EnsembleHMD: Accurate Hardware Malware Detectors with Specialized Ensemble Classifiers

Khaled N. Khasawneh*, *Student Member, IEEE*, Meltem Ozsoy§, *Member, IEEE*, Caleb Donovick‡, *Student Member IEEE*, Nael Abu-Ghazaleh*, *Senior Member, IEEE*, Dmitry Ponomarev†, *Senior Member, IEEE*,

Abstract—Hardware-based malware detectors (HMDs) are a promising new approach to defend against malware. HMDs collect low-level architectural features and use them to classify malware from normal programs. With simple hardware support, HMDs can be always on, operating as a first line of defense that prioritizes the application of more expensive and more accurate software-detector. In this paper, our goal is to increase the accuracy of HMDs, to improve detection, and reduce overhead. We use specialized detectors targeted towards a specific type of malware to improve the detection of each type. Next, we use ensemble learning techniques to improve the overall accuracy by combining detectors. We explore detectors based on logistic regression (LR) and neural networks (NN). The proposed detectors reduce the false-positive rate by more than half compared to using a single detector, while increasing their sensitivity. We develop metrics to estimate detection overhead; the proposed detectors achieve more than 16.6x overhead reduction during online detection compared to an idealized software-only detector, with an 8x improvement in relative detection time. NN detectors outperform LR detectors in accuracy, overhead (by 40%), and time-to-detection of the hardware component (by 5x). Finally, we characterize the hardware complexity by extending an open-core and synthesizing it on an FPGA platform, showing that the overhead is minimal.

Index Terms—Malware detection, specialized detectors, ensemble learning, architecture, security.

1 Introduction

Computing systems at all scales face a significant threat from malware; for example, over 600 million malware sample were reported by AV TEST in their malware zoo, with over 120 million coming in 2016 [6]. Obfuscation and evasion techniques increase the difficulty of detecting malware after a machine is infected [51]. Zero-day exploits –novel exploits never seen before– defy signature based static analysis since their signatures have not been yet encountered in the wild. Thus, dynamic detection techniques [14] that can detect the malicious behavior during execution are needed [9], [24], to protect against such attacks. However, the difficulty and complexity of dynamic monitoring in software have traditionally limited its use.

Recent studies have shown that Hardware Malware Detectors (HMDs) that carry out anomaly detection in low-level feature spaces such as hardware events, can distinguish malware from normal programs [11], [7]. In our prior work, we introduced a hardware supported classifier trained using supervised learning that continuously monitors and differentiates malware from normal programs while the programs run [32], [33]. 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.

In this paper, we pursue approaches to enhance the classification accuracy of HMDs. Improving accuracy increases their ability to detect malware, and reduces the overhead that results from false positives. In particular, we explore two approaches: (1) Specialized detectors: we study, in Section 3, whether specialized detectors, each targeting a specific type of malware, can more successfully classify that type of malware. After confirming that specialized detectors perform better than general detectors, we identify the features that perform best for each specialized detector; and (2) Ensemble detection: in Section 4, we combine multiple detectors, general or specialized, to improve the overall performance of the detection. Combining specialized detectors is different from classical ensemble learning where multiple diverse detectors with identical goals are combined to enhance their accuracy. In particular, in our problem the specialized detectors each answers a different question in the form of: ”is the current program a malware of type X?” where X is the type of malware the detector is specialized for. Combining such detectors requires different forms of combination functions to produce the ensemble decision. We evaluate the performance of the ensemble detectors in both offline and online detection. We analyze the implications on the hardware complexity of the different configurations in Section 6.

The advantage of HMDs in a two-level detection system with a more accurate software detector is not directly measured by accuracy: if accuracy were the only metric, we are better off using the software detector alone. The advantage of HMDs result from reducing the overhead necessary for software detection and from prioritizing the efforts of a software detector. To better measure this advantage, we develop metrics that translate detection performance of HMDs to overhead and time-to-detection advantages of

*CSE and ECE Departments, University of California, Riverside, Email:{kkhas001,naelag}@ucr.edu †CS Department, Binghamton University ‡CS Department, Stanford University §Security and Privacy Lab., Intel Corp.

the whole system (Section 5). Our ensemble approaches substantially improve the detection of HMDs, reducing the false positives by over half for our best configurations, while also significantly improving the detection rate. As a result, we achieve over 16x reduction in overhead of the two-level detection framework compared to a software only detector. Compared to using a single HMD detector [32], [33], the ensemble detector achieves 2x reduction in overhead and 2.7x reduction in time to detection of the system during online detection.

The paper next conducts a longitudinal study to explore whether detectors trained on a set of malware continue to be effective over time as malware evolves (Section 7. We discover that the detection performance degrades substantially, motivating the need for a secure update facility to allow the detector configuration to be updated as malware evolves. We compare our approach to related work in Section 8. Finally, Section 9 presents some concluding remarks.

In summary, the 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 performance advantages.
- We use ensemble learning to improve the performance of the hardware detector. However, combining specialized detectors that answer different questions is a non-classical application of ensemble learning, which can challenge established approaches for combining ensemble decisions.
- We define metrics for the two-level detection framework that translate detection performance to expected reduction in overhead, and time to detection.
- We explore the use of both Logistic Regression (LR) and Neural Networks (NN) as the base classification algorithm for the detectors.
- We evaluate the hardware complexity of the proposed designs by extending the AO486 open core. We propose and evaluate some hardware optimizations to both the LR and NN implementations.
- We explore the question of whether detectors trained on an old generation of malware would continue to successfully detect malware as it evolves (i.e., from a more recent data set). We discover that the classification performance significantly deteriorates as malware evolves. In addition, we checked if detectors traind on recent malware would be able to detect old malware and the answer was no. These results highlights the need to continuously and securely adapt the learning configuration of the detector to track evolving malware.

2 Approach and Evaluation Methodology

We consider a system with a hardware malware detector (HMD) similar to those recently proposed in literature [11], [32], [20]. HMD exploit the fact that the computational footprint of malware differs from that of normal programs in low-level feature spaces. Detectors built using such features appear to be quite successful; Qualcomm announced the use of a similar technology in their Snapdragon processor [35]. Early studies relied on opcode mixes [7], [39], [48], [37]. More recently, Demme et al. [11] showed that malware programs can be classified effectively by the use of offline machine learning model applied to low-level features; in this case they used features available through hardware performance counters of the ARM processor collected periodically.

This paper improves on prior work by Ozsoy et al. [32] who built an *online* hardware-supported, low-complexity, malware detector. 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 low-level architectural features can be used to detect malware in real-time (i.e., not only after the fact).

In this study, our goal is to improve the effectiveness of online HMD. Improving the detection performance leads to more malware being detected with fewer false positives. We explore using specialized detectors for different malware types to improve detection. We show that specialized detectors are more effective than general detectors in classifying their malware type. Furthermore, we study different approaches for combining the decisions of multiple detectors to achieve better classification. In this section, we present some details of the methodology including the Data Set and the choice of features for classification.

