Static Malware Detection & Subterfuge: Quantifying the Robustness of Machine Learning and Current Anti-Virus

William Fleshman[1] Edward Raff[1,2,3] Richard Zak[1,2] Mark McLean[1] Charles Nicholas[3]
[1]Laboratory for Physical Sciences, [2]Booz Allen Hamilton, [3]University of Maryland, Baltimore County
{william.fleshman,edraff,rzak,mrmclea}@lps.umd.edu, nicholas@umbc.edu

Abstract

As machine-learning (ML) based systems for malware detection become more prevalent, it becomes necessary to quantify the benefits compared to the more traditional anti-virus (AV) systems widely used today. It is not practical to build an agreed upon test set to benchmark malware detection systems on pure classification performance. Instead we tackle the problem by creating a new testing methodology, where we evaluate the change in performance on a set of known benign & malicious files as adversarial modifications are performed. The change in performance combined with the evasion techniques then quantifies a system's robustness against that approach. Through these experiments we are able to show in a quantifiable way how purely ML based systems can be more robust than AV products at detecting malware that attempts evasion through modification, but may be slower to adapt in the face of significantly novel attacks.

1 Introduction

The threat and impact of malicious software (malware) has continued to grow every year. The annual financial impact is already measured in the hundreds of billions of dollars (Hyman 2013; Anderson et al. 2013). Simultaneously, there are worries that the classical anti-virus approach may not continue to scale and fail to recognize new threats (Spafford 2014).

Anti-virus systems were historically built around signature based approaches. Wressnegger et al. (2016) discussed a number of issues with signatures, but the primary shortcoming is an intrinsically static nature and inability to generalize. Most anti-virus companies have likely incorporated machine learning into their software, but to what extent remains unclear due to the nature of proprietary information. Regardless, adversaries can still successfully evade current detection systems with minimal effort. The success of methods like obfuscation and polymorphism is evident by its prevalence, and recent work has suggested that the majority of unique malicious files are due to the use of polymorphic malware (Li et al. 2017).

 In: Joseph Collins, Prithviraj Dasgupta, Ranjeev Mittu (eds.): Proceedings of the AAAI Fall 2018 Symposium on Adversary-Aware Learning Techniques and Trends in Cybersecurity, Arlington, VA, USA, 18-19 October, 2018, published at http://ceur-ws.org

Given these issues and urgency, Machine Learning would appear to be a potential solution to the problem of detecting new malware. Malware detection can be directly phrased as a classification problem. Given a binary, we define some feature set, and learn a binary classifier that outputs either benign or malicious. The output of this model could be calibrated to reach any desired false-positive ratio.

However, one should never switch to a new technology for its own sake. It is necessary to have empirical and quantifiable reasons to adopt a new approach for malware detection. Ideally, this would be based off the accuracy of current anti-virus systems compared to their machine-learning counterparts. In reality, this is non-trivial to estimate, and many arguments currently exist as to how this should be done (Jordaney et al. 2016; Deo et al. 2016; Sommer and Paxson 2010). Designing a corpus and test protocol to determine accuracy at large is hampered by issues like concept drift (Jordaney et al. 2017), label uncertainty, cost (Miller et al. 2016), and correlations over time (Kantchelian et al. 2013; Singh, Walenstein, and Lakhotia 2012).

Toward remedying this issue, we propose a new testing methodology that can highlight a detector's strength or weakness with regard to specific attack types. We first utilize the framework from Biggio et al., to specify a model for our adversary (Biggio, Fumera, and Roli 2014). After defining the adversary's goal, knowledge, and capabilities, we look at how difficult it is for an adversary with known malware to evade malware detection systems under comparison. As examples, we compare two machine learning approaches, one a simpler n-gram model and one neural network based, to a quartet of production AV systems. While we do not exhaustively test all possible evasion techniques, we show the potential for our protocol using a handful of relevant tests: non-destructive automated binary modification, destructive adversarial attacks, and malicious injection. We are specifically looking only at black box attacks which can be applied to any potential malware detector, assuming that the adversary is looking for maximal dispersion and not a targeted attack. This new evaluation protocol is our primary contribution, and a black-box adversarial attack for byte-based malware detectors our second contribution. This evaluation protocol allows us to make new conclusions on the potential benefits and weakness of two byte based machine learning malware detectors.

We will review previous work in section 2. The experi-

ments and methodology we use to compare our ML based systems to anti-virus will be given in section 3 and section 4, followed by their results in section 5 and conclude in section 6.

2 Related Work

Malware detection and analysis has been an area of active research for several years. One class of techniques that is of particular note is *dynamic analysis*, where the binary itself is run to observe its behavior. Intuitively, malicious behavior is the best indicator of a malicious binary — making dynamic analysis a popular approach. However, dynamic analysis has many complications. It requires significant effort and infrastructure to make it accurately reflect user environments, which are often not met in practice (Rossow et al. 2012). Furthermore, the malware author can use a number of techniques to detect dynamic execution, hide behavior, or otherwise obfuscate such analysis. This means effective dynamic analysis systems are often a cat-and-mouse game of technical issues, and can require running binaries across many layers of emulation (Egele et al. 2017).

