International Journal of Scientific Research in Science, Engineering and Technology
Print ISSN: 2395-1990 | Online ISSN : 2394-4099 (www.ijsrset.com)
doi : https://doi.org/10.32628/IJSRSET

Buffer Overflow Detection and Avoidance Technique

P. B. Niranjane
Department of Computer Science & Engineering, Babasahe Naik College of Engineering , Pusad, Maharastra, India

ARTICLEINFO

Article History:

Accepted: 10 April 2023
Published: 29 April 2023

Publication Issue
Volume 10, Issue 2
March-April-2023

Page Number
726-731

ABSTRACT

Today's world totally work through important asset i.e. information and there should protective shield to protect such asset with good performance. Therefore Security issue is the important and elicited topic among IT professionals. The Buffer Overflow which is one of the most frequently occurring security vulnerabilities on network. Buffer Overflow occurs while writing data to a buffer and it overruns the buffer's threshold and overwrites it to neighboring memory. This paper focused on finding and detection of buffer overflow occurs during transmission of jpg and gif file format over network. The signature and signature free detection mechanism is implemented for detection of buffer overflow during transmission. First the signature based detection finds a particular signature after the signature free technology is implanted for testing the buffer overflow. The discussion is concluded with some thoughts on buffer overflow detection in general, and directions for the analysis and remediation of buffer overflow detection for gif and jpg file format.

Keywords: Security, Attacks, Buffer-Overflow, Signature, Signature free, Malicious code, Intrusion

I. INTRODUCTION

Buffer overflow is a problem where a program tries to store a string of arbitrary length in a fixed size buffer, without checking for whether the string can fit inside the buffer. It results in inadvertently overwriting the memory location that follows the buffer. If the buffer resides in the data section, it could corrupt other global variables [06]. If the buffer is heap allocated, it could corrupt data structure used by memory management routines. If the buffer resides on the stack the overflow could overwrite the stack frame, including the return address, which can alter the program's control flow. In the context of networked programs, buffer overflow allows the execution of arbitrary code injected by a remote client or peer. This could result in compromise of sensitive data. If the buffer overflow happens in an operating system kernel, it could lead to privilege escalation. When remote code injection is combined with privilege escalation, it could result in the compromise of the whole system.

Any C program that manipulates strings may be prone to buffer overflow problem if care is not taken. In fact, we can argue that buffer overflow is almost a C language feature, inherent in the design of the C language [6]. It has to do with the way strings are represented: a character array of indefinite length, until it is terminated by a NUL character (which has the numeric byte value of 0). C does not distinguish an array from a pointer, so string arguments to function typically bear the type char * or const char *. However, memory resources in C all have definite length. The programmer has to be careful when juggling possibly infinite sequence of characters in a finite amount of space.

Buffer overflow attack is an attack in which a malicious user exploits an unchecked buffer in a program and overwrites the program code with own data inject gif and jpg file. If the program code is overwritten with new executable malicious code, the effect is to change the program's operation as dictated by the hacker. If overwrite with other data, the likely effect is to direct the program to crash [1]. Today‛s software has been widely targeted by buffer overflows. Detecting and eliminating buffer overflows would thus make server far more secure.

A buffer temporarily stores data waiting to be processed by a program. A buffer overflow occurs when more data is inserted into the buffer than the buffer was intended to hold. Buffer overflows occur because languages or programmers do not have or perform adequate bounds checking. Since detection is based on pattern matching, a signature modelling the attack must exist for the IDS to detect it, and therefore it is capable of detecting known attacks. This paper defines a method to detect buffer overflow attacks by parsing the payload of network packets in search of IP address which is the remotely executable component of a buffer overflow attack. By analyzing the IP it is possible to determine which system calls the unauthorized uses, and hence the operation of the exploit. Current network-based buffer overflow detection techniques mainly rely upon specific signatures for each new attack. Our approach is capable to detect previously unseen buffer overflow attacks, in addition to existing ones, with the need for specific signatures and Signature free for each new attack related to jpg and gif file format

II. RELATED WORK

Methods for detecting buffer overflow vulnerabilities can be divided into three groups static or compile time detection, host based detection, and network based detection. A compile time solution has been proposed in [3]. The solution given by authors was the development of a static analysis tool that analyses application source code in search of likely buffer overflow vulnerabilities.

This solution is capable of improving an application by eliminating possibilities of successfully executing buffer overflow attacks, but it requires modification to the source code and recompilation to work in addition to the requirement of source availability.

Stack Guard is an extension to the freely available and very popular gcc compiler that allows detection or prevention of alterations of the return address of a stack frame. Detection is executed by inserting a random word value immediately following the return address for the process on the stack. This value is confirmed when the process returns. Since it is difficult to alter the return address without altering the following bytes, this method is capable of detecting buffer overflow attacks. Some study suggests that the protection mechanism may still be circumvented by exploiting function pointers or "long jumps" [7].

