. On the off chance that the payload_length of a got pulse message is too vast or if a HeartbeatResponse does not contain the normal payload the message must be disposed of quietly.

The pulse expansion convention comprises of two message sorts: HeartbeatRequest message and HeartbeatResponse message and the augmentation convention relies on upon which TLS convention is being utilized as portray underneath:

- ***When utilizing solid transport convention:***

One side of the companion association sends a HeartbeatRequest message to the next side. The opposite side of the association ought to instantly send a HeartbeatResponse message. This makes one fruitful Heartbeat and consequently, keeping association alive – this is called 'keep-alive' usefulness. On the off chance that no reaction is gotten inside a predefined timeout, the TLS association is ended.

- ***Unreliable transport convention:***

One side of the companion association sends HeartbeatRequest message to the next side. The other side of the association ought to instantly send a HeartbeatResponse message. On the off chance that no reaction is gotten inside determined timeout another HeartbeatRequest message is retransmitted. On the off chance that normal reaction is not got for determined number of retransmissions, the DTLS association is ended. At the point when a recipient gets a HeartbeatRequest message, the collector ought to send back a precise of the got message in the HeartbeatResponse message. The sender checks that the HeartbeatResponse message is same as what was initially sent. In the event that it is same, the association is kept alive. On the off chance that the reaction does not contain the same message, the HeartbeatRequest message is retransmitted for a predetermined number of retransmissions.

E. HeartBeat Implementation in OpenSSL

The OpenSSL group actualized the pulse augmentation in December 2011. This segment quickly clarifies the code for pulse execution for both HeartbeatRequest message and HeartbeatResponse message. It additionally clarifies the bug in the code and its fix in point of interest. The OpenSSL source code can be downloaded from the gatherings site ate https://www.openssl.org/source/or ftp://ftp.openssl.org/source/. The bug exists in OpenSSL from rendition 1.0.1 to 1.0.1f.

```
int
tls1_process_heartbeat(SSL *s)
 {
 unsigned char *p = &s->s3->rrec.data[0], *pl;
 unsigned short hbtype;
 unsigned int payload;
 unsigned int padding = 16; /* Use minimum padding */

 /* Read type and payload length first */
 hbtype = *p++; // p points on the request
       // TYPE(1)|PAYLOAD_LENGHT(2)|PAYLOAD|PADDING(16)
 n2s(p, payload); // macro which read payload_lenght from p
                  // and put in into. p point now on PAYLOAD
 pl = p;

 ... // some stuff we don't care about

 if (hbtype == TLS1_HB_REQUEST)
  {
  unsigned char *buffer, *bp;
  int r;

  /* Allocate memory for the response, size is 1 bytes
   * message type, plus 2 bytes payload length, plus
   * payload, plus padding
   */
  buffer = OPENSSL_malloc(1 + 2 + payload + padding); // (1)
  bp = buffer;

  /* Enter response type, length and copy payload */
  *bp++ = TLS1_HB_RESPONSE;
  s2n(payload, bp);
  memcpy(bp, pl, payload); // (2)
  bp += payload;
  /* Random padding */
  RAND_pseudo_bytes(bp, padding);
  ...
  }
...
}
```

Figure 2.4 vulnerable code

Accordingly the server sends a Server Hello message containing the adaptation on SSL/TLS utilized (the higher, the better obviously), an arbitrary number utilized as decoded salt, the picked figure suit strategy among the figure suite list given by the customer and the pressure technique. The server sends a second parcel containing its endorsement and a Server key Exchange with a Hello Done. The Server key trade can be figured or not depending of the key trade assention. The welcome done implies that the procedure is over on the server side.Then the customer sends a Client Key Exchange containing data to produce the symmetric key. Thereafter the customer will send a Change Cipher Spec message to the server. It implies that now every message will be figured with the symmetric key. Test bundles are sent to check all is well. The server do likewise furthermore send a sessionID to the customer. From now trades are figured in both sides with the symmetric key thus information can be sent securely. The helplessness if present of both sides of the application(client and server). It permits to peruse the memory of the pile of the OpenSSL application. However misusing the bug on the customer side is really futile in light of the fact that to do as such an adulterated server utilizing a stolen declaration. In the event that somebody can set up such a server he essentially can utilize his fake server to take the information he needs. On the server side, the adventure is extremely intriguing on the grounds that each clients who trade information with a server which is helpless can see their information be bargained. That is the reason the abuse of this bug has been made on the server side and not on the customer side.