It is for these reasons that we focus on the static analysis case. This removes the unlimited degrees of freedom provided by actually running code. However, this does not mean the malicious author has no recourse. Below we will review the related work in this area.

Questions regarding the security of machine learning based systems have been an active area of research for over a decade (Barreno et al. 2010), but have recently received increased attention. This is in particular due to the success in generating *adversarial* inputs to neural networks, which are inputs that induce an incorrect classification despite only minor changes to the input (Szegedy et al. 2014). These approaches generally work by taking the gradient of the network with respect to the input, and adjusting the input until the network's output is altered. This does not directly translate to malware detection when using the raw bytes, as bytes are discontinuous in nature. That is to say, any change in a single byte's value is an equally "large" change, whereas in images, adjusting a pixel value can result in visually imperceptible changes. Arbitrary byte changes may also result in a binary that does not execute, making it more difficult to apply these attacks in this space (Grosse et al. 2016; Russu et al. 2016). While such attacks on individual models are possible (Kolosnjaji et al. 2018; Kreuk et al. 2018), we are concerned with attacks that can be applied to any detector, therefore these methods are out of the scope of this work.

To overcome the issue with arbitrary byte changes breaking executables, Anderson, Filar, and Roth (2017) developed a set of benign actions that can modify a binary without changing its functionality. Selecting these actions at random allowed malware to evade a ML model 13% of the time. Introducing a Reinforcement Learning agent to select the actions increased this to 16% of the time. Their ML model used features from the PE header as well as statistics on strings and bytes from the whole binary. We will replicate this type of attack as one of the comparisons between our ML systems and anti-virus.

Another approach is to modify the bytes of a binary, and run the malware detection system after modification. While this risks breaking the binary, it can still be effective, and it is important to still quarantine malware even if it does not execute as intended. Wressnegger et al. (2016) modified each binary one byte at a time, and found that single byte changes could evade classical AV detectors. They used this approach to find which bytes were important to the classification, and reverse-engineered signatures used from multiple AV products. We extend this technique in subsection 4.2 to create a new adversarial approach against our models, which finds a contiguous byte region which can be altered to evade detection.

For the machine learning based malware detection methods, we look at two approaches: byte n-grams and neural networks. The byte n-gram approach was one of the first machine learning methods proposed for malware detection (Schultz et al. 2001), and has received significant attention (Kolter and Maloof 2006). We use the same approach for training such models as presented in Raff et al. (2016). For the neural network approach we use the recently proposed MalConv, which processes an entire binary by using an embedding layer to map bytes to feature vectors followed by a convolutional network (Raff et al. 2017). Because the embedding layer used by MalConv is non-differentiable with respect to the input bytes, the aforementioned adversarial approaches applied to image classification tasks are not as easily applicable. Similarly, anti-virus products do not provide a derivative for such attacks. This is why we develop a new black-box adversarial approach in subsection 4.2.

Using these modeling methods is beneficial for our analysis because they are trained with minimal assumptions about the underlying data. Other approaches to malware detection that process the PE-header (Shafiq et al. 2009) or use disassembly (Moskovitch et al. 2008) exist. We avoid these under the belief that their use of explicit feature sets aids in evading them (e.g, modifying the PE-Header is easy and allows for direct evasion of models using those features).

We leverage the existing framework from (Biggio, Fumera, and Roli 2014) to model our adversary for each comparison made. This framework prescribes defining the adversary's goals, knowledge, and capabilities in regards to the classifiers.

3 Adversarial Model

For each experiment we specify the model of our adversary by outlining the goals, knowledge, and capabilities available (Biggio, Fumera, and Roli 2014). These make clear the assumptions and use cases presented by each scenario.

The goal of our adversary is the same across all experiments – using a single attack to maximize the misclassification rate of malicious files across many classifiers. The one caveat is that in subsection 4.2 we only goes as far as identifying a small portion of the file which can be altered to evade detection. A real attacker would then have to alter those bytes while retaining the malicious functionality of the file. It is unlikely that authors of benign software would seek a malicious classification – therefore we limit ourselves to experiments on malware only.

We severely constrain the knowledge of our adversary. In all cases we assume the adversary has no knowledge of the classifiers' algorithms, parameters, training data, features, or decision boundaries. This is the best case scenario for security applications, as knowledge of any of the previous attributes would increase the sophistication of possible attacks.

Similarly, the experiments outlined in subsection 4.1 and subsection 4.3 require no prior interaction with the classifiers before deploying a malicious file to the wild. In subsection 4.2, the adversary has the capability of querying our n-gram model only. The inner workings of the model remain hidden, but the output of the model is available for deductive reasoning.

These assumptions are restrictive to the adversary's actions and knowledge, but are important because current adversaries are able to succeed under such restrictions today. This means malware authors can obtain success with minimal effort or expense. Through our evaluation procedure we are able to better understand which techniques / products are most robust to this restricted scenario, and thus will increase the effort that must be expended by malware authors.

