SKIP A HEARTBEAT: OPENSSL HEARTBLEED VULNERABILITY & PREDICTION OF EXPLOITATION

BASED ON CVSS USING NAIVE BAYES ALGORITHM

Mehak Bashir

SKIP A HEARTBEAT: OPENSSL HEARTBLEED VULNERABILITY & PREDICTION OF EXPLOITATION BASED ON CVSS USING NAIVE BAYES ALGORITHM

Er. Mehak Bashir

MASTER OF TECHNOLOGY

IN

COMPUTER SCIENCE ENGINEERING

Under the Supervision of:

Dr. MUHEET AHMED BUTT (Scientist 'D', Kashmir University)

Dr. MAJID ZAMAN BABA (Scientist 'D', Kashmir University)

Prof. HARKESH SEHRAWAT (Professor, Maharishi Dayanand University)

ACKNOWLEDGEMENTS

This book would not have been possible without the input and support provided by several people. The author would like to express a deep sense of gratitude and special thanks to Dr. Muheet Butt, Scientist ‘D’, Kashmir University. It would have been impossible to complete this work in this manner without his wise counsel and able guidance.

The constant guidance and encouragement received from Dr. Majid zaman, Scientist ‘D’, Kashmir University, who has been very helpful in completing the book, for which the author is highly indebted to his good-self.

The author expresses profound gratitude to Prof. Harkesh Sehrawat, Professor, Maharishi Dayanand University, for his intellectual support throughout the course of this work.

Finally, the author would like to express thanks to the many silent professionals whose work, advice, and feedback influenced my thinking and work on this book. Despite the efforts of so many to make this report better, there are undoubtedly weaknesses and errors that remain. These are entirely the fault of the author.

Mehak

Mehak Bashir

TABLE OF CONTENTS

Topic	Page No.
LIST OF FIGURES	**iv**
LIST OF TABLES	**iv**
ABSTRACT	**v**
CHAPTER 1 INTRODUCTION	**1 -12**
1.1 OpenSSL Heartbleed	1
1.2 Naive Bayes Classifier	2
1.3 Vulnerability	3
1.4 Types of Vulnerabilities	
CHAPTER 2 LITERATURE SURVEY	**13 - 17**
2.1 Introduction to Survey Report	13
2.2 General Survey	16
CHAPTER 3 THEORETICAL EXPLAINATIONS	**18 - 30**
3.1 How the Heartbeat Works	18
3.2 Data Leakage Leading to Heartbleed	19
3.3 Code Fix	21
3.4 Real world Impact of Heartbleed	22
3.5 Factors to Determine Severity of a Vulnerability- Common Vulnerability Scoring System (CVSS)	24
3.6 Naive Bayes Classification	25
CHAPTER 4 PROPOSED WORK	**31 - 37**
4.1 Algorithm for Predicting severity/Threat of Exploitation Using Naïve Bayes Approach	31
4.2 Frequency Table for Some Common Vulnerabilities Based on CVSS (Version 2) parameters	31
4.3 Likelihood Table for Finding the Probabilities of Various CVSS (Version2) Parameters	34
4.4 Using Naive Bayes Equation to Calculate the Posterior Probability for a Sample class of Vulnerability, to predict its Severity	36

TABLE OF CONTENTS

Topic	Page No.
CHAPTER 5 RESULTS AND OUTPUTS	**38 - 42**
5.1 Checking Heartbleed Vulnerability with nmap in Kali Linux	38
5.2 Exploiting Heartbleed Vulnerability with Metasploit	39
5.3 Output of Naive Bayes Method for Prediction of Severity of Exploitation for OpenSSL Heartbleed Vulnerability	41
5.4 C# Code Segments for Predicting Severity/Threat of Exploitation Using Naive Bayes Approach	42
CHAPTER 6 CONCLUSION AND RECOMMENDATIONS	**48 - 50**
6.1 Conclusions	48
6.2 Recommendations	49
CERTIFICATE OF PUBLICATION	**51**
REFERENCES	**52**

LIST OF FIGURES

S.No.	Title	Page No.
Figure 0	Graphic 1 and 2 shows the Heartbleed code	17
Figure 1	Memory Leak	20
Figure 2	The OpenSSL code fix for the Heartbleed bug	21
Figure 3	OpenSSL Security Advisory	22
Figure 4	Exploiting the Heartbleed Vulnerability	23

LIST OF TABLES

S.No.	Title	Page No.
Table 1	CVSS (Version 2) Base Metrics, with definitions from Mell et al. (2007)	25
Table 2	Frequency table for some common vulnerabilities using CVSS (Version 2) Base Metrics	32 - 34
Table 3	Likelihood table for calculation of probabilities of CVSS (Version 2) Parameters	35

ABSTRACT

The Open Secure Sockets Layer (OpenSSL) provides secure platform for transactions, such as online shopping, online banking and emails etc., that take place over/across the internet. It is widely used open source implementation of the Secure Sockets Layer (SSL) and Transport Layer Security (TLS). Vulnerabilities have however been found in the OpenSSL which has resulted in a wide public outcry all over the world. A confounding computer bug called “Heartbleed” is causing major security worries across the internet. Heartbleed affects many things, including web servers, routers that connect office networks to the internet, mobile apps and VPNs (Virtual Private Network). It has been estimated that 60 percent of secure web sites that are using OpenSSL are affected. In addition, Heartbleed cannot be traced. The Heartbleed Bug has sent shockwaves all over the internet. Not only has all of this user data been directly compromised, but, what are worse, the private keys of the servers running the vulnerable versions of OpenSSL were also almost certainly compromised. Patching of affected applications or/and upgrade to versions that are not vulnerable, is recommended/suggested, in order to mitigate the risks identified.