A heartbeat message has the following form : TYPE(1) PAYLOAD LENGHT(2) PAYLOAD PADDING(16) The numbers represent the number of bytes which are always the same in each fields except in the field payload which represents the information sent in the heartbeat protocol. Here is the code which treat the heartbeat request.

As we can see the receiver trust the sender about the length of the payload. There is no check. The size is saved in the variable payload and then a malloc(1) is done. These new data allocated will be the future response sent by the server. Moreover a memcpy(2) is done with the payload length thus a big part of the memory can be dumped. Actually up to 64 kB (216).To exploit this vulnerability, it's quite easy. An attacker has just to send a heartbeat request with the biggest payload length (64 KB) but with no payload. The corrupted packet is actually 19(1+2+16) bytes length. The server will copy it into the bu_er which is its heartbeat response the request of the attacker (actually the byte for the type of the heartbeat will be changed as an heartbeat response but the rest of the message is the same). But since the heartbeat request of the attacker is only 19 bytes length, the server will also copy around 64kB of the heap of the application into the bu_er. In the heap there are a lot of things such as plain request, that means that there are plain login/password. There are also cookies and particularly session cookies. Theses cookies allow the access of pages which are accessible only with an identification. All these things are really interesting.

F. Extensible Authentication Protocol

Extensible Authentication Protocol, or EAP, is a verification structure oftentimes utilized as a part of remote systems and point-to-point associations . EAP just characterizes message groups (it is not a wire convention) and every convention that utilizations it characterizes an approach to exemplify EAP messages inside that convention's messages. EAP is generally utilized and it gives some transaction of verification capacities called EAP techniques. Five of them are the official confirmation systems for the WPA and WPA2 benchmarks. In our theory we talk about just about those techniques which utilize TLS over EAP to secure some a player in the confirmation procedure [13]: EAP-TLS, EAP-TTLS and EAP-PEAP. EAP-Transport Layer Security (EAP-TLS) is an IETF open standard that uses the Transport Layer Security (TLS) convention [14] and is viewed as a standout amongst the most secure EAP norms accessible. The larger part of executions of EAP-TLS requires customer side X.509 endorsements which expand the security level and guarantees a two component validation system. EAP-Tunneled Transport Layer Security (EAP-TTLS) is an EAP convention that develops TLS [14]. The server is safely verified to the customer and after that the server can utilize the built up secure association with validate the customer. In this convention the username utilized for confirmation is never transmitted decoded. The safe passage gives assurance from listening in and man-in-the-center assault. EAP-Protected Extensible Authentication Protocol (EAP-PEAP) typifies EAP inside a safe TLS burrow, giving offices to security of the EAP discussion. The reason for existing was to right insufficiencies in EAP."The Heartbleed Bug" is a basic weakness found in OpenSSL's execution of the TLS Heartbeat Extension. When it is abused it prompts the break of memory substance amongst server and customer and the other way around.

The Heartbleed weakness starts in the HeartbeatResponse messages, which assume to contain a duplicate of the self-assertive payload from the solicitation. An aggressor sends a HeartbeatRequest message with a little measured subjective payload and a high payload_length which is set to the greatest conceivable size 0xFFFF (65535 bytes).

```
/* Allocate memory for the response, size is 1 bytes
 * message type, plus 2 bytes payload length, plus
 * payload, plus padding
 */
buffer = OPENSSL_malloc(1 + 2 + payload + padding);
bp = buffer;

/* Enter response type, length and copy payload */
*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpy(bp, pl, payload);
```

Figure 2.5 Payload

The softened OpenSSL source code uncovered up Figure 2, call attention to that the HeartbeatResponse is developed as takes after: the code composes the reaction sort to the begin of the support, increases the cradle pointer, compose the 16-bit payload_length to memory, augment the cushion pointer by two bytes and afterward it duplicates the got self-assertive payload into the active payload for the answer.

To embody, we can send a HeartbeatRequest message having the payload_length field set to 0xFFFF (65535 bytes) which is controlled by the aggressor and a self-assertive payload with one byte. The code needs to send back a duplicate of the approaching pulse message, so it designates a cushion sufficiently huge to hold the 64KBytes payload in addition to one byte to store the message sort, two bytes to store the payload length, and some cushioning bytes. So regardless of the fact that the cushion assigned size is greater than the got subjective payload, the memcpy() capacity will read past the end of the information got. That implies OpenSSL keeps running off the end of your information and gather whatever else is by it in memory at the flip side of the association, for a potential information spillage of roughly 64KB every time you send a distorted HeartbeatRequest.