4 Experiments and Methodology

We now perform experiments under our proposed testing methodology to compare machine learning models to commercial anti-virus systems. To do so, we compare two machine learning classifiers and four commercial anti-virus products: AV1, AV2, AV3, and AV4. We use these anonymous names to be compliant with their End User License Agreements (EULA), which bars us from naming the products in any comparative publication. We are prohibited from using the internet based mass scanners, like VirusTotal, for similar reasons. Ideally, we could test many products, but with our large corpus of files we limit ourselves to these four – which we believe to be representative of the spectrum of commercially used AV platforms. We purposely did not chose any security products which advertise themselves as being primarily powered by machine learning or artificial intelligence. Additionally, any cloud based features which could upload our files were disabled to protect the proprietary nature of our dataset. Our machine learning based approaches will be built upon n-grams and neural networks which process a file's raw bytes. We will compare these systems on a number of tasks to evaluate their robustness to evasion.

We now describe our new testing protocol. Given a corpus $\{x_1, \ldots, x_n\}$ of n files with known labels $y_i \in \{\text{Benign}, \text{Malicious}\}$, and detector $C(\cdot)$ we first measure the accuracy of $C(x_i), \forall i$. This gives us our baseline performance. We then use a suite of evasion techniques $\phi_1(\cdot), \ldots, \phi_K(\cdot)$ and look at the difference in accuracy between $C(x_i) = y_i$ and $C(\phi(x_i)) = y_i$. The difference in these scores tells us the ability of the detector $C(\cdot)$ to generalize past its known data and catch evasive modifications. Any approach for which $C(x_i) \neq C(\phi_j(x_i))$ for many different evasion methods ϕ which preserve the label y_i, is then an intrinsically brittle detector, even if it obtains perfect accuracy before ϕ is used.

The raw detection accuracy is not the point of this test. Each system will have been built using different and distinct corpora of benign and malicious files. For the AV products, these are used to ensure that a certain target level of false positives and false negatives are met. In addition, the training corpus we use for our models is orders of magnitude smaller than what the AV companies will have access to. For this reason we would not necessarily expect our approaches to have better accuracy. The goal is to quantify the robustness of any classifier to a specific threat model and attack.

One novel and three pre-existing techniques ("ϕ"s) will be used to perform this evaluation. These are not exhaustive, but show how our approach can be used to quantify relative performance differences. Below we will review their methodology, and the information that we can glean from their results. In section 5 we will review the results of running these experiments.

Dataset

We take only a brief moment to expound upon the machine learning models and data used for training, as the information disparity with other AV products makes direct comparison difficult. We trained the machine learning models on a large dataset from industry discussed in (Raff and Nicholas 2017) which contains approximately 2 million samples balanced almost evenly between benign and malicious. Our test set consists of approximately 80,000 files held out for our post-training experimentation. The test set is also almost perfectly balanced. This set was not seen by our machine learning models during training, but could have files previously seen by the anti-virus corporations. The n-gram model follows the procedure used in (Raff et al. 2016), but uses one million features instead of 100,000. The MalConv approach is specified in (Raff et al. 2017). Both of these approaches require no domain knowledge to apply.

4.1 Randomly Choosing Benign Modifications

In this experiment, we use a subset[1] of the modifications used by Anderson, Filar, and Roth (2017) to alter malicious files before scanning them with the classifiers. The objective is to test the brittleness of the classifiers' decision mechanisms by making small changes that do not alter the functionality of the malware. This experiment is ideal because it produces a new file that is still functional, and should have no impact on an *ideal* malware detector.

There are nine different modification actions, and grouped by type are:

- rename or create new sections
- append bytes to the end of a section or the file
- add an unused function to the import table
- create a new entry point (which jumps to the old entry)
- modify the header checksum, the signature, or debug info

Each malicious file was scanned before the experiment to see if it was already being misclassified as benign (false negative). If it was correctly classified as malicious then a modification was randomly selected from the set and applied. This process was repeated up to ten times or until the file evaded the classifier. Approaches that rely too heavily on exact signatures or patterns should suffer in this test, and we

[1] A UPX (un)packing option did not operate correctly.

would expect any approach that uses purely dynamic features to be unaffected by the modifications.

This experiment is time intensive, and so we use a subset of 1,000 files randomly chosen from our test set. Anderson, Filar, and Roth limited themselves to a small 200 file sample, so our larger test set may provide additional confidence in the results. The original paper proposed using a reinforcement learning algorithm (RLA) to select the modifications in an adversarial manner. We found that including a RLA did not change results, but did limit our ability to test with all products in a timely manner.

4.2 Targeted Occlusion of Important Bytes

The first approach described requires extensive domain knowledge based tooling, and would be difficult to adapt to new file types. We introduce a novel approach to find important byte regions of a malicious file given a working detector. Identifying the most important region of a file gives feedback to adversaries, and allows us to occlude that region as an evasion technique.

Our approach works by occluding certain bytes of a file, and then running the malware detector on the occluded file. By finding a contiguous region of bytes that reduced the detector's confidence that a file is malicious, we can infer that the bytes in that region are important to the detector. If occluding a small region causes the detector to change its vote from malicious to benign, then we can infer that the detector is too fragile. In this case it is plausible for a malware author to determine what is stored in the small region detected, and then modify the binary to evade detection.