The thesis/work 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.

CHAPTER 1

INTRODUCTION

1.1 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.

1.2 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:

$$\begin{aligned}\mathbf{MAP(H)} &= \max(P(H|E)) \\ &= \max((P(E|H)*P(H))/P(E)) \\ &= \max(P(E|H)*P(H))\end{aligned}$$

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].

1.3 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].

1.4 Types of Security Vulnerabilities

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

- buffer overflows
- unvalidated input
- race conditions
- access-control problems
- weaknesses in authentication, authorization, or cryptographic practices

1.4.1 Buffer Overflows

A buffer overflow occurs when an application attempts to write data past the end (or, occasionally, past the beginning) of a buffer.

Buffer overflows can cause applications to crash, can compromise data, and can provide an attack vector for further privilege escalation to compromise the system on which the application is running.

Books on software security invariably mention buffer overflows as a major source of vulnerabilities. Exact numbers are hard to come by, but as an indication, approximately 20% of the published exploits reported by the United States Computer Emergency Readiness Team (US-CERT) for 2004 involved buffer overflows.

Any application or system software that takes input from the user, from a file, or from the network has to store that input, at least temporarily. Except in special cases, most application memory is stored in one of two places:

- *stack*— A part of an application's address space that stores data that is specific to a single call to a particular function, method, block, or other equivalent construct.
- *heap*— General purpose storage for an application. Data stored in the heap remains available as long as the application is running (or until the application explicitly tells the operating system that it no longer needs that data).

Class instances, data allocated with malloc, core foundation objects, and most other application data resides on the heap. (Note, however, that the local variables that actually point to the data are stored in the stack.)

Buffer overflow attacks generally occur by compromising the stack, the heap, or both [1].

1.4.2 Unvalidated Input

As a general rule, we should check all input received by our program to make sure that the data is reasonable.

For example, a graphics file can reasonably contain an image that is 200 by 300 pixels, but cannot reasonably contain an image that is 200 by -1 pixel. Nothing prevents a file from claiming to contain such an image, however (apart from convention and common sense). A naive program attempting to read such a file would attempt to allocate a buffer of an incorrect size, leading to the potential for a heap overflow attack or other problem. For this reason, we must check our input data carefully. This process is commonly known as input validation or sanity checking.

Any input received by our program from an untrusted source is a potential target for attack. (In this context, an ordinary user is an untrusted source.) Examples of input from an untrusted source include (but are not restricted to):

- *text input fields*
- *commands passed through a URL used to launch the program*
- *audio, video, or graphics files provided by users or other processes and read by the program*
- *command line input*
- *any data read from an untrusted server over a network*
- *any untrusted data read from a trusted server over a network (user-submitted HTML or photos on a bulletin board, for example)*

Hackers look at every source of input to the program and attempt to pass in malformed data of every type they can imagine. If the program crashes or otherwise misbehaves, the hacker then tries to find a way to exploit the problem. Unvalidated-input exploits have been used to take control of operating systems, steal data, corrupt users' disks, and more. One such exploit was even used to "jail break" iPhones [1].

Validating Input and Inter-process Communication describes common types of input-validation vulnerabilities and what to do about them.

1.4.3 Race Conditions

A race condition exists when changes to the order of two or more events can cause a change in behavior. If the correct order of execution is required for the proper functioning of the program, this is a bug. If an attacker can take advantage of the situation to insert malicious code, change a filename, or otherwise interfere with the normal operation of the program, the race condition is security vulnerability. Attackers can sometimes take advantage of small time gaps in the processing of code to interfere with the sequence of operations, which they then exploit [2].

1.4.4 Inter-process Communication

Separate processes—either within a single program or in two different programs—sometimes have to share information. Common methods include using shared memory or using some messaging protocol, such as Sockets, provided by the operating system. These messaging protocols used for inter-process communication are often vulnerable to attack; thus, when writing an application, we must always assume that the process at the other end of our communication channel could be hostile.

1.4.5 Insecure File Operations

In addition to time-of-check–time-of-use problems, many other file operations are insecure. Programmers often make assumptions about the ownership, location, or attributes of a file that might not be true. For example, we might assume that we can always write to a file created by our program. However, if an attacker can change the permissions or flags on that file after we create it, and if we fail to check the result code after a write operation, we will not detect the fact that the file has been tampered with.

Examples of insecure file operations include:

- *writing to or reading from a file in a location writable by another user*
- *failing to make the right checks for file type, device ID, links, and other settings before using a file*

- *failing to check the result code after a file operation*
- *assuming that if a file has a local pathname, it has to be a local file*

1.4.6 Access Control Problems

Access control is the process of controlling who is allowed to do what. This ranges from controlling physical access to a computer—keeping our servers in a locked room, for example—to specifying who has access to a resource (a file, for example) and what they are allowed to do with that resource (such as read only). Some access control mechanisms are enforced by the operating system, some by the individual application or server, some by a service (such as a networking protocol) in use. Many security vulnerabilities are created by the careless or improper use of access controls, or by the failure to use them at all.

Much of the discussion of security vulnerabilities in the software security literature is in terms of privileges, and many exploits involve an attacker somehow gaining more privileges than they should have. Privileges, also called permissions, are access rights granted by the operating system, controlling who is allowed to read and write files, directories, and attributes of files and directories (such as the permissions for a file), who can execute a program, and who can perform other restricted operations such as accessing hardware devices and making changes to the network configuration.

Of particular interest to attackers is the gaining of root privileges, which refers to having the unrestricted permission to perform any operation on the system. An application running with root privileges can access everything and change anything. Many security vulnerabilities involve programming errors that allow an attacker to obtain root