2.1 Data Set

Our data set consists of 3,653 malware programs and 554 regular Windows programs (the malware samples that we use are Windows-based). This regular program set contains the SPEC 2006 benchmarks [17], Windows system binaries, and many popular applications such as Acrobat Reader, Notepad++, and Winrar. The malware programs were chosen from the MalwareDB malware set [30].

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. We observed that this desktop malware does not require user interaction to operate maliciously, in contrast to prior work that showed that mobile malware does not run correctly without user interaction [20]. We verified that a large sample (more than half) of our malware ran correctly by manually checking run-time behaviour. In fact, the intrusion detection monitoring systems on our network were tripped several times due to malware trying to search for and attack other machines. Eventually, we set up the environment in an independent subnet. However, for the regular programs, we manually interacted with them to trigger an expressive representation. The Pin instrumentation tool [10] was used to gather the dynamic traces of programs as they were executed. Each trace was collected after 150 system calls 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: (1) **Backdoors** which bypass the normal authentication of the system; (2) **Password Stealers (PWS)** which steals user credentials using a key-logger and sends them along with the visited website to the attacker; (3) **Rogues** which pretend to be an antivirus program and try to sell the victim its services; (4) **Trojans** which appear to be harmless programs but contain malicious code; and (5) **Worms** which attempt to spread to other machines using various methods. We selected only malware programs that were labeled as malware by Microsoft and used the Microsoft classification for their type [28]. Each malware set (corresponding to the malware type) and the regular programs set were randomly divided into three subsets; training (60%), testing (20%) and validation (20%) as shown in Table 1. These are typical ratios used in training classifiers. The training and testing sets were used to train and test the detectors respectively. The validation set was used for exploring the settings of training and detection.

We note that both the number of programs and the duration of the profiling of each program is limited by the computational and storage overheads; since we are collecting dynamic profiling information through Pin [10] 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	Training	Testing	Validation
Backdoor	815	489	163	163
Rogue	685	411	137	137
PWS	557	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

At the architecture/hardware level, there are many features that could be collected. To enable direct comparison of the proposed ensemble detector against a single detector, we use the same features used by Ozsoy et al. [32]. For completeness, we describe the rationale behind these features:

- **Instruction mix features**: collected based on 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. The top 35 opcodes with the largest difference (delta) in frequency between malware and regular programs were aggregated and used as feature (INS2). Finally, INS3 and INS4 are a binary version of INS1 and INS2 respectively; INS3 tracks the presence of opcodes in each category and INS4 indicating opcode presence for the 35 largest difference opcodes.
- **Memory reference patterns**: collected based on memory addresses used by the program. Feature MEM1 keeps track of the memory reference distance in quantized bins (i.e., creates a histogram of the memory reference distance). The binary version of MEM1 is feature MEM2 that tracks the presence of a load/store in each of the distance bins.
- **Architectural events**: collected based on architectural events. 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.

Consistent with the methodology used by earlier works [11], [32], we collected the features once every 10K committed instructions of the running program. The selected frequency (10K) effectively balances complexity and detection accuracy for offline [11] and online [32] detection. Thus, 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 *specialized detectors*: those that are trained to identify a specific type of malware. First, we investigate whether such detectors' performance exceeds that of *general detectors*, which are trained to classify any type of malware. After establishing that they do indeed outperform general detectors, we proceed by exploring how to use such detectors to improve the overall detection of the system.

We used two different classification algorithms in our experiments: (1) Logistic Regression (LR), which is a simple classification algorithm [18] that separates two classes using a linear boundary in the feature space. The motivation behind using LR is the ease of implementation in hardware; and (2) Neural Networks (NN) which is a network of perceptrons that can be trained to approximate a classification function that is generated from the training data. Note that a single perceptron in NN is equivalent to LR [2]; thus, NN is expected to outperform LR but with the cost of additional implementation complexity. The motivation of using NN is its more effective classificaiton due to its non-linear boundary. Additionally, NN can detect evasive malware when retrained, while LR cannot [21].

The collected feature data for programs and malware is used to train LR and NN 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 (recall) and specificity. Sensitivity is the proportion of malware that the system correctly identifies as malware while specificity is the proportion of regular programs that the system correctly identifies as regular programs [43]. For each detector in this paper, we present the threshold values to enable reproduction of our experiments.

Training General Detectors The general detectors are designed to detect any type of malware. 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. We identify the malware type and separate our malware sets into these types based on Microsoft Malware Protection Center classification [28]. 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.

3.1 Specialized Detectors: Is There an Opportunity?

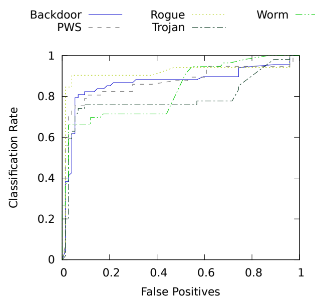
Intuitively, each malware type has different behaviour allowing specialized detectors to more accurately carry out classification. Thus, in this section, we explore this intuition and quantify the advantage obtained from specializing detectors.

We built specialized detectors for each type of malware we have in the data set (Backdoor, PWS, Rogue, Trojan and Worm). Next, we compared the performance of each of the seven general detectors against each of the specialized detectors performance with respect to classifying the specific malware type for which the specialized detector was trained. Each comparison between specialized and general detectors uses the same testing set for both of detectors. The testing set includes regular programs and the malware type that the specialized detector was designed for.

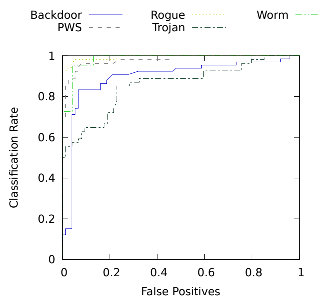
We compared the performance of the best performing LR and NN general detector against the best LR and NN specialized detector for each type of malware. Figure 1a shows the Receiver Operating Characteristic (ROC) curves of the LR INS4 general detector (best performing LR general detector) while Figure 1b shows the ROC curves for the best LR specialized detectors for each type of malware (MEM1 for Trojans, MEM2 for PWS, INS4 for Rogue, and INS2 for both Backdoor and Worms). The ROC curves represent the classification rate (i.e., Sensitivity) as a function of false positives (100-Specificity) for different threshold values between 0 and 1. In most cases, the specialized detectors outperform the general detector, sometimes significantly.

Figure 2 shows the average improvement of the Area Under the Curve (AUC) for each type of malware using the best LR general detector (INS4) and the best NN general detector (INS2). It also shows the performance of the best LR specialized detector, and the best NN specialized detector for each type of malware. Overall, the improvement opportunity is 0.0904 for using specialized LR detectors over general LR detectors and 0.06 for using specialized NN detectors over general NN detectors improving the AUC by more than 9% and 6% respectively. This improvement has a substantial impact on performance. For example, the improvement in Rogue detection, 8% in the AUC, translates to a 4x reduction in overhead needed for detection according to the work metric we define in Section 5.2). In addition, the specialized NN detectors outperform all other detectors.

Figure 3 shows the accuracy values for each of the previous AUC when picking the best operating point (maximum sensitivity+specificity). This results also supports the conclusion that specialized NN detectors outperform all other detectors.