Manually editing groups of bytes one at a time would be computationally intractable. Given a file F of $|F|$ bytes, and a desired contiguous region size of β, it would take $O(|F|)$ calls to the detector to find the most important region. We instead develop a binary-search style procedure, which is outlined in Algorithm 1. Here $C(\cdot)$ returns the malware detector's confidence that a given file is malicious, and $\mathcal{D}$ is a source of bytes to use for the occlusion of the original bytes. This approach allows us to find an approximate region of importance in only $O(\log |F|)$ calls to the detector $C(\cdot)$. In the event that $C(F_l) = C(F_r)$, ties can be broken arbitrarily – which implicitly covers the boolean decisions from an anti-virus product.

This method starts with a window size equal to half of the file size. Both halves are occluded and the file is analyzed by the classifier. The half that results in the largest drop in classification confidence is chosen to be split for the next time step. This binary search through the file is continued until the window size is at least as small as the target window size.

The last component of our approach is to specify what method $\mathcal{D}$ should be used to replace the bytes of the original file. One approach, which we call *Random Occlusion*, is to simply select each replacement byte at random. This is similar to prior work in finding the important regions of an image according to an object detector, where it was found that randomly blocking out the pixels was effective — even if the replacement values were nonsensical (Zeiler and Fergus 2014). We also look at *Adversarial Occlusion*, where we use a contiguous region selected randomly from one of our own benign training files to occlude the bytes in the file under test. Since we use this approach only with malicious files, Adversarial Occlusion is an especially challenging test for our machine learning based methods: they have seen these byte regions before with an explicit label of benign. This test also helps to validate that all approaches aren't simply detecting high-entropy regions that "look" packed, and then defaulting to a decision of malicious.

Algorithm 1 Occlusion Binary Search

Require: A file F of length $|F|$, a classifier $C(\cdot)$, target occlusion size β, byte replacement distribution $\mathcal{D}$

split $\leftarrow |F|/2$, size $\leftarrow |F|/2$
start $\leftarrow 0$, end $\leftarrow |F|$
while size $> \beta$ **do**
    $F_l \leftarrow F$, $F_r \leftarrow F$
    F_l[split$-$size:split] $\leftarrow$ contiguous sample $\sim \mathcal{D}$
    F_r[split:split$+$size] $\leftarrow$ contiguous sample $\sim \mathcal{D}$
    if $C(F_l) < C(F_r)$ **then**
        split $\leftarrow$ split $-$ size$/2$
        start $\leftarrow$ split $-$ size, end $\leftarrow$ split
    else
        split $\leftarrow$ split $+$ size$/2$
        start $\leftarrow$ split, end $\leftarrow$ split $+$ size
    size $\leftarrow$ size$/2$
return start, end

To show that our search procedure provides meaningful improvements for the Random and Adversarial Occlusions, we will also compare against an *Undirected Occlusion* approach that eschews our search procedure. In this case we randomly select a region of the binary to occlude with high-entropy random bytes.

Recall, that this method identifies the critical region – for a successful attack the adversary would then have to manually alter the portion of the file at that location while maintaining their intended outcome. For this reason, we use a single target window size of 2048 bytes. The resulting occluded area will fall in the range of (1024-2048] bytes. On average, this restricts the occluded section to under 0.5% of the file size for our testing set. We also limit ourselves to searching for critical regions with our n-gram model. It would be infeasible for an adversary to conduct this search across all possible classifiers, and we would expect there to be some overlap among which regions are critical. This has been shown to be true in gradient based classifiers, where an adversarial example generated by one model can fool many others (Papernot, McDaniel, and Goodfellow 2016).

4.3 ROP Injection

The last method we will consider is of a different variety. Instead of modifying known malicious files in an attempt to evade detection, we will inject malicious functionality into otherwise benign applications.

Many such techniques for this exist. We used the Return Oriented Programming (ROP) Injector (Poulios, Ntantogian, and Xenakis 2015) for our work. The ROP Injector converts malicious shellcode into a series of special control flow in-

structions that are already found inside the file. These instructions are inherently benign, as they already exist in the benign file. The technique patches the binary in order to modify the memory stack so that the targeted instructions, which are non-contiguous in the file, are executed in an order equivalent to the original shellcode. The result is that the functionality of the file is maintained, with the added malicious activity executing right before the process exits. We use the same reverse Meterpreter shellcode as Poulios, Ntantogian, and Xenakis (2015) for our experiment.

We note that not all benign executables in our testing set were injectable. The files must either have enough instructions to represent the shellcode already, or have room for further instructions to be added in a non-contiguous manner so as to prevent raising suspicion. Additionally, if the malware requires external functionality such as networking capabilities, and these are not inherent to the original executable, then portions of the PE header such as the import table must be adjusted. For these reasons, only 13,560 files were successfully infected.

Poulios, Ntantogian, and Xenakis claim they were able to evade anti-virus over 99% of the time using this technique. This approach should be nearly undetectable with static analysis alone, as all instructions in the file remain benign in nature. This makes it an extremely difficult test for both the machine learning and anti-virus detectors.

