ournal of Electronicsl and Communication Engineering (IOSR-JECE)
ISSN: 2278-2834-, ISBN: 2278-8735, PP: 54-58
www.iosrjournals.org

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 distils 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,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 false 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 hardware3)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 transparent. Sigfree uses the code abstraction technique where data is separated from malicious executable. Sigfree has the low maintenance and deployment cost. Sigfree analyse the machine code i.e. program is nothing but flow of instruction. Sigfree check the instruction one by one and blocks the unwanted instruction coming from unwanted user. Sigfree assign the address to each instruction and decode the instruction. Sigfree uses the recursive traversal algorithm and code abstraction technique. Server always needs a data instead of executable. So data is being separated from code.Several researchers have developed distributed protocols to detect such traffic manipulations, typically by validating that traffic transmitted by one router is received unmodified by another [8], [9].Generated rules are used to detect network intrusions[10].

The rest of this paper is organized as follows: In Section 2, we give an overview of Sigfree. In Sections 3 and 4, working of the Sigfree with the instruction sequence distiller component and the instruction sequence analyse. In Section 5, we show our experimental results. Finally, we conclude this paper in Section 6.

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 using different techniques such as finding errors in the source code due to sloppy coding [11].C and C++ are the language which do not do bound checking i.e without checking the size of the data it is been copied to memory which will cause buffer overflow attack. Even by modifying operating system or hardware we can detect buffer overflow attack tools such as Pax [12], LibSafe [13], and e-NeXsh [14].Even some tools extends the functionality of the compiler so it can detect buffer overflow attack such as ProPolice [15], and Return Address Defender (RAD) [16]. DIRA [17] is another compiler that can detect control hijacking attacks, identify the malicious input.

Capturing code running symptoms and to achieve 100 percent coverage in capturing buffer overflow symptoms, dynamic data flow/taint analysis/program shepherding techniques were proposed in Vigilante [18], TaintCheck [19], and [20].Considering these features Sigfree is being proposed. Sigfree works on Machine code rather than source code. Sigfree does not require modifying the operating system or any kind of hardware. Sigfree do not extend the features of compiler. Sigfree filter out the illegal data coming from various network with low false positive rate and low false negative rate.

III.SIGFREE ARCHITECTURE

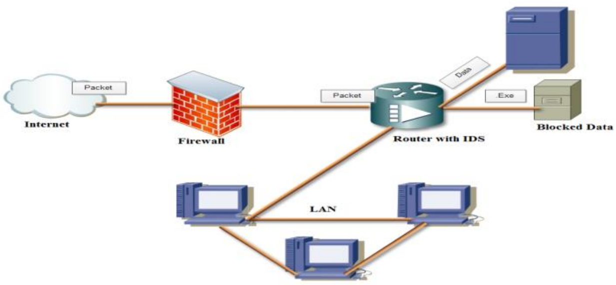

Fig.3.0 Architecture of Sigfree

In Fig.3.0 Sigfree is being installed on the router in between Firewall and network. The Packets or data coming from different host system is being filtered out before moving to firewall. Sigfree uses Code abstraction techniques which separates data and executable. SigFree first uses a new O(N) algorithm, where N is the byte length of the message, to disassemble and distil all possible instruction sequences from the message's payload, where every byte in the payload is considered as a possible starting point of the code embedded.

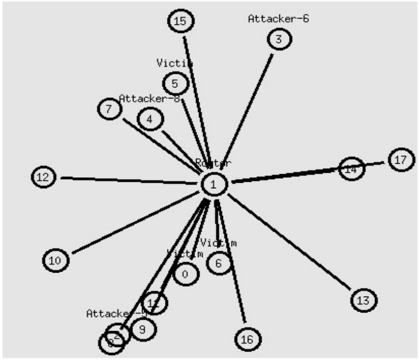

Fig 3.1Figure with different attacker and victim host system in the network.

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.

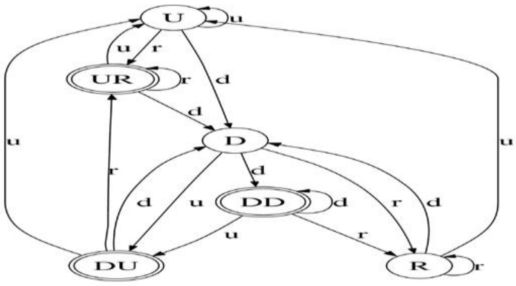

Fig.3.2. State diagram of a variable. State U: undefined, state D: defined but not referenced, state R: defined and referenced, state DD: abnormal state define-define, state UR: abnormal state undefine-reference, and state DU: abnormal state define-undefine.

In figure 3.2 to detect the a forementioned obfuscated buffer overflow attacks. Scheme exploits the data flow characteristics of a program. Normally, a random instruction sequence is full of data flow anomalies, whereas a real program has few or no data flow anomalies. However, the number of data flow anomalies cannot be directly used to distinguish a program from a random instruction sequence because an attacker may easily obfuscate his program by introducing enough data flow anomalies.
In this paper, we have use the code abstraction technique, which analyse the code and detect the data anamoly based on its property whether it is refrence,undefined.So such instruction falls under useless instruction category.We observe 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, we conclude the instruction sequence is a segment of a program.

