Salus: Stackelberg Games for Malware Detection with Microarchitectural Events

by

Elly Khodaei

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2024

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Microarchitectural events have been the subject of previous investigations for malware detection. While some studies assert the effectiveness of utilizing hardware events in detecting malware, others contend that they may not be beneficial for this purpose. We argue and empirically show that the efficacy of using hardware events for malware detection relies on accurately selecting hardware events during detector training. Through rigorous analysis, we demonstrate that the conventional approach of selecting a single subset of hardware events for training a malware detection model is insufficient for creating a robust system capable of effectively handling all types of malware, even when using a ensemble of powerful classifiers. Accordingly, we propose the use of multiple subsets of hardware events, each dedicated to training a distinct malware detection model. Since only a single subset of events can be monitored at any given time, we adopt a game-theoretic approach to determine the optimal strategy for selecting the subset of hardware events to be monitored. In addition to the theoretical analysis of our approach, we empirically demonstrate its effectiveness by comparing it to other baselines.

Acknowledgements

I am deeply grateful to Dr. Seyed Majid Zahedi, my supervisor, for his invaluable guidance and support throughout the duration of this project. His expertise played a pivotal role in the successful completion of this study.

I would also like to express my appreciation to Professor Tripunitara and Professor Pellizzoni for their time and effort in reviewing my thesis.

Finally, I would like to extend my sincere gratitude to my family and Matheus for their unwavering love and support. Their encouragement has been a steadfast source of strength throughout this journey, and I am deeply grateful for their presence in my life.

Dedication

This thesis is dedicated to my parents for their unconditional love and support.

Table of Contents

Author's Declaration ii

Abstract iii

Acknowledgements iv

Dedication v

List of Figures ix

List of Tables x

1 Introduction 1

2 Background 3

2.1 Hardware Performance Counters 3

2.2 HW Events & Malware Detection 4

3 Methodology 6

3.1 Malware and Benignware Samples 6

3.2 Execution Environment 6

3.3 Feature Selection 7

3.4 Machine Learning Model 7

4 Hardware Events Selection **9**
4.1 Clustering Malware . . . 9
4.2 Constructing Malware Detection Models . . . 11
4.2.1 Cluster-based Models . . . 11
4.2.2 Unified Model . . . 11
4.3 Predictive Performance of Models . . . 11

5 Malware Detection Game **15**
5.1 System Settings and Assumptions . . . 15
5.2 Stackelberg Model . . . 16
5.3 Game Formulation . . . 16
5.4 Utility Models . . . 17
5.5 Mixed Strategies and Expected Utilities . . . 17
5.6 Stackelberg Equilibrium . . . 18
5.7 Finding an SSE . . . 20

6 Evaluation **21**
6.1 Sensitivity . . . 21
6.2 Comparing with baselines . . . 23
6.3 Comparing baselines results . . . 24
6.4 Computational time . . . 28

7 Conclusions **29**

8 Related Work **30**
8.1 Stackelberg security games . . . 30
8.2 Hardware event-based malware detection . . . 33
8.3 Beyond hardware events . . . 34

References **36**

APPENDICES **46**

.1 Appendix 46

List of Figures

4.1 Avg. dissimilarities for clustering malware samples 10

4.2 Recall of the models for samples in each cluster. 12

4.3 F1-score of the models for samples in each cluster. 12

6.1 Defender's Normalized Expected Utility with varying reward to cost ratio (a) R^{TP}/C^{FN} and (b) R^{FN} / C^{TP}. 22

6.2 Evaluating the defender's expected utility across various baselines while altering the reward-to-cost ratio. 26

6.4 MILP solvers models 26

6.3 Evaluating the defender's expected utility across various baselines while altering the cluster sizes. 27

6.5 RBDR-2 models 27

List of Tables

6.1 MILP solvers computational time per cluster 28
6.2 Average inference time over all malware samples for cluster models classifiers. 28

1 Evaluation of previous research on malware detection using hardware events. 47
2 Hardware events employed in prior studies. 48
3 Hardware events employed in this study. 49

Chapter 1

Introduction

In today's world, security concerns have become paramount, and the evolution of machine learning (ML) models has resulted in the proliferation of ML-based malware detection frameworks. Designing a robust malware detection system has consistently been an active field of research since the inception of the first computer malware. In this busy field, the use of hardware events for malware detection has been extensively studied in prior work [57, 88, 29, 16, 14, 86].

The fundamental concept behind using hardware events for malware detection lies in the recognition that running programs exhibit phase behavior. The execution dynamics of a program can generate specific hardware events with unique frequencies and patterns corresponding to its various phases. Additionally, execution phases and their behavioral characteristics are typically distinctive for each program. Therefore, monitoring hardware events and their patterns can be employed to characterize and identify malicious program behaviors.

At a high level, the majority of previous works leveraging hardware events for malware detection typically follow three main steps: (1) data collection, (2) feature selection, and (3) model construction. Starting with (1), a dataset is compiled, comprising executable binaries of both malware and benignware. These binaries are then profiled, often through multiple executions, to generate hardware event traces for a predefined set of hardware events. Moving to (2), feature-selection algorithms analyze these traces to identify the most significant hardware events that should be monitored. This step is essential since modern processors can only monitor a fixed number of hardware events simultaneously. Finally, in (3), various machine learning algorithms are employed to construct a model based on the selected hardware events.

While some research highlights the effectiveness of leveraging hardware performance counters for detecting malware, there are opposing viewpoints suggesting their limitations [88, 26] for this purpose. Building on insights form prior works, we argue and empirically demonstrate that the effectiveness of malware detection using hardware event data depends on the selection of the subset of hardware events to monitor. Our findings lead us to the conclusion that the conventional practice of selecting a single subset of hardware events for monitoring lacks robustness against persistent and stealthy attackers.

To address this problem, we advocate a departure from the conventional approach. Instead, we propose the use of multiple subsets of hardware events, each employed to train a distinct malware detection model. As only a single subset of events can be monitored at any given time, we employ a game-theoretic approach to determine the optimal strategy for selecting the subset of hardware events to be monitored. We formulate the problem of selecting between subsets of hardware events as a *Stackelberg* game and provide a formal analysis of the equilibrium strategies in the game. In addition to the theoretical analysis of our approach, we empirically demonstrate its effectiveness by comparing it to other baselines.

Chapter 2

Background

In this section, we provide a brief overview of hardware events and hardware performance counters. Subsequently, we outline the application of hardware events in malware detection frameworks and discuss their limitations.

2.1 Hardware Performance Counters

Hardware Performance Counters (HPCs) [59, 81, 87], also referred to as performance counters, are specialized registers found in the majority of modern processors. These counters can be configured to track the occurrence of various hardware events, commonly known as microarchitectural events. Modern processors offer a multitude of these hardware events, such as the execution of load or store instructions, cache hits or misses, and branch (mis)predictions. However, it is noteworthy that most architectures typically provide only a limited number of HPCs, ranging from 2 to 8.

Microarchitectural events can be obtained at different frequencies and various levels of granularity, ranging from arbitrary code blocks to functions, libraries, and processes. Several tools and methods are available to monitor such events, including, but not limited to, dynamic or static binary instrumentation tools (e.g., `Pin` [54], `DynamoRIO` [20], and Valgrind [61]), low-level register accesses, Intel Performance Counter Monitor, `perf-tools` [28], `LIKWID` [79], and PAPI [59].

Monitoring HPC measurements can be accomplished using an interrupt-driven sampling approach. This feature is typically enabled in most modern processors through Performance Monitoring Interrupt (PMI), which generates an interrupt once a given number of events

occur in the system. For instance, HPC measurements may be sampled when the number of retired instructions exceeds a specified threshold.

It is crucial to note that measurement data may exhibit non-deterministic characteristics even during deterministic code execution. Sources of non-determinism include interference on shared resources from other running processes, out-of-order execution, and branch (mis)predictions.

2.2 HW Events & Malware Detection

Previous research extensively explores the application of hardware events in various domains, including workload profiling, power modeling, compiler optimization, and real-time power estimation, as well as thread scheduling [11, 60, 23, 13, 77]. Additionally, the utilization of hardware events for malware detection has been a subject of thorough investigation [88, 29, 16, 14, 86]. This approach aims to characterize and identify both normal and abnormal program behaviors based on hardware events and their patterns observed during program execution.

The fundamental concept underlying the use of hardware events for malware detection is the recognition that running programs exhibit phase behavior. The execution dynamics of a program can generate specific hardware events with unique frequencies and patterns corresponding to its various phases. Furthermore, execution phases and their behavioral characteristics are typically distinctive for each program. Hence, the monitoring of hardware events and their patterns can be employed to characterize and identify malicious program behaviors.

At a high level, the majority of prior works on leveraging hardware events for malware detection typically follow three main steps: (1) data collection, (2) feature selection, and (3) model construction. Starting with (1), a dataset is compiled, comprising executable binaries of both malware and benignware. These binaries are then profiled, often through multiple executions, to generate hardware event traces for a predefined set of hardware events. Moving to (2), feature-selection algorithms analyze these traces to identify the most significant hardware events that should be monitored. This step is essential since, as mentioned before, modern processors can only monitor a fixed number of hardware events simultaneously. Finally, in (3), various machine learning algorithms are employed to construct a model based on the selected hardware events. Table 1 and Table 2 in Appendix .1 provide a summary of related work, outlining their approaches in these three steps.

Regarding the use of hardware events in malware detection, there are some notable limitations in the prior works. Here, we briefly outline some of these limitations.

Verification of malware execution. Ensuring an accurate execution of malware samples is challenging. Merely running a malware binary file does not guarantee that the malware operates at its full capacity. Many malware types check for specific conditions before launching their intended attacks, and if any of these conditions are not met, the malware binary might exit. Moreover, issues such as missing dependencies can impact the execution of malware samples.

Virtualized environments. Although running malware on a bare-metal system could offer a more accurate profiled data, it is often unfeasible to execute malware directly on a system. To address this issue, prior work has employed virtualized environments, such as VirtualBox [35]. While virtualized environments provide a safe and sandboxed space for executing malicious files, they could interfere with the intended execution of malware, leading to noisy hardware event data [33]. In fact, in some cases, hardware events measured within a virtual machine could substantially differ from those obtained on a bare-metal system [88].

Time series-based analysis. Accurate analysis of hardware event data is crucial due to its multivariate time series nature. Selecting the top hardware events requires employing feature selection methods specifically designed for multivariate time series data. Some prior works manually choose hardware events [57, 35], while others utilize dimension reduction methods such as Principal Component Analysis (PCA) [45, 74, 88]. However, capturing the time-series nature of samples with PCA may present challenges, leading to noticeable variations in the selection of hardware events across different studies (refer to Table 2).

Closely tied to feature selection is model training. When training a machine learning model on hardware event data, a time series-based approach is crucial to capture temporal patterns. Previous studies have often used models not explicitly designed for time series data, such as decision trees, SVM, Bayes Net, and KNN [45, 74, 29, 57].

