Shallow Security: on the Creation of Adversarial Variants to Evade Machine Learning-Based Malware Detectors

Fabrício Ceschin
fjoceschin@inf.ufpr.br
Federal University of Paraná, Brazil

Marcus Botacin
mfbotacin@inf.ufpr.br
Federal University of Paraná, Brazil

Heitor Murilo Gomes
heitor.gomes@waikato.ac.nz
University of Waikato, New Zealand

Luiz S. Oliveira
lesoliveira@inf.ufpr.br
Federal University of Paraná, Brazil

André Grégio
gregio@inf.ufpr.br
Federal University of Paraná, Brazil

ABSTRACT

The use of Machine Learning (ML) techniques for malware detection has been a trend in the last two decades. More recently, researchers started to investigate adversarial approaches to bypass these ML-based malware detectors. Adversarial attacks became so popular that a large Internet company has launched a public challenge to encourage researchers to bypass their (three) ML-based static malware detectors. Our research group teamed to participate in this challenge in August/2019, accomplishing the bypass of all 150 tests proposed by the company. To do so, we implemented an automatic exploitation method which moves the original malware binary sections to resources and includes new chunks of data to it to create adversarial samples that not only bypassed their ML detectors, but also real AV engines as well (with a lower detection rate than the original samples). In this paper, we detail our methodological approach to overcome the challenge and report our findings. With these results, we expect to contribute with the community and provide better understanding on ML-based detectors weaknesses. We also pinpoint future research directions toward the development of more robust malware detectors against adversarial machine learning.

ACM Reference Format:
Fabrício Ceschin, Marcus Botacin, Heitor Murilo Gomes, Luiz S. Oliveira, and André Grégio. 2019. Shallow Security: on the Creation of Adversarial Variants to Evade Machine Learning-Based Malware Detectors. In *Proceedings of ACM ROOTS (ROOTS 19).* ACM, New York, NY, USA, 9 pages.

1 INTRODUCTION

Malware detection is an ever growing research field due to the challenges imposed by constant evolving threats. The current state-of-the-art techniques for malware detection are Machine Learning (ML)-based approaches. However, they are not perfect and still have breaches to be exploited by attackers [15], despite solving many previously existing problems, such as the classification of dense volume of malware data. Therefore, the use of ML stimulated the already existing arms race, with attackers generating malware variants to exploit the drawbacks of ML-based approaches and defenders developing new classification models.

Adversarial attacks against machines learning models have become so popular to the point of an Internet company (Endgame, Inc) launching a challenge [10] in August/2019 to evaluate the resistance of three static analysis-based ML models against malware variants. Two of these models are deep neural networks which use raw data as input, while the last model is a decision tree which uses file metadata as input. Our team participated in this challenge and was able to bypass the three models using modified versions of the 50 samples originally provided by the organizers. To generate the adversarial malware, we implemented an automatic exploitation method moving the original malware binary sections to resources and including new chunks of data to it to create adversarial samples that not only bypassed challenge's detectors, but also real AVs as well (with a lower detection rate than the original samples).

More than competing, our main goal was to investigate real models robustness against adversarial malware samples. Our experiments revealed that the models have severe weaknesses so that they can be easily bypassed by attackers motivated to exploit real systems. These findings motivated us to write this report pinpointing possible mitigation and future research work in the ML-based malware detection field.

During our evaluation of the provided models, we discovered that:

(1) ML models based on raw binary data can be bypassed by appending data to the binaries.
(2) Frequency-based ML models can be bypassed by embedding goodware strings in malware binaries.
(3) PE format-aware classifiers can be biased towards the detection of packers instead of actually learning the concept of a malicious binary.

We propose three approaches to mitigate the drawbacks we found:

(1) OS loaders should be more aware of binary inconsistencies when loading files so as to avoid loading malformed binaries derived from binary data insertion.
(2) ML models should focus more in the presence of malicious features instead on their frequency to be more resistant to data appendix.
(3) We advocate for the assessment of malware variant-robustness in the evaluation of hereafter proposed ML-based malware detectors.

In summary, our contributions are as follows:

(1) We describe our participation in a challenge to develop adversarial attacks against ML-based malware detectors.

ROOTS 19, 2019, Vienna
2019.

(2) We describe the model's weaknesses we found during our experiments.
(3) We propose measures to the development of upcoming ML-based malware detection research work.

The remainder of the paper is organized as follows: in Section 2, we describe the challenge, datasets and models; in Section 3, we show the model's weaknesses by discussing our experiments and results; in Section 4, we present how we automated the exploitation; in Section 5, we show the weaknesses identified by us and pinpoint possible mitigation; in Section 6, we present the related work; finally, we draw our conclusions in Section 7.

2 THE CHALLENGE

The challenge hereby described [10] is a competition run by an Internet company whose winner is the one to first achieve more points, up to 150. The challenge is composed of a total of 50 tasks, in which each one of the 50 distributed distinct binaries are classified by three distinct models. Each bypassed classifier for each binary accounts for 1 point. Two of the models are based on raw data classification and one is based on PE features. All models are based on static analysis feature extraction procedures. However, all submitted binaries are executed on a sandboxed environment and must produce the same Indicators of Compromise (IoCs) as the originally distributed binaries. Thus, our objective was to create adversarial malware that behave the same way the original do by moving their binary sections and appending chunks of data (goodware bytes and strings) to them.

The competition has started in August/2019 and it is still taking place. We opted to report our findings even before its finish since we already investigated all samples and gained insights that we consider important to be shared with the community. During the competition, the scoreboard (one can check the current scoreboard online [26]) has been reset many times by the organizers due to problems with the sandbox solution. After all resets, our team figured among the top-scorer participants. In spite of that, our main goal was not to win the competition but to investigate the drawbacks of a real ML model deployed for the classification of actual malware samples. We consider the investigation of third-part models more realistic than developing our own because these would be subject to our development biases.