III. EXPERIMENTAL EXPLOITATION

A. The heartbleed exploitation

First of all, we will explain how we can exploit the Heartbleed bug in practice. In order to perform the exploitation, we have created a python script named heartbleed.py (./heartbleed/heartbleed.py). This script opens a new connection with the server speci_ed and send the "Client Hello" message. It waits the answer "Server Hello Done" from the server and then, sends the Heartbeat request 01 FF FF. This request is supposed to ask for a 64kB Heartbeat response but there is not any data behind. This is the Heartbleed request. If the server is vulnerable, it will answer with a Heartbeat response 02 FF FF followed by 64kB data leaked from the memory. This response will be split in five packets due to the maximum length of each packet. The python script will receive it and store it in logs _les in preparation for a posterior processing. Here, some extracts from the memory that you can read with such an attack.

00f0: 01 03 02 03 03 02 01 02 02 02 03 01 01 00 0F 00 .....
0100: 01 01 2C 65 6E 2D 75 73 3B 71 3D 30 2E 35 2C 65 ..,en-us;q=0.5,e
0110: 6E 3B 71 3D 30 2E 33 0A 41 63 63 65 70 74 2D 45 n;q=0.3.Accept-E
0120: 6E 63 6F 64 69 6E 67 3A 20 67 7A 69 70 2C 20 64 ncoding: gzip, d
0130: 65 66 6C 61 74 65 0A 78 52 65 66 65 72 65 72 3A eflate.xReferer:
0140: 20 68 74 74 70 73 3A 2F 2F 61 63 63 6F 75 6E 74 https://account
0150: 73 2E 67 6F 6F 67 6C 65 2E 63 6F 6D 2F 53 65 72 s.google.com/Ser
0160: 76 69 63 65 4C 6F 67 69 6E 3F 73 61 63 75 3D 31 viceLogin?sacu
00f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0100: 21 00 00 00 00 00 00 00 A0 B6 AB 01 00 00 00 00 !.....
0110: 88 37 A9 D9 20 7F 00 00 90 00 00 00 00 00 00 00 .7.. .....
0120: A1 00 00 00 00 00 00 00 61 A1 FA 14 6C 48 EC 7D .....a...lH.}
0130: 5F 40 97 35 0F 5C D8 31 B5 B8 C2 0D 77 D8 BD 2C _@.5.\.1....w..,
0140: 23 C3 9B 56 3B CC 5E AF 3F D6 04 1C 88 9D 77 E6 #..V;.^.?.....w.
0150: AB 2B 0F D2 4D 70 00 BB D3 32 A8 C1 7A C6 84 DB .+..Mp...2..z...
0160: 71 BD CC 3B EB F5 5E 21 89 8C E5 F6 11 C6 F2 11 q..;..^!.....

These leaks are extracted from the RAM dedicated to the OpenSSL process. We can see that some areas of the memory are empty (they have never been allocated). The other areas are composed by superposition of different memory elements. We cannot get back all the process execution because some have been erased. Then, we cannot find every confidential data which could have circulate on the server. Nevertheless, as we will see, among these memory residues, we can get back some very protected data with the attack.

B. Ethical Hacking

Since we have concentrated on the powerlessness, we can perform the moral hacking exercise. Since this assault is moral, we assaulted ourselves from outside. Our objective was a straightforward site with an essential login structure secured by the SSL/TLS encryption (Figure 3). The web application was facilitated on an Apache Server v2.4.7 with mod_ssl introduced and OpenSSL form 1.0.1.e.