Chapter 3

Methodology

In this section, we describe our approach in collecting malware and benignware samples. We further discuss our execution environment and setup. At the end, we outline our feature selection and classification methods based on time series data.

3.1 Malware and Benignware Samples

For our malware samples, we focus on Linux-based binaries. We collect a total of 190 binaries for our study from various sources and malware databases. Examples include MalwareBazaar [4], VirusSamples [7], and GonnaCry [3]. These binaries represent diverse malware families, encompassing ransomware, viruses, rootkits, and backdoor malware. We specifically select binaries that exhibit observable activity during execution, such as printing logs, establishing connections to remote servers, or activating the filesystem. As for our benignware, we compile 50 binaries from benchmarks such as `LMbench`[58] and `MiBench`[38], along with frequently used Linux applications like text editors, browsers, and lightweight Linux binaries, such as `chmod`, `grep`, and `rm`.

3.2 Execution Environment

We execute each binary file within a Linux container (`LXC`). Containers are chosen over virtual machines (VMs) due to prior research demonstrating significant performance degradation when using VMs in comparison to Linux containers (see, for example, [33]). A new container is initialized for each malware run.

For our data collection, we focus on 23 different hardware events carefully selected to cover most features used in prior published research. We use `perf tools` to monitor these hardware events at 10ms intervals during a one-minute execution of each binary sample. To collect data for all 23 hardware events, we run each binary four times. This process generates a single timestamped trace data for each malware and benignware.

We run our experiments on an AMD Ryzen Threadripper 3945WX processor, which, like many modern AMD processors, allows concurrent access to a maximum of 6 hardware event counters. To replicate a standard computer environment, we create shared directories populated with random files, serving as potential malware targets. Additionally, we grant network access to the container to allow binaries to establish remote connections.

3.3 Feature Selection

Due to the limited number of hardware performance counters (HPCs) in modern processors, only a small number of hardware events (e.g., 4 or 8) can be monitored at any given time. Consequently, it becomes necessary to choose a subset of hardware events for monitoring. This selection process can be achieved using feature selection methods from machine learning. These methods involve choosing the most relevant subset of features from all available features that can effectively represent the original data.

In this study, we interpret our collected hardware event traces as multivariate time series (MTS) data, where each monitored hardware event serves as a statistical variable. For MTS-based feature selection, we utilize the `fsMTS` package [66]. Extracting stable features for high-dimensional data with low sample size is known to be challenging [12, 47].

To enhance the robustness of the feature selection process, ensemble techniques are shown to be promising [70]. As a result, we use `fsMTS`'s ensemble feature selection method [67] with the majority voting rule to combine several different feature selection algorithms. These algorithms include those based on independent autoregressive lags [68], mutual information [52], cross-correlation [62], random forests [65], least-angle regression [36], graphical lasso [34], and partial spectral coherence [27]. Using this approach, we select the top 6 hardware events for each malware sample.

3.4 Machine Learning Model

To detect malware from benignware, we train a machine learning model on our dataset using the `sktime` library [53]. This open-source Python library is specifically designed for

time series data analysis and forecasting. For our classifier, we employ `sktime`'s implementation of the bootstrap aggregating (bagging) classifier [18]. Bagging is an ensemble learning method that combines multiple classifiers to improve overall predictive performance.

The bagging classifier takes a base classifier as input and creates multiple copies of the classifier. Each copy is then trained on a randomly sampled subset of the training data. Once all base classifiers are trained, their predictions are aggregated to form the final ensemble prediction, typically achieved through majority voting. This approach reduces overfitting and enhances the model's robustness by minimizing prediction variance.

In our study, for the base classifier, we utilize `sktime`'s ROCKET (randomized convolutional kernel transform) classifier [30]. ROCKET is a fast kernel-based linear classification method for time series data that achieves high classification accuracy by generating a large number of random convolutional kernels.

Chapter 4

Hardware Events Selection

In this section, we argue and empirically demonstrate that the effectiveness of malware detection using hardware event data depends on the selection of the subset of hardware events to monitor. To accomplish this, we begin by clustering our malware samples based on their similarities, specifically focusing on their top six representative hardware events. For each cluster, we identify the most frequently occurring events and utilize them to construct a distinct malware detection model. Simultaneously, we determine the overall top six hardware events across all malware samples and use them to create a unified malware detection model. Subsequently, we compare the predictive performance of these models across different clusters. Our findings lead us to the conclusion that the conventional practice of selecting a single subset of hardware events for monitoring lacks robustness against persistent and stealthy attackers.

4.1 Clustering Malware

As discussed earlier in §3.3, we utilize an ensemble feature selection method to identify the top six hardware events for each malware sample. The top hardware events may vary among different malware samples. To underscore these variations, we employ clustering techniques to group malware samples based on their similarities in terms of their top selected features. To implement this, we represent each malware sample as a categorical data point, employing a one-hot encoding of its top six features. Subsequently, we employ the k-modes clustering algorithm [42] to group our malware samples.

The k-modes clustering is an unsupervised technique designed for grouping categorical data, akin to how k-means clusters numerical data. For a specified value of k clusters, the

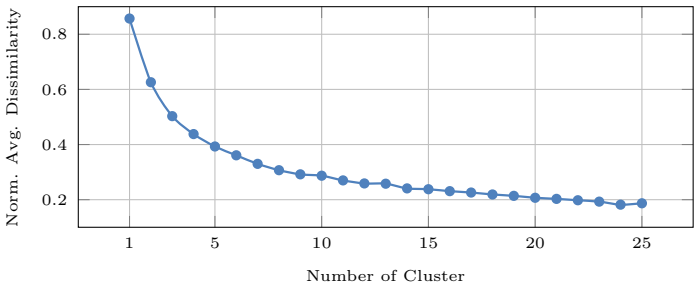

Figure 4.1: Avg. dissimilarities for clustering malware samples

clustering process involves the following iterative steps. Initially, k samples are randomly chosen to act as the initial cluster centroids. Dissimilarities between each sample and the cluster centroids are then computed based on a given dissimilarity measure. Each sample is subsequently assigned to its closest cluster. After this assignment, new cluster modes are defined by identifying the most frequent category for each attribute within each cluster. This process of calculating dissimilarities, reassigning observations to clusters, and updating cluster modes is iteratively repeated until the maximum number of iterations is reached or no further reassignments are necessary.

The dissimilarity metric in kmodes is determined as follows: If x and y represent two categorical data objects defined by m features or attributes, the dissimilarity metric is calculated as such:

$$d(x, y) = \sum_{j=1}^{m} \delta\left(x_j, y_j\right)$$

$$\delta\left(x_j, y_j\right) = \begin{cases} 0 & \text{if } x_j = y_j \\ 1 & \text{if } x_j \neq y_j \end{cases}$$

We execute k-modes with $k = 1, 2, \ldots, 25$ for 200 iterations each. For every k, we compute the dissimilarity between each sample and the mode of the final cluster to which it is assigned. The (normalized) average of these dissimilarities is depicted in Figure 4.1. As anticipated, the average dissimilarity diminishes with an increase in the number of clusters. For the subsequent analyses in this paper, we select 15 clusters without loss of generality; our results and conclusions generalize to other numbers of clusters.

4.2 Constructing Malware Detection Models

To investigate the impact of hardware event selection on the predictive performance of malware detection models, we create multiple malware detection models using various subsets of hardware events.

4.2.1 Cluster-based Models

Employing 15 clusters results in 15 distinct subsets of hardware events, each representing the top six features of a specific cluster. For each subset, we construct a unique malware detection model in the form of a machine learning classifier. For model verification and testing, we split our dataset into training and testing sets using random sampling without replacement. Specifically, 70% of the samples are utilized for training, while the remaining 30% are allocated to the testing set. To maintain an unbiased division, we allocate 70% of both malware and benignware samples for training, and 30% for testing. Additionally, to ensure the inclusion of every malware cluster in both the training and testing phases, we independently sample from each cluster to create our training and testing sets. For the evaluation of the model's performance on malware within each specific cluster, we conduct separate tests for each cluster.

4.2.2 Unified Model

We determine the overall top six features based on their frequency of appearance across all malware samples, akin to the centroid of the single cluster produced by the k-modes algorithm with $k = 1$. These features serve as the basis for training a single machine learning classifier. For model verification and testing, we follow the same data splitting approach mentioned earlier for cluster-based models.

4.3 Predictive Performance of Models

To evaluate the performance of various models, this subsection primarily focuses on the *recall* and *F1-score* metrics.

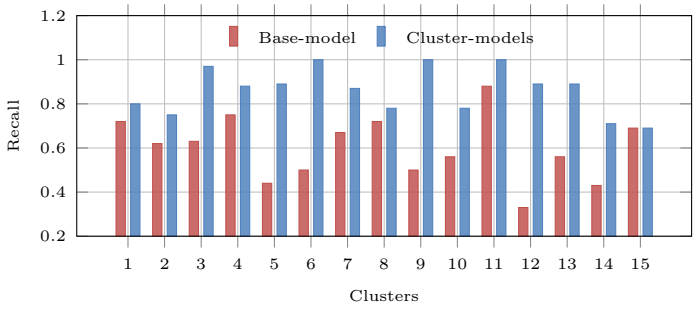

Figure 4.2: Recall of the models for samples in each cluster.

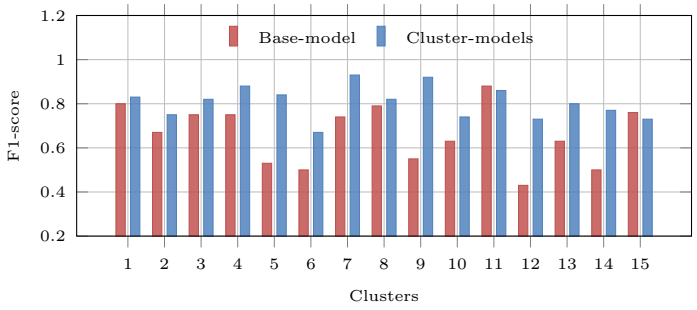

Figure 4.3: F1-score of the models for samples in each cluster.

The recall metric measures the proportion of actual positives that were correctly identified and is calculated as follows:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positive} + \text{False Negative}}.$$

In contrast, the *precision* metric assesses the fraction of all positives that were correctly identified:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}.$$

The F1-score provides a balanced evaluation of the model performance by incorporating both precision and recall:

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}.$$

Figure 4.2 and Figure 4.3 illustrate the recall and F1-score values of our models restricted to samples from each cluster[1]. The red bars in these figures represent the unified model, presenting its recall and F1-score metrics across different clusters. Meanwhile, the blue bars correspond to per-cluster models, displaying the recall and F1-score of each model concerning the specific cluster whose top features are utilized for constructing the model.