The Dataset provided by the organizer is composed of 50 PE (Portable Executable [24]) samples of varied malware families (grouping of malware based on their common characteristics [19]) for Microsoft Windows. The variety of samples aims to ensure a diversity of implementations so as the challenge also requires diversified approaches to bypass sample's detection. Figure 1 shows malware families distribution according to VirusTotal [37] labels, normalized using AVClass [32]. In total, there are 21 malware families in which four of them have at least four samples. **Emotet** is the most prevalent family in the dataset (5 samples) and is a banking trojan malware that steals sensitive and private information (such as credit card details) by downloading or dropping other banking malware, typically started by phishing emails and spread to other devices on the network after the infection [9]. **Loki** has 4 samples in the dataset and is a malware built to steal private data (such as stored passwords, login credentials and cryptocurrency wallets) and exfiltrate it to a C&C (Command and Control) host via `HTTP POST` [21]. **Ramnit** also has 4 samples in the dataset and is a worm that can steal cookies, login credentials and files from the infected machine. It is also able to create a backdoor that allows the attackers to send all the data to the C&C server [35]. **Xtrat** also has 4 samples in the dataset and allows the attacker to interact with the victim via one or more C&C servers. Attackers are able to manage the infected machine's registry keys, files, process, servers, and also record content from any connected devices, such as webcameras and/or microphones [13]

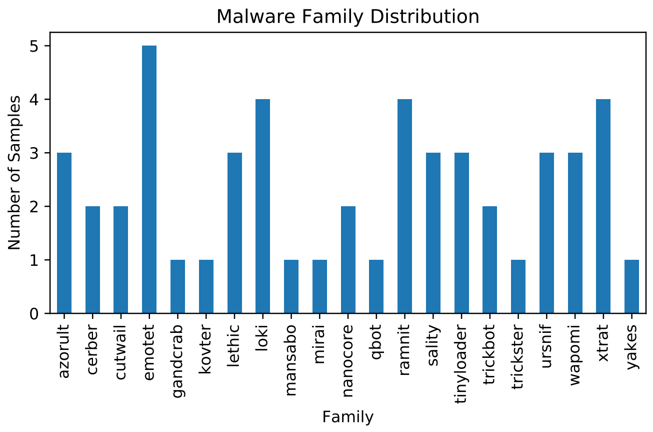

Figure 1: Malware Family Distribution. The dataset is composed of varied malware families, each one supposedly implemented in a distinct way, thus requiring distinct approaches to bypass their detection.

All samples are real malware samples that actually executed in sandboxed environments. We confirmed that by executing all samples in an available monitoring solution [3], as the challenge's organizers have not initially provided access to their sandbox's logs to the competitors.

The ML models provided by the organizers were: (i) `MalConv` [29]; (ii) `Non-Negative MalConv` [14]; and (iii) `LightGBM` [20]. All models were trained using the `Ember 2018` dataset [1], which is composed of 1.1M binary files: 900K training samples and 200K testing samples. `MalConv` is an end-to-end deep learning model, which takes as input raw bytes of a file to determine its maliciousness. The model first creates a representation of the input using an 8-dimensional embedding (which takes as input tokenized bytes). The output of this embedding is then presented to a gated 1D convolution layer, with a filter width of 500 bytes, stride of 500 and 128 filters, followed by a fully connected layer of 128 units and a softmax output for each class. `Non-Negative MalConv` has an identical structure to `MalConv`, but it has only non-negative weights, forcing the model to look only for malicious evidences rather than looking for both malicious and benign ones. `LightGBM` is a gradient boosting decision tree which operates over a feature matrix. This matrix is created using hashing trick and histograms based on the inputted binary files characteristics, such as PE header information, file size, timestamp, imported libraries, strings, etc.

We noticed that the models which use raw data are very biased towards the detection of malware samples, which results in a high

False Positive Rate (FPR) when handling benign data. Table 1 shows FPR for the 448 exe and 2422 DLLs of a pristine Windows installation.

Table 1: False Positive Rate of native Windows files classification. The raw models are very biased towards the detection of malware samples.

FileType	MalConv	Non-Neg. MalConv	LightGBM
EXEs	71.21%	87.72%	0.00%
DLLs	56.40%	80.55%	0.00%

3 MODEL'S WEAKNESSES

We conducted a series of experiments to identify model's weaknesses before starting bypassing the models. In this section, we present the conducted experiments and discuss their results.

Appending random data to the binaries might be a successful strategy to bypass ML models based on raw binary inputs. We evaluated this hypothesis by repeatedly generating growing chunks of random data, up to the limit of 5MB defined by the challenge's organizers, and testing the resulting binaries' detection by all models.

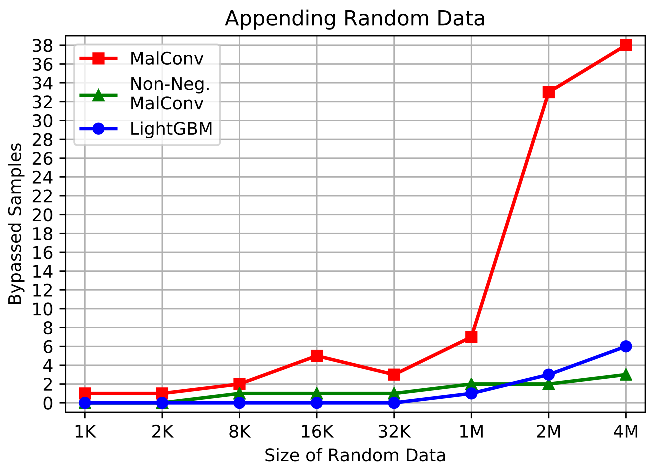

Figure 2: Models FN after appending random data to malware binaries. ML models based on raw data are susceptible to be evaded by this technique.

Figure 2 presents the false negatives of the multiple models for the distinct random data chunks. We notice that some models, noticeably the original Malconv, are more susceptible to this strategy than others. It highlights the need of developing and evaluating distinct classifiers and features for the same task, since some might be more robust than others. We also discovered that the effects are more severe for data chunks greater than 1MB. This type of evaluation is important to understand the limits of the proposed solutions. Finally, we notice that the model based on the PE structure is the less affected by the appending of unrelated data.

Appending goodware strings to malware binaries might be a successful strategy to bypass classifiers based on features frequency. We evaluated this hypothesis by repeatedly retrieving strings presented by goodware files and appending them to the malware binaries before submitting them to all ML models.