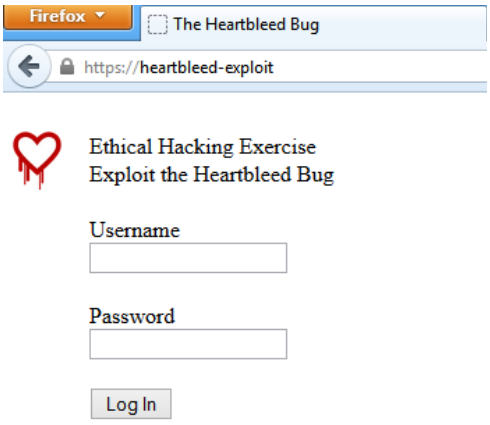

Figure 3.1 Test web application

On the assailant side we utilized a machine with Kali Linux 1.0.5 x64 conveyance introduced and an endeavor script gave by [10] adjusted to our requirements. The endeavor must be keep running against an objective which is connected to a powerless OpenSSL (forms 1.0.1 through 1.0.1f are defenseless). To sum things up, this adventure utilizes OpenSSL to make a scrambled association and trigger the heartbleed spill. The spilled data is returned inside scrambled SSL bundles and is then decoded and kept in touch with a document. The adventure can set pulse payload_length self-assertively and spills up to 65532 bytes of remote memory for every solicitation. The endeavor can be keep running in a circle until the associated peer closes association.

Further, the endeavor can be utilized for plain content correspondence conventions which utilize STARTTLS for overhaul a plain content association with an encoded one, for example, SMTP, IMAP or POP3. Amid the activity we needed to diminish the payload_length asked for size in light of the fact that the customer regularly commandingly shut the association amid vast break demands. In the first place, the objective must have one SSL/TLS port open to have the capacity to introduce the encoded association. We filter our objective for open ports utilizing Nmap instrument (Figure 5.2).

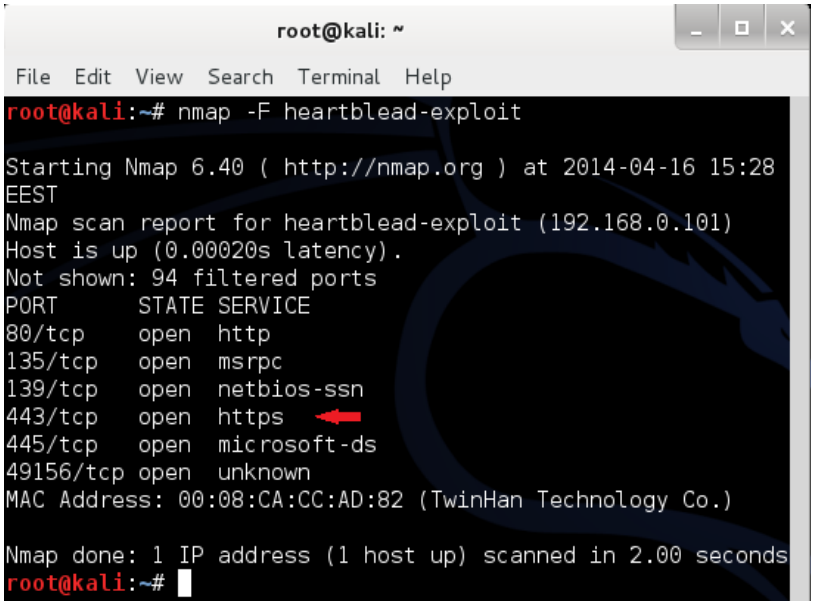

Figure 3.2 Nmap output

Next, we run the heartbleed exploit script sending as input the target IP address, the SSL/TLS port, the padding_lenght and the output file to write data to.

