Journal of Computer Applications ISSN: 0974 – 1925,
Volume-5, Issue EICA2012-4, February 10, 2012

Detection and Prevention of Code Injection Attacks Using Monitoring Mechanism

Raichal S
Prathyusha Institute of Technology and Management, Thiruvallur, Tamilnadu.
E-mail:s.raichal@gmail.com

Durga Devi S
Prathyusha Institute of Technology and Management, Thiruvallur, Tamilnadu.
E-mail:s.durgadevi@gmail.com

Abstract - Intrusion detection systems play an important role in detecting and disrupting attacks before they can compromise software. Multivariant execution is an intrusion detection mechanism that executes several slightly different versions, called variants, of the same program in lockstep. The variants are built to have identical behavior under normal execution conditions. However, when the variants are under attack, there are detectable differences in their execution behavior. At runtime, a monitor compares the behavior of the variants at certain synchronization points and raises an alarm when a discrepancy is detected. The variants with a down-ward growing stack are given the exploit code the exploits succeed and an attacker is able to obtain illicit access to the target computer. When an upward growing stack variant is presented with the same exploit code, the variant continues to run since the buffer overflow writes into unused memory. In this project in order to recover the solution for code injection attack is to propose a scheduling algorithm to prevent the damage from the attack. The number of variants increases, the performance penalty of multivariant execution increases. There are two main reasons for this: first, the monitor has to compare the data flowing out of a larger number of variants and also copy results of system calls to them.

Keywords - *Intrusion detection, multivariant execution,n- variant execution, system call.*

I. INTRODUCTION

Security vulnerabilities in software have been a for decades. While the use of safer programming languages such as Java and C# has alleviated the problem, there are still many software packages that are created and maintained in C and C++. This is primarily driven by concerns about performance and access to low-level constructs, which is not always possible in languages executed in a managed environment. Despite an increase in education and the availability of safer APIs designed to help detect errors.

As a result, the challenge of finding mechanisms to detect and remove vulnerabilities persists. With the large amount of code written every year, it should be noted that despite the fact that the vulnerability density is decreasing, the overall number of vulnerabilities is increasing. Multivariant code execution is a runtime monitoring technique that prevents system damage resulting from malicious code execution and addresses the above problems with dynamic detection tools. Multivariant execution protects against malicious code execution attacks by running two or more slightly different versions of the same program, called variants, in lockstep. At defined synchronization points, the variants' behavior is compared against each other. Divergence among the behavior is an indication of an anomaly and raises an alarm.

An obvious drawback of multivariant execution is the extra processing overhead, since at least two variants of the same program must be executed in lockstep to provide the benefits mentioned above. Our experimental results show that this overhead is in the range afforded by most security sensitive applications where performance is not the first priority, such as government and banking software. Besides, the large amount of parallelism that inherently exists in multivariant execution helps it take advantage of multicore processors. Currently, cores are often idle due to the lack of extractable parallelism in many applications or due to the bottlenecks imposed by memory or I/O devices . Moreover, the number of cores is increasing rapidly. A (MVEE) can engage the idle cores in these systems to improve security with little performance overhead. Unlike many previously proposed techniques to prevent malicious code execution , that use random and / or secret keys in order to prevent attacks, multivariant execution is a secret-less system. It is designed on the assumption that program variants have identical behavior under normal execution conditions but their behavior differs under abnormal conditions. Therefore, the choice in what to vary, e.g., stack layout or instruction set, defines which classes of attacks can be stopped and which vulnerabilities still can be exploited

It is important that every variant be fed identical copies of each input from the system simultaneously. This design makes it difficult for an attacker to send individual malicious inputs to different variants and compromise them one at a time. If the variants are chosen properly, a malicious input to one variant causes compromise them one at a time. If the variants are chosen properly, a malicious input to one variant causing them to deviate from each other. The deviation is then detected by a monitoring agent that enforces a security policy and raises an alarm.

MVEE is an unprivileged user-space application that does not need kernel privileges to monitor the variants and, therefore, does not increase the trusted computing base (TCB) for processes not running on top of it. Increasing the size of the TCB is detrimental to the overall security of a system. This has raised concerns in recent years and many researchers investigate methods to reduce the TCB size.