Figure 3 presents the FP rate of each ML model according to the number of appended goodware strings. We notice that as for the previous case, each model is affected by this technique in a distinct manner. However, all models are significantly affected when 10K+ strings are appended. This result holds true even for the model that also considers PE data.

Changing binary headers might be a successful strategy to bypass classifiers based on PE features. We evaluated this hypothesis by replacing some header fields of malware binaries with the values of header fields of goodware binaries. This replacement is enabled by the project decision took by Microsoft when implementing the Windows' binary loader: it silently ignores some PE binary fields [12], such as version numbers and checksums, thus allowing even corrupted binaries to run.

We replaced all binaries automatically by developing a Python script powered by the PEFile library [11], although the modification could be manually performed by using any hexadecimal editor, such as hteditor [31].

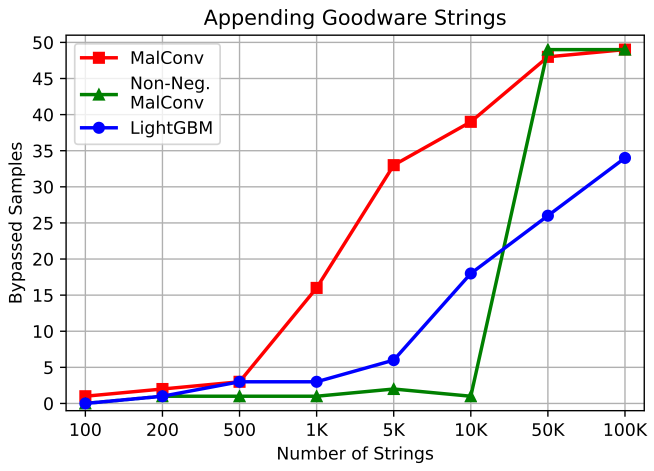

Figure 3: Models FN after appending goodware strings to binaries. All models are significantly affected by this technique.

```
# open base gw
base_pe = pefile.PE(GWR_BASE)
# iterate over output samples, changing their PE
    HEADER
for m in adv_list:
    # open adversarial sample
    m_pe = pefile.PE(m)
    # update adversarial header
    m_pe.OPTIONAL_HEADER.MajorLinkerVersion =
         base_pe.OPTIONAL_HEADER.MajorLinkerVersion
    m_pe.OPTIONAL_HEADER.MinorLinkerVersion =
         base_pe.OPTIONAL_HEADER.MinorLinkerVersion
    m_pe.OPTIONAL_HEADER.CheckSum = base_pe.
         OPTIONAL_HEADER.CheckSum
    m_pe.OPTIONAL_HEADER.MajorOperatingSystemVersion
          = base_pe.OPTIONAL_HEADER.
         MajorOperatingSystemVersion
    m_pe.OPTIONAL_HEADER.MinorOperatingSystemVersion
          = base_pe.OPTIONAL_HEADER.
         MinorOperatingSystemVersion
    m_pe.OPTIONAL_HEADER.MajorImageVersion = base_pe
         .OPTIONAL_HEADER.MajorImageVersion
    m_pe.OPTIONAL_HEADER.MinorImageVersion = base_pe
         .OPTIONAL_HEADER.MinorImageVersion
    # write updated sample
    m_pe.write(m)
```

Code Snippet 1: PE header modification script. Modifying header values did not significantly increased the FP rate.

Code Snippet 1 illustrates the script operation and which fields were modified. This approach leads to the overall bypass of only six samples in all models. This result suggests that the model based on PE features learned other characteristics than the header values.

Packing and Unpacking samples with UPX, the popular open-source packer [23], might be a successful strategy to bypass classifiers based on general PE features other than the headers. This approach might succeed on bypassing models as the UPX solution compresses entire PE into other PE sections, changing the external PE binary's aspect. We evaluated this hypothesis by packing and unpacking the provided binary samples.

The obtained results showed that classifiers were easily bypassed when strings were appended to the UPX-extracted payloads but not when directly appended to the UPX-packed payloads. This result suggested that the distributed model presented a bias against the UPX packer, since any file packed with UPX was classified as malicious. We evaluated this possibility in an exploratory fashion by randomly picking 150 UPX-packed and 150 non-packed samples from the `malshare` database [22] and classifying them using the provided models.

Table 2: UPX-Packed Samples Detection. Results suggest that models might have a bias against UPX-packed samples.

Dataset	MalConv	Non-Neg MalConv	LightGBM
Originally Packed			
UPX	63.64%	55.37%	89.26%
Extracted UPX	59.50%	53.72%	66.12%
Originally Non-Packed			
Original	65.35%	54.77%	67.23%
UPX Packed	67.43%	56.43%	88.12%

Table 2 shows the overall detection results for the samples originally packed with UPX, their extracted version, the originally unpacked samples, and their packed version. We notice that the UPX-packed versions are more detected by all classifiers than the unpacked versions, thus suggesting that the classifiers might be biased towards the detection of UPX binaries, despite their content. In summary, although we were able to bypass one of the UPX-packed binaries provided by the challenge via packer extraction and string appendix, we decided to investigate additional approaches that operate in context for which the classifiers are less biased.

Packing samples with a distinct packer might be a successful strategy to bypass classifiers that present a bias against the popular UPX. We evaluated this hypothesis by packing the provided samples with `TeLock` [36]. This strategy might succeed because this packer compresses and encrypts the original binary sections into a new one so as the original content cannot be identified by the classifiers. This strategy has proven to be effective on bypassing all models when combined with data appendix. However, some samples such as the ones from the `Extreme RAT` family [13] do not execute properly when packed with this solution. Therefore, we opted to investigate more alternatives.

Embedding samples in a dropper [17, 18] might be a successful strategy to bypass classifiers and keep samples execution properly. This strategy might succeed because despite embedding the binary in a new section, this one is not encrypted nor compressed, avoiding unpacking issues. We evaluated this hypothesis by embedding the provided samples in the `Dr0p1t` dropper [7]. This approach along data appendix has proven to be able to bypass all detectors without breaking sample's execution. However, this solution generates binaries greater than 5MB, incompatible with the challenge's rules.