The unified model exhibits a broad range of recall and F1-score values across clusters. For example, the model's recall values for samples in clusters 5 and 12 are less than 50%, while the recall value for samples in cluster 11 is 90%. Similarly, the model's F1-score values for samples in clusters 12 and 14 are less than 50%, while the F1-score value for samples in cluster 11 is close to 90%. In contrast, the recall and F1-score values of per-cluster models for their corresponding classes are almost consistently higher[2], ranging approximately from 70% to 100% for recall and 70% to 95% for F1-score. This is expected as each model is trained on the representative hardware events of samples within each cluster.

Our results highlight the critical importance of selecting the "right" subset of hardware events for constructing effective malware detection systems. It is evident that the conventional practice of selecting a single subset of hardware events and using it to train a malware detection model does not result in a robust malware detection system capable

[1]Note that for F1-score, all benignware samples are included with each cluster to measure false positives.

[2]There is one exception. The recall value of the model trained on top features of cluster 15 for samples of cluster 15 is the same as the recall value of the unified model for the same samples, and the F1-score value of the model is 3% lower than that of the unified model. This can be attributed to the fact that the top features of cluster 15 bear a high resemblance to the overall top features of the unified model.

of effectively handling all types of malware. This is true even when the malware detection model is an ensemble of powerful classifiers. Building on this insight, we propose a departure from the conventional approach. Instead, we advocate for the use of multiple subsets of hardware events. We use each subset to train a different malware detection model. Since only a single subset of events can be monitored at any given time, in the next section, we leverage a game-theoretic approach to determine the optimal strategy for selecting the subset of hardware events that should be monitored.

Our results underscore the critical importance of carefully selecting the "right" subset of hardware events when constructing effective malware detection systems. It becomes evident that the common practice of choosing a single subset of hardware events for training a malware detection model falls short in creating a robust system capable of effectively handling all types of malware, even when employing an ensemble of powerful classifiers.

Building upon this insight, we advocate a departure from the conventional approach. Instead, we propose the use of multiple subsets of hardware events, each employed to train a distinct malware detection model. As only a single subset of events can be monitored at any given time, the next section employs a game-theoretic approach to determine the optimal strategy for selecting the subset of hardware events to be monitored.

Remarks. It is crucial to note that our proposed approach is orthogonal to ensemble learning methods (e.g., [75]). Indeed, as discussed in §3, all malware detection models in this paper utilize an ensemble of powerful ML classifiers. However, in each ensemble, all classifiers use the same subset of hardware events. Our proposed strategy involves deploying multiple models, each trained on a distinct subset of hardware events, offering a complementary perspective to ensemble methods.

Chapter 5

Malware Detection Game

In this section, we formulate the problem of selecting between subsets of hardware events as a *Stackelberg* game. We then provide an analysis of the equilibrium strategies in the game. In the text, our hardware event-based malware detection model is referred to as Salus.

5.1 System Settings and Assumptions

The input to Salus consists of a finite set of trained malware detection models. Each model is associated with a specific subset of hardware events. As discussed earlier, due to the limited number of HPCs, only a single model can be deployed at any given time. The primary challenge is to find an optimal strategy for choosing between malware detection models.

To address this challenge, we adopt a game-theoretic approach, modeling the interactions between Salus and attackers (e.g., hackers or malware developers) as a *security game*. In our model, Salus chooses a malware detection model to be deployed, while a potential attacker chooses a specific malware to run on the system.

We make few key assumptions in our model. First, we assume that attackers are strategic, meaning they adapt their attack strategy based on their observations of Salus's actions. Second, we assume that attackers are persistent and have complete visibility into Salus's actions over time before deciding on their attack strategy. While these assumptions grant significant power to attackers and may not universally apply in practical scenarios, they allow us to optimize Salus's strategy for a worst-case scenario.

We additionally assume that the system faces many non-colluding attackers. We model each attack as a separate game with a single attacker. All attackers share a common objective–to inflict maximum damage. We model this by a fixed utility function for all attackers, enabling Salus's optimal strategy to generalize across all potential attacks.

5.2 Stackelberg Model

When confronted with persistent and stealthy attackers, opting for a malware detection model with highest predictive performance on historical data becomes suboptimal. Such a strategy is vulnerable because attackers can adapt their behavior to exploit the chosen model. In fact, deterministically selecting a specific malware detection model is an easily exploitable strategy, as attackers can deploy malware that evades detection by that particular model. Therefore, to optimize its defense strategy and counter adaptive attackers, Salus must introduce randomization when selecting between malware detection models.

To determine an optimal randomized strategy, we formulate the security game between Salus and a potential attacker using the *Stackelberg* model (also known as the *leadership* or *commitment* model). The Stackelberg model, introduced by von Stackelberg [83] in the context of Cournot's duopoly model [25], involves a game played sequentially between two players: the *leader* and the *follower*. The leader plays first by committing to a mixed strategy[1], and the follower observes the distribution of the leader's actions before choosing a strategy in response.

5.3 Game Formulation

In our setting, Salus assumes the role of the leader, responsible for selecting a malware detection model to deploy, while a potential attacker acts as the follower, choosing a malware in response to the leader's strategy. Although a malware could theoretically be selected from a continuum of codes, for simplicity, we model the game as a finite one. We assume that there exists only a finite set of *behavioral* malware classes (determined, for example, by their representative top hardware events, as discussed in §3.3), denoted by set M, from which an attacker makes their selection. On the Salus side, the set of available actions is denoted by D, representing all trained detection models.

[1] A mixed strategy is a probability distribution over all available actions.

5.4 Utility Models

We estimate the (dis)utility of Salus as the leader for choosing detection model $d \in D$ when an attacker chooses malware $m \in M$ as follows:

$$U_l(d, m) = R^{TP} p_d(m) - C^{FN}(1 - p_d(m))$$

where R^{TP} represents the reward of true positives (i.e., the reward of detecting a malware), C^{FN} represents the cost of false negatives (i.e., the cost of misclassifying a malware), and $p_d(m)$ is the probability that model d detects malware m.

We further estimate the utility of the follower (i.e., an attacker) for choosing malware $m \in M$ in response to Salus choosing d as follows:

$$U_f(d, m) = R^{FN}(1 - p_d(m)) - C^{TP} p_d(m),$$

where R^{FN} is the reward of false positives (i.e., the reward of evading the detection model) and C^{TP} is the cost of true positives (i.e., the cost of being detected by the model).

Remarks. We note that $p_d(\cdot)$'s are parameters of the utility functions that depend on the detection models. These parameters can be estimated by constructing the detection models and testing them on historical data. On the other hand, R^{TP}, C^{FN}, R^{FN}, and C^{TP} are model-agnostic parameters, and estimating them requires domain-specific knowledge. In the following chapter, we instantiate the model-dependent parameters of our utility functions using data from our experiments. Additionally, we conduct sensitivity analysis on the model-agnostic parameters to demonstrate how our results change with variations in these parameters.

It is important to note that the utility models presented in this section are not intended to be universally applicable to all real-world scenarios. Instead, they serve to illustrate the construction of a realistic utility model tailored to represent a precision/recall perspective towards detection models. When applying our framework to different real-world scenarios, adjustments and customizations to the utility model may be necessary to accurately represent other perspectives.

5.5 Mixed Strategies and Expected Utilities

The set of *mixed strategies* (also known as randomized strategies) for the leader is denoted by $S_l = \Delta(D)$, where $\Delta(D)$ represents the set of all probability distributions over the set

D. By $s_l(d)$, we denote the probability that detection model d is employed under the mixed strategy s_l. A *pure strategy* is a mixed strategy that puts probability 1 on a single action. Similarly, we can define the set of mixed strategies for the follower, $S_f = \Delta(M)$. We can then overload U_l (and U_f) to capture the expected utility of the leader (and the follower) when the arguments are mixed strategies:

$$U_l(s_l, s_f) \triangleq \sum_{d \in D} \sum_{m \in M} s_l(d) s_f(m) U_l(d, m).$$

In the context of our formalized notation, the execution of our security game follows a structured sequence. Initially, Salus chooses a distribution over detection models ($s_l \in S_l$). Subsequently, an attacker observes the Salus's selected strategy (s_l) and responds by choosing a strategy ($s_f \in S_f$). The game proceeds to randomly sample an action profile (d, m) based on the selected strategies, where $d \sim s_l$ and $m \sim s_f$. Finally, both Salus and the attacker receive their utility ($U_l(d, M)$ and $U_f(d, M)$).

5.6 Stackelberg Equilibrium

Salus's (dis)utility depends not only on its own defense strategy but also on the attack strategy. The same holds for an attacker's utility. In game theory, a strategy is considered a player's best-response to the other players' strategy when it maximizes the expected utility of the player. In our setting, the set of Salus's best-responses, BR_l, to an attacker's strategy s_f is represented by:

$$BR_l(s_f) \triangleq \{d \in D \mid U_l(d, s_f) \geq U_l(d', s_f)\ \forall d' \in D\}.$$

Similarly, we can define $BR_f(s_l)$.

In a Stackelberg game, a pair of strategies for the leader and the follower form a *Stackelberg equilibrium* if they are best-responses to each other. Following the security games literature, we adopt a common refinement to the Stackelberg game called the Strong Stackelberg Equilibrium (SSE) [19]: A pair of strategies (s_l^*, s_f^*) form an SSE if they satisfy all the following conditions:

- The leader plays a best-response:

$$U_l(s_l^*, BR_f(s_l^*)) \geq U_l(s_l', BR_f(s_l'))\ \ \forall s_l' \in S_l.$$

- The follower plays a best-response:

$$U_f(s_l^*, s_f^*) \geq U_f(s_l^*, s_f') \;\; \forall s_f' \in S_f.$$

- The follower breaks ties in favor of the leader:

$$U_l(s_l^*, s_f^*) \geq U_l(s_l^*, s_f) \;\; \forall s_f \in BR_f(s_l^*).$$

The first two conditions are crucial for establishing an equilibrium, and they are common to both the Stackelberg equilibrium and SSE. The third condition requires that if the follower faces indifference between multiple strategies, they opt for the one that maximizes the leader's utility. While this condition might appear inconsistent with the follower's objective in the context of security games, it remains justifiable, as the leader can almost always select a strategy very close to the equilibrium strategy to make the follower strictly prefer the desired SSE strategy [84].

Compared to some other refinements of the Stackelberg equilibrium with different tie-breaking rules, an SSE always exists in all Stackelberg games, making it an attractive solution concept. Additionally, to find an SSE, it is enough to only consider a pure strategy for the follower [24]. This is because if the follower randomizes between multiple pure strategies in response to the leader's strategy, then all those pure strategies must yield the same utility to the follower and also to the leader (due to the tie-breaking rule). Therefore, we can just focus on one of those pure strategies as all of them produce the same outcome. We use this fact in the next subsection when we formulate the game to find its SSE.

