Towards building active defense systems for software applications

Zara Perumal and Kalyan Veeramachaneni

Data to AI Lab, MIT LIDS, Cambridge, MA 02139,
{zperumal, kalyanv}@mit.edu
(617) 452-3968

Abstract. Over the last few years, cyber attacks have become increasingly sophisticated. PDF malware – a continuously effective method of attack due to the difficulty of classifying malicious files – is a popular target of study within the field of machine learning for cybersecurity. The obstacles to using machine learning are many: attack patterns change over time as attackers change their behavior (sometimes automatically), and application security systems are deployed in a highly resource-constrained environments, meaning that an accurate but time-consuming machine learning cannot be deployed.
Motivated by these challenges, we propose an *active defender* system to adapt to evasive PDF malware in a resource-constrained environment. We observe this system to improve the f_1 score from 0.17535 to 0.4562 over five stages of receiving unlabeled PDF files. Furthermore, average classification time per file is low across all 5 stages, and is reduced from an average of 1.16908 seconds per file to 1.09649 seconds per file. Beyond classifying malware, we provide a general *active defender* framework that can be used to deploy decision systems for a variety of applications operating under resource-constrained environments with adversaries.

1 Introduction

In recent years, cyber attacks have increased dramatically in both scale and sophistication. Last spring, the WannaCry ransomware attack crippled computers around the world [9]. Soon after, attacks on the Equifax credit reporting agency compromised the personal information of millions of users [20]. In addition, banks and Bitcoin exchanges have been subject to an increasing number of attacks. Despite the wide-ranging nature of these attacks, a few commonalities exist. First, most of these attacks enter an enterprise network through an application endpoint, generally when a user unknowingly lets a file with "malware" inside the network - for example, by downloading a malicious "*pdf*" (file) that was delivered *via* an email (application). Second, the most recent attacks are increasingly attributed to Nation-State actors, or Nation-State sponsored cyber-gangs [7, 19]. These powerful attackers often target individuals or small-scale enterprises. Such large adversaries can devote many more resources to attacking a system than their targets can devote to preventing such attacks– an asymmetry that presents a challenging problem [17].

The increasing complexity and scale of software applications makes it even more difficult to monitor their use, find their vulnerabilities, and defend them against attacks.

Application developers have to constantly make trade-offs, balancing between the usability, effectiveness and security of the application. Simple rule-based or signature-based defense systems, while quick to respond to attacks, are not robust enough to provide true protection. Meanwhile, the fact that large amounts of data are constantly being collected has led developers to seek machine learning-based solutions [3].

However, many significant challenges stand in the way of using machine learning for cyber security. First of all, the evolving nature of cyber attacks breaks the assumption that the historic attacks used to train a predictive model would resemble what will actually arrive when the system is deployed. Instead of trying the same type of attack over and over, attackers design automated evasive algorithms specifically to evade these deployed models and create new variants [23, 30].

The second challenge stems from the complex dynamics of the security ecosystem. The actors in a given security problem generally include sophisticated attackers, overburdened security analysts, enterprises who want to defend themselves but not forgo the efficacy of their function, and end-users with a limited knowledge of how to protect themselves (and, subsequently, the enterprise). Complications might include certain detection strategies being public knowledge, or the limited availability of real-time computational resources to run sophisticated detection approaches. Existing solutions fall short in a number of ways. For instance, a highly optimized and accurate attack detection solution could be useless if it also delays people's ability to access and use the application it is defending.

To mitigate these problems, we present an *active defender* system aimed at providing accurate detection in a resource-constrained, adversarial environment.

An active defender system: As shown in Figure 1, our active defender utilizes a "Synthesize-Model-Decide-Adapt" (SMDA) framework to maintain high accuracy while reducing classification time and resource usage. In this paper, we focus on a use case involving PDF malware and test how our approach performs in presence of evasive adversaries. [1].

Our contributions through this paper are as follows:

1. We present a general-purpose *Synthesize-Model-Decide-Adapt* framework to enable the building, evaluating, deploying and subsequently adapting machine learning-based application security systems.
2. We propose a multi detector based hierarchical *decision making* system. We tune the system using Bayesian optimization methods to optimize the usage of the detectors.
3. We present a simple Max-Diff approach, which we show evades even the most sophisticated attack classification system.

The paper is organized as follows: Section 2 presents the use case we focus on in this paper. Section 3 presents our method for synthesizing training examples, Section 4 presents the classifiers we use, Section 5 presents a tunable decision system, and Section 6 presents how we adapt our system. Section 7 presents the experimental settings and our results. Section 8 presents the conclusions and discusses future work. Rather

[1] Several recent studies suggest that PDF malware is evading classification using various automated methods [8, 11, 23, 30]

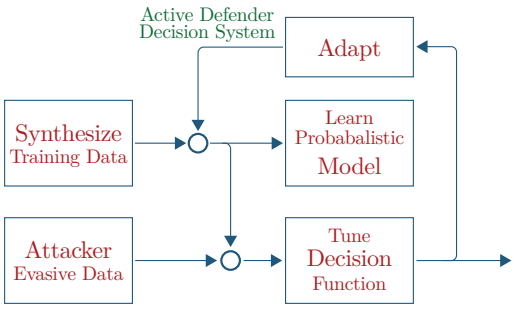

Fig. 1: Active Defender Sytem: The Active Defender system uses the "Synthesize-Model-Decide-Adapt" framework. First the system is initialized by *Synthesizing* training data, then learning several machine learning-based detection *models*, and tuning the *decision system*. After the system is deployed it is used to *decide* on new data, including evasive data generated by the attacker. After a decision is made on newly received data, the system *adapts* to update the *models* and *decision* system

than dedicate a separate section to related work, we have included it in context across different sections.