4 AUTOMATIC EXPLOITATION

In this section, we present how we automated the exploitation of the weaknesses previously identified in the ML models and also present their impact on real AV detection engines.

Automating Models Exploitation. Once we discovered the major weaknesses of the provided models, we automated the procedure of generating malware variants able to bypass their detection.

In particular, our previous model's analyses revealed that:

(1) Some samples, such as RATs, do not work well when data is directly appended.
(2) `LightGBM` model detection is activated when unusual headers and sections are present.
(3) `LightGBM` model can be bypassed by packing and/or embedding the original binary within a dropper with standard header and sections.
(4) Appending data to packed and embedded code samples allows bypassing the `Malconv`-based models without negatively affecting the dropped code execution.

Based on these findings, we automated the process of packing/embedding all payloads within a new file, with standard header and sections, and after appending data by using a `Python` script. As the challenge limited the maximum file size to 5MB, we were not able to use existing solutions such as `Telock` [36] and `Dr0p1t` [7] because they generated larger binary files. Instead, we implemented our own dropper.

We implemented the dropper by embedding the original malware sample as a PE binary resource [25]. Code Snippet 2 illustrates the dropper operation. It (i) retrieves a pointer to the binary resource (line 3 to 5); (ii) creates a new file to drops the resource content (line 7); (iii) drop the entire content (line 8 to 10); and (iv) launches a process based on the dropped file (line 13). We were able to bypass all challenges without breaking samples execution by using this technique.

```
1  int main(){
2    HMODULE h = GetModuleHandle(NULL);
3    HRSRC r = FindResource(h, ...);
4    HGLOBAL rc = LoadResource(h,r);
5    void* data = LockResource(rc);
6    DWORD size = SizeofResource(h,r);
7      FILE *f = fopen("dropped.exe","wb");
8      for(int i=0;i<size;i++){
9          unsigned char c1 = ((char*)data)[i];
10         fprintf(f,"%c",c1);
11     }
12     fclose(f);
13   CreateProcess("dropped.exe", ...);
```

Code Snippet 2: Malware Dropper. The original malware file is embedded as a PE binary resource at compile time and extracted in runtime.

Figure 4 shows an overview of the variant generation process, which takes an original malware (*mw*) from the dataset to generate an adversarial malware (*mw*+), as defined by Equation 1. More specifically, the original malware (*mw*) is first used as input to an embedding function (f), which generates an entirely new file with standard PE headers and section definitions to host the original malware payload as a resource. We select one or more goodware gw_i from the set of all n goodware samples available, which, in our case, consists of all system files from a pristine Windows installation. After that, we retrieve strings and/or bytes information of each one of the goodware samples via an extraction function (*data*). The extracted chunks $data(gw_i)$ are appended to the new file created using the function $f(mw)$ so as to ensure a bias towards the goodware class. The function outcome is an adversarial malware sample (*mw*+). One can iterate this procedure so as to consider multiple goodware samples, thus repeatedly appending data to the end of $f(mw)$. In the example shown in Table 3, the malware (*mw*), a sample from the family `xtrat` (sample number 35 on Table 4), was classified by the three models as malware, with 99.99% of confidence by `malConv` and `LightGBM`, and with 75.05% of confidence by `Non-Negative Malconv`. The `ntoskrnl.exe` goodware (gw_i) used to produce the appending data was classified as goodware with 69.54% of confidence by `MalConv`, 73.32% by `Non-Negative MalConv` and 99.99% by `LightGBM`. The resulting adversarial sample (*mw*+) completely deceived all classifiers into considering it as being a goodware with 81.21%, 98.65% and 99.99% of confidence by `MalConv`, `Non-Negative MalConv` and `LightGBM`, respectively. On average, the confidence changed from 91.68% of being a malware (*mw*) to 93.28% of being a goodware (*mw*+).

$$mw+ = f(mw) + \sum_{i}^{n} data(gw_i) \tag{1}$$

Adversarial malware in real world. After we bypassed all challenge's models, we investigated whether the strategies we deployed for experimentation purposes could also be successfully leveraged in practice by actual attackers. To do so, we submitted the original and the modified samples to the VirusTotal service and retrieved the overall detection rates, as shown in Figure 5.

We noticed that our approach also affected real AV engines; the retrieved detection rates were smaller for all samples, in some cases (sample 6) dropping almost in half. This result is explained by current AV engines also being powered by ML models, which might be subject to the same weaknesses and biases that we identified for the challenge's models. This result highlights the need of developing more robust ML models and features for malware variants detection.

A practical drawback of generating adversarial malware samples is that their binaries become larger than the original ones due to additional data appended to the original malware. The appended data is not even used by the malware, but must be there to create a bias towards goodware class. Figure 6 presents a comparison between original and ML-evasive samples per malware family. We discovered that adversarial malware (whose maximum size is around 5MB due to the limits imposed by the challenge organizers) are, in general, at least around twice the size of original ones (whose maximum size is around 1.5MB).

5 DISCUSSION

Bypassing a detector means that one reasoned about an implementation and/or technology and identified weaknesses. In this section, we present the weaknesses identified by our team and pinpoint possible mitigation.

Susceptibility to appended data is a major weakness of raw models. In most cases, this simple strategy was enough to defeat the two raw data-based models, indicating that the concept learned by these models is not robust enough against adversarial attacks.

Appending data affects detection but not PE loading, since the Windows loader ignores some PE fields and even resolve conflicting sizes and checksums in runtime [12]. This lax policy allows attackers to append content to the binaries without affecting its functionalities. We advocate for more strict loading policies so as to mitigate the impact of this type of bypass technique. For instance, the loader should check if a binary has more sections than declared in its header and/or if the section content exceeds the boundaries defined by the section size header field.

Adversarial malware are much bigger than original ones due to additional data appended to them, which are needed to bypass classifiers, such as strings and bytes. These data create a bias towards goodware class but also make their size greater, which can make it difficult for attackers to distribute them for new victims. We consider this as a challenge to be considered by any attacker, which needs to create an adversarial sample with the minimum size as possible.

