Ensemble Learning for Low-level Hardware-supported Malware Detection[*]

Khaled N. Khasawneh[1], Meltem Ozsoy[2], Caleb Donovick[3],
✉Nael Abu-Ghazaleh[1], and Dmitry Ponomarev[3]

[1] University of California, Riverside
{kkhas001,naelag}@ucr.edu
[2] Intel Corporation
meltem.ozsoy@intel.com
[3] Binghamton University
{cdonovi1,dima@cs}.binghamton.edu

Abstract. Recent work demonstrated hardware-based online malware detection using only low-level features. This detector is envisioned as a first line of defense that prioritizes the application of more expensive and more accurate software detectors. Critical to such a framework is the detection performance of the hardware detector. In this paper, we explore the use of both specialized detectors and ensemble learning techniques to improve performance of the hardware detector. The proposed detectors reduce the false positive rate by more than half compared to a single detector, while increasing the detection rate. We also contribute approximate metrics to quantify the detection overhead, and show that the proposed detectors achieve more than 11x reduction in overhead compared to a software only detector (1.87x compared to prior work), while improving detection time. Finally, we characterize the hardware complexity by extending an open core and synthesizing it on an FPGA platform, showing that the overhead is minimal.

1 Introduction

Malware continues to be a significant threat to computing systems at all scales. For example, AV TEST reports that 220,000 new malicious programs are registered to be examined every day and around 220 million total malware signatures are available in their malware zoo in the first quarter of 2014 [2]. Moreover, detection is becoming more difficult due to the increasing use of metamorphic and polymorphic malware [37]. Zero-day exploits also defy signature based static analysis since their signatures have not been yet encountered in the wild. This necessitates the use of dynamic detection techniques [9] that can detect the malicious behavior during execution, often based on the detection of anomalies, rather than signatures [4, 16]. However, the complexity and difficulty of continuous dynamic monitoring have traditionally limited its use.

[*] This research was partially supported by the US National Science Foundation grants CNS-1018496 and CNS-1422401.

Recent work has shown that malware can be differentiated from normal programs by classifying anomalies in low-level feature spaces such as hardware events collected by performance counters on modern CPUs [3, 6]. We call such features *sub-semantic* because they do not rely on a semantic model of the monitored program. In recent work, a classifier trained using supervised learning to differentiate malware from normal programs while the programs run was introduced [23]. To tolerate false positives, this system is envisioned as a first step in malware detection to prioritize which processes should be dynamically monitored using a more sophisticated but more expensive second level of protection.

The objective of this paper is to improve the classification accuracy of sub-semantic malware detection, allowing us to detect malware more successfully while reducing the burden on the second level of protection in response to false positives. We base our study on a recent malware data set [21]; more details are presented in Section 2. We pursue improved detection using two approaches. First, we explore, in Section 3, whether specialized detectors, each tuned to a specific type of malware, can more successfully classify that type of malware. We find that this is indeed the case, and identify the features that perform best for each specialized detector. Second, in Section 4, we explore how to combine multiple detectors, whether general or specialized, to improve the overall performance of the detection. We also evaluate the performance of the ensemble detectors in both offline and online detection.

To quantify the performance advantage from the improved detection, we develop metrics that translate detection performance to expected overhead in terms of the second level detector (Section 5). We discover that the detection performance of the online detection is substantially improved, reducing the false positives by over half for our best configurations, while also significantly improving the detection rate. This advantage translates to over 11x reduction in overhead of the two-level detection framework. We analyze the implications on the hardware complexity of the different configurations in Section 6. We compare this approach to related work in Section 7.

This paper makes the following contributions:

- We characterize how specialized detectors trained for specific malware types perform compared to a general detector and show that specialization has significant performamce advantages.
- We use ensemble learning to improve the performance of the hardware detector. However, combining specialized detectors is a non-classical application of ensemble learning, which requires new approaches. We also explore combining general detectors (with different features) as well as specialized and general detectors.
- We evaluate the hardware complexity of the proposed designs by extending the AO486 open core. We propose and evaluate some hardware optimizations.
- We define metrics for the two-level detection framework that translate detection performance to expected reduction in overhead, and time to detection.

2 Approach and Evaluation Methodology

Demme et al. [6] showed that malware programs can be classified effectively by the use of offline machine learning model applied to low-level features; in this case, the features were the performance counters of a modern ARM processor collected periodically. Subsequently, Ozsoy et al. [23] explored a number of low-level features, not only those available through performance counters, and built an *online* hardware-supported, low-complexity, malware detector. Such low-level features are called *sub-semantic* since they do not require knowledge of the semantics of the executing program. The online detection problem uses a time-series window based averaging to detect transient malware behavior. As detection is implemented in hardware, simple machine learning algorithms are used to avoid the overhead of complex algorithms. This work demonstrated that sub-semantic features can be used to detect malware in real-time (i.e., not only after the fact).

The goal of this work is to improve the effectiveness of online hardware-supported malware detection. Better machine learning classifiers can identify more malware with fewer false positives, substantially improving the performance of the malware detection system. To improve detection, we explore using specialized detectors for different malware types. We show that such specialized detectors are more effective than general detectors in classifying their malware type. Furthermore, we study different approaches for ensemble learning: combining the decisions of multiple detectors to achieve better classification. In this section, we overview the approach, and present the methodology and experimental details.

2.1 Programs Used for This Study

We collected samples of malware and normal programs to use in the training, cross validation and testing of our detectors. Since the malware programs that we use are Windows-based, we only used Windows programs for the regular program set. This set contains the SPEC 2006 benchmarks [12], Windows system binaries, and many popular applications such as Acrobat Reader, Notepad++, and Winrar. In total 554 programs were collected as the non-malware component of the data.

Our malware programs were collected from the MalwareDB malware set [21]. We selected only malware programs that were found between 2011-2014. The malware data sets have a total of 3,690 malware programs among them.

The group of regular and malware programs were all executed within a virtual machine running a 32-bit Windows 7 with the firewall and security services for Windows disabled, so that malware could perform its intended functionality. Moreover, we used the Pin instrumentation tool [5] to gather the dynamic traces of programs as they were executed. Each trace was collected for a duration of 5,000 system calls or 15 million committed instructions, whichever is first.

The malware data set consists of five types of malware: Backdoors, Password Stealers (PWS), Rogues, Trojans, and Worms. The malware groups and the regular programs were divided into three sets; training (60%), testing (20%) and cross-validation (20%). Table 1 shows the content of these sets.