5 Results

Before delving into results, we make explicit that we considered Malware the positive class and Benign the negative class . We remind the reader that not all tested systems are on equal footing. Our n-gram and MalConv models are trained on the same corpus, but we have no knowledge of the corpora used by the AV companies and their products. In all likelihood, they will be using datasets orders of magnitude larger than our own to ensure the quality of their products and to reach a desired false-positive rate. From this perspective alone, we would not necessarily expect the machine learning based methods to have better accuracies, as there is a disparity of information at training time. Our methods are also not optimized for the same low false-positive goal.

The accuracies and metrics of each method are presented in Table 1. Because of the mentioned information disparity between each system, we do not present this information for direct comparison. Our goal is to use these metrics as a base-line measure of performance for each method, and look at how these baselines are affected by various evasion techniques.

5.1 Benign Modifications

The results of the experiment described in subsection 4.1 are shown in Figure 1. All four anti-virus products were significantly degraded by making seemingly insignificant changes to the malware files. The machine learning models were immune to this evasion technique. Both model's confidence that these files were malware changed by less than 1% on average. The few files that did evade were very close to the models' decision boundaries before the modifications were performed.

Table 1. Baseline accuracy and true/false positive/negative results for each model on the test set.

Classifier	TN%	TP%	FN%	FP%	Accuracy%
N-Gram	92.1	98.7	1.3	7.9	95.5
MalConv	90.7	97.2	2.8	9.3	94.1
AV1	94.3	99.5	0.5	5.7	97.0
AV2	99.4	64.9	35.1	0.6	81.6
AV3	98.5	80.5	19.5	1.5	89.2
AV4	93.8	91.9	8.1	6.6	92.6

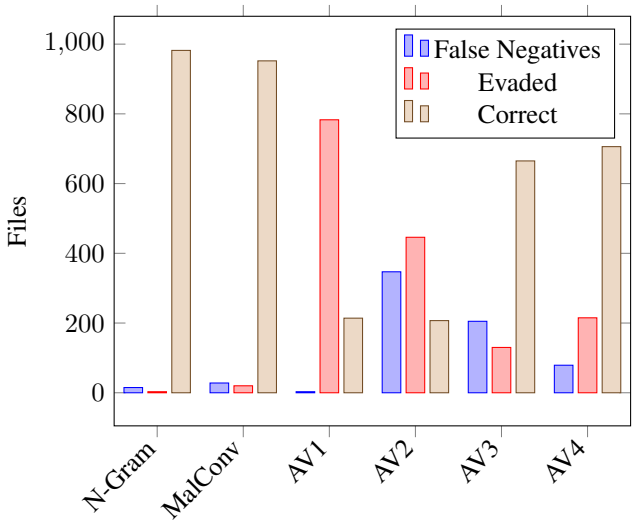

Figure 1. Robustness of classifiers to benign modifications. The blue bar (left) represents the number of false negatives before any modifications were made. The red bar (center) represents the number of files that were able to evade. The tan bar (right) represents the number of files that were still classified correctly after 10 modifications. Higher is better for the tan bar, and lower is better for all others.

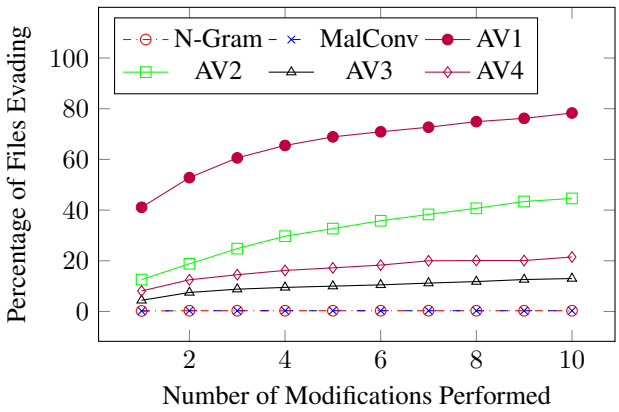

Figure 2. The number of files that evade increase as the number of modifications increases. Lower is better.

The anti-virus products did not perform as well in this test. AV3 and AV4 were the most robust to these modifications, with 130 and 215 files evading each respectively. While this

is better than products like AV1, where 783 out of 1000 malicious files where able to evade, it is still an order of magnitude worse than the 3 and 20 files that could evade the n-gram and MalConv models, respectively.

Given these results, one might wonder — could the ML based models be evaded by this approach if more modifications were performed? In particular, a number of the modifications have multiple results for a single action. So re-applying them is not unreasonable. To answer this question, we look at the evasion rate as a function of the number of modifications performed in Figure 2. In this plot we can see that all four AV products have an upward trending slope as the number of modifications is increased. In contrast, the n-gram model is completely unchanged from 3 or more modifications, and MalConv at 4 or more. This would appear to indicate that the ML approaches are truly immune to this evasion technique, whereas the AVs are not.

These results also speak to the use of dynamic evaluation, or lack thereof, in the AV products at scan time. Any dynamic analysis based approach should be intrinsically immune to the modifications performed in this test. Because we see all AV products fail to detect 13-78% of modified malware, we can postulate that if any dynamic analysis is being done its application is ineffectual.