2 Malware through PDFs

Of all the different file types available, users trust portable document formats the most. People use portable document formats – PDFs for short – to upload everything from academic conference paper submissions to government tax forms. They are also often passed through emails as attachments. Despite their popularity, these unassuming documents contain a powerful format that enables attackers to embed and hide malicious code, spy on users, or encrypt an end user's computer in a ransomware attack [2]. In the next subsection we present multiple ways to detect malware embedded in PDFs.

2.1 PDF Malware Detection

Network detection: Network detection aims to prevent the delivery of malicious content by intercepting it before a user has a chance to download it. Through email analysis such as spam detection or network frequency methods, enterprises can filter out anomalous behaviour and limit the phishing emails and attached malware that make it to the end user. This is usually done in combination with static or dynamic analysis.
Static classifiers: These classifiers use static features of the PDF to quickly detect anomalies before passing the document on to the users. These methods are preferred for their low latency, but have higher error rates. Static classification methods include signature-based detection methods, which can search a received file for the unique bit strings of known malicious files. Other methods attempt to utilize higher-level features, such as n-gram analysis or JavaScript pattern recognition [16, 18]. The most successful

static feature-based classifiers have been $PDFRate$ and $Hidost$. The PDFRate classifier extracts 135 features based on the structure and metadata of a PDF document [25] and uses them to train a machine learning classifier (see [4, 24]). Once the model is trained, it works fast, taking less than a second to classify each new PDF. However, the classifier can be evaded by adversarial algorithms using genetic programming methods [30].

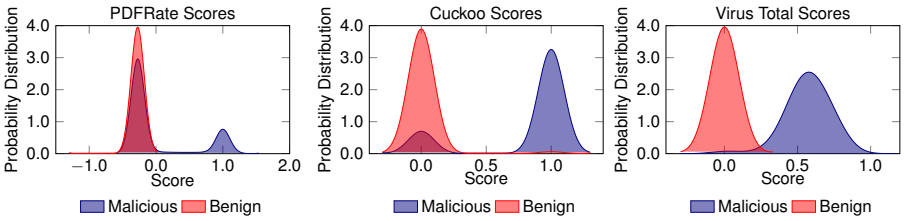

Fig. 2: Kernel density estimation approximation of Probability Density for scores generated using the PDFRate classifier (left), Cuckoo (center) and VirusTotal (right). This plot shows the classifiers' scores for malicious PDFs in pink and benign PDFs in blue. The KDE plot was generated with a Gaussian kernel of width 0.1. A higher overlap indicates an inability to accurately detect the *malicious* PDF. Predictably, out of the three methods, VirusTotal is the best (with the least overlap), and PDFRate is the worst (with the most overlap).

Dynamic behavioural analysis: The Cuckoo Sandbox runs each PDF dynamic analysis sandbox on an isolated "sandboxed" environment. A Cuckoo server runs on the host computer, receives files, and sends them to a virtual machine for analysis. In the virtual machine, Cuckoo simulates opening PDFs in a vulnerable version of Adobe Acrobat, collects information, and compares this information to a set of known behavioural signatures. Cuckoo is fairly accurate for known malicious signatures, and usually requires 30 seconds of simulation time in a virtual machine. For our experiments we used virtual machines set up in VMWare, running Windows XP and a vulnerable version of Adobe Acrobat Reader 8.1.1. Using the results of the Cuckoo classification, we recorded the list of known malicious behavioural signatures observed on the virtual machine when the PDF was processed, and a Boolean indicating if any signatures were observed.

Publicly available APIs: VirusTotal is an API-based detection system. After a user uploads a PDF, it is checked using up to 59 static, dynamic and anti-virus classifiers, and the results of this classification are returned. Some of the classifiers used by VirusTotal include Endgame, Kaspersky Antivirus, Symantec, and Sophos. Aggregating these results can result in a highly accurate classification (see Figure 2), but can also be time- and resource-intensive: a single classification can take up to two minutes, or longer in times of high server load. Furthermore, corporations may be rate-limited by the API and may have to pay for uploads. We collect the aggregate percentage of antivirus engines that classified the uploaded PDF as *malicious*, and we also collect the following

classification attributes for each of the scanners used by VirusTotal: version of scanner used, scan result, and malicious classification.

Human expert analysis: Using human analysts to inspect malware samples is the most accurate form of detection. However, the sheer rate of incoming PDFs requires faster methods. Analysts can compare samples through a variety of methods, including comparing network calls and memory access, running the sample on a hardware sandbox, or comparing activity on a device through a firewall.

Our dataset and method evaluation strategy: To demonstrate the efficacy of different detection techniques, we accumulated a repository of 207,119 PDFs. We created this dataset from a combination of existing, externally provided PDF files, and variations of these PDF files generated via a process called *mutation* (we will describe this briefly in Section 3). We gathered these PDFs from the following sources:

- Contagio: The Contagio dataset provided a corpus of 9,000 benign PDFs and 10,597 malicious files [1].
- EvadeML: EvadeML data provided by Weilin Xu contains 16440 malicious PDF files developed using the "EvadeML" algorithm. These files are based off of 500 malicious files in the Contagio data set and are designed to confuse the PDFRate classifier [30].
- Self-generated: PDFs can be generated from existing PDFs according to the "Random-Mutation","Evade-ML" [30] or "Max-Diff" algorithms. Both the "Evade-ML" and "Max-Diff" algorithms are based on genetic programming. These algorithms create pools of samples, score them, and mutate the best-scoring samples to create more malicious files. In this data set, we generated 8232 malicious files using the "Max-Diff" algorithm and 35,680 benign files using random mutation.

