Heartbleed vulnerability and CVSS: A predictive approach for threat of exploitation

Mehak Bashir[1], Muheet Ahmed Butt[2], Majid Zaman[3]
[1]M.Tech. (CSE), [2]Scientist 'D', Kashmir University, [3]Scientist 'D', Kashmir University.

***Abstract*—. The proposed research describes 'OpenSSL Heartbleed' vulnerability and also proposes a methodology that explains the severity of exploitation posed by some common types of vulnerabilities, based on Common Vulnerability Scoring System (CVSS), using Naive Bayes classification algorithm.**

Keywords—OpenSSL; Heartbleed; Vulnerability; CVSS; Naive Bayes classification.

I. INTRODUCTION

A. OpenSSL Heartbleed

The OpenSSL is an open source implementation of the Secure Sockets Layer (SSL) and the Transport Layer Security (TLS) [7].The OpenSSL platform provides security when data is transferred from one point of the internet to another part [1]. The Secure socket layer (SSL) is the most popular protocol used on the Internet for secure transfer of data [4]. The OpenSSL protocol is used in two-thirds of all websites to prevent hackers from stealing sensitive information like passwords or credit card data [5]. If the data being transferred is edited/changed/ updated along the way, data integrity is compromised and if the data is accessed and falls into the wrong hands, confidentiality of data is lost. Data Integrity and confidentiality should be maintained as data moves from point to point. The OpenSSL protocol works by authenticating the server to the client and client to server through the use of digital certificates signed by a trusted third party. Private and public keys are also used in the OpenSSL to provide security. The OpenSSL protocol is however subject to vulnerabilities [2], [3] whether directly or indirectly. This can be seen by the trusted third parties who authenticate the identities of transacting individuals have been exposed to continuous attacks/threats. [6]. various other vulnerabilities have been found within the OpenSSL protocol and the most notable has been the Heartbleed bug.

The name 'Heartbleed' itself explains the vulnerability – 'Heart' of the Heartbleed came from Heartbeat protocol and 'bleed' stands for data leakage. That means data leakage in the Heartbeat protocol implementation, specifically the OpenSSL implementation of the protocol.

B. Naive Bayes Classifier

Naive Bayes is a kind of classifier which uses the Bayes Theorem. It predicts membership probabilities for each class such as the probability that given record or data point belongs to a particular class. The class with the highest probability is considered as the most likely class. This is also known as Maximum A Posteriori (MAP).

The MAP for a hypothesis is:

$$MAP(H) = \max(P(H|E))$$
$$= \max((P(E|H)*P(H))/P(E))$$
$$= \max(P(E|H)*P(H))$$

P (E) is evidence probability, and it is used to normalize the result. It remains same so, removing it won't affect.

Naive Bayes is a classification algorithm for binary (two-class) and multi-class classification problems. The technique is easiest to understand when described using binary or categorical input values [8].

C. Vulnerability

Vulnerability, in information technology (IT), is a flaw in code or design that creates a potential point of security compromise for an endpoint or network.

Vulnerabilities create possible attack vectors, through which an intruder could run code or access a target system's memory. The means by which vulnerabilities are exploited are varied and include code injection and buffer overruns; they may be conducted through hacking scripts, applications and free hand coding.

Vulnerabilities are constantly being researched and detected by the security industry, software companies, cybercriminals and other individuals. Some companies offer bug bounties for these discoveries. Nevertheless, when vulnerability disclosure is considered, the question of how much information to provide and when to make it public is a contentious issue.

Some people argue for full and immediate disclosure, including the specific information that could be used to exploit the vulnerability; others believe that vulnerability information should not be published at all because the information can be used by an intruder. A zero-day

exploit, for example, takes place as soon as vulnerability becomes generally known. To mitigate risk, many experts believe that limited information should be made available to a selected group after some specified amount of time has elapsed since detection.

Both black hats and white hats regularly search for vulnerabilities and test exploits, however, and if a cybercriminal finds a useful and unreported security hole, he is likely to take advantage of it. Proponents of disclosure maintain that it leads to more patching of vulnerabilities and more secure software [2].

D. Types of Security Vulnerabilities

Most software security vulnerabilities fall into one of a small set of categories:

1) buffer overflows
2) unvalidated input
3) race conditions
4) access-control problems
5) weaknesses in authentication, authorization, or cryptographic practices

II. LITERATURE SURVEY

A. Introduction to Survey Report

In April 7, 2014 The Heartbleed Bug was independently discovered by a team of security engineers (Riku, Antti, and Matti, 2014) at Codenomicon and Neel Mehta of Google Security, who first reported it to the OpenSSL team. The security engineers did not have an idea of the vulnerability until the team found heartbleed bug while improving the Safeguard features. This was the city Codenomicon's Defense security testing tools and reported this bug to the NCSC-FI for vulnerability coordination and reporting to OpenSSL team [5].

In addition, Bloomberg (2014) accused the U.S National Security Agency (NSA) of knowing the Heartbleed Bug for the last two years. Although, the report says the NSA was using it to gain information instead of disclosing it to the OpenSSL developer. After the NSA declining to comment to report of knowing about the Heartbleed Bug, NSA also denied that they were aware of Heartbleed Bug until the vulnerability was made public by the private security engineering of Google. Overall, the questions remain about whether anyone from the NSA or U.S government might have exploited the code for their benefits before published to the public.