5.7 Finding an SSE

To find an SSE, the game can be formulated as the following mixed-integer linear program (MILP) [43]:

$$
\begin{aligned}
&\text{Max.} && u_l, && && (5.1)\\
&\text{s.t.} && p_d \in [0,1] && \forall d \in D,\\
& && \sum_{d \in D} p_d = 1, &&\\
& && q_m \in \{0,1\} && \forall m \in M,\\
& && \sum_{m \in M} q_m = 1, &&\\
& && 0 \leq u_f - \sum_{d \in D} U_f(d,m) p_d \leq L(1-q_m) && \forall m \in M,\\
& && u_l - \sum_{d \in D} U_l(d,m) p_d \leq L(1-q_m) && \forall m \in M,
\end{aligned}
$$

where L is a constant number with an arbitrary large value. In this formulation, the variables p_d determine the mixed strategy of the leader. The first and second constraints ensure that these variables define a probability distribution over all detection models available to Salus. Similarly, the variables q_m determine the pure strategy of the follower. The third and fourth constraints ensure that only a single q_m is set to one, indicating the optimal pure strategy of the follower.

In addition to p_d's and q_m's, (5.1) introduces two other variables: u_l and u_f to capture the expected utility of the leader and follower, respectively. For the follower, this is enforced by the fifth constraint. When $q_m = 1$, u_f must be both weakly greater and weakly less than $\sum_{d \in D} U_f(d,m) p_d$, which means it has to be equal to $\sum_{d \in D} U_f(d,m) p_d$. When $q_m = 0$, u_f only has to be weakly greater than $\sum_{d \in D} U_f(d,m) p_d$. The last constraint serves a similar role for the leader's expected utility. Since u_l is being maximized in the objective, there is no need for a lower bound.

Solving the MILP in (5.1) finds the optimal (mixed) strategy of Salus assuming that the attacker best-responds.

Chapter 6

Evaluation

To evaluate the effectiveness of our proposed method, we employ two assessment techniques. Initially, a sensitivity analysis is conducted to explore the impact of adjusting the solution's parameters on the final outcome. Following this, we compare the proposed solution with baseline strategies.

In our experimental configuration, we utilize the Mixed-Integer Linear Programming (MILP) solver to resolve our game model. After determining the defender's strategy using the solver, we iterate the game for 100 rounds. In each iteration, the leader selects a malware detection model, while simultaneously, the attacker chooses a specific malware. At the end of each round, the defender's utility is computed. Finally, we calculate the average utility for the defender.

6.1 Sensitivity

In this section, we analyze the effects of variations in the reward-to-cost ratios for both the attacker and defender on the outcomes of the study. To assess this, we systematically modified the reward and cost parameters of the defender and the attacker. Subsequently, we examined the defender's utility across multiple cases to gauge the sensitivity to these ratio changes. We performed the sensitivity analysis in two cases:

In the initial case, we altered the reward and cost parameters of the attacker, denoted as R^{FN} and C^{TP} in the attacker's utility model, while holding the defender's reward-to-cost ratio constant. This facilitated an examination of the impact on the defender's utility stemming from adjustments in R^{FN} and C^{TP}.

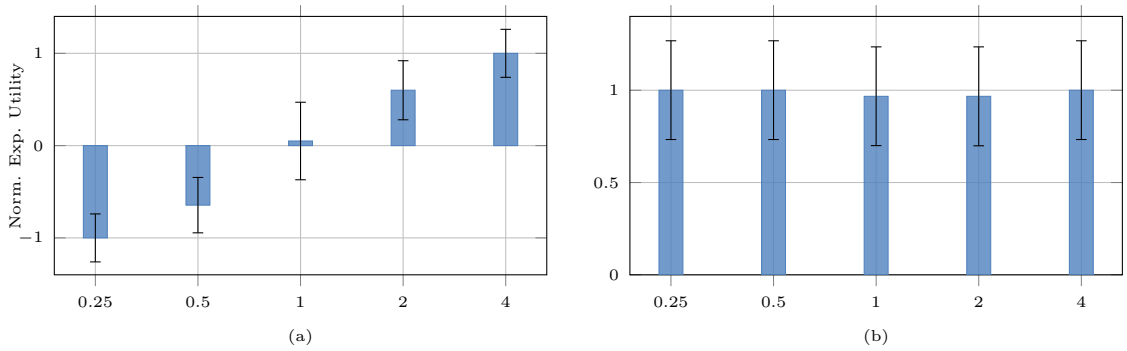

Figure 6.1: Defender's Normalized Expected Utility with varying reward to cost ratio (a) R^{TP}/C^{FN} and (b) R^{FN} / C^{TP}.

In the second case, we modified the reward and cost parameters of the defender, represented by R^{TP} and C^{FN} in the defender's utility model. While maintaining a constant attacker's reward-to-cost ratio. This allowed us to observe and analyze the influence of variations in the defender's utility resulting from changes in R^{TP} and C^{FN}.

Figure 6.1(b) illustrates the impact of varying the R^{FN} to C^{TP} ratio on the defender's utility. Notably, adjusting this ratio has minimal effect on the defender's utility. This lack of sensitivity is advantageous, considering the inherent challenge of accurately modeling the attacker's utility without comprehensive information. The limited changes in utility are favorable for system design, offering stability even when dealing with uncertainties in the attacker's model.

Figure 6.1(a) illustrates the scenario where we vary the defender's reward-to-cost ratio: R^{FN} and C^{TP}. As depicted in the data presented in Figure 6.1(a), when the reward and cost values align, the defender experiences a utility close to zero. In addition, as the reward for detecting the attackers' attacks increases, the defender achieves a higher utility.

We analyzed the defenders' strategy vector across different reward-to-cost ratios (R^{TP}/C^{FN}) and observed a consistent pattern: the defender's mixed strategy remained unchanged despite variations in these ratios. Moreover, when adjustments were made to the attacker's reward-to-cost ratio (R^{FN}/ C^{TP}), the stability of the defender's mixed strategy was evident. These observations highlight a robust insensitivity to variations for game parameters in the defenders strategy. This suggests that the optimal defender strategy can be reliably determined irrespective of specific values assigned to game parameters. In the context of our specific game model, the MILP solver consistently employed mixed strategies between two models, Model-3 and Model-5. The top features of these two models considering the

common events between the two model are : `branches`, `branch-misses`, `branch-loads`, `branch-load-misses`, `dTLB-load-misses`, `l2-dtlb-misses`, `l1-dtlb-misses`, `ls-mab-alloc-stores`, `L1-dcache-loads` and `l2-cache-accesses-from-dc-misses`.

6.2 Comparing with baselines

In order to evaluate the efficacy of our proposed solution, we conduct a comprehensive comparison with a variety of baseline strategies that the defender might potentially employ. The objective of this comparative analysis is to ascertain whether our proposed solution outperforms the baseline strategies in terms of utility for the defender.

In the following discussion, we delve into the details of each baseline strategy that forms part of our comparative analysis. This clarification involves a comprehensive exploration of the underlying principles, mechanisms, and considerations associated with each baseline strategy. By elucidating the intricacies of these baseline strategies, we aim to establish a clear understanding of their functionalities and, in turn, facilitate a robust comparative assessment against our proposed solution.

This detailed investigation and comparison contribute to a sophisticated evaluation of the proposed solution's effectiveness, offering valuable insights into its potential advantages and areas of improvement when faced with various strategies that represent the defender's alternative courses of action. In the following, each baseline strategy that we employed is elucidated.

- **Uniform Randomization:** In the Uniform Randomization Strategy, the defender employs a tactic where they deliberately randomize their choice among all available pure strategies. This approach involves assigning equal probabilities to each pure strategy, essentially distributing the defender's choices uniformly across the entire set of strategies.

- **Best average detection rate**

  Best average detection rate refers to a strategy employed by the defender in a decision-making scenario. In this context, the defender utilizes a pure strategy denoted as $s^{'}$, and the criterion for selecting this strategy is based on achieving the highest average detection rate. In the following formula, the variable M denotes the set of malware samples, and $p_d(m)$ signifies the probability associated with the detection of a particular malware sample.

Mathematically, this strategy is formulated as follows:

$$s^{'} \in \arg\max_{i} \left(\frac{1}{|M|} \sum_{m \in M} p_d(m) \right)$$

- **Randomized best average detection rate** In the Best average detection method, the objective is to identify the strategy that maximizes the average detection rate. with extending, the Randomized best average detection rate method involves a randomization process, wherein the defender's strategy is selected from a set of the m strategies with the best average detection rates. In this experimental setup, we employed the values of m = 2 and m = 5 for the comparative analyses.

- **Deterministic best response:**

  In this approach, the defender, following the deterministic best response strategy, responds to the attacker's actions by choosing a single strategy. Unlike the preceding strategies, the deterministic best response strategy integrates the actions of the attacker into its decision-making process. Algorithm 1 gives the procedure for finding s^*, the attacker's best response v^* to s^*, and the utilities for each player.

Algorithm 1 Deterministic Best Response

$\Delta BR_a(s) :=$ attackers best response $\forall s \in S$
for $s \in S$ **do**
    $BR_a(s) \leftarrow \underset{v \in V}{\text{argmax}} u_a(s, v)$
    $s^* \leftarrow \text{argmax}\, u_d(s, BR_a(s))$ ▷ defender best response
    $v^* \leftarrow BR_a(s^*)$ ▷ attacker best response to s^*
end for
return $u_d(s^*, v^*)$, $u_a(s^*, v^*)$ ▷ utilities for players

6.3 Comparing baselines results

In this section, we assess the employed method, which utilizes the MILP (Mixed-Integer Linear Programming) solver. We compare it with several baseline strategies, including

Uniform Randomization (UR), Best Average Detection Rate (BDR), Randomized Best Average Detection Rate with $m = 2$ (RBDR-2), and $m = 5$ (RBDR-5) and Deterministic Best Response as (DBR).

Figure 6.2 shows the comparison of the MILP solution with the baselines. In these experiments, the ratio of the attacker's reward to cost remained constant, while the defender's parameters were systematically adjusted, corresponding to varying reward-to-cost ratios.

The x-axis in Figure 6.2 illustrates various reward-to-cost ratios, while the y-axis depicts the expected utility of the defender corresponding to each strategy. As illustrated in the plot, the MILP-based solution outperforms all baseline strategies across various reward-to-cost ratios (0.25, 0.5, 1, 2, 4). As evident from the plots in Figure 6.2, when the reward-to-cost ratio exceeds 1, the defender's expected utility becomes positive, while all baseline strategies maintain negative values. An additional inference drawn from the graphical representations is that randomizing the best average detection rate with a parameter $m = 5$ (RBDR-5) yields superior results compared to the strategies with $m = 2$ (RBDR-2) or the original strategy (BDR).