We thus have a total labeled set of 79,949 files: 35,269 malicious and 44,680 benign.

3 Synthesizing training data

To develop an adaptive machine learning solution, we need labeled training examples from the past and an ability to generate evasive versions over time. In recent years, machine learning has been used to automatically create malware samples.

Machine learning to create malicious samples: Beyond the direct methods used to inject malicious code and create PDF files, attackers can *mutate* existing PDF malware in order to create new ones. Many recent studies have focused on methods for generating adversarial PDF files to evade machine learning classifiers.

The mimicus framework presents a method for manipulating PDF classification, through modifying mutable features and through gradient descent methods using attributes of the model [4, 15]. The EvadeML framework presents a black box genetic programming approach to evade a classifier when the classification score is known [30]. The EvadeHC method evades machine learning classifiers without knowledge of the model or classification score [11]. The SeedExploreExploit framework presents another evasion method for deceiving black box classifiers by allowing adversaries to prioritize level diversity and accuracy to generate samples [23].

Other methods operate on the feature space and generate evasive features that could confuse classifiers; however, it is not always clear how to convert evasive features back into malicious files [13]. Many additional methods have been presented to deceive machine learning classifiers based on stationarity assumptions that do not hold in an adversarial environment [5, 6, 10–12, 14, 21, 22, 27]. These attacks often focus on complex classifiers, such as deep learning systems, which can be overfit to rely on features that are correlated with malware, rather than those that are necessary for malware. In [29], Wang et al show that complex classifiers can be evaded if even one unnecessary feature is present.

EvadeML: EvadeML uses a genetic programming method to produce variants of the malicious files using a method they call *mutate*.

$f_{mutate}()$ The malicious files are mutated using components from the pool of benign files S_b. The mutation method is implemented using a modified version of the PDFRW software package[2] and works as follows:

Step 1: Load all PDF files into a tree structure.
Step 2: Mutate each malicious PDF by randomly doing one of the following:
- Insert randomly selected sub-tree from a benign file.
- Swap a randomly selected sub-element with a randomly selected sub-tree from the benign file.
- Delete a randomly selected element in the malicious tree representation.

Step 3: Write the mutated tree back as a PDF file.

These variants are tested against the Cuckoo sandbox to ensure that their malicious nature is preserved, then scored using static classification scores [30]. EvadeML found variants that received classification scores of < 0, with PDFRate classifier. For this classifier, scores for benign are supposed to be closer to -1. A negative score for malicious files indicate that the method was able to evade the classifier. In summary the algorithm works as follows:

Step 0: Start with an empty set $S_{evade} = \{\}$
Step 1: Create a set of mutant files using f_m using the set of S_m. Call this set $S_{mutants}$.
Step 2: Check which among the mutants are malicious using the oracle function $o()$. In our case this is the Cuckoo classifier. Call this set S^m_{mutant}
Step 3: Apply PDFRate classifier to the set S^m_{mutant} and generate classification scores.
Step 4: Select the mutants that have classification scores less than the *cutoff*. Add these to the set S_{evade}. These files are the ones that evade the PDFrate classifier.
Step 5: Repeat steps 1 -4 until $|S_{evade}| \geq n_v$, where n_v is number of evasive variants desired.

Max-Diff approach: We propose the *Max-Diff* approach as an alternative method for generating malicious files. It is similar to the EvadeML algorithm in that it uses a malicious and benign pool of variants, scores the malicious variants, mutates the best-scoring variants, adds them to the pool of malicious files, and then continues. However, unlike the Evade-ML algorithm, it does not seek to find files that receive a classification

[2] https://github.com/mzweilin/pdfrwructure

score less than the *cutoff* for a single classifier. Instead, it selects for files that receive different classification scores from different classifiers in the system. For the *active defender* system, Max-Diff targets files that evade PDFRate or VirusTotal. The algorithm works are follows:

Step 0: Start with an empty set $S_{evade} = \{\}$
Step 1: Create a set of mutant files using f_m and using the set of S_m. Call this set $S_{mutants}$.
Step 2: Check which among the mutants are malicious using the oracle function $o()$. In our case this is the Cuckoo classifier. Call this set S^m_{mutant}
Step 3: Apply PDFRate classifier to set S^m_{mutant} and generate classification scores as p_1
Step 4: Upload the set to the VirusTotal website and generate VirusTotal classifier scores for S^m_{mutant} as well as classification scores. Collect these scores in p_3
Step 5: Select the mutants that have $\text{abs}(p_1 - p_3)$ greater than a threshold specified by *cutoff*. Add these to the set S_{evade}. These files represent files that receive different scores from PDFRate vs. VirusTotal, and could confuse a classification system.
Step 6: Repeat steps 1 - 4 until $|S_{evade}| \geq n_v$.

Evasive performance : We analyze these approaches based on their capability to evade PDFRate (the fastest classifier) and VirusTotal (the most accurate classifier). As we see in Figure 3, malicious files generated using the *Evade-ML* are effective in evading PDFRate classification. Additionally, we observe that evasive files generated using the *Max-Diff* approach are especially effective at evading the VirusTotal classifier. In comparing these results we see that the more time-consuming classifier, VirusTotal, does achieve higher accuracy against evasive variants than the PDFRate classifier. However, even Virus Total is not foolproof, which motivates the need for use of human analysts.

