International Journal of Enhanced Research in Science, Technology & Engineering
ISSN: 2319-7463, Vol. 6 Issue 12, December-2017, Impact Factor: 4.059

Heartbleed: Lessons to be learnt

Salman Muther Hussain

Department of Information, Deakin University

ABSTRACT

Heartbleed, a bug that had taken whole of the internet community by surprise because of various reasons. This paper focuses upon discussing what heartbleed is, moving further we would learn about the current researches that have been done on heartbleed. This paper tries to explain the way in which this vulnerability was exploited by giving practical examples of attacks along with a detailed anatomy of the defective code. In this paper I try to emphasise the importance of proper testing methodologies and propose a solution which involves an algorithm for detection and prevention of such bugs.

INTRODUCTION

Heartbleed a one of its kind vulnerability in the Open SSL library, this bug was present from the very implementation of the open SSL and TLS and was being exploited since then. It was found only until recent years when an employee of google Mr Neel Mehta found the vulnerability and patched it (Marco et al; heartbleed et al). The irony about this bug is that it was present from the very start of the implementation and was very simple to find, but the damage it caused or has been expected to cause is very severe, as by exploiting this vulnerability the attacker would have been able to retrieve public and private keys, along with the passwords of the users and also their private data (heartbleed et al).

The Name and its meaning:

Heartbleed bug is a bug that arises due to buffer over read and the wrong implementation of the heartbeat extension of the TLS/SSL, wherein the validation at server side grants access to more data than should actually be allowed. The Heartbleed was an implementation error and was patched as soon as the open ssl core team was informed about it (Zakir et al). The new version 1.0.1g was patched and is claimed not be vulnerable to the Heartbleed problem. The bug is named Heartbleed because when the implementation of open SSL's TLS/DTLS which are transport layer security protocols a message named Heartbeat was introduced to keep a check on the connection of the client to the server in transfers using UDP protocol. This extension of heartbeat sends a message to the server containing a message type, length, payload and some random padding, the server replies back with a message type, length, payload and random padding, this bug acts like this to exploit this bug the attacker sends the server with a message whose length is greater than the length of the payload, thereby leading the server to leak some information in the response this information could be of the at maximum of 64kb but is capable enough of revealing important data to the attacker such as the private keys, passwords etc.

The figures 1,2 below show how the heartbeat request and response work

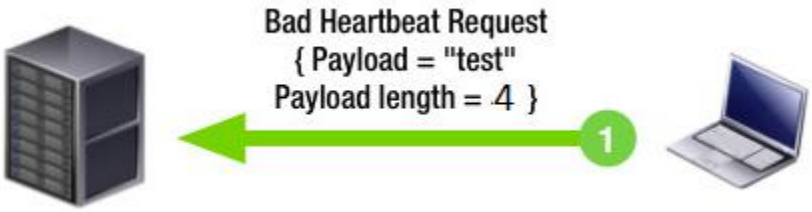

Fig 1: HeartBeat Request (Siddharth Gujrathi 2014)

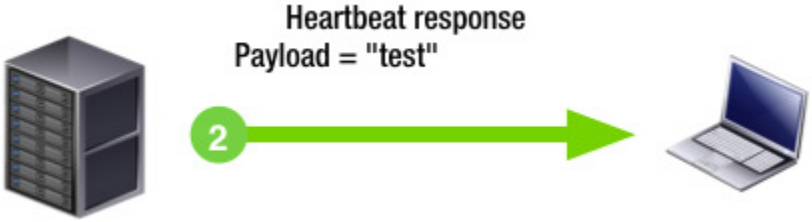

Fig 2: Heartbeat Response (Siddharth Gujrathi 2014)

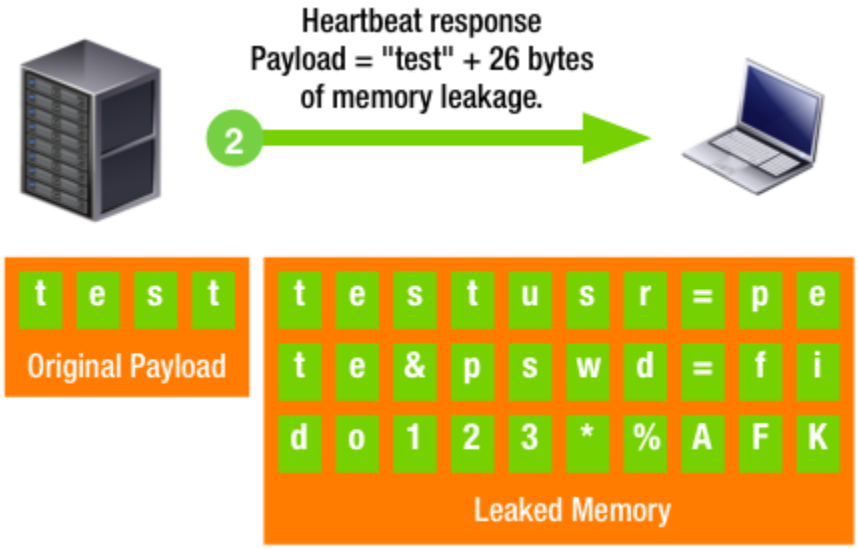

Fig 3: Heartbleed (Siddharth Gujrathi 2014)