(a) Best general detector (INS4)



(b) Best specialized detector

Fig. 1: Opportunity: best specialized vs. best general detector

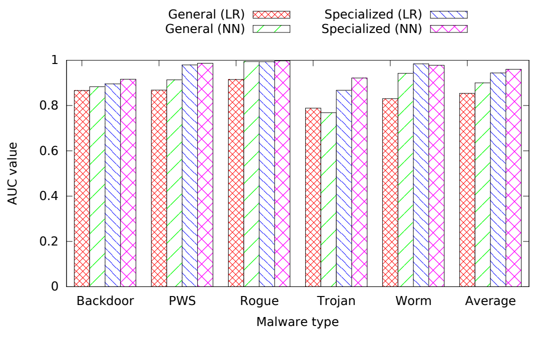


Fig. 2: AUC opportunity size

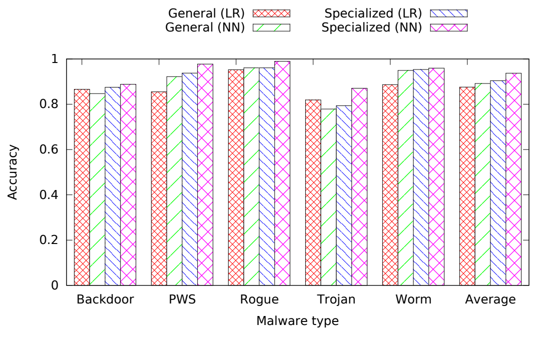


Fig. 3: Accuracy opportunity size

Its 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 low-level feature space. To attempt to answer this question, we examined the weights in the Θ vector of the LR ARCH feature specialized detector for Rogue and Worm respectively. This feature obtains 0.97 AUC for Rogue but only 0.56 for Worm. We find that the Rogue classifier discovered that the number of branches in Rogue where significantly less than normal programs while the number of misaligned 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.

4 Malware Detection Using Ensemble Learning

The next problem we consider is to how to use the specialized detectors to perform overall detection performance. The problem is challenging since we do not know the type of malware (or indeed if a program is malware) during classification. We start with a set of general detectors, each trained using each of our features, and a set of specialized detectors, each trained using one feature and for one malware type. The combining problem considers how to combine the decisions of multiple detectors (base detectors). To combine multiple base detectors, a decision function is used to fuse their result into a final decision. Figure 4 illustrates the combined detector components and overall operation.

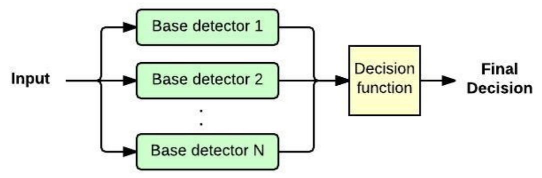


Fig. 4: Combined detector

The general technique of combining multiple detectors is called *ensemble learning*; the classic type combines multiple independent detectors that are each trained for the same classification problem (i.e, to classify the same phenomena) [13]. For example, for malware detection, all the general detectors were designed to detect any type of malware. Thus, ensemble learning techniques apply to the problem of combining their decisions directly.

When using specialized detectors, each detector is trained to classify a different phenomena (a 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 a practical detection solution. In addition, we do not know 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 evaluate 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

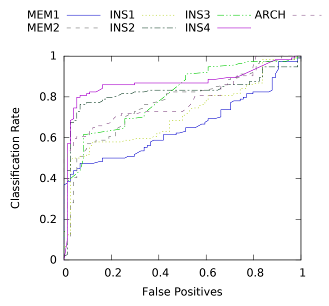
We evaluated the following decision functions.

- **Or'ing**: the final decision is malware if any of the base detectors detects the input as malware. This approach is likely to improve sensitivity, but result in a high number of false positives (reduce specificity).
- **High confidence**: the final decision is selected using the Or'ing decision function. However, in this decision function, 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 based on the decision of the majority of the base 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*) [47]). The key idea, is to train a second-level detector based on the output of first-level (base) detectors via validation data set. The final decision is the output of the second level detector.

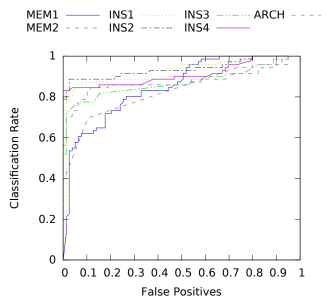
The stacking procedure operates as follows: we from a new data set from collecting the output of each of the base detectors using the validation set. The collected data set consists of every base detector decision for each instance in the validation data set as well as the true classification label (malware 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. Figures 5a and 5b show the ROC graph that compares all the LR and NN general detectors respectively. We used a test data set that includes the test sets of all types of malware added to the regular programs test set. The best performing LR general detector uses the INS4 feature vector and the best performing NN general detector uses the INS2 feature vector; we used them as the baseline detectors.



(a) Logistic regression



(b) Neural network

Fig. 5: General detectors comparison

We tested different decision functions and applied as input to them different selections of base detectors. An ROC curve based on a validation data set was generated for each base detector to enable identification of the best threshold

	INS	MEM	ARCH
Best Threshold (LR)	0.812	0.599	0.668
Best Threshold (NN)	0.421	0.581	0.622
High Confidence Threshold (LR)	0.893	0.927	0.885

TABLE 2: General ensemble base detectors threshold values

	Backdoor	PWS	Rogue	Trojan	Worm
Best Threshold (LR)	0.765	0.777	0.707	0.562	0.818
Best Threshold (NN)	0.473	0.337	0.180	0.760	0.269
High Confidence Threshold (LR)	0.879	0.890	0.886	0.902	0.867

TABLE 3: Specialized ensemble base detectors threshold values

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

TABLE 4: Logistic regression mixed ensemble base detectors threshold values

	INS2	Backdoor	PWS	Worm
Best Threshold	0.421	0.473	0.870	0.269

TABLE 5: Neural networks mixed ensemble base detectors threshold values

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 on the training set, was use to select the threshold since we considered the sensitivity and specificity to be 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.

The 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 for which the specialized detector is designed in addition to regular programs. We consider the following combinations of base detectors:

- *General ensemble* detectors: combine 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 for LR detectors and INS2, MEM2, and ARCH for NN detectors. Table 2 shows the threshold values for the selected LR base detectors which achieve the best detection (highest sum of sensitivity and specificity). The table also shows the threshold values for the selected NN base detectors. Finally, using the same process, we find the best threshold values for the stacking second-level detector to be 0.781 and 0.542 for the LR and NN stacking detectors respectively.
- *Specialized ensemble* detectors: these combine multiple specialized detectors. For each malware type, we used the best specialized detector. Thus, from LR detectors, we selected the detectors trained using MEM1 for Trojans, MEM2 for PWS, INS4 for Rogue, and INS2 for both Backdoor and Worms. On the other hand, for NN detectors, we selected MEM2 for both PWS and Trojans, and INS2 for the other malware types, with the threshold values shown in Table 3. In addition, the threshold value for the stacking second-level detector is 0.751 and 0.597 for LR and NN respectively.
- *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. This approach allows us to control the complexity of the ensemble (by limiting the number of specialized detectors) while taking advantage of the best specialized detectors. In our experiments, we used two LR specialized detectors for Worms and Rogue built using the INS4 features vector, because they performed significantly better than the LR general detector for detecting their type. We combine these with an INS4 general detector to build the LR general ensemble detector. For the NN general ensemble detector, we used three specialized detectors for Backdoor, PWS, and Worm built using INS2 features vector along with an INS2 general detector. The threshold values of the base detectors are shown in Table 4 and 5 for the LR general ensemble and the NN general ensemble respectively. The threshold value for the stacking second-level detector is 0.5 for LR general ensemble and 0.52 for NN general ensemble.