4 Learning models from training data

In our *active defender* system, we use the training data provided to us in the form of file sets S_m and S_b. We divide the classifiers into two types, *primary* and *secondary*. Primary classifiers/models take the files as input and produce a probabilistic score p. Secondary classifiers/models take the output of the primary classifiers and deliver a probabilistic score. Although all secondary models are machine learning models, not all primary models are.

Primary classifiers: The *active defender* system uses the classifiers described in Section 2. These are: PDFRate (denoted as C_1), the Cuckoo classifier (denoted as C_2, which returns scores p_2 indicating if known behavioural signatures of malicious files were detected) and a VirusTotal (C_3) classifier that outputs the percentage of VirusTotal classifiers that classify the file as malicious (p_3)

Secondary Classifiers: Secondary classifiers are designed to take in output scores from the primary classifiers and learn a machine learning model. Two secondary classifiers are developed in our *active defender* system. They are:

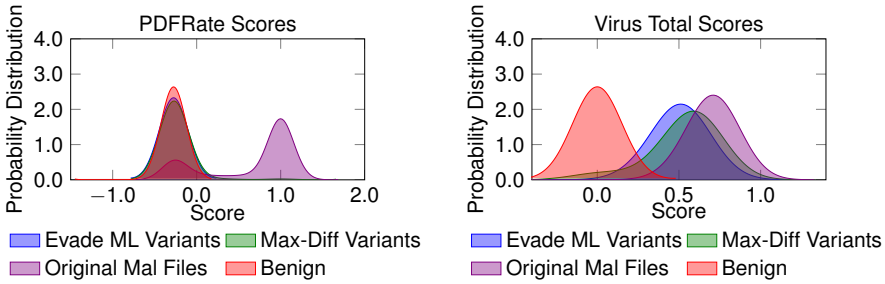

Fig. 3: KDE approximation of Probability Density for the PDFRate (top) and VirusTotal (bottom) scores. This plot shows the classification scores for different types of files. The benign files are shown as pink, the Contagio malware samples are shown in purple, the EvadeML variants are shown in blue, and the Max-Diff variants are shown in green. The KDE plot was generated with a Gaussian kernel of widths 0.15, 0.15, 0.17, and 0.17 for the Benign, Contagio, EvadeML, and Max-Diff files respectively. In this case, the Evade-ML, Max-Diff, and Benign files had very similar probability densities, and differing width kernels were used in order to distinguish them.

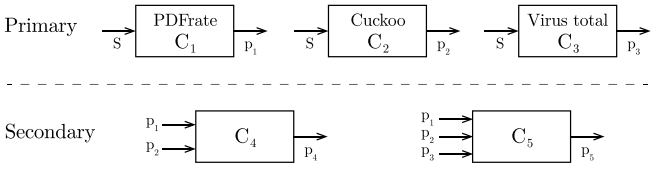

Fig. 4: Active Defender Classifiers: Primary classifiers (C_1,C_2,C_3) receive samples and produce probabilistic scores (p_1,p_2,p_3). Secondary classifiers use these probabilistic scores as inputs. Secondary classifier C_4 uses inputs (p_1,p_2) to produce the probabilistic score p_4, and C_5 uses inputs (p_1,p_2,p_3) to output the probabilistic score p_5.

– C_4 uses the outputs of PDFRate (C_1) and Cuckoo (C_2) as inputs and produces a probabilistic score (p_4).
– C_5 uses the outputs of PDFRate (C_1), Cuckoo (C_2), and Virus Total (C_3) as inputs and produces a probabilistic score (p_5).

5 A tunable decision system

In real time, in order to determine whether or not a new input file s is malicious, we apply a hierarchical decision system that *adaptively* makes use of multiple classifiers. In developing such a decision system, we considered the following goals:

– Increase throughput: We would like to make decisions about PDFs as fast as possible. Because PDFRate is the fastest in giving us the prediction (and VirusTotal is the slowest), we would like to use PDFRate as much as possible.

– Maintain accuracy: While it is easiest to increase our throughput by choosing to use the PDFRate classifier every time, this will lead to a lot of false positives if we have to maintain a high recall of 90%, and we would have to augment with Cuckoo or VirusTotal.

To achieve the goals above, we propose the following:

– a bi-level decision function for classifiers, described in section 5.1,
– a hierarchical, tunable decision system, described in section 5.2,
– A cost function that evaluates the efficacy of a given decision system, described in section 5.3,
– a tuning algorithm that produces a viable decision system, described in section 5.4.

5.1 Bi-level decision function

Given a classifier C_i and its output score p_i, a bi-level decision function allows us to make a decision D_i based on two decision thresholds, t_i^1 and t_i^2, as depicted in Figure 5 and more formally given by:

$$D_i = \begin{cases} \text{Benign} & \text{if } p_i < t_i^1 \\ \text{Uncertain, output } p_i & \text{if } p_i \geq t_i^1 \text{ and } p_i < t_i^2 \\ \text{Malicious} & \text{if } p_i \geq t_i^2 \end{cases} \quad (1)$$

This allows us to make a decision when we are absolutely confident, and enables us to postpone the decision in regions where we are uncertain.

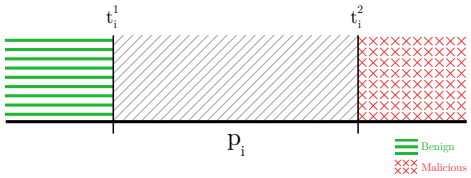

Fig. 5: Bi-level decision function. With input of p_i, the bi-level decision returns a result if it is certain of the classification. It classifies an input as benign if $p_i < t_i^1$ and malicious if $p_i \geq t_i^2$. If $t_i^1 < p_i < t_i^2$, the function returns p_i, as the result is uncertain.