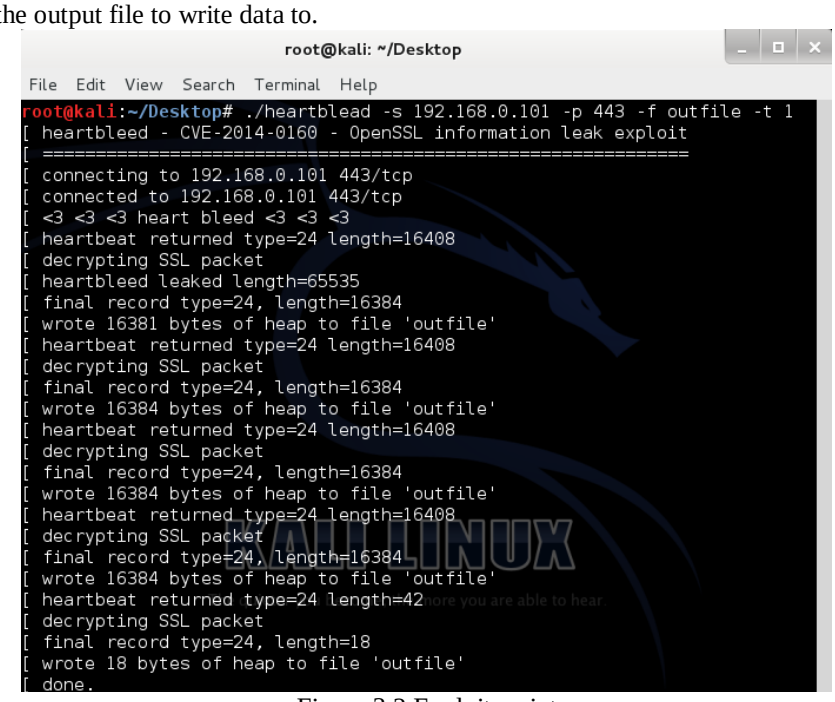

Figure 3.2 Exploit script

The result can be seen in Figure 3.3, where it is exposed the content of the output file generated by the script. We clearly see that we were able to read private data and other critical information from target memory, even if the website uses encryption and should be protected.

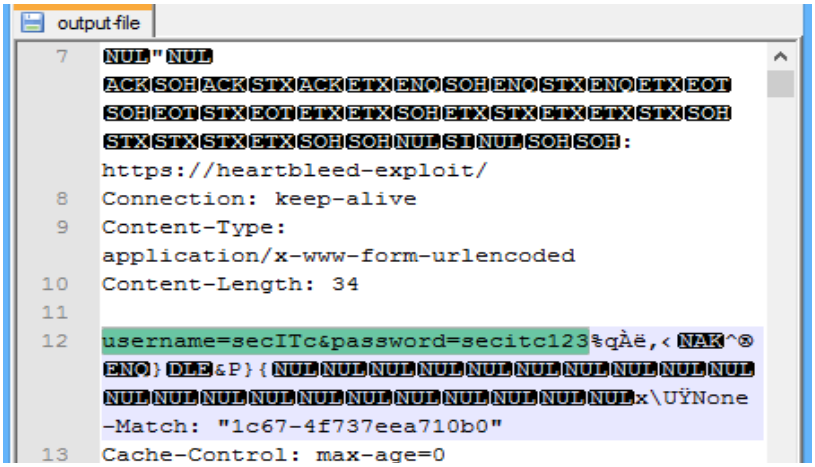

Figure 3.3 Attack results

During our tests we were able to obtain user names, passwords, instant messages, secret keys used for X.509 certificates, critical documents and many other sensitive data. Another important observation is that our attack had not leaved any traces in network traffic or anything abnormal happened to the logs.

C. Cupid

"Cupid", depends on a noxious pulse bundle which is transmitted on standard TLS associations over TCP. Both customers and servers can be abused and memory can be perused off procedures on both closures of the association. The distinction between "The Heartbleed Bug" situation and this one is that the TLS association is being made over EAP. The situation, introduced in employments the fixed adaptations of two Linux based applications to abuse the heartbleed imperfection on TLS associations:

1. "hostapd" is a client space daemon for remote access point and verification servers [15] utilized on Linux, which is competent to make any sort of Wireless Network design and let customers interface with it.
2. "wpa_supplicant" is a free programming execution of IEEE 802.11i (connected as WPA2) standard used to associate with remote systems on Linux (counting Android portable working framework).

Initial one it is utilized to misuse powerless customers by setting up a fake system. At the point when a customer tries to interface and the TLS association is introduced, the "hostapd" will send malignant pulse demands, setting off the heartbleed helplessness. At the point when the customer gets the solicitation, it naturally sends back a reaction which contains information from target memory.

For the second application, the objective is a powerless server (system). So when we ask for another association and send a pulse ask for directly after the TLS association is made, the assault begins. Considering the EAP validation system, an aggressor will just need a substantial username to perform an association endeavor and send distorted pulse demands. Another essential perspective is that a full TLS association is not required, in light of the fact that the pulse solicitation can be sent and got before keys and testaments are traded.

Utilizing a system activity analyzer device we can discover bundles with the pulse conduct and catch the 64KB pulse reaction spilled from memory (Figure 3.4).