The Heartbleed Bug is not a virus, it's not a worm or a malicious code, and it has nothing to do with the Man-in-the-Middle, but it's a simple programming mistake. However, the Heartbleed Bug is a serious vulnerability in the most popular OpenSSL cryptographic software library. This software allows anyone with little knowledge to steal the information such as the names and passwords of the users and the actual content protected, under normal conditions, by the SSL/TLS encryption used to secure the internet. In addition, the code of the Heartbleed Bug is available to the public and there are several sites that have tutorials to teach the use of the software, therefore this vulnerability is most critical.

The purposes of the SSL/TLS are to provide communication security and privacy over the internet for applications such as web, email, VPNs and social media [5]. Smartphones are the best practical example of client side attack, which lead to Blackberry (Z10) products to be vulnerable to Heartbleed Bug, in contrast of Apple's iOS devices are not affected by OpenSSL. There are other devices affected by Heartbleed such as; IP Phones, Routers, Medical Devices and Smart TV sets. In addition, about 34 percent of Android devices run on version 4.1.x of the mobile OS, which according to Google millions of Android smartphones never, or only rarely receive available updates that patch dangerous security defects. For that reason, Android users should download Heartbleed Detector, a free application developed by Lookout.

The Heartbleed Bug attack works in several steps: First, the attacker creates a custom Heartbleed. Second, the packet is transmitted to vulnerable OpenSSL web server. Third web server processes packet. Fourth, the code grabs up 64KB of extra memory and hopes of capturing something sensitive from memory. Fifth, web server responds by sending a packet back which knowingly includes this extra sensitive data. Sixth, attacker analyzes packets to see if there is anything interesting, if not reruns attack to capture more memory. Lastly, if web server's certificates private key is captured, it can be used to decrypt current and historical user data and credentials. Overall, is not complex to use the Heartbleed software. As mentioned before, any Heartbleed based attacks are not traceable, due that the problem has existed for the past 2 years without the knowledge of the public. Most server operators use a vulnerable method of the OpenSSL versions 1.0.1 – 1.0.1f and likely don't have enough logs/monitoring to determine whether a site was compromised.

The Heartbleed bug reflects one of the most catastrophic vulnerabilities during the OpenSSL history for several reasons: it allowed attackers to retrieve private information and user data, it was easy to exploit and HTTPS and other TLS services have become increasingly popular by the resulting in more affected services [6].In addition, Stephen Solis-Reyes 19 year-old from Canada was arrested for exploiting the Heartbleed Bug to attack the website of the Canada Revenue Agency. As result, of the attack, Mr. Solis-Reyes had stolen 900 social insurances numbers (Elsevier, 2014). According, to Ivan Ristic, director of engineering at Qualys, has claimed that the percentage of websites vulnerable to the flaw had dropped from 25 percent since the bug was discovered.

"Assistant Research Scientist Dave Levin and Assistant Professor of Electrical and Computer Engineering Tudor Dumitras were part of a team that analyzed the most popular websites in the United States-more than one million sites were examined-to better understand the extent to which systems administrators followed specific protocols to fix the problem"(NewsRx, 2014) [5].

B. General Survey

Who and what caused Heartbleed Bug? This question is answered with two graphics, displaying the bad code and the good code. The programmer Robin Seggelmann, a 31 year old based in Germany, submitted the code. The purpose of the software was to enable a function called "Heartbeat" in OpenSSL. This software package was to be used by nearly half of all web servers to enforce the connections. "In one of the new features, unfortunately, I missed validating a variable containing a length" (Seggelmann, 2012). In addition, the code went undetected by several code reviewers and everyone else for over two years. The graphics below shows the c- language code for the Heartbeat message in the OpenSSL source code. In the first graphic, it shows the data structure and the length of the message is given as payload_length.

As it shows below in the Graphic 1(Figure 0), the incoming data contains a payload length "payload", the mistake of the code is that it trusts the request without bounds checks. OpenSSL then allocates a buffer for its response, and copies "payload" data bytes from the pointer "pl" into it. As result, there's no "if statement" to make sure that there are actually "payload" bytes in data, or that this is in bounds. Since, there is no "if statement" the attacker gets a 64KB of data in length from main memory. When the attacker gets the 64KB of data the connection is no longer secure between servers and computers [7].

On the other hand, Graphic 2(Figure 0) shows the correct code with the "if statement" placed in the correct place. However, by making the correction of the code it does not guarantee that our server is secure and is no longer vulnerable. In order, to have a secure server or routers the security technician must take the following actions; upgrade your server to the latest version of OpenSSL, reissue and then revoke all certificates used with the vulnerable version of OpenSSL, and upgrade your security patches. As social media and online shopping user such as; Facebook, Google, eBay, Instagram and other sites that require user credentials may have to change our password if we haven't change within the past 6 months.

III. THEORETICAL EXPLAINATIONS

A. How The Heartbeat Works

The heartbeat extension protocol consists of two message types: HeartbeatRequest message and HeartbeatResponse message and the extension protocol depends on which TLS protocol is being used as describe below:

1) When Using Reliable Transport Protocol

One side of the peer connection sends a HeartbeatRequest message to the other side. The other side of the connection should immediately send a HeartbeatResponse message. This makes one successful Heartbeat and thus, keeping connection alive – this is called 'keep-alive' functionality. If no response is received within a specified timeout, the TLS connection is terminated.