Furthermore, our evaluation involved a comparison between MILP solutions and baseline strategies while varying the defender's strategy space, specifically focusing on other cluster sizes (10, 20, 25). This experiment aimed to assess the performance across various cluster sizes and analyze how the MILP solutions compared to the baselines under these changing conditions.

As illustrated in Figure 6.3, it is apparent that irrespective of the cluster size under consideration, the MILP solution consistently outperforms all baseline strategies. This observation implies that the MILP solution consistently delivers optimal outcomes across diverse scenarios, showcasing robustness and independence from variations in the defender's parameters or strategy space.

Figure 6.4 demonstrates the predictions generated by the models chosen through the MILP solver for each individual malware sample. The MILP solver combines information from two models, specifically model 3 and model 5, as evident in the visual representation. The horizontal line in the figure represents the threshold for distinguishing between malware and benignware. Probabilities above this line are classified as malware predictions. Notably, the figure reveals instances where model 5 outperforms model 3 in predicting malware for certain samples.

Upon solving the game with the MILP solver, the optimal response for the attacker

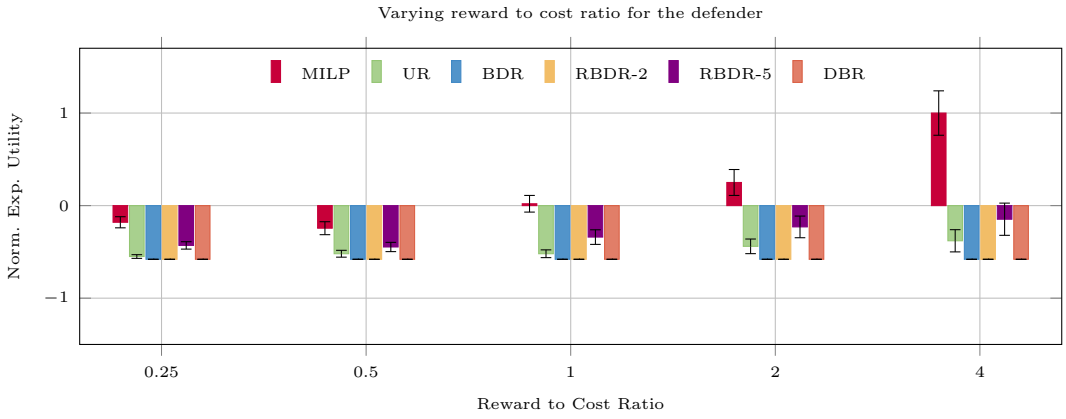

Figure 6.2: Evaluating the defender's expected utility across various baselines while altering the reward-to-cost ratio.

is to select malware sample 170. This choice is substantiated by the positive predictions from both models for this particular sample, as illustrated in Figure 6.4. Consequently, the defender receives a positive utility.

However, Figure 6.5 illustrates the models selected by the RBDR-2 baseline strategy. This strategy opts for models 3 and 11, blending a defender strategy between these two models. Despite the RBDR-2 selecting the top two models with the highest prediction rates, it results in a negative utility for the defender. The negative utility arises from the attacker's optimal response to this strategy, which is to choose malware 20. Accordingly, both selected models fail to detect this specific malware, leading to a negative outcome for the defender.

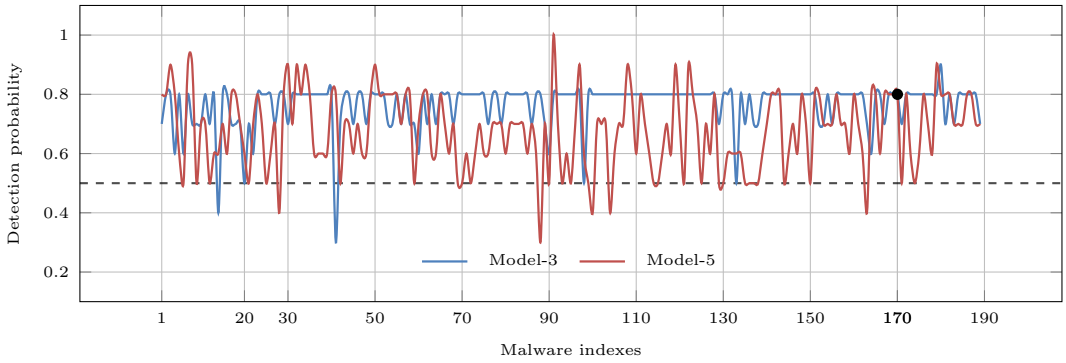

Figure 6.4: MILP solvers models

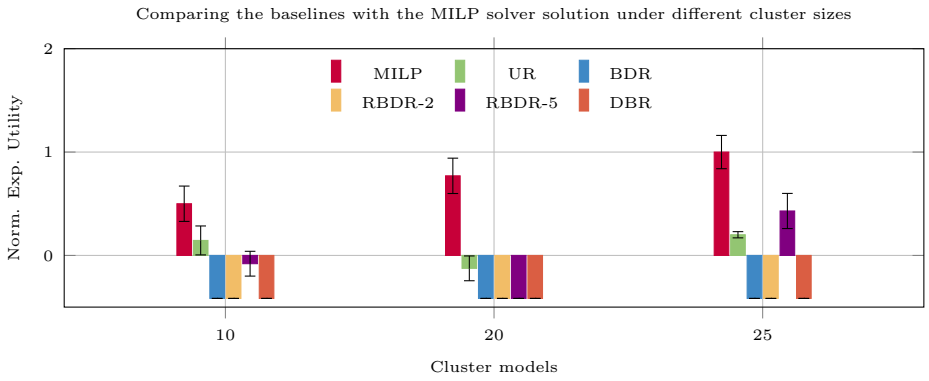

Figure 6.3: Evaluating the defender's expected utility across various baselines while altering the cluster sizes.

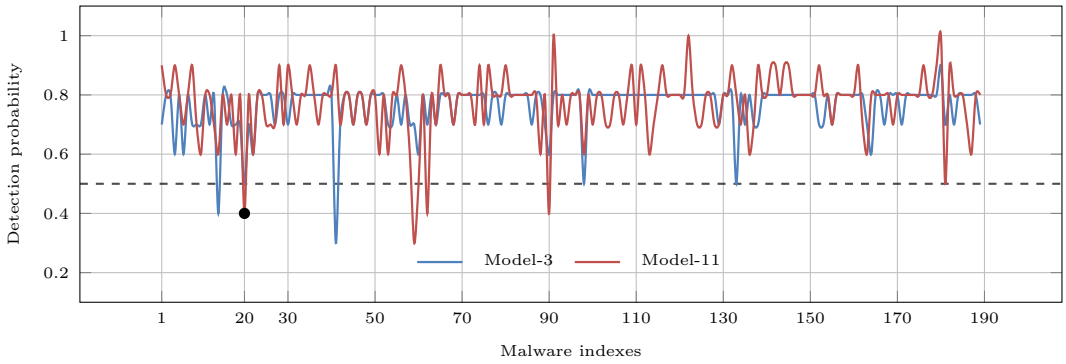

Figure 6.5: RBDR-2 models

6.4 Computational time

Cluster models	Computational time
10	8.463 s
15	12.231 s
20	22.320 s
25	28.866 s

Table 6.1: MILP solvers computational time per cluster

Table 6.1 displays the computation times for determining the optimal strategy using the MILP solver in cvxpy across various clusters. As observed in the table, augmenting the strategy space, such as with cluster models, results in an increase in computational time required to determine the optimal solution.

Cluster models classifiers	Avg inference time per sample
15	0.5687 s

Table 6.2: Average inference time over all malware samples for cluster models classifiers.

In Table 6.2, the average inference time is presented for obtaining predictions across all 15 trained classifiers for the entirety of malware samples. The inference times were recorded using a T4 GPU accessible on Google Colab, equipped with the following specifications: a system RAM of 12.7 GB, GPU RAM of 15.0 GB, and a disk capacity of 201.2 GB.

Chapter 7

Conclusions

The exploration of hardware events in the realm of malware detection has been a subject of extensive research, yielding diverse perspectives on their effectiveness. This study contributes to this body of knowledge by showcasing that the success of malware detection critically depends on the precise selection of hardware events during the training of detectors.

Moreover, we underscored the inadequacy of the traditional approach, which involves choosing a singular subset of hardware events for training a malware detection model. This limited strategy falls short in constructing a resilient detection system capable of effectively addressing the diverse forms of malware. In response to this challenge, we advocated for the adoption of multiple subsets of hardware events to create distinct malware detection models.

Given the constraint that only one subset of events can be monitored at any given time, our approach embraced a game-theoretic perspective. This strategic framework was employed to determine the optimal strategy for selecting the most relevant subset, ensuring an adaptive and dynamic approach to malware detection.

Furthermore, the empirical validation of our proposed methodology was substantiated through a comparative analysis with other baseline approaches. This empirical evidence serves to affirm the efficacy of our multifaceted approach in enhancing the robustness and adaptability of malware detection systems.

Chapter 8

Related Work

Within this section, we commence on a comprehensive exploration of related works, structured around three key criteria. Firstly, our inquiry extends into the diverse applications of Stackelberg security games across multifarious fields. Here, we aim to unravel the varied contexts in which these strategic games have been applied, shedding light on their adaptive and versatile nature in optimizing security approaches.

Following this, our attention turns to a detailed examination of previous studies in the specialized realm of hardware event-based malware detection. By delving into these investigations, we seek to discern the nuanced approaches, methodologies, and findings that have contributed to the evolution of this specific domain. This literature provides a foundation for understanding the intricacies and advancements within hardware event-based malware detection.

Concluding our exploration, we shift our focus to malware detection studies that diverge from the reliance on hardware events. This encompasses a broader spectrum of methodologies, showcasing the diversity in approaches to identifying and mitigating malware threats. By considering alternative methods, we aim to capture the comprehensive landscape of contemporary research in malware detection, offering insights into the efficacy and innovation present in varied detection strategies.

8.1 Stackelberg security games

Stackelberg games emerge as a cornerstone in the realm of fortifying defense strategies across a spectrum of security domains. Their significance lies in providing profound insights

that empower security authorities to fine-tune resource allocation and decision-making processes with a heightened level of strategic acumen and efficiency. By adopting a Stackelberg game framework, security entities gain a strategic advantage, allowing them to not only anticipate but also proactively shape the dynamics of security scenarios.

These games offer valuable insights, empowering security authorities to optimize resource allocation and decision-making.

One of the initial instances where Stackelberg security games found application was in the realm of infrastructure security. This is particularly relevant considering the worldwide concern for security at vital economic or political locations, driven by the looming threat of terrorism. In addition, limited security resources result in incomplete coverage, allowing adversaries to exploit patterns and plan attacks around existing patrols.