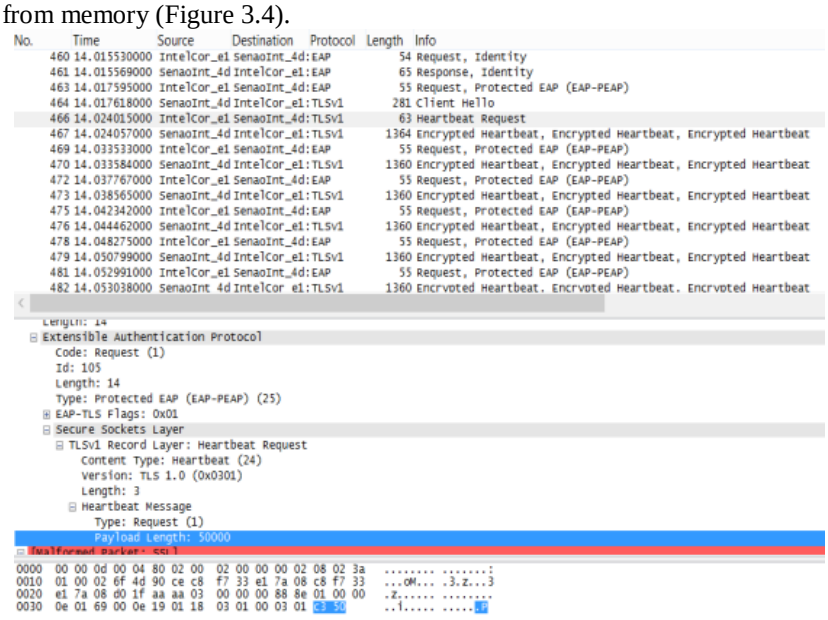

Figure 3.4 heartbeat response leaked from memory

The results suggest that you could be able to obtain the private keys used for TLS connection and the credentials used for connection authentication .

This security problem affects some version of Linux based operating systems which have vulnerable versions of OpenSSL (including Ubuntu and Android) and some wireless solutions (especially those used by corporations or large organizations) which uses EAP authentication mechanisms. Unlike heartbleed, which is best known for giving end users the ability to collect data out of vulnerable servers, "Cupid", shows that the bug can be used, with the same effect, against any device running a vulnerable version of OpenSSL.

IV. RESULTS

The results suggest that you could be able to obtain the private keys used for TLS connection and the credentials used for connection authentication. This security problem affects some version of Linux based operating systems which have vulnerable versions of OpenSSL (including Ubuntu and Android) and some wireless solutions (especially those used by corporations or large organizations) which uses EAP authentication mechanisms. Unlike heartbleed, which is best known for giving end users the ability to collect data out of vulnerable servers, "Cupid", shows that the bug can be used, with the same effect, against any device running a vulnerable version of OpenSSL.

V. CONCLUSIONS

Fixes and Guidelines

First off all we must highlight that this vulnerability in not a design flaw in SSL/TLS protocol specification. This is a programming mistake in OpenSSL library.

Fixes

The bug was introduced to OpenSSL in December 2011 and has been out in the wild since OpenSSL release 1.0.1 on 14th of March 2012. The vulnerable versions of the OpenSSL affected are 1.0.1 through 1.0.1f (inclusive). For fixing this bug you should upgrade to OpenSSL version 1.0.1g or newer. If this is not possible software developers can alternatively recompile OpenSSL with the option -DOPENSSL_NO_HEARTBEATS . Besides, any protection given by the encryption and the signatures in the X.509 certificates can be bypassed if the secret keys were leaked. This allows the attacker to decrypt any past and future traffic to the protected services and to impersonate the service at will. The owners of services must revoke the compromised keys and reissues and redistributes new keys. After this operation is complete, regular users should start changing their credentials (especially passwords) and any possible compromised encryption keys. All session keys and session cookies should be invalidated and considered compromised.

Guidelines

If we look at the bright side of this security flaw, it was a good opportunity for all service providers to upgrade their security strength of the secret keys used and to increase the security measurements. The vulnerability presented in this thesis is based on a simple programming error, and the chance that another similar flaw occurs in the future should not be ruled out. Although there can be no universal security measures to combat any possible security incident, a few basic rules for information security can be followed.

Taking into account that "the Heartbleed Bug" is actually a buffer overflow problem we present some protection measures to defend against it. A programmer should always design a program with security in mind, test and debug the code for errors, do the manual auditing of code, prevent use of dangerous functions like memcpy, malloc, strcpy etc. and prevent all sensitive information from being overwritten. One of the main reasons of security breaches is the human factor mistake. The proper implementation of a strong security policy is the most important measure that must be taken to reduce the risk of security flaws.