We note that both the number of programs and the duration of the profiling of each program is limited by the computational overhead; since we are collecting dynamic profiling information through Pin [5] within a virtual machine, collection requires several weeks of execution on a small cluster, and produces several terabytes of compressed profiling data. Training and testing is also extremely computationally intensive. This dataset is sufficiently large to establish the feasibility and provide a reasonable evaluation of the proposed approach.

	Total	**Traning**	**Testing**	**Cross Validation**
Backdoor	815	489	163	163
Rogue	685	411	137	137
PWS	558	335	111	111
Trojan	1123	673	225	225
Worm	473	283	95	95
Regular	554	332	111	111

Table 1: Data set breakdown

2.2 Feature Selection

There are numerous features present at the architecture/hardware level that could be used. We use the same features as Ozsoy et al. [23], to enable direct comparison of ensemble learning against a single detector. For completeness, we describe the rationale behind these features:

- **Instruction mix features**: these are features that are derived from the types and/or frequencies of executed opcodes.We considered four features based on opcodes. Feature INS1 tracks the frequency of opcode occurrence in each of the x86 instruction categories. INS3 is a binary version of INS1 that tracks the presence of opcodes in each category. The top 35 opcodes with the largest difference (delta) in frequency between malware and regular programs were aggregated and used as feature (INS2). Finally, INS4 is a binary version of INS2 indicating opcode presence for the 35 largest difference opcodes.
- **Memory reference patterns**: these are features based on memory addresses used by the program. The first feature we consider in this group is MEM1, which keeps track of the memory reference distance in quantized bins (i.e., creates a histogram of the memory reference distance). The second feature we consider (MEM2) is a simpler form of MEM1 that tracks the presence of a load/store in each of the distance bins.
- **Architectural events**: features based on architectural events such as cache misses and branch predictions. The features collected were: total number of memory reads, memory writes, unaligned memory accesses, immediate branches and taken branches. This feature is called ARCH in the remainder of the paper.

The features were collected once every 10,000 committed instructions of the program, consistent with the methodology used by earlier works that use this approach [6, 23]. These prior studies demonstrated that classification at this frequency effectively balances complexity and detection accuracy for offline [6] and online [23] detection. For each program we maintained a sequence of these feature vectors collected every 10K instructions, labeled as either malware or normal.

3 Characterizing Performance of Specialized Detectors

In this section, we introduce the idea of *specialized detectors*: those that are trained to identify a single type of malware. First, we explore whether such detectors can outperform *general detectors*, which are detectors trained to classify any type of malware. If indeed they outperform general detectors, we then explore how to use such detectors to improve the overall detection of the system.

We separate our malware sets into types based on Microsoft Malware Protection Center classification [19].We use logistic regression for all our experiments because of the ease of hardware implementation [13]. In particular, the collected feature data for programs and malware is used to train logistic regression detectors. We pick the threshold for the output of the detector, which is used to separate a malware from a regular program, such that it maximizes the sum of the sensitivity and specificity. For each detector in this paper, we present the threshold value used.

Training General Detectors A general detector should be able to detect all types of malware programs. Therefore, a general detector is trained using a data set that encompasses all types of malware programs, against another set with regular programs. We trained seven general detectors, one for each of the feature vectors we considered.

Training Specialized Detectors The specialized detectors are designed to detect a specific type of malware relative to the regular programs. Therefore, the specialized detectors were trained only with malware that matches the detector type, as well as regular programs, so that it would have a better model for detecting the type of malware it is specialized for. For example, the Backdoors detector is trained to classify Backdoors from regular programs only. We chose this approach rather than also attempting to classify malware types from each other because false positives among malware types are not important for our goals. Moreover, types of malware may share features that regular programs do not have and thus classifying them from each other makes classification against regular programs less effective.

Each specialized detector was trained using a data set that includes regular programs and the malware type that the specialized detector is built for. On the other hand, the general detectors were trained using all the training data sets that we used for the specialized detectors combined plus the regular programs. In experiments that evaluate specialized detectors, the testing set (used for all detectors in such experiments including general detectors) consists only of normal programs and the specialized detector malware type. The reasoning for this choice is that we do not care about the performance of the specialized detector on other types of malware; if included these malware types would add noise to the results.

3.1 Specialized Detectors: Is There an Opportunity?

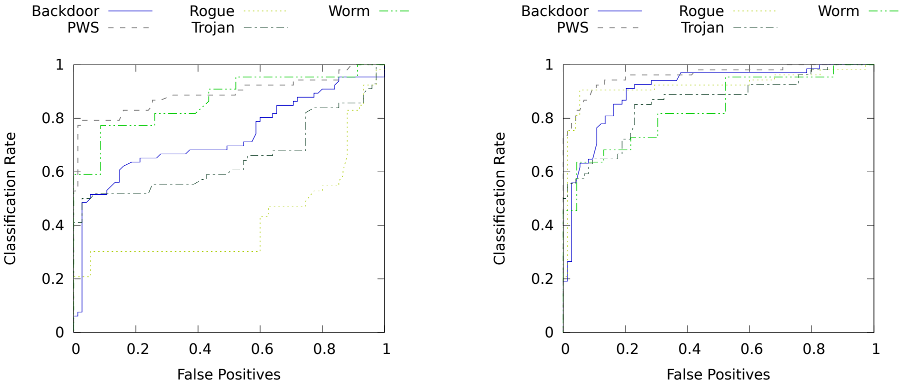
Next, we investigate whether specialized detectors outperform general detectors when tested against the malware type they were trained for. Intuitively, each malware type has different behavior allowing a specialized detector to more effectively carry out classification. Moreover, the detectors in this section were evaluated using the offline detection approach explained in Section 4.3.

General vs. Specialized Detectors We built specialized detectors for five types of malware which are Backdoor, PWS, Rogue, Trojan and Worm. Each of the seven general detectors' performance was compared against the performance of each specialized detectors in detecting the specific malware type for which the specialized detector was trained. Moreover, each comparison between specialized and general detectors used the same testing set for both of the detectors. The testing set includes regular programs and the malware type that the specialized detector was designed for.

Figures 1a, 1b show the Receiver Operating Characteristic (ROC) curves separated by type of malware using the general detector and the specialized detectors which were built using MEM1 features vector. Table 2 shows the Area Under the Curve (AUC) values for the ROC curves that resulted from all the comparisons between the general and specialized detectors in each feature vector. The ROC curves represent the classification rate (Sensitivity) as a function of false positives (100-Specificity) for different threshold values between 0 and 1. We found that in the majority of the comparisons, the specialized detectors indeed perform better than or equal to the general detector.