Graphic 1

```
/* Read type and payload length first */

hbtype = *p++;

n2s(p, payload);

pl = p;
```

Graphic 2

```
hbtype = *p++;

n2s(p, payload);

if (1 + 2 + payload + 16 > s->s3->rrec.length)

    return 0; /* silently discard per RFC 6520 sec. 4 */

pl = p;
```

Figure 0: Graphic 1 and 2 shows the Heartbleed code

2) Unreliable Transport Protocol

One side of the peer connection sends HeartbeatRequest message to the other side. The other side of the connection should immediately send a HeartbeatResponse message. If no response is received within specified timeout another HeartbeatRequest message is retransmitted. If expected response is not received for specified number of retransmissions, the DTLS (Datagram Transport Layer Security) connection is terminated.

When a receiver receives a HeartbeatRequest message, the receiver should send back an exact copy of the received message in the HeartbeatResponse message. The sender verifies that the HeartbeatResponse message is same as what was originally sent. If it is same, the connection is kept alive. If the response does not contain the same message, the HeartbeatRequest message is retransmitted for a specified number of retransmissions [7].

B. Data Leakage Leading to Heartbleed

There is a bug in the implementation of the Heartbeat reply to the received Heartbeat request message. Heartbeat reply copies the received payload to the Heartbeat response message to verify that the secured connection is still active, without checking if the payload length is same as the length of the request payload data.

The problem here is that the OpenSSL heartbeat response code does not check to make sure that the payload length field specified in the heartbeat request message matches the actual length of the payload.

If the heartbeat request payload length field is set to a value larger than the actual payload, it would result in a return of the payload followed by whatever contents are currently contained in active memory buffer, beyond the end of the payload. A heartbeat request the payload length can be set to a maximum value of 65535 bytes. Therefore the bug in the OpenSSL heartbeat response code could copy as much as 65535 bytes from the machine's memory and send it to the requestor [6].

This bug is illustrated below in “Figure 1: Memory Leak.

“Figure 1: Memory leak” shows that when the request payload data is ‘ma’ and payload length is ‘2’ then 2 bytes from source (i.e. ‘ma’) is copied to the ‘destination’ memory area. But when the request payload data is ‘ma’ and payload length falsely indicates that it is 8 bytes instead of 2, 8 bytes (i.e. ‘madadbro’) from the ‘source’ memory area to the ‘destination’ memory area. This ‘destination’ data is finally sent to the requestor, causing the memory leak that is now known as the Heartbleed bug [9].

C. Code Fix

“Figure 2: The OpenSSL code fix for the Heartbleed bug” shows the change in OpenSSL's file t1_lib.c between version 1.0.1 and OpenSSL version 1.0.1g that was made to fix the Heartbleed bug [7].

This code fix has two tasks to perform:

First, it checks to determine if the length of the payload is zero or not. It simply discards the message if the payload length is 0.

The second task performed by the bug fix makes sure that the heartbeat payload length field value matches the actual length of the request payload data. If not, it discards the message.

The official notice about the bug was published by the OpenSSL group at https://www.openssl.org/news/secadv_20140407.txt and is reproduced in “Figure 3: OpenSSL Security Advisory.”

D. Real-World Impact of Heartbleed

By exploiting the Heartbleed vulnerability, an attacker can send a Heartbeat request message and retrieve up to 64 KB of memory from the victim's server. The contents of the retrieved memory depends on what's in memory in the server at the time, but could potentially contain usernames, passwords, session IDs or secret private keys or other sensitive information. Following figure illustrates how an attacker can exploit this vulnerability. This attack can be made multiple times without leaving any trace of it. "Figure 4: Exploiting the Heartbleed vulnerability" illustrates how an attacker can exploit the Heartbleed vulnerability.

E. Factors to Determine Severity of a Vulnerability- Common Vulnerability Scoring System (CVSS)

The data used in this section comes from many different sources. The main reference source is the National Vulnerability Database (NVD), which includes Information for all Common Vulnerabilities and Exposures (CVEs). As of May 2015, there are close to 69,000 CVEs in the database. Connected to each CVE is also a list of external references to exploits, bug trackers, vendor pages, etc. Each CVE comes with some Common Vulnerability Scoring System (CVSS) metrics and parameters, which can be found in Table 1. A CVE-number is of the format CVE-Y-N, with a four number year Y, and a 4-6 number identifier N per year. Major vendors are pre-assigned ranges of CVE-numbers to be registered in the oncoming year, which means that CVE-numbers are not guaranteed to be used or to be registered in consecutive order [9].