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

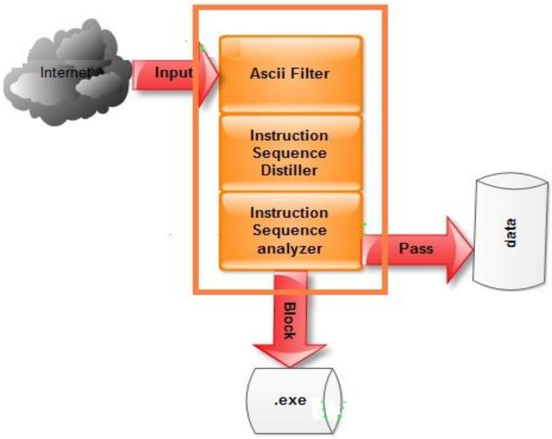

Fig.4.0 Block architecture of Signature free intrusion detection system.

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()

begin

V. EXPERIMENT RESULT

In the above the 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 itsown 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

[1]M. Zitser. Securing software: An evaluation of static source code analyzers. Master's thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, Aug. 2003.
[2]Parasoft. Insure++: Automatic runtime error detection. http://www.parasoft.com, 2004.
[3] H. Etoh. GCC extension for protecting applications from stack smashing attacks. http://www.trl.ibm.com/projects/security/ssp/, Dec. 2003.
[4]C. Cowan, C. Pu, D. Maier, J. Walpole, P. Bakke,S. Beattie, A. Grier, P. Wagle, Q. Zhang, andH. Hinton. Stackguard: Automatic adaptive detection and prevention of buffer-overflow attacks. In Proceedings of the 7th USENIX Security Conference, pages 63–78, San Antonio, Texas, Jan. 1998.
[5] AlephOne. Smashing the stack for fun and profit.Phrack Magazine, 7(47), 1998.
[6] F. Bellard. TCC: Tiny C compiler.http://www.tinycc.org, Oct. 2003.
[7] Xinran Wang, Chi-Chun Pan, Peng Liu, and Sencun Zhu, "Sigfree: A Signature-Free Buffer Overflow Attack Blocker",*IEEE Trans. On Dependable and Secure Computing, VOL. 7,NO. 1, Jan-Mar 2010.*
[8] K.A. Bradley, S. Cheung, N. Puketza, B. Mukherjee, andR.A. Olsson, "Detecting Disruptive Routers: A Distributed Network Monitoring Approach," Proc. IEEE Symp. Securityand Privacy (S&P '98), pp. 115-124, May 1998.
[9] A.T. Mizrak, Y.-C. Cheng, K. Marzullo, and S. Savage, "Detecting and Isolating Malicious Routers," IEEE Trans. Dependable and Secure Computing, vol. 3, no. 3, pp. 230-244, July-Sept. 2006.
[10]A surveyofintrusiondetectiontechniquesinCloud Chirag Modi a,n, DhirenPatel a, BhaveshBorisaniya a, HirenPatel b, Avi Patel c, MuttukrishnanRajarajan c.Journal ofNetworkandComputerApplications36(2013).
[11]H. Chen, D. Dean, and D. Wagner, "Model Checking One Million Lines of C Code," Proc. 11th Ann. Network and Distributed System Security Symp. (NDSS), 2004.
[12] H.-A. Kim and B. Karp, "Autograph: Toward Automated,Distributed Worm Signature Detection," Proc. 13th USENIXSecurity Symp. (Security), 2004.
[13] J. Newsome, B. Karp, and D. Song, "Polygraph: AutomaticSignature Generation for Polymorphic Worms," Proc. IEEE Symp. Security and Privacy (S&P), 2005.
[14] R. Chinchani and E.V.D. Berg, "A Fast Static Analysis Approachto Detect Exploit Code inside Network Flows," Proc. Eighth Int'l Symp. Recent Advances in Intrusion Detection (RAID), 2005.
[15] GCC Extension for Protecting Applications from Stack-Smashing Attacks, http://www.research.ibm.com/trl/projects/security/ssp, 2007.
[16] T. cker Chiueh and F.-H. Hsu, "Rad: A Compile-Time Solution to Buffer Overflow Attacks," Proc. 21st Int'l Conf. Distributed Computing Systems (ICDCS), 2001.
[17] A. Smirnov and T. cker Chiueh, "Dira: Automatic Detection,Identification, and Repair of Control-Hijacking Attacks," Proc. 12th Ann. Network and Distributed System Security Symp. (NDSS), 2005.
[18] Jempiscodes—A Polymorphic Shellcode Generator, http://www.shellcode.com.ar/en/proyectos.html, 2007.
[19] S. Macaulay, Admmutate: Polymorphic Shellcode Engine, http://www.ktwo.ca/security.html, 2007.
[20] T. Detristan, T. Ulenspiegel, Y. Malcom, and M.S.V. Underduk, Polymorphic Shellcode Engine Using Spectrum Analysis, http:// www.phrack.org/show.php?p=61&a=9, 2007.