Extent of damage Caused:

The question of what is being leaked has a very terrifying answer, in order to co-ordinate recovery from this bug Open SSL has classified the compromised secrets into four categories as described by authors of heartbleed.com:

- Primary key material
- Secondary key material
- Protected content
- Collateral damage

The data leaked under the first category is nothing but the most important one the encryption keys, these keys allow the attacker to be able to decrypt any past or future information and to impersonate the service at will.

The secondary key material leaked involves the leak of user credentials for example usernames and passwords that are used in the vulnerable service by the clients. The protected content leaked includes the data of person or financial importance also data such as emails, communications, documents etc.

Collateral damages are any other data that has been leaked from the server side such as addresses, memory content etc. The widespread effect of this bug can be calculated by taking into account that open web servers such as apache and nginx who share almost more than half of the markets active sites. The more extent of use of open Ssl is that it is used by almost all mal servers, chat servers, vpns etc. Many large organisations are very selective about the implementation of and use SSL/TLS termination equipment and software. But on the other side many small organisations and upcoming organisation use the implementation as it is thereby making them vulnerable. The extent of this bug being exploited is not known till date as it doesn't leave any trace behind (Maduhu et al 2014; Han Vu 2014).

LITERATURE REVIEW

As soon as the news of this bug went public there have been many researchers conducted by universities, private organisations and even members of the cyber space community. As we can see how Marco et al 2014 discusses the problem Heartbleed in brief and try to asses why different testing criteria's which are generally used failed and what could had been done. The authors discuss the technique of static analysis and argue convincingly over why this technique failed and the reasons behind it. The authors also discuss the technique of fuzzing and conclude with giving reasons about how the technique failed. The authors conclude by telling that how advances in the current techniques used and a case by case use of testing methodologies would be effective in properly testing codes. Some critics have gone to the extent of telling that even software should also be put under liability like any other product sold in the market.

This has been proposed by the authors of the paper inviting more Heartbleed where they argue about how to possibly reduce the possibility and extent of damage caused by such vulnerabilities by taking into account all of the stakeholders and distinguishing responsibility of misuse of a software and attribute the responsibility either on the customer or the developer. They also post a point of giving the user the option of disabling certain functions of the software and also holding the company responsible in case of any inbuilt error in the software that would lead to discomfort of the user in any form.

There have been extensive research on how companies responded towards the disclosure of the bug, a study conducted by Zakir et al. 2014 defines in detail the responses of companies. The authors ran tests which would result in whether the sites were still vulnerable or not. They checked whether the source code was patched and whether the certificates were changed. They also did an ipv4 address test to check which IP's were more vulnerable to Heartbleed, apart from this they also conducted network operator checks and also checked major mail servers, android, bitcoin etc. for this vulnerability and documented the results. They found that maximum of the companies had patched their systems in just matter of a day, 44% of them didn't do so but maximum of them did in 48 hours following the disclosure. Certificates of almost all high end website, servers were also revoked and they were not responding to the Heartbleed vulnerability now. However the paper also found out that there was a large number of web services out there who had only patched their source code and didn't really care of changing the certificates, even if they changed the certificates they failed in changing the private still leaving them vulnerable to deciphering of the data.

Heartbleed wasn't caused by general causes that would have possibly been detected by analysers using fuzzy logic or dynamic analysers as the vulnerability didn't actually did an over write or had a wrong code. But it actually had a missing code which caused buffer over read, thereby making it undetectable. In practice most testing software's try to test seeing what the output would be for a correct input, but however the Heartbleed is caused because of wrong input itself. There have been a large amount of solutions proposed like negative input checker, fuzzing with output examination, context configured source code weakness analysers, using a safer language etc. there have `also been discussion in regards with reducing the amount of code written to decrease the number of bugs and exceptions(Daniel,poul et al 2014;David 2014).

