Sigfree: Buffer Overflow Attack Detection

P. H. Rathod , S. N. Dhage
Computer Engineering, MGM COE/Mumbai, India
Computer Engineering, SPIT/Mumbai, India

ABSTRACT: Sigfree act as a Defender or protective shield for the today’s Information world from insider or outsider attacker. Any activity aimed at disrupting a service or making a resource unavailable or gaining unauthorized access can be termed as an intrusion. Buffer Overflow attack is an attack which gives rise to other attacks as DOS attack. Buffer overflow is activated when a large amount of data is copied into fixed size memory. Buffer overflow attack gives opportunity to the attacker to change the return address of the function by some malicious content. In this paper we have proposed a system Sigfree which has two approaches. First approach analyses and distills machine code and second approach removes the data anomaly. Sigfree blocks the new and unknown attack. Sigfree uses the code abstraction method which separates the data and executable separately. Sigfree is generic i.e. Sigfree does not have the predefined signature or pattern for matching with worm signature. It generates the signature during the runtime. Sigfree detect the illegal or external instruction by monitoring the network traffic. Sigfree uses the recursive traversal algorithm with its ability to deal intelligently with control flow. Sigfree has low deployment and maintenance cost. Sigfree cuts down the false positive and false negative rate as compared to other network based or host based IDS.

Keywords: Buffer Overflow Attack, Dynamic Techniques, Intrusion detection System, Static Techniques.

I. INTRODUCTION

Information plays a vital role in today’s information world. Tons of research has been done to protect information from insider and outsider attacker. Many new and unknown attacks has been occurred from last decade. Buffer overflow attack is one of the most occurring attack. Buffer overflow is a large amount of data is copied into fixed size memory. Huge amount of vulnerabilities exist in each domain (Operating System, Databases, Network Applications etc) of computer world. From all the domains we have seen so far most common vulnerabilities are Buffer overflow and Format String vulnerabilities. Buffer overflow attack occurred due to sloppy coding. When Buffer overflow occurs return address of the function is being altered by an attacker by external address and malicious code is executed points to that address. Different Intrusion detection system has been developed for detecting buffer overflow and different kinds of attack. They deal with low positive rate and low false negative rate. There were different Intrusion detection system which were categories based on their features. They were categories into different classes as follows 1)Techniques depends on source code 2)Techniques need to modify the operating system or hardware 3)Techniques by analysing symptoms of attack. 4) Techniques use code obfuscation method. 5) Techniques need to extend the features of compiler. There were static as well as Dynamics tools and technique to make a network attack free. Static analysis tools have unacceptably high false alarm rates and insufficient detection rates [1]. Dynamic buffer overflow detection and prevention is an attractive approach, because fundamentally there can be no false alarms.

Tools that provide dynamic buffer overflow detection can be used for a variety of purposes, such as preventing buffer overflows at runtime, testing code for overflows, and finding the root cause of segfault behaviour. Chaperon [2] works directly with binary executable and thus can be used when source code is not available. ProPolice [3] is similar to StackGuard [4], and performed it on artificial exploits. It works by inserting a “canary” value between the local variables and the stack frame whenever a function is called. It also inserts appropriate code to check that the “canary” is unaltered upon return from this function. The “canary” value is picked randomly at compile time, and extra care is taken to reorder local variables such that pointers are stored lower in memory than stack buffers. The “canary” approach provides protection against the classic “stack smashing attack” [5]. TinyCC [6] works by inserting code to check buffer accesses at compile time; however, the representation of pointers is unchanged, so code compiled with TinyCC can interoperate with unchecked code compiled with gcc. Insure++ examines source code and inserts instrumentation to check for memory corruption, memory leaks, memory allocation errors and pointer errors, among other things. The resulting code is executed, and errors are reported when they occur.

In this paper we have proposed Technique called Sigfree i.e Signature free buffer overflow attack which blocks the new and unknown attacks [7]. Sigfree has certain advantages over the previous tools and techniques. Sigfree does not require the source code but sigfree works on machine code. Sigfree do not do changes on the server side i.e. transparency is high. Modification done on Server side, operating system, or on hardware is
Sigfree: Buffer Overflow Attack Detection

Sigfree is a tool that aims to detect and prevent buffer overflow attacks. It uses a combination of different methods to achieve this goal. The methods include:

1. Buffer Overflow Detection and Prevention Method
2. Worm detection and signature generation
3. Machine code analysis

Buffer overflow is detected by identifying errors in the source code due to sloppy coding. C and C++ are languages that do not perform bound checking, meaning they do not check the size of data being copied to memory, which can lead to buffer overflow attacks. Tools like Pax, LibSafe, and e-NeXsh extend the functionality of the compiler to detect buffer overflow attacks. Other tools like ProPolice and Return Address Defender (RAD) provide similar capabilities.

Capturing code running symptoms and achieving 100% coverage in capturing buffer overflow symptoms is challenging. Techniques such as Vigilante, TaintCheck, and program shepherding are used to address this issue.