II. EXISTING METHODOLOGY

Multivariant execution is a monitoring mechanism that controls the states of the variants being executed and verifies that the variants are complying to defined rules. A monitoring agent, or monitor, is responsible for performing the checks and ensuring that no program instance has been corrupted. This can be achieved at varying granularities, ranging from a coarse-grained approach that only checks that the final output of each variant is identical, all the way to a checkpointing mechanism that compares each executed instruction. The granularity of monitoring does not impact what can be detected, but it determines how soon an attack can be caught. We use a monitoring technique that synchronizes program instances at the granularity of system calls shown in fig 1.

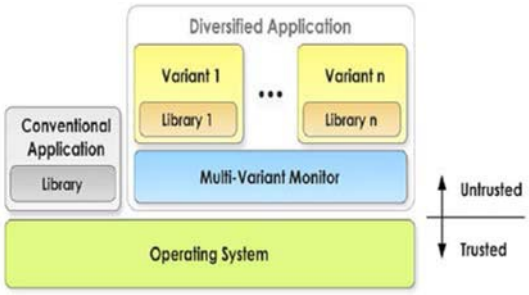

Figure 1: The Architecture of MVEE.

2.1 Monitor Security

The monitor isolates the variants from the OS kernel and monitors all communications between them and the kernel. The monitor is implemented as an unprivileged process that uses the process debugging facilities of the host operating system (Linux) to intercept system calls. This mechanism simplifies maintenance as patches to the OS kernel need not be reapplied to an updated version of the kernel. Moreover, errors in the monitor itself are less severe since the monitor is a regular unprivileged process, as opposed to a kernel patch or module running in privileged mode. If the monitor was compromised, an attacker would be limited to userlevel privileges(fig 2) and would need a privilege escalation to gain system-level access.

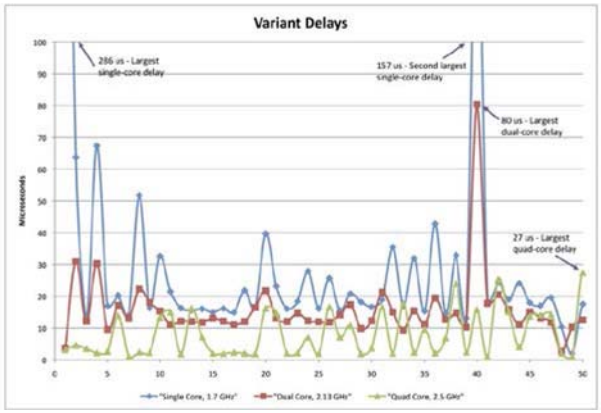

Figure 2: Intersystem call delays experienced by a variant in an MVEE vary according to the hardware of the underlying system.

2.2 System Call Execution

An MVEE and all the variants executed in this system must act as if only one variant was running conventionally on the host operating system. The monitor is responsible for providing this behavior by running certain system calls on behalf of the variants and providing the variants with the results. We examined the system calls of the host operating system (Linux) one by one and considered the number and types of possible arguments that can be passed to them. Depending on the effects of these system calls and their results, we specified which ones can be executed by the variants and which ones must be run by the monitor

2.3 Monitor-Variant Communication

The monitor spawns the variants as its own children and traces them. Since the monitor is executed in user mode, it is not allowed to directly read from or write to the variants' memory spaces. In order to compare the contents of buffers passed to the system calls, the monitor needs to read from the memory of the variants. Also, it needs to write to their address spaces if a system call executed by the monitor on behalf of the variants returns results in memory.

2.4 Variant Generation

One of the key features of the multivariant execution technique that distinguishes it from n-version programming is automated variant generation. The variants of a program are generated automatically from the same source code, eliminating the need to rewrite the variants manually. This feature significantly reduces the costs of development and maintenance of the variants..

