IPASJ International Journal of Electronics & Communication (IIJEC)
Web Site: http://www.ipasj.org/IIJEC/IIJEC.htm
A Publisher for Research Motivation.....
Email: editoriijec@ipasj.org
Volume 5, Issue 2, February 2017
ISSN 2321-5984

A SYSTEM CALL RANDOMIZATION BASED METHOD FOR COUNTERING CODE-INJECTION ATTACKS

Mrs. Tiya Khanna

Lakshmi Narain College of Technology and Science, Bhopal

ABSTRACT

Code-injection attacks cause serious threat to today's net. the present code-injection attack defense ways have some deficiencies on performance overhead and effectiveness. to the current finish, we tend to propose a technique that uses system referred to as randomisation to counter code injection attacks supported instruction set randomisation plan. System calls should be used once Associate in Nursing injected code would perform its actions. By making irregular system calls of the target method, Associate in Nursing assailant UN agency doesn't apprehend the key to the randomisation rule can inject code that isn't irregular like because the target method and is invalid for the corresponding de-randomized module. The injected code would fail to execute while not line of work system calls properly. Moreover, with extended complier, our technique creates ASCII text file randomisation throughout its collection and implements binary feasible files randomisation by feature matching. Our experiments on engineered epitome show that our technique will effectively counter selection code injection attacks with low-overhead.

1. INTRODUCTION

Code injection attacks area unit to take advantage of software package vulnerabilities and inject malicious code into a computer program. the method management flow is changed in a way that the injected code is finally dead. In general, the term "shellcode" is employed to talk to injected code. several techniques are introduced to forestall code injection attacks from numerous angles. the foremost notable technique is Instruction Set Randomization (ISR) [1-5]. ISR randomizes instruction set for each process of target system, performs de-randomization before executing on CPU to recover the original instruction set and execute.

An attacker does not know the key of the randomization algorithm. The shellcode can't be randomized like the target program so that it is invalid for that de-randomized process, causing a runtime error. Code injection attack would fail to execute.

Although ISR can effectively thwart code injection attacks, it incurs enormous performance cost because of its per-instruction de-randomization on a virtual processor and lack of hardware support. Such a system cannot be practically deployed.

Moreover, using a extended compiler, we perform source code randomization during compiling and implement binary executable files randomization by feature matching. Target Process Randomization shellcode De-Randomization Attacker System Call System Call Xo CPU System Call Xn Figure 2. System Call Randomization and De-randomization In the rest of paper, we first present the defense principle in Section 2. We then describe the implementation in Section 3. We demonstrate effectiveness and efficiency by experiments in Section 4. We explain related work in Section 5. Finally, we conclude in Section 6.

2. PRINCIPLE

The majority of injected code is machine instruction, so we focus on machine instruction code injected attacks in this paper. The characteristics of the shellcode need to be noticed including (1) machine instruction complied, (2) attacking target platform oriented, (3) short code and (4) system call must be used. According to the architecture of computer system, system calls are the only interfaces for a program to access system resources. The program would fail to execute without calling system calls correctly. In essence, a shellcode would perform its actions with system calls like a normal

program. Each system call has an index called system call number. OS will call the implement functions according to this number. System call number randomization on operating system level will prevent shellcode from successful execution. Our method can defeat a wide variety of code injection attacks while incurring low performance penalty. Source code System Call Xo ELF files Extended GCC Processing Tools Target Process System Callr Xn System Call Randomization System Call Y shellcode User Space Kernel Space System Call De-Randomization System Call Target Process System Call Number Xo Lookup Table Execute Correctly Fail System Call Z shellcode Figure 3. The Prototype of Our System In general, OS maintains a consistent and backward compatible mapping between system call numbers and their implement functions. First, the system call numbers of target program are randomized.

Attackers do not know that the target program has been randomized, in the kernel space our de-randomization module transforms the system call number in shellcode into another one which can not be corresponding to the implement function expected by the attacker. Finally the shellcode fails to execute because of invalid parameters or meaningless system call number.

A. Randomization On one hand, a program can call system calls directly or via library functions indirectly. In this paper, the randomization to source programs will be enforced through extended GCC and GLIBC. On the other hand, the randomization to the binary executable files without source code is also implemented. As a consequence, our system can provide full protection against code injection attackers. Figure 4. System call instruction node
1) Extended GCC System call number transformation In this paper, GCC compiler is extended to randomize source code. A program are going to be translated into RTL format by GCC. the most structure of RTL format may be a two-way joined list that consists by instruction nodes. Through feature matching, the extended GCC will establish these instruction nodes. randomisation are going to be done on these matched instruction nodes. the data regarding supervisor call instruction numbers has 2 types of modes.

To the latter, GCC can forward searches the two-way joined list from the present node to seek out the register that contains the supervisor call instruction range , search the instruction node that performs the last assignment operation to the current register, and skim the supervisor call instruction range, then follow an equivalent steps because the former.
B. De-randomization A kernel module supported the Loadable Kernel Module (LKM) is style to intercept supervisor call instruction requests, de-randomize the supervisor call instruction range before the supervisor call instruction invoked within the kernel and store the first supervisor call instruction handler within the memory. If the present method is that the target method that the user desires to shield, the kernel de-randomizes the supervisor call instruction range victimization the strategy provided by the user, so invokes the corresponding supervisor call instruction handler. Otherwise, the first supervisor call instruction handler are going to be invoked. solely the supervisor call instruction range within the target method are going to be de-randomized.

