Thesis for the Degree
of Master

DROPSYS: Detection of ROP Attacks Using System Information

by
Kim Seon Kwon

Department of Cyber Security

Graduate School

Korea University

December 2022

李 東 勳　教 授 指 導

碩 士 學 位 論 文

DROPSYS: Detection of ROP Attacks Using System Information

이 論文을 工學 碩士學位 論文으로 提出함.

2023 年　2 月

高麗大學校　情報保護大學院

情報保安學科

金 善 權　(印)

金善權의 工學 碩士學位論文 審査를 完了함.

2022 年　　12 月

委員長　　　李 東 勳 (印)

委　員　　　신 영 주 (印)

委　員　　　최 원 석 (印)

DROPSYS: Detection of ROP Attacks Using System Information

Kim Seon Kwon

Department of Cyber Security

Graduate School

Korea University

godmakessky@gmail.com

December 29, 2022

Abstract

As modern return-oriented programming (ROP) attacks have become more sophisticated, preventing or detecting these attacks is essential for real-world exploit defense. As an alternative to many defense techniques of ROP attacks that require software modification and hardware assistance, researchers have proposed ROP defense techniques using hardware performance counters (HPCs) to mitigate concerns about additional cost and compatibility issues. However the existing HPC data-based ROP detection techniques typically suffer from low detection performance mainly because of the non-deterministic nature of HPCs.

To address these issues, we propose DROPSYS, an enhanced Detection of ROP attacks using SYStem information. Differing from the existing techniques, DROPSYS harnesses not only HPC data, but also system utilization data to mitigate the non-deterministic nature of HPCs and overcome the limited number of available HPCs. For anomaly detection model, a long short-term memory-based variational autoencoder is used to effectively analyze the multi-dimensional time-series data from HPCs and system utilization data. DROPSYS also performs feature selection for low computational overhead while maintaining the attack detection performance. Evaluation results show that DROPSYS effectively captures the behaviors and effects of ROP attacks and can detect the attacks with a 0.046% false positive rate. The accuracy of DROPSYS is 92.7%, and its f1 score is 92.1%—a figure much higher than those of existing techniques that utilize only HPC data.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Return-oriented programming (ROP) has been used in malicious codes to extract sensitive information and install malware [1]. Reports cite that 80% of exploits in software vulnerabilities are made through ROP [2]. Since modern exploits have become sophisticated, preventing or detecting ROP is imperative for real-world exploit defense. For example, address space layout randomization (ASLR) [3] randomly arranges the virtual memory address space to prevent attackers from analyzing and manipulating program memory. Attackers who use ROP, however, can choose from among various techniques [4, 5, 6] that are based on information leakage and brute force attacks to bypass ASLR.

To ensure that a program is safely protected against ROP attackers, researchers have developed a number of hybrid defense techniques using both software and hardware [7, 8, 9, 10]. Although hybrid defense techniques are practical in terms of performance overhead, these come with additional hardware cost and compiler compatibility issues. For example, RetTag [10] re-

Table 1.1: Monitored HPC data.

Index	HPC data	Explanation
1	Instruction Retired	Number of instructions at retirement
2	UnHalted Core Cycles	Core clock cycles
3	UnHalted Reference Cycles	Reference clock cycles
4	BR_MISP_RETIRED.NEAR_TAKEN	Number of near branch instructions retired that were mispredicted and taken
5	L2_LINES_OUT.NON_SILENT	Number of lines evicted by L2 cache
6	FRONTEND_RETIRED.L2_MISS	Number of L2 cache true misses
7	LONGEST_LAT_CACHE.MISS	Number of the last-level cache misses

quires additional hardware support to check the pointer authentication code (PAC) in every function call with a return address authentication key. HAFIX [7] also requires modification of the existing compiler to use a special instruction set architecture (ISA) that assigns a unique label to each function and checks the control flow of a program. To overcome this limitation, several techniques [11, 12, 13, 14, 15, 16, 17, 18] have been designed to monitor and check the control flow of a program during runtime via hardware performance counters (HPCs), which are available by default in processors. These techniques are based on the observation that ROP incurs abnormal values from HPCs, and as such, they generally rely on heuristics to decide whether an ROP attack has occurred or not. Unfortunately, this approach turned out to be not trustful. That is, these techniques depend entirely on irregularly measured HPC data that stem from the non-deterministic nature of HPCs [19]. More importantly, the techniques failed to take into account the non-determinism of HPCs in their design process [20].

In response to this, the paper proposes DROPSYS, an enhanced Detection of ROP attacks using SYStem information. The system information includes

(1) HPC data from the hardware layer and **(2) system utilization data from the kernel layer**. While the existing techniques use HPC data only, DROPSYS additionally uses system utilization data since HPC data only do not represent the features incurred from ROP attacks. That is, the non-deterministic nature of HPC makes unstable measurement, and the limited number of HPCs supported by a modern processor (for example, 7 HPCs in Intel CPUs) can only provide a limited analysis of ROP attacks. System utilization data provide a useful data to capture features caused by ROP attacks such as call instructions not paired with return instructions. For detecting anomalies, DROPSYS uses a long short-term memory-based variational autoencoder (LSTM-VAE). A VAE is an autoencoder that encodes input data into a new representation as a distribution over the encoded space (also called the latent space) and then displays an output as encoded-decoded data. The characteristics of a VAE is that it regularises the covariance matrix and the mean of the distributions from the encoder to avoid overfitting and keep the latent space enabling generative process. However a VAE is known to often fail in detecting long term anomalies. LSTM is a recurrent neural network that can process entire sequence of data and hence used to overcome this demerit of a VAE. In DROPSYS, LSTM-VAE encodes system information from each time series sequence into a latent space and decodes the expected distribution of the input from the latent space. It turns out that LSTM-VAE is effectively reflect the characteristics of time series data in ROP attack detection by modeling the distribution of multidimensional system information. DROPSYS calculates reconstruction errors via mean squared errors (MSE) between the original input and the results of the decoding. DROPSYS detects any anomaly as an ROP attack when the reconstruction error is higher than a set threshold. Furthermore, DROPSYS utilizes the Shapley values which indicate the contribution of each feature to the prediction result of LSTM-VAE to interpret the system information contributed and then conducts feature selection for better detection performance. We evaluated DROPSYS using real-world ROP exploits from the Metasploit framework [21] that provides penetration test. Our evaluation results show that DROPSYS is able to detect the behaviors and effects of ROP attacks, and the average accuracy rate is 92.7%. The f1 score is 92.1%, unlike previous HPCs-based approaches [15, 16] whose f1 scores ranged from 61.5% to 82.2%. Besides this, DROPSYS reduces the false positive rate by about 85% compared to previous approaches [15, 16].

We summarize our contributions as follows:

- **A comprehensive ROP attack detection technique.** We propose a new ROP attack detection technique that monitors system information during runtime and detects the causalities of ROP attacks. DROPSYS mitigates the effect of HPC non-determinism in regard to ROP attack detection and overcomes the limited number of available HPCs.

- **Interpretable anomaly detection.** We employ an LSTM-VAE to improve the detection performance of ROP attacks and perform explaining anomaly detection, which optimizes the combining of system information with interpreting the prediction of DROPSYS.

- **Accurate ROP attack detection.** DROPSYS improves both the accuracy and the f1 score of ROP attack detection and significantly reduces the false positive rate as compared to previous approaches.

The remainder of the paper is organized as follows: Section 2 provides background. Section 3 describes our threat model. Section 4 provides an overview. Section 5 describes the methodology of DROPSYS, and Section 6 summarizes its implementation and presents our evaluation results. Section 7 summarizes related works, and Section 8 concludes this paper.

Chapter 2

Related Works

Hardware-assisted defense techniques. Hardware-assisted defense techniques included new approaches to overcome the limitations of existing software-only defense techniques, which required source code and symbolic debugging information of the target program to defend against control-flow attacks [22]. For example, CFIMon [13] utilizes the branch-tracing store mechanism provided by the processor's PMU along with static analysis to detect any abnormal control flow in the program. CFIMon also detects the integrity violation of the program's control flow in accordance with several proposed rules. After CFIMon, kBouncer [23] involves a technique that can detect ROP attacks using the last branch recording (LBR) mechanism. Lastly, kBouncer checks the LBR register when a system API function is called and detects ROP attacks according to the proposed CFI policy. However, API hooking defense techniques such as these can easily be bypassed by elaborate ROP attack payloads, such as hook hopping [24].

HPC-based defense techniques. Techniques using HPCs to detect

ROP attacks are based on heuristics that use the number of mispredicted return instructions. These techniques take advantage of the fact that ROP attacks change the control flow, causing branch prediction on return instructions to fail. The following three techniques utilize HPC data as thresholds for ROP attack detection: HDROP [14], SIGDROP [15], and ROPSENTRY [16]. First, HDROP detects ROP attacks by monitoring the mispredicted return instruction values. HDROP is a compiler-based approach that requires source code instrumentation in order to insert the necessary monitoring checkpoints. Next, SIGDROP detects ROP attacks by creating rules for hardware performance events, namely BrMispRetired.Ret, InstRetired.Any, and BrInstRetired.NearReturn. Similarly, ROPSentry utilizes BrMispRetired.Ret, BrInstRetired.NearReturn, InstRetired.Any, FrontedRetired.ITLBMiss, and LongestLatCache.Miss events to detect ROP attacks. In particular, ROPSentry includes an ROP-only defense method and a self-adaptive defense method to reduce performance overhead. However, detection results for techniques using only HPC data are less than desirable [20]. In addition, Tang et al. [18] proposed a technique to detect ROP attacks using the HPC-based one-class support vector machine. However, that technique focused more on what happens after an ROP attack, so the actual ROP attack detection rate remains low. Also, Pfaff et al. [17] proposed an ROP attack detection technique using HPC data and a support vector machine. However, the hardware performance event used in their technique is currently only found on discontinued processors. Outside of these existing techniques, many fields have turned to using HPC data [12]. For example, HPC data is used in other security fields like program integrity [25], malware detection [26], rootkits detection [27], and cache side-channel attack defense [28].

Chapter 3

Background

3.1 Return-oriented programming

ROP attacks exploit various memory vulnerabilities to manipulate the control flow of a program for the intended direction of an attacker [1]. Especially, ROP attacks exploit instruction sets, which exist in a program's memory space, without requiring the attacker to inject any code into the program [29]. An instruction set used in ROP attacks is called a gadget. Each gadget consists of a small instruction sequence, and the last instruction of each gadget ends with a return instruction. An ROP attacker essentially links the addresses of gadgets in a chain when constructing an attack payload. In the ROP attack process, an attacker uses memory vulnerabilities, such as buffer overflow, in the target program to overwrite the attack payload on the program's stack return address. Then, when the program's stack is returned, the return address that the attacker overwrote executes, and this causes the program's stack pointer to point to the attacker's gadget address.

Table 3.1: Monitored system utilization data.

Index	System utilization	Explanation
1	CPU user time	Process time spent in user mode
2	CPU system time	Process time spent in kernel mode
3	Read count	Number of read operations performed by a process
4	Write count	Number of write operations performed by a process
5	Read byte	Number of bytes read
6	Write byte	Number of bytes written
7	Other count	Number of I/O operations performed (other than read and write operations)
8	Other byte	Number of bytes transformed (other than read and write operations)
9	Page fault	Number of page faults
10	Peak Wset	Peak non-swapped physical memory used by a process
11	Wset	Non-swapped physical memory used by a process
12	Peak paged pool	Peak paged pool usage by a process
13	Paged pool	The current paged pool usage by a process
14	Peak non-paged pool	Peak non-paged pool usage by a process
15	Non-paged pool	Current non-paged pool usage by a process
16	Page file	Total amount of virtual memory used by a process
17	Peak page file	Peak total amount of virtual memory used by a process
18	USS	Memory size (unique to a process)
19	Memory map number	Number of memory maps per process
20	Memory map amount	Total amount of memory maps per process
21	Memory percent	Process memory utilization as a percentage
22	Voluntary context switch	Number of voluntary context switches by a process
23	Thread user time	Time spent by threads in user mode
24	Thread system time	Time spent by threads in kernel mode

The gadgets execute consecutively and manipulate the program using the control flow that the attacker desires.

3.2 Hardware performance counters

HPCs stand for a special-purpose register set built into the performance monitoring unit (PMU) of a processor. These register sets store hardware performance event information. In general, HPCs provide three fixed counters and four programmable counters per core in Intel CPUs, and modern processors provide many hardware performance events. Evaluating various processor performances, the PMU monitors hardware performance events through a set of model-specific HPCs. Hardware performance events for each processor can be found in the Intel manual [30].

3.3 System information

We define system information as HPC data in the hardware layer and system utilization data in the kernel layer. HPC data is information about hardware performance events measured through HPCs. Hardware performance events can be divided into architectural or non-architectural events. First, note that the architectural events are identical in the majority of processors. However, in contrast, the non-architectural events are specific to each generation of processors. Next, system utilization data relates to any system resources that the processes use. In other words, system utilization data indicates how many system resources a process is using. For example, system utilization data includes information on CPU usage, memory, disks, network, and so on [31, 32].

3.4 Anomaly detection explanation

Anomaly detection is a technique used to detect anomalies that are distinct from normal data [33, 34]. Anomaly detection based LSTM-VAE is a transformation of the encoder and decoder configuration of an VAE into an LSTM structure. An LSTM-VAE encodes system information from each time series sequence into a latent space and decodes the expected distribution of the input from the latent space. In detail, for the input received, the encoder estimates the latent Gaussian distribution. The latent variable z, which is randomly sampled from the latent distribution, is supplied to the input of the decoder. The decoder restores the input data using the latent variable z created by the encoder. The loss function of an LSTM-VAE consists of the sum of the reconstruction term and the regularization term to find optimal parameters for both terms.

Explaining anomaly detection has been used to increase interpretability for neural network model predictions. The Shapley values [35], a conditional game theory technique, can numerically express how much each feature contributed to the model results and then be utilized to interpret the results of anomaly detection. The value function of computing the Shapley values can be defined as the reconstruction error of the probabilistic principal component analysis (PCA) [36]. In particular, an LSTM-VAE and the probabilistic PCA are trained using the expectation-maximization algorithm (EM) to maximize the likelihood of the data by their parameterized Gaussian distribution.

Chapter 4

Threat Model

We assume that security bugs and vulnerabilities exist in any program that an attacker could exploit. Specifically, for our use cases, we assume that an attacker can use memory vulnerabilities to take control of a program's stack and perform arbitrary remote code execution. For example, memory vulnerabilities include integer overflow, heap buffer overflow, double-free, and use-after-free. We assume that protection techniques such as data execution protection (DEP) or ASLR are applied to the protection system. We assume that an attacker can bypass protection techniques through information leakage or java runtime environment (JRE), and that an attacker can access binary files and shared libraries to obtain ROP gadgets. Also, we assume that all hardware components on a target machine are reliable. Therefore, our study does not consider hardware-based attacks such as cache side-channel attacks [37].

Chapter 5

Our method: DROPSYS

5.1 Overview

Figure 5.1 shows the architecture of DROPSYS, which consists of four main components: system information utilization, data preprocessing, explainability, and anomaly detection. In system information utilization, two modules sample all HPC data and system utilization data.

System information utilization. DROPSYS first collects all necessary system information related to ROP attacks.

Data preprocessing. For data preprocessing, the collected HPC data and system utilization data transform into meaningful data, like instructions per cycle (IPC) or a Z-Score standardization [38], which DROPSYS uses later to enhance detection performance. DROPSYS also upsamples the data in every 1ms with zero and nearest interpolation. The upsampling module matches the time points and time intervals between the HPC data and system utilization data, and then it merges them into one data set.

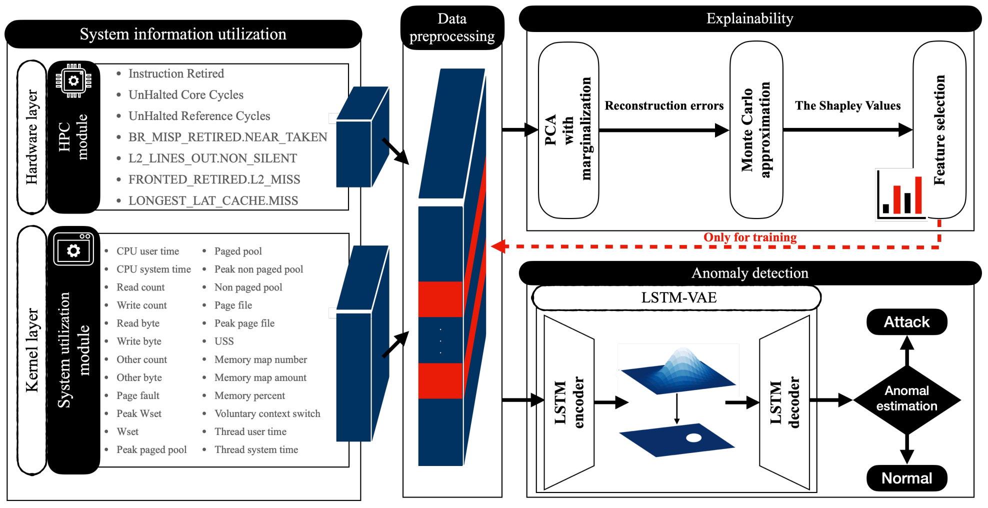

Figure 5.1: Overview of the DROPSYS architecture.

Explainability. In the explainability step, DROPSYS utilizes the probabilistic view of the PCA with marginalization to define the LSTM-VAE reconstruction errors. To compute the Shapley values, the value function is defined as a PCA reconstruction error. Given the value function, DROPSYS approximates the Shapley values using the Monte Carlo approximation. The Shapley values represent the contribution of each system information to the prediction results of DROPSYS. We select the features based on the top contribution to minimize the number of system information, which is used for an LSTM-VAE. By doing so, DROPSYS shows the optimal overhead while maintaining attack detection performance.

Anomaly detection. In anomaly detection, DROPSYS builds an LSTM-VAE by semi-supervised manner that employs only legitimate system information as a training data. Since it is hard to collect sufficient volume of real-world ROP attack exploits to train a DNN module, semi-supervised learing is affordable approach to build DNN module that identifies system information under benign operation. Based on the anomaly score, DROPSYS detects ROP attacks.

5.2 System information utilization

The distinctive feature of ROP attacks is found on the payload, which uses a chain of gadgets with only a few instructions [39]. On a system, the execution of the payload leaves several clues of an ROP attack, one of these being that the call instructions are not paired with return instructions during the execution of a program.

To capture the clues, DROPSYS not only monitors HPC data (Table 1.1) which is related to the behaviors of an ROP attack but also system utilization data (Table 3.1), enabling DROPSYS to analyze the effects of the behaviors. DROPSYS consists of two modules. The first module takes samples of system utilization data every 1ms through Python's `psutil` library [40]. We utilize the performance monitoring interrupt that considers context switches during the measurement of HPC data, which is referred to CS-PMI [20]. The second module takes samples of HPC data with CS-PMI of 2.9×10^6 cycles through a Windows kernel drive. CS-PMI prevents the HPC data of a target process from being contaminated by other processes, which enables per-process filtering.

Based on our observation of ROP attacks, DROPSYS recognizes four categories of system information that the attacks trigger in an abnormal way rather than a normal execution.

Branch misprediction. Branch misprediction is when a processor fails to predict the direction and destination of branch instructions by a branch prediction unit. More specifically, the branch prediction unit uses the return address stack (RAS) for call-return instructions[41]. If branch prediction does not match the actual branch, a CPU discards the prediction results from a pipeline and performs the actual branch. In the opposite case, the CPU uses the prediction results. ROP attacks degrade the performance of branch prediction because the attacks change the actual return address into an manipulated return address, which cause RAS entries not to match up consecutively. DROPSYS is able to capture the first behavior of ROP attacks using HPC data related to branch misprediction.

Level 2 (L2) cache eviction, miss and last-level cache (LLC) miss. L2 cache eviction as well as miss and LLC miss means that the desired data does not exist in either the L2 cache or the LLC. Note that the caches are what pass a program's executable code and data to a processor. Caches work well in a computer system because the processor accesses their code and data by utilizing the locality of reference [42]. In terms of the locality of reference, cache miss can be caused by requests for code and data that are not peripheral parts of the code or data being executed. It can also be caused by requests for recently unused code and data. ROP attacks artificially damage the locality of reference on caches and cause consecutive cache misses as these attacks utilize gadgets that are spread across the program's memory area. However, we do not use Level 1 (L1) cache miss in our technology. Another reason is that L1 cache miss is frequent in normal executions because the capacity of

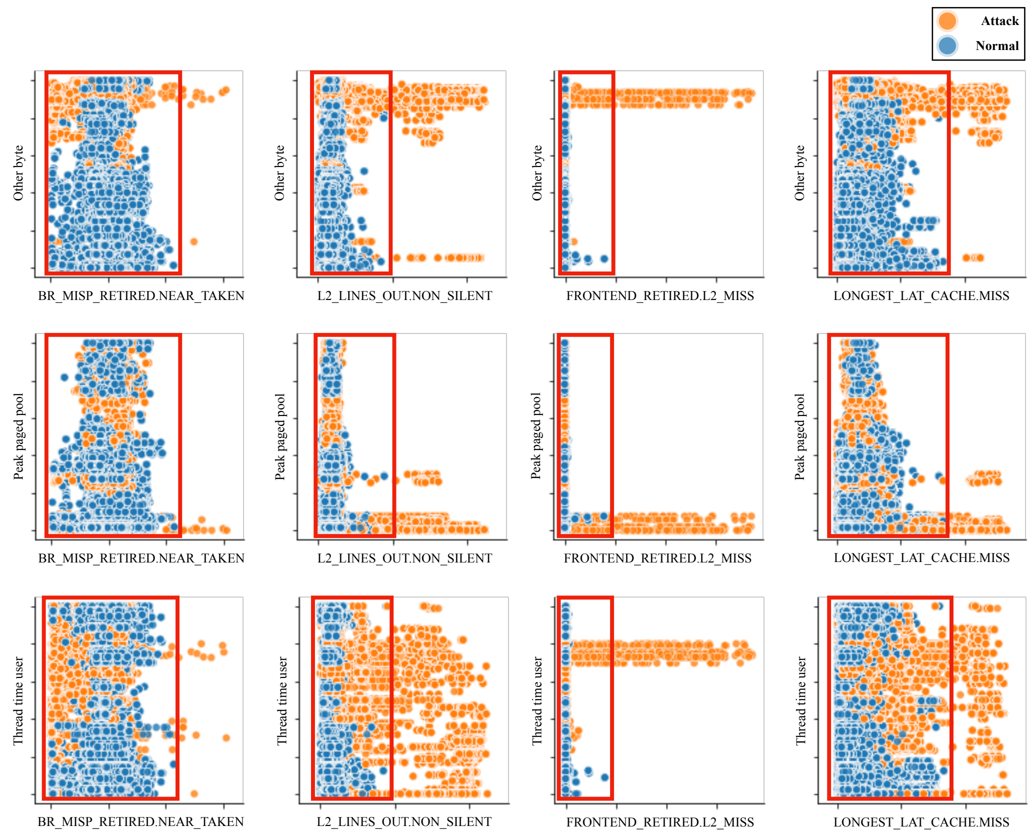

Figure 5.2: The scatter graph between HPC data and system utilization data.

L1 cache is small. For example, in the Ice Lake Client Microarchitecture, the capacity of L1 cache is 48KB, which is smaller than L2 cache (512KB) and the LLC (up to 2MB per core). DROPSYS catches the second behavior of ROP attacks using HPC data related to L2 cache eviction along with miss and LLC miss.

Instructions per cycle (IPC). The IPC refers to the average number of instructions executed for each clock cycle. ROP attacks decrease IPC and increase program execution time [43] due to branch misprediction, L2 cache eviction, and the miss and LLC miss, which consume cycles, but do not execute instructions. For example, if branch prediction is wrong, the prediction results are all discarded from the pipeline, and the correct instructions must be executed again. In case of a cache miss, the pipeline stops and brings about the penalty for a cache miss, which is an additional delay during which caches must receive the requested code and data. DROPSYS monitors the third behavior of ROP attacks using HPC data related to the IPC.

System utilization. System utilization is part of process context, which indicates how much the process uses each system resource. A process means that a program is allocated in the main memory and runs on a processor. The system utilization of a process is changed not only by ROP attacks directly but also by the aftereffects of the aforementioned three attack behaviors. Therefore, we utilize a correlation between HPC data and system utilization data. In Figure 5.2, the x-axis represents HPC data and the y-axis represents system utilization data as a Z-Score standardization [38]. Figure 5.2 shows the two distributions of data between a normal execution (blue) and an attack execution (orange). However, as shown in the red rectangle, it is difficult just by simple observation to distinguish clearly between a normal execution and an attack execution based on the correlation of the two data sets. To resolve the non-linear correlation between HPC data and system utilization data, DROPSYS utilizes an LSTM-VAE because it can faithfully reflect the characteristics of multivariate and nonlinear data.

Algorithm 1: Data preprocessing algorithm for system information.

```
    Input: $X_{hpc} \in \mathbb{R}^{samples \times features}$ $X_{util} \in \mathbb{R}^{samples \times features}$
    Output: $X_{sysInfo} \in \mathbb{R}^{resamples \times features}$
 1  $N1 \leftarrow samples$;
 2  $N2 \leftarrow features$;
 3  while $N1 \neq 0$ do
 4      $IPC1 \leftarrow \frac{X_{hpc}(N1,\ Instruction\ Retired)}{X_{hpc}(N1,\ UnHalted\ Core\ Cycles)}$;
 5      $IPC2 \leftarrow \frac{X_{hpc}(N1,\ Instruction\ Retired)}{X_{hpc}(N1,\ UnHalted\ Reference\ Cycles)}$;
 6      $X_{hpcA} \leftarrow [IPC1,\ IPC2,\ X_{hpc}(N1,\ others)]$;
 7  while $N2 \neq 0$ do
 8      $X_{utilZ} \leftarrow zscore(X_{util}(\ :\ ,\ N2))$;
 9  $X_{hpcR} \in \mathbb{R}^{resamples \times features} \leftarrow resample\ X_{hpcA}\ to\ 1ms$;
10  $X_{utilR} \in \mathbb{R}^{resamples \times features} \leftarrow resample\ X_{utilZ}\ to\ 1ms$;
11  $X_{hpcZ} \leftarrow interpolate\ X_{hpcR}\ with\ zero\ value$;
12  $X_{utilN} \leftarrow interpolate\ X_{utilR}\ with\ nearest\ value$;
13  $X_{sysInfo} \leftarrow [X_{hpcZ},\ X_{utilN}]$;
```

5.3 Data preprocessing

DROPSYS collects system information during the execution of a program and then transforms the collected system information into essential type of data. Algorithm 1 shows DROPSYS data preprocessing. To refine IPC, DROPSYS uses two kinds of cycles from HPC data and calculates IPC1 and IPC2. Next, DROPSYS applies a Z-Score standardization [38] on system utilization data to monitor any change of system utilization of a process.

Lastly, DROPSYS upsamples the measured HPC data and system utilization data at the time interval of 1ms to match the time intervals and time points between each interval. Missing values caused by upsampling are handled through interpolation. Zero interpolation is used to interpolate HPC data, and nearest interpolation is used for the interpolation of system utilization data.

5.4 Explainability

We utilize the technique for explaining anomaly detection [44] to interpret the prediction of DROPSYS. We define the value function that represents the predictive ability of DROPSYS by using the reconstruction error of probabilistic PCA. Furthermore, we use the marginalization technique [44] which simplifies the value function to improve efficiency in calculation of the Shapley values. Then, we calculate the Shapley values using the value function with the Monte Carlo approximation [45] that randomly selects some subsets of features. We measured contributions by the Shapley values for 30 system information to examine which data contributes to ROP attack detection. We used 110 data sets from normal executions to define the predictive ability of DROPSYS via the reconstruction error of probabilistic PCA. We set the PCA dimensionality to 8. The 110 data sets from ROP attack executions were used to calculate the Shapley values with the defined predictive ability. The number of Monte Carlo iterations was set to 1,000. The Shapley values were evaluated for each feature per sample of the data sets. We averaged contributions by each feature across all samples of the data sets. The

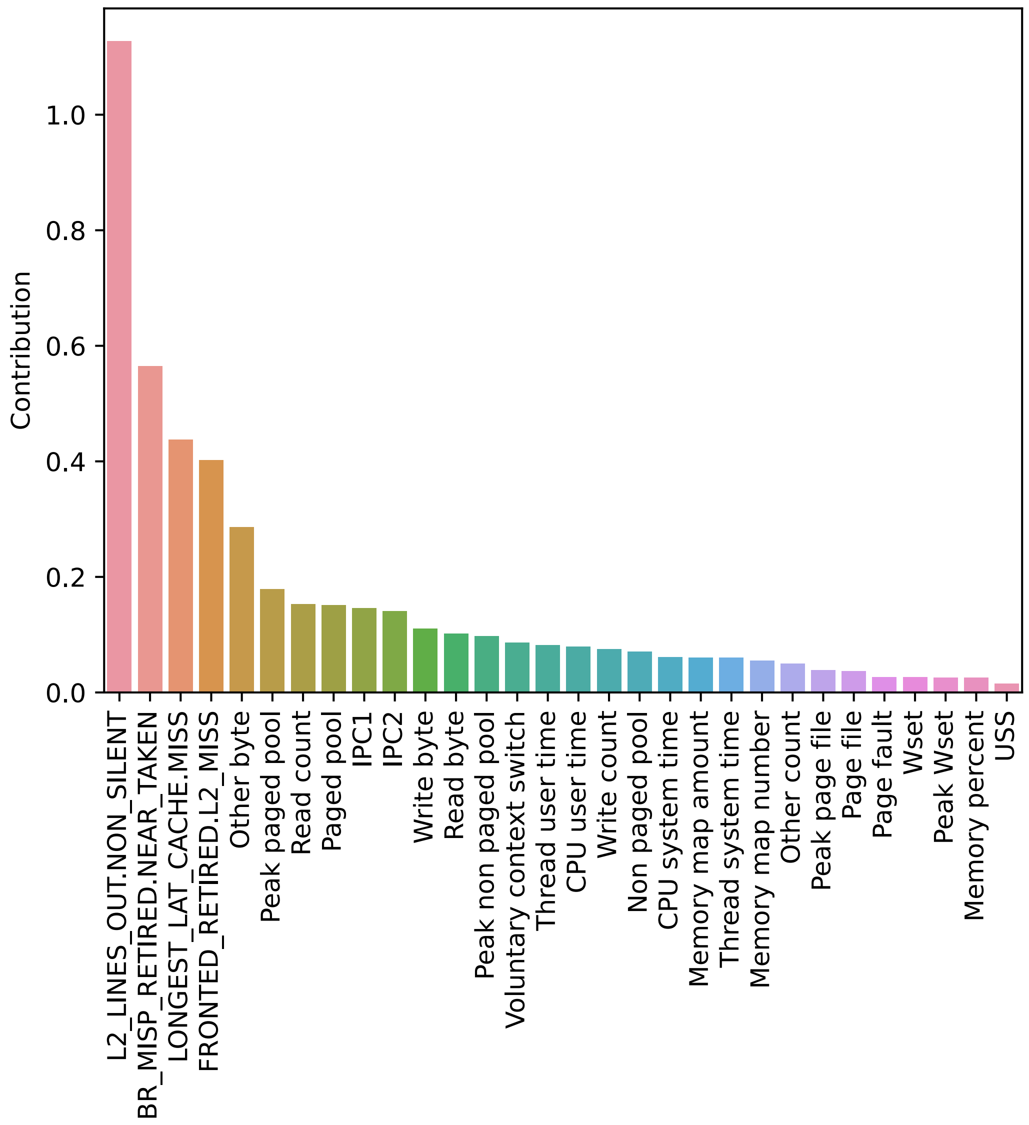

Figure 5.3: Contribution of system information.

contribution of 30 system information to ROP attacks is shown in Figure 5.3.

5.5 Anomaly detection

Anomaly detection is useful when there is a severe class imbalance between normal and abnormal data, or when there is insufficient abnormal data in comparison to normal data. To improve detection performance in a situation in which it is difficult to collect real-world ROP attack data, DROPSYS is based on semi-supervised anomaly detection that does not require prior information about the attack during model training [34].

DROPSYS uses an LSTM-VAE because the LSTM-VAE for ROP attack detection can effectively reflect the temporal dependencies of multidimensional time-series system information according to a normal program execution because it models the distribution of normal multidimensional time series data and reconstructs the data with an expected distribution. The LSTM-VAE receives 3D sequences ($samples \times time\ window\ size \times features$) as input data. To distinguish input data between normal execution and ROP attack execution, we use the LSTM-VAE reconstruction error as an anomaly score. The reconstruction error is calculated as mean squared error (MSE). A high anomaly score indicates that the input data was not well reconstructed by the LSTM-VAE. To obtain optimal threshold, we set an anomaly score's threshold to the highest value at which the difference between precision and recall scores is minimized using the precision-recall curve.

Chapter 6

Implementation and Evaluation

We implemented a prototype of DROPSYS in a target machine running Windows 7 Professional SP1. On the attack machine running Kali Linux, we used the Metasploit framework to execute the ROP attacks in Table 6.1. In DROPSYS, we implemented an LSTM-VAE using a stateless LSTM model in the Keras' deep learning library [46]. We trained our LSTM-VAE with the Adam optimizer at a 0.0001 learning rate and used LSTM layers with tanh.

Our evaluation addresses the following questions:

- **RQ 1. How much advantage from explainability does DROPSYS gain?**
- **RQ 2. How accurately does DROPSYS capture abnormal system information?**
- **RQ 3. How well does DROPSYS detect ROP attacks?**

Table 6.1: Real-world ROP attacks from the Metasploit framework.

Index	CVEs	Description
1	2011-1996	Internet Explorer Option Element Use-After-Free
2	2012-1535	Adobe Flash Kern Parsing Integer Overflow
3	2012-1875	Internet Explorer Same ID Property
4	2012-1876	Internet Explorer Fixed Table Col Span Heap Overflow
5	2012-4792	Internet Explorer CButton Use-After-Free
6	2012-4969	Internet Explorer Execommand Use-After-Free
7	2013-1347	Internet Explorer CGenericElement Use-After-Free
8	2013-2551	Internet Explorer COALineDashStyleArray Integer Overflow
9	2013-3897	Internet Explorer CDisplayPointer Use-After-Free
10	2014-0569	Adobe Flash Casi32 Integer Overflow
11	2015-0359	Adobe Flash Domain Memory Use-After-Free

In our evaluation environment, we set up both the target and attack machines with an Intel Core i7-10700 2.9Ghz (Ice Lake Client Microarchitecture). Configured with one total processor core, we used VMware virtualization to run both machines.

6.1 Data collection

We used 11 sets of scenarios with normal executions and 11 sets of ROP attack scenarios to collect system information. The scenarios proceed with the interaction between a target machine and an attack machine. In both the normal and attack scenarios, DROPSYS on the target machine monitors

Internet Explorer. Internet Explorer in the normal scenario accesses the URL of a general web search engine. Internet Explorer in the attack scenario accesses the URL provided by the Metasploit framework on the attack machine. The attack scenarios consist of two stages. In the first stage, reverse_tcp and exploit-specific code are sent as payloads and executed; ROP attacks occur at this stage. In the second stage, the payload for the meterpreter session connection is transmitted and executed. Note that we only monitor ROP attacks during the first stage. We check the start time of the second stage through Wireshark and collect any data present before that time as attack data.

In general, one data set from the observation contains about 2,500 samples (i.e., $2,500 \times 30$). We used the 770 data sets we obtained by monitoring 11 sets of scenarios with normal executions and 11 sets of ROP attack scenarios 70 times each. We divided the 770 data sets into 550 (11×50) training/validation data sets and 220 (11×20) test data sets. All 550 training/validation data sets (11×50) were obtained from normal scenarios, and 20% of the training data sets are validation data sets. The test data sets consist of 110 (11×10) data sets from normal scenarios and 110 (11×10) data sets from attack scenarios. We standardized the training/validation data sets with Scikit-learn's StandardScaler and scaled the test data sets with Scaler used for the training/validation data sets.

6.2 Baseline techniques

To evaluate the performance of the proposed method, we implemented the following baseline techniques:

- SIGDROP [15]: SIGDROP is a signature-based ROP detection using HPC data. We employed the identical HPC data and thresholds to implement the SIGDROP and set the average number of instructions in a gadget X = 3. We set up both the target and attack machines with an Intel Xeon Silver 4208 2.1Ghz.

$$INST\_RETIRED.ANY \leq BR\_MISP\_RETIRED.RET \times X \tag{6.1}$$

- ROPSENTRY [16]: ROPSENTRY is a runtime defense against ROP attacks using HPC data. We collected the identical HPC data and selected the thresholds as described in [16]. We set up both the target and attack machines with an Intel Xeon Silver 4208 2.1Ghz.

$$\frac{BR\_MISP\_RETIRED.RET}{BR\_INST\_RETIRED.NEAR\_RETURN} \geq 0.9 \tag{6.2}$$

$$\frac{BR\_INST\_RETIRED.NEAR_{R}ETURN}{INST\_RETIRED.ANY} \geq 0.2 \tag{6.3}$$

$$\frac{FRONTEND\_RETIRED.ITLB\_MISS}{INST\_RETIRED.ANY} \geq 0.8 \tag{6.4}$$

$$\frac{LONGEST\_LAT\_CACHE.MISS}{INST\_RETIRED.ANY} \geq 2.0 \tag{6.5}$$

- LSTM-AE [47]: To compare the attack detection performance of DROP-SYS against LSTM-based anomaly detection techniques, we employed an LSTM autoencoder (LSTM-AE) as another baseline technique. We set a window size of LSTM-AE as 25, and computed the reconstruction error to detect the ROP attacks.

Table 6.2: The change of detection performance in view of the number of features with four metrics.

Detection Performance	**Features (K)**				
	10	15	20	25	30
Accuracy	0.850	0.885	0.912	0.890	0.878
False Positive Rate	0.100	0.099	0.049	0.080	0.088
F1 Score	0.826	0.876	0.902	0.880	0.867
AUC-ROC	0.924	0.958	0.975	0.954	0.954

Table 6.3: The change of detection performance in view of the size of time window with four metrics.

Detection performance	**Time window (T)**			
	15	25	35	50
Accuracy	0.915	0.927	0.919	0.912
False Positive Rate	0.064	0.046	0.037	0.049
F1 Score	0.909	0.921	0.909	0.902
AUC-ROC	0.967	0.976	0.974	0.975

6.3 Detection performance

RQ 1. How much advantage from explainability does DROPSYS gain? Table 6.2 shows the average attack detection performance of 11 attack

Table 6.4: The detection performance with four metrics on two data sets.

CVEs	Data sets based on HPC data				Data sets based on system information			
	Accuracy	False Positive Rate	F1 Score	AUC-ROC	Accuracy	False Positive Rate	F1 Score	AUC-ROC
2011-1996	0.507	0.054	0.120	0.580	0.923	0.018	0.894	0.978
2012-1535	0.649	0.095	0.523	0.744	0.856	0.045	0.840	0.937
2012-1875	0.591	0.046	0.357	0.768	0.960	0.061	0.962	0.983
2012-1876	0.552	0.056	0.262	0.635	0.965	0.042	0.964	0.988
2012-4792	0.489	0.086	0.109	0.502	0.930	0.004	0.918	0.978
2012-4969	0.516	0.070	0.171	0.634	0.927	0.063	0.923	0.984
2013-1347	0.781	0.049	0.737	0.859	0.983	0.023	0.983	0.998
2013-2551	0.728	0.071	0.660	0.839	0.917	0.085	0.918	0.978
2013-3897	0.780	0.066	0.738	0.834	0.956	0.084	0.960	0.997
2014-0569	0.527	0.045	0.171	0.655	0.868	0.043	0.855	0.956
2015-0359	0.702	0.047	0.601	0.775	0.914	0.033	0.909	0.962
Average	**0.620**	**0.062**	**0.404**	**0.711**	**0.927**	**0.046**	**0.921**	**0.976**

Table 6.5: The Overhead in view of the number of features with three metrics.

Overhead	**Features (K)**				
	10	15	20	25	30
Test elapsed time (s)	105.25	104.98	105.59	111.37	110.35
Model size (kb)	836	871	910	949	994
Total parameters	50790	53440	56340	59490	62890

scenarios according to the numbers of features (K is the number of features selected from the top contribution based on the Shapley values in Figure 5.3). We set DROPSYS with a time window size as 50 and evaluated the average detection performance under 11 scenarios. It is observed that DROPSYS obtains the best performance when K is set to 20 (i.e., 20 features) When the number of selected features exceeds 20, it degrades the attack detection performance. Overall, the results suggest that explainability can reduce the complexity of the model while improving detection performance.

RQ 2. How accurately does DROPSYS capture abnormal system information? Table 6.3 summarizes the results for DROPSYS from window times 15 to 50 (Note that T is the size of time window). We evaluated the average detection performance of DROPSYS about 11 scenarios when using the top 20 features of the Shapley values. We observed that DROPSYS has the best performance when T = 25. Table 5 shows that T slightly improves the detection performance of DROPSYS, which means that balancing the time windows is useful. Next, we evaluated the reconstruction function of DROPSYS.

The upper six subgraphs in Figure 6.1 show the observation of system information from both normal and attack executions in Internet Explorer. Compared to Figure 6.1b and 6.1c, Figure 6.1a does not show the abnormal series of values in the observed system information. Consequently, we observed that the anomaly score in the seventh subgraphs of Figure 6.1b and 6.1c dramatically increases, since the abnormal series of system information is not easily observable from normal executions. The pink vertical lines show the time of anomaly detection when the first execution of ROP matches with the initial occurrence of abnormal system information.

RQ 3. How well does DROPSYS detect ROP attacks? We evaluated the difference in detection performance of DROPSYS between data sets based on HPC data and data sets based on system information. Table 6.4 shows the evaluation results for accuracy, false positive rates, F1 scores, and

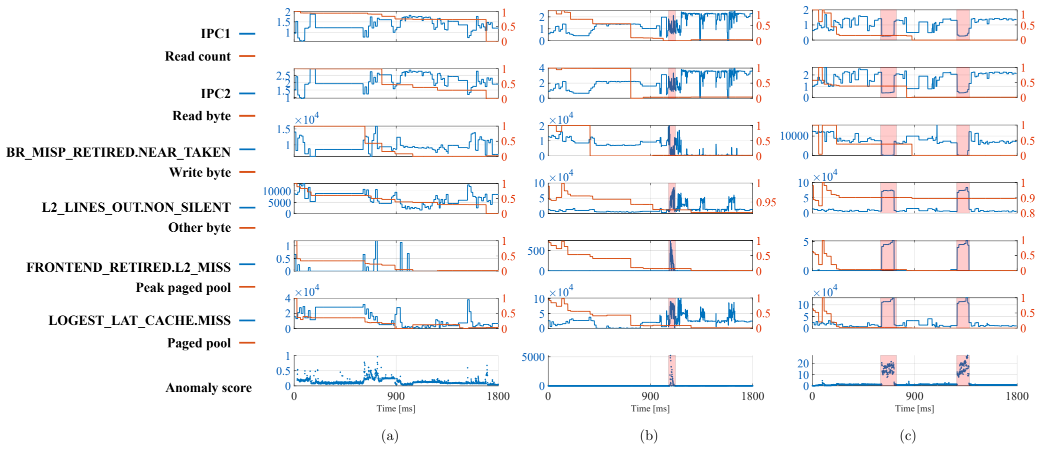

Figure 6.1: Visualization of system information and the anomaly score over time. The upper six graphs show observations. The lower subgraphs show the anomaly scores. The pink vertical lines represent the time of ROP attack detection. (a) A normal execution. (b) An attack execution with CVE-2015-0359. (c) An attack execution with CVE-2012-1876.

AUC-ROC. The LSTM-VAE with HPC data only shows average accuracy as 0.620 and the average F1 score as 0.404. However, DROPSYS has an average accuracy of 0.927 and an average F1 score of 0.921. These results shows that the ROP attack detection techniques based on system information using both HPC data and system utilization data has good performance. In addition, we observed that both techniques—based on HPC data or based on system information—have very low false positive rates of 0.062 and 0.046, respectively. Finally, the deep learning-based approach also mitigated a drawback of existing HPC-based ROP detection techniques, i.e., the false prediction rate for the positive class. We also compared DROPSYS with three other baseline techniques. Note that we did not need to train or validate SIGDROP

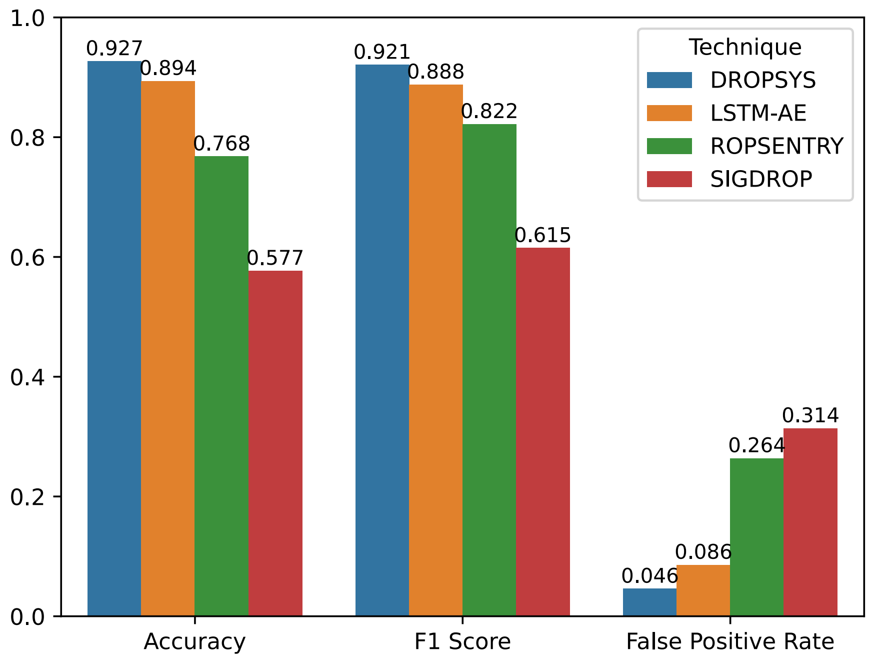

Figure 6.2: Comparison of DROPSYS and three baseline techniques.

and ROPSENTRY because these are rule-based techniques and not machine learning based. Figure 6.2 shows the evaluation results for accuracy, false positive rates, F1 scores, and AUC-ROC. SIGDROP represents an average accuracy of 0.577 across 11 ROP scenarios and an average F1 score of 0.615. In addition, ROPSENTRY is evaluated through experiments with an average accuracy of 0.768 and an average F1 score of 0.822. These results show that the accuracy of 0.927 and F1 score of 0.921 of DROPSYS (i.e., detection performance) are superior to these existing techniques. In addition, the average false positive rates of SIGDROP and ROPSENTRY are 0.314 and 0.264,

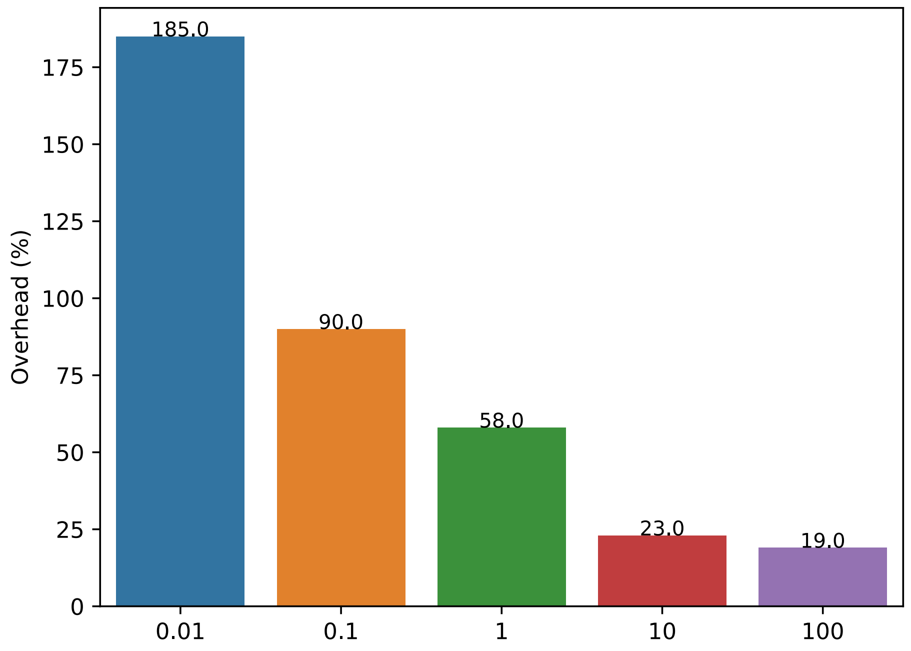

Figure 6.3: Overhead with regard to sampling rate (ms).

which are both much higher than the 0.046 false positive for DROPSYS. These results indicate that DROPSYS alleviates the false positive problem that is present in existing HPC-based ROP detection techniques.

In addition to the research questions, we observed performance overhead. Table 6.5 shows the results of the advantage from the explanability of DROP-SYS. We evaluated the three metrics with using 188,304 samples. As a result, The selected 20 features based on the explanability of DROPSYS show the lowest performance overhead among others. In addition, Figure 6.3 shows the process' private working set overhead with regard to sampling rate The private working set represents the size of the memory for one process.

Chapter 7

Conclusion

In this paper, we presented a new technology called DROPSYS, which is a comprehensive ROP attack detection technique that has proven effective detection performance without additional cost or compatibility issues. By additionally leveraging system information, DROPSYS mitigates the non-determinism of HPCs and the limited number of available HPCs as well as utilizes an LSTM-VAE and explainability to capture the behaviors and effects of ROP attacks. We subsequently demonstrated that DROPSYS can successfully detect ROP attacks at a high rate as a result of our fine-grained analysis of real-world ROP exploits.

Bibliography

[1] Laszlo Szekeres, Mathias Payer, Tao Wei, and Dawn Song. Sok: Eternal war in memory. In *2013 IEEE Symposium on Security and Privacy*, pages 48–62. IEEE, 2013.

[2] Tim Rains, Matt Miller, and David Weston. Exploitation trends: From potential risk to actual risk. In *RSA Conference*, 2015.

[3] Matt Miller, Tim Burrell, and Michael Howard. Mitigating software vulnerabilities, july 2011.

[4] Giampaolo Fresi Roglia, Lorenzo Martignoni, Roberto Paleari, and Danilo Bruschi. Surgically returning to randomized lib (c). In *2009 Annual Computer Security Applications Conference*, pages 60–69. IEEE, 2009.

[5] Peter Vreugdenhil. Pwn2own 2010 windows 7 internet explorer 8 exploit, 2010.

[6] Hovav Shacham, Matthew Page, Ben Pfaff, Eu-Jin Goh, Nagendra Modadugu, and Dan Boneh. On the effectiveness of address-space ran-

domization. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 298–307, 2004.

[7] Lucas Davi, Matthias Hanreich, Debayan Paul, Ahmad-Reza Sadeghi, Patrick Koeberl, Dean Sullivan, Orlando Arias, and Yier Jin. Hafix: Hardware-assisted flow integrity extension. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2015.

[8] Ruan De Clercq and Ingrid Verbauwhede. A survey of hardware-based control flow integrity (cfi). *arXiv preprint arXiv:1706.07257*, 2017.

[9] Sandeep Kumar, Diksha Moolchandani, and Smruti R Sarangi. Hardware-assisted mechanisms to enforce control flow integrity: A comprehensive survey. *Journal of Systems Architecture*, page 102644, 2022.

[10] Yu Wang, Jinting Wu, Tai Yue, Zhenyu Ning, and Fengwei Zhang. Rettag: hardware-assisted return address integrity on risc-v. In *Proceedings of the 15th European Workshop on Systems Security*, pages 50–56, 2022.

[11] Liwei Yuan, Weichao Xing, Haibo Chen, and Binyu Zang. Security breaches as pmu deviation: detecting and identifying security attacks using performance counters. In *Proceedings of the Second Asia-Pacific Workshop on Systems*, pages 1–5, 2011.

[12] Nishad Herath and Anders Fogh. These are not your grand daddys cpu performance counters–cpu hardware performance counters for security. *Black Hat Briefings*, 2015.

[13] Yubin Xia, Yutao Liu, Haibo Chen, and Binyu Zang. Cfimon: Detecting violation of control flow integrity using performance counters. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*, pages 1–12. IEEE, 2012.

[14] HongWei Zhou, Xin Wu, WenChang Shi, JinHui Yuan, and Bin Liang. Hdrop: Detecting rop attacks using performance monitoring counters. In *International Conference on Information Security Practice and Experience*, pages 172–186. Springer, 2014.

[15] Xueyang Wang and Jerry Backer. Sigdrop: Signature-based rop detection using hardware performance counters. *arXiv preprint arXiv:1609.02667*, 2016.

[16] Sanjeev Das, Bihuan Chen, Mahintham Chandramohan, Yang Liu, and Wei Zhang. Ropsentry: Runtime defense against rop attacks using hardware performance counters. *Computers & Security*, 73:374–388, 2018.

[17] David Pfaff, Sebastian Hack, and Christian Hammer. Learning how to prevent return-oriented programming efficiently. In *International Symposium on Engineering Secure Software and Systems*, pages 68–85. Springer, 2015.

[18] Adrian Tang, Simha Sethumadhavan, and Salvatore J Stolfo. Unsupervised anomaly-based malware detection using hardware features. In *International Workshop on Recent Advances in Intrusion Detection*, pages 109–129. Springer, 2014.

[19] Vincent M Weaver, Dan Terpstra, and Shirley Moore. Non-determinism and overcount on modern hardware performance counter implementations. In *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 215–224. IEEE, 2013.

[20] Sanjeev Das, Jan Werner, Manos Antonakakis, Michalis Polychronakis, and Fabian Monrose. Sok: The challenges, pitfalls, and perils of using hardware performance counters for security. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 20–38. IEEE, 2019.

[21] Metasploit. Metasploit — penetration testing software, pen testing security — metasploit. `https://www.metasploit.com/`. (Accessed on 12/16/2022).

[22] Enes Göktas, Elias Athanasopoulos, Herbert Bos, and Georgios Portokalidis. Out of control: Overcoming control-flow integrity. In *2014 IEEE Symposium on Security and Privacy*, pages 575–589. IEEE, 2014.

[23] Vasilis Pappas, Michalis Polychronakis, and Angelos D Keromytis. Transparent {ROP} exploit mitigation using indirect branch tracing. In *22nd USENIX Security Symposium (USENIX Security 13)*, pages 447–462, 2013.

[24] X Li and M Crouse. Transparent rop detection using cpu performance counters. *THREADS*, 2014.

[25] Corey Malone, Mohamed Zahran, and Ramesh Karri. Are hardware performance counters a cost effective way for integrity checking of pro-

grams. In *Proceedings of the sixth ACM workshop on Scalable trusted computing*, pages 71–76, 2011.

[26] John Demme, Matthew Maycock, Jared Schmitz, Adrian Tang, Adam Waksman, Simha Sethumadhavan, and Salvatore Stolfo. On the feasibility of online malware detection with performance counters. *ACM SIGARCH Computer Architecture News*, 41(3):559–570, 2013.

[27] Baljit Singh, Dmitry Evtyushkin, Jesse Elwell, Ryan Riley, and Iliano Cervesato. On the detection of kernel-level rootkits using hardware performance counters. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 483–493, 2017.

[28] Yinqian Zhang and Michael K Reiter. Düppel: Retrofitting commodity operating systems to mitigate cache side channels in the cloud. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 827–838, 2013.

[29] Ryan Roemer, Erik Buchanan, Hovav Shacham, and Stefan Savage. Return-oriented programming: Systems, languages, and applications. *ACM Transactions on Information and System Security (TISSEC)*, 15(1):1–34, 2012.

[30] Part Guide. Intel® 64 and ia-32 architectures software developer's manual. *Volume 3*, 2022.

[31] Alberto Avritzer, Rajanikanth Tanikella, Kiran James, Robert G Cole, and Elaine Weyuker. Monitoring for security intrusion using perfor-

mance signatures. In *Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering*, pages 93–104, 2010.

[32] EN Preeth, Fr Jaison Paul Mulerickal, Biju Paul, and Yedhu Sastri. Evaluation of docker containers based on hardware utilization. In *2015 international conference on control communication & computing India (ICCC)*, pages 697–700. IEEE, 2015.

[33] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin, and Kuang-Yuan Tung. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1):16–24, 2013.

[34] Jason Andress. *The basics of information security: understanding the fundamentals of InfoSec in theory and practice.* Syngress, 2014.

[35] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

[36] Ian T Jolliffe. *Principal component analysis for special types of data.* Springer, 2002.

[37] Daniel Gruss, Raphael Spreitzer, and Stefan Mangard. Cache template attacks: Automating attacks on inclusive {Last-Level} caches. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 897–912, 2015.

[38] Masashi Sugiyama. *Section 2.5 – transformation of random variables*, page 22–23. MK Morgan Kaufmann, 2016.

[39] Yueqiang Cheng, Zongwei Zhou, Yu Miao, Xuhua Ding, and Robert H Deng. Ropecker: A generic and practical approach for defending against rop attack. 2014.

[40] Giampaolo Rodola. Psutil package: a cross-platform library for retrieving information on running processes and system utilization. *Google Scholar*, 2016.

[41] Kevin Skadron, Pritpal S Ahuja, Margaret Martonosi, and Douglas W Clark. Improving prediction for procedure returns with return-address-stack repair mechanisms. In *Proceedings. 31st Annual ACM/IEEE International Symposium on Microarchitecture*, pages 259–271. IEEE, 1998.

[42] Peter J Denning. The locality principle. *Communications of the ACM*, 48(7):19–24, 2005.

[43] Stijn Eyerman, James E Smith, and Lieven Eeckhout. Characterizing the branch misprediction penalty. In *2006 IEEE International Symposium on Performance Analysis of Systems and Software*, pages 48–58. IEEE, 2006.

[44] Naoya Takeishi. Shapley values of reconstruction errors of pca for explaining anomaly detection. In *2019 international conference on data mining workshops (icdmw)*, pages 793–798. IEEE, 2019.

[45] Erik Strumbelj and Igor Kononenko. An efficient explanation of individual classifications using game theory. *The Journal of Machine Learning Research*, 11:1–18, 2010.

[46] François Chollet et al. Keras documentation. *keras. io*, 33, 2015.

[47] Pankaj Malhotra, Anusha Ramakrishnan, Gaurangi Anand, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. Lstm-based encoder-decoder for multi-sensor anomaly detection. *arXiv preprint arXiv:1607.00148*, 2016.