Many have even argued that would only certificate revocation and patching the code solve the effect of the bug, the answer would be a plain no, because of the fact that even if certificates are changed there are many clients who do not default check for certificates revocation, and are ok with revoked certificates even, examples of prone clients are chrome, internet explorer etc. these browsers do not have the default settings as to check for revoked certificates and this has to be set manually. Many users however are not that computer savvy and do not know the implications of conditions like these. Apart from this there are even many clients who do not change their private keys thereby making certificate revocation a waste as these keys could be used to decipher more data (James, Barton 2014;troyhunt 2014).
Most of the research has been focused upon how Heartbleed effects the server side and many of these have not concentrated how a client can be vulnerable to this attack, however a website stackexchange.com has conducted a test on the clients and found out that if a heartbeat request is sent to a client then the client also leak valuable information and become prone to the bug (ACCUVANT-LABS, 2014; ARBOR-NETWORKS, 2014).

Has there been any solution out yet which would have prevented the Heartbleed and identified it in early stage itself, the answer however is no, the reason behind this being even if it were to be found the techniques that could be used in likes of reverse testing, regression testing and negative testing would generally cost more and consume more time, which is the drawback of this method and because of which many companies do not like to use it.

TECHNICAL ANALYSIS OF HEARTBLEED

Here under this topic we would go in depth about what technically caused this vulnerability and how exactly this was exploited by the attackers. To understand what actually went wrong in the code we have to first take a glance at why was such a small mistake overlooked and was there for quite a long period. The very first reason for why this happened is the length of the written code. The code of open SSL is no less than 60 pages, this makes it impossible for human analysis to detect these small mistakes. Second reason is because of the programming language used. The code of open SSL was written in C this language doesn't explicitly correct memory allocations and also supports pointers which can be easily used to exploit vulnerabilities by attackers.

Let us now look into the code which caused this bug, for the amusement of many of us this code is just a single and simple line, which is as follows:

```
Buffer = OPENSSL_malloc(1+2+payload+padding);
```

In simple words on the anatomy of the above code memory is allocated from payload + padding by the use of a user controlled value. There was no length check condition that was applied for the allocation which resulted in the case where an attacker could compel the server to read and give back arbitrary memory locations, which would thereby enable the hacker to read data from the server by simply performing a heartbeat request every time with a different length. It is just as simple as that.

In other words, an attacker can actually exercise control over the heartbeat size and structure it in a way that it is larger than what it should be, then he can dispose it off to the target server machine using TCP on port 443 and can effectively receive a data of up to 64 kb from outside of what actual bounds of the heartbeat should be. Do it again with a different heartbeat size and get another 64kb response from other memory space in the server.

Struct
{
HeartbeatMessageType type;
Uint16 payload_lemgth;
Opaque payload[HeartbeatMessage.payload_length];
Opaque padding[padding_length];
}HeartbeatMessage;

This is the way the heart beat message looks like according to the official standard. The message arrives via SSL3_RECORD structure, which is a basic building block of the SSl/Tls Communication. The fields in the SSL3_record structure are as follows:
Length: which defines the number of bytes in the message
Data: which is pointer to the heart beat message.

```
struct ssl3_record_st
{
  unsigned int length;    /* How many bytes available */
  [...]
  unsigned char *data;    /* pointer to the record data */
  [...]
} SSL3_RECORD;
```

To clear the air off, the SSL3 records data points to the start of the received heartbeat message and the length field points out the number of bytes in the heartbeat message received. Going in depth of the received heartbeat message one finds out that there is a field of payload_length which is the number of bytes in the arbitrary payload which has to be sent back.

The diagram below shows how the attack actually works at technical level:

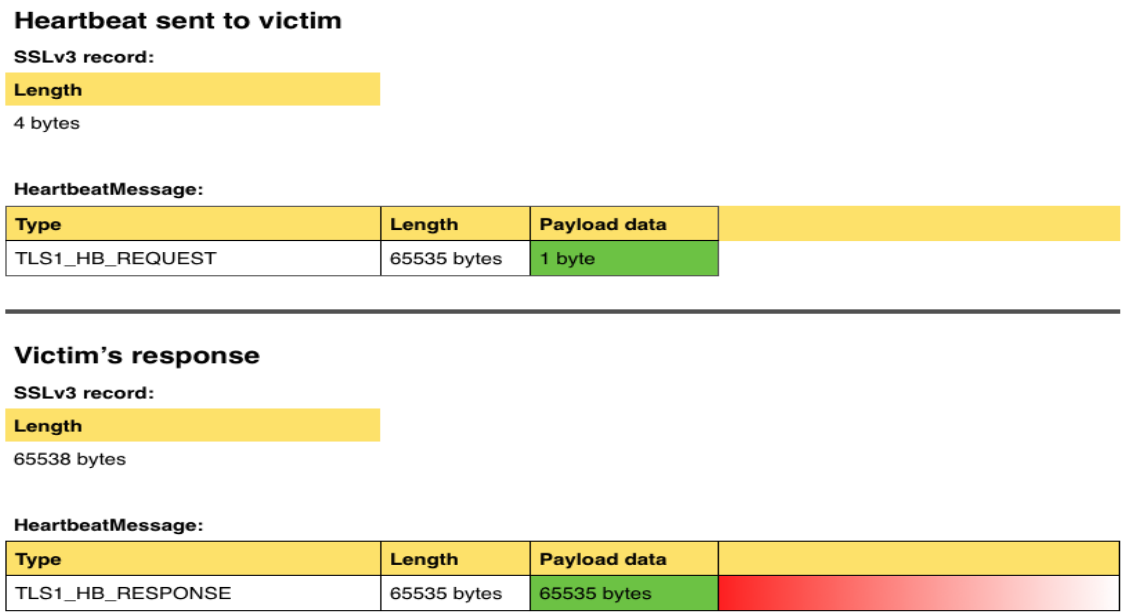

Figure 4: Actual anatomy of the attack

From the diagram it is easy to understand that if an attacker sends a 4 byte HeartBeat Message including a single byte payload, which is due to bb acknowledge correctly by the SSL3's length record. But the attacker lies in the payload length and claims it to be 65535 bytes in size when it is actually not. Victim however doesn't take this into account and reads 65535 bytes of data from its own memory, starting from the received Heartbeat Message payload and copies it into a suitable sized buffer to transfer it back to the attacker thereby leaking information as indicated in red.

PRACTICAL ATTACKS

Attack 1: The Flickr server

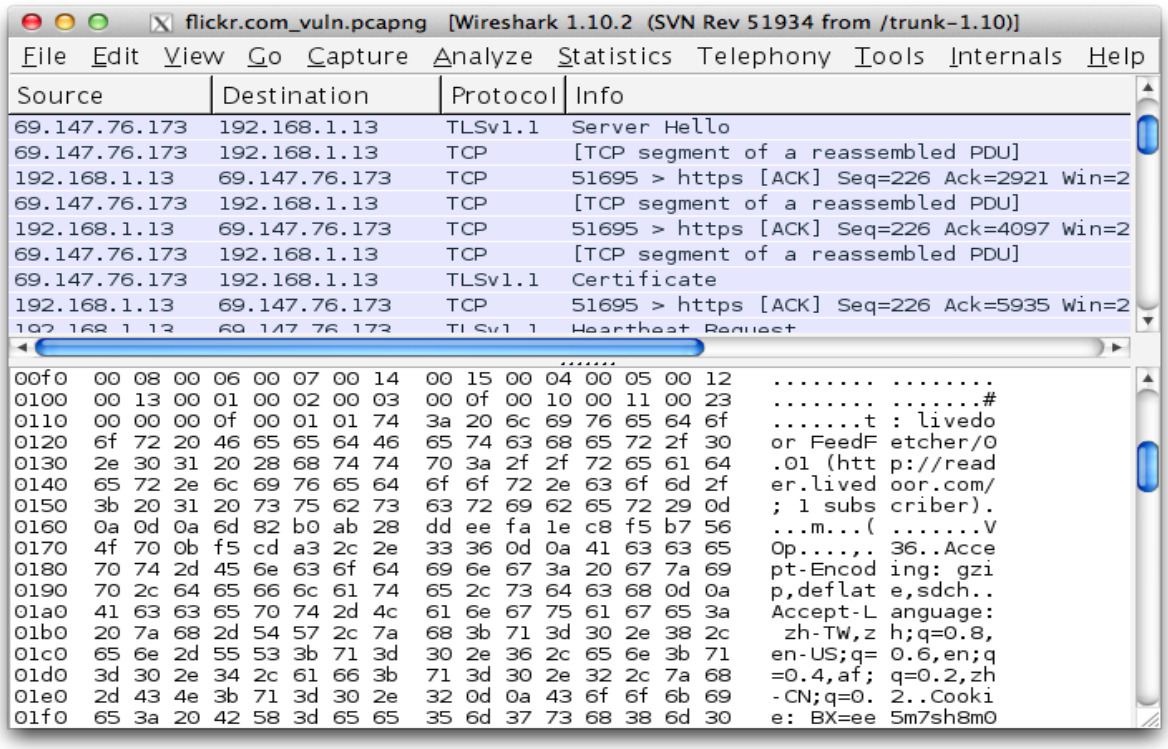

Figure 5: Flickr server attack (blog.erratasec.com)

As it can be seen in the figure the exploit has resulted in getting access to a user's session cookie, which has been cropped so that it is not used for any negative purpose, if this session cookie is copied and pasted into the address bar or what is called as side jacking is done then this would allow temporary access to the persons account.