4.3 Offline Detection Effectiveness

As discussed in Section 2.2, each program have a feature instance collected for each 10K committed instructions during execution. To evaluate the offline detection performance of a detector, a decision for each instance is first evaluated. As a proof of concept, we consider programs where most of the decisions are malware to be malware; otherwise, the program is considered to be a regular program.

Table 6 and 7 show the sensitivity, specificity and accuracy for the different LR and NN ensemble detectors using different combining decision functions. It also presents the work and time advantage, which represent the reduction in work and time to achieve the same detection performance as a software detector; we define these metrics more precisely in Section 5. The specialized ensemble detector using the stacking decision function outperforms all other detectors built using the same classification method. The LR specialized ensemble achieves 95.8% sensitivity and only 4% false positive rate, which translates to a 24x work advantage and 12.2x time advantage compared to software only detector. On the other hand, the NN specialized ensemble resulted in 92.9% sensitivity and 0% false positive, which translates to unbounded work advantage and 14.1x time advantage compared to software only detector. Thus, NN specialized ensemble have unbounded work advantage, since they have no false positives on our dataset, and 15% faster detection than the LR specialized ensemble.

The Or'ing decision function results in poor specificity for most LR ensembles, since it results in a false positive whenever any detector encounters one. Majority voting was used only for LR general ensembles as it makes no

	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 6: Logistic regression offline detection with different combining decision functions

	Sensitivity	Specificity	Accuracy	Work Advantage	Time Advantage
Best General	88.7%	88.6%	88.3%	7.8	4.4
General Ensemble (*Stacking)*	83.1%	100%	91.7%	∞	5.9
Specialized Ensemble (*Stacking)*	92.9%	100%	96.5%	∞	14.1
Mixed Ensemble (*Stacking)*	80.2%	98.7%	89.7%	61.6	5.4

TABLE 7: Neural networks offline detection

sense to vote when the detectors are voting on different questions. Majority voting performed reasonably well for the LR general ensemble.

For the LR 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

Thus far, we have investigated the *offline* detection success: 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 following a design in our previous work [32]. 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. During a window of time of 32 consecutive decisions, if the decision of the detector reflects malware with an average that crosses a preset threshold, we classify the program as malware.

4.4.1 Online detection performance

	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: Logistic regression online detection performance

The result of the online detection performance are in Table 8 and 9. We evaluate candidate detectors in the online detection scenario. The performance for LR detectors is slightly worse for online detection than offline detection, which benefits from the full program execution history.

	Sensitivity	Specificity	Accuracy
Best General	89.2%	85.6%	86.9%
General Ensemble (Stacking)	91.6%	89.9%	90.6%
Specialized Ensemble (Stacking)	93.2%	94.4%	93.8%
Mixed Ensemble (Stacking)	94.4%	89.8%	91.7%

TABLE 9: Neural networks online detection performance

However, for the NN detectors the sensitivity increased, but the specificity and accuracy decreased. The NN specialized ensemble using the stacking decision function outperforms all other detectors by detecting 93.2% of the malware with only 5.6% of false alarms.

4.4.2 Online detection time

Next, we study the time for detecting a malware program in the hardware detector during execution. This time is defined as the number of decision windows (10K committed instructions) that a detector took since a malware started running to classify it as malware. Figure 6 shows the cumulative probability distribution of the detected malware programs as a function of the number of decision windows for both specialized ensemble detectors LR and NN. NN clearly outperforms LR in this metric. For example, NN detects 88% of malware in 500 periods or less, while LR only detects 7%, and over 95% of the malware in 1000 periods or less, while LR detects less than 60%. On average, NN detected malware 5x faster than LR. On a 3 GHz processor, assuming an Instruction Per Cycle (IPC) of 2, this translates to around 500 μ-seconds for NN compared to around 2.5 milliseconds for LR. Thus, NN detectors can alert the software detector more quickly, and limit the opportunity the malware has to do damage to the system.

5 Two-level Framework Performance

We envision our HMD to be used as a first level of a two-level detection (TLD) system. The advantage of this approach is that the second level can clean up false positives

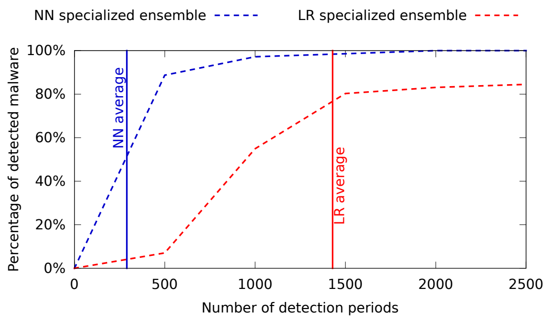


Fig. 6: Online detection time

that arise due to the fact that the HMD uses low level features and simple classifiers. The hardware 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 [55] or a Software Fault Isolation (SFI) [45] monitor, that prevents a suspicious process from overstepping its boundaries. An interesting possibility is to make the second level detector a cloud based system; if malware is found the results can be propagated to other subscribers in the system. 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.

One possible design point is to use a software only malware detector (i.e., make the second level always on). This detector is likely to be more accurate, but unfortunately, it also incurs high overhead. Thus, the advantage of the HMD is to prioritize the operation of the second level detector to lower its overhead, and reduce the detection time. Improving the detection accuracy of the hardware detector can substantially improve the overall system. Reducing false negatives will lead to the detection of more malware, while reducing false positives will reduce the overhead of the system by avoiding unnecessary invocations of the second level detector.

The performance of an HMD in terms of classification accuracy on its own does not reflect the advantage of using it within a TLD (detection overhead and time to detection). To be able to quantify these advantages, in this section we build a model of the two level detector and derive approximate metrics to measure its performance advantage relative to a system consisting of software protection only. Our goal is to evaluate how improvements in detection translate to run-time advantages for 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 low-level architecture 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 finer prioritization of the software classifier, further improving the detection advantage.

5.1 Assumptions and Basic Models

We call the percentage of positive instances correctly classified malware, Sensitivity (S) of the classifier. Similarly, we call the percentage of correctly classified normal programs the Specificity (C). Conversely, the misclassified malware is referred to as *False Negatives - FN*, while the misclassified normal programs are referredto as *False Positives -FP*. For a classification algorithm to be effective, it is important to have high values of both 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).

In the base case a software 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.