Developing models based on the presence of features instead on their frequency is an strategy that should be adopted to mitigate the impact of appended data on classification models. We discovered that most classifiers changed their decision from malware to goodware when goodware strings were added to the binary, thus masking the impact of malware strings in the overall model. The use of pertinence models mitigate this effect as malicious strings need to be still present in the binaries to keep its functional.

Domain-specific models might present biases and not effectively learn a concept. We discovered that the model based on PE

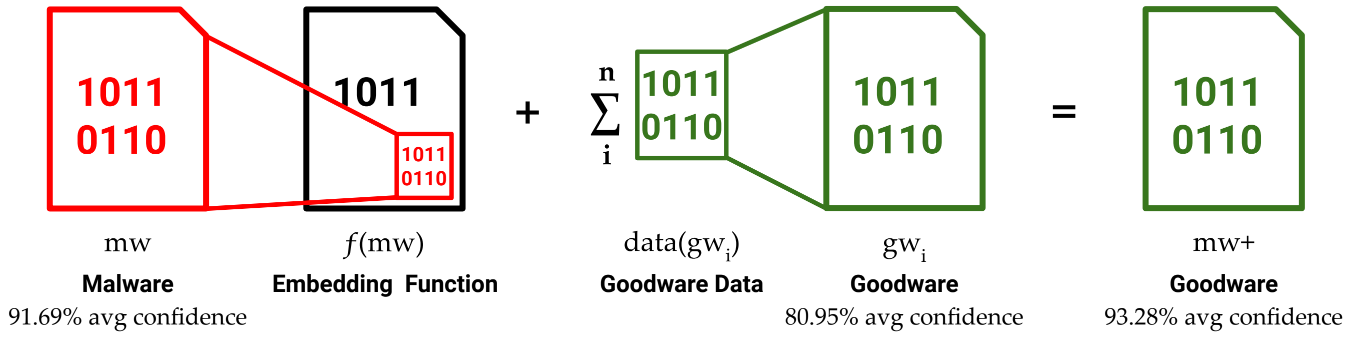

Figure 4: Adversarial malware generation. By just using an embedding function to add malware payloads within a new file and adding goodware data to it, such as strings and bytes from a set of goodware, we can change classifiers' output.

Table 3: Models' confidence when classifying a malware (*mw*), a goodware (*gw*$_i$) and an adversarial malware (*mw*$_i$). Results show that the adversarial sample is classified as goodware with higher confidence than a real goodware.

	Malware (*mw*)		Goodware (*gw*$_i$)		Adversarial Malware (*mw+*)	
Model	**Classification**	**Confidence**	**Classification**	**Confidence**	**Classification**	**Confidence**
MalConv	Malware	99.99%	Goodware	69.54%	Goodware	81.22%
Non-Neg. MalConv	Malware	75.09%	Goodware	73.32%	Goodware	98.65%
LightGBM	Malware	100.00%	Goodware	99.99%	Goodware	99.97%
Average	Malware	91.69%	Goodware	80.95%	Goodware	93.28%

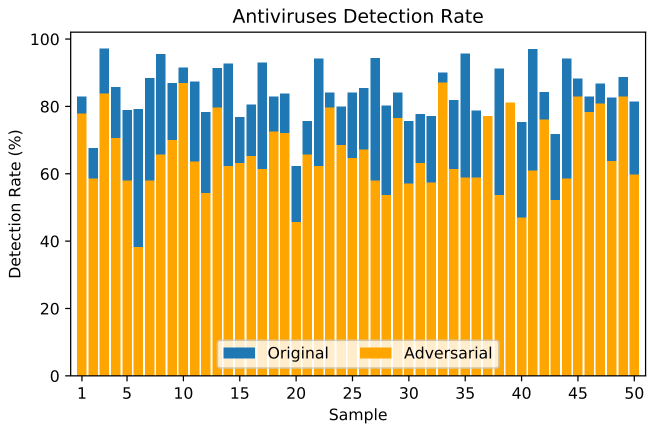

Figure 5: Samples Detection Rate. The developed malware variant samples were also less detected by the Virustotal's AV engines in addition to bypassing the challenge's models.

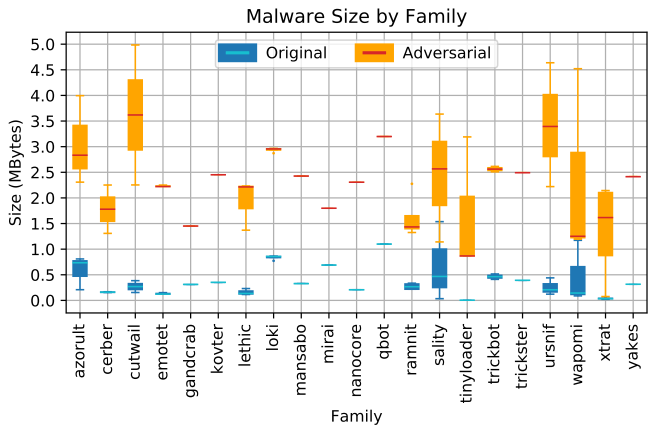

Figure 6: Comparison between original and adversarial malware size in MBytes by malware family. Adversarial samples are much bigger than the original ones due to the additional data used to bypass classifiers.

binaries features presented a bias against the UPX packer, used by most malware but also present in benign software. Packing benign software with UPX revealed that the detector learned to mistakenly always flag UPX binaries as malicious.

Adopting malware variants robustness as a criteria to assess ML-based detectors is an essential step to moving forward the malware detection field. Our results revealed that even deep learning models might be easily bypassed, which makes them less effective in practice. We advocate for the adoption of variants robustness testing as a criteria for assessing the further developed malware classifiers so as to contribute for more real world-targeted solutions. This criteria might be included in the process of correct evaluating a malware detector, which already includes handling concept drift and evolution, class imbalance, degradation, etc [5, 8, 28]

Creating a robust representation is another essential step for malware detection field, given that attackers might include goodware characteristics into their malware to evade any model. Thus, creating a representation that is invariant to these characteristics is fundamental to avoid adversarial malware.