There were some cases where the general detector outperforms the specialized detectors for some features. We believe this behavior occurs because the general detector is trained using a larger data set than the specialized detector (it includes the other malware types). There are only a limited number of cases where the generalized detector outperforms the specialized ones for a specific feature. In most of these cases, the detection performance is poor, indicating that the feature is not particularly useful for classifying the given malware type.

Estimating the opportunity from deploying specialized detectors compared to general detectors is important since it gives an intuition of the best performance that could be reached using the specialized detectors. Thus, we compared the performance of the best performing general detector against the best specialized detector for each type of malware. Figure 2a shows the ROC curves of the INS4 general detector (best performing general detector) while Figure 2b shows the ROC curves for the best specialized detectors. In most cases, the specialized detectors outperform the general detector, sometimes significantly. Table 3



(a) General

(b) Specialized

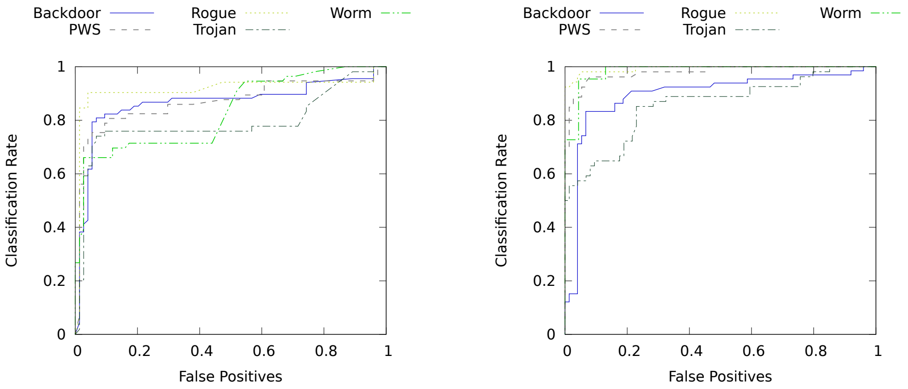
Fig. 1: MEM1 detectors performance

		Backdoor	PWS	Rogue	Trojan	Worm
INS1	**General**	0.713	0.909	0.949	0.715	0.705
	Specialized	0.715	0.892	0.962	0.727	0.819
INS2	**General**	0.905	0.946	0.993	0.768	0.810
	Specialized	0.895	0.954	0.976	0.782	0.984
INS3	**General**	0.837	0.909	0.924	0.527	0.761
	Specialized	0.840	0.888	0.991	0.808	0.852
INS4	**General**	0.866	0.868	0.914	0.788	0.830
	Specialized	0.891	0.941	0.993	0.798	0.869
MEM1	**General**	0.729	0.893	0.424	0.650	0.868
	Specialized	0.868	0.961	0.921	0.867	0.871
MEM2	**General**	0.833	0.947	0.761	0.866	0.903
	Specialized	0.843	0.979	0.931	0.868	0.871
ARCH	**General**	0.702	0.919	0.965	0.763	0.602
	Specialized	0.686	0.942	0.970	0.795	0.560

Table 2: AUC values for all general and specialized detectors

demonstrates this observation by showing that the average improvement opportunity using the AUC values is about 0.0904, improving the AUC by more than 10%. Although the improvement may appear to be modest, it has a substantial impact on performance. For example, the improvement in Rogue detection, 8% in the AUC, translates to a 4x reduction in overhead according to the work metric we define in Section 5.2).

These results make it clear that specialized detectors are more successful than general detectors in classifying malware. However, it is not clear why different features are more successful in detecting different classes of malware, or indeed why classification is at all possible in this subsemantic feature space. To attempt to answer this question, we examined the weights in the Θ vector of the logistic regression ARCH feature specialized detector for Rogue and Worm respectively. This feature obtains 0.97 AUC for Rogue but only 0.56 for Worm (see Table 2). We find that the Rogue classifier discovered that the number of branches in Rogue where significantly less than normal programs while the number of mis-



(a) Best general detector (INS4)

(b) Best specialized detector

Fig. 2: Opportunity size: best specialized vs. best general detector

	General	**Specialized**	**Difference**
Backdoor	0.8662	0.8956	0.0294
PWS	0.8684	0.9795	0.1111
Rogue	0.9149	0.9937	0.0788
Trojan	0.7887	0.8676	0.0789
Worm	0.8305	0.9842	0.1537
Average	0.8537	0.9441	0.0904

Table 3: Improvement opportunity: Area Under Curve

aligned memory addresses were significantly higher. In contrast, Worm weights were very low for all ARCH vector elements, indicating that Worms behaved similar to normal programs in terms of all architectural features. Explaining the fundamental reasons behind these differences in behavior is a topic of future research.

4 Malware Detection Using Ensemble Learning

Starting from a set of general detectors (one per feature) and a set of specialized detectors, our next goal is to explore how to compose these detectors to improve overall detection; such composite detectors are called ensemble detectors [8]. A decision function is used to combined the results of the base detectors into a final decision. Figure 3 illustrates the combined detector components and overall operation.

The general technique of combining multiple detectors is called *ensemble learning*; the classic type considers combining multiple independent detectors which are trained to classify the same phenomena [8]. For example, for malware detection, all the general detectors were designed to detect any type of malware.

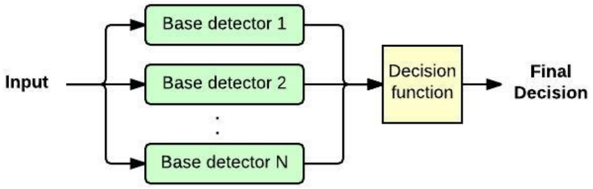


Fig. 3: Combined detector

Thus, ensemble learning techniques apply to the problem of combining their decisions directly.

On the other hand, for the specialized detectors, each detector is trained to classify a different phenomena (different type of malware); they are each answering a different classification question. Given that we do not know if a program contains malware, let alone the malware type, it is not clear how specialized detectors can be used as part of an overall detection solution. In particular, its unclear whether common ensemble learning techniques, which assume detectors that classify the same phenomena, would successfully combine the different specialized detectors.

In order to solve this problem, we evaluated different decision functions to combine the specialized detectors. We focused on combining techniques which use all the detectors independently in parallel to obtain the final output from the decision function. Since all the detectors are running in parallel, this approach speeds up the computation.