III. PROPOSED METHODOLOGY

The variants are built to have identical behavior under normal execution conditions. When the variants are under attack, there are detectable differences in their execution behavior. At runtime, a monitor compares the behavior of the variants at certain synchronization points and raises an alarm when a discrepancy is detected. We present a monitoring mechanism that does not need any kernel privileges to supervise the variants.

Many sources of inconsistencies, including asynchronous signals and scheduling of multithreaded or multiprocess applications, can cause divergence in behavior of variants. This can be achieved at varying multiprocess applications, can cause divergence in granularities, ranging from a coarse-grained approach that only checks that the final output of each variant is identical, all the way to a check pointing mechanism that compares each executed instruction. The proposed is a scheduling algorithm to schedule the task to reduce the synchronizing timing to the multipath variants. The priority algorithm is also proposed to recover the false alarms and identify the damage system that can be repaired in quick time. Discrepancy in behavior of the variants is an indication of an attack. Using this technique we prevent exploitation of vulnerabilities at runtime

3.1 Priority Scheduling

Each process is assigned a priority, and priority is allowed to run. Equal-Priority processes are scheduled in FCFS order. The Shortest-Job-First (SJF) algorithm is a special case of general priority scheduling algorithm.An SJF algorithm is simply a priority algorithm where the priority is the inverse of the (predicted) next CPU burst. That is, the longer the CPU burst, the lower the priority and vice versa.Priority can be defined either internally or externally. Internally defined priorities use some measurable quantities or qualities to compute priority of a process.Examples of Internal priorities are,Time limits,Memory requirements,File requirements, CPU Vs I/O requirements.Externally defined priorities are set by criteria that are external to operating system such as,The importance of process, Type or amount of funds being paid for computer use,The department sponsoring the work,Politics. Priority scheduling can be either preemptive or non preemptive,A preemptive priority algorithm will preemptive the CPU if the priority of the newly arrival process is higher than the priority of the currently running process,A non-preemptive priority algorithm will simply put the new process at the head of the ready queue.A major problem with priority scheduling is indefinite blocking or starvation. A solution to the problem of indefinite blockage of the low-priority process is *aging*. Aging is a technique of gradually increasing the priority of processes that wait in the system for a long period of time.

3.2 WinPcap

WinPcap is an open source library for packet capture and network analysis for the Win32 platforms. Most networking applications access the network through widely used operating system primitives such as sockets. It is easy to access data on the network with this approach since the operating system copes with the low level details (protocol handling, packet reassembly, etc.) and provides a familiar interface that is similar to the one used to read and write files.Sometimes, however, the 'easy way' is not up to the task, since some applications require direct access to packets on the 111
network. That is, they need access to the "raw" data on
the network without the interposition of protocol processing by the operating system.

3.3 Jpcap

Jpcap is an open source library for capturing and sending network packets from Java applications. It provides facilities to:

- Capture raw packets live from the wire.
- Save captured packets to an offline file, and read captured packets from an offline file
- Filter the packets according to user-specified rules before dispatching them to the application.
- send raw packets to the network Jpcap is based on libpcap/winpcap, and is implemented in C and Java

3.4 Improvement of Security

The system call execution timing is faster than the normal speed compare to the existing methods. The runtime exception is cleared and finds the accurate values that can be accessed by the asynchronous signals and delivery the packet at the exact time. In the random process the memory is allocated free and diverts the attack through check point techniques and

increase the lifetime of the attacked nodes. The scheduling allocates the security analysis through free loop that can be accessed by the false alarm that is completely reduced in the synchronous signals thorough physical memory. The variants is to downward the explicit the code by the check point and performance analysis will be analyzed the system call execution in runtime.

IV. PERFORMANCE EVALUATION

To demonstrate the effectiveness of the multivariant execution environment, use of a customized test suite that includes common benchmarks and frequently used applications. This suite allows to evaluate the security claims and assess the computational trade-off in CPU- and I/O-bound operations. While MVEE is capable of running different number of variants and many types of variation techniques, evaluation is with two, three, and four variants shown in fig 3.