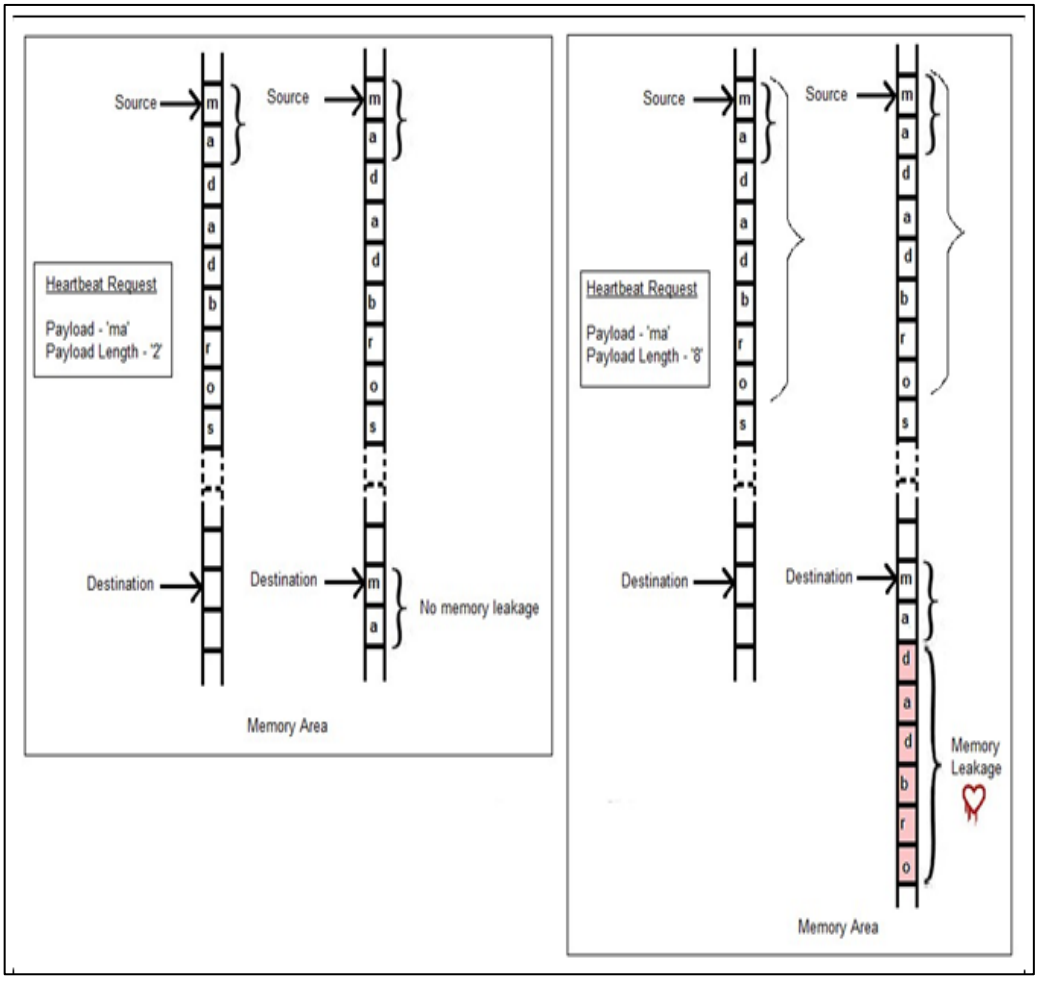

Figure 1: Memory Leak

\openssl-1.0.1\ssl\t1_lib.c

```
/* Read type and payload length first */
hbtype = *p++;
n2s(p, payload);
pl = p;

if (s->msg_callback)
    s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT
        &s->s3->rrec.data[0], s->s3->rrec.length,
        s, s->msg_callback_arg);
```

\openssl-1.0.1g\ssl\t1_lib.c

```
if (s->msg_callback)
    s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
        &s->s3->rrec.data[0], s->s3->rrec.length,
        s, s->msg_callback_arg);

/* Read type and payload length first */
if (1 + 2 + 16 > s->s3->rrec.length)
    return 0; /* silently discard */
hbtype = *p++;
n2s(p, payload);
if (1 + 2 + payload + 16 > s->s3->rrec.length)
    return 0; /* silently discard per RFC 6520 sec. 4 */
pl = p;
```

Figure 2: The OpenSSL code fix for the Heartbleed bug

F. Naive Bayes Classification

Naive Bayes is a classification algorithm for binary (two-class) and multi-class classification problems. The technique is easiest to understand when described using binary or categorical input values [8].

It is called naive Bayes or idiot Bayes because the calculation of the probabilities for each hypothesis is simplified to make their calculation tractable. Rather than attempting to calculate the values of each attribute value P (d1, d2, d3|h), they are assumed to be conditionally independent given the target value and calculated as P (d1|h) * P (d2|H) and so on.

This is a very strong assumption that is most unlikely in real data, i.e. that the attributes do not interact. Nevertheless, the approach performs surprisingly well on data where this assumption does not hold.

1) Representation Used By Naive Bayes Models

The representation for naive Bayes is probabilities.

A list of probabilities is stored to file for a learned naive Bayes model. This includes:

- *Class Probabilities*—the probabilities of each class in the training dataset.
- *Conditional Probabilities*—the conditional probabilities of each input value given each class value.

2) Learn a Naive Bayes Model from Data

Learning a naive Bayes model from training data is fast. Training is fast because only the probability of each class and the probability of each class given different input (x) values need to be calculated. No coefficients need to be fitted by optimization procedures.

1) Calculating Class Probabilities

The class probabilities are simply the frequency of instances that belong to each class divided by the total number of instances.

For example in a binary classification the probability of an instance belonging to class 1 would be calculated as:

P(class=1) = count(class=1) / (count(class=0) + count(class=1))

In the simplest case each class would have the probability of 0.5 or 50% for a binary classification problem with the same number of instances in each class.