5.2 Metrics to Assess Relative Performance of TLD

In a TLD, 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 $(1 - C) \cdot (1 - m)$.

5.2.1 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 the system dedicates to detection 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 $k \cdot s_{suspect}$ (as discussed in the previous subsection). 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 the ratio of the detection of the TLD to the software baseline:

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

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 if the classifier is better than random. 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 (lower than the range that our 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 a period where the system has no malware 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.

5.2.2 *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.

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

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:

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

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

5.2.3 *Time to Detection*

Finally, we derive the expected time to detect a malware given an effort sufficient to scan all programs. Please note that this metric, which quantifies the time for detection in the overall system including the second level detector, differs from the detection time metric introduced in Section 4.4.2 which refers to the time of detection within the hardware detector. 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:

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

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:

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

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

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

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

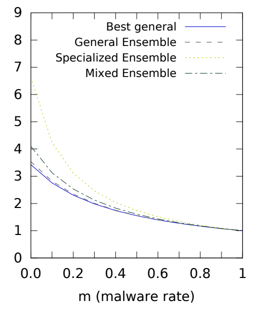
$$T_{advantage} = \frac{1}{1 - (1-m)(C + S - 1)} \qquad (8)$$

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.

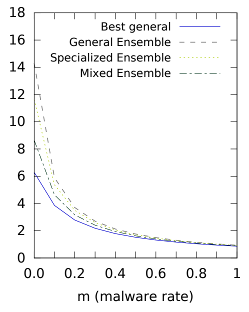
5.3 Evaluating Two Level Detection Overhead

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 also use them to understand the advantage obtained from increasing the detection accuracy using our ensemble techniques.

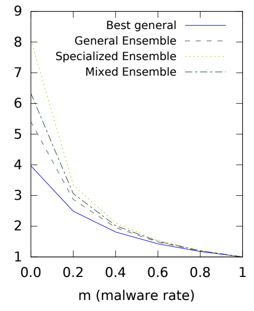
The time and work advantages for LR detectors are depicted in Figure 7a and 7b as the percentage of malware processes increases. The average time to detect for the LR specialized ensemble detector is 6.6x faster than the software only detector when the fraction of malware programs is low. This advantage drops as more malware is encountered in the system; for example, when 10% of the programs are malware (m=0.1), these advantages drop to 4.2x. We observe that the LR specialized ensemble detector has the best average time-to-detection among LR detectors. The amount of work required for detection is improved by 11x by the LR specialized ensemble detector compared to using the software detector only. Although the LR general ensemble detector had a 14x improvement due to the reduction in the number of false positives, its detection rate is significantly



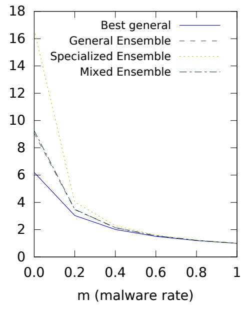
(a) LR time advantage



(b) LR work advantage



(c) NN time advantage



(d) NN work advantage

Fig. 7: Online detection time and work advantage as a function of malware rate

lower than that of the LR specialized ensemble due to its lower sensitivity.

Figures 7c and 7d shows the time and work advantage for the NN detectors. The NN specialized ensemble detector outperforms all other detectors with an average time for detection 8x faster than software only detector (1.2x faster than the LR specialized ensemble detector) when the fraction of malware is low. For the same detector, the the work advantage was 16.6x compared to the software detector. This advantage is 1.4x better than the LR specialized ensemble detector.

6 Hardware Implementation

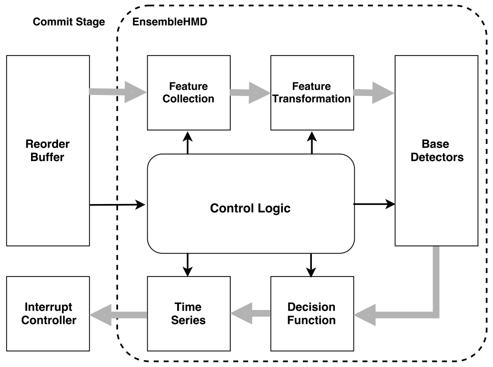


Fig. 8: Ensemble malware detection framework

We design the ensemble detector using 5 major pipeline stages (shown in Figure 8) each of these stages is further pipelined. The first stage of the pipeline (feature collection) performs feature extraction on the instructions being committed from the tail end of the processor pipeline. For example, the INS4 feature is collected by examining the opcodes of committed instructions and outputting the instruction category to the next stage of the pipeline.

The second stage of the pipeline (feature transform) performs context aware transformation on the raw feature information. For instance, INS2 is generated by taking the INS4 feature and echoing the specific feature if it has not yet been committed during the current detection cycle, transforming it from an integer feature to a binary one. The division of feature collection and feature transform reduces the need for multiple highly similar feature collection units.

The third stage consists of the base detectors (neural networks or perceptrons). Perceptrons require minimal hardware investment [32].This low overhead is achievable because perceptrons can be optimized into a single accumulator, feature weight look up table and a comparison (Figure 9).Typically, perceptrons typical operate using the following formula:

$$\begin{cases} 1 & \text{if } \bar{W} \cdot \bar{X} + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

where $\bar{W}$ is the weights vector, $\bar{X}$ is the feature vector, and b is the bias. However, as we commit only 1 feature per cycle we can perform the dot product without actually ever doing a multiply by initializing an accumulator to b then accumulating the weight of each feature when it is committed. Unfortunately, neural networks can not be optimized in the same way. Neural networks require a full multiply accumulate loop and a sigmoid function for each neuron in the network. This causes neural networks to have a significantly higher hardware overhead (as high as 25x for this stage) as compared to perceptrons. Therefore, despite there is an interesting complexity performance trade-off between LR and NN.

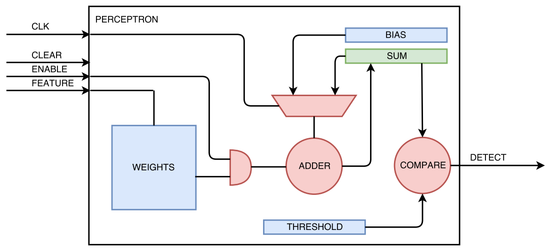


Fig. 9: Optimized perceptron

The fourth stage takes the outputs of the base detectors and combines them to form a single prediction. As mentioned in Section 4 we explored a number of different methods for combining the outputs of the base detectors, all of which are simple to implement in hardware. For instance, the most powerful and complex decision function, stacking, can be implemented as a look up table as the number base detectors is small. For example, in our implementation we perform stacking on 5 detectors, using a single 32 bit register to hold all the possible outputs.

Finally, in the fifth stage, time series analysis of the decision function is performed to increase the specificity of the model. In our implementation we use a windowed moving average model. More complex models such as an auto-regressive integrated moving average could possibly provide better results but require a larger hardware budget.