4.1 Decision Functions

We evaluated the following decision functions.

- **Or'ing**: If any of the detectors detects that a given input is a malware then the final detection result is a malware. This approach is likely to improve sensitivity, but result in a high number of false positives (reduce selectivity).
- **High confidence**: This decision function is an improved version of the or'ing decision function. In particular, the difference is that we select the specialized detector thresholds so that their output will be malware only when they are highly confident that the input is a malware program. Intuitively, specialized detectors are likely to have high confidence only when they encounter the malware type they are trained for.
- **Majority voting**: The final decision is the decision of the majority of the detectors. Thus, if most of them agreed that the program is a malware the final decision will be that it is a malware program.
- **Stacking (Stacked Generalization)**: in this approach, a number of first-level detectors are combined using a second-level detector (*meta-learner*) [33]).

> The key idea, is to train a second-level detector based on the output of first-level (base) detectors via cross-validation.

The stacking procedure operates as follows: we first collect the output of each of the base detectors to form a new data set using cross-validation. The collected data set would have every base detector decision for each instance in the cross-validation data set as well as the true classification (malware or regular program). In this step, it is critical to ensure that the base detectors are formed using a batch of the training data set that is different from the one used to form the new data set. The second step is to treat the new data set as a new problem, and employ a learning algorithm to solve it.

4.2 Ensemble Detectors

To aid with the selection of the base detectors to use within the ensemble detectors, we compare the set of general detectors to each other. Figure 4 shows the ROC graph that compares all the general detectors. We used a testing data set that includes the testing sets of all types of malware plus the regular programs testing set. The best performing general detectors use the INS4 feature vector; we used it as the baseline.

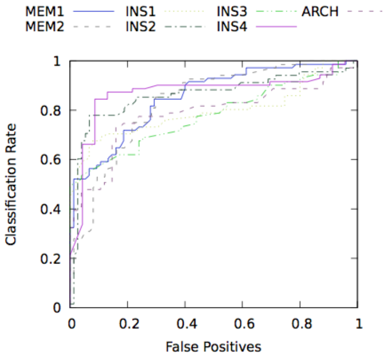


Fig. 4: General detectors comparison

We tested different decision functions and applied to them different selections of base detectors. An ROC curve based on a cross-validation data set was generated for each base detector to enable identification of the best threshold values for the base detectors. Subsequently, the closest point on the ROC curve to the upper left corner of the graph, which represents the maximum Sensitivity+Specificity, was selected since the sensitivity and specificity are equally important. However, for the High confidence decision function, the goal is to minimize the false positives. Therefore, we selected the highest sensitivity value achieving less than 3% false positive rate. Since the output of logistic regression classifier is a probability between 0 and 1, the threshold value is a fraction in this range.

The cross-validation data set used for the general detectors includes all types of malware as well as regular programs. However, for the specialized detector, it only includes the type of malware the specialized detector designed for and regular programs. We consider the following combinations of base detectors:

- *General ensemble* detector: combines only general detectors using classical ensemble learning. General ensemble detectors work best when diverse features are selected. Therefore, we use the best detector from each feature group (INS, MEM, and ARCH), which are INS4, MEM2, and ARCH respectively. Table 4 shows the threshold values for the selected base detectors which achieves the best detection (highest sum of sensitivity and specificity). Furthermore, the best threshold value is 0.781 for the stacking second-level detector.

	INS4	**MEM2**	**ARCH**
Best Threshold	0.812	0.599	0.668
High Confidence Threshold	0.893	0.927	0.885

Table 4: General ensemble base detectors threshold values

- *Specialized ensemble* detector: combines multiple specialized detectors. For each malware type, we used the best specialized detector. Thus, we selected the specialized detectors trained using MEM1 features vector for Trojans, MEM2 for PWS, INS4 for Rogue, and INS2 for both Backdoor and Worms. The selected threshold values of the selected detectors are shown in Table 5. In addition, the threshold value for the stacking second-level detector is 0.751.

	Backdoor	**PWS**	**Rogue**	**Trojan**	**Worm**
Best Threshold	0.765	0.777	0.707	0.562	0.818
High Confidence Threshold	0.879	0.89	0.886	0.902	0.867

Table 5: Specialized ensemble base detectors threshold values

- *Mixed ensemble* detector: combines one or more high performing specialized detectors with one general detector. The general detector allows the detection of other malware types unaccounted for by the base specialized detectors. In addition, this approach allows us to control the complexity of the ensemble (number of detectors) while taking advantage of the best specialized detectors. In our experiments, we used two specialized detectors for Worms and Rogue built using INS4 features vector because they performed significantly better than the general detector for detecting their type. The threshold values of the base detectors are shown in Table 6. The threshold value for the stacking second-level detector is 0.5.

	INS4	Rogue	Worm
Best Threshold	0.812	0.707	0.844
High Confidence Threshold	0.893	0.886	0.884

Table 6: Mixed ensemble base detectors threshold values

	Decision Function	Sensitivity	Specificity	Accuracy	Work Advantage	Time Advantage
Best General	–	82.4%	89.3%	85.1%	7.7	3.5
General Ensemble	*Or'ing*	99.1%	13.3%	65.0%	1.1	1.1
	High Confidence	80.7%	92.0%	85.1%	10.1	3.7
	Majority Voting	83.3%	92.1%	86.7%	10.5	4.1
	Stacking	80.7%	96.0%	86.8%	20.1	4.3
Specialized Ensemble	*Or'ing*	100%	5%	51.3%	1.1	1.1
	High Confidence	94.4%	94.7%	94.5%	17.8	9.2
	Stacking	95.8%	96.0%	95.9%	24	12.2
Mixed Ensemble	*Or'ing*	84.2%	70.6%	78.8%	2.9	2.2
	High Confidence	83.3%	81.3%	82.5%	4.5	2.8
	Stacking	80.7%	96.0%	86.7%	20.2	4.3

Table 7: Offline detection with different combining decision functions

4.3 Offline Detection Effectiveness

As discussed in Section 2.2, each program is represented as multiple feature instances collected as the program executes. To evaluate the offline detection of a detector, a decision for each vector in a program is made. If most of the decisions of that program records are malware, then the program is detected as malware. Otherwise, the program is detected as regular program.