2) Calculating Conditional Probabilities

The conditional probabilities are the frequency of each attribute value for a given class value divided by the frequency of instances with that class value.

For example, if a "weather" attribute had the values "sunny" and "rainy" and the class attribute had the class values "go-out" and "stay-home", then the conditional probabilities of each weather value for each class value could be calculated as:

P(weather=sunny|class=go-out) = count(instances with weather=sunny and class=go-out) / count(instances with class=go-out)

P(weather=sunny|class=stay-home) = count(instances with weather=sunny and class=stay-home) / count(instances with class=stay-home)

P(weather=rainy|class=go-out) = count(instances with weather=rainy and class=go-out) / count(instances with class=go-out)

P(weather=rainy|class=stay-home) = count(instances with weather=rainy and class=stay-home) / count(instances with class=stay-home)

1) Make Predictions with a Naive Bayes Model

Given a naive Bayes model, you can make predictions for new data using Bayes theorem.

MAP(h) = max(P(d|h) * P(h))

Using our example above, if we had a new instance with the weather of sunny, we can calculate:

go-out = P(weather=sunny|class=go-out) * P(class=go-out)
stay-home = P(weather=sunny|class=stay-home) * P(class=stay-home)

We can choose the class that has the largest calculated value. We can turn these values into probabilities by normalizing them as follows:

P(go-out|weather=sunny) = go-out / (go-out + stay-home)
P(stay-home|weather=sunny) = stay-home / (go-out + stay-home)

If we had more input variables we could extend the above example. For example, pretend we have a "car" attribute with the values "working" and "broken". We can multiply this probability into the equation.

OpenSSL Security Advisory [07 Apr 2014]

TLS heartbeat read overruns (CVE-2014-0160)

A missing bounds check in the handling of the TLS heartbeat extension can be used to reveal up to 64k of memory to a connected client or server.

Only 1.0.1 and 1.0.2-beta releases of OpenSSL are affected including 1.0.1f and 1.0.2-beta1.

Thanks for Neel Mehta of Google Security for discovering this bug and to Adam Langley <agl@chromium.org> and Bodo Moeller <bmoeller@acm.org> for preparing the fix.

Affected users should upgrade to OpenSSL 1.0.1g. Users unable to immediately upgrade can alternatively recompile OpenSSL with -DOPENSSL_NO_HEARTBEATS.

1.0.2 will be fixed in 1.0.2-beta2.

Figure 3: OpenSSL Security Advisory

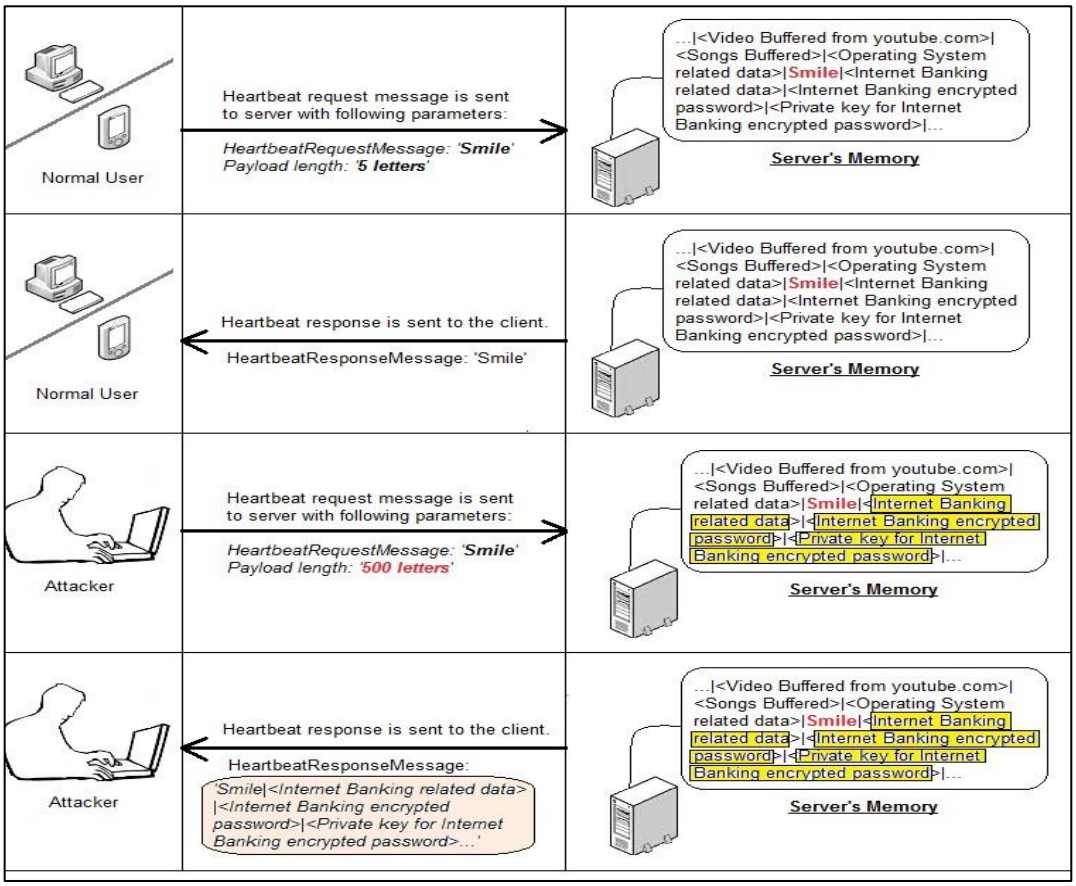

Figure 4: Exploiting the Heartbleed vulnerability

TABLE 1 CVSS (version 2) Base Metrics, with definitions from Mell et al. (2007).