Attack 2: Yahoo server

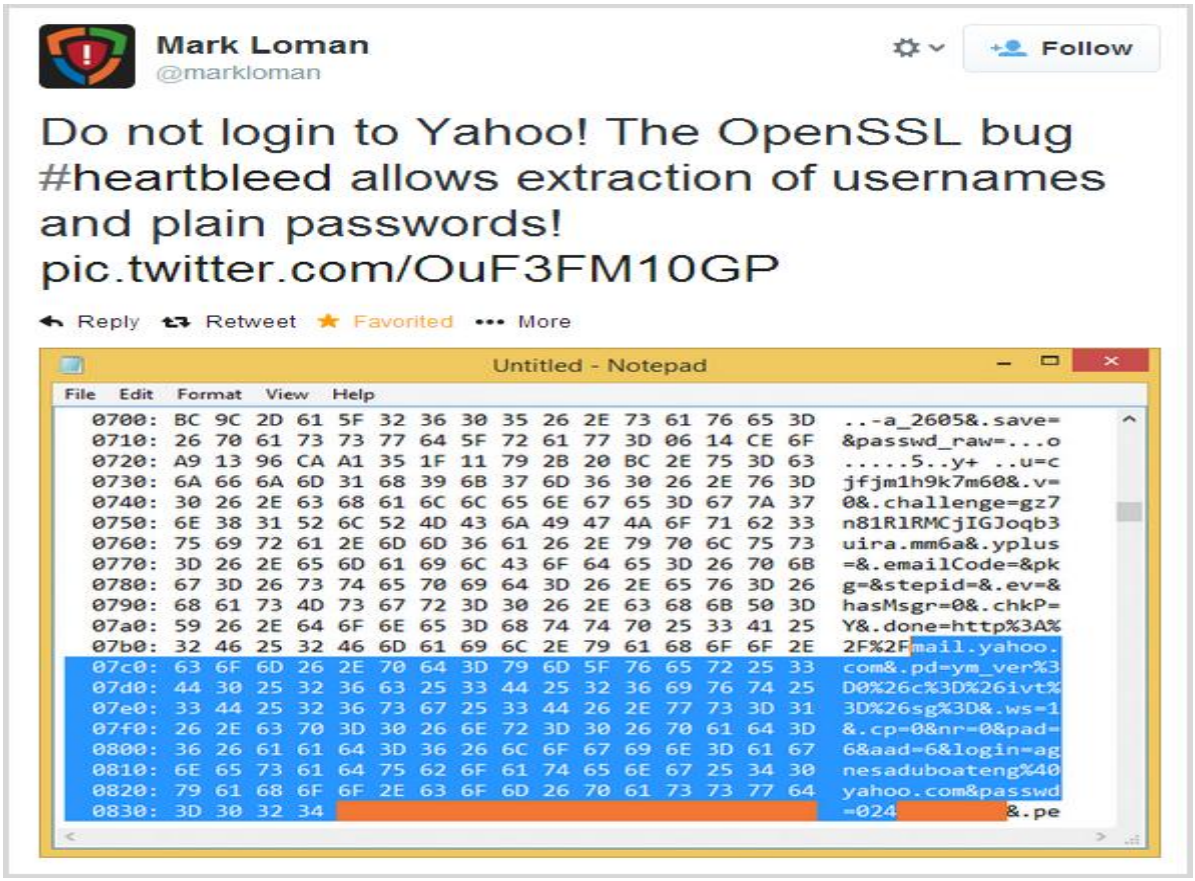

Figure 6: Attack on Yahoo (troyhunt.com)

As it can be seen in the image the exploit of yahoo, the attacker was able to retrieve plain text login ids and passwords of users thereby making the website and server vulnerable. The id and password here are also covered so as to avoid misuse.