III. METHODOLOGY

The proposed technique is Signature and signature free for detecting buffer overflow with jpg and gif file formats. Firstly check the signature based detection.

This technique capable of detecting malicious code by applying Pattern matching scheme on Signature, here IP address is chosen as a signature in our study. If the signature is found block that request remaining request are send to the signature free detection techniques.

In signature free detection techniques the log file from server contains the all information about the requests along with necessary metadata. Original image is separated out from malicious image by detecting the buffer overflow occurred during transmission over the network with Signature and Without Signature. Each time during receiving end we check for buffer overflow and if found then we are blocking that images and forwarding the remaining images to server for processing. Figure 4.1 shows the architecture of our purposed work. The request from client is verified on Signature based detection if the signature is found block that request remaining request are send to the Signature free detection techniques.

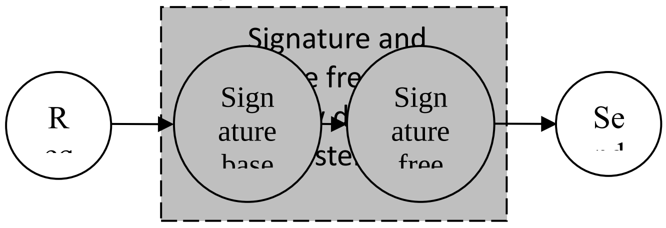

Figure 1: Architecture for Proposed Methodology

In Signature free detection are block the unwanted request by using buffer overflow detection techniques and the valuable request are send to server. In a typical buffer overflow attack, the attacker injects malicious code into the victim application and transfers the control of the application to the injected code. As an application's text segment is typically read-only, the only way to hijack the control of an application is to dynamically modify the target address of its branch instructions whose target is not fixed at compilation time. Such dynamic branch instructions include function returns, pointer-based function calls, and C-style switch statements. These branch instructions typically have their target addresses stored in some stack or heap.

The whole implementation is divided into two main modules i.e. signature based detection module and signature free detection module for buffer overflow detection of jpg and gif file format.

The signature based detection only check particular signature in the request generated by client if the signature is found in the request then the request containing that signature is blocked and remaining requests are send to signature free detection in that all focus is given on finding the buffer overflow with various intermediate steps.

A. Signature Based Detection

The request from the client is first Encoded based on UTF-8 Scheme the duplicates of the request is blocked these remaining request does not contain duplicates is now checked for the IP address defined on server. If the generated request from client contains the signature then it is immediately blocked and remaining request are send to signature free detection of buffer overflow. Any request depending on IP address can be block just by modifying the IP address on server [1]. UTF stands for Unicode Transformation Format. The '8' means it uses 8-bit blocks to represent a character. The number of blocks needed to represent a character varies from 1 to 4. UTF-8 is a compromise character encoding that can be as compact as ASCII (if the file is just plain English text) but can also contain any Unicode characters (with some increase in file size).

B. Signature Free Detection

The request from signature based detection is input for signature free detection the various steps involved in detecting buffer overflow for gif jpg file format is categorized in to following modules.

1) Data Collection & Pre-processing
2) URL decoder
3) ASCII Filter
4) Instruction distiller

5) Instruction Sequence Analysis

1) Data Collection & Pre-processing
   Signature free is going to apply over real traces of a web server [01]. The requests include manually typed URLs, clicks through various web sites, searching's from search engines such as Google and Yahoo, secure logins to email servers and bank servers, and HTTPs requests. In this way, we believe our data set is diverse enough, not worse than that we might have got if we install Signature free in a single web server that provides only limited Internet services.
2) URL decoder
   The specification for URLs limits the allowed characters in a Request-URL to only a subset of the ASCII character set. This means that the query parameters of a request-URL beyond this subset should be encoded. Because a malicious payload may be embedded in the request-URL as a request parameter, the first step of Signature free is to decode the request-URL.
3) ASCII Filter
   Malicious executable codes are normally binary strings. In order to guarantee the throughput and response time of the protected web system, if a request is printable ASCII ranging from 20 to 7E in hex, Signature free allows the request to pass. Note that ASCII filter does not prevent the service from receiving non-ASCII strings.

4) Instruction distiller
   To distil an instruction sequence, we first assign an address (starting from zero) to every byte of a request, where address is an identifier for each location in the request. Then, we disassemble the request from a certain address until the end of the request is reached or an illegal instruction code is encountered.
   The recursive traversal algorithm is used, because it can obtain the control flow information during the disassembly process. Intuitively, to get all possible instruction sequences from an N-byte request, simply execute the disassembly algorithm N times and each time start from a different address in the request. This gives a set of instruction sequences.
5) Instruction Sequence Analysis
   A distilled instruction sequence may be a sequence of random instructions or a fragment of a program in machine language. In this section, three schemes are proposed to differentiate these two cases. Scheme 1 exploits the operating system characteristics of a program; Scheme 2 and Scheme 3 exploit the data flow characteristics of a program.