Parameter	Values	Description
CVSS Score	0-10 [Low (0.1 -3.9), Medium (4.0 – 6.9), High (7.0 – 8.9), Critical (9.0 – 10.0)]	This value is calculated based on the next six parameters, with a formula (Mell et al., 2007).
Access Vector	Local Adjacent Network	The access vector (AV) determines how vulnerability can be exploited. A local attack requires physical access to the computer or a shell account. Vulnerability with Network access is also called remotely exploitable.
Access Complexity	Low Medium High	The access complexity (AC) classifies the difficulty to exploit the vulnerability.
Authentication	None Single Multiple	The authentication (Au) categorizes the number of times that an attacker must authenticate to a target to exploit it, but does not measure the difficulty of the authentication process itself.
Confidentiality	None Partial Complete	The confidentiality (C) metric assorts the impact of the confidentiality, and amount of information access and disclosure. This may include partial or full access to file systems and/or database tables.
Integrity	None Partial Complete	The integrity (I) metric categorizes the impact on the integrity of the exploited system. For example, if the remote attack is able to partially or fully modify information in the exploited system.
Availability	None Partial Complete	The availability (A) metric categorizes the impact on the availability of the target system. Attacks that consume network bandwidth, processor cycles, memory or any other resources affect the availability of a system.

For example below is the calculation for the "go-out" class label with the addition of the car input variable set to "working":

go-out = P(weather=sunny|class=go-out) * P(car=working|class=go-out) * P(class=go-out)

IV. PROPOSED WORK

A. Algorithm for Predicting Severity/Threat Of Exploitation Using Naive Bayes Approach

- Convert the data set into a frequency table.
- Create Likelihood table by finding the probabilities, like probability of High threat of exploitation is (4/7) = 0.57 and probability of Low threat of exploitation is (3/7) = 0.43.
- Now, use Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of prediction.

B. Frequency Table for Some Common Vulnerabilities Based on CVSS (Version2) Parameters

The values for CVSS (Version2) parameters: CVSS Score, Access Vector, Access, Complexity, Authentication, Confidentiality, Integrity and Availability for some common types of vulnerabilities such as PhpMyAdmin Reflected cross- Site Scripting Vulnerability (CVE-2013-1937), MySQL Stored SQL Injection (CVE-2013-0375), SSL v3 POODLE Vulnerability (CVE_2014-3568), VMWare Guest to Host Escape Vulnerability (CVE-2012-1516), Apache Tomcat XML Parser Vulnerability (CVE-2009-0783), OpenSSL Heartbleed Vulnerability (CVE-2014-0160) etc. are tabulated in table 2.

C. Likelihood Table for Finding the Probabilities(P) Of Various CVSS (version 2) Parameters

The likelihood table for finding the probabilities of various CVSS parameters : CVSS Score, Access Vector, Access, Complexity, Authentication, Confidentiality, Integrity and Availability, as defined in table 1, using the data presented in above frequency table (table 2) is depicted in table 3.

TABLE 2 Frequency table for some common vulnerabilities using CVSS (version 2) Base Metrics.

Vulnerability	CVSS V2 Base score	Access Vector	Access Complexity	Authentic ation	Confidentiality	Integrity Impact	Availability Impact	Severity/ Threat of Exploitation
PhpMyAdmin Reflected cross- Site Scripting Vulnerability (CVE-2013-1937)	Medium	Network	Medium	None	None	Partial	None	Low
MySQL Stored SQL Injection (CVE-2013-0375)	Medium	Network	Low	Single	Partial	Partial	None	High
SSL v3 POODLE Vulnerability (CVE_2014-3568)	Medium	Network	Medium	None	Partial	None	None	Low
VMWare Guest to Host Escape Vulnerability (CVE-2012-1516)	Critical	Network	Low	Single	Complete	Complete	Complete	High
Apache Tomcat XML Parser Vulnerability (CVE-2009-0783)	Medium	Local	Low	None	Partial	Partial	Partial	High
Cisco IOS Arbitrary Command Execution Vulnerability (CVE-2012-0384)	High	Network	Medium	Single	Complete	Complete	Complete	High
Apple iWork Denial of Service Vulnerability (CVE-2015-1098)	Medium	Network	Medium	None	Partial	Partial	Partial	Low
OpenSSL Heartbleed Vulnerability (CVE-2014-0160)	Medium	Network	Low	None	Partial	None	None	High
GNU Bourne-Again Shell(Bash) 'ShellShock' Vulnerability (CVE-2014-6271)	Critical	Network	Low	None	Complete	Complete	Complete	High
DNS Kaminsky Bug (CVE-2008-1447)	Medium	Network	Low	None	None	Partial	None	Low

TABLE 2 (continued)

