Exploring Adversarial Examples in Malware Detection

Octavian Suciu
University of Maryland, College Park

Scott E. Coull and **Jeffrey Johns**
FireEye, Inc.

Abstract

The Convolutional Neural Network (CNN) architecture is increasingly being applied to new domains, such as malware detection, where it is able to learn malicious behavior from raw bytes extracted from executables. These architectures reach impressive performance with no feature engineering effort involved, but their robustness against active attackers is yet to be understood. Such malware detectors could face a new attack vector in the form of adversarial interference with the classification model. Existing evasion attacks intended to cause misclassification on test-time instances, which have been extensively studied for image classifiers, are not applicable because of the input semantics that prevents arbitrary changes to the binaries. This paper explores the area of adversarial examples for malware detection. By training an existing model on a production-scale dataset, we show that some previous attacks are less effective than initially reported, while simultaneously highlighting architectural weaknesses that facilitate new attack strategies for malware classification. Finally, we explore more generalizable attack strategies that increase the potential effectiveness of evasion attacks.

Introduction

The popularity of Convolutional Neural Network (CNN) classifiers has lead to their adoption in fields which have been historically adversarial, such as malware dectection (Raff et al. 2017; Krčál et al. 2018). Recent advances in adversarial machine learning have highlighted weaknesses of classifiers when faced with adversarial samples. One such class of attacks is evasion (Biggio et al. 2013), which acts on test-time instances. The instances, also called adversarial examples, are modified by the attacker such that they are misclassified by the victim classifier even though they still resemble their original representation. State-of-the-art attacks focus mainly on image classifiers (Szegedy et al. 2013; Goodfellow, Shlens, and Szegedy 2014; Papernot et al. 2017; Carlini and Wagner 2017), where attacks add small perturbations to input pixels that lead to a large shift in the victim classifier feature space, potentially shifting it across the classification decision boundary. The perturbations do not change the semantics of the image as a human oracle easily identifies the original label associated with the image.

In the context of malware detection, adversarial examples could represent an additional attack vector for an attacker determined to evade such a system. However, domain-specific challenges limit the applicability of existing attacks designed against image classifiers on this task. First, the strict semantics of binary files disallows arbitrary perturbations in the input space. This is because there is a structural interdependence between adjacent bytes, and any change to a byte value could potentially break the functionality of the executable. Second, limited availability of representative datasets or robust public models limits the generality of existing studies. Existing attacks (Kolosnjaji et al. 2018) use victim models trained on very small datasets and avoid the semantic issues entirely by appending adversarial noise at the end of the binary files, and their generalization effectiveness is yet to be evaluated.

This paper sheds light on the generalization property of adversarial examples against CNN-based malware detectors. By training on a production-scale dataset of 12.5 million binaries, we are able to observe interesting properties of existing attacks, and propose a more effective and generalizable attack strategy. Our contributions are as follows:

- We measure the generalization of existing adversarial attacks and highlight the limitations that prevents them from being widely applicable.
- We unearth an architectural weaknesses of a published CNN architecture that facilitates the existing append-based attack by Kolosnjaji et al. (Kolosnjaji et al. 2018).
- We propose a new attack which, by modifying the existing bytes of a binary, has the potential to outperform append-based attacks without semantic inconsistencies.

Background

The Convolutional Neural Network (CNN) architecture has proven to be very successful across popular vision tasks, such as image classification (He et al. 2016). This lead to an increased adoption in other fields and domains, with one such example being text classification from character-level features (Zhang, Zhao, and LeCun 2015), which turns out to be extremely similar to the malware classification problem

 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

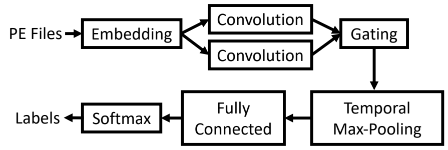

Figure 1: Architecture for the MalConv Model.

discussed in this paper. In this setting, a natural language documents is represented as a sequence of characters, and the CNN is applied on that one-dimensional stream of characters. The intuition behind this approach is that a CNN is capable of automatically learning complex features, such as words or word sequences, by observing compositions of raw signals extracted from single characters. This approach also avoids the requirement of defining language semantic rules, and is able to tolerate anomalies in features, such as word misspellings. The classification pipeline first encodes each character into a fixed-size embedding vector. The sequence of embeddings acts as input to a set of convolutional layers, intermixed with pooling layers, then followed by fully connected layers. The convolutional layers act as receptors, picking particular features from the input instance, while the pooling layers act as filters to down-sample the feature space. The fully connected layers act as a non-linear classifier on the internal feature representation of instances.