Checking file resources and embedded PE files should be part of ML feature extraction procedures. It would allow classifiers to detect embedded malicious payload instead of being easily deceived by malware droppers.

Converting samples into downloaders [30] is also a successful strategy to bypass static detectors. In this case, the malicious payload is retrieved from the Internet whereas the undetected loader is submitted to ML scan. This shows that defenders should not only focus in the classifier accuracy rate but also should reason about the whole threat model to cover all attack possibilities. We implemented downloader versions of all malware samples provided by the challenge organizers, but we did not submit them to the challenge validation system because the challenge's sandbox solutions was network-isolated.

Generating adversarial samples to malware detector is a particular case of adversarial attacks, that can be performed against multiple domains and targets, such as image classifiers [15]. Despite of having the same goal of bypassing a classification, different techniques are required depending on the problem domain. For some image classifiers, for example, adversarial images should look similar to the original ones (usually being indistinguishable to the human eye). In contrast, for malware detectors, adversarial malware only need to perform the same action as the original ones, even if they are completely different in their structures, as we did by embedding the original malware into a dropper. Therefore, simply adding a noise to a malware (as done in adversarial images [15]) might generate an invalid adversarial malware that does not work or does not perform the same action as the original one.

6 RELATED WORK

In this section, we present related work that provide insights about malware evasion techniques and countermeasures. We believe that these related work might be useful in future community's research work, although they are not directly related to the described malware evasion challenge.

Adversarial Machine Learning are techniques aimed to defeat ML models. These techniques can rely on manual, heuristics or even ML-based strategies. A noticeable example of the latter are Generative Adversarial Networks (GANs). GANs are 2-part, coupled deep learning systems in which one part is trained to classify the inputs generated by the other. The two parts simultaneously try to maximize their performance. As a side effect, the input generator learns the best way to defeat its corresponding classifier. In this sense, GANs have been successfully used to create adversarial samples to bypass malware detectors [16]. In some cases, even proposed defensive solutions, such as the use of defensive distillation, a procedure to train deep neural networks that are more robust to perturbations [27], are prone to adversarial machine learning attacks (in this case, just by using different attack algorithms [4]). We believe that the use GANs will be the standard approach to solve future challenges such as the one that our team participated due to their classification power and scalability to large-scale datasets.

Binary Mutations were the so-far prevalent techniques to generate malware variants. These techniques implement transformations such as code replacement, instruction swapping, variable changes, dead code insertion and control flow obfuscation to bypass malware detectors [2]. These techniques can also be applied In the context of this work. However, they are now targeting implicit neural-network models instead of the previously clear-defined behavioral [6] or Intermediate Representation (IR)-defined [33] models.

Automated Binary Exploitation are techniques to automatically discover flaws and binaries and exploit them. These type of techniques have also been the subject of public challenges (e.g., DARPa challenge [34]). We believe that this type of technique can also be applied for malware detection evasion, despite being originally designed for a distinct purpose. The automatic identification of a critical execution path might indicate potential binary regions to be patched by an attacker.

7 CONCLUSION

In this paper, we reported the participation of our team in a public challenge launched by an Internet company in August/2019 to develop adversarial attacks against Machine Learning-based malware detectors. It challenged the participants to simultaneously bypass three distinct static analysis-based ML models. Our team developed custom binaries able to bypass all 150 challenges. We discovered that models leveraging raw binary data are easily evaded by appending additional data to the original binary files. We also discovered that models based on the Windows PE file structure learn malicious section names as suspicious, such that these detectors can be bypassed by replacing section names. We suggest the adoption of malware variant-resilience testing as an additional criteria for the evaluation and assessment of future developments of ML-based malware detectors. This ensures that the developed detectors can be applied to actual scenarios without the risk of being easily bypassed by attackers.

Reproducibility. The source code of the developed prototype to embed malware samples into unsuspicious binaries was released as open source and it is available on `github`: https://github.com/marcusbotacin/Dropper. All analysis reports of evasive and non-evasive samples execution and their similarities are available on the `corvus` platform: https://corvus.inf.ufpr.br/.

Acknowledgments. Fabrício would like to thank the Coordination for the Improvement of Higher Education Personnel (CAPES, PhD Scholarship, process 88882.382195/2019-01). Marcus would like to thank the the Brazilian National Counsel of Technological and Scientific Development (CNPq, PhD Scholarship, process 164745/2017-3). Marcus and André would like to thank the Coordination for the Improvement of Higher Education Personnel (CAPES, Project FORTE, Forensics Sciences Program 24/2014, process 23038.007604/2014-69).

REFERENCES

[1] H. S. Anderson and P. Roth. 2018. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. *ArXiv e-prints* (April 2018). arXiv:cs.CR/1804.04637

[2] Jean-Marie Borello and Ludovic Mé. 2008. Code obfuscation techniques for metamorphic viruses. https://www.semanticscholar.org/paper/Code-obfuscation-techniques-for-metamorphic-Borello-M%C3%A9/647daaad8fce2d47ca0e2fba7e0ccbf2113146a2/pdf. *JICVHT.* (2008).

[3] Marcus Felipe Botacin, Paulo Lício de Geus, and André Ricardo Abed Grégio. 2018. The other guys: automated analysis of marginalized malware. *Journal of*

Computer Virology and Hacking Techniques 14, 1 (01 Feb 2018), 87–98. https://doi.org/10.1007/s11416-017-0292-8

[4] N. Carlini and D. Wagner. 2017. Towards Evaluating the Robustness of Neural Networks. In *2017 IEEE Symposium on Security and Privacy (SP)*. 39–57. https://doi.org/10.1109/SP.2017.49

[5] F. Ceschin, F. Pinage, M. Castilho, D. Menotti, L. S. Oliveira, and A. Gregio. 2018. The Need for Speed: An Analysis of Brazilian Malware Classifiers. *IEEE Security Privacy* 16, 6 (Nov 2018), 31–41. https://doi.org/10.1109/MSEC.2018.2875369

[6] M. Christodorescu, S. Jha, S. A. Seshia, D. Song, and R. E. Bryant. 2005. Semantics-aware malware detection. In *2005 IEEE Sec. & Priv.* 32–46. https://doi.org/10.1109/SP.2005.20

[7] D4Vinci. 2017. Dr0p1t. https://github.com/D4Vinci/Dr0p1t-Framework.