Vulnerability	CVSS V2 Base score	Access Vector	Access Complexity	Authentication	Confidentiality	Integrity Impact	Availability Impact	Severity/ Threat of Exploitation
Joomla Directory Traversal Vulnerability (CVE-2010-0467)	Medium	Network	Low	None	Partial	None	None	Low
Cisco Access Control ByPass Vulnerability (CVE-2012-1342)	Medium	Network	Low	None	None	Partial	None	Low
Juniper Proxy ARP Denial of Service Vulnerability (CVE-2013-6014)	Medium	Adjacent	Low	None	None	Complete	None	High
DokuWiki Reflected Cross-Site Scripting Attack (CVE-2014-9253)	Medium	Network	Medium	None	None	Partial	None	Low
Adobe Acrobat Buffer Overflow Vulnerability (CVE-2009-0658)	Critical	Network	Medium	None	Complete	Complete	Complete	High
Microsoft Windows Bluetooth Remote Code Execution Vulnerability (CVE-2011-1265)	High	Network	Low	None	Complete	Complete	Complete	High
Apple ios Security control Bypass vulnerability (CVE-2014-2019)	Medium	Local	Low	None	None	Complete	None	High
SearchBlox Cross-Site Request Forgery Vulnerability (CVE-2015-0970)	Medium	Network	Medium	None	Partial	Partial	Partial	Low
SSL/TLS MITM Vulnerability (CVE-2014-0224)	Medium	Network	Medium	None	Partial	Partial	Partial	Low
Google Chrome ByPass Vulnerability (CVE-2012-5376)	Critical	Network	Low	None	Complete	Complete	Complete	High

TABLE 3 Likelihood table for calculation of probabilities of CVSS (version 2) parameters

Severity/Threat of Exploitation	
P (High) = 12/21 = 4/7	P (Low) = 9/ 21 = 3/7
CVSS V2 Base score	
P (Low/ High) = 0/12 = 0	P(Low/ Low) = 0/9 = 0
P (Medium/ High) = 6/12 =1/2	P(Medium/ Low) = 9/9 = 1
P (High/ High) = 2/12 = 1/6	P(High/ Low) = 0/9 = 0
P (Critical/ High) = 4/12 = 1/3	P(Critical/ Low) = 0/9 = 0
Access Vector (AV)	
P (Local/ High) = 3/12 = 1/4	P (Local/ Low) = 0/9 = 0
P (Adjacent/ High)= 1/12	P (Adjacent/ Low) = 0/9 = 0
P (network/ High) = 8/12 = 2/3	P (Network/ Low) = 9/9 = 1
Access Complexity (AC)	
P (Low/ High) = 9/12 = 3/4	P (Low/ Low) = 3/9 = 1/3
P (Medium/ High)= 3/12 = 1/4	P (Medium/ Low) = 6/9 = 2/3
P (High/ High) = 0/12 = 0	P (High/ Low) = 0/9 = 0
Authentication (Au)	
P (None/ High) = 9/12 = 3/4	P (None/ Low) = 9/9 = 1
P (Single/ High)= 3/12 = 1/4	P (Single/ Low) = 0/9 = 0
P (Multiple/ High) = 0/12 = 0	P (Multiple/ Low) = 0/9 = 0
Confidentiality Impact (C)	
P (None/ High) = 2/12 = 1/6	P (None/ Low) = 4/9
P (Partial/ High)= 3/12 = 1/4	P (Partial/ Low) = 5/9
P (Complete/ High) = 7/12	P (Complete/ Low) = 0/9 = 0
Integrity Impact (I)	
P (None/ High) = 1/12	P (None/ Low) = 2/9
P (Partial/ High)= 2/12 = 1/6	P (Partial/ Low) = 7/9
P (Complete/ High) = 9/12 = 3/4	P (Complete/ Low) = 0/9 = 0
Availability Impact (A)	
P (None/ High) = 4/12 = 1/3	P (None/ Low) = 6/9 = 2/3
P (Partial/ High)= 1/12	P (Partial/ Low) = 3/9 = 1/3
P (Complete/ High) = 7/12	P (Complete/ Low) = 0/9 = 0

D. Using Naive Bayes Equation to Calculate the Posterior Probability for a Sample Class of Vulnerability, to Predict its Severity of Exploitation

Let A be a sample vulnerability with CVSS parameters as:
<Medium, Local, Low, None, Partial, Partial, Partial>

The posterior probability of sample class A, for given set of CVSS parameters, is calculated from table 3 as:

$$P(A/High) \times P(High) = P(Medium/High) \times P(Local/High) \times P(Low/High) \times P(None/High) \times P(Partial/High) \times P(Partial/High) \times P(Partial/High) \times P(High)$$

$$P(A/High) \times P(High) = (1/2) \times (1/4) \times (3/4) \times (3/4) \times (1/4) \times (1/6) \times (1/12) \times (4/7)$$

$$P(A/High) \times P(High) = 36/258048$$

$$P(A/High) \times P(High) = 0.0001395089$$

Now we will calculate P (A/Low) × P (Low) as:

$$P(A/Low) \times P(Low) = P(Medium/Low) \times P(Local/Low) \times P(Low/Low) \times P(None/Low) \times P(Partial/Low) \times P(Partial/Low) \times P(Partial/Low) \times P(Low)$$

$$P(A/Low) \times P(Low) = (9/9) \times (0) \times (1/3) \times (1) \times (5/9) \times (7/9) \times (1/3) \times (3/7)$$

$$P(A/Low) \times P(Low) = 0$$

Now the highest posterior probability is calculated to be:

$$MAX\{P(A/High) \times P(High), P(A/Low) \times P(Low)\} = MAX\{0.0001395089, 0\}$$

$$MAX\{P(A/High) \times P(High), P(A/Low) \times P(Low)\} = 0.0001395089$$

Since {P (A/High) × P (high)} is evaluated to be greater than {P (A/Low) × *P* (Low)}, hence the sample vulnerability class A with the CVSS parameters as:
< Medium, Local, Low, None, Partial, Partial, Partial> is predicted to pose high threat of exploitation and thus should quickly be reported for immediate remediation, to prevent the hackers from stealing the valuable data.