Sigfree uses code abstraction techniques to separate data from executable code. It employs a new O(N) algorithm to disassemble and distill all possible instruction sequences from the message's payload, considering every byte as a possible starting point of the code embedded.

II. OVERVIEW OF SIGFREE

Sigfree uses three different methods:

1. Buffer Overflow Detection and Prevention Method
2. Worm detection and signature generation
3. Machine code analysis

Buffer overflow is detected by different techniques such as finding errors in the source code due to sloppy coding. C and C++ are languages that do not perform bound checking, which leads to buffer overflow attacks. Tools like Pax, LibSafe, and e-NeXsh, as well as some compilers like ProPolice and Return Address Defender (RAD), provide functionality to detect buffer overflow attacks.

Capturing code running symptoms and achieving 100% coverage in capturing buffer overflow symptoms is challenging. Techniques like Vigilante, TaintCheck, and program shepherding are used to address this issue.

Sigfree works on machine code rather than source code. It filters out illegal data coming from various networks with low false positive and false negative rates.

III. SIGFREE ARCHITECTURE

In Fig.3.0, Sigfree is installed on the router between Firewall and network. The packets or data coming from different host systems are filtered out before moving to Firewall. Sigfree uses code abstraction techniques to separate data and executable. SigFree first uses a new O(N) algorithm, where N is the byte length of the message, to disassemble and distill all possible instruction sequences from the message's payload, considering every byte as a possible starting point of the code embedded.
In the above figure 3.0 we have represent the network consists of host system with attacker and victim. At the centre we have router through which the information is being passed. Sigfree is being deployed on the router. Sigfree is located in between firewall and the host system. Sigfree act as online buffer overflow attack Blocker. Sigfree deals with low false positive rate and low false negative rate.

A data flow anomaly is caused by an improper sequence of actions performed on a variable. There are three data flow anomalies: define-define, define-undefine, and undefine-reference [18]. The define-define anomaly means that a variable was defined and is defined again, but it has never been referenced between these two actions. The undefine-reference anomaly indicates that a variable that was undefined receives a reference action. The define-undefine anomaly means that a variable was defined, and before it is used it is undefined.

As a result of the code abstraction of an instruction, a variable could be in one of the six possible states. The six possible states are state U: undefined; state D: defined but not referenced; state R: defined and referenced; state DD: abnormal state define-define; state UR: abnormal state undefine-reference; and state DU: abnormal state define-undefine. Fig. 6 depicts the state diagram of these states. Each edge in this state diagram is associated with d, r, or u, which represents “define,” “reference,” and “undefine,” respectively.
IV. WORKING OF SIGFREE

In figure 4.0 as shown Sigfree is being deployed on the router. So data coming from various host system is being filter out. There are victim as well as attacker system in the network. Sigfree consist of different component i.e. Ascii filter, Sequence instruction distiller and Sequence instruction analyzer. The data coming from various host system is being passed through sigfree component step by step. Ascii filter filters out the Instruction which comes under those ascii value. Those instruction that are not filter is being passed through the instruction sequence distiller. During this phase the instruction flow graph is used with the recursive traversal algorithm. Here illegal and external address is being blocked. But still some of the instruction has the data anomaly and which is then passed through the sequence instruction analyzer where we have used code abstraction technique. Here we have used algorithm to check the data anomaly during the flow of instruction. Data is being separated from executables where executable are blocked and data is passed to the destination host.

Algorithm 1
To block useless, illegal address instruction
initialize EISG G and instruction array A to empty

global startAddr, endAddr;
proc DisasmRec(addr)
begin
while(startAddr <= addr < endAddr)
do
add instruction node i to G
if (addr has been visited already) return;
I := decode instruction at address addr;
mark addr as visited;
if inst is illegal then
A[i] ← illegal instruction inst
set type of node i “illegal node” in G
else
A[i] ← instruction inst
if inst is a control transfer instruction then
for each possible target t of inst do
if target t is an external address then
add external address node t to G
add edge e(node i, node t) to G
else
add edge e(node i, node i + inst.length) to G
i ← i + 1
end

proc main()
V. EXPERIMENT RESULT

In the above figure 5.0 we have plotted the graph where x-axis represents the time in second and y-axis represents the number of node. Above graph gives the throughput of the system. The unwanted data will be blocked at the router. Which increases the performance of the system making host system free from different kind of attack.

VI. CONCLUSION

By implementing the Sigfree we come to conclusion that still we have not achieve 100 % security. There are various approaches to detect the attacks in an Intrusion Detection System. Each of the approaches has its own advantages and disadvantages. Sigfree has low deployment and maintenance cost. Still Sigfree deals low false positive rate and low false negative rate. New techniques keep emerging which will remove the drawbacks of the previous methods of implementation. Thus a judicious approach has to be made while selecting a mode to implement attack detection in an intrusion detection system.

REFERENCES