CNNs for Malware Classification. Similar to this approach, the security community explored the applicability of CNNs to the task of malware detection on binary files (Raff et al. 2017; Krčál et al. 2018). Analogous to text, a file could be conceptualized as a sequence of bytes that are arranged into higher-level features, such as instructions or functions. By allowing the classifier to automatically learn features indicative of maliciousness, this approach avoids the labor-intensive feature engineering process typical of malware classification tasks. Manual feature engineering proved to be challenging in the past and lead to an arms race between antivirus developers and attackers aiming to evade them (Ugarte-Pedrero et al. 2015). However, the robustness of these automatically learned features in the face of evasion is yet to be understood.

In this paper, we explore evasion attacks by focusing on a byte-based convolutional neural network for malware detection, called MalConv (Raff et al. 2017), whose architecture is shown in Figure 1. MalConv reads up to 2MB of raw byte values from a Portable Executable (PE) file as input, appending a distinguished padding token to files smaller than 2MB and truncating extra bytes from larger files. The fixed-length sequences are then transformed into an embedding representation, where each byte is mapped to an 8-dimensional embedding vector. These embeddings are then passed through a gated convolutional layer, followed by a temporal max-pooling layer, before being classified through a final fully connected layer. Each convolutional layer uses a kernel size of 500 bytes with a stride of 500 (i.e., non-overlapping windows), and each of the 128 filters is passed through a max-pooling layer. This results in a unique architectural feature that we will revisit in our results: each pooled filter is mapped back to a specific 500-byte sequence and there are at most 128 such sequences that contribute to the final classification across the entire input. Their reported results on a testing set of 77,349 samples achieved a Balanced Accuracy of 0.909 and Area Under the Curve (AUC) of 0.982.

Adversarial Binaries. Unlike evasion attacks on images (Szegedy et al. 2013; Goodfellow, Shlens, and Szegedy 2014; Papernot et al. 2017; Carlini and Wagner 2017), attacks that alter the raw-bytes of PE files must maintain the syntactic and semantic fidelity of the original file. The Portable Executable (PE) standard (Microsoft 2018) defines a fixed structure for these files. A PE file contains a leading header enclosing file metadata and pointers to the sections of the file, followed by the variable-length sections which contain the actual program code and data. Changing bytes arbitrarily could break the malicious functionality of the binary or, even worse, prevent it from loading at all.

Recent work (Kolosnjaji et al. 2018) avoided this problem by appending adversarial noise to the end of the binary. Since the appended adversarial bytes are not within the defined boundaries of the PE file, their existence does not impact the binary's functionality and there are no inherent restrictions on the syntax of bytes (i.e., valid instructions and parameters). The trade-off, however, is that the impact of the appended bytes on the final classification is offset by the features present in the original sample, which remain unchanged. As we will see, these attacks take advantage of certain vulnerabilities in position-independent feature detectors present in the MalConv architecture.

Datasets. To evaluate the success of evasion attacks against the MalConv architecture, we collected 16.3M PE files from a variety of sources, including VirusTotal, Reversing Labs, and proprietary FireEye data. The data was used to create a production-quality dataset of 12.5M training samples and 3.8M testing samples, which we refer to as the *Full* dataset. It contains 2.2M malware samples in the training set, and 1.2M in testing, which represents a realistic ratio of goodware to malware. The dataset was created from a larger pool of more than 33M samples using a stratified sampling technique based on VirusTotal's vhash clustering algorithm, which is based on dynamic and static properties of the binaries. Use of stratified sampling ensures uniform coverage over the canonical 'types' of binaries present in the dataset, while also limiting bias from certain overrepresented types (e.g., popular malware families). In addition, we also created a smaller dataset whose size and distribution is more in line with Kolosnjaji et al.'s evaluation (Kolosnjaji et al. 2018), which we refer to as the *Mini* dataset. The Mini dataset was created by sampling 4,000 goodware and 4,598 malware samples from the Full dataset. Note that both datasets follow a strict temporal split where test data was observed strictly later than training data. We use the Mini dataset in order to explore whether the attack results demonstrated by Kolosnjaji et al. would generalize to a production-quality model, or whether they are artifacts of the dataset properties.