5.2 Hierarchical tunable decision system

The hierarchical decision system is shown in Figure 6. This system determines a final classification result (y) and a probabilistic score (P_{final}) for each input sample using layers of $bi-level$ classifiers. The P_{final} score is calculated using the output of the last classifier (p_{last}), the threshold used in the last decision (t_{last}), the number of primary classifiers used (N_{pc_used}), and the total available primary classifiers (N_{pc_total}) as shown below:

$$P_{final} = \frac{N_{pc\_used}}{N_{pc\_total}} * \text{abs}(p_{last} - t_{last}) \tag{2}$$

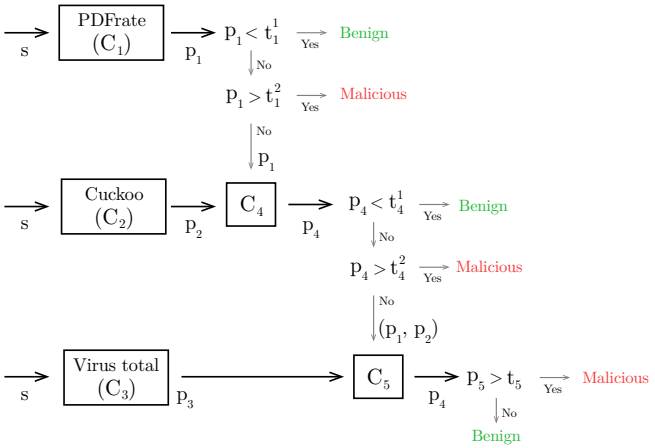

Fig. 6: Active Defender Hierarchical Decision System: A PDF is first sent to the PDFRate classifier (C_1). Based on the output of PDFRate, p_1, a decision is made whether to return a result or send the file to the Cuckoo classifier (C_2). If the file is sent to the Cuckoo classifier, the results from PDFRate (p_1), and Cuckoo (p_2) are sent to the secondary classifier C_4 and a decision is made as to whether to return a result or sent the file to VirusTotal (C_3). If the file is sent to the VirusTotal classifier, classification scores from the PDFRate (p_1), Cuckoo (p_2), and VirusTotal (p_3) classifiers are sent to the C_5 secondary classifier and a final decision is made.

5.3 Cost function

The cost function expresses the two objectives we specified above – the throughput, and whether the desired accuracy is achieved. Given a fully specified decision system, with classifiers $C_{1...5}$, decision thresholds $t_1^1, t_1^2, t_4^1, t_4^2, t_5$, and a set of files S, the cost c incurred by the system is evaluated as:

$$= -\gamma * g(\hat{Y}, Y) + (1 - \gamma) * \frac{1}{|S|} \sum_i r_i \tag{3}$$

where $\hat{Y}$ is the set of predicted labels, Y are the corresponding true labels, $g(.)$ measures the accuracy of the predicted labels, $\frac{1}{|S|} \sum_i r_i$ is the average classification time taken to make these decisions based on the subset of models used for each file in the set per sample, and γ is a weight associated with each of the factors.

$g(.)$ function The $g(.)$ function describes the accuracy of a system. We provide two methods of characterizing system accuracy. In $g_1(.)$, the f1 score is optimized to improve precision and recall.

$$g_1(.) = f_1(predicted, true\_labels) \tag{4}$$

In $g_2(.)$, the function requires a minimal threshold of precision and then optimizes for recall. This function is especially applicable for malware detection, as allowing an additional malicious file to enter the system can be very costly, but is required to keep false rejection of benign files below a certain specified rate for user happiness.

$$g_2 = \begin{cases} \text{recall}(\hat{Y}, Y) & \text{if precision}(\hat{Y}, Y) \geq 0.9 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

5.4 Tuning algorithm

Since the *active defender* system utilizes a set of thresholds to determine the decision for an input sample, *tuner* optimizes these thresholds based on their effect on a cost function. The tuning algorithm uses additional training data to optimize.

Tuner comprises two main steps. First, the tuner algorithm enumerates an initial set of classifier thresholds using an enumeration function $e()$ to generate a set of thresholds T, and scores them with the cost function g. This is done in two steps.

- First, we produce a 2 lists of possible "threshold pairs" for each pair of thresholds: ℓ_{t_1},ℓ_{t_4},(t_1^1,t_1^2),(t_4^1,t_4^2) respectively.
- For the last threshold t_5 we produce a list of possible thresholds ℓ_{t_5}.
- Finally, we create ℓ_T using all possible combinations of threshold pairs across lists ℓ_{t_1},ℓ_{t_4} and ℓ_{t_5}.

In our enumeration function $e()$ we produce threshold pairs using the 0%, 20%, 40%, 60%, 80%, and 100% percentile values of previous classification scores for that classifier. For example, if previous PDFRate classification scores p_1 were observed between 0.0 and 0.5, then:
$\ell_{t_1} \equiv \{(0.0, 0.1), (0.1, 0.2), (0.3, 0.4), (0.4, 0.5)\}$. The last threshold list is (ℓ_{t_5}) a list of the 0% , 20%, 40%, 60%, 80%, and 100% percentiles for respective score p_5. More complex enumeration functions can be developed to capture a more expressive range of thresholds.

Enumerating a large threshold set is important in systems with complex cost functions such as $g_2(.)$ which are not monotonic. If too few initial thresholds are enumerated, optimization can result in thresholds that find a local rather than a global minimum cost function value.

Second, *tuner* uses a maximum of $n_{iterations}$ of Bayesian hyperparameter tuning [3] to propose an additional candidate threshold sets, evaluate it using g, add it to the threshold set T, and find the thresholds that minimize the cost function g. In iterative tuning, ϵ specifies the minimum distance between successive minimum scores to stop optimization [26].