Attack 3: Using Cupid to exploit servers

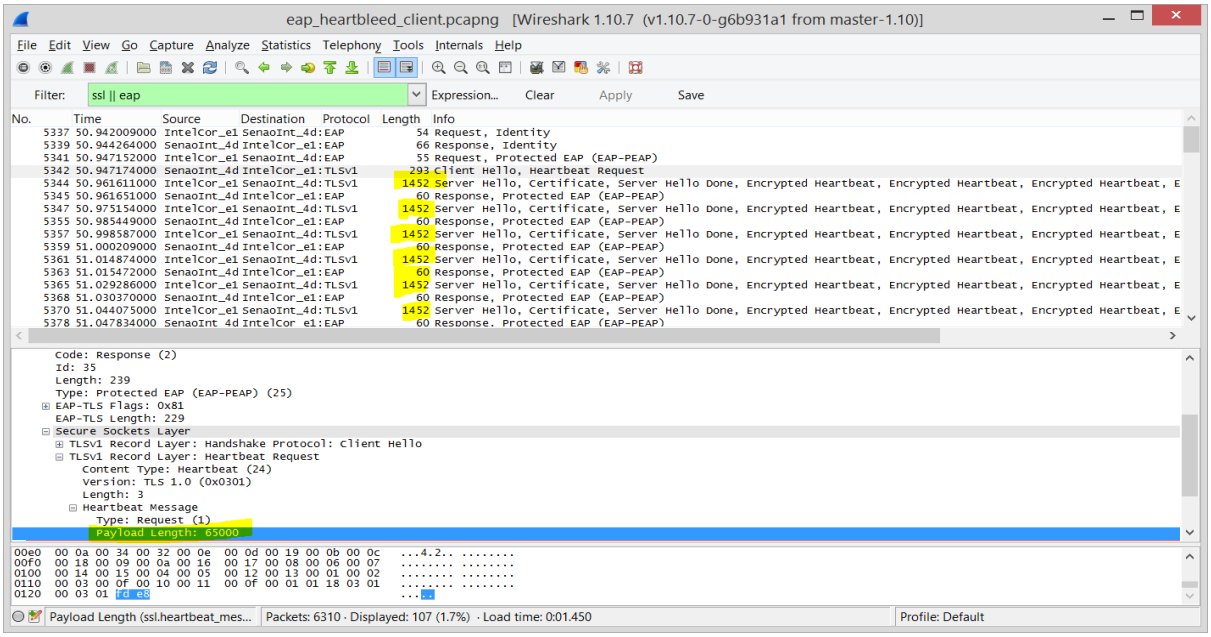

Figure 7: Exploiting using Cupid (sysvalue.com)

As it can be seen at first a username is provided (EAP Identity), then a TLS connection is established over EAP to authenticate. The software Cupid sends a heartbeat request before a handshake is made i.e. right after Client Hello and is successful in extracting up to 64kb of data from the server.