ARMOR describes the transition of advanced multi-agent algorithms into a deployed application called ARMOR (Assistant for Randomized Monitoring over Routes).The authors address the patrolling/monitoring problem by framing it as a Bayesian Stackelberg game, enabling appropriate weighting of actions and handling uncertainty about adversary types. Additionally they implement A mixed-initiative interface allowing occasional user adjustments or overrides based on local constraints. ARMOR has been operational at the Los Angeles International Airport (LAX) since August 2007, randomizing checkpoints on airport roadways and canine patrol routes within terminals.showcasing the practical application of Stackelberg models [69].

Beyond traditional security applications, the influence of Stackelberg games extends to unconventional domains, particularly within the environmental landscape. In this context, the strategic principles of Stackelberg games are harnessed to address challenges related to environmental protection, resource management, and ecological sustainability.

Poaching holds a significant threat to the conservation of crucial species and entire ecosystems. While foot patrols are commonly used in many countries to counter poaching, these patrols often do not optimize the limited resources available. To address this issue, prior research introduced PAWS [32] (Protection Assistant for Wildlife Security), an innovative application grounded in game theory, specifically "security games," designed to enhance the efficiency of patrolling resources. PAWS [32], a wildlife protection assistant system, has undergone extensive evaluation in Malaysia and the Queen Elizabeth National Park in Uganda.

In this study, the evolution journey, from initial tests in Africa in Spring 2014 to ongoing deployment in Southeast Asia, is outlined, along with plans for future global deployment. Collaborating closely with NGOs Panthera and Rimba, and incorporating feedback from professional patrolling teams, has been integral to refining and enhancing PAWS.

Technical advancements facilitating PAWS's regular deployment include addressing complex topographic features, handling uncertainties in species distribution, ensuring scalability for patrolling expansive conservation areas, and managing intricate patrol scheduling constraints. Elevation information and land features are incorporated using a hierarchical modeling approach to construct a virtual "street map" of the conservation area. This street map facilitates efficient scaling while offering detailed guidance, connecting the entire conservation area through easily navigable route segments like ridgelines, streams, and riverbanks – features commonly used by animals, poachers, and patrollers alike.

To address uncertainties and ensure scalability, a novel algorithm is introduced, integrating minimax regret for handling payoff uncertainty and the cutting plane framework for scalability. Additionally, the algorithm is equipped to handle constraints such as patrol time limits and starting and ending at the base camp.

In the ever-evolving landscape of cybersecurity, scholars and experts have delved into innovative approaches to fortify digital defenses. Notably, within this realm, researchers have actively investigated the integration of security game models to enhance two critical aspects of cybersecurity: deep packet inspection and the strategic deployment of honeypots.

Within the realm of cybersecurity, investigators have delved into the utilization of security game models. Deep packet inspection, a pivotal component in network security, involves scrutinizing the content of data packets traversing a network. By leveraging security game models, as elucidated by Van Vek et al. in their seminal work [82], cybersecurity practitioners can infuse a strategic and game-theoretic dimension into the analysis of these packets. This approach facilitates a more nuanced understanding of potential threats, enabling a proactive response to evolving cyber threats. By framing the interaction between defenders and attackers as a game.

In a parallel exploration, the application of security game models extends to the realm of honeypots, as expounded by Durkota et al. in their research [31]. Honeypots are decoy systems designed to attract and deceive malicious actors, allowing cybersecurity professionals to study their tactics and gather valuable intelligence. Security game models bring a strategic perspective to the utilization of honeypots, optimizing their deployment and operation. By viewing the interaction between attackers and honeypots as a game, researchers can devise optimal strategies for placing honeypots strategically within a network, maximizing their effectiveness in luring and identifying potential threats.

In summary, the incorporation of security game models into the cybersecurity domain represents a forward-thinking approach to addressing the dynamic and sophisticated nature of cyber threats. By applying these models to deep packet inspection and the deployment of honeypots, researchers aim to elevate the efficacy of security measures, fostering a more

proactive and strategic defense against the ever-evolving landscape of cyber threats.

Stackelberg games have been also used for malware detection [55]. In this study, macqueen et al. propose a novel method for determining an optimal randomization strategy across feasible sets of malware detection tools, considering costs for both attackers and defenders. They demonstrate that their best-responding randomization approach surpasses deterministic strategies and naive randomization methods. The defender's costs incorporate both objective factors (e.g., false positive rates from past studies and the VirusTotal dataset) and subjective factors (e.g., vulnerability severity ratings from the National Vulnerability Dataset). Notably, their model allows customization to reflect specific domain costs, preferences, and tool availability.

However in their approach the authors mention that estimating the attacker's utility is more challenging, and the approach accommodates partial information without requiring full knowledge. The defender can adopt a conservative approach by assigning the attacker a utility equal to the severity of each attack. In this study they explicitly consider strategic attacker behavior while optimizing the tradeoff between costs and benefits in the malware detection domain.

Our research represents a pioneering application of Stackelberg games within the realm of hardware cybersecurity. In essence, Stackelberg games, a strategic modeling framework, traditionally involve a leader and a follower, each making sequential decisions to optimize their respective objectives.

8.2 Hardware event-based malware detection

Extensive research efforts have been devoted to investigating malware detection through the analysis of hardware events. Among the notable contributions in this domain is the work of Sayadi et al. [75], where the application of ensemble methods takes center stage. In their comprehensive study, the authors advocate for an ensemble learning approach, a technique that involves the integration of multiple classifiers to enhance the overall predictive performance.

The notable feature of Sayadi et al.'s approach lies in the concerted training of all classifiers on the same subset of hardware events. This strategic choice emphasizes the importance of a unified and cohesive training foundation, wherein each classifier gains insights from a consistent set of hardware-related data points.

The utilization of ensemble methods in the context of hardware event-based malware

detection reflects a growing recognition of the effectiveness of collaborative and integrative approaches.

Moreover, researchers have delved into the targeted examination of distinct malware categories, exemplified by Alam et al. in their work [10]. In this study, the focus is directed towards ransomware, a malicious software type notorious for its disruptive capabilities. The researchers employ a sophisticated approach, leveraging a Deep Neural Network architecture coupled with Fast Fourier Transformation. This strategic combination not only showcases the versatility of modern machine learning techniques but also underscores the commitment to crafting an efficient and robust ransomware detection system. The utilization of advanced methodologies, as demonstrated in this research, contributes to the ongoing efforts to enhance cybersecurity measures by addressing the unique challenges posed by specific malware types, such as ransomware.

Another noteworthy contribution to the field is the research conducted by He et al. [40], which delves into the complexities of zero-day malware attacks. These particular cyber threats pose a significant challenge as they are unleashed before robust defense mechanisms have been established, leaving systems vulnerable to exploitation. By leveraging advanced algorithms and predictive models, they strive to develop proactive measures capable of identifying and mitigating the impact of such emerging threats.

Our approach diverges from the aforementioned works as we advocate for a novel strategy: employing multiple subsets of hardware events for training detection models. Unlike previous studies that predominantly rely on a fixed set of hardware events, our methodology introduces diversity and adaptability into the training process, recognizing the dynamic nature of cyber threats.

Furthermore, our commitment to designing a robust detection framework extends beyond conventional methodologies. We incorporate tools from game theory, introducing a strategic dimension to the development of our models. By leveraging insights from game theory, we aim to optimize decision-making processes and enhance the overall efficacy of our detection system. This distinctive combination of utilizing diverse hardware event subsets and integrating game theory principles underscores our approach to advancing malware detection capabilities.

8.3 Beyond hardware events

The identification of malware has extended to the analysis of software and system call events. This novel approach involves vigilant monitoring of system calls and thoughtfully

selecting the most insightful events to fortify a resilient malware detection framework, as elucidated in the study by Canzanese et al. [22].

Furthermore, certain studies delve into the intricacies of particular malware categories, such as Android gaming malware, as demonstrated in the research by Jaiswal et al. [44]. In their study, they conducted a comprehensive system call analysis of Android applications, shedding light on the behavior and patterns of such malware within the Android ecosystem.

In a distinct methodology presented by Nikolopoulos et al. [63], the authors advocate for a graph-based technique to delineate malware behavior. This innovative method entails constructing a directed acyclic graph from system-call traces to adeptly capture and illustrate malware activities.

In contrast, our research stands apart from these endeavors. We propose an orthogonal approach that introduces game-theoretic principles into software event-based malware detection systems. This integration opens up exciting avenues for future exploration and refinement in the realm of cybersecurity.

References

[1] CTuning: Benignware samples repository. https://ctuning.org/.

[2] Futuremark: Software development company that produces computer benchmark applications for home, business, and press use. https://www.3dmark.com.

[3] GonnaCry ransomware repository. https://github.com/tarcisio-marinho/GonnaCry.

[4] MalwareBazaar database. https://bazaar.abuse.ch.

[5] OpenMalware: Malware samples repository. https://openmalware.com.

[6] SPEC: Standardized benchmarks and tools. https://www.spec.org.

[7] VirusSample: Malware samples repository. https://www.virussamples.com.

[8] VirusShare: Malware samples repository. https://virusshare.com.

[9] VirusTotal: Malware samples repository and online scan engines. https://www.virustotal.com.

[10] Manaar Alam, Sarani Bhattacharya, Swastika Dutta, Sayan Sinha, Debdeep Mukhopadhyay, and Anupam Chattopadhyay. RATAFIA: Ransomware analysis using time and frequency informed autoencoders. In *Proceedings of the IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 218–227, 2019.

[11] Jennifer M Anderson, Lance M Berc, Jeffrey Dean, Sanjay Ghemawat, Monika R Henzinger, Shun-Tak A Leung, Richard L Sites, Mark T Vandevoorde, Carl A Waldspurger, and William E Weihl. Continuous profiling: Where have all the cycles gone? *ACM Transactions on Computer Systems (TOCS)*, 15(4):357–390, 1997.

[12] Wael Awada, Taghi M Khoshgoftaar, David Dittman, Randall Wald, and Amri Napolitano. A review of the stability of feature selection techniques for bioinformatics data. In *Proceedings of the 13th IEEE International Conference on Information Reuse & Integration (IRI)*, pages 356–363, 2012.

[13] Reza Azimi, Michael Stumm, and Robert W Wisniewski. Online performance analysis by statistical sampling of microprocessor performance counters. In *Proceedings of the 19th Annual International Conference on Supercomputing (ISCA)*, pages 101–110, 2005.

[14] Mohammad Bagher Bahador, Mahdi Abadi, and Asghar Tajoddin. HPCMalHunter: Behavioral malware detection using hardware performance counters and singular value decomposition. In *Proceedings of the 4th International Conference on Computer and Knowledge Engineering (ICCKE)*, pages 703–708, 2014.

[15] Maria-Florina Balcan, Avrim Blum, Nika Haghtalab, and Ariel D Procaccia. Commitment without regrets: Online learning in Stackelberg security games. In *Proceedings of the 16th ACM Conference on Economics and Computation (EC)*, pages 61–78, 2015.