We used the Quartus II 17.1 software to synthesize implementation of the ensemble detectors attached to the commit stage of the processor pipeline on an DE2-115 FPGA board [1]. Our complete implementation using perceptrons requires a minimal hardware investment. Taking up only 2.88% of logic cells on the core and using only 1.53% of the power compared to an open source processor [3]. Figure 10 shows the FPGA layout of the implementation integrated to the processor. The implementation was functionally validated to collect features and classify them correctly. 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 more cycles this cost be significantly reduced or eliminated.

We also note that the area overheads are relative to a small in-order core [3]; compared to a modern core with caches the overhead is likely to be negligible. The frequency overheads are based on the FPGA implementation of the detectors which are known not to correspond directly to delays incurred in a custom implementation of the logic [25].

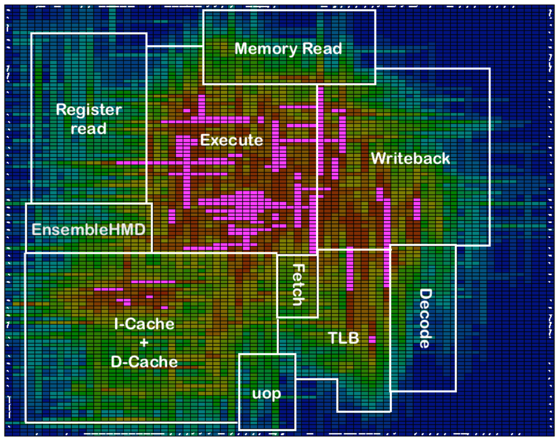
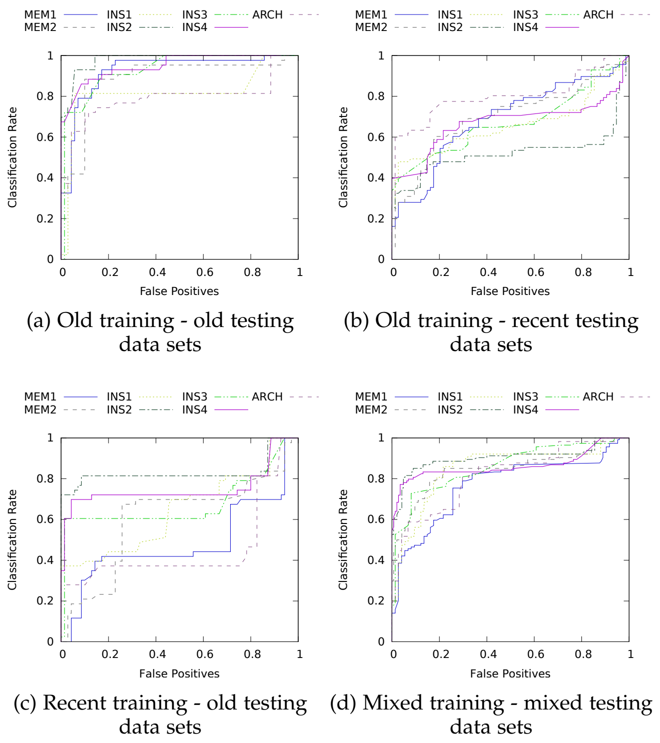


Fig. 10: EnsembleHMD integrated into AO486 processor core

7 Resilience to Malware Evolution

In this section, we study the questions: (1) is a detector trained on a malware training set effective in detecting malware that emerges in the future? If the answer is yes, then there is no need to continue to update the detector to reflect malware evolution. Conversely, if the answer is no, there is a need for a secure mechanism to update the training of the detector over time, even for a hardware-supported detector such as the ones we are studying. (2) When malware evolve, will they insert additional features to the existing ones or they will use different features? These two question were answered for android malwre feature space [4], and we are interested in studying them in the low-level feature space.



(a) Old training - old testing data sets

(b) Old training - recent testing data sets

(c) Recent training - old testing data sets

(d) Mixed training - mixed testing data sets

Fig. 11: Malware Evolution

To answer the first question, we use two malware sets with a 4 years difference in the constituent malware. Specifically, we train seven detectors using data sets for malware found in 2009, corresponding to the different feature vectors. We evaluate the malware using two testing data sets: one that contains only malware found in 2009 and another that contains only malware found in 2013.

Figures 11a and 11b show the ROC curves using the old and new generations of malware respectively. Examining these figures, it is clear that classification performance of the detectors trained on the old malware significantly deteriorates when applied to recent malware. Thus, it is essential to develop mechanisms to securely update the hardware detector (e.g., by updating the θ weights vector for the logistic regression) to continue to track the evolution of malware. This secure update can be integrated our system using microcode update function that is used by Intel and other processor manufacturer.

To answer the second question, we used the 2013 malware in training the detectors and the 2009 malware in testing. This detector is used to test if recent malware have representatives features of older malware. The resulting ROC curves is shown in Figure 11c. The results clearly show that using recent malware in training the detector can't detect older malware effectively. Therefore, Figure 11d show the results when both the 2013 and 2009 datasets are used in both training and testing. The results emphasize the importance of including recent and old malware in the training set of low-level detectors to make sure that they are effective in detecting malware when deployed.

8 Related Work

Hardware based Malware Detectors (HMD) have attracted significant interest recently. Bilar et al. were the first to use the frequency of opcodes occurrence in a program as a feature for discriminating normal programs from malware [7]. Santos et al. and Yan et al. use opcode sequence signatures as a feature [39], [48]. Runwal et al. use similarity graphs of opcode sequences [37]. Recently, Demme et al. suggested using features based on performance counters [11]. Tang et al. conducted a similar study, but used unsupervised learning [44]. Kazdagli et al. [20] introduced several new improvements to the construction methodology for both supervised and unsupervised learning based HMDs applied to mobile malware. All of these studies conduct offline analysis; none of the studies explore online detection or implementation using hardware.

Chavis et al. proposed an enterprise-class antivirus analysis framework, called SNIFFER [8]. Similar to our paper, each machine in the network collect low-level features online using hardware. For each detection period, each machine transfers its collected features securely to a server which classifies them (hardware feature collection, but software classification). The detectors are general detectors; we believe that the ensemble techniques that improve the accuracy can reduce the false positives in this system as well. Both the settings and the malware types (network attacks) considered by the paper are different than our environment.

Ensemble learning is a well-known technique in machine learning to combine the decisions of multiple base detectors to improve accuracy [46]. Ensemble learning is attractive because of its generalization ability which is much powerful than using one learner [12]. 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 [36]. In this paper, the diversity is based on different features (general ensemble detector), data sets (mixed ensemble detector), or both (specialized ensemble detector). In contrast to traditional machine learning approaches that use the training data to learn one hypothesis, some of our ensembles learn a set of subset-hypotheses (specialized detectors) and combine them. Ensemble learning may be considered a form of two-stage detection with the base detectors in the first stage, and the synthesis of their decision in the second. A two stage anomaly detector was proposed by Zhang et al. [53], [54] who use machine learning classifiers to build network traffic dependency graph, and then they used a root-trigger policy to identify outlier network requests. These works focus on the dependency knowledge of the first level, while our work focuses on combining the results of detectors that are trying to answer different questions.