Table 7 shows the sensitivity, specificity and accuracy for the different ensemble detectors using different combining decision functions. Also, it presents the work and time advantage, which represent the reduction in work and time to achieve the same detection performance as a software detector; these metrics are defined in Section 5.2. The specialized ensemble detector using stacking decision function outperforms all the other detectors with 95.8% sensitivity and only 4% false positive, which translates to 24x work advantage and 12.2x time advantage. The high confidence OR function also performs very well. This performance represents a substantial improvement over the baseline detector.

The Or'ing decision function results in poor specificity for most ensembles, since it results in a false positive whenever any detector encounters one. Majority voting was used only for general ensembles as it makes no sense to vote when the detectors are voting on different questions. Majority voting performed reasonably well for the general ensemble.

For the general ensemble detector, Stacking performs the best, slightly improving performance relative to the baseline detector. The majority voting was almost as accurate as stacking but results in more false positives. The mixed ensemble detector did not perform well; with stacking, it was able to significantly improve specificity but at low sensitivity.

4.4 Online Detection Effectiveness

The results thus far have investigated the detection success offline: i.e., given the full trace of program execution. In this section, we present a moving window approach to allow real-time classification of the malware. In particular, the features are collected for each 10,000 committed instructions, and classified using the detector. We keep track of the decision of the detector using an approximation of Exponential Moving Weighted Average. If during a window of time of 32 consecutive decisions, the decision of the detector reflects malware with an average that crosses a preset threshold, we classify the program as malware.

We evaluate candidate detectors in the online detection scenario. The performance as expected is slightly worse for online detection than offline detection, which benefits from the full program execution history. The overall accuracy, sensitivity, and specificity all decreased slightly with online detection.The result of the online detection performance are in Table 8.

	Sensitivity	**Specificity**	**Accuracy**
Best General	84.2%	86.6%	85.1%
General Ensemble (*Stacking*)	77.1%	94.6%	84.1%
Specialized Ensemble (*Stacking*)	92.9%	92.0%	92.3%
Mixed Ensemble (*Stacking*)	85.5%	90.1%	87.4%

Table 8: Online detection performance

5 Two-level Framework Performance

One of the issues of using a low-level detector such as the ensemble detector we are trying to implement, lacking the sophistication of a rich-semantic detector, is that false positives are difficult to eliminate. Thus, using the low-level detector on its own would result in a system where legitimate programs are sometimes identified as malware, substantially interfering with the operation of the system. Thus, we propose to use the low-level detector as the first level of a two-level detection (TLD) system. The low-level detector is always on, identifying processes that are likely to be malware to prioritize the second level. The second level could consist of a more sophisticated semantic detector, or even a protection mechanism, such as a Control Flow Integrity (CFI) monitor [39] or a Software Fault Isolation (SFI) [31] monitor, that prevents a suspicious process from overstepping its boundaries. The first level thus serves to prioritize the operation of the second level so that the available resources are directed at processes that are suspicious, rather than applied arbitrarily to all processes.

In this section, we analyze this model and derive approximate metrics to measure its performance advantage relative to a system consisting of a software protection only. Essentially, we want to evaluate how improvements in detection translate to run-time capabilities of the detection system. Without loss of generality, we assume that the second level consists of a software detector that can perfectly classify malware from normal programs, but the model can be adapted to consider other scenarios as well.

The first level uses sub-semantic features of the running programs to classify them. This classification may be binary (suspicious or not suspicious) or more continuous, providing a classification confidence value. In this analysis, we assume binary classification: if the hardware detector flags a program to be suspicious it will be added to a priority work list. The software detector scans processes in the high priority list first. A detector providing a suspicion index can provide more effective operation since the index can serve as the software monitoring priority.

5.1 Assumptions and Basic Models

In general, in machine learning the percentage of positive instances correctly classified as positives is called the Sensitivity (S). The percentage of correctly classified negative instances is called the Specificity (C). Applied to our system, S is a fraction of malware identified as such, while C is a fraction of regular programs identified correctly. Conversely, the misclassified malware is referred to as *False Negatives - FN*, while the misclassified normal programs are referred as *False Positives -FP*. For a classification algorithm to be effective, it is important to have high values of S and C.

We assume a discrete system where the arrival rate of processes is N with a fraction m of those being malware. We also assume that the effort that the system allocates to the software scanner is sufficient to scan a fraction e of the arriving processes (e ranges from 0 to 1). Note that we derive these metrics for a continuous system for convenience of derivation. Assuming a system with large N this approach should not affect the derived expected values.

In the base case a software scanner/detector scans a set of running programs that are equally likely to be malware. Thus, given a detection effort budget e, a corresponding fraction of the arriving programs can be covered. Increasing the detection budget will allow the scanner to evaluate more processes. Since every process has an equal probability of being malware, increasing the effort increases the detection percentage proportionately. Thus, the detection effectiveness (expected fraction of detected malware) is simply e.

Clearly, this is a first-order model in that they use simple average values for critical parameters such as the effort necessary to monitor a processes. However, we believe the metrics are simple and useful indicators to approximately quantify the computational performance advantage obtained in a TLD system as the detection performance changes.

5.2 Metrics to Assess Relative Performance of TLD

In contrast to the baseline model, the TLD works as follows. The hardware detector informs the system of suspected malware, which is used to create a priority list consisting of these processes. The size of this suspect list, $s_{suspect}$, as a fraction of the total number of processes is:

$$s_{suspect} = S \cdot m + (1 - C) \cdot (1 - m) \tag{1}$$

Intuitively, the suspect list size is the fraction of programs predicted to be malware. It consists of the fraction of malware that were successfully predicted to be malware ($S \cdot m$) and the fraction of normal programs erroneously predicted to be malware $(1 - C) \cdot (1 - m)$.

Work advantage Consider a case where the scanning effort e is limited to be no more than the size of the priority list. In this range, the advantage of the TLD can be derived as follows. Lets assume that the effort is $k \cdot s_{suspect}$ where k is some fraction between 0 and 1 inclusive. The expected fraction of detected malware for the baseline case is simply the effort, which is $k \cdot s_{suspect}$. In contrast, we know that S of the malware can be expected to be in the $s_{suspect}$ list and the success rate of the TLD is $k \cdot S$. Therefore, the advantage, W_{tld}, in detection rate for the combined detector in this range is:

$$W_{tld} = \frac{k \cdot S}{k \cdot s_{suspect}} = \frac{S}{S \cdot m + (1 - C) \cdot (1 - m)} \tag{2}$$

The advantage of the TLD is that the expected ratio of malware in the suspect list is higher than that in the general process list under the following conditions. It is interesting to note that when S+C=1, the advantage is 1 (i.e., both systems are the same); to get an advantage, S+C must be greater than 1. For example, for small m, if S=C=0.75, the advantage is 3 (the proposed system finds malware with one third of the effort of the baseline). If S=C=0.85 (in the range that our sub-semantic features are obtaining), the advantage grows to over 5.