[3] We make use of the open source library: https://github.com/HDI-Project/BTB

6 Adapting over time

One of the most important aspects of the *active defender* system is the ability of the entire system to adapt over time, enabling it to overcome attackers who build evasive variants. Known as active learning, this adaptation can happen over time, simply by adding *verified* labeled training examples.

In [28], Veeramachaneni and Arnaldo study the use of active learning in cybersecurity. They use multiple outlier detection systems, send suspicious activity to analysts and seek their input in order to be able to provide more training examples to the machine learning model over time. We build on this idea, generating more labeled training data using a variety of possible methods:

- higher accuracy classifiers: we can incorporate predictions from VirusTotal as possible source of true labels.
- human analysts: we can send some examples to humans to get their analysis. This is expensive, but still doable.
- synthetically generated evasive variants: From time to time, we can create evasive confirmed malicious variants using the machine learning methods we described in Section 3.

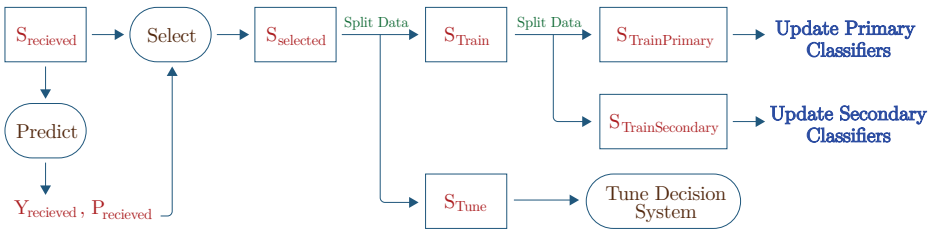

Fig. 7: Data flow diagram of how new training examples are used to adapt the system. 1) Input data $S_{received}$ sent through the decision system to produce predicted labels $Y_{received}$ and probabilities. 2) Samples are selected in using probabilities $P_{received}$. 3) The selected data $S_{selected}$ is split into S_{train} and S_{tune}. 4) The training data S_{train} is split into $S_{primary}$ used to train the primary classifiers and $S_{secondary}$ used to update the secondary classifiers. 5) The tuning data S_{tune} is used to tune the decision system

Adapt in Active Defender: In an active learning scenario, we use additional data to *update* the models and *tune* the decision system. For unlabeled data, the system generates labels and final probabilities using the predictions from the previous learned models and the decision system. The adapt algorithm uses the following steps also shown in Figure 7.

- `SELECT`: chooses the files that are above a set minimum probability threshold (α) to be used as malicious training examples.

– `UPDATE`: uses a subset of the selected files specified by (λ) to learned model. This subset is further divided into two parts: one used to train the primary and the second used to train secondary classifiers, specified by parameter μ. In the *active defender* system, the $PDFRate$ classifier (C_1) is the only primary classifier that can be retrained.

  The second part is used to update the secondary classifiers, C_4, C_5 using the new predictions of PDFRate for the labeled data.
– `TUNE`: uses the remaining files to tune the decision function given the enumeration function, $e()$, maximum number of tuning iterations ($n_{iterations}$), and difference between successive minimum scores ϵ.

7 Experimental Setup

In order to understand the performance of the *active defender* system, we analyze its accuracy and resource use as it adapts. In the experimental design, we first split the data into two data sets, as shown in Figure 8.

Training Data: D_1 corresponds to data used to train the initial system. In our experimental setup, D_1 consists of the 10,597 Contagio malware files and 10,597 benign PDFs randomly selected from the 44,680 benign files discussed in Section 2.

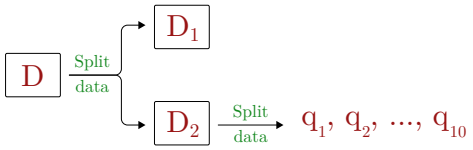

Fig. 8: Splitting Experimental Data. In the following experiment, the data is split into data sets D_1 and D_2. D_1 is used to initialize the decision system. D_2 represents data received by the system after it is deployed. D_2 is split into subsets q_i, representing the files received in each successive stage.

Adaptation Data: D_2 is data received by the system after it is deployed. The adaptation data, D_2, consists of the evasively generated malware and remaining benign PDF files. As shown in Figure 8, this data is split into subsets $q1$ through $q5$ and is sent to the decision system across 5 stages or time periods.

Settings: We perform 25 random trials. In each trial, the order of the files is randomized giving D_1 and D_2 different files across trials. As a result the subsets q_1 ... q_5 are also different.

In setting up the decision system, we set the following parameters for the tuning algorithm described in Section 5 and Section 6.The cost function is $g_1()$; a γ value of 0.9 is used to prioritize accuracy over resource usage and an epsilon value of $\epsilon \equiv 0.1$ is set.

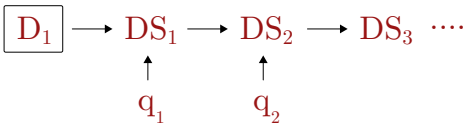

Fig. 9: Updating the decision system. In the experiment, training data D_1 is used to initialize the decision system. After the system is deployed, it receives additional data. After each additional received dataset q_i, the decision system *adapts*.

7.1 Results

Overall, we see that the system is able to adapt to achieve high accuracy in the presence of evasive adversaries, and to reduce resource usage over time.

Accuracy: As shown in Figure 10 and Table 1, we observe the performance of the decision system when classifying successive sets of received files. We characterize accuracy by observing the f_1 score, comparing truth versus labeled data. As evasive variants are introduced in stage 1, we observe a low f_1 score. However, as the stages progress, we observe that the system is able to adapt to improve accuracy over time.