[8] Disha Dasgupta. 2018. Beware Steep Decline: Understanding Model Degradation In Machine Learning Models. https://www.endgame.com/blog/technical-blog/beware-steep-decline-understanding-model-degradation-machine-learning-models.

[9] Michelle Drolet. 2019. What is Emotet? And how to guard against this persistent Trojan malware. https://www.csoonline.com/article/3387146/what-is-emotet-and-how-to-guard-against-this-persistent-trojan-malware.html.

[10] EndGame. 2019. Machine Learning Static Evasion Competition. https://www.endgame.com/blog/technical-blog/machine-learning-static-evasion-competition.

[11] Erocarrera. 2019. PEfile is a Python module to read and work with PE (Portable Executable) files. https://github.com/erocarrera/pefile.

[12] Peter Ferrie. 2008. Anti-unpacker tricks. http://pferrie.tripod.com/papers/unpackers.pdf.

[13] FireEye. 2014. XtremeRAT: Nuisance or Threat? https://www.fireeye.com/blog/threat-research/2014/02/xtremerat-nuisance-or-threat.html.

[14] William Fleshman, Edward Raff, Jared Sylvester, Steven Forsyth, and Mark McLean. 2018. Non-Negative Networks Against Adversarial Attacks. https://arxiv.org/abs/1806.06108.

[15] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and Harnessing Adversarial Examples. *arXiv e-prints*, Article arXiv:1412.6572 (Dec 2014), arXiv:1412.6572 pages. arXiv:stat.ML/1412.6572

[16] Weiwei Hu and Ying Tan. 2017. Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN. *CoRR* abs/1702.05983 (2017). arXiv:1702.05983 http://arxiv.org/abs/1702.05983

[17] Min Gyung Kang, Pongsin Poosankam, and Heng Yin. 2007. Renovo: A Hidden Code Extractor for Packed Executables. In *Proceedings of the 2007 ACM Workshop on Recurring Malcode (WORM '07)*. ACM, New York, NY, USA, 46–53. https://doi.org/10.1145/1314389.1314399

[18] Bum Jun Kwon, Jayanta Mondal, Jiyong Jang, Leyla Bilge, and Tudor Dumitraş. 2015. The Dropper Effect: Insights into Malware Distribution with Downloader Graph Analytics. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15)*. ACM, New York, NY, USA, 1118–1129. https://doi.org/10.1145/2810103.2813724

[19] Beth Levin, Andres Mariano Gorzelany, Daniel Simpson, and Dani Halfin. 2019. Malware names. https://docs.microsoft.com/en-us/windows/security/threat-protection/intelligence/malware-naming.

[20] LightGBM. 2018. LightGBM. https://lightgbm.readthedocs.io/en/latest/.

[21] Malpedia. 2019. Loki Password Stealer (PWS). https://malpedia.caad.fkie.fraunhofer.de/details/win.lokipws.

[22] Malshare. 2019. A free Malware repository providing researchers access to samples, malicous feeds, and Yara results. https://malshare.com/.

[23] J. F. Reiser M.F.X.J. Oberhumer, László Molnár. 2018. UPX the Ultimate Packer for eXecutables. https://upx.github.io/.

[24] Microsoft. 2017. Peering Inside the PE: A Tour of the Win32 Portable Executable File Format. http://bytepointer.com/resources/pietrek_peering_inside_pe.htm.

[25] Microsoft. 2018. Resource Files (C++). https://docs.microsoft.com/en-us/cpp/windows/resource-files-visual-studio?view=vs-2019.

[26] MLSec. 2019. Scores. https://evademalwareml.io/scores/.

[27] Nicolas Papernot, Patrick D. McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. 2015. Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks. *CoRR* abs/1511.04508 (2015). arXiv:1511.04508 http://arxiv.org/abs/1511.04508

[28] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. 2018. TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time. *CoRR* abs/1807.07838 (2018). arXiv:1807.07838 http://arxiv.org/abs/1807.07838

[29] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles Nicholas. 2017. Malware Detection by Eating a Whole EXE. https://arxiv.org/abs/1710.09435.

[30] Christian Rossow, Christian Dietrich, and Herbert Bos. 2013. Large-Scale Analysis of Malware Downloaders. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, Ulrich Flegel, Evangelos Markatos, and William Robertson (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 42–61.

[31] S. Biallas S. Weyergraf. 2015. HT Editor. http://hte.sourceforge.net/index.html.

[32] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero. 2016. AVclass: A Tool for Massive Malware Labeling. In *Research in Attacks, Intrusions, and Defenses*, Fabian Monrose, Marc Dacier, Gregory Blanc, and Joaquin Garcia-Alfaro (Eds.). Springer International Publishing, Cham, 230–253.

[33] Peng Shao and Randy K. Smith. 2009. Feature Location by IR Modules and Call Graph. In *ACM-SE 47*.

[34] Yan Shoshitaishvili, Ruoyu Wang, Christopher Salls, Nick Stephens, Mario Polino, Audrey Dutcher, John Grosen, Siji Feng, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna. 2016. SoK: (State of) The Art of War: Offensive Techniques in Binary Analysis. In *IEEE Symposium on Security and Privacy*.

[35] Symantec. 2015. W32.Ramnit. https://www.symantec.com/security-center/writeup/2010-011922-2056-99.

[36] Telock. 2018. Telock. https://www.softpedia.com/get/Programming/Packers-Crypters-Protectors/Telock.shtml.

[37] Virus Total. 2019. Virus Total. https://www.virustotal.com.

A DATASET SAMPLES CLASSIFICATION

Table 4 shows the labels assigned to each sample by each classifier and their respective confidence levels.

Table 4: Class (M for malware and G for goodware) and confidence (%) of each original and adversarial sample when classifying them using the three ML models proposed by the challenge. All adversarial samples are considered as being a goodware, the majority of them with a high confidence level.