V. CONCLUSION AND RECOMMENDATIONS

A. Conclusions

All Heartbleed-vulnerable systems should immediately upgrade to OpenSSL 1.0.1g. If we are not sure whether an application we want to access is Heartbleed vulnerable or not – we should try any one of the Heartbleed detector tools. No action required, if application that we are using, is not vulnerable. But if the application is vulnerable, wait for it to be patched with OpenSSL 1.0.1g. Once the patch is applied, all the users of such applications should follow the application's release documents from the service providers. Typically, steps to follow once the patch is applied are:

1) changing our password
2) generating private keys again
3) certificate revocation and replacement

An important step is to restart the services that are using OpenSSL (like HTTPS, SMTP etc.). Before accessing any SSL/TLS application such as HTTPS, check to see if the application is vulnerable. Do not access or login to any affected sites. Ensure all such vendors or enterprises related to your business have applied this security patch. Keep your eyes open on such news of security vulnerabilities[7].

The Heartbleed bug has shaken the Internet community on its dependency on the open source software. Even though OpenSSL is a very popular library, it was not properly scrutinized. One reason might be because of lack of resources and funds. The organizations and developers using open source software should contribute back to these open source communities in terms of donations, reviewing the code, testing and designing. Amazon, Facebook, Google have recently come forward to donate funds to improve open-source security systems [6].

Naive Bayes Classification enables us to prioritize vulnerabilities for remediation. The type of vulnerabilities which are classified as highly exploitable by the proposed methodology ,can be easily exploited with minimum efforts by the hackers, therefore the particular vulnerability needs headlong attention and should be remediated & fixed as early as possible, to prevent the exploitation of any kind.

B. Recommendations

To obtain the fix in your application simply upgrade to OpenSSL 1.0.1g.

If upgrading is not practical, we can rebuild our current version of OpenSSL from source without

TLS Heartbeat support by adding the following compile switch:

-DOPENSSL_NO_HEARTBEATS

This switch ensures that the defected code never gets executed.

An effective vulnerability assessment and remediation program must be able to prevent the exploitation of vulnerabilities by detecting and remediating vulnerabilities in covered devices in a timely fashion. Proactively managing vulnerabilities on covered devices will reduce or eliminate the potential for exploitation and save on the resources otherwise needed to respond to incidents after exploitation has occurred. Information Security and Policy (ISP) provides a centrally managed campus service that campus units can use to comply with this requirement [2].

REFERENCES

[1]. Yogesh Joshi, Debabrata Das, Subir Saha, "Mitigating Man in the Middle Attack over Secure Sockets Layer," IEEE, pp 1-5, 2009.
[2]. Eman Salem Alashwali , "Cryptographic Vulnerabilities in Real-Life Web Servers," In Proceedings of the The 3rd International Conference on Communication and Information Technology(ICCIT-2013):Digital Information Management and Security Beirut, pp 6-11, 2013.
[3]. Krishna Kant and Ravishankar Iyer, Prasant Mohapatra, "Architectural Impact of Secure Socket Layer on Internet Servers: A Retrospect," IEEE pp 25-26, 2012.
[4]. Neal Leavitt "Internet Security under Attack: The Undermining of Digital Certificates," IEEE Computer Society, pp 17-20, 2011.
[5]. University of Maryland; cybersecurity experts discover lapses in heartbleed bug fix. (2014). NewsRx Health & Science, Retrieved from http://search.proquest.com. jproxy.lib.ecu.edu/docview/1626397458?accountid=10639
[6]. OpenSSL Team" OpenSSL Project," openssl.org, 2014 [Online]. Available: https://www.openssl.org/ [Accessed: June. 12, 2014]
[7]. CODENOMICON "The Heartbleed Bug," heartbleed.com , 29 April 2014 [Online]. Available: http://heartbleed.com/. [Accessed: June. 12, 2014].
[8]. Rish, Irina (2001). An empirical study of the naive Bayes classifier(PDF). IJCAI Workshop on Empirical Methods in AI.
[9]. "Common Vulnerability Scoring System, V2 Development Update". First.org, Inc. Retrieved November 13, 2015.
[10]. My paper publication links (OpenSSL Heartbleed vulnerability & prediction of severity of exploitation posed by some of the common types of vulnerabilities, based on Common Vulnerability Scoring System (CVSS), using Naive Bayes classification algorithm):
https://archive.org/details/MehakBashir
http://www.ijtre.com/images/scripts/2017040917.pdf
Google Scholar link:
http://scholar.google.co.in/scholar?hl=en&q=Mehak+Bashir