The specialized ensemble detector combines multiple specialized detectors and dynamically collects the features to perform online detection. Researchers built ensemble malware detectors [52], [26], [49], [5], [29], [15], [50], [27], [40], [41], [19], [31], [16], [34], based on combining general detectors. Moreover, most of them used off-line analysis [49], [5], [40], [41], [15], [50], [19]. A few used dynamic analysis [29], [16], [34] and some used both static and dynamic analysis [26], [27], [31]. None of these works uses architecture features or is targeted towards hardware implementation (which requires simpler machine learning algorithms). Specialized detectors were previously proposed [23] for use in malware classification (i.e., labeling malware). Labeling is used to classify collected malware using offline analysis which is a different application than the one we consider.

We present a few recent examples of the use of ensemble learning for malware detection in more detail. Zhang et al. [52] extracted n-gram features from them programs binary code and used it in malware detection. The ensemble composed of multiple probabilistic neural network (PNN) classifiers and the Dempster Shafer theory was utilized to combine them. Sami et al. [38] used API calls extracted from the Portable Executable (PE) Import Address table to build an ensemble detector using random forests. Mehmet et al. [31] created an ensemble detector for android malware by creating base detectors based on different features and learning algorithms. After that, the base detectors was combined using stacking or majority voting to form the ensemble system. However, the previous work does not try to combine specialized detectors to build the ensemble system. Smutz et al. recently explored the use of a diversified ensemble detector to classify possibly evasive PDF malware [42]. Note that all the above techniques use software detectors, on rich features and using advanced machine learning algorithms.

This paper extends our prior work [22] which considered ensembles of only LR detectors. The NN ensemble detectors presented in this paper are over 20% faster than the best LR ensemble detector while requiring 40% less overhead. Additionally, we study the speed of hardware component in detecting malware online and show that NN ensemble is 5x faster than the LR ensemble detector, potentially limiting the damage of the malware. This paper also presents a hardware design of the NN ensemble detectors and analyzes their complexity. The malware longitudinal study in Section 7 is also a new contribution of this paper, and demonstrates the need for HMDs to be retrainable.

In our recent work [21], we explored how malware writers may attempt to evade HMDs. The paper shows that NN detectors are amenable to retraining when malware evolves (e.g., as malware behavior changes as shown in Section 7). The paper proposed creating multiple diverse detectors and switching between them randomly, which makes HMDs provably more robust to evasion attacks. While such detectors use multiple base detectors they use only one at a time. It is interesting to explore the combination of ensemble and evasion resilient HMDs.

9 Concluding Remarks

In this paper, we seek to improve the detection performance of hardware malware detectors (HMDs) through ensemble learning to increase the efficiency of a Two-Level Detector (TLD). We envision an HMD 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 TLD.

We started by evaluating whether specialized detectors can be more effectively classify one given class of malware.

We found out that this is almost always 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 developed metrics to evaluate the performance advantage from better detection in the context of a TLD. Ensemble learning provides more than 16.6x reduction in the online detection overhead with the NN specialized ensemble detector. This represents 2x improvement in performance (overhead) with respect to Ozsoy et al. [32] single detector implementation. 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 optimization of the MEM feature collection circuitry could alleviate most of the cycle time reduction.

We compared performance and overhead of LR to NN as base classifiers. . NN based ensemble detectors provide the highest classification accuracy of all the detectors we developed. Although they are more complicated to implement, we used hardware optimizations to reuse a single perceptron sequentially to implement the neural network, making the hardware overhead small compared to LR.

Finally, we carried out a study of how the detector performs as malware evolves over time. Specifically, we trained a detector with an old malware data set and evaluated its performance on both malware from the same generation, as well as more recent malware. We discovered that the detection performance rapidly deteriorates as malware evolves. In addition, the detectors also fail when trained using only recent malware when classifying old malware. These results emphasize the necessity to provide a secure way to update the hardware detector weights and thresholds as malware continues to evolve.

For our future work, we will study the effect of choosing a thresholds for the detectors that favors sensitivity or specificity over the other, by inserting weights for them in the objective function, on the performance of the two-level detection system.

Acknowledgements

The work in this paper is partially supported by National Science Foundation grants CNS-1422401, CNS-1619322 and CNS-1617915.

References

[1] "Altera de2-115 development and education board," https://www.altera.com/solutions/partners/partner-profile/terasic-inc-/board/altera-de2-115-development-and-education-board.html#overview, 2010.

[2] C. Aldrich and L. Auret, *Unsupervised process monitoring and fault diagnosis with machine learning methods*. Springer, 2013.

[3] O. Aleksander, "The ao486 project," https://github.com/alfikpl/ao486, 2014.

[4] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Are your training datasets yet relevant?" in *International Symposium on Engineering Secure Software and Systems*. Springer, 2015, pp. 51–67.

[5] Z. Aung and W. Zaw, "Permission-based android malware detection," *International Journal of Scientific and Technology Research*, vol. 2, no. 3, pp. 228–234, 2013.

[6] "Malware Statistics," 2014, available online: http://www.av-test.org/en/statistics/malware/.

[7] D. Bilar, "Opcode as predictor for malware," *International Journal of Electronic Security and Digital Forensic*, 2007.

[8] E. Chavis, H. Davis, Y. Hou, M. Hicks, S. F. Yitbarek, T. Austin, and V. Bertacco, "SNIFFER: A high-accuracy malware detector for enterprise-based systems," in *2017 IEEE 2nd International Verification and Security Workshop (IVSW)*, July 2017, pp. 70–75.

[9] 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)*, 2005, pp. 32–46.

[10] 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.

[11] 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.

[12] T. G. Dietterich, "Machine learning research: Four current directions," 1997.

[13] T. G. Dietterich, "Ensemble methods in machine learning," in *International workshop on multiple classifier systems*. Springer, 2000, pp. 1–15.

[14] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM Computing Surveys*, vol. 44, no. 2, Mar. 2008.

[15] M. Eskandari and S. Hashemi, "Metamorphic malware detection using control flow graph mining," *International Journal of Computer Science and Network Security*, vol. 11, no. 12, pp. 1–6, 2011.

[16] G. Folino, C. Pizzuti, and G. Spezzano, "Gp ensemble for distributed intrusion detection systems," in *Pattern Recognition and Data Mining*, 2005, pp. 54–62.

[17] J. L. Henning, "SPEC CPU2006 benchmark descriptions," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, 2006.

[18] D. W. Hosmer Jr. and S. Lemeshow, *Applied Logistic Regression*. John Wiley & Sons, 2004.

[19] 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*. IEEE, 2015, pp. 189–196.

[20] M. Kazdagli, V. J. Reddi, and M. Tiwari, "Quantifying and improving the efficiency of hardware-based mobile malware detectors," in *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*. IEEE, 2016, pp. 1–13.

[21] K. N. Khasawneh, N. Abu-Ghazaleh, D. Ponomarev, and L. Yu, "RHMD: Evasion-resilient hardware malware detectors," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-50 '17. New York, NY, USA: ACM, 2017, pp. 315–327. [Online]. Available: http://doi.acm.org/10.1145/3123939.3123972

[22] K. N. Khasawneh, M. Ozsoy, C. Donovick, N. Abu-Ghazaleh, and D. Ponomarev, "Ensemble learning for low-level hardware-supported malware detection," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2015, pp. 3–25.