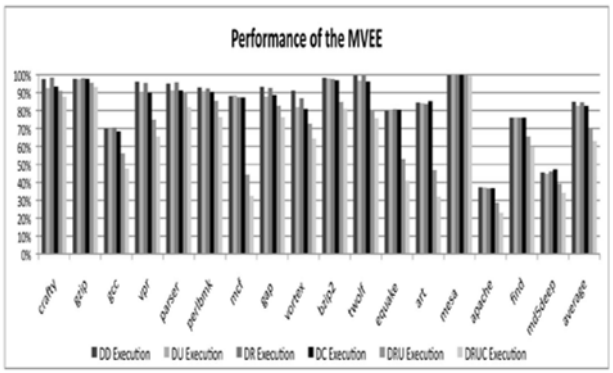

Figure 3: Comparison of the performance of the MVEE relative to conventional programs.

V. CONCLUSION

A multivariant execution environment runs multiple versions of a program simultaneously and monitors their behavior. Discrepancy in behavior of the variants is an indication of an attack. Using this technique, we prevent exploitation of vulnerabilities at runtime. It is complementary to other methods that remove vulnerabilities, such as static analysis. Instead of finding and removing the vulnerabilities, our method accepts the inevitable existence of vulnerabilities and prevents their exploitations. A major advantage of this approach is that it enables us to detect and prevent a wide range of threats, including “zero-day” attacks. Multivariant execution is effective even against sophisticated polymorphic and metamorphic viruses and worms.

REFERENCES

1. Avizienis .A, “The N-Version Approach to Fault- tolerant Software,”IEEE Trans. Software Eng., vol. SE-11, no. 12, pp. 1491-1501,Dec
2. A. Avizienis and L. Chen, “ On the Implementation of N-VersionProgramming for Software Fault Tolerance during Execution,”Proc. Int’l Computer Software and Applications Conf., pp. 149-155,1977.

3. E. Barrantes, D. Ackley, T. Palmer, D. Stefanovic, and D. Zovi, "Randomized Instruction Set Emulation to Disrupt Binary Code Injection Attacks," Proc. ACM Conf. Computer and Comm. Security,pp. 281-289, 2003.
4. E. Berger and B. Zorn, "DieHard: Probabilistic Memory Safety forUnsafe Languages," Proc. ACM SIGPLAN Conf. Rogramming Language Design and Implementation, pp. 158-168, 2006.
5. K. Birman, "Replication and Fault-Tolerance in the ISIS System," ACM SIGOPS Operating Systems Rev., vol. 19, no. 5, pp. 79-86, 1985
6. Black.D, C. Low, and S.K. Shrivastava, "The Voltan a pplication Programming Environment for Fail-Silent Processes," Distributed Systems Eng., vol. 5, pp. 66-77, 1998..
7. T.C. Bressoud and F.B. Schneider, "Hypervisor-Based Fault Tolerance," ACM Trans. Computer Systems, vol. 14, no. 1, pp. 80-107, 1996
8. D. Bruschi, L. Cavallaro, and A. Lanzi, "Diversified Process Replicae for Defeating Memory Error Exploits," Proc. Int'l Workshop Information Assurance, pp. 434-441, 2007.
9. M. Chereque, D. Powell, P. Reynier, J. Richier, and J. Voiron, "Active Replication in Delta-4," Proc. Int'l Symp. Fault-Tolerant Computing, pp. 28-37, 1992.
10. M. Chew and D. Song, "Mitigating Buffer Overflows by Operating System Randomization," Technical Report CMU-CS-02-197, Dept.of Computer Science, Carnegie Mellon Univ., 2002.B.N. Levine, C. Shields, and N.B. Margolin, "A Survey of Solutions to the Sybil Attack," Technical Report 2006-052, Univ. of Massachusetts, Oct. 2006.
11. Salamat .B, T. Jackson, A. Gal, and M. Franz, "Orchestra: Intrusion Detection Using Parallel Execution and Monitoring of Program Variants in User-Space," Proc. European Conf. Computer Systems, pp. 33-46, 2009.
12. Salamat.B, C. Wimmer, and M. Franz, "Synchronous Signal Delivery in a Multi-Variant Intrusion Detection System," technicalreport, School of Information and Computer Sciences, Univ. ofCalifornia, 2009.