A program in machine language is dedicated to a specific operating system; hence, a program has certain characteristics implying the operating system on which it is running, for example calls to operating system or kernel library. A random instruction sequence does not carry this kind of characteristics. By identifying the call pattern in an instruction sequence, a real program from a random instruction sequence can effectively differentiated.

The detection of data flow anomaly is used in a different way called code abstraction. It is observed that that when there are data flow anomalies in an execution path of an instruction sequence some instructions are useless, whereas in a real program at least one execution path has a certain number of useful instructions. Therefore, if the number of useful instructions in an execution path exceeds a threshold, it concludes the instruction sequence is a segment of a program.

C. RESULTS

Specifically in the Signature based detection, first the signature is verified which is defined on server to block unauthorized request. These verified requests then filtered on basis signature. And requests then check against the buffer size depending on the buffer size the requests are then forwarded to server for processing.

In the Signature free technique, split URL and remove the duplicate of the requests. The encode URL is now decoded on the basis of UTF-8 Scheme and then these requests are distilled. The ASCII conversion is done on the distilled requests and the requests are filtered and checked against the Buffer size. If the Buffer size required by request is more than threshold value then the request is blocked, otherwise it is forwarded to server

The result is obtained by sending request to buffer overflow detection based on signature. Here the request not containing the defined signature is allowed to pass for signature free detection and all other requests gets blocked. The signature free detection first split the request which do not contain jpg and gif file. Only the request containing jpg and gif file format are considered for further processing.