We make note that in the original work of Anderson, Filar, and Roth (2017), they applied this attack to a simple machine learning based model that used only features from the PE header. This attack was able to evade the PE-feature based model, but not the byte based models tested under our framework.

5.2 Targeted Occlusion

The experiment detailed in subsection 4.2 attempts to modify the most important region for detecting the maliciousness of a binary as deduced from querying only the n-gram model and attempting to transfer that knowledge to all others.

The results for the byte occlusion tests are given in Figure 3, where the blue bar shows the accuracy on 40,000 malicious files when no bytes are occluded. For both the n-gram and MalConv models, we see that the Random, Undirected, and Adversarial occlusion attacks have almost no impact on classification accuracy. In the case of the n-gram model, only 0.13% of the files where able to evade its detection after occlusion. Again evasions were highly correlated with files close to the decision boundary.

In particular, we remind the reader that the Adversarial occlusion is replacing up to 2KB of the malicious file with bytes taken from benign training data. This is designed to maximally impact the ML approaches, as the model was trained with the given bytes as having an explicit label of benign. Yet, the Adversarial choice has almost no impact on results. This is a strong indicator of the potential robustness of the ML approach, and that simply adding bytes from benign programs is not sufficient to evade their detection.

Considering the AV products we again see a different story. AV1 had the highest detection rate when no occlusion occurred, but also had the largest drop in detection rate for all three types of occlusion. AV4 had the best performance

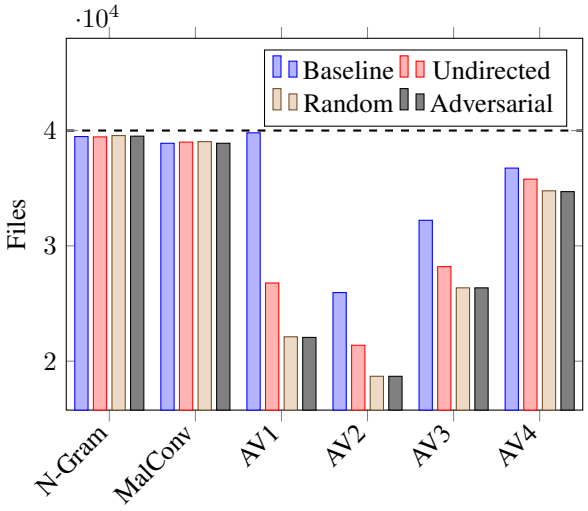

Figure 3. Robustness to byte level manipulations. Black dashed line indicates maximum. The blue bar (far left) represents number of files classified correctly before occlusion. The red bar (left center) represents the number of files classified correctly after random occlusion. The tan bar (center right) represents the number of files classified correctly after targeted occlusion using random bytes. The dark gray bar (far right) represents the number of files classified correctly after targeted occlusion using bytes from known good files.

among the AV vendors, but was still impacted by the occlusion of bytes and did not perform as strongly as the ML models.

From the AV results it is also clear that the Targeted Random occlusion is an improvement over the Undirected occlusion of random bytes, showing that Algorithm 1 is effective at identifying important regions. This was not discernible from the n-gram and MalConv models, which are relatively immune to this scenario.

Although on average the occlusion was limited to 0.5% of a file's bytes, we acknowledge that one could argue this transformation may have altered the true label of the file. The search could be potentially removing the malicious payload of the application, rendering it muted (if runnable). We accept this possibility under the assumption that the adversary now has enough information to alter this small portion of the file in order to break possible signatures.

The results from the machine learning models would suggest that maliciousness is not contained to a small contiguous section of most files. This makes intuitive sense, as malicious functionality being employed in executable sections of a file might also require functions to be referenced in the import table. The non-contiguous nature of the PE file format allows for many similar circumstances, and so we believe it unlikely that all of a file's maliciousness would be contained within a small contiguous region of a binary. In addition, this results in binaries that are analogous to deployed malware with bugs. Just because malware may not function properly, and thus not negatively impact users, doesn't remove its malicious intent and the benefit in detecting it.

5.3 ROP Injection Results

In Table 2 we show the results of running the ROPInjector on the benign test files. The results are shown only for the 13,560 files that the ROPInjector was able to infect. Before infection, the Pre-ROP Accuracy column indicates the percentage of files correctly labeled benign. After infection, the Post-ROP Accuracy column indicates what percentage were correctly labeled as malicious. The Post-ROP Lift then shows how many percentage points of Post-ROP Accuracy came from the classifier correctly determining that a formerly benign file is now malicious, rather than simply being a false-positive. That is to say, if a model incorrectly called a benign file malicious, it's not impressive when it calls an infected version of the file malicious as well.

Table 2. Accuracy on originally benign binaries before and after applying the ROPInjector.

Classifier	Pre-ROP Accuracy	Post-ROP Accuracy	Post-ROP Lift
N-Gram	85.1	15.3	0.4
MalConv	82.4	18.8	1.2
AV1	99.3	1.3	0.6
AV2	98.7	1.2	−0.1
AV3	97.9	0.7	−1.4
AV4	89.2	32.9	22.1

Most malware detectors had significant difficulty with this task. The machine learning based models, n-gram and MalConv, showed only modest improvements of 0.4 and 1.2 percentage points. AV1 had a similarly small 0.6 improvement, but surprisingly AV2 and AV3 had negative improvements. That means for 1.4% of the benign files, AV3 called the original benign file malicious. But when that same file was infected by the ROPInjector, AV3 changed its decision to benign. This is a surprising failure case, and may be caused by the AV system relying too heavily on a signature based approach (i.e., a signature had a false-positive on the benign file, but the infection process broke the signature — turning a would-be true positive into a false-negative).