Note that with a perfect hardware predictor (S=1, C=1), the advantage in the limit is $\frac{1}{m}$; thus, the highest advantage is during "peace-time" when m approaches 0. Under such a scenario, the advantage tends to $\frac{S}{1-C}$. However, as m increases, for imperfect detectors, the size of the priority list is affected in two ways: it gets larger because more malware processes are predicted to be malware (true positives), but it also gets smaller, because less processes are normal, and therefore less are erroneously predicted to be malware (false positives). For a scenario with a high level of attack (m tending to 1) there is no advantage to the system as all processes are malware and a priority list, even with perfect detection, does not improve on arbitrary scanning.

Detection success given a finite effort In this metric, we assume a finite amount of work, and compute the expected fraction of detected malware. Given enough resources to scan a fraction a of arriving processes, we attempt to determine the probability of detecting a particular infection.

We assume a strategy where the baseline detector scans the processes in arbitrary order (as before) while the TLD scans the suspect list first, and then, if there are additional resources, it scans the remaining processes in arbitrary order.

When $e <= s_{suspect}$, analysis similar to that above shows the detection advantage to be $(\frac{S}{s_{suspect}})$. When $e >= s_{suspect}$, then the detection probability can be computed as follows.

$$D_{tld} = S + (1 - S) \cdot \frac{e \cdot N - N \cdot s_{suspect}}{N \cdot (1 - s_{suspect})}. \tag{3}$$

The first part of the expression (S) means that if the suspect list is scanned, the probability of detecting a particular infection is S (that it is classified correctly and therefore is in the suspect list). However, if the malware is misclassified (1-S), malware could be detected if it is picked to be scanned given the remaining effort. The expression simplifies to:

$$D_{tld} = S + \frac{(1 - S) \cdot (e - s_{suspect})}{1 - s_{suspect}} \tag{4}$$

Note that the advantage in detection can be obtained by dividing D_{tld} by $D_{baseline}$ which is simply e.

Time to Detection Finally, we derive the expected time to detect a malware given an effort sufficient to scan all programs. In the baseline, the expected value of the time to detect for a given malware is $\frac{1}{2}$ of the scan time. In contrast, with the TLD, the expected detection time is:

$$T_{tld} = S \cdot \frac{s_{suspect}}{2} + (1 - S) \cdot (s_{suspect} + \frac{(1 - s_{suspect})}{2}), \tag{5}$$

The first part of the expression accounts for S of the malware which are correctly classified as malware. For these programs, the average detection time is half of the size of the suspect list. The remaining (1-S) malware which are misclassified have a detection time equal to the time to scan the suspect list (since that is scanned first), followed by half the time to scan the remaining processes. Simplifying the equation, we obtain:

$$T_{tld} = S \cdot \frac{s_{suspect}}{2} + (1 - S) \cdot (\frac{(1 + s_{suspect})}{2}), \tag{6}$$

Recalling that $T_{baseline} = \frac{1}{2}$, the advantage in detection time, which is the ratio $\frac{T_{tld}}{T_{baseline}}$ is:

$$T_{advantage} = S \cdot s_{suspect} + (1 - S) \cdot (1 + s_{suspect}), \tag{7}$$

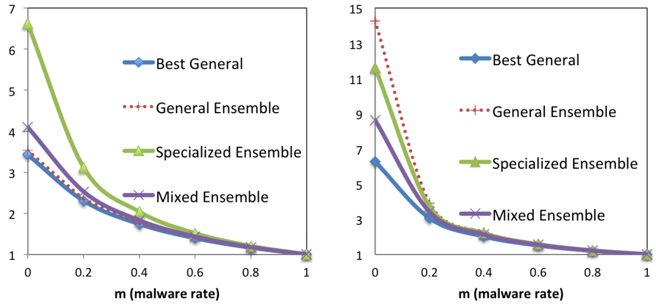
substituting for $s_{suspect}$ and simplifying, we obtain:

$$T_{advantage} = \frac{1}{1-(1-m)(C+S-1)} \tag{8}$$

The advantage again favors the TLD only when the sum of C and S exceeds 1 (the area above the 45 degree line in the ROC graph. Moreover, the advantage is higher when m is small (peace-time) and lower when m grows. When m tends to 0, if C+S = 1.5, malware is detected in half the time on average. If the detection is better (say C+S = 1.8), malware can be detected 5 times faster on average. We will use these metrics to evaluate the success of the TLD based on the Sensitivity and Specificity derived from the hardware classifiers that we implemented.

5.3 Evaluating Two Level Detection Overhead

Next, we use the metrics introduced in this section to analyze the performance and the time-to-detect advantage of the TLD systems based on the different hardware detectors we investigated. We selected the work and time advantage from these metrics to evaluate our detectors as a TLD systems; the first stage (hardware detector) will report the suspected malware programs to the second stage to be examined.



(a) Time advantage (b) Work advantage

Fig. 5: Time and work advantage as a function of malware rate

The time and work advantages for the online detectors are depicted in Figure 5 as the percentage of malware processes increases. The specialized ensemble detector reduced the average time of detection to 1/6.6 of the heavy-software only detector that is 2x faster than single general detector when the fraction of malware programs is low. This advantage was at 1/3.1 when malware intensity increased to the point where 20% of the programs are malware (m=0.2). In addition, the specialized ensemble detector has the best average time-to-detection.

The amount of work required for detection is improved by 11x by the specialized ensemble detector compared to using heavy-software detector only (1.87x compared to the best single detector). Although the general ensemble detector had a 14x improvement due to the reduction in the number of false positives, its detection rate is significantly lower than that of the specialized ensemble due to its lower sensitivity.

In Figure 6, we show the effort required by the online detectors to achieve 50% and 80% detection rate for the different detectors. Note that the effort increases with the percentage of malware. However, under common circumstances when the incidence of malware is low, the advantage of effective detection is most important. We see this for the specialized ensemble detector which is able to detect 80% of the malware while scanning less than 10% of the programs.