Baseline Performance

To validate our implementation of the MalConv architecture (Raff et al. 2017), we train the classifier on both the Mini and the Full datasets, leaving out the DeCov regularization addition suggested by the authors. Our implementation uses a momentum-based optimizer with decay and a batch size of 80 instances. We train on the Mini dataset for 10 full epochs. We also trained the Full dataset for 10 epochs, but stopped the process early due to a small validation loss[1]. To assess and compare the performance of the two models, we test them on the entire Full testing set. The model trained on the Full dataset achieves an accuracy of 0.89 and an AUC of 0.97, which is similar to the results published in the original MalConv paper. Unsurprisingly, the Mini model is much less robust, achieving an accuracy of 0.73 and an AUC of 0.82.

Append Attacks

In this section we present various attack strategies that address the semantic integrity constraints of PE files by appending adversarial noise to the original file. We start by presenting two attacks first introduced by Kolosnjaji et al. (Kolosnjaji et al. 2018) and evaluated against MalConv.

Random Append. The Random Append attack works by appending byte values sampled from a uniform distribution. This baseline attack measures how easily an append attack could offset features derived from the file length, and helps compare the actual adversarial gains from more complex append strategies over random appended noise.

Gradient Append. The Gradient Append strategy uses the input gradient value to guide the changes in the appended byte values. The algorithm appends $numBytes$ to the candidate sample and updates their values over $numIter$ iterations or until the victim classifier is evaded. The gradient of the input layer with respect to the classification loss ∇_l indicates the direction in the input space of the change required to shift the instance towards the other class. The representation of all appended bytes is iteratively updated, starting from random values. However, as the input bytes are mapped to a discrete embedding representation in MalConv, the end-to-end architecture becomes non-differentiable and its input gradient cannot be computed analytically. Therefore, this attack uses a heuristic to instead update the embedding vector and discretize it back in the byte space to the closest byte value along the direction of the embedding gradient. We refer interested readers to the original paper for details of this discretization process (Kolosnjaji et al. 2018). The attack requires $numBytes*numIter$ gradient computations and updates to the appended bytes in the worst case, which could be prohibitively expensive for large networks.

[1]This was also reported in the original MalConv study.

Benign Append. We propose two new append strategies, one intended to highlight a vulnerability specific to the MalConv architecture, and the second to address the potentially long convergence time of the previously-proposed gradient-based attack. First, the Benign Append attack allows us to observe how the MalConv architecture encodes positional features extracted from the input byte sequences. The attack appends leading bytes extracted from benign instances that are correctly classified with high confidence by the victim classifier. The intuition behind this attack uses the observation that the leading bytes of a file are the most influential towards the classification decision (Raff et al. 2017). Therefore, it signals whether the maliciousness of the target could be offset by appending highly-influential benign bytes.

Algorithm 1 The FGM Append attack

function FGMAPPEND(x_0, $numBytes$, ϵ)
    $x_0 \leftarrow$ PADRANDOM$(x_0, numBytes)$
    $e \leftarrow$ GETEMBEDDINGS(x_0)
    $e_u \leftarrow$ GRADIENTATTACK(e, ϵ)
    for i **in** $|x_0|...|x_0| + numBytes - 1$ **do**
        $e[i] \leftarrow e_u[i]$
    end for
    $x^* \leftarrow$ EMBEDDINGMAPPING(e)
    return x^*
end function
function GRADIENTATTACK(e, ϵ)
    $e_u \leftarrow e - \epsilon * sign(\nabla_l(e))$
    return e_u
end function
function EMBEDDINGMAPPING(e_x)
    $e \leftarrow$ ARRAY(256)
    for $byte$ **in** $0...255$ **do**
        $e[byte] \leftarrow$ GETEMBEDDINGS$(byte)$
    end for
    for i in $0...|e_x|$ **do**
        $x^*[i] \leftarrow argmin_{b \in 0...255}(||e_x[i] - e[b]||_2)$
    end for
    return x^*
end function

FGM Append. Based on the observation that the convergence time of the Gradient Append attack grows linearly with the number of appended bytes, we propose the "one-shot" FGM Append attack, an adaptation of the Fast Gradient Method (FGM) originally described in (Goodfellow, Shlens, and Szegedy 2014). The pseudocode is described in Algