Computers & Security 140 (2024) 103813

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

DROPSYS: Detection of ROP attacks using system information

Seon Kwon Kim [a], Hongjoo Jin [b], Kyungho Joo [c], Jiwon Lee [b], Dong Hoon Lee [b,*]

[a] Department of Cyber Security, Korea University Graduate School, Seoul, Republic of Korea
[b] Graduate School of Information Security, Korea University, Seoul, Republic of Korea
[c] School of Software, Soongsil University, Seoul, Republic of Korea

ARTICLE INFO

Keywords:
Software security
Control/data flow integrity
ROP attacks
Software side channels
System information
Exploit defense
LSTM-VAE

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. DROPSYS is based on the detection of the abnormal change of system information that takes place during ROP attacks. Differing from the existing techniques, DROPSYS harnesses not only HPC data, but also system utilization data to mitigate the non-deterministic nature of HPCs. Using both HPCs of processors and system utilization of operating systems makes transparent operation without requiring any modifications to the protected programs. DROPSYS uses a long short-term memory-based variational autoencoder to effectively analyze the multivariate time-series HPC data and system utilization data for better detection performance. DROPSYS also performs feature selection for low computational overhead while maintaining the attack detection performance.

In our experiments with real-world ROP exploits, DROPSYS successfully detected ROP code execution in all tested programs. Evaluation results show that DROPSYS effectively captures the behaviors and effects of ROP attacks and can detect the attacks with a 0.028% false positive rate. The accuracy of DROPSYS is 95.3%, and its F1 score is 94.9%—a figure much higher than those of existing techniques that utilize only HPC data.

1. Introduction

Return-oriented programming (ROP) has been used in malicious codes to extract sensitive information and install malware (Shacham, 2007; Szekeres et al., 2013). Reports cite that 80% of exploits in software vulnerabilities are made through ROP (Rains et al., 2015). Since modern exploits have become sophisticated, preventing or detecting ROP is imperative for real-world exploit defense. For example, to prevent attackers from analyzing and manipulating program memory, address space layout randomization (ASLR) (Miller et al., 2011) randomly arranges the virtual memory address space, and data execution prevention (DEP) (van de Ven, 2004) prevents code injection by attackers from being executed. Attackers who use ROP, however, can choose from among various techniques (Roglia et al., 2009; Vreugdenhil, 2010; Shacham et al., 2004) that are based on information leakage and brute force attacks to bypass ASLR and DEP.

To ensure that a program is safely protected against ROP attackers, researchers have developed a number of defenses such as hybrid defense techniques using both software and hardware (Davi et al., 2015; De Clercq and Verbauwhede, 2017; Kumar et al., 2022; Wang et al., 2022; Shanbhogue et al., 2019). Although hybrid defense techniques are practical in terms of performance, these come with additional hardware cost and compiler compatibility issues. For example, RetTag (Wang et al., 2022) requires additional hardware support to check the pointer authentication code (PAC) in every function call with a return address authentication key. HAFIX (Davi et al., 2015) 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. Intel has recently developed Control-flow Enforce-

* Corresponding author.
E-mail addresses: godmakessky@gmail.com (S.K. Kim), realredwine@korea.ac.kr (H. Jin), khjoo@ssu.ac.kr (K. Joo), hisdory728@korea.ac.kr (J. Lee), donghlee@korea.ac.kr (D.H. Lee).

https://doi.org/10.1016/j.cose.2024.103813
Received 24 May 2023; Received in revised form 9 January 2024; Accepted 7 March 2024
Available online 13 March 2024

ment Technology (CET) (Shanbhogue et al., 2019) to protect both forward and backward indirect control flows. Intel CET is a hybrid defense technique that is composed of two main components: shadow stack and indirect branch tracking. Shadow stack is a second stack used for ensuring integrity by comparing its return addresses with the main stack's return addresses. Indirect branch tracking is a technique that uses the new branch termination instructions to prevent attackers from jumping to arbitrary memory addresses. However, similar to RetTag and HAFIX, Intel CET also requires supported Windows builds and chipsets for its application (note that Intel CET is available across the 11th Gen Intel Core processors and later). Furthermore, applications that are not compiled with Intel CET support may still be vulnerable to attacks even if the underlying hardware supports Intel CET.

To overcome this limitation, several techniques (Yuan et al., 2011; Herath and Fogh, 2015; Xia et al., 2012; Zhou et al., 2014; Wang and Backer, 2016; Das et al., 2018; Pfaff et al., 2015; Tang et al., 2014) 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 stemming from the non-deterministic nature of HPCs (Weaver et al., 2013). More importantly, the techniques failed to take into account the non-determinism of HPCs in their design process (Das et al., 2019).

The research problems that we deal with encompass two aspects. The first problem pertains to additional costs and compatibility issues that arise when applying a technique to a system. The second problem involves the low ROP attack detection performance observed in previous research. In response to these, the paper proposes DROPSYS, an enhanced detection of ROP attacks using system information. DROPSYS is a transparent technique that provides runtime protection to third-party applications against ROP attacks. Since DROPSYS utilizes system information that is available by default in systems and does not require any modifications to existing binaries, it can be easily deployed and can protect applications that are already installed on end-user systems. System information includes HPC data from the hardware layer and system utilization data from the kernel layer. Existing techniques rely solely on HPC data, exposing issues such as measurement instability due to the non-deterministic nature of HPC and limitations in analyzing ROP attacks because of the constrained number of HPCs (e.g., 7 HPCs in Intel CPUs). DROPSYS addresses these challenges by advancing its approach, integrating system utilization data for a more comprehensive analysis. This addition is crucial as HPC data alone may not adequately represent the features associated with ROP attacks. Since the attributes of ROP code manipulate a program counter and a stack pointer through return instructions, they cause unanticipated control flow in programs. The execution of ROP code affects not only instruction processing by processors but also operations on processes, resulting in overall abnormal system behavior.

For detecting ROP attacks, DROPSYS uses a generative model that learns the distribution of system information using semi-supervised learning: a long short-term memory-based variational autoencoder (LSTM-VAE) (Park et al., 2018). An LSTM-VAE utilizes LSTM for both an encoder and a decoder networks in VAE, which can model the distribution of time-series data with its temporal dependencies and generate new data that is similar to the input data. DROPSYS uses LSTM-VAE to detect ROP attacks by learning normal program behavior through multivariate time-series and nonlinear system information. DROPSYS calculates reconstruction errors between the original input and the results of the decoding via mean squared error (MSE). DROPSYS detects any anomaly as an ROP attack when the reconstruction error is higher than a set threshold.

Furthermore, DROPSYS utilizes the Shapley value (Lundberg and Lee, 2017) to explain the degree of contribution of each feature, enabling the interpretation of which system information is relevant to ROP attacks. Based on the Shapley value, we conduct feature selection to improve detection performance and reduce overhead. It turns out that the selected system information effectively reflects the characteristics of ROP attacks.

We evaluated DROPSYS using real-world ROP exploits from the Metasploit framework (Metasploit, 2022) that provides penetration test. Our evaluation results show that DROPSYS is able to detect the behaviors and effects of ROP attacks. DROPSYS achieves an average accuracy rate of 95.3% and an average F1 score of 94.9%, which outperformed previous HPCs-based approaches (Wang and Backer, 2016; Das et al., 2018). Besides this, DROPSYS reduces the false positive rate by about 91% compared to previous approaches (Wang and Backer, 2016; Das et al., 2018).

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 characteristics of ROP attacks. By using system information, DROPSYS mitigates the effect of HPC non-determinism in regard to ROP attack detection and overcomes the limited number of available HPCs.
- **A transparent ROP attack detection technique.** DROPSYS does not require source code or symbolic debugging information as it operates independently from monitored programs. This makes DROPSYS a cost-effective and compatible technique for protecting applications.
- **Accurate ROP attack detection.** DROPSYS utilizes an LSTM-VAE which can analyze multivariate time-series system information. DROPSYS enhances the detection performance of ROP attacks. Specifically, it improves both accuracy and the F1 score, substantially reducing the false positive rate 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 describes the methodology of DROPSYS, and Section 5 presents our evaluation results. Section 6 discusses the limitations and outlines future work for DROPSYS. Section 7 summarizes related works, and Section 8 concludes this paper.

2. Background

2.1. Return-oriented programming

ROP attacks exploit various memory vulnerabilities to manipulate the control flow of a program for the intended direction of an attacker (Szekeres et al., 2013). Especially, ROP attacks exploit instruction sets, which exist in executable segments of the vulnerable process, without requiring the attacker to inject any code into the process (Roemer et al., 2012). 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. When constructing an attack payload, an ROP attacker essentially links the addresses of gadgets. The link is called as a gadget chain. The gadget chain runs consecutively according to a sequence of gadgets contained in the 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. The gadgets execute consecutively and manipulate the program using the control flow that the attacker desires.

2.2. Hardware performance counters

HPCs stand for special-purpose register sets built into the performance monitoring unit (PMU) of a processor. Evaluating various processor performances, the PMU monitors hardware performance events through a set of model-specific HPCs. The PMU provides the interface to read the data of hardware performance events of interest from HPCs. Especially, HPCs can be read by either event-based sampling or polling (Das et al., 2019). Event-based sampling reads HPCs by using Performance Monitoring Interrupt (PMI), which can be generated after the occurrence of a certain number of hardware performance events. Polling reads HPCs by using the *rdpmc* instructions which are inserted in the desired locations in a program's code.

The number of available HPCs is diverse between processors. In general, HPCs are composed of three fixed counters and four programmable counters per core in Intel CPUs. Three fixed counters are used for monitoring fixed hardware performance events such as Instruction Retired, UnHalted Core Cycles, and UnHalted Reference Cycles. On the other hand, four programmable counters are used for monitoring other hardware performance events.

2.3. System information

We define system information as a combination of HPC data in the hardware layer and system utilization data in the kernel layer. First, note that HPC data is information about hardware performance events measured through HPCs. Modern processors provide many hardware performance events. Hardware performance events can be divided into architectural or non-architectural events. The architectural events are identical in the majority of processors. However, in contrast, the non-architectural events are specific to each generation of processors. Hardware performance events for Intel processors can be found in the Intel manual (Guide, 2015). Next, system utilization data relates to any system resources that a process uses. In other words, system utilization data indicates how many system resources that the process is occupying. For example, system utilization data includes information on CPU usage, memory, disks, network, and so on (Avritzer et al., 2010; Preeth et al., 2015).

2.4. Anomaly detection explanation

Anomaly detection is a technique used to detect anomalies that are distinct from normal data (Chandola et al., 2009; Liao et al., 2013; Andress, 2014). For detecting anomalies, a generative model learns the distribution of the training data sets so that it reconstructs the input data to measure reconstruction error with the anomaly score.

An autoencoder (AE) is a deep generative model that consists of both an encoder which compresses the data from the input space to the latent space (encoded space) and a decoder which decompresses the compressed data. The problem of an AE is overfitting during learning the encoding/decoding scheme using iterative optimization. To overcome the problem of an AE, a variational autoencoder (VAE) focuses on the regularity of the latent space, which guarantees the quality and relevance of generated data. A VAE represents the latent space regularization by making its encoder return a distribution over the latent space unlike an AE which compresses data of a single point. Besides, a VAE adds a regularization term to a loss function. The loss function of a VAE consists of the sum of the reconstruction term and the regularization term. Especially, the regularization term is expressed as the Kullback-Leibler divergence between the returned distribution and a standard Gaussian to organize the latent space. The above scheme of a VAE makes its latent space possible to generate new data, resulting in mitigation of overfitting. Afterwards, a VAE develops into an LSTM-VAE that transforms both an encoder and a decoder networks into LSTM networks to detect long term anomalies. LSTM networks are recurrent neural networks, which can model time series data and process entire sequence of data. An LSTM-VAE that has the characteristics of both a VAE and an LSTM encodes the data from each time series sequence as distribution over a latent space, samples a point from that distribution in the latent space, and decodes the sampled point.

For explaining anomaly detection, the Shapley value has recently been employed to provide interpretability for predictions from neural network models. It numerically expresses the contribution of each input feature to the predictions made by the models. In a conditional game theory, the Shapley value evaluates each participant's contribution to the profits in a cooperative game. The principal of the Shapley value is calculating the contribution degree of a specific input feature by comparison between the prediction results obtained from all combinations related to the specific input feature. Interpreting what input features mainly work can increase transparency that gives the reasoning behind certain predictions of the models.

3. Threat model

We define our threat model to ensure the efficiency of our technique by incorporating the attack assumptions used in research related to ROP attacks (Bittau et al., 2014; Evans et al., 2015; Lu et al., 2015). We assume that target machines use widely available 32-bit or 64-bit processors. We assume that security mechanisms such as DEP or ASLR are applied to a standard operating system like Windows, and that all hardware elements on the target machines are dependable. For remote attackers, we assume their objective is to alter the normal control flow of target programs and execute their own code without causing the target programs to crash. Note that attackers can have the same target programs because these programs are accessible on the Internet. This implies that attackers can discover security bugs, vulnerabilities, and code gadgets by analyzing the target programs. We assume that there are security bugs and vulnerabilities in the target programs that the attackers can take advantage of. We assume that the attackers can not only bypass protection techniques through information leakage or the Java runtime environment (JRE) but also specially exploit memory vulnerabilities. This allows them to read from and write to an arbitrary memory address. For example, memory vulnerabilities include Use-After-Free, integer overflow, heap buffer overflow, double dereference, and stack buffer overflow. Given our threat model, we focus on preventing attacks that exploit vulnerabilities to bypass DEP or ASLR and overwrite return addresses to hijack control flow. Our scope excludes considerations of hardware-based attacks (Gruss et al., 2015) and non-control-data attacks (Chen et al., 2005).

4. Our method: DROPSYS

4.1. Overview

DROPSYS is based on monitoring the change of system information affected by abnormal control flow transfers that take place during ROP attacks. In fact, the execution of ROP code adversely affects instruction processing by processors and operations on processes, resulting to abnormal system information.

Fig. 1 illustrates the architecture of DROPSYS, which follows a four-step process:

1. **System Information Utilization** involves collecting relevant system information about ROP attack behavior and effects.
2. **Data Preprocessing** transforms the collected HPC data and system utilization data into meaningful formats.
3. **Feature Selection** utilizes probabilistic PCA to approximate the Shapley value for understanding system information contributions. It then selects top-contributing features for efficient model training.
4. **Anomaly Detection** employs semi-supervised learning to model an LSTM-VAE using only legitimate system information as training data. DROPSYS estimates ROP attacks based on the reconstruction error, which serves as an anomaly score.

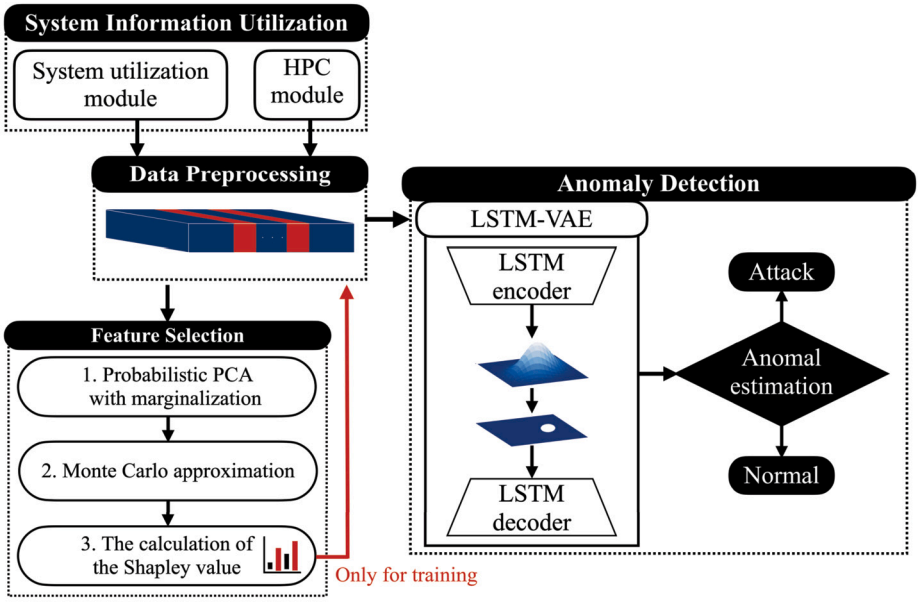

Fig. 1. Overview of the DROPSYS architecture.

Table 1
Monitored HPC data.

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

Table 2
Monitored system utilization data.

System utilization	Explanation
CPU user time	Process time spent in user mode
CPU system time	Process time spent in kernel mode
Read count	Number of read operations performed by a process
Write count	Number of write operations performed by a process
Read byte	Number of bytes read
Write byte	Number of bytes written
Other count	Number of I/O operations performed (other than read and write operations)
Other byte	Number of bytes transformed (other than read and write operations)
Page fault	Number of page faults
Peak Wset	Peak non-swapped physical memory used by a process
Wset	Non-swapped physical memory used by a process
Peak paged pool	Peak paged pool usage by a process
Paged pool	The current paged pool usage by a process
Peak non-paged pool	Peak non-paged pool usage by a process
Non-paged pool	Current non-paged pool usage by a process
Page file	Total amount of virtual memory used by a process
Peak page file	Peak total amount of virtual memory used by a process
USS	Memory size (unique to a process)
Memory map number	Number of memory maps per process
Memory map amount	Total amount of memory maps per process
Memory percent	Process memory utilization as a percentage
Voluntary context switch	Number of voluntary context switches by a process
Thread user time	Time spent by threads in user mode
Thread system time	Time spent by threads in kernel mode

4.2. System information utilization

DROPSYS not only monitors HPC data (Table 1) which is related to the behaviors of an ROP attack but also system utilization data (Table 2), enabling DROPSYS to analyze the effects of the behaviors. To utilize system information, DROPSYS generates two modules for sampling HPC data and system utilization data, respectively. The first module takes samples of system utilization data using a Python library (Rodola, 2016), while the second module samples HPC data through the Intel Performance Counter Monitor (PCM) (Willhalm and Dementiev, 2022) and CS-PMI (Das et al., 2019). We used Intel PCM for running Windows 10 and CS-PMI for Windows 7 to determine the optimal technique for each operating system. Both techniques were utilized in event-based sampling mode to read HPC data in real-time. Since a sampling rate faster than 1 ms results in overhead that is more than double the runtime, exceeding the acceptable range (Szekeres et al., 2013), DROPSYS avoids rates faster than 1 ms. We set 1 ms as the baseline sampling rate and collect system information at slower rates. The examination of changes in detection performance and overhead based on sampling rates is presented in Section 5.

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 (Skadron et al., 1998). 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. In terms of branch misprediction, ROP attacks degrade the performance of branch prediction because the attacks change the actual return address into a manipulated return address, causing RAS entries not to match consecutively. Fig. 2 compares control flow transfers between a normal execution (top) and an ROP attack execution (bottom) at the instruction level. In the normal execution, when a subroutine ends with a return instruction, the instruction right after the call instruction of the caller functions is processed. On the other hand, ROP attacks transfer the control flow from each gadget with only a few instructions (Cheng et al., 2014) to the next one through the successive execution of illegal return instructions. When a gadget ends with a return instruction, the first instruction of the following gadget is processed. Since the execution behavior of ROP code incurs successive illegal return instructions that prevent a processor from predicting branch instructions, it leads to a decrease in instruction processing performance. DROPSYS utilizes

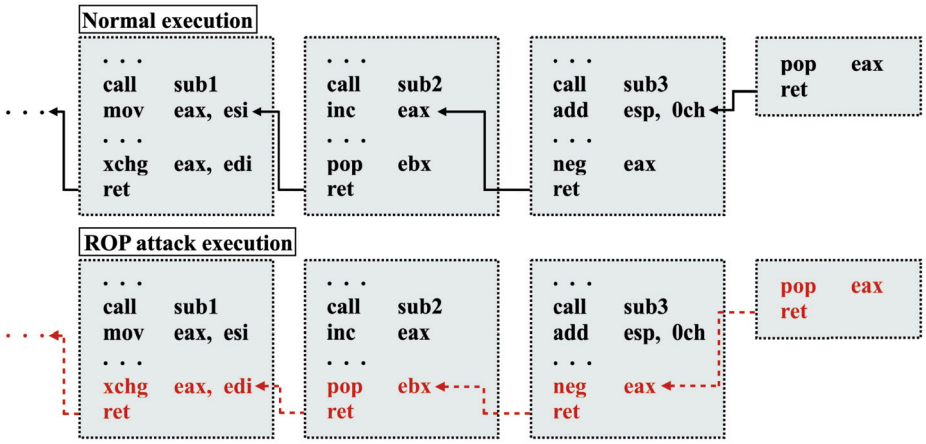

Fig. 2. Comparison of control flow transfers between a normal execution (top) and an ROP attack execution (bottom) at the instruction level.

system information related to branch mispredictions caused by ROP attacks.

Level 2 (L2) cache eviction, miss and last-level cache (LLC) miss. L2 cache eviction, as well as L2 cache miss and LLC miss, indicate that the desired data is not present 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 (Denning, 2005). 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. However, 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. DROPSYS catches ROP attacks using system information related to L2 cache eviction along with miss and LLC miss. Since L1 cache miss is frequent in normal executions due to the small capacity of L1 cache, we do not use Level 1 (L1) cache miss in our technique. For example, in the Ice Lake Client Microarchitecture, the capacity of L1 cache is 48 KB, which is smaller than L2 cache (512 KB) and the LLC (up to 2 MB per core).

Instructions per cycle (IPC). IPC refers to the average number of instructions executed for each clock cycle. ROP attacks decrease IPC and increase program execution time (Eyerman et al., 2006) due to branch misprediction, L2 cache eviction, misses, and LLC misses. These events 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 cache misses, the pipeline stops and brings about the penalty, which is an additional delay during which caches must receive the requested code and data. DROPSYS monitors ROP attacks using system information related to the IPC.

System utilization. We utilize system utilization data to catch the effect of ROP attacks on a process. 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. ROP attacks can lead to the atypical utilization of CPU, memory, and I/O operations in a process. This is because they require additional actions, such as executing gadgets, Windows API functions, and system calls, achieved through triggering abnormal control flow transfers. Fig. 3 illustrates snapshots (vertical bars) of the address space of a process in which ROP attacks occurred. Typically, ROP attacks based on a long sequence of gadget chain utilize not only memory corruption but also Windows API functions (Erlingsson, 2007) to invoke system calls instead of directly invoking system calls. This approach enables successful interaction with the victim operating system, granting full control (Pappas et al., 2013). Compared to normal execution, this attribute of ROP attacks abnormally affects both the memory usage (user space, system dll space, and kernel space) of a process and various I/O operations of the process. Note that the system utilization of a process is affected not only directly by ROP attacks but also by the aftereffects of the aforementioned three attack behaviors. Therefore, we leverage the causality between HPC data and system utilization 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: $X_{sysInfo} \in \mathbb{R}^{resamples \times features}$
$N1 \leftarrow samples$;
$N2 \leftarrow features$;
while $N1 \neq 0$ **do**
  $IPC1 \leftarrow \frac{X_{hpc}(N1,\ Instruction\ Retired)}{X_{hpc}(N1,\ UnHalted\ Core\ Cycles)}$;
  $IPC2 \leftarrow \frac{X_{hpc}(N1,\ Instruction\ Retired)}{X_{hpc}(N1,\ UnHalted\ Reference\ Cycles)}$;
  $X_{hpcA} \leftarrow merge\ IPC1,\ IPC2,\ and\ X_{hpc}(N1,\ others)$;
  $N1 \leftarrow N1 - 1$;
end
while $N2 \neq 0$ **do**
  $X_{utilZ} \leftarrow zscore(X_{util}(:, N2))$;
  $N2 \leftarrow N2 - 1$;
end
$X_{hpcR} \in \mathbb{R}^{resamples \times features} \leftarrow resample\ X_{hpcA}\ to\ 1ms$;
$X_{utilR} \in \mathbb{R}^{resamples \times features} \leftarrow resample\ X_{utilZ}\ to\ 1ms$;
$X_{hpcZ} \leftarrow interpolate\ X_{hpcR}\ with\ zero\ value$;
$X_{utilN} \leftarrow interpolate\ X_{utilR}\ with\ nearest\ value$;
$X_{sysInfo} \leftarrow merge\ X_{hpcZ},\ X_{utilN}$;

4.3. Data preprocessing

For 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 enhance detection performance, DROPSYS uses core clock cycles to calculate IPC1 and reference clock cycles to calculate IPC2. Next, DROPSYS applies a Z-Score standardization (Sugiyama, 2016) on system utilization data to monitor any change of system utilization of a process. Subsequently, DROPSYS upsamples the measured HPC data and system utilization data to the sampling rate's time interval, ensuring consistency in time intervals and points across 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

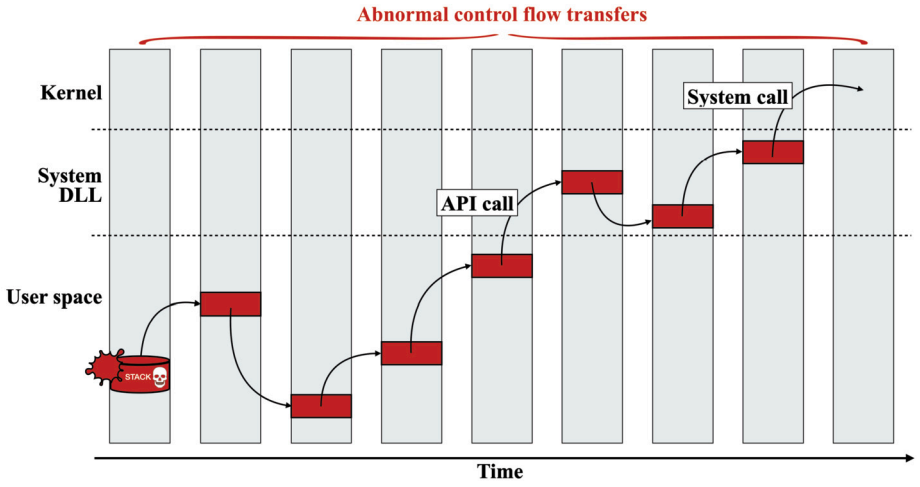

Fig. 3. Illustration of ROP code execution affecting system utilization by generating abnormal control flow transfers.

data. Finally, DROPSYS merges HPC data with system utilization data based on timestamps.

4.4. Feature selection

We utilize the Shapley value to perform feature selection. Our motivation is to enhance our overall model by leveraging insights gained from the Shapley value, which has the ability to measure the contribution of each feature (Arrieta et al., 2020). Specifically, we focus on the ability of the Shapley value that naturally accounts for interactions between features by capturing their impact within different feature subsets. We define the value function using the reconstruction error of probabilistic PCA, which is used for explaining anomaly detection (Takeishi, 2019). Furthermore, we use the marginalization technique which simplifies the value function to improve efficiency in calculation of the Shapley value. Then, we calculate the Shapley value using the value function with the Monte Carlo approximation (Strumbelj and Kononenko, 2010) that randomly selects some subsets of features. We measured contributions by the Shapley value for 30 system information to examine which data contributes to ROP attacks. We used 130 data sets from normal executions to define the reconstruction error of probabilistic PCA. We set the PCA dimensionality to 8. The 130 data sets from ROP attack executions were used to calculate the Shapley value. The number of Monte Carlo iterations was set to 1,000. The Shapley value was evaluated for each feature per sample of the data sets. We averaged contributions by each feature across all samples of the data sets. The contribution of 30 system information to ROP attacks is shown in Fig. 4, illustrating how system information influences ROP attack detection across four aspects covered earlier. Branch misprediction contributes 12%, and L2 cache eviction, miss, and LLC miss collectively make up 41%. IPC contributes 6%, while system utilization accounts for another 41%. Even though system utilization in the kernel layer significantly contributes, the 59:41 ratio between HPC data and system utilization highlights a distinct variation in HPC data at the hardware layer under normal and attack conditions. This finding suggests that our technique is designed to identify ROP attacks by placing emphasis on analyzing slightly more hardware events than system utilization. Utilizing the Shapley value for feature selection ensures that DROPSYS maintains high attack detection performance with minimal computational overhead. The examination of changes in detection performance and overhead based on the number of features is presented in Section 5.

4.5. Anomaly detection

We employ semi-supervised anomaly detection to enhance detection performance because this approach does not require prior information

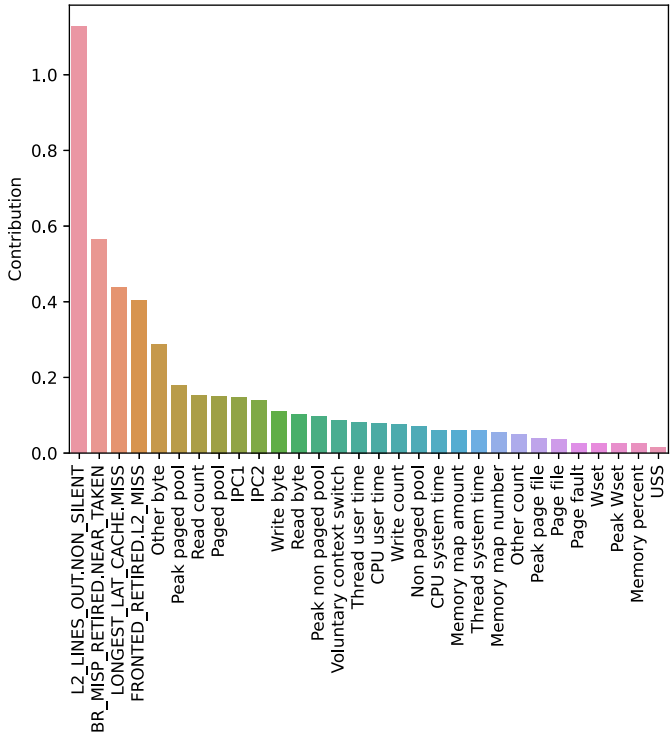

Fig. 4. The result of contribution degree of system information using the Shapley value.

about the attack during model training. In fact, only a few publicly available working ROP exploits exist, and making them function in a specific environment is challenging (Pappas et al., 2013; Tang et al., 2014; Cristalli et al., 2016; Das et al., 2018). Given the difficulty of collecting real-world ROP attack data, semi-supervised anomaly detection enables model training without the need for prior information about ROP attacks. In detail, semi-supervised anomaly detection involves labeling only the normal data (Alla and Adari, 2019). The model learns the characteristics of normal data, allowing it to identify anomalous data as deviations from the learned normal pattern. Therefore, semi-supervised anomaly detection is particularly useful when abnormal data is scarce compared to normal data or when there is a severe class imbalance between normal and abnormal data (Chandola et al., 2009).

We specifically utilize an LSTM-VAE to detect ROP attacks by analyzing the inherent characteristics of system information, placing particular emphasis on variability. Our approach is motivated by the fact that

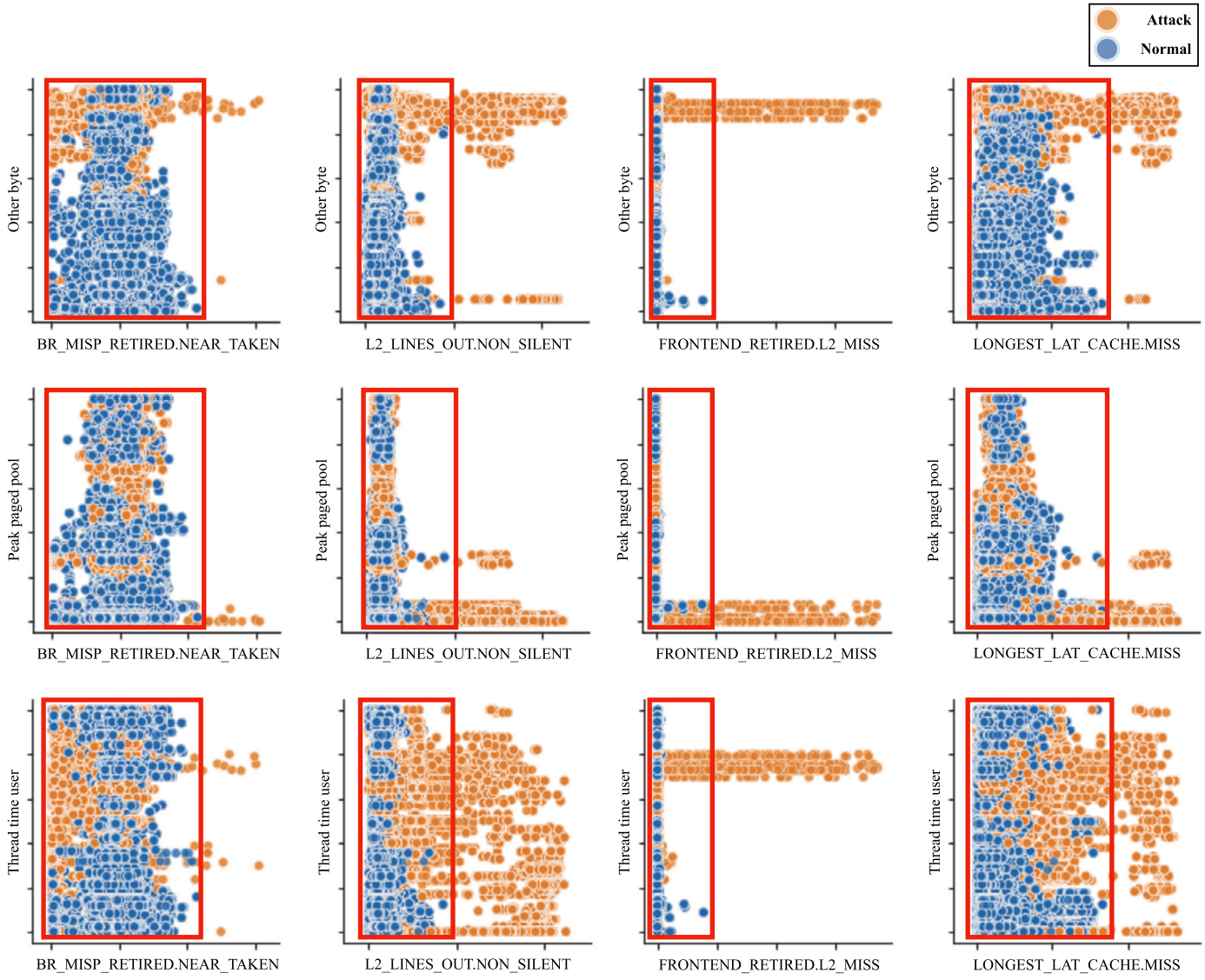

Fig. 5. The scatter plot depicting the relationship between HPC data and system utilization in each attack execution (orange) and normal execution (blue). (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

an LSTM-VAE captures the inherent distribution of system information across multiple dimensions and reconstructs system information using anticipated distribution (Park et al., 2018). Fig. 5 illustrates a scatter plot depicting the two distributions of system information for normal execution (blue) and attack execution (orange). The x-axis represents HPC data, and the y-axis represents system utilization data standardized using a Z-Score. Fig. 5 provides some visual insights into the variability of system information. In the horizontal and vertical spread of points on Fig. 5, widely scattered points with a random distribution indicate higher variability, while a narrow spread suggests lower variability. As depicted in the red rectangle in Fig. 5, distinguishing clearly between a normal execution and an attack execution based on the distributions of HPC data and system utilization is challenging with simple techniques.

Our LSTM-VAE model learns a latent representation of sequential data and generates new sequential data that is similar to the input data. In detail, our model encodes input data from each time-series sequence as a distribution over a latent space. The latent variables of the LSTM-VAE are stochastic variables. This provides extended expressive power and the ability to address differences in variability between normal data and anomaly data (An and Cho, 2015). From this distribution, a point is randomly sampled in the latent space, which is then decoded to generate a new time-series sequence. The reconstruction of LSTM-VAE depends on the variability, considering the variance parameter of the distribution function. This property allows for selective sensitivity to reconstruction based on variable variance (An and Cho, 2015). To assess detection performance considering variability of system information, we examine performance differences between LSTM-AE, which relies on a deterministic nature, and LSTM-VAE. The results are presented in Section 5.

The architecture of our LSTM-VAE model, illustrated in Fig. 6, utilizes a stateless LSTM model from the Keras deep learning library (Chollet et al., 2015). LSTM layers with *tanh* activation are employed. The model uses a batch size of 64, runs for 100 epochs, and utilizes *adam* optimizer with a learning rate of 1×10^{-4}. The encoder comprises two LSTM layers, succeeded by a mean and variance layer. The first layer has 50 units, and the second has 40 units, processing sequences based on the time window and number of features. Input data (X) is fed into the encoder, mapping it to a distribution within a latent space set at a dimensionality of 40. The encoder's mean (μ) and variance (σ) outputs are utilized to sample a latent vector (Z) from the latent space. The sampled latent vector enters the decoder, consisting of two LSTM layers (50 units for the first and 20 units for the second). The decoder generates sequences of the same length as the input, and the final dense layer ensures that the reconstructed output ($\hat{X}$) maintains the original

Table 3
Real-world ROP exploits from the Metasploit framework.

CVE	Application	# gadgets	Avg. Length	Vulnerability
2011-1996 (Ms11-081, 2011)	Internet explorer 8	19	2.05	Use-After-Free
2012-1535 (Adobe, 2012)	Adobe flash 11.3	18	2.17	Integer overflow
2012-1875 (Ms12-037, 2012a)	Internet explorer 8	45	1.44	Use-After-Free
2012-1876 (Ms12-037, 2012b)	Internet explorer 8	14	1.86	Heap overflow
2012-4792 (Ms13-008, 2012)	Internet explorer 8	72	2.03	Use-After-Free
2012-4969 (Ms12-063, 2012)	Internet explorer 9	19	2.05	Use-After-Free
2013-1347 (Ms13-038, 2013)	Internet explorer 8	45	2.04	Use-After-Free
2013-2551 (Ms13-037, 2013)	Internet explorer 8	3	1.67	Integer overflow
2013-3897 (Ms13-080, 2013)	Internet explorer 8	17	2.29	Use-After-Free
2014-0569 (Adobe, 2014)	Adobe flash 15.0	4	2	Integer overflow
2015-0359 (Adobe, 2015)	Adobe flash 17.0	4	2	Use-After-Free
2017-14016 (Advantech, 2017)	WebAccess 8.2	32	1.78	Stack buffer overflow
2018-11529 * (Vlc, 2018)	VLC 2.2	4	2.75	Use-After-Free

* Windows 10. Others: Windows 7.

Table 4
The change of detection performance and overhead in view of the number of features.

Test Program	# training/validation sets	# test sets (# records)		Threshold
		Normal	Attack	
WebAccess (CVE-2017-14016)	1×50	1×10 (29399)	1×10 (28044)	1.266
VLC (CVE-2018-11529)	1×50	1×10 (323034)	1×10 (314347)	0.297
Internet Explorer (11 Other CVEs)	11×50	11×10 (99762)	11×10 (99762)	0.549

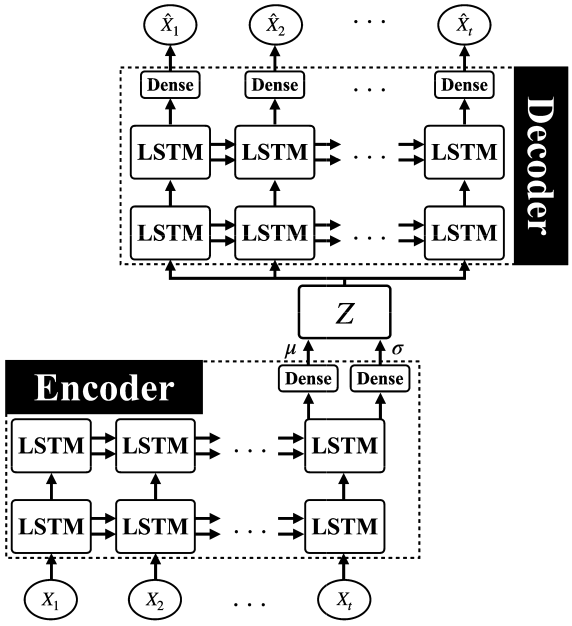

Fig. 6. The structure of our LSTM-VAE.

number of features. The hyperparameters were selected through trial and error, and these values have consistently produced satisfactory results in training the proposed model. The loss function consists of two components: the reconstruction loss and the KL divergence loss. The former measures the mean squared error between input and output, while the latter enforces a probabilistic structure in the latent space. The total loss is the sum of these components, guiding the model toward a balance between accurate reconstructions and a well-behaved latent space. We use the reconstruction error as an anomaly score. Calculated as the mean squared error between the original input data and the reconstructed data generated by the decoder, a high reconstruction error indicates that the input data was not well-reconstructed by the model, signifying the occurrence of an ROP attack.

To identify the optimal threshold for reconstruction error in anomaly estimation, we employ the precision-recall break-even point (Caruana and Niculescu-Mizil, 2004). This point represents the juncture where both precision and recall values are equal, ensuring a balanced trade-off between false positives and false negatives. Fig. 7 illustrates the precision-recall curve. The curve represents precision values (blue) and recalls values (orange) on the y-axis against thresholds on the x-axis. The precision-recall break-even point corresponds to the intersection of these precision and recalls curves. We start with the calculation of precision and recall values for each test set. Next, we identify the intersection points. Finally, we determine the average of all the identified intersection points and use this average as a threshold for assessing the model across all test sets. The thresholds we obtained are presented in Table 4.

5. Evaluation

In this section, we present the results of testing DROPSYS against real-world ROP attacks and measuring its runtime overhead. We set up both a target and an attack machine with an Intel Core i7-10700 2.9 Ghz (Ice Lake Client Microarchitecture). We used VMware (VMware, 2023) virtualization to run both machines. We implemented a prototype of DROPSYS on target machines running x86 Windows 7 Professional SP1 and x64 Windows 10 Pro. The selection of target machine platforms is a requirement for testing ROP attacks corresponding to each common vulnerability and exposure (CVE), rather than a limitation of implementation. On the attack machine running Kali Linux, we used the Metasploit framework to execute the ROP attacks in Table 3. Our evaluation addresses the following questions:

- **RQ 1. What advantages does DROPSYS gain from optimizing its hyperparameters?**
- **RQ 2. How accurately does DROPSYS capture abnormal system information?**

5.1. Data collection

We used 13 sets of evaluation scenarios based on CVEs listed in Table 3. Each set consists of a normal scenario and an ROP attack scenario, which were designed to collect system information. For instance, in the normal scenario, we observed Internet Explorer executing a general web search page like MSN, while in the attack scenario, we observed its ex-

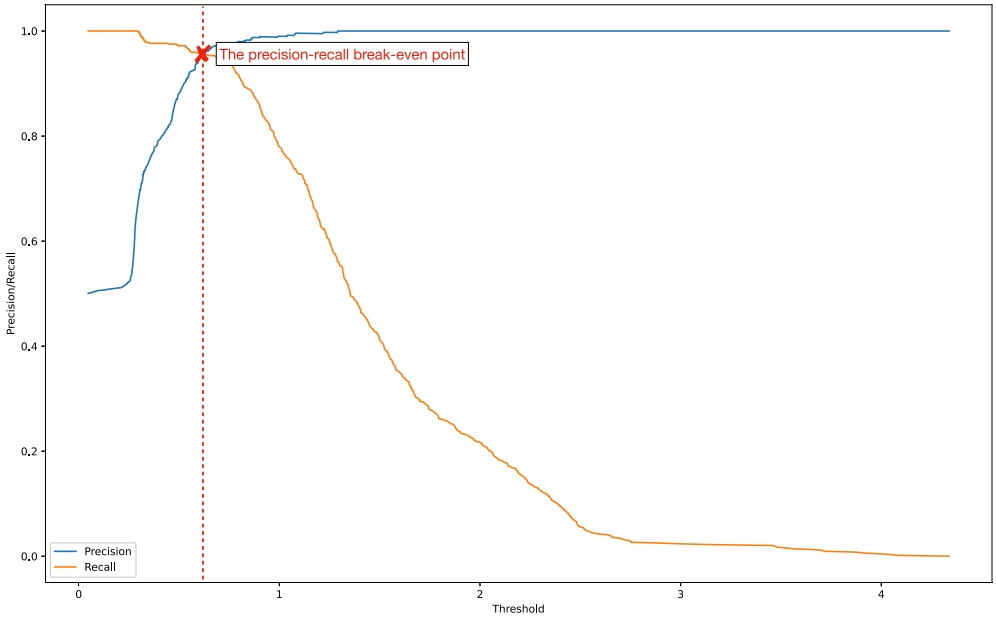

Fig. 7. An example of the precision-recall curve for CVE-2013-1347, illustrating precision-recall pairs for different thresholds.

ecution with either a malicious HTML file or a URL provided by the Metasploit framework. The evaluation scenarios vary slightly, particularly for the two CVEs: 2017-14016 and 2018-11529. In the normal scenario of CVE-2017-14016, the *webvrpcs* service of WebAccess runs normally, but in the attack scenario, the service is actively targeted by the attack machine. In the attack scenario of CVE-2018-11529, the malicious mkv file (959 MB) generated by the Metasploit framework is executed using VLC, whereas in the normal scenario, a regular mkv file of the same size is executed.

Each ROP attack scenario listed in Table 3 collectively exploits memory corruption vulnerabilities in various applications, utilizing unique gadgets with an average length of instructions. Note that we considered any combination of indirect control transfer instructions as a gadget (Checkoway et al., 2010). The ROP attack scenarios used in our experiment are divided into active exploits and passive exploits. As the names suggest, the former actively exploits a specific host, while the latter, such as browser-based exploits and file format exploits, waits for vulnerable hosts to connect or open a malicious file and then exploits them. The ROP attack scenarios proceed with the interaction between a target machine and an 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 shellcode, such as the meterpreter session connection, is transmitted and executed. Note that we focus on system information during the first stage because the main objective of ROP attacks is usually to enable execution of the second stage shellcode. We checked the start time of the second stage through Wireshark or FFDEC, and collected 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 910 data sets we obtained by monitoring 13 sets of scenarios for normal executions and ROP attack executions. We divided the 910 data sets into 650 (13×50) training/validation data sets and 260 (13×20) test data sets. All 650 training/validation data sets (13×50) were obtained from normal scenarios, with 20% of the data sets used for training also serving as validation data sets. The 260 test data sets consisted of 130 (13×10) data sets from normal scenarios and 130 (13×10) data sets from attack scenarios. We created three models for each test program, each using different datasets and thresholds, as shown in Table 4. 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.

5.2. Baseline techniques

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

- SIGDROP (Wang and Backer, 2016): 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 2.1 Ghz.

$$INST\_RETIRED.ANY \leq BR\_MISP\_RETIRED.RET \times X \tag{1}$$

- ROPSENTRY (Das et al., 2018): ROPSENTRY is a runtime defense against ROP attacks using HPC data. We collected the identical HPC data and selected the thresholds as described in Das et al. (2018). We set up both the target and attack machines with an Intel Xeon Silver 4208 2.1 Ghz.

$$\frac{BR\_MISP\_RETIRED.RET}{BR\_INST\_RETIRED.NEAR\_RETURN} \geq 0.9 \tag{2}$$

$$\frac{BR\_INST\_RETIRED.NEAR\_RETURN}{INST\_RETIRED.ANY} \geq 0.2 \tag{3}$$

$$\frac{FRONTEND\_RETIRED.ITLB\_MISS}{INST\_RETIRED.ANY} \geq 0.8 \tag{4}$$

$$\frac{LONGEST\_LAT\_CACHE.MISS}{INST\_RETIRED.ANY} \geq 2.0 \tag{5}$$

- LSTM-AE (Malhotra et al., 2016): To assess the effectiveness of DROPSYS in detecting attacks, we compared its performance with another LSTM-based anomaly detection technique, LSTM autoencoder (LSTM-AE), which relies on a deterministic nature. We configured the window size of LSTM-AE to 25 and calculated the reconstruction error, following the approach used by DROPSYS, for the detection of ROP attacks.

5.3. Detection performance

RQ 1. What advantages does DROPSYS gain from optimizing its hyperparameters? To achieve the best possible performance of LSTM-VAE, it is necessary to determine its appropriate hyperparameters. We

Table 5
The change of detection performance and overhead in view of the number of features.

Detection Performance	Features (K)				
	10	15	20	25	30
Accuracy	0.847	0.881	0.908	0.886	0.874
False Positive Rate	0.104	0.103	0.051	0.083	0.092
F1 Score	0.822	0.873	0.897	0.876	0.864
AUC-ROC	0.922	0.956	0.972	0.952	0.952
Model size (kb)	836	871	910	949	994
Total parameters	50790	53440	56340	59490	62890

Table 6
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	0.911	0.923	0.914	0.908
False Positive Rate	0.067	0.048	0.038	0.051
F1 Score	0.905	0.916	0.905	0.897
AUC-ROC	0.964	0.973	0.971	0.972

consider two key hyperparameters that impact the performance: the number of features, the time window size, and the sampling rate of the data. We evaluated these hyperparameters to identify the optimal values and improve the performance of the model.

The number of features refers to the dimensions in the input data. We tested to determine enough features that not only capture important information, but also do not make the model overly complex. Table 5 shows the average attack detection performance and overhead according to the number of features (K is the number of features selected from the top contribution based on the Shapley value in Fig. 4). We set DROPSYS with a time window size as 50 and evaluated the average detection performance. It is observed that DROPSYS obtains the best detection performance when K is set to 20 (i.e., 20 features). When 20 features are selected, DROPSYS outperforms other feature numbers in all metrics: average accuracy (0.908), false positive rate (0.051), F1 score (0.897), and AUC-ROC (0.972). However, if the number of selected features is greater than 20 or less than 20, the performance of DROPSYS deteriorates. From an overhead perspective, it is observed that as the dimensions of the input data increase, both the number of model parameters and the size of the model also increase, which has direct impact on training time, inference time, and memory requirements. The highest detection performance model has a size of 910 KB and a total of 56340 parameters. Overall, the results suggest that feature selection based on the Shapley value can reduce model complexity and improve detection performance. To maintain a reasonable performance overhead while ensuring high detection performance, we selected the top 20 system information based on the Shapley value as the input data.

The time window size refers to the number of time steps that are used as the input data. Increasing the time window size can capture long-term patterns in the data, but it also makes the model more complex and requires more training data. On the other hand, decreasing the time window size can capture local patterns, but may not properly capture long-term patterns in the data. Table 6 summarizes the results for DROPSYS from time window size 15 to 50 (T is the size of time window). We evaluated the average detection performance of DROPSYS when using the top 20 features of the Shapley values. We observed that DROPSYS has the best performance when T = 25. Table 6 shows that T slightly improves the detection performance of DROPSYS, which means that balancing the time windows is useful.

Additionally, we compared the detection performance of DROPSYS using two data sets to investigate the impact of system information on detecting ROP attacks. One data set has only 6 HPC data based on the Shapley value, while the other has 20 system information based on the

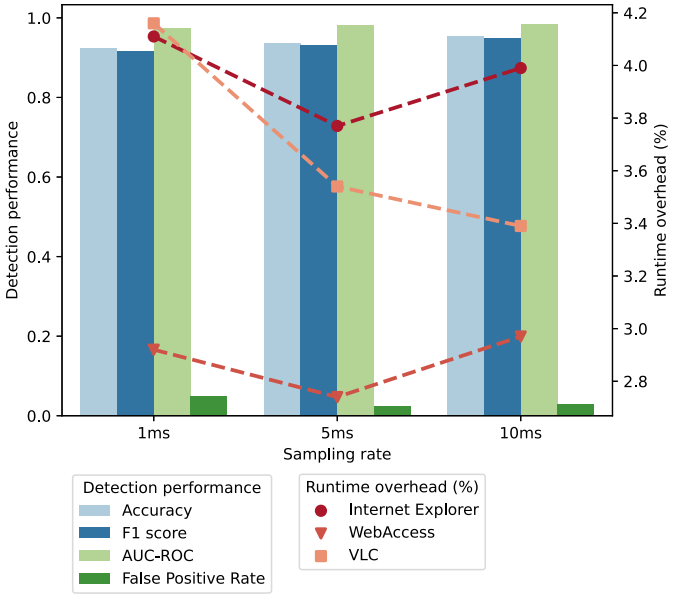

Fig. 8. The change of detection performance and runtime overhead in view of sampling rate (ms).

Shapley value. Note that the data set of 20 system information includes not only the 6 HPC data but also 14 extra system utilization data. Table 7 shows the evaluation results for accuracy, false positive rates, F1 scores, and AUC-ROC between two models. The LSTM-VAE with HPC data only shows average accuracy as 0.572 and the average F1 score as 0.561. However, the LSTM-VAE with 20 system information has an average accuracy of 0.923 and an average F1 score of 0.916. Notably, we observed a substantial difference in false positive rates of 0.411 and 0.048 for the models based on HPC data and system information, respectively. These results show that DROPSYS based on system information has good performance across all metrics.

Capturing the temporal dependencies of time-series data requires not only considering the number of features and the time window size, but also the sampling rate of the data. We tested the detection performance using three different sampling rates. Note that the size of the time window varies with the change in sampling rate as follows: 25 for 1 ms, 5 for 5 ms, and 2 for 10 ms. Fig. 8 shows that the detection performance improves with changes in sampling rate. These are the detection performance metrics for different sampling rates:

- 1 ms: Accuracy = 0.923, F1 score = 0.916, AUC-ROC = 0.973, False Positive Rate = 0.048
- 5 ms: Accuracy = 0.937, F1 score = 0.930, AUC-ROC = 0.982, False Positive Rate = 0.024
- 10 ms: Accuracy = 0.953, F1 score = 0.949, AUC-ROC = 0.984, False Positive Rate = 0.028

In addition, we assessed the runtime overhead of the three programs using three different sampling rates. As shown in Fig. 8, the runtime overhead improves as the sampling rate changes. For 1 ms, the average runtime overhead is 3.73% (Internet Explorer: 4.11%, WebAccss: 2.92%, VLC: 4.16%). For 5 ms, it is 3.35% (Internet Explorer: 3.77%, WebAccss: 2.74%, VLC: 3.54%), and for 10 ms, it is 3.45% (Internet Explorer: 3.99%, WebAccss: 2.97%, VLC: 3.39%). Szekeres et al. (2013) categorized overhead as unacceptable only if it doubled the runtime. Our runtime overhead is within this acceptable range, so it can be rated as favorable.

RQ 2. How accurately does DROPSYS capture abnormal system information? We evaluated the reconstruction function of DROPSYS to verify the accuracy of our model in detecting ROP attacks. First, We

Table 7
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	0.673	0.243	0.643	0.757	0.923	0.018	0.894	0.978
2012-1535	0.618	0.297	0.586	0.693	0.856	0.045	0.840	0.937
2012-1875	0.701	0.281	0.695	0.771	0.960	0.061	0.962	0.983
2012-1876	0.543	0.271	0.438	0.624	0.965	0.042	0.964	0.988
2012-4792	0.491	0.264	0.328	0.495	0.930	0.004	0.918	0.978
2012-4969	0.449	0.552	0.448	0.487	0.927	0.063	0.923	0.984
2013-1347	0.694	0.29	0.69	0.769	0.983	0.023	0.983	0.998
2013-2551	0.478	0.702	0.557	0.578	0.917	0.085	0.918	0.978
2013-3897	0.499	0.577	0.534	0.57	0.956	0.084	0.960	0.997
2014-0569	0.544	0.553	0.581	0.637	0.868	0.043	0.855	0.956
2015-0359	0.501	0.548	0.525	0.582	0.914	0.033	0.909	0.962
2017-14016	0.618	0.394	0.626	0.691	0.907	0.034	0.901	0.978
2018-11529	0.633	0.37	0.638	0.683	0.892	0.084	0.884	0.937
Average	**0.572**	**0.411**	**0.561**	**0.641**	**0.923**	**0.048**	**0.916**	**0.973**

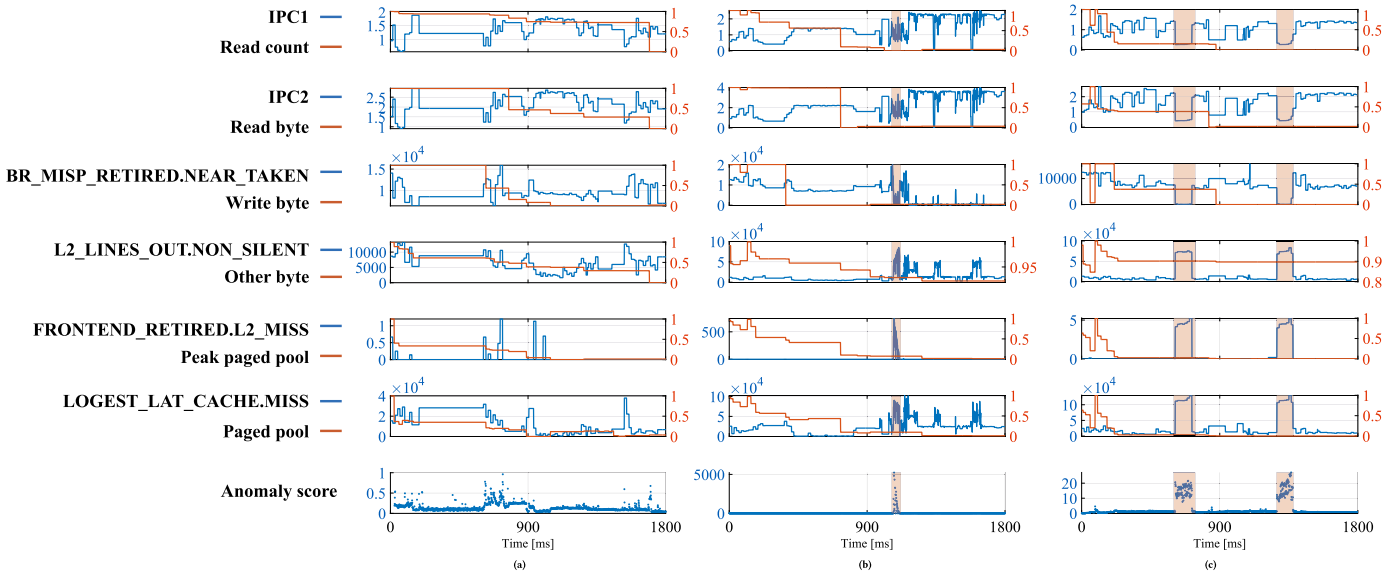

Fig. 9. Visualization of system information and the anomaly score over time. The upper six graphs show observations. The lower subgraphs show the anomaly scores. The pink vertical lines represent the time of ROP attack detection. (a) A normal execution. (b) An attack execution with CVE-2015-0359. (c) An attack execution with CVE-2012-1876.

monitored how system information and the anomaly score appeared during the duration of the attacks. Fig. 9 illustrates that the time points and time intervals between ROP attacks and the increase in anomaly score from our model are almost identical. 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. The upper 6 subgraphs in Fig. 9 show the observation of system information from both normal and attack executions in Internet Explorer. The observed increase in the anomaly score in the seventh subgraphs of Fig. 9(b) and 9(c) compared to Fig. 9(a) suggests that the abnormal series of system information is not typical in normal executions. While some system information may exhibit visually discernible patterns in normal and attack executions, most are challenging to differentiate with certainty. This is why we need DNNs to analyze system information.

Next, We compared DROPSYS with three other baseline techniques. Note that we did not need to train or validate SIGDROP and ROPSENTRY because these are rule-based techniques and not machine learning based. Fig. 10 shows the evaluation results for accuracy, false positive rates, and F1 scores. SIGDROP represents an average accuracy of 0.575 across 13 sets of scenarios and an average F1 score of 0.612. In addition, ROPSENTRY is evaluated through experiments with an average accuracy of 0.765 and an average F1 score of 0.816. These results show

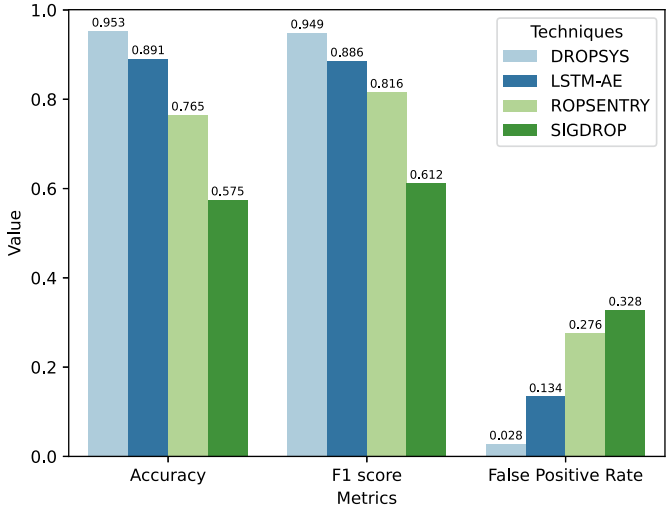

Fig. 10. Comparison of DROPSYS and three baseline techniques with three metrics.

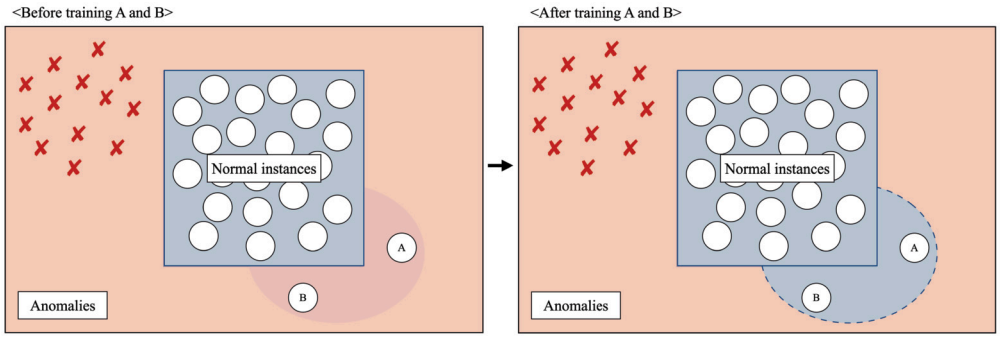

Fig. 11. The way anomaly detection learns from data.

that the accuracy of 0.953 and F1 score of 0.949 of DROPSYS (i.e., detection performance) are superior to these existing techniques. In addition, the average false positive rates of SIGDROP and ROPSENTRY are 0.328 and 0.276, which are both much higher than the 0.028 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 conclusion, we addressed the research problems we set by utilizing LSTM-VAE based on system information. DROPSYS not only achieves high detection performance but also incurs no additional costs when applying it to a system. Especially, compared to existing techniques, DROPSYS exhibits high detection performance. On the other hand, a disadvantage of DROPSYS, in comparison to existing techniques, is its requirement for extensive data during training, posing challenges due to the large amounts needed. Furthermore, understanding the internal workings of deep learning models can be challenging.

6. Discussion and future work

While DROPSYS significantly enhances detection performance against ROP attacks, there are certain aspects that could be considered limitations. First of all, we want to clarify that false positives may occur in DROPSYS due to anomalies not necessarily related to ROP attacks. However, DROPSYS can mitigate this issue in two ways. First, our model employs anomaly detection based on semi-supervised learning. The model's determination of what is considered normal is influenced by the definition of the training data (normal instances). Depending on the learned training data, the normal area can either narrow or widen. As illustrated in Fig. 11, anomaly detection determines the normal area from the training data to detect attack data (represented by red Xs). Before training on normal data A and B, any data outside the normal area is considered anomalous. If we identify A and B as normal and train on them, the boundary between normal and anomaly is defined more appropriately. Likewise, if DROPSYS learns normal conditions that might be overlooked, such as A and B, it can reduce false positives. Secondly, the system information mechanism used by DROPSYS can be collected within the scope we defined. We can obtain system information clearly from the operation of a target application, which ensures that the system information is minimally affected by other conditions in the system. This means that DROPSYS can focus on the target program. The ability to control the scope of system information's source allows us to monitor both hardware events and system utilization only for a target program. Therefore, even if our model produces false positives, it does not affect our overall detection. In future work, to address the challenge of the ambiguous boundary between normal and outlier objects, we will define what is considered normal and evaluate our model by incorporating formal specifications based on formal methods.

Additionally, we would like to discuss our experiments conducted with a limited number of real-world ROP exploits. While the results are promising, this issue may be considered a limitation in generalizing our technique. Other studies have also observed similar situations regarding this issue (Pappas et al., 2013; Tang et al., 2014; Cristalli et al., 2016; Das et al., 2018). In reality, finding real-world ROP exploits that function in a specific environment is challenging, especially when considering that creating such exploits often requires legacy programs. These programs are not only no longer distributed but also fail to execute properly due to limited compatibility. Nevertheless, DROPSYS can be considered to generalize its key characteristics for detecting ROP attacks. DROPSYS is based on a key principle in semi-supervised anomaly detection, which enables it to generalize its approach by learning the characteristics of normal data. This allows it to identify anomalous data as deviations from the learned normal pattern. (Alla and Adari, 2019). In our experiment, we expanded upon previous research that covers the entire spectrum of ROP attacks (Das et al., 2018). Subsequently, we evaluated the effectiveness of our approach by testing it against real-world ROP exploits, revealing very high accuracy in detecting them. Furthermore, while empirical values for ROP attacks may change with the introduction of new exploits, DROPSYS can seamlessly adapt and integrate new patterns by monitoring various HPC data and system utilization data.

Finally, we would like to discuss our feature selection experiment. Our approach is based on the Shapley value, aiming to enhance the predictive performance of our model and generate a more computationally efficient model. However, we acknowledge a limitation in our experiment, which lies in not testing several existing methods for determining the importance or contribution of features. In future work, we plan to conduct experiments by utilizing two key categories of feature selection techniques (Chandrashekar and Sahin, 2014): Wrapper methods (model-driven evaluations) and Filter methods (general model-independent evaluations). Furthermore, we will focus on analyzing system information based on feature relevance explanations from eXplainable AI (XAI) (Gunning, 2017). This approach aims not only to improve learning performance but also to enable humans to understand model predictions. For instance, to explain how our model predicts ROP attacks based on system information, we can use model-agnostic techniques that are specifically designed to be incorporated into any model (Arrieta et al., 2020).

7. Related works

In this section, we discuss ROP defenses in general, with a particular emphasis on hardware-assisted and HPC-based defense techniques that are closely related to our work. Control-Flow Integrity (CFI) (Burow et al., 2017) is a general defense that protects forward edges such as function pointers or virtual table pointers. It enforces policies on indirect control transfer instructions to ensure that the program's execution flow follows valid paths in its Control-Flow Graph (CFG). However, CFI has a weakness: static CFG construction is imprecise, resulting in numerous possible targets with the same label. Although the CFI enforcement component validates labels at runtime, this ambiguity may still enable attackers to launch attacks using allowed function-level gadgets (Koruyeh et al., 2020). Code Pointer Integrity (CPI) (Kuznetzov et al., 2018) divides a stack into a safe stack and an unsafe stack. CPI protects stack variables that could potentially be unsafe by moving them to the safe

stack, preventing them from being tampered with. However, CPI has limited compatibility with unprotected code. Automated software diversity defenses (Larsen et al., 2014) utilize binary rewriting techniques at post-compile or run-time. These defenses can involve stack canaries or ASLR schemes. But attackers can use various techniques, such as information leakage and brute force attacks, to dynamically get the location of the code and then bypass automated software diversity defenses. A shadow stack (Burow et al., 2019) protects backward edge, i.e., return addresses. This technique uses a secondary stack, known as a shadow stack, to save return addresses in an isolated memory region and to verify the saved return addresses. But attackers can exploit memory vulnerabilities to infer the location of the shadow stack.

Hardware-assisted defense techniques. Hardware-assisted defense techniques included new approaches to mitigate the limitations of existing software-based defense techniques, which required source code and symbolic debugging information of the target program to defend against control-flow attacks (Göktas et al., 2014). For example, CFIMon (Xia et al., 2012) utilizes the branch-tracing store mechanism provided by the processor's PMU along with static analysis. CFIMon detects the integrity violation of the program's control flow in accordance with several proposed rules. After CFIMon, kBouncer (Pappas et al., 2013) involves a technique that can detect ROP attacks using the last branch recording (LBR) mechanism. Especially, kBouncer checks the LBR register when a system API function is called and detects ROP attacks according to the proposed policy for the integrity of indirect control transfers. However, these API hooking defense techniques can easily be bypassed by elaborate ROP attack payloads, such as hook hopping (Li and Crouse, 2014).

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 (Zhou et al., 2014), SIGDROP (Wang and Backer, 2016), and ROPSENTRY (Das et al., 2018). 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, described as instructions retired, mispredicted return instructions, and near return instructions. Similarly, ROPSENTRY utilizes hardware performance events, such as mispredicted return instructions, near return instructions, instructions retired, misses in all ITLB levels, and longest latency cache misses 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 (Das et al., 2019). In addition, Tang et al. (2014) 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. (2015) 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 (Herath and Fogh, 2015). For example, HPC data is used in other security fields like program integrity (Malone et al., 2011), malware detection (Demme et al., 2013), rootkits detection (Singh et al., 2017), and cache side-channel attack defense (Zhang and Reiter, 2013).

8. Conclusion

In this paper, we presented a new technique called DROPSYS, which is a comprehensive ROP attack detection technique that has proven effective detection performance without additional cost or compatibility issues. By leveraging system information, DROPSYS mitigates the non-determinism of HPCs and the limited number of available HPCs. Additionally, DROPSYS utilizes LSTM-VAE 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.

CRediT authorship contribution statement

Seon Kwon Kim: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Project administration, Resources, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Hongjoo Jin:** Conceptualization, Formal analysis, Investigation, Methodology, Validation. **Kyungho Joo:** Conceptualization, Formal analysis, Investigation, Methodology, Validation. **Jiwon Lee:** Conceptualization, Formal analysis, Investigation, Methodology, Validation. **Dong Hoon Lee:** Conceptualization, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Supervision, Validation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The authors are unable or have chosen not to specify which data has been used.

Acknowledgements

This work was supported as part of Military Crypto Research Center (UD210027XD) funded by Defense Acquisition Program Administration (DAPA) and Agency for Defense Development (ADD).

References

Ms11-081 Microsoft Internet explorer option element use-after-free. URL. https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/windows/browser/ms11_081_option.rb.

Adobe flash player 11.3 kern table parsing integer overflow. URL. https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/windows/browser/adobe_flash_otf_font.rb.

Ms12-037 Microsoft Internet explorer same id property deleted object handling memory corruption. URL. https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/windows/browser/ms12_037_same_id.rb.

Ms12-037 Microsoft Internet explorer fixed table col span heap overflow. URL. https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/windows/browser/ms12_037_ie_colspan.rb.

Ms13-008 Microsoft Internet explorer cbutton object use-after-free vulnerability. URL. https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/windows/browser/ie_cbutton_uaf.rb.

Ms12-063 Microsoft Internet explorer execcommand use-after-free vulnerability. URL. https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/windows/browser/ie_execcommand_uaf.rb.

Ms13-038 Microsoft Internet explorer cgenericelement object use-after-free vulnerability. URL. https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/windows/browser/ie_cgenericelement_uaf.rb.

Ms13-037 Microsoft Internet explorer coalinedashstylearray integer overflow. URL. https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/windows/browser/ms13_037_svg_dashstyle.rb.

Ms13-080 Microsoft Internet explorer cdisplaypointer use-after-free. URL. https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/windows/browser/ms13_080_cdisplaypointer.rb.

Adobe flash player casi32 integer overflow. URL. https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/windows/browser/adobe_flash_casi32_int_overflow.rb.

Adobe flash player domainmemory bytearray use after free. URL. https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/windows/browser/adobe_flash_domain_memory_uaf.rb.

Advantech webaccess webvrpcs service opcode 80061 stack buffer overflow. URL. https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/windows/scada/advantech_webaccess_webvrpcs_bof.rb.

Vlc media player mkv use after free. URL. https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/windows/fileformat/vlc_mkv.rb.

Alla, S., Adari, S.K., 2019. Beginning Anomaly Detection Using Python-Based Deep Learning. Springer.

An, J., Cho, S., 2015. Variational autoencoder based anomaly detection using reconstruction probability. In: Special Lecture on IE 2 (1), 1–18.

Andress, J., 2014. The basics of information security: understanding the fundamentals of InfoSec in theory and practice. In: Syngress.

Arrieta, A.B., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., García, S., Gil-López, S., Molina, D., Benjamins, R., et al., 2020. Explainable artificial intelligence (xai): concepts, taxonomies, opportunities and challenges toward responsible ai. Inf. Fusion 58, 82–115.

Avritzer, A., Tanikella, R., James, K., Cole, R.G., Weyuker, E., 2010. Monitoring for security intrusion using performance signatures. In: Proceedings of the First Joint WOSP/SIPEW International Conference on Performance Engineering, pp. 93–104.

Bittau, A., Belay, A., Mashtizadeh, A., Mazières, D., Boneh, D., 2014. Hacking blind. In: 2014 IEEE Symposium on Security and Privacy. IEEE, pp. 227–242.

Burow, N., Carr, S.A., Nash, J., Larsen, P., Franz, M., Brunthaler, S., Payer, M., 2017. Control-flow integrity: precision, security, and performance. ACM Comput. Surv. 50 (1), 1–33.

Burow, N., Zhang, X., Payer Sok, M., 2019. Shining light on shadow stacks. In: 2019 IEEE Symposium on Security and Privacy (SP). IEEE, pp. 985–999.

Caruana, R., Niculescu-Mizil, A., 2004. Data mining in metric space: an empirical analysis of supervised learning performance criteria. In: Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 69–78.

Chandola, V., Banerjee, A., Kumar, V., 2009. Anomaly detection: a survey. ACM Comput. Surv. 41 (3), 1–58.

Chandrashekar, G., Sahin, F., 2014. A survey on feature selection methods. Comput. Electr. Eng. 40 (1), 16–28.

Checkoway, S., Davi, L., Dmitrienko, A., Sadeghi, A.-R., Shacham, H., Winandy, M., 2010. Return-oriented programming without returns. In: Proceedings of the 17th ACM Conference on Computer and Communications Security, pp. 559–572.

Chen, S., Xu, J., Sezer, E.C., Gauriar, P., Iyer, R.K., 2005. Non-control-data attacks are realistic threats. In: USENIX Security Symposium, vol. 5, p. 146.

Cheng, Y., Zhou, Z., Miao, Y., Ding, X., Deng, R.H. Ropecker: a generic and practical approach for defending against rop attack.

Chollet, F., et al. Keras documentation, keras. io 33.

Cristalli, S., Pagnozzi, M., Graziano, M., Lanzi, A., Balzarotti, D., 2016. Micro-virtualization memory tracing to detect and prevent spraying attacks. In: 25th {USENIX} Security Symposium. In: {USENIX} Security, vol. 16, pp. 431–446.

Das, S., Chen, B., Chandramohan, M., Liu, Y., Zhang, W., 2018. Ropsentry: runtime defense against rop attacks using hardware performance counters. Comput. Secur. 73, 374–388.

Das, S., Werner, J., Antonakakis, M., Polychronakis, M., Monrose Sok, F., 2019. The challenges, pitfalls, and perils of using hardware performance counters for security. In: 2019 IEEE Symposium on Security and Privacy (SP). IEEE, pp. 20–38.

Davi, L., Hanreich, M., Paul, D., Sadeghi, A.-R., Koeberl, P., Sullivan, D., Arias, O., Jin Hafix, Y., 2015. Hardware-assisted flow integrity extension. In: 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC). IEEE, pp. 1–6.

De Clercq, R., Verbauwhede, I., 2017. A survey of hardware-based control flow integrity (cfi). arXiv preprint. arXiv:1706.07257.

Demme, J., Maycock, M., Schmitz, J., Tang, A., Waksman, A., Sethumadhavan, S., Stolfo, S., 2013. On the feasibility of online malware detection with performance counters. ACM SIGARCH Comput. Architect. News 41 (3), 559–570.

Denning, P.J., 2005. The locality principle. Commun. ACM 48 (7), 19–24.

Erlingsson, Ú., 2007. Low-level software security: attacks and defenses. In: Foundations of Security Analysis and Design IV: FOSAD 2006/2007 Tutorial Lectures. Springer, pp. 92–134.

Evans, I., Fingeret, S., Gonzalez, J., Otgonbaatar, U., Tang, T., Shrobe, H., Sidiroglou-Douskos, S., Rinard, M., Okhravi, H., 2015. Missing the point (er): on the effectiveness of code pointer integrity. In: 2015 IEEE Symposium on Security and Privacy. IEEE, pp. 781–796.

Eyerman, S., Smith, J.E., Eeckhout, L., 2006. Characterizing the branch misprediction penalty. In: 2006 IEEE International Symposium on Performance Analysis of Systems and Software. IEEE, pp. 48–58.

Göktas, E., Athanasopoulos, E., Bos, H., Portokalidis, G., 2014. Out of control: overcoming control-flow integrity. In: 2014 IEEE Symposium on Security and Privacy. IEEE, pp. 575–589.

Gruss, D., Spreitzer, R., Mangard, S., 2015. Cache template attacks: automating attacks on inclusive {Last-Level} caches. In: 24th USENIX Security Symposium. In: USENIX Security, vol. 15, pp. 897–912.

Guide, P. Intel® 64 and ia-32 architectures software developer's manual, vol. 3.

Gunning, D., 2017. Explainable artificial intelligence (xai), defense advanced research projects agency (DARPA). In: nd Web 2 (2), 1.

Herath, N., Fogh, A. These are not your grand daddys cpu performance counters–cpu hardware performance counters for security, Black Hat Briefings.

Koruyeh, E.M., Shirazi, S.H.A., Khasawneh, K.N., Song, C., Abu-Ghazaleh, N., 2020. Speccfi: mitigating spectre attacks using cfi informed speculation. In: 2020 IEEE Symposium on Security and Privacy (SP). IEEE, pp. 39–53.

Kumar, S., Moolchandani, D., Sarangi, S.R., 2022. Hardware-assisted mechanisms to enforce control flow integrity: a comprehensive survey. J. Syst. Archit., 102644.

Kuznetzov, V., Szekeres, L., Payer, M., Candea, G., Sekar, R., Song, D., 2018. Code-pointer integrity. In: The Continuing Arms Race: Code-Reuse Attacks and Defenses, pp. 81–116.

Larsen, P., Homescu, A., Brunthaler, S., Franz Sok, M., 2014. Automated software diversity. In: 2014 IEEE Symposium on Security and Privacy. IEEE, pp. 276–291.

Li, X., Crouse, M. Transparent rop detection using cpu performance counters. THREADS.

Liao, H.-J., Lin, C.-H.R., Lin, Y.-C., Tung, K.-Y., 2013. Intrusion detection system: a comprehensive review. J. Netw. Comput. Appl. 36 (1), 16–24.

Lu, K., Song, C., Lee, B., Chung, S.P., Kim, T., Lee, W., 2015. Aslr-guard: stopping address space leakage for code reuse attacks. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 280–291.

Lundberg, S.M., Lee, S.-I., 2017. A unified approach to interpreting model predictions. Adv. Neural Inf. Process. Syst. 30.

Malhotra, P., Ramakrishnan, A., Anand, G., Vig, L., Agarwal, P., Shroff, G., 2016. Lstm-based encoder-decoder for multi-sensor anomaly detection. arXiv preprint. arXiv:1607.00148.

Malone, C., Zahran, M., Karri, R., 2011. Are hardware performance counters a cost effective way for integrity checking of programs. In: Proceedings of the Sixth ACM Workshop on Scalable Trusted Computing, pp. 71–76.

Metasploit, 2022. Metasploit — penetration testing software, pen testing security — metasploit. https://www.metasploit.com/. (Accessed 16 December 2022).

Miller, M., Burrell, T., Howard, M., 2011. Mitigating software vulnerabilities.

Pappas, V., Polychronakis, M., Keromytis, A.D., 2013. Transparent {ROP} exploit mitigation using indirect branch tracing. In: 22nd USENIX Security Symposium. In: USENIX Security, vol. 13, pp. 447–462.

Park, D., Hoshi, Y., Kemp, C.C., 2018. A multimodal anomaly detector for robot-assisted feeding using an lstm-based variational autoencoder. IEEE Robot. Autom. Lett. 3 (3), 1544–1551.

Pfaff, D., Hack, S., Hammer, C., 2015. Learning how to prevent return-oriented programming efficiently. In: International Symposium on Engineering Secure Software and Systems. Springer, pp. 68–85.

Preeth, E., Mulerickal, F.J.P., Paul, B., Sastri, Y., 2015. Evaluation of docker containers based on hardware utilization. In: 2015 International Conference on Control Communication & Computing India (ICCC). IEEE, pp. 697–700.

Rains, T., Miller, M., Weston, D., 2015. Exploitation trends: from potential risk to actual risk. In: RSA Conference.

Rodola, G. Psutil package: a cross-platform library for retrieving information on running processes and system utilization. Google Scholar.

Roemer, R., Buchanan, E., Shacham, H., Savage, S., 2012. Return-oriented programming: systems, languages, and applications. ACM Trans. Inf. Syst. Secur. 15 (1), 1–34.

Roglia, G.F., Martignoni, L., Paleari, R., Bruschi, D., 2009. Surgically returning to randomized lib (c). In: 2009 Annual Computer Security Applications Conference. IEEE, pp. 60–69.

Shacham, H., 2007. The geometry of innocent flesh on the bone: return-into-libc without function calls (on the x86). In: Proceedings of the 14th ACM Conference on Computer and Communications Security, pp. 552–561.

Shacham, H., Page, M., Pfaff, B., Goh, E.-J., Modadugu, N., Boneh, D., 2004. On the effectiveness of address-space randomization. In: Proceedings of the 11th ACM Conference on Computer and Communications Security, pp. 298–307.

Shanbhogue, V., Gupta, D., Sahita, R., 2019. Security analysis of processor instruction set architecture for enforcing control-flow integrity. In: Proceedings of the 8th International Workshop on Hardware and Architectural Support for Security and Privacy, pp. 1–11.

Singh, B., Evtyushkin, D., Elwell, J., Riley, R., Cervesato, I., 2017. 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, pp. 483–493.

Skadron, K., Ahuja, P.S., Martonosi, M., Clark, D.W., 1998. Improving prediction for procedure returns with return-address-stack repair mechanisms. In: Proceedings. 31st Annual ACM/IEEE International Symposium on Microarchitecture. IEEE, pp. 259–271.

Strumbelj, E., Kononenko, I., 2010. An efficient explanation of individual classifications using game theory. J. Mach. Learn. Res. 11, 1–18.

Sugiyama, M., 2016. Section 2.5 – Transformation of Random Variables. MK Morgan Kaufmann, pp. 22–23.

Szekeres, L., Payer, M., Wei, T., Song, D., 2013. Sok: eternal war in memory. In: 2013 IEEE Symposium on Security and Privacy. IEEE, pp. 48–62.

Takeishi, N., 2019. Shapley values of reconstruction errors of pca for explaining anomaly detection. In: 2019 International Conference on Data Mining Workshops (icdmw). IEEE, pp. 793–798.

Tang, A., Sethumadhavan, S., Stolfo, S.J., 2014. Unsupervised anomaly-based malware detection using hardware features. In: International Workshop on Recent Advances in Intrusion Detection. Springer, pp. 109–129.

van de Ven, A. New security enhancements in red hat enterprise Linux v. 3, update 3. Red Hat. Raleigh, North Carolina, USA.
VMware, 2023. Vmware — introducing vmware cross-cloud services — vmware. https://www.vmware.com/. (Accessed 17 January 2023).
Vreugdenhil, P., 2010. Pwn2own 2010 windows 7 Internet explorer 8 exploit.
Wang, X., Backer, J., 2016. Sigdrop: signature-based rop detection using hardware performance counters. arXiv preprint. arXiv:1609.02667.
Wang, Y., Wu, J., Yue, T., Ning, Z., Zhang, F., 2022. Rettag: hardware-assisted return address integrity on risc-v. In: Proceedings of the 15th European Workshop on Systems Security, pp. 50–56.
Weaver, V.M., Terpstra, D., Moore, S., 2013. Non-determinism and overcount on modern hardware performance counter implementations. In: 2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). IEEE, pp. 215–224.
Willhalm, T., Dementiev, R., 2022. Intel® performance counter monitor - a better way to measure cpu utilization. https://www.intel.com/content/www/us/en/developer/articles/technical/performance-counter-monitor.html.
Xia, Y., Liu, Y., Chen, H., Zang, B., 2012. Cfimon: detecting violation of control flow integrity using performance counters. In: IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012). IEEE, pp. 1–12.
Yuan, L., Xing, W., Chen, H., Zang, B., 2011. Security breaches as pmu deviation: detecting and identifying security attacks using performance counters. In: Proceedings of the Second Asia-Pacific Workshop on Systems, pp. 1–5.
Zhang, Y., Reiter, M.K., 2013. 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, pp. 827–838.
Zhou, H., Wu, X., Shi, W., Yuan, J., Liang, B., 2014. Hdrop: detecting rop attacks using performance monitoring counters. In: International Conference on Information Security Practice and Experience. Springer, pp. 172–186.

Seon Kwon Kim received the B.S. degree in computer science and electronics engineering from Handong Global University, Pohang, South Korea, in 2021 and the M.S. degrees in cyber security from Korea University, Seoul, Korea, in 2023. His research interests include software protection and IoT/CPS security.

Hongjoo Jin received the B.S. degree in computer engineering from Hongik University, Seoul, South Korea, in 2017, and the M.S. degree in information security from Korea University, Seoul, in 2019, where he is currently pursuing the Ph.D. degree in information security with the Graduate School of Information Security. His research interests include virtual memory protection, embedded device security, and static program analysis.

Kyungho Joo received the B.S. degree from the College of Information and Communication, Korea University, Seoul, South Korea, in 2016. He also received the M.S. and Ph.D. degree in information security from Korea University in 2024 and 2018, resepctively. From 2024, he is an Assistant Professor at Soongsil University, Republic of Korea. His research interests include Wireless security, Automotive security.

Jiwon Lee received the B.S. degree in information security cryptography and mathematics from Kookmin University, Seoul, South Korea, in 2021, where he is currently pursuing a combined M.S. and Ph.D. degree in information security with the Graduate School of Information Security. His research interests include software protection, code obfuscation, and reverse engineering.

Dong Hoon Lee received the B.S. degree from Korea University, Seoul, South Korea, in 1985 and the M.S. and Ph.D. degrees in computer science from the University of Oklahoma, Norman, OK, USA, in 1988 and 1992, respectively. Since 1993, he has been with the Faculty of Computer Science and Information Security, Korea University. He is currently a Professor and the Director with the Graduate School of Information Security, Korea University. His research interests include cryptographic protocol, applied cryptography, functional encryption, software protection, mobile security, vehicle security, and ubiquitous sensor network security.