[16] Kanad Basu, Prashanth Krishnamurthy, Farshad Khorrami, and Ramesh Karri. A theoretical study of hardware performance counters-based malware detection. *IEEE Transactions on Information Forensics and Security*, 15:512–525, 2020.

[17] Purnima Bholowalia and Arvind Kumar. EBK-means: A clustering technique based on elbow method and k-means in WSN. *International Journal of Computer Applications*, 105(9), 2014.

[18] Leo Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.

[19] M Breton, A Alj, and A Haurie. Sequential Stackelberg equilibria in two-person games. *Journal of Optimization Theory and Applications*, 59(1):71–97, 1988.

[20] Derek Bruening, Timothy Garnett, and Saman Amarasinghe. An infrastructure for adaptive dynamic optimization. In *Proceedings of the International Symposium on Code Generation and Optimization (CGO)*, pages 265–275, 2003.

[21] Derek L Bruening. *Efficient, Transparent, and Comprehensive Runtime Code Manipulation*. PhD thesis, Department of Electrical Engineering and Computer Science, 2004.

[22] Raymond Canzanese, Spiros Mancoridis, and Moshe Kam. System call-based detection of malicious processes. In *Proceedings of the 15th IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 119–124, 2015.

[23] John Cavazos, Grigori Fursin, Felix Agakov, Edwin Bonilla, Michael FP O'Boyle, and Olivier Temam. Rapidly selecting good compiler optimizations using performance counters. In *Proceedings of the International Symposium on Code Generation and Optimization (CGO)*, pages 185–197, 2007.

[24] Vincent Conitzer and Tuomas Sandholm. Computing the optimal strategy to commit to. In *Proceedings of the 7th ACM Conference on Electronic Commerce (EC)*, pages 82–90, 2006.

[25] Antoine Augustin Cournot. *Recherches sur les principes mathématiques de la théorie des richesses*. L. Hachette, 1838.

[26] 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 *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, pages 20–38, 2019.

[27] Richard A Davis, Pengfei Zang, and Tian Zheng. Sparse vector autoregressive modeling. *Journal of Computational and Graphical Statistics*, 25(4):1077–1096, 2016.

[28] Arnaldo Carvalho De Melo. The new Linux 'perf' tools. In *Slides from The Linux Kongress*, volume 18, pages 1–42, 2010.

[29] 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. In *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA)*, pages 559–570, 2013.

[30] Angus Dempster, François Petitjean, and Geoffrey I Webb. ROCKET: Exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, 34(5):1454–1495, 2020.

[31] Karel Durkota, Viliam Lisy, Christofer Kiekintveld, and Branislav Bosansky. Game-theoretic algorithms for optimal network security hardening using attack graphs. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1773–1774, 2015.

[32] Fei Fang, Thanh Nguyen, Rob Pickles, Wai Lam, Gopalasamy Clements, Bo An, Amandeep Singh, Milind Tambe, and Andrew Lemieux. Deploying paws: Field optimization of the protection assistant for wildlife security. In *Proceedings of the 16th AAAI Conference on Artificial Intelligence*, volume 30, pages 3966–3973, 2016.

[33] Wes Felter, Alexandre Ferreira, Ram Rajamony, and Juan Rubio. An updated performance comparison of virtual machines and linux containers. In *Proceedings of the IEEE International Symposium on Performance Aanalysis of Systems and Software (ISPASS)*, pages 171–172, 2015.

[34] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Sparse inverse covariance estimation with the graphical LASSO. *Biostatistics*, 9(3):432–441, 2008.

[35] Gaddisa Olani Ganfure, Chun-Feng Wu, Yuan-Hao Chang, and Wei-Kuan Shih. DeepWare: Imaging performance counters with deep learning to detect ransomware. *IEEE Transactions on Computers*, 72(3):600–613, 2022.

[36] Sarah Gelper and Christophe Croux. Least angle regression for time series forecasting with many predictors. *FBE Research Report KBI_0801*, 2018.

[37] Kevin Immanuel Gubbi, Han Wang, Hossein Sayadi, and Houman Homayoun. Machine learning based malware detection for secure smart grids. In *Proceedings of the 11th International Conference on Renewable Energy Research and Application (ICRERA)*, pages 330–334, 2022.

[38] MR Guthaus, JS Ringenberg, D Ernst, TM Austin, T Mudge, and RB Brown. MiBench: A free, commercially representative embedded benchmark suite. In *Proceedings of the 4th Annual IEEE International Workshop on Workload Characterization (WWC)*, pages 3–14, 2001.

[39] Zhangying He, Hosein Mohammadi Makrani, Setareh Rafatirad, Houman Homayoun, and Hossein Sayadi. Breakthrough to adaptive and cost-aware hardware-assisted zero-day malware detection: A reinforcement learning-based approach. In *Proceedings of the 40th International Conference on Computer Design (ICCD)*, pages 231–238, 2022.

[40] Zhangying He, Tahereh Miari, Hosein Mohammadi Makrani, Mehrdad Aliasgari, Houman Homayoun, and Hossein Sayadi. When machine learning meets hardware cybersecurity: Delving into accurate zero-day malware detection. In *Proceedings of the 22nd International Symposium on Quality Electronic Design (ISQED)*, pages 85–90, 2021.

[41] Zhangying He, Amin Rezaei, Houman Homayoun, and Hossein Sayadi. Deep neural network and transfer learning for accurate hardware-based zero-day malware detection. In *Proceedings of the 22nd Great Lakes Symposium on VLSI (GLSVLSI)*, pages 27–32, 2022.

[42] Zhexue Huang. Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data mining and knowledge discovery*, 2(3):283–304, 1998.

[43] Manish Jain, Jason Tsai, James Pita, Christopher Kiekintveld, Shyamsunder Rathi, Milind Tambe, and Fernando Ordóòez. Software assistants for randomized patrol planning for the LAX Airport Police and the Federal Air Marshal Service. *Interfaces*, 40(4):267–290, 2010.

[44] Mayank Jaiswal, Yasir Malik, and Fehmi Jaafar. Android gaming malware detection using system call analysis. In *Proceedings of the 6th International Symposium on Digital Forensic and Security (ISDFS)*, pages 1–5. IEEE, 2018.

[45] Sai Praveen Kadiyala, Pranav Jadhav, Siew-Kei Lam, and Thambipillai Srikanthan. Hardware performance counter-based fine-grained malware detection. *ACM Transactions on Embedded Computing Systems (TECS)*, 19(5):1–17, 2020.

[46] Debarun Kar, Thanh H Nguyen, Fei Fang, Matthew Brown, Arunesh Sinha, Milind Tambe, and Albert Xin Jiang. Trends and applications in Stackelberg security games. In Tamer Başar and Georges Zaccour, editors, *Handbook of Dynamic Game Theory*, pages 1–47. 2017.

[47] Utkarsh Mahadeo Khaire and R Dhanalakshmi. Stability of feature selection algorithm: A review. *Journal of King Saud University-Computer and Information Sciences*, 34(4):1060–1073, 2022.

[48] Christopher Kiekintveld, Manish Jain, Jason Tsai, James Pita, Fernando Ordóñez, and Milind Tambe. Computing optimal randomized resource allocations for massive security games. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 689–696, 2009.

[49] Dhilung Kirat, Giovanni Vigna, and Christopher Kruegel. BareCloud: Bare-metal analysis-based evasive malware detection. In *Proceedings of the 23rd USENIX Security Symposium (USENIX Security)*, pages 287–301, 2014.

[50] Ludmila I Kuncheva. A stability index for feature selection. In *Proceedings of the 25th Conference on IASTED International Multi-Conference: Artificial Intelligence and Applications*, pages 390–395, 2007.

[51] Hojoon Lee, Hyungon Moon, Daehee Jang, Kihwan Kim, Jihoon Lee, Yunheung Paek, and Brent ByungHoon Kang. KI-Mon: A hardware-assisted event-triggered monitoring platform for mutable kernel object. In *Proceedings of the 22nd USENIX conference on Security (USENIX Security)*, pages 511–526, 2013.

[52] Tianhong Liu, Haikun Wei, Kanjian Zhang, and Weili Guo. Mutual information based feature selection for multivariate time series forecasting. In *Proceedings of the 35th Chinese Control Conference (CCC)*, pages 7110–7114, 2016.

[53] Markus Löning, Anthony Bagnall, Sajaysurya Ganesh, Viktor Kazakov, Jason Lines, and Franz J Király. sktime: A unified interface for machine learning with time series. *arXiv preprint arXiv:1909.07872*, 2019.

[54] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. Pin: Building customized program analysis tools with dynamic instrumentation. *Acm Sigplan Notices*, 40(6):190–200, 2005.

[55] Revan MacQueen, Natalie Bombardieri, James R Wright, and Karim Ali. Game-theoretic malware detection. *arXiv preprint arXiv:2012.00817*, 2020.

[56] Hosein Mohammadi Makrani, Zhangying He, Setareh Rafatirad, and Hossein Sayadi. Accelerated machine learning for on-device hardware-assisted cybersecurity in edge platforms. In *Proceedings of the 23rd International Symposium on Quality Electronic Design (ISQED)*, pages 77–83, 2022.

[57] Corey Malone, Mohamed Zahran, and Ramesh Karri. Are hardware performance counters a cost effective way for integrity checking of programs. In *Proceedings of the 6th ACM Workshop on Scalable Trusted Computing (STC)*, pages 71–76, 2011.

[58] Larry W. McVoy and Carl Staelin. LMbench: Portable tools for performance analysis. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, pages 279–294, 1996.

[59] Philip J Mucci, Shirley Browne, Christine Deane, and George Ho. PAPI: A portable interface to hardware performance counters. In *Proceedings of the Department of Defense HPCMP Users Group Conference*, volume 710, 1999.

[60] Hitoshi Nagasaka, Naoya Maruyama, Akira Nukada, Toshio Endo, and Satoshi Matsuoka. Statistical power modeling of gpu kernels using performance counters. In *Proceedings of the International Conference on Green Computing*, pages 115–122. IEEE, 2010.

[61] Nicholas Nethercote and Julian Seward. Valgrind: A framework for heavyweight dynamic binary instrumentation. *ACM Sigplan notices*, 42(6):89–100, 2007.

[62] Theoden I Netoff, Thomas L Carroll, Louis M Pecora, and Steven J Schiff. Detecting coupling in the presence of noise and nonlinearity. In Björn Schelter, Matthias Winterhalder, and Jens Timmer, editors, *Handbook of Time Series Analysis: Recent Theoretical Developments and Applications*, pages 265–282. 2006.

[63] Stavros D Nikolopoulos and Iosif Polenakis. A graph-based model for malware detection and classification using system-call groups. *Journal of Computer Virology and Hacking Techniques*, 13(1):29–46, 2017.

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

[65] Dmitry Pavlyuk. Random forest variable selection for sparse vector autoregressive models. In *Proceedings of the International Conference on Time Series and Forecasting*, pages 3–17, 2019.