Overall these models showed no major impact from the ROPInjector. Only AV4 was able to significantly adjust its decision for 22.1% of the infected files to correctly label them as malicious. Though the evasion rate for AV4 remains high at 77%, it performed best in this scenario. Given the muted and negative performance of the other AV systems in this test, we suspect AV4 has developed techniques specifically for the ROPInjector approach.

We also notice that the benign files that were injectable include 67% of our original false positives for the n-gram model. We suspect that this is due to those files already containing the networking functionality required by the shellcode. The majority of malware requires communication over the Internet, therefore our machine learning detectors may view networking related features as somewhat malicious in nature.

Overall the ROPInjection of malicious code into otherwise benign applications presents an apparent weakness for our machine learning approaches. An important question is whether the ROPInjection is functionally invisible to the machine learning based models (i.e., it could never detect features of the injection), or is simply not sufficiently represented in our training data for the model to learn.

This question was tested by applying the ROPInjector to all of our training data, both benign and malicious files, which resulted in 235,865 total ROPInjected binaries. We then trained an n-gram model that tried to distinguish between ROPInjected vs Not-ROPInjected binaries. The n-gram model was able to obtain 97.6% accuracy at this task with an AUC of 99.6%. This indicates that the models could learn to detect ROPInjection, but that it is not sufficiently prevalent in our training corpus for the models to have learned.

Overall this case highlights a potential advantage of the classical AV systems with extensive domain knowledge. When sufficiently novel attacks are developed for which current models have almost no predictive power, it may be easier to develop and deploy new signatures to catch the attacks — while the machine learning approaches may require waiting for data, or creating synthetic data (which has its own risks) to adjust the model.

6 Conclusion

We have demonstrated a new testing methodology for comparing the robustness of machine learning classifiers to current anti-virus software. Furthermore, we have provided evidence that machine learning approaches may be more successful at catching malware that has been manipulated in an attempt to evade detection. Anti-virus products do a good job of catching known and static malicious files, but their rigid decision boundaries prevent them from generalizing to new threats or catching evolutionary malware. We demonstrated that top tier anti-virus products can be fooled by simple modifications including changing a few bytes or importing random functions. The machine learning models appear to better generalize maliciousness — leading to an increased robustness to evasion techniques compared to their anti-virus counterparts.

References

[2013] Anderson, R.; Barton, C.; Böhme, R.; Clayton, R.; van Eeten, M. J. G.; Levi, M.; Moore, T.; and Savage, S. 2013. *Measuring the Cost of Cybercrime*. Berlin, Heidelberg: Springer Berlin Heidelberg. 265–300.

[2017] Anderson, H. S.; Filar, B.; and Roth, P. 2017. Evading Machine Learning Malware Detection. *Black Hat USA*.

[2010] Barreno, M.; Nelson, B.; Joseph, A. D.; and Tygar, J. D. 2010. The security of machine learning. *Machine Learning* 81(2):121–148.

[2014] Biggio, B.; Fumera, G.; and Roli, F. 2014. Security evaluation of pattern classifiers under attack. *IEEE Transactions on Knowledge and Data Engineering* 26(4):984–996.

[2016] Deo, A.; Dash, S. K.; Suarez-Tangil, G.; Vovk, V.; and Cavallaro, L. 2016. Prescience: Probabilistic Guidance on the Retraining Conundrum for Malware Detection. In *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security*, AISec '16, 71–82. New York, NY, USA: ACM.

[2017] Egele, M.; Scholte, T.; Kirda, E.; and Barbara, S. 2017. A Survey On Automated Dynamic Malware Analysis Eva-

sion and Counter-Evasion. In *Proceedings of Reversing and Offensive-oriented Trends Symposium*.

[2016] Grosse, K.; Papernot, N.; Manoharan, P.; Backes, M.; and McDaniel, P. D. 2016. Adversarial perturbations against deep neural networks for malware classification. *CoRR* abs/1606.04435.

[2013] Hyman, P. 2013. Cybercrime. *Communications of the ACM* 56(3):18.

[2016] Jordaney, R.; Wang, Z.; Papini, D.; Nouretdinov, I.; and Cavallaro, L. 2016. Misleading Metrics : On Evaluating Machine Learning for Malware with Confidence. Technical report, University of London.

[2017] Jordaney, R.; Sharad, K.; Dash, S. K.; Wang, Z.; Papini, D.; Nouretdinov, I.; and Cavallaro, L. 2017. Transcend: Detecting Concept Drift in Malware Classification Models. In *26th USENIX Security Symposium (USENIX Security 17)*, 625–642. Vancouver, BC: {USENIX} Association.