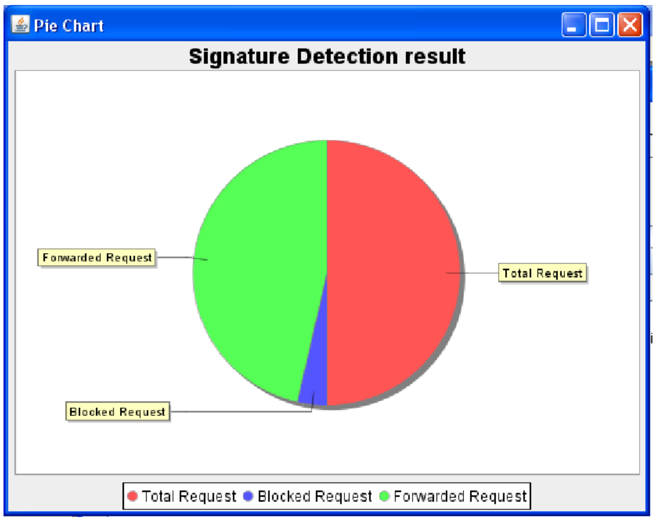

Figure 2: Signature detection chart

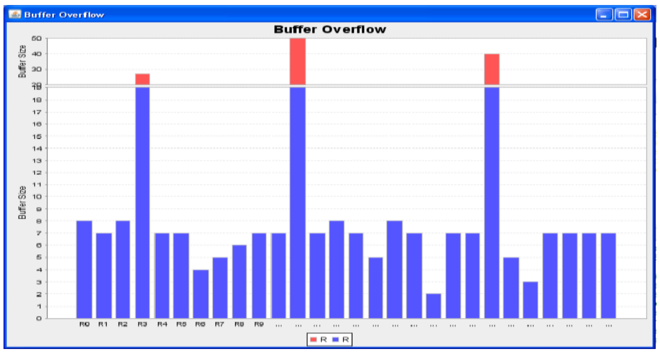

Figure 3: Signature free graph

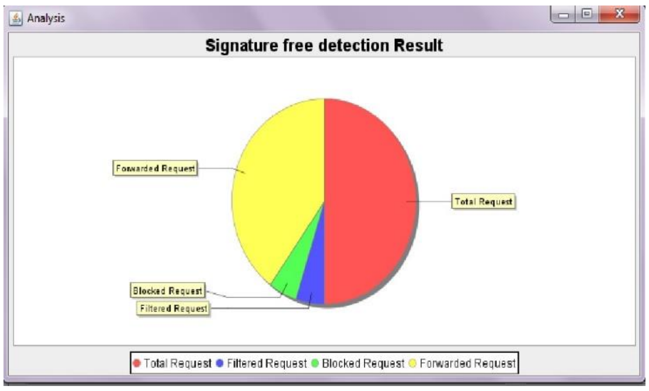

Figure 4: Signature free detection chart

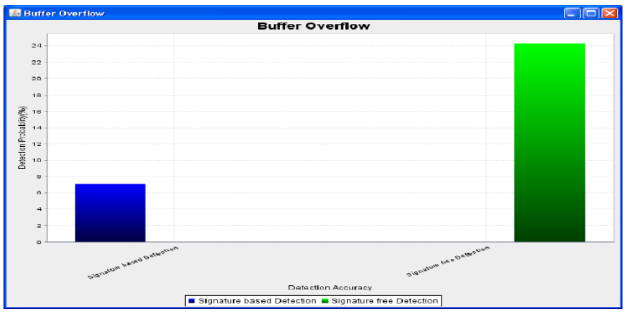

Figure 5: Analysis of Signature & Signature free graph

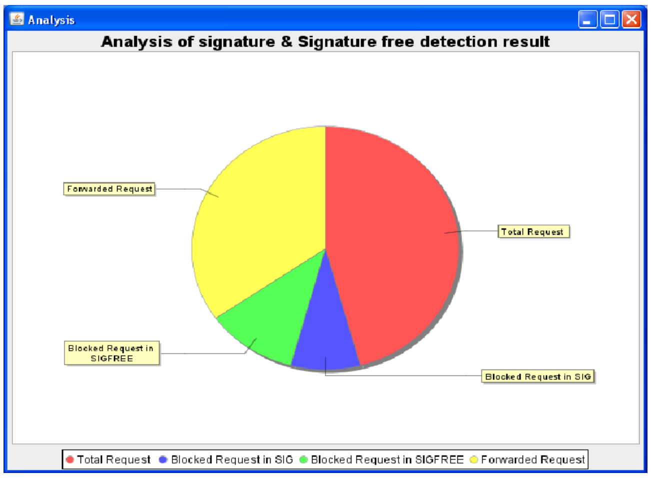

Figure 6: Analysis of Signature & Signature free chart

Figure 2 shows behavior of requests against the Buffer size for Signature based detection. Figure 4 shows behavior of requests against the Buffer size for Signature free based detection and Figure 6 shows Analysis requests blocked against the Buffer size for Signature free technique for jpg and gif file format. If the buffer size required by request is more than the threshold value the then requests are blocked, otherwise it is forwarded. Figure 3 and 5 shows the analysis of buffer overflow detection with Signature and without Signature for jpg and gif formats.

IV.CONCLUSION

The proposed Signature and signature-free buffer overflow detection for jpg and gif systems that can filter code-injection buffer overflow attack, one of the most serious cyber security paradigms. The Signature based buffer overflow detection finds the particular Signature and if that found it blocks it to protect form malicious attack. Signature free does not require any signatures, thus it can block new malicious code and provide security for the systems. With the used techniques maximum requests can be blocked using Signature free technique rather than signature based. Signature and Signature free is less affected from malicious attack, and easy for deployment, low performance overhead with less maintenance cost. In the future expansion the various parameters of image can be consider for detecting malicious code in image. Wavelet transformer can be applied for detecting malicious code accurately.

V. REFERENCES

[1]. Z. Liang and R. Sekar, —Fast and Automated Generation of Attack Signatures: A Basis for Building Self-Protecting Servers, Proc. 12th ACM Conf. Computer and Comm. Security (CCS), 2005.

[2]. B.A. Kuperman, C.E. Brodley, H. Ozdoganoglu, T.N. Vijaykumar, and A. Jalote, "Detecting and Prevention of Stack Buffer Overflow Attacks," Comm. ACM, vol. 48, no. 11, 2005

[3]. D. Evans and D. Larochelle, "Improving Security Using Extensible Lightweight Static Analysis," IEEE Software, vol. 19, no. 1, 2002.

[4]. Xinran Wang, Chi-Chun Pan, Peng Liu, and Sencun Zhu, "Signature free: A Signature-Free Buffer Overflow Attack Blocker", Ieee Transactions On Dependable And Secure Computing, Vol. 7, No. 1, January-March 2010.

[5]. J. Pincus and B. Baker, "Beyond Stack Smashing: Recent Advances in Exploiting Buffer Overruns," IEEE Security and Privacy, vol. 2, no. 4, 2004.

[6]. Eric Haugh and Matt Bishop, "Testing C Programs for Buffer Overflow Vulnerabilities". University of Californai Devis. 2004

[7]. O. Ruwase and M. Lam. A practical dynamic buffer overflow detector. In Proceedings of Network and Distributed System Security Symposium, pages 159–169, 2004.

Cite this article as :

P. B. Niranjane, "Buffer Overflow Detection and Avoidance Technique", International Journal of Scientific Research in Science, Engineering and Technology (IJSRSET), Online ISSN : 2394-4099, Print ISSN : 2395-1990, Volume 10 Issue 2, pp. 726-731, March-April 2023. Available at doi : https://doi.org/10.32628/IJSRSET23102126
Journal URL : https://ijsrset.com/IJSRSET23102126