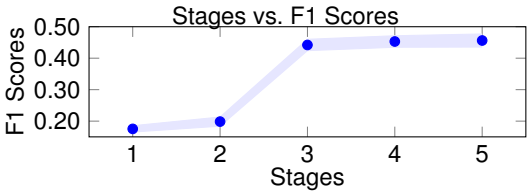

Fig. 10: Accuracy over Adaptation: In this figure, we observe the f_1 score vs. the experimental stage over time. We plot the mean f_1 score as points and show the standard deviation in the surrounding band. We observe poor results in Stage 1, when evasive samples are introduced. Over time, we observe that the f_1 score increases as the system adapts to evasive samples.

Resource Usage: For the purposes of this experiment, we characterize resource usage by studying the average time used to classify each file. As shown in Figure 11 and Table 1, when the system is initialized, classification time is relatively low, at around 1(s) per file. However, we observe that the classification time continues to decrease over time, indicating that the PDFRate static classifier is improving and being utilized. In calculating the estimated classification time, we model the PDFRate as taking 1 second, Cuckoo as taking 25 seconds and VirusTotal as taking 90 seconds. These have come from our own experience running three classifiers for several thousands of PDFs. Notably, the standard deviation in classification time is too small to observe using four decimals of precision. This is likely due to the majority of files being classified by the static classifier and our estimation function limiting the variability in time.

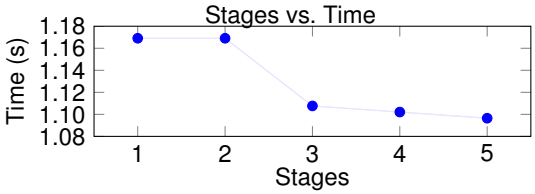

Fig. 11: Time taken to classify as the system runs for several stages: In this figure, we observe the estimated classification time per file at each stage. We plot the mean time as points and show the standard deviation in the surrounding band. Here we see that the average classification time is pretty low, around 1.16 second, and decreases throughout the course of the experiment. The deviation in time is small per stage, and is not observable due to the estimation function.

Stage	μ_{f1}	σ_{f1}	$\mu_{\text{Time per File}}$	$\sigma_{\text{Time per File}}$
1	0.17535	0.01003	1.16908	<0.0001
2	0.19852	0.01459	1.16908	<0.0001
3	0.44201	0.01804	1.10766	<0.0001
4	0.45301	0.01829	1.10208	<0.0001
5	0.4562	0.02082	1.09649	<0.0001

Table 1: Experimental data: 25 trials of the Active Defender System performance over 5 stages. Column μ_{F1} corresponds to the average f_1 score across all trials. Column σ_{f1} corresponds to the standard deviation in f_1 score across all trials. Column $\mu_{TimeperFile}$ corresponds to the average estimated classification time per file. Column $\sigma_{TimeperFile}$ corresponds to the standard deviation in approximated classification time per file.

8 Discussion and Future Work

We were able to make four contributions through this paper. First, we developed a method to use machine learning in application security in a resource-constrained environment. Second, we developed algorithms that use active learning to improve fast classifiers in the presence of adversaries. Third, we provided an extensible framework to facilitate building, evaluating and deploying decision systems in an adversarial and resource-constrained environment. Fourth, we provided a simple evasive algorithm that was shown to confuse automated classifiers.

While studying the adversarial and resource-constrained problem of detecting evasive PDF malware and building these solutions, we identified a few takeaways that motivate future work.

Evasive approaches motivate adaptation over time: In studying the available classifiers, we were surprised to see that the *max-diff* approach was effective in evading the VirusTotal classifier. VirusTotal is a powerful classification system that has been acquired by Google and was considered to be one of the best products of 2007. If this genetic programming-based algorithm can cause confusion in malicious and benign files,

this suggests that adversaries are more than capable of deploying their own evasive algorithms to evade automated classifiers. This motivates the need for human-in-the-loop systems and systems that adapt over time.

Active Defender In studying the behaviour of the *active defender* decision system, we identified aspects of the decision and adaptation methods that could be improved upon in future work. Exploring different methods for tuning the decision system could reduce the tuning time necessary to achieve high accuracy. Studying different ways of using available primary classifiers could decrease classification time. Using randomization to select a small number of files to be sent to the most accurate classifiers can make the system more robust to files that can completely evade simple classifiers. Finally, studying more complex methods could improve adaptation – for example, automated synthesis to create new samples to improve confidence in predictions after adaptation.

8.1 Conclusion

As motivated attackers use more and more computational resources and state-of-the-art algorithms to persistently attack smaller corporations, it is necessary to figure out how to allow detection methods to *adapt* in a resource-constrained environment. As enterprises collect more and more data, machine learning can be an asset to application security; however, each institution looking to defend their system will have different limitations on the resources they can devote to analyzing this data. In this paper, we propose an active defense system that utilizes the SMDA framwork. This system can be tailored for different resource limitations and environments. Furthermore, we believe that this software framework and algorithms can generalize beyond PDF malware detection, enabling researchers and corporations to work together to secure systems against powerful and evolving adversaries.