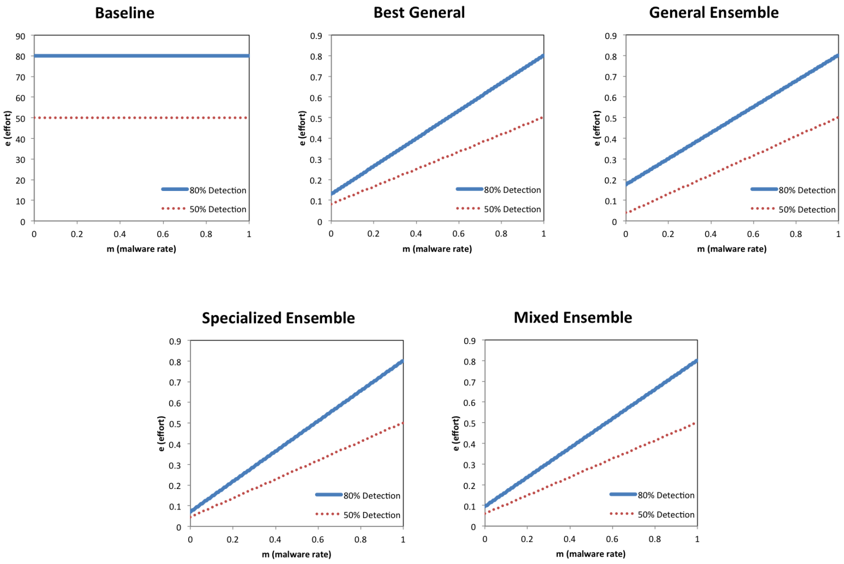


Fig. 6: Detection performance as a function of effort and malware rate

6 Hardware Implementation

In this section, we describe the hardware implementation of the ensemble detectors. During the execution of the program, instruction categories are collected at the feature collection unit (FCU), after the features are sent to prediction unit (PU) to create the prediction for 10K periods of committed instructions and finally online detection unit (ODU) creates a continuous signal with a value every 10,000 instructions for the executing program during runtime.

The FCU is implemented as an observer of the Reorder Buffer (ROB). ROB is a processor structure that keeps track of all in-flight instructions in their program order. The feature collection implementation differs with the type of feature. For example, for INS4, each ROB entry is augmented with instruction category information (6 bits). The feature vector used for classification is a bit vector with a bit for every instruction category. It is updated with each committed instruction by setting the corresponding category to 1.

We use logistic regression to implement the detectors due to its simplicity. The PU consists of different logistic regression units, one for each detector.

For the last part of the detection module, we use two counters in order to keep track of the malware and normal behavior. These counters are incremented at every 10K instructions accordingly and subtracted from each other to make the final decision.

The ensemble detector requires a minimal hardware investment. Taking up only 2.88% of logic cells on the core and using only 1.53% of the power. While the detector may be lightweight in terms of physical resources, the implementation required a 9.83% slow down of frequency. However, while this may seem high, the vast majority of this overhead comes from collecting the MEM feature vectors; when we do not collect this feature, the reduction in frequency was under 2%. If feature collection was pipelined over two cycles this cost be significantly reduced or eliminated. Moreover, we could use detectors that use simpler features to avoid using the MEM feature.

7 Related Work

Malware detection at the sub-semantic level was explored by several studies. Bilar et al. use the frequency of opcodes that a specific program uses [3]. Others use sequence signatures of the opcodes [28, 34]. Runwal et al. use similarity graphs of opcode sequences [27]. However, these works used offline analysis. In addition, Demme et al. use features based on performance counters [6] but did not explore online detection.

Ensemble learning can combine multiple base detectors to take a final decision for the improved accuracy [32]. The different base detectors are trained to solve the same problem. In contrast to traditional machine learning approaches that use the training data to learn one hypothesis, our ensemble approach learns a set of hypotheses and combines them.

Ensemble learning is attractive because of its generalization ability which is much powerful than using one learner [7]. In order for an ensemble detector to work, the base detectors have to be diverse; if the detectors are highly correlated, there is little additional value from combining them [26]. In this paper, the diversity is based on different features (general ensemble detector), data sets (mixed ensemble detector), or both (specialized ensemble detector).

The proposed specialized ensemble detector in this paper combines multiple specialized detectors and dynamically collects sub-semantic features to perform online detection. Researchers built ensemble malware detectors [1, 10, 11, 14, 17, 18,20,22,24,25,29,30,35,36,38], based on combining general detectors. Moreover, most of them used off-line analysis [1,10,14,25,29,30,35,36]. A few used dynamic analysis [11, 20, 24] and some used both static and dynamic analysis [17, 18, 22]. None of these works uses sub-semantic features or is targeted towards hardware implementation (which requires simpler machine learning algorithms). Specialized detectors were previously proposed [15] for use in malware classification (i.e., labeling malware). Labeling is used to classify collected malware using offline analysis. This is quite a different application of specialized detectors than the one we introduce in this paper.

8 Concluding Remarks

We build on Ozsoy et al. [23] work that uses low level features to provide a first line of defense to detect suspicious processes. This detector then prioritizes the effort of a heavy weight software detector to look only at programs that are deemed suspicious, forming a two-level detector (TLD). In this paper, we seek to improve the detection performance through ensemble learning to increase the efficiency of the TLD.

We start by evaluating whether specialized detectors can be more effectively classify one given class of malware. We found out that this is almost true for the features and malware types we considered. We then examined different ways of combining general and specialized detectors. We found that ensemble learning by combining general detectors provided limited advantage over a single general detector. However, combining specialized detectors can significantly improve the sensitivity, specificity, and accuracy of the detector.

We develop metrics to evaluate the performance advantage from better detection in the context of a TLD. Ensemble learning provides more than 11x reduction in the detection overhead with the specialized ensemble detector. This represents 1.87x improvement in performance (overhead) with respect to Ozsoy et al. [23] work previously introduced single detector. We implemented the proposed detector as part of an open core to study the hardware overhead. The hardware overhead was minimal: around 2.88% increase in area, 9.83% reduction in cycle time, and less than 1.35% increase in power. We believe that minor optimizations to the MEM feature collection circuitry could alleviate most of the cycle time reduction.

Acknowledgements

The authors would like to thank the anonymous reviewers and especially the shepherd for this paper, Tudor Dumitras, for their valuable feedback and suggestions, which significantly improved the paper.

References

1. Z. Aung and W. Zaw. Permission-based android malware detection. *International Journal of Scientific and Technology Research*, 2(3):228–234, 2013.