The version of this template is V2. Most of the formatting instructions in this document have been compiled by Causal Productions from the IEEE LaTeX style files. Causal Productions offers both A4 templates and US Letter templates for LaTeX and Microsoft Word. The LaTeX templates depend on the official IEEEtran.cls and IEEEtran.bst files, whereas the Microsoft Word templates are self-contained. Causal Productions has used its best efforts to ensure that the templates have the same appearance.

REFERENCES

[1] EC-Council, Ethical Hacking and Countermeasures CEH v8, 2013
[2] Codenomicon Ltd., The Heartbleed Bug, www.heartbleed.com, April 2014 CVE-2014-0160, cve.mitre.org, 2014
[3] OpenSSL Security Advisory, www.openssl.org/news/secadv_20140407.txt,Aprilm 2014
[4] Seggelmann R., Tuexen M., Williams M., RFC6520 - Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension, IETF, tools.ietf.org/html/rfc6520, February 2012
[5] Dierks T., Rascorla E., RFC5246 – The Transport Layer Security (TLS) Protocol Version 1.2, IETF, tools.ietf.org/html/rfc5246, August 2008
[6] Rescorla E., Modadugu N., RFC6347 – Datagram Transport Layer Security Version 1.2, IETF, tools.ietf.org/html/rfc6347, January 2012
[7] Williams C., Anatomy of OpenSSL's Heartbleed: Just for bytes trigger horror bug,www.theregister.co.uk, April 2014

[8] Ducklin P., Anatomy of a data leakage bug – the OpenSSL "heartbleed" buffer overflow, nakedsecurity.sophos.com, April 2014
[9] Heartbleed OpenSSL – Information Leak Exploit, www.exploit-db.com/exploits/32791/, April 2014
[10] RFC2246 : The TLS protocol http://www.ietf. org/rfc/rfc2246.txt
[11] RFC6520 : Transport Layer Security(TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension http://www.ietf.org/rfc/rfc6520.txt
[12] TrustInSoft : no more heartbleed http://trust-in-soft.com/no-more-heartbleed
[13] SSL CRL Activity https://isc.sans.edu/ crls.html
[14] The Heartbleed Aftermath http://blog.cloudflare.com/the-heartbleed-aftermath- all-cloudflare-certificates-revoked-and-reissued
[15] RFC2437 : RSA Cryptography Specifications http://www.ietf.org/rfc/rfc2437.txt
[16] RSA Cryptosystem http://en.wikipedia.org/ wiki/RSA_%28cryptosystem%29
[17] Factoring problem http://en.wikipedia.org/ wiki/Factoring_problem
[18] Montgomery reduction http://en.wikipedia. org/wiki/Montgomery_reduction
[19] Chinese remainder algorithm http://en. wikipedia.org/wiki/RSA_%28cryptosystem%29#Using_the_Chinese_remainder_algorithm
[20] Secure storage of BIGNUM's http://marc. info/?l=openssl-dev&m=139800505608917&w= 2
[21] Cloudare Challenge http://blog. cloudflare.com/the-results-of-the- cloudflare-challenge
[22] John Viega, Matt Messier, Pravir Chandra, "Network Security with OpenSSL: Cryptography for Secure Communications" O'Really Medi Inc., First Edition, pp. 21-22, 2002.
[23] Behrouz A Forouzan, "Data Communication and Networking", The McGraw-Hills Companies, Fourth Edition, pp. 1008-1014
[24] "Growth of Internet ", http://www.socialmarketingforum.net/2012/09/the-Internetexplosive- growth-and-changes/, 2013
[25] "Heartbleed affected Sites", http://visual.ly/major-sitesaffected- heartbleed, 2014
[26] "Definition", http://en.wikipedia.org/wiki/OpenSSL, 2014
[27] "Heartbleed intro", http://heartbleed.com/, 2014. 1999.
[28] J. Padhye, V. Firoiu, and D. Towsley, "A stochastic model of TCP Reno congestion avoidance and control," Univ. of Massachusetts, Amherst, MA, CMPSCI Tech. Rep. 99-02, 1999.
[29] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification*, IEEE Std. 802.11, 1997.