Bibliography

[1] Contagio dump, http://contagiodump.blogspot.com (accessed on 2016.11.11)., http://contagiodump.blogspot.com

[2] The rise of document-based malware. https://www.sophos.com/en-us/security-news-trends/security-trends/the-rise-of-document-based-malware.aspx

[3] The rise of machine learning (ml) in cybersecurity. https://www.crowdstrike.com/resources/white-papers/rise-machine-learning-ml-cybersecurity/

[4] Mimicus framweork. https://github.com/srndic/mimicus (2017)

[5] Argyros, G., Stais, I., Jana, S., Keromytis, A.D., Kiayias, A.: Sfadiff: Automated evasion attacks and fingerprinting using black-box differential automata learning. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 1690–1701. ACM (2016)

[6] Argyros, G., Stais, I., Kiayias, A., Keromytis, A.D.: Back in black: towards formal, black box analysis of sanitizers and filters. In: Security and Privacy (SP), 2016 IEEE Symposium on. pp. 91–109. IEEE (2016)

[7] Ashford, W.: Cyber criminals catching up with nation state attacks, https://www.computerweekly.com/news/252435701/Cyber-criminals-catching-up-with-nation-state-attacks

[8] Biggio, B., Corona, I., Maiorca, D., Nelson, B., Šrndić, N., Laskov, P., Giacinto, G., Roli, F.: Evasion attacks against machine learning at test time. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. pp. 387–402. Springer (2013)

[9] Bossert, T.P.: It's official: North korea is behind wannacry (Dec 2017), https://www.wsj.com/articles/its-official-north-korea-is-behind-wannacry-1513642537

[10] Chen, Y., Nadji, Y., Kountouras, A., Monrose, F., Perdisci, R., Antonakakis, M., Vasiloglou, N.: Practical attacks against graph-based clustering. arXiv preprint arXiv:1708.09056 (2017)

[11] Dang, H., Huang, Y., Chang, E.C.: Evading classifiers by morphing in the dark (2017)

[12] Hosseini, H., Xiao, B., Clark, A., Poovendran, R.: Attacking automatic video analysis algorithms: A case study of google cloud video intelligence api. arXiv preprint arXiv:1708.04301 (2017)

[13] Hu, W., Tan, Y.: Generating adversarial malware examples for black-box attacks based on gan. arXiv preprint arXiv:1702.05983 (2017)

[14] Kantchelian, A., Tygar, J., Joseph, A.: Evasion and hardening of tree ensemble classifiers. In: International Conference on Machine Learning. pp. 2387–2396 (2016)

[15] Laskov, P., et al.: Practical evasion of a learning-based classifier: A case study. In: Security and Privacy (SP), 2014 IEEE Symposium on. pp. 197–211. IEEE (2014)

[16] Li, W.J., Stolfo, S., Stavrou, A., Androulaki, E., Keromytis, A.D.: A study of malcode-bearing documents. In: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. pp. 231–250. Springer (2007)

[17] MacFarlane, D., Network, I.C.: Why even smaller enterprises should consider nation-state quality cyber defenses (Sep 2017), https://www.csoonline.com/article/3223866/cyberwarfare/nation-state-quality-cyber-defenses.html
[18] Maiorca, D., Corona, I., Giacinto, G.: Looking at the bag is not enough to find the bomb: an evasion of structural methods for malicious pdf files detection. In: Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security. pp. 119–130. ACM (2013)
[19] Millman, R.: Nation state cyber-attacks on the rise - detect lateral movement quickly (Feb 2018), https://www.scmagazineuk.com/nation-state-cyber-attacks-on-the-rise--detect-lateral-movement-quickly/article/746561/
[20] Riley, M., Robertson, J., Sharpe, A.: The equifax hack has the hallmarks of state-sponsored pros (Sep 2017), https://www.bloomberg.com/news/features/2017-09-29/the-equifax-hack-has-all-the-hallmarks-of-state-sponsored-pros
[21] Rosenberg, I., Shabtai, A., Rokach, L., Elovici, Y.: Generic black-box end-to-end attack against rnns and other api calls based malware classifiers. arXiv preprint arXiv:1707.05970 (2017)
[22] Sethi, T.S., Kantardzic, M.: Data driven exploratory attacks on black box classifiers in adversarial domains. arXiv preprint arXiv:1703.07909 (2017)
[23] Sethi, T.S., Kantardzic, M., Ryu, J.W.: security theater: On the vulnerability of classifiers to exploratory attacks. In: Pacific-Asia Workshop on Intelligence and Security Informatics. pp. 49–63. Springer (2017)
[24] Smutz, C., Stavrou, A.: Malicious pdf detection using metadata and structural features. In: Proceedings of the 28th Annual Computer Security Applications Conference. pp. 239–248. ACM (2012)
[25] Smutz, C., Stavrou, A.: When a tree falls: Using diversity in ensemble classifiers to identify evasion in malware detectors. NDSS (2016)
[26] Swearingen, T., Drevo, W., Cyphers, B., Cuesta-Infante, A., Ross, A., Veeramachaneni, K.: Atm: A distributed, collaborative, scalable system for automated machine learning. In: IEEE International Conference on Big Data (2017)
[27] Tong, L., Li, B., Hajaj, C., Vorobeychik, Y.: Feature conservation in adversarial classifier evasion: A case study. arXiv preprint arXiv:1708.08327 (2017)
[28] Veeramachaneni, K., Arnaldo, I., Korrapati, V., Bassias, C., Li, K.: Aiˆ 2: training a big data machine to defend. In: Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS), 2016 IEEE 2nd International Conference on. pp. 49–54. IEEE (2016)
[29] Wang, B., Gao, J., Qi, Y.: A theoretical framework for robustness of (deep) classifiers under adversarial noise. arXiv preprint arXiv:1612.00334 (2016)
[30] Xu, W., Qi, Y., Evans, D.: Automatically evading classifiers. In: Proceedings of the 2016 Network and Distributed Systems Symposium (2016)