		MalConv				Non-Neg. MalConv				LightGBM			
Sample	Family	Original		Adversarial		Original		Adversarial		Original		Adversarial	
		Class	%	Class	%	Class	%	Class	%	Class	%	Class	%
001	gandcrab	M	97.83%	G	65.96%	M	54.30%	G	90.53%	M	100.00%	G	87.83%
002	cerber	M	96.28%	G	85.34%	M	63.96%	G	98.65%	M	99.91%	G	82.83%
003	xtrat	M	99.60%	G	76.84%	M	60.81%	G	98.41%	M	100.00%	G	51.56%
004	lethic	M	95.33%	G	83.61%	M	63.33%	G	98.65%	M	100.00%	G	99.28%
005	tinyloader	M	66.92%	G	65.96%	M	59.90%	G	98.47%	M	100.00%	G	80.86%
006	tinyloader	M	64.22%	G	78.18%	M	94.85%	G	95.35%	M	100.00%	G	90.82%
007	sality	M	99.69%	G	83.25%	M	63.81%	G	98.65%	M	100.00%	G	99.98%
008	xtrat	M	99.90%	G	72.25%	M	50.27%	G	66.01%	M	100.00%	G	63.04%
009	emotet	M	99.81%	G	75.34%	M	59.95%	G	98.41%	M	100.00%	G	62.58%
010	sality	M	87.27%	G	73.72%	M	66.53%	G	98.41%	M	100.00%	G	78.61%
011	lethic	M	61.42%	G	87.97%	M	59.80%	G	98.41%	M	100.00%	G	96.74%
012	mirai	M	53.11%	G	68.05%	M	64.03%	G	98.65%	M	100.00%	G	99.91%
013	nanocore	M	59.90%	G	78.98%	M	50.03%	G	98.41%	M	100.00%	G	94.55%
014	ramnit	M	99.80%	G	87.12%	M	54.61%	G	98.65%	M	100.00%	G	99.84%
015	emotet	M	97.46%	G	73.52%	M	67.72%	G	98.41%	M	99.79%	G	99.70%
016	emotet	M	96.13%	G	81.04%	M	52.18%	G	98.41%	M	99.96%	G	99.72%
017	ramnit	M	88.61%	G	87.86%	M	66.76%	G	95.20%	M	100.00%	G	99.97%
018	emotet	M	95.95%	G	70.34%	M	51.16%	G	98.41%	M	99.73%	G	99.75%
019	trickbot	M	65.76%	G	85.18%	M	50.07%	G	97.98%	M	100.00%	G	51.74%
020	yakes	M	92.88%	G	74.48%	M	63.52%	G	98.41%	M	99.78%	G	99.93%
021	azorult	M	99.17%	G	90.95%	M	83.35%	G	81.54%	M	100.00%	G	56.41%
022	wapomi	M	97.75%	G	84.08%	M	51.15%	G	98.65%	M	100.00%	G	99.91%
023	loki	M	78.93%	G	65.96%	M	64.89%	G	98.79%	M	100.00%	G	60.90%
024	cutwail	M	98.86%	G	89.39%	M	57.83%	G	90.54%	M	99.38%	G	95.07%
025	trickster	M	67.38%	G	75.15%	M	58.97%	G	98.41%	M	100.00%	G	97.77%
026	ursnif	M	99.43%	G	76.26%	M	61.83%	G	98.78%	M	99.93%	G	99.66%
027	ramnit	M	71.84%	G	84.76%	M	66.78%	G	98.65%	M	100.00%	G	99.94%
028	lethic	M	99.71%	G	80.90%	M	51.98%	G	98.41%	M	100.00%	G	99.57%
029	loki	M	96.29%	G	67.93%	M	94.36%	G	96.67%	M	100.00%	G	59.55%
030	tinyloader	M	90.86%	G	76.63%	M	79.99%	G	95.35%	M	100.00%	G	90.60%
031	kovter	M	95.08%	G	94.44%	M	63.90%	G	97.27%	M	99.99%	G	88.78%
032	cerber	M	98.84%	G	71.65%	M	51.51%	G	98.41%	M	100.00%	G	65.99%
033	wapomi	M	74.44%	G	65.96%	M	57.32%	G	97.46%	M	100.00%	G	82.24%
034	azorult	M	61.45%	G	79.49%	M	66.28%	G	98.41%	M	100.00%	G	99.82%
035	xtrat	M	99.99%	G	81.22%	M	75.09%	G	98.65%	M	100.00%	G	99.97%
036	ursnif	M	95.56%	G	78.34%	M	57.14%	G	98.41%	M	100.00%	G	81.82%
037	loki	M	62.48%	G	65.96%	M	77.32%	G	98.41%	M	100.00%	G	64.30%
038	xtrat	M	100.00%	G	87.43%	M	59.22%	G	95.36%	M	100.00%	G	99.95%
039	loki	M	70.72%	G	79.54%	M	62.07%	G	98.41%	M	100.00%	G	50.10%
040	cutwail	M	98.79%	G	65.96%	M	50.89%	G	66.29%	M	100.00%	G	98.94%
041	ramnit	M	97.61%	G	82.01%	M	55.83%	G	98.65%	M	100.00%	G	99.91%
042	mansabo	M	74.98%	G	65.96%	M	77.01%	G	98.41%	M	100.00%	G	81.72%
043	qbot	M	87.64%	G	65.96%	M	51.17%	G	98.87%	M	100.00%	G	99.02%
044	wapomi	M	86.50%	G	82.33%	M	70.80%	G	98.65%	M	100.00%	G	99.98%
045	sality	M	58.42%	G	86.55%	M	81.19%	G	92.57%	M	100.00%	G	82.26%
046	azorult	M	99.52%	G	69.33%	M	98.83%	G	94.59%	M	100.00%	G	73.76%
047	emotet	M	98.84%	G	74.40%	M	51.48%	G	98.41%	M	100.00%	G	65.38%
048	ursnif	M	99.96%	G	93.20%	M	51.36%	G	81.54%	M	99.99%	G	76.67%
049	nanocore	G	95.83%	G	74.79%	M	50.59%	G	98.41%	M	100.00%	G	94.56%
050	trickbot	M	99.92%	G	66.87%	M	58.13%	G	98.41%	M	100.00%	G	80.62%