[2013] Kantchelian, A.; Afroz, S.; Huang, L.; Islam, A. C.; Miller, B.; Tschantz, M. C.; Greenstadt, R.; Joseph, A. D.; and Tygar, J. D. 2013. Approaches to Adversarial Drift. In *Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security*, AISec '13, 99–110. New York, NY, USA: ACM.

[2018] Kolosnjaji, B.; Demontis, A.; Biggio, B.; Maiorca, D.; Giacinto, G.; Eckert, C.; and Roli, F. 2018. Adversarial Malware Binaries: Evading Deep Learning for Malware Detection in Executables.

[2006] Kolter, J. Z., and Maloof, M. A. 2006. Learning to Detect and Classify Malicious Executables in the Wild. *Journal of Machine Learning Research* 7:2721–2744.

[2018] Kreuk, F.; Barak, A.; Aviv-Reuven, S.; Baruch, M.; Pinkas, B.; and Keshet, J. 2018. Adversarial Examples on Discrete Sequences for Beating Whole-Binary Malware Detection.

[2017] Li, B.; Roundy, K.; Gates, C.; and Vorobeychik, Y. 2017. Large-Scale Identification of Malicious Singleton Files. *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy - CODASPY '17* 227–238.

[2016] Miller, B.; Kantchelian, A.; Tschantz, M. C.; Afroz, S.; Bachwani, R.; Faizullabhoy, R.; Huang, L.; Shankar, V.; Wu, T.; Yiu, G.; Joseph, A. D.; and Tygar, J. D. 2016. Reviewer Integration and Performance Measurement for Malware Detection. In *Proceedings of the 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment - Volume 9721*, DIMVA 2016, 122–141. New York, NY, USA: Springer-Verlag New York, Inc.

[2008] Moskovitch, R.; Feher, C.; Tzachar, N.; Berger, E.; Gitelman, M.; Dolev, S.; and Elovici, Y. 2008. Unknown Malcode Detection Using OPCODE Representation. In *Proceedings of the 1st European Conference on Intelligence and Security Informatics*, EuroISI '08, 204–215. Berlin, Heidelberg: Springer-Verlag.

[2016] Papernot, N.; McDaniel, P. D.; and Goodfellow, I. J. 2016. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *CoRR* abs/1605.07277.

[2015] Poulios, G.; Ntantogian, C.; and Xenakis, C. 2015. ROPInjector: Using Return- Oriented Programming for Polymorphism and AV Evasion. In *Black Hat USA*.

[2017] Raff, E., and Nicholas, C. 2017. Malware Classification and Class Imbalance via Stochastic Hashed LZJD. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, AISec '17, 111–120. New York, NY, USA: ACM.

[2016] Raff, E.; Zak, R.; Cox, R.; Sylvester, J.; Yacci, P.; Ward, R.; Tracy, A.; McLean, M.; and Nicholas, C. 2016. An investigation of byte n-gram features for malware classification. *Journal of Computer Virology and Hacking Techniques*.

[2017] Raff, E.; Barker, J.; Sylvester, J.; Brandon, R.; Catanzaro, B.; and Nicholas, C. 2017. Malware Detection by Eating a Whole EXE. *arXiv preprint arXiv:1710.09435*.

[2012] Rossow, C.; Dietrich, C. J.; Grier, C.; Kreibich, C.; Paxson, V.; Pohlmann, N.; Bos, H.; and van Steen, M. 2012. Prudent Practices for Designing Malware Experiments: Status Quo and Outlook. In *2012 IEEE Symposium on Security and Privacy*, 65–79. IEEE.

[2016] Russu, P.; Demontis, A.; Biggio, B.; Fumera, G.; and Roli, F. 2016. Secure kernel machines against evasion attacks. In *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security*, AISec '16, 59–69. New York, NY, USA: ACM.

[2001] Schultz, M.; Eskin, E.; Zadok, F.; and Stolfo, S. 2001. Data Mining Methods for Detection of New Malicious Executables. In *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001*, 38–49. IEEE Comput. Soc.

[2009] Shafiq, M. Z.; Tabish, S. M.; Mirza, F.; and Farooq, M. 2009. PE-Miner: Mining Structural Information to Detect Malicious Executables in Realtime. In *Recent Advances in Intrusion Detection*. 121–141.

[2012] Singh, A.; Walenstein, A.; and Lakhotia, A. 2012. Tracking Concept Drift in Malware Families. In *Proceedings of the 5th ACM Workshop on Security and Artificial Intelligence*, AISec '12, 81–92. New York, NY, USA: ACM.

[2010] Sommer, R., and Paxson, V. 2010. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, SP '10, 305–316. Washington, DC, USA: IEEE Computer Society.

[2014] Spafford, E. C. 2014. Is Anti-virus Really Dead? *Computers & Security* 44:iv.

[2014] Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; and Fergus, R. 2014. Intriguing properties of neural networks. In *ICLR*.

[2016] Wressnegger, C.; Freeman, K.; Yamaguchi, F.; and Rieck, K. 2016. From Malware Signatures to Anti-Virus Assisted Attacks.

[2014] Zeiler, M. D., and Fergus, R. 2014. Visualizing and understanding convolutional networks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8689 LNCS(PART 1):818–833.