Attack 4: Cupid is used to exploit client

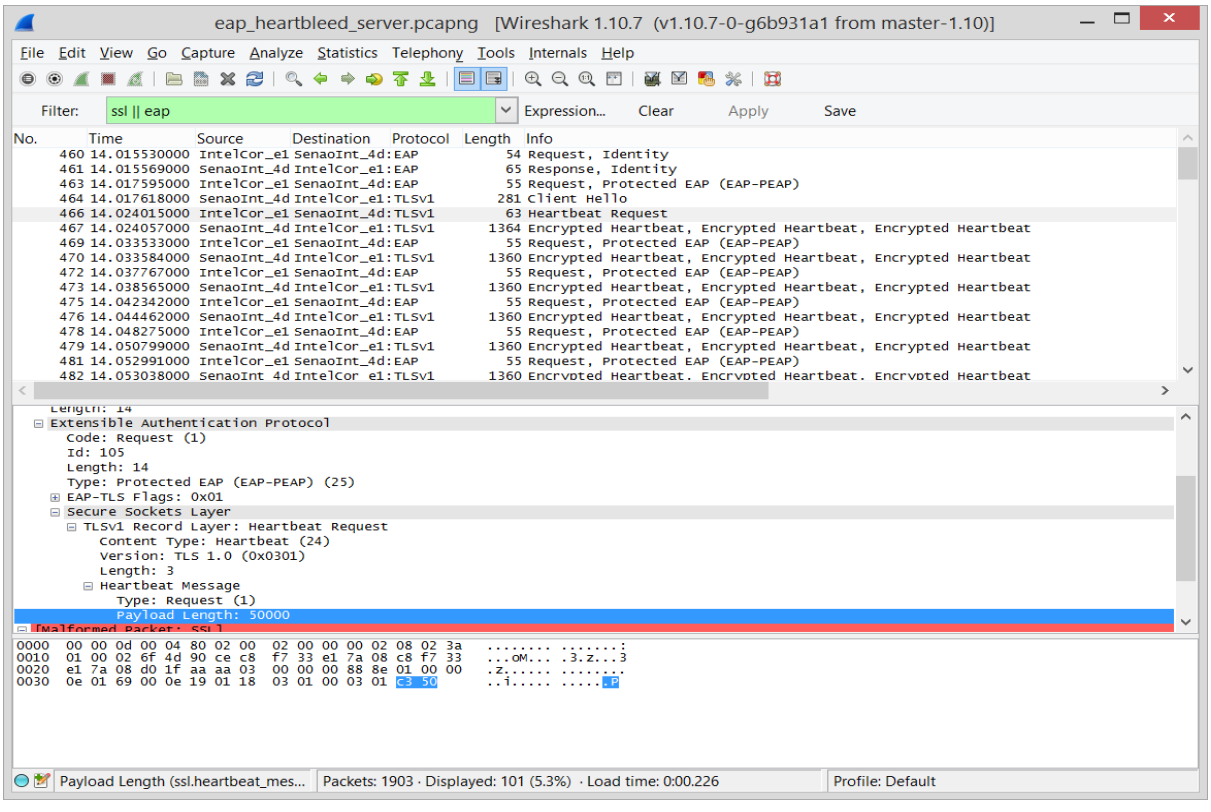

Figure 8: Exploitation of client using Cupid

As it can be seen in the image the vulnerable client sends a request for a TLS connection, by sending a client hello and then the cupid proceeds and sends a heartbeat request. The client happily responds back thereby leaking 64 kb of data. There are many more attacks that can be shown and there are many tools that are used to test whether a website is prone or vulnerable to Heartbleed. Some of them are Cupid, Filippo.io, ssl labs etc. There are even Metaspoilt and Nmap based tools to check for vulnerability in a server for Heartbleed. More number of these tools are listed at the website with URL blog.bugcrowd.com/heartbleed-exploit-yet/.

A POSSIBLE DETECTION SYSTEM FOR HEARTBLEED

Yes there was a vulnerability named heartbleed which impacted the internet very badly, so badly that the damage it caused can never be assessed. There has been a solution to this problem even with the new version of open SSL 1.0.1g with patches the current vulnerability and discards any heartbeat request whose payload length is greater than what is specified in the length bit of the message. But, as people of information technology we have to analyse and come up with a technique that could have possibly detected the vulnerability beforehand. There are many reasons given as for why this vulnerability went undetected and there have been many tools in market with which we can assess if servers are still vulnerable.

In my proposal of the solution for detection of this vulnerability and any other vulnerability I would like to propose an algorithm that would test the source code for possible missing blocks of codes, unchecked memory allocations, pointers, buffer over reads and unchecked conditional statements. This solution would be unique as it incorporates all the simple possible ways that could lead to such kind of exploitation. The problem here is not about the language used, the amount of code written, the quality of code, but the problem is that proper testing mechanism has to deployed while using low level languages as the code written in low level languages tends to be generally lengthy which thereby leads to ignoring such small pieces of code.

The algorithm would be as follows:

Step 1: Check all variable for the datatype used and type conversions.

Here in this step the tester is to check for the datatypes used for every variable to confirm whether the purpose of the variable and the type are a perfect match. Also the tester needs to check if there are any implicit or explicit type conversions in the code written, if any are found then it would be better to minimise these as much as possible and test them by giving negative inputs.

Step 2: Check for vague or unchecked memory allocations

In this step the tester needs to check how is memory being allocated and de allocated. This has to be done because the probability of data leaks increases with an increase in number of unchecked memory allocations. Here the tester should see to it that the memory being allocated is minimal and also there has been a condition check applied for the allocation. Secondly, the tester should also see to it that data transfer should not be allowed until a secure connection has been established and both of the client and server are sure of one another. By doing so the probability of giving the attacker a chance to extract data from the server or client reduces drastically and this is what has been applied to fix the heartbleed bug in the patch released.

Step 3: Deal properly with pointers

In this step the testers should see that all pointers being used in the initial stage of data transfers such as handshakes etc. should be checked so as to see whether they can be used to point in memory locations which should not generally be accessed at this stage. This can again be done by giving negative inputs. Pointers can be used better off if they are allocated to point to a guarded memory space than any memory space in the memory. If it is found that a pointer could be used as an exploit then it is the tester's duty to fix it by giving it some boundary memory.

Step 4: Check for loops and conditional statements,

Loop conditions are another weak points of programming languages this is because if a looping condition is not correctly specified then it could very well cause a catastrophe, if a looping condition or a conditional statement is accompanied by buffer allocation then if iti si not properly checked it can be used to exploit the code which would result in leak of data from the buffer beforehand.

Step 5: Check for Buffer allocations

Generally a weak point of programming languages is the use of buffer allocations, if the buffer allocation is not appropriate then it could lead the program into being vulnerable. The testers should check for these buffer allocations and make sure they are properly coded.

Step 6: Check for negative inputs

The best possible way to test any program would be give it inputs which you know would be wrong for the program and see how it reacts to it, this would lead to a disclosure of what the attacker would possibly be able to see if he tries to exploit the system using negative inputs. This is also the technique used to prevent SQL injection attacks.

If all the steps of the algorithm are followed alongside with static and fuzzy techniques then this would result in the proper and complete testing of programs. The algorithm applies techniques of reverse testing and negative testing along with basic testing principles which are generally overlooked while large scale software's are tested.

Had this algorithm been applied to test the open ssl code then the heartbleed bug would have been discovered in the very 2nd step and could possibly been avoided.