2. Malware Statistics, 2014. Available online: `http://www.av-test.org/en/statistics/malware/`.
3. D. Bilar. Opcode as predictor for malware. *International Journal of Electronic Security and Digital Forensic*, 2007.
4. M. Christodorescu, S. Jha, S. A. Seshia, D. Song, and R. E. Bryant. Semantics-aware malware detection. In *Proc. IEEE Symposium on Security and Privacy (SP)*, pages 32–46, 2005.
5. C.Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. Reddi, and K. Hazelwood. Pin: building customized program analysis tools with dynamic instrumentation. In *Proc. PLDI*, 2005.
6. J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, and S. Stolfo. On the feasibility of online malware detection with performance counters. In *Proc. Int. Symposium on Computer Architecture (ISCA)*, 2013.
7. T. G. Dietterich. Machine learning research: Four current directions, 1997.
8. T. G. Dietterich. Ensemble methods in machine learning. In *Proceedings of the First International Workshop on Multiple Classifier Systems*, MCS '00, pages 1–15, London, UK, UK, 2000. Springer-Verlag.
9. M. Egele, T. Scholte, E. Kirda, and C. Kruegel. A survey on automated dynamic malware-analysis techniques and tools. *ACM Computing Surveys*, 44(2), Mar. 2008.
10. M. Eskandari and S. Hashemi. Metamorphic malware detection using control flow graph mining. *International Journal of Computer Science and Network Security*, 11(12):1–6, 2011.
11. G. Folino, C. Pizzuti, and G. Spezzano. Gp ensemble for distributed intrusion detection systems. In *Pattern Recognition and Data Mining*, pages 54–62. 2005.
12. J. L. Henning. Spec cpu2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, 34(4):1–17, 2006.
13. D. W. Hosmer Jr. and S. Lemeshow. *Applied Logistic Regression*. John Wiley & Sons, 2004.
14. S. Hou, L. Chen, E. Tas, I. Demihovskiy, and Y. Ye. Cluster-oriented ensemble classifiers for intelligent malware detection. In *Semantic Computing (ICSC), 2015 IEEE International Conference on*, pages 189–196. IEEE, 2015.
15. J. Z. Kolter and M. A. Maloof. Learning to detect and classify malicious executables in the wild. *The Journal of Machine Learning Research*, 7:2721–2744, 2006.
16. C. Kruegel, W. Robertson, and G. Vigna. Detecting kernel-level rootkits through binary analysis. In *Proc. Annual Computer Security Applications Conference (ACSAC)*, pages 91–100, 2004.
17. J.-C. Liu, J.-F. Song, Q.-G. Miao, Y. Cao, and Y.-N. Quan. An ensemble cost-sensitive one-class learning framework for malware detection. *International Journal of Pattern Recognition and Artificial Intelligence*, page 1550018, 2012.
18. Y.-B. Lu, S.-C. Din, C.-F. Zheng, and B.-J. Gao. Using multi-feature and classifier ensembles to improve malware detection. *Journal of CCIT*, 39(2):57–72, 2010.
19. How Microsoft antimalware products identify malware and unwanted software. Available online: `https://www.microsoft.com/security/portal/mmpc/shared/objectivecriteria.aspx`.
20. P. Natani and D. Vidyarthi. Malware detection using api function frequency with ensemble based classifier. In *Security in Computing and Communications*, pages 378–388. Springer, 2013.
21. Malwaredb Website, 2015. Available online (last accessed, May 2015): `www.malwaredb.malekal.com`.

22. M. Ozdemir and I. Sogukpinar. An android malware detection architecture based on ensemble learning. *Transactions on Machine Learning and Artificial Intelligence*, 2(3):90–106, 2014.
23. M. Ozsoy, C. Donovick, I. Gorelik, N. Abu-Ghazaleh, and D. Ponomarev. Malware aware processors: A framework for efficient online malware detection. In *Proc. Int. Symposium on High Performance Computer Architecture (HPCA)*, 2015.
24. S. Peddabachigari, A. Abraham, C. Grosan, and J. Thomas. Modeling intrusion detection system using hybrid intelligent systems. *Journal of network and computer applications*, 30(1):114–132, 2007.
25. R. Perdisci, G. Gu, and W. Lee. Using an ensemble of one-class svm classifiers to harden payload-based anomaly detection systems. In *Proc. IEEE International Conference on Data Mining (ICDM)*, 2006.
26. J. R. Quinlan. Simplifying decision trees. *Int. J. Man-Mach. Stud.*, 27(3):221–234, Sept. 1987.
27. N. Runwal, R. M. Low, and M. Stamp. Opcode graph similarity and metamorphic detection. *J. Comput. Virol.*, 8(1-2):37–52, May 2012.
28. I. Santos, F. Brezo, J. Nieves, Y. K. Penya, B. Sanz, C. Laorden, and P. G. Bringas. Idea: Opcode-sequence-based malware detection. In *Engineering Secure Software and Systems*, pages 35–43. Springer, 2010.
29. R. K. Shahzad and N. Lavesson. Veto-based malware detection. In *Proc. IEEE Int. Conf. on Availability, Reliability and Security (ARES)*, pages 47–54, 2012.
30. S. Sheen, R. Anitha, and P. Sirisha. Malware detection by pruning of parallel ensembles using harmony search. *Pattern Recognition Letters*, 34(14), 2013.
31. R. Wahbe, S. Lucco, T. Anderson, and S. Graham. Efficient software-based fault isolation. In *ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, pages 203–216, New York, 1993. ACM Press.
32. I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.
33. D. H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.
34. G. Yan, N. Brown, and D. Kong. Exploring discriminatory features for automated malware classification. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 41–61. Springer, 2013.
35. Y. Ye, L. Chen, D. Wang, T. Li, Q. Jiang, and M. Zhao. Sbmds: an interpretable string based malware detection system using svm ensemble with bagging. *Journal in computer virology*, 5(4):283–293, 2009.
36. S. Y. Yerima, S. Sezer, and I. Muttik. High accuracy android malware detection using ensemble learning. *IET Information Security*, 2015.
37. I. You and K. Yim. Malware obfuscation techniques: A brief survey. In *Proc. International Conference on Broadband, Wireless Computing, Communication and Applications*, pages 297–300, 2010.
38. B. Zhang, J. Yin, J. Hao, D. Zhang, and S. Wang. Malicious codes detection based on ensemble learning. In *Lecture Notes in Computer Science*, volume 4610, pages 468–477. Springer, 2007.
39. M. Zhang and R. Sekar. Control flow integrity for cots binaries. In *Proc. 22nd Usenix Security Symposium*, 2013.