[66] Dmitry Pavlyuk. fsMTS: Feature selection for multivariate time series. https://cran.r-project.org/web/packages/fsMTS/index.html, 2022.

[67] Barbara Pes. Ensemble feature selection for high-dimensional data: A stability analysis across multiple domains. *Neural Computing and Applications*, 32(10):5951–5973, 2020.

[68] Phillip E Pfeifer and Stuart Jay Deutrch. A three-stage iterative procedure for space-time modeling phillip. *Technometrics*, 22(1):35–47, 1980.

[69] James Pita, Manish Jain, Janusz Marecki, Fernando Ordóñez, Christopher Portway, Milind Tambe, Craig Western, Praveen Paruchuri, and Sarit Kraus. Deployed armor protection: The application of a game theoretic model for security at the los angeles international airport. In *Proceedings of the 7th international conference on Autonomous agents and multiagent systems (AAMAS)*, pages 125–132, 2008.

[70] Yvan Saeys, Thomas Abeel, and Yves Van de Peer. Robust feature selection using ensemble feature selection techniques. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases-Volume Part II*, pages 313–325, 2008.

[71] Hossein Sayadi, Yifeng Gao, Hosein Mohammadi Makrani, Tinoosh Mohsenin, Avesta Sasan, Setareh Rafatirad, Jessica Lin, and Houman Homayoun. Stealthminer: Specialized time series machine learning for run-time stealthy malware detection based on microarchitectural features. In *Proceedings of the 20th Great Lakes Symposium on VLSI (GLSVLSI)*, pages 175–180, 2020.

[72] Hossein Sayadi, Amir Houmansadr, Setareh Rafatirad, and Houman Homayoun. Comprehensive assessment of run-time hardware-supported malware detection using general and ensemble learning. In *Proceedings of the 15th ACM International Conference on Computing Frontiers (CF)*, pages 212–215, 2018.

[73] Hossein Sayadi, Hosein Mohammadi Makrani, Sai Manoj Pudukotai Dinakarrao, Tinoosh Mohsenin, Avesta Sasan, Setareh Rafatirad, and Houman Homayoun. 2Smart: A two-stage machine learning-based approach for run-time specialized hardware-assisted malware detection. In *Proceedings of the 19th Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 728–733, 2019.

[74] Hossein Sayadi, Hosein Mohammadi Makrani, Onkar Randive, Sai Manoj PD, Setareh Rafatirad, and Houman Homayoun. Customized machine learning-based hardware-assisted malware detection in embedded devices. In *Proceedings of the 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 1685–1688, 2018.

[75] Hossein Sayadi, Nisarg Patel, Avesta Sasan, Setareh Rafatirad, and Houman Homayoun. Ensemble learning for effective run-time hardware-based malware detection: A comprehensive analysis and classification. In *Proceedings of the 55th Annual Design Automation Conference (DAC)*, pages 1–6, 2018.

[76] Eric Shieh, Bo An, Rong Yang, Milind Tambe, Craig Baldwin, Joseph DiRenzo, Ben Maule, and Garrett Meyer. Protect: A deployed game theoretic system to protect the ports of the united states. In *Proceedings of the 11th international conference on autonomous agents and multiagent systems (AAMAS)*, pages 13–20, 2012.

[77] Karan Singh, Major Bhadauria, and Sally A McKee. Real time power estimation and thread scheduling via performance counters. *ACM SIGARCH Computer Architecture News*, 37(2):46–55, 2009.

[78] Arunesh Sinha, Fei Fang, Bo An, Christopher Kiekintveld, and Milind Tambe. Stackelberg security games: Looking beyond a decade of success. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 5494–5501, 2018.

[79] Jan Treibig, Georg Hager, and Gerhard Wellein. LIKWID: A lightweight performance-oriented tool suite for x86 multicore environments. In *Proceedings of the 39th International Conference on Parallel Processing Workshops (ICPP)*, pages 207–216, 2010.

[80] Kristal K Trejo, Julio B Clempner, and Alexander S Poznyak. Adapting strategies to dynamic environments in controllable Stackelberg security games. In *Proceedings of the 55th IEEE Conference on Decision and Control (CDC)*, pages 5484–5489, 2016.

[81] Leif Uhsadel, Andy Georges, and Ingrid Verbauwhede. Exploiting hardware performance counters. In *Proceedings of the 5th Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 59–67, 2008.

[82] Ondřej Vaněk, Zhengyu Yin, Manish Jain, Branislav Bošanskỳ, Milind Tambe, and Michal Pěchouček. Game-theoretic resource allocation for malicious packet detection in computer networks. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, volume 2, pages 905–912, 2012.

[83] Heinrich Freiherr von Stackelberg. *Marktform und Gleichgewicht*. Springer, 1934.

[84] Bernhard Von Stengel and Shmuel Zamir. Leadership with commitment to mixed strategies. Technical report, London School of Economics, 2004.

[85] Han Wang, Hossein Sayadi, Sai Manoj Pudukotai Dinakarrao, Avesta Sasan, Setareh Rafatirad, and Houman Homayoun. Enabling micro ai for securing edge devices at hardware level. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems (JETCAS)*, 11(4):803–815, 2021.

[86] Xueyang Wang, Sek Chai, Michael Isnardi, Sehoon Lim, and Ramesh Karri. Hardware performance counter-based malware identification and detection with adaptive compressive sensing. *ACM Transactions on Architecture and Code Optimization (TACO)*, 13(1):1–23, 2016.

[87] Vincent M Weaver and Sally A McKee. Can hardware performance counters be trusted? In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*, pages 141–150, 2008.

[88] Boyou Zhou, Anmol Gupta, Rasoul Jahanshahi, Manuel Egele, and Ajay Joshi. Hardware performance counters can detect malware: Myth or fact? In *Proceedings of the ACM Asia Conference on Computer and Communications Security (ASIA-CCS)*, pages 457–468, 2018.

APPENDICES

.1 Appendix

In this section, the subsequent two tables are presented for the comparison of previous studies on the detection of malware related to hardware events. Table 1 illustrates the methods for feature selection and machine learning classifiers employed in earlier studies. Following that, Table 2 outlines the hardware events utilized in various studies.

We have used abbreviations in Table 1, including Not Clear (NC), Principle Component Analysis (PCA), Decision Trees (DT), Convolutional Neural Network (CNN), Random Forest (RF), Neural Networks (NN), Deep Neural Network (DNN), K-Nearest Neighbour(KNN), Long Short-Term Memory (LSTM), Support Vector Machine (SVM), Multilayer Perceptron (MLP), Stochastic Gradient Descent (SGD), Control Flow Graphs (CFG), Correlation Attribute Evaluation (CAE), Mutual Information (MI), Logistic Regression (LR), Recursive Feature Elimination (RFE), Gaussian Naive Bayes(GNB), ExtraTreeClassifier (ExtraTree), RidgeClassifier (Ridge), Repeated Incremental Pruning to Produce Error Reduction (JRip), Fully Convolutional Network (FCN), Sequential Minimal Optimization(SMO), One Rule (OneR), Repeated Incremental Pruning to Produce Error Reduction (RepTree)

Ref.	No. Benignware / Malware	FS Method	Classification Method
[35]	N/A / 515	M/S	CNN-based
[45]	293 / 322	PCA	DT, RF, NN, AdaBoost, KNN
[10]	N/A	N/A	LSTM-Autoencoder
[74]	N/A / 3000	PCA, AE	SVM, BN, MLP, DT
[88]	N/A / 962	PCA	DT, RF, NN, KNN, AdaBoost, NB
[29]	210 / 503	N/A	KNN, DT
[57]	N/A	M/S	CFG
[37]	>100	N/A	BN, JRip, OneR, REPTree, SGD, MLP, SMO
[41]	>5000	MI	RF, DT, GNB, LR, ExtraTree, Ridge, KNN, SVM, DNN, BaggedDT
[56]	200 / 200	CAE	BN, LR, MLP, PART, SMO
[85]	100 / 200	CAE	NB, LR, MLP, SGD, JRip, OneR, J48
[72]	>100	CAE	LR, BayesNet, MLP, J48, JRip, AdaBoost
[73]	3000	CAE , PCA	MLP, J48, JRip, OneR, Ensemble learning
[75]	>100	CAE	AdaBoost, Bagging
[71]	3500	CAE , PCA	FCN-based
[40]	>5000	RFE	RF, DT, GNB, SGD, LR, ExtraTree, AdaBoost
[39]	>5000	RFE and MI	JRip, J48, OneR, MLP, LR, RepTree

Table 1: Evaluation of previous research on malware detection using hardware events.

Feature	Used in	tag
Branches	[45, 10, 74, 57, 56, 85, 72, 73, 75, 71, 40]	Branch
Branch-misses	[35, 10, 74, 73, 71]	Branch
Branch-loads	[37, 72, 75]	Branch
Branch-reference	[35]	Branch
Instructions	[35, 45, 10, 57]	Ins
Cache-references	[35, 10, 74, 73, 71]	Cache
Cache-misses	[35, 10, 72]	Cache
L1-dcache-store	[45, 56, 85, 72, 39]	L1-dcache
L1-dcache-loads	[56, 85, 39]	L1-dcache
L1-dcache-load-misses	[41]	L1-dcache
L1-icache-load-misses	[45, 56]	L1-icache
LLC-prefetch-misses	[72]	LLC
Node-stores	[74, 73, 71, 39]	Node
Node-loads	[41, 40, 39]	Node
ITLB-loads	[56, 85]	ITLB
ITLB-load-misses	[37, 56, 72, 75]	ITLB
dTLB-load-misses	[37, 72, 75]	dTLB
dTLB-stores	[40]	dTLB
dTLB-store-misses	[37, 72]	dTLB
ls-mab-alloc-stores	[41]	Mem
cycles -ct	[40]	Cyc

Table 2: Hardware events employed in prior studies.

Feature	**tag**
Branches	Branch
Branch-misses	Branch
Branch-loads	Branch
Branch-load-misses	Branch
Cache-references	Cache
Cache-misses	Cache
Node-stores	Node
Node-loads	Node
L1-dcache-loads	L1-dcache
L1-dcache-load-misses	L1-dcache
L1-icache-load-misses	L1-icache
ls-mab-alloc-stores	Mem
ls-dc-accesses	dcache
ITLB-loads	ITLB
ITLB-load-misses	ITLB
dTLB-load-misses	dTLB
L1-dtlb-misses	L1-dTLB
L2-dtlb-misses	L2-dTLB
L2-itlb-misses	L2-ITLB
L2-cache-accesses-from-ic-misses	L2-dcache
L2-cache-accesses-from-dc-misses	L1-icache
ls-l1-d-tlb-miss-all	L1-dTLB
all-dc-accesses	dcache

Table 3: Hardware events employed in this study.