# Monitoring System Calls for Anomaly Detection in Modern Operating Systems

Shayan Eskandari, Wael Khreich, Syed Shariyar Murtaza and Abdelwahab Hamou-Lhadj
Software Behaviour Analysis (SBA) Research Lab
Concordia University, Montreal, QC, Canada
{s_eskand, wkhreich, smurtaza, abdelw}@ece.concordia.ca

Mario Couture
Software Analysis and Robustness Group
Defence Research and Development Canada
Valcartier, QC, Canada
mario.couture@drdc-rddc.gc.ca

## Abstract

*Host-based intrusion detection systems monitor systems in operation for significant deviations from normal (and healthy) behaviour. Many approaches have been proposed in the literature. Most of them, however, do not consider even the basic attack prevention mechanisms that are activated by default on today's many operating systems. Examples of such mechanisms include Address Space Layout Randomization and Data Execution Prevention. With such security methods in place, attackers are forced to perform additional actions to circumvent them. In this research, we conjecture that some of these actions may require the use of additional system calls. If so, one can trace such attacks to discover attack patterns that can later be used to enhance the detection power of anomaly detection systems. The purpose of this short paper is to motivate the need to investigate the impact of attack on system calls while trying to overcome these prevention mechanisms.*

**Keywords:** Host-Based Intrusion Detection Systems, Address space layout randomization, data execution prevention, software security and reliability.

## 1. Introduction

Anomalies are patterns in data that do not conform to the expected normal behavior [1]. An anomaly detection system (ADS) constructs a profile of expected normal behaviour using data, collected over a period of normal (attack-free) operation. During operation, an ADS looks for events that deviate significantly from the normal profile. These deviations are considered as anomalous activities. They can be caused by attacks, program, or configuration errors. Unlike signature-based detection techniques, which look for patterns of known attacks, anomaly detection is capable of detecting novel attacks, but suffers from high false alarm rate [1, 2, 3].

Most anomaly detection systems monitor for significant deviation by observing system calls–system calls provide a gateway between user and kernel mode. The temporal order of system calls used by a process to request kernel services has been shown to be effective in describing normal process behaviour (see [1]). Many research studies use system call sequences to model normal behaviour of the system (see [2][3] for examples). Modeling techniques vary to include statistical models, machine learning, and data mining. A good survey of host-based anomaly detection techniques is presented in [1].

However, most existing techniques for detecting anomalies at the system call level do not consider security mechanisms, which are enabled by default on modern operating systems [7]. These mechanisms include the Address Space Layout Randomization (ASLR) [4, 6] and the Data Execution Prevention (DEP) [8] techniques. ASLR introduces randomness into memory addresses of libraries, stack, etc., to prevent buffer memory corruption attacks. DEP does not allow applications to execute from a writeable memory region. Both security mechanisms are supported by most operating systems including Linux and Windows.

These advances in prevention mechanisms have challenged the attackers to evolve their techniques and create more complex attacks. Although some defense strategies have been proposed to address these complex attacks [7], their impact on anomaly detection using system calls has not been addressed yet. The main objective of this work is to motivate the need to investigate the impact of these attacks – aimed at bypassing the prevention mechanisms–at the system call level (including, system call sequences, arguments, and return values). The anticipated findings of this research (e.g., new pattern of attacks and execution of foreign system calls) would help reduce the false alarm rate while improving the detection accuracy of anomaly detection systems based on system calls.

## 2. Escaping Security Mechanisms

To escape ASLR and DEP, attackers often resort to a form of brute-force attack. The most common way is Return-

Oriented Programming such as return-to-libc [9]. In order to use return-to-libc attacks, the attacker should first 'guess' the address of the functions in the libc library. This step is usually done by a trial-and-error method known as brute-forcing.

It is, however, possible to discover the libc address by exploiting the forked child process memory as shown by Shacham et al. [7]. While trying to guess the address of a libc function (e.g., system(), usleep(), etc.), a wrong deduction will crash the process and make it fork itself (if it set to automatic restart such as network daemons). Since the forked process would have the same address layout as its parent, the attacker would continue brute-forcing until he or she finds the correct memory address of the required libc function. For example, having the address of system() function in libc allows the attacker to execute any desired system command. There are also other methods to bypass the prevention security mechanisms such as stack juggling methods, return into non-randomized memory (return to text, return to heap, return to data, etc.) and other application specific techniques [6].

## 3. Potential Impact on Anomaly Detection

The ways that attackers are forced to create new exploits to overcome the defense mechanisms may have several impacts at the system call level. For instance, brute-forcing techniques will lead to a repetitive usage of system calls, such as fork(), nanosleep() and system(). This frequent usage could be detected as foreign system call anomaly or foreign sequence anomaly (a classification of system call anomaly types can be found in [5]). Other patterns may involve the invocation of unusual system call arguments, for example calling "system(/bin/bash)" within a particular process.

In other words, we need to work towards finding and analyzing such anomalous patterns or behaviors at the system call level to improve the anomaly detection performance. A possible direction to follow is by conducting a conceptual analysis of some core attacks, which overcome the prevention mechanisms described above. The result could be a taxonomy of attacks with their behavioural model depicted. This would require static analysis of the attack source code (if available) combined with excellent knowledge of the kernel operations.

Another possible direction would be to execute attacks and collect traces that would allow the analysis of the impact of attacks on the system call sequences and arguments. The challenge with this approach is to find a decent set of running attacks that can be experimented with.


## Acknowledgment

This research is supported partially by the Natural Sciences and Engineering Research Council of Canada (NSERC), DRDC (Defence R&D Canada), Valcartier, QC, Quebec, Ericsson Canada.


## 4. References


[1]. S. Forrest, S. Hofmeyr, and A. Somayaji, "The evolution of system-call monitoring," *In Proc. of the 2008 Annual Computer Security Applications Conference*, pp. 418-430, 2008.

[2]. D. Gao, M. K. Reiter, and D. Song, "On gray-box program tracking for anomaly detection," *In Proc. of the 13th conference on USENIX Security Symposium*, pp. 1-8, 2004.

[3]. C. Kruegel, E. Kirda, D. Mutz, W. Robertson, and G. Vigna, "Automating mimicry attacks using static binary analysis," *In Proc. of the 14th conference on USENIX Security Symposium*, pp. 161–176, 2005.

[4]. L. Lixin, J.E, Just, R. Sekar, "Address-Space Randomization for Windows Systems," *In Proc. of the 22nd Annual Computer Security Applications Conference*, pp. 329 - 338, 2006.

[5]. R. Maxion and K. Tan, "Benchmarking anomaly-based detection systems," *In Proc. of the 2000 International Conference on Dependable Systems and Networks*, pp. 623–630, 2000.

[6]. T. Muller, "Aslr smack & laugh reference," *Presented at the Seminar on Advanced Exploitation Techniques*, 2008.

[7]. H. Shacham, M. Page, B. Pfaff, E.-J. Goh, N. Modadugu, and D. Boneh, "On the effectiveness of address-space randomization," *In Proc. of the 11th ACM conference on Computer and communications security*, pp. 298–307, ACM, 2004.

[8] N. Stojanovski, M. Gusev, D. Gligoroski, S.J. Knapskog, "Bypassing Data Execution Prevention on MicrosoftWindows XP SP2," *In Proc. of the 2nd International Conference on Availability, Reliability and Security*, pp.1222-1226, 2007.

[9] P. Chen,H. Xiao, X. Shen, X. Yin, B. Mao, L. Xie, "DROP: Detecting Return-Oriented Programming Malicious Code," *In Proc. of the 5th International Conference on Information Systems Security*, pp 163-177, 2009.