[23] J. Z. Kolter and M. A. Maloof, "Learning to detect and classify malicious executables in the wild," *The Journal of Machine Learning Research*, vol. 7, pp. 2721–2744, 2006.

[24] C. Kruegel, W. Robertson, and G. Vigna, "Detecting kernel-level rootkits through binary analysis," in *Proc. Annual Computer Security Applications Conference (ACSAC)*, 2004, pp. 91–100.

[25] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Transactions on computer-aided design of integrated circuits and systems*, vol. 26, no. 2, pp. 203–215, 2007.

[26] 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*, p. 1550018, 2012.

[27] 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*, vol. 39, no. 2, pp. 57–72, 2010.

[28] "How Microsoft antimalware products identify malware and unwanted software," available online: https://www.microsoft.com/security/portal/mmpc/shared/objectivecriteria.aspx.

[29] P. Natani and D. Vidyarthi, "Malware detection using API function frequency with ensemble based classifier," in *Security in Computing and Communications*. Springer, 2013, pp. 378–388.

[30] "Malwaredb Website," 2015, available online (last accessed, May 2015): www.malwaredb.malekal.com.

[31] M. Ozdemir and I. Sogukpinar, "An android malware detection architecture based on ensemble learning," *Transactions on Machine Learning and Artificial Intelligence*, vol. 2, no. 3, pp. 90–106, 2014.

[32] 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.

[33] M. Ozsoy, K. N. Khasawneh, C. Donovick, I. Gorelik, N. Abu-Ghazaleh, and D. Ponomarev, "Hardware-based malware detection using low-level architectural features," *IEEE Transactions on Computers*, vol. 65, no. 11, pp. 3332–3344, 2016.

[34] S. Peddabachigari, A. Abraham, C. Grosan, and J. Thomas, "Modeling intrusion detection system using hybrid intelligent systems," *Journal of network and computer applications*, vol. 30, no. 1, pp. 114–132, 2007.

[35] "Qualcomm smart protect technology," 2016, last Accessed September 2016 from https://www.qualcomm.com/products/snapdragon/security/smart-protect.

[36] J. R. Quinlan, "Simplifying decision trees," *Int. J. Man-Mach. Stud.*, vol. 27, no. 3, pp. 221–234, Sep. 1987. [Online]. Available: http://dx.doi.org/10.1016/S0020-7373(87)80053-6

[37] N. Runwal, R. M. Low, and M. Stamp, "Opcode graph similarity and metamorphic detection," *J. Comput. Virol.*, vol. 8, no. 1-2, pp. 37–52, May 2012. [Online]. Available: http://dx.doi.org/10.1007/s11416-012-0160-5

[38] A. Sami, B. Yadegari, H. Rahimi, N. Peiravian, S. Hashemi, and A. Hamze, "Malware detection based on mining API calls," in *Proceedings of the 2010 ACM Symposium on Applied Computing*, ser. SAC '10. New York, NY, USA: ACM, 2010, pp. 1020–1025. [Online]. Available: http://doi.acm.org/10.1145/1774088.1774303

[39] 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*. Springer, 2010, pp. 35–43.

[40] R. K. Shahzad and N. Lavesson, "Veto-based malware detection," in *Proc. IEEE Int. Conf. on Availability, Reliability and Security (ARES)*, 2012, pp. 47–54.

[41] S. Sheen, R. Anitha, and P. Sirisha, "Malware detection by pruning of parallel ensembles using harmony search," *Pattern Recognition Letters*, vol. 34, no. 14, 2013.

[42] C. Smutz and A. Stavrou, "When a tree falls: Using diversity in ensemble classifiers to identify evasion in malware detectors," in *Proc. Network and Distributed System Security Symposium (NDSS)*, 2016.

[43] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information Processing and Management*, vol. 45, no. 4, pp. 427–437, jul 2009.

[44] A. Tang, S. Sethumadhavan, and S. J. Stolfo, "Unsupervised anomaly-based malware detection using hardware features," in *International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*. Springer, 2014, pp. 109–129.

[45] R. Wahbe, S. Lucco, T. Anderson, and S. Graham, "Efficient software-based fault isolation," in *ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*. New York: ACM Press, 1993, pp. 203–216.

[46] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005.

[47] D. H. Wolpert, "Stacked generalization," *Neural Networks*, vol. 5, pp. 241–259, 1992.

[48] G. Yan, N. Brown, and D. Kong, "Exploring discriminatory features for automated malware classification," in *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2013, pp. 41–61.

[49] 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*, vol. 5, no. 4, pp. 283–293, 2009.

[50] S. Y. Yerima, S. Sezer, and I. Muttik, "High accuracy android malware detection using ensemble learning," *IET Information Security*, 2015.

[51] I. You and K. Yim, "Malware obfuscation techniques: A brief survey," in *Proc. International Conference on Broadband, Wireless Computing, Communication and Applications*, 2010, pp. 297–300.

[52] B. Zhang, J. Yin, J. Hao, D. Zhang, and S. Wang, "Malicious codes detection based on ensemble learning." in *Lecture Notes in Computer Science*, vol. 4610. Springer, 2007, pp. 468–477.

[53] H. Zhang, D. D. Yao, and N. Ramakrishnan, "Detection of stealthy malware activities with traffic causality and scalable triggering relation discovery," in *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*, ser. ASIA CCS '14. New York, NY, USA: ACM, 2014, pp. 39–50. [Online]. Available: http://doi.acm.org/10.1145/2590296.2590309

[54] H. Zhang, D. D. Yao, and N. Ramakrishnan, "Causality-based sensemaking of network traffic for android application security," in *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security*, ser. AISec '16. New York, NY, USA: ACM, 2016, pp. 47–58. [Online]. Available: http://doi.acm.org/10.1145/2996758.2996760

[55] M. Zhang and R. Sekar, "Control flow integrity for cots binaries," in *Proc. 22nd Usenix Security Symposium*, 2013.

Khaled N. Khasawneh is a PhD student in the Department of Computer Science and Engineering at the University of California, Riverside. He received his MS degree in Computer Science from SUNY Binghamton in 2014. His research insterests are in architecture support for security and adversarial machine learning.

Meltem Ozsoy is a Research Scientist at Intel Labs, Hillsboro, OR. She received her PhD in Computer Science from SUNY Binghamton. Her research interests are in the areas of computer architecture and secure system design.

Caleb Donovick is a PhD student in the Department of Computer Science at Stanford University.

Nael Abu-Ghazaleh is a Professor in the Computer Science and Engineering department and the Electrical and Computer Engineering department at the University of California at Riverside. His research interests are in the areas of secure system design, parallel discrete event simulation, networking and mobile computing. He received his PhD from the University of Cincinnati in 1997.

Dmitry Ponomarev is a Professor in the Department of Computer Science at SUNY Binghamton. His research interests are in the areas of computer architecture, secure and power-aware systems and high performance computing. He received his PhD from SUNY Binghamton in 2003.