Conditions for the algorithm to be successful:

The algorithm solely relies on human eyes, there is no better way to test a software that have maximum numbers of humans analyse it. The time spent on the analyses is to be adequately balanced. Yes the drawback of this algorithm would be the fact that this testing would take a lot of time and would probably cost more money than automated testing techniques. But this technique would reveal bugs that are left behind by using popular testing techniques.

REFERENCES

[1]. Carvalho, M.; DeMott, J.; Ford, R.; Wheeler, D.A., \Heartbleed 101," Security & Privacy, IEEE ,vol.12, no.4, pp.63{67, July-Aug. 2014.
[2]. Wheeler, D.A., \Preventing Heartbleed," Computer, vol.47, no.8, pp.80{83, Aug. 2014.
[3]. Geer, D.E.; Kamp, P.H., \Inviting More Heartbleed," Security & Privacy, IEEE , vol.12, no.4,pp.46{50, July-Aug. 2014.
[4]. Sigholm, J.; Larsson, E., \Determining the Utility of Cyber Vulnerability Implantation: The Heart-bleed Bug as a Cyber Operation," Military Communications Conference (MILCOM), 2014 IEEE ,pp.110{116, 6-8 Oct. 2014.
[5]. Hao, Yongle; Jia, Yizhen; Cui, Baojiang; Xin, Wei; Meng, Dehu, \OpenSSL HeartBleed: Security Management of Implements of Basic Protocols," P2P, Parallel, Grid, Cloud and Internet Computing(3PGCIC), 2014 Ninth International Conference on, pp.520{524, 8-10 Nov. 2014.
[6]. Tsoutsos, N.G.; Maniatakos, M., \Trust No One: Thwarting `heartbleed' Attacks Using Privacy-Preserving Computation," VLSI (ISVLSI), 2014 IEEE Computer Society Annual Symposium on,pp.59{64, 9-11 July 2014.
[7]. Zakir Durumeric, James Kasten, David Adrian, J. Alex Halderman, Michael Bailey, Frank Li,Nicolas Weaver, Johanna Amann, Jethro Beekman, Mathias Payer, and Vern Paxson. 2014. \TheMatter of Heartbleed". In Proceedings of the 2014 Conference on Internet Measurement Conference(IMC '14). ACM, New York, NY, USA, pp.475{488.
[8]. Benjamin Edwards, Michael Locasto, and Jeremy Epstein. 2014. \Panel Summary: The Future of Software Regulation". In Proceedings of the 2014 workshop on New Security Paradigms Workshop (NSPW '14). ACM, New York, NY, USA, pp. 117{126.
[9]. ACCUVANT-LABS, 2014. Heartbleed Bug Advisory (CVE-2014-0160). [Online] Accuvant-Labs Available at: http://accuvantstorage.blob.core.windows.net/web/file/2016b4dc040c49ee991b5721e0dd62b3/HeartBleed-Bug-CVE-2014-0160-release.pdf [Accessed June 2014].
[10]. ARBOR-NETWORKS, 2014. The Heartburn Over Heartbleed: OpenSSL Memory Leak Burns Slowly. [Online] Available at: http://www.arbornetworks.com/asert/2014/04/heartbleed/ [Accessed 22 July 2014].
[11]. FILIPPO.IO, 2014. Heartbleed test. [Online] Available at: https://filippo.io/Heartbleed/#udsm.ac.tz [Accessed 02 July 2014].
[12]. GONÇALVESA, A., SOUSA, P. AND ZACARIAS, M., 2013. Using DEMO and activity theory to manage organization change. Procedia Technology, Vol.1, pp.563 – 572.
[13]. KRAWCZYK, K., 2014. Which CISCO Routers, Modems And Networking Gear Are Affected By And Safe From The Heartbleed Bug? [Online] Available at: http://www.digitaltrends.com/computing/which-cisco-routers-modems-networking-gear-safe-from-heartbleed-open-bug/#!bjRjZF [Accessed 22 July 2014].
[14]. Errastec, 2014. What Heartbleed bug looks like on wire [Online] available at: http://blog.erratasec.com/2014/04/what-heartbleed-bug-looks-like-on-wire.html#.VTR-9CGeDGc .[Accessed on April 2015]
[15]. Troyhunt, 2014. Everything yo need to know about heartbleed.[online]. Available at: http://www.troyhunt.com/2014/04/everything-you-need-to-know-about.html [Accessed April 2015]
[16]. Sysvalue, 2014. Cupid.[Online]. http://www.sysvalue.com/en/heartbleed-cupid-wireless/ [Accessed april 2015]
[17]. The register,2014. Heartbleed explained. [Online]. Available at: http://www.theregister.co.uk/2014/04/09/heartbleed_explained/ [Accessed april 2015]