3. EXPERIMENT

Our epitome system is known as as CIAS. it's evaluated from 2 aspects.
A. Effectiveness Some real code injection attacks area unit used to demonstrate that CIAS will effectively thwart a good kind of code injection attacks. as an example, one is that the shellcode invokes the supervisor call instruction 'execve ("/bin/sh")', attackers will begin a shell and execute any direction. Another is that the shellcode invokes the supervisor call instruction 'root' on to restart the pc. CIAS will defeat these attacks with success.
B. potency The physical check platform is UNIX operating system two.4.20-8 with two.40GHz Intel Pentium IV processor, 256M RAM and GCC three.2.2. There area unit 3 experiments as following: initial, we tend to measured some single supervisor call instruction like getpid, sethostname and open. compared with, we tend to tested the add .

4. CONNECTED WORK

There area unit 2 main randomisation techniques proposed: one is ISR [1-5], another is Address area Layout randomisation (ASLR) [6-9]. ISR creates a irregular instruction set for every method in order that directions in shellcode fail to execute properly albeit attackers have already hijacked the management flow of the vulnerable method. ASLR, instead, randomizes the memory address layout of a running method (including library, heap, stack, and relative distances between knowledge and code) in order that it's arduous for attackers to find injected shellcode or existing program code, preventing attackers from hijacking the management flow. tho' ISR may be a powerful technique, it incurs a lot of performance penalty, because it requires the introduction of an emulator and the binary transformation of

applications. ASLR has been deployed by many operating systems such as Linux kernel 2.6 and Windows Vista. However, ASLR suffers from a number attacks. Michal Bucko from HACKPL Security Lab [10] pointed out that some attack techniques such as Heap Spraying could bypass ASLR. RandSys [11] implemented randomization on system call level and used DES algorithm to encrypt important data. But lack of stability is the most serious disadvantage as the result of its modification of the kernel code of Linux. StackGuard [12,13] encrypts control information in a stack by XORing it with a random number. But it is not easy to use since the kernel of Linux need to be recompiled and the performance overhead is very high. According to Monica Chew [14], the cost of StackGuard is up to 30%. Monica Chew [15] proposed several methods of mitigating buffer overflows by introducing randomness into the implementation of system software. One of their methods changes the mapping between system call IDs and system call handlers by mixing up the system call table using random numbers. This is achieved by recompilation of the kernel and binary rewriting of applications to fit them to the new kernel. In their method, one mapping is shared by all processes and does not change except when the kernel is recompiled. Yoshihiro Oyama [15] enhanced the system performance and usability by using kernel modules.

5. CONCLUSION

We described our randomization scheme on the level of OS kernel to counter code injection attacks. We randomize only the system call numbers rather than the entire instruction set, therefore effectively solve the performance drawback of ISR. we've got additionally developed some techniques to boost the system performance. Firstly, the preprocessing is introduced. A majority of tasks of the randomisation and de-randomization were pre-processed once the system began to run. The table was engineered to store the mapping between the first supervisor call instruction numbers and therefore the irregular ones. The derandomization will be accomplished quickly by lookingup the table. in order that the performance value of our system is extremely low. Secondly, the computer program is configurable by users. the protection of a system is typically determined by the vulnerable space that ought to be protected first. In our system, users will tack together the computer program per their own demand. it's a lot of versatile throughout deploying the system and reduces the system overhead greatly. Thirdly, a whole protective tool set is achieved, which may give full protection against code injection attacks. we tend to implement ASCII text file randomisation by extended GCC and GLIBC and binary feasible files randomisation by matching the supervisor call instruction directions within the ELF files. Finally, a dynamic randomisation policy is used. ancient randomisation policy is static, like random numbers and coding algorithms (AES, XOR) that area unit restricted in security and strength. The dynamic randomisation policy isn't obsessed with the random numbers and therefore the coding algorithms and is configurable by users. So that it is more secure from attack.

REFERENCE

[1]. Elena Gabriela Barrantes, David H. Ackley,Stephanie Forrest, Trek S. Palmer, Darko Stefanovi′c, and Dino Dai Zovi. Randomized Instruction Set Emulation to Disrupt Binary Code Injection Attacks.

[2]. Gaurav S. Kc, Angelos D. Keromytis, and Vassilis Prevelakis. Countering Code-Injection Attacks With Instruction-Set Randomization. In Proceedings of the 10th ACM Conference on Computerand Communications Security (CCS 2003), P.272–280, Washington DC, Oct. 2003.

[3]. Elena Gabriela Barrantes, David H. Ackley, Stephanie Forrest, Darko Stefanovic, and Dino Dai Zovi, "Randomized Instruction Set Emulation to Disrupt Binary Code Injection Attacks," ACM Transactions on info and System Security, 2005.

[4]. Stelios Sidiroglou, Michael E. Locasto, Stephen W. Boyd, and Angelos D. Keromytis, "Building a Reactive system for software package Services," In Proceedings of the USENIX Annual Technical Conference, P.149 - 161, Anaheim, CA, April 2005.

[5]. Noritaka Osawa. a wise Virtual Machine for Heterogeneous Distributed Environments: PivotVM. Transactions on scientific discipline Society of Japan, 40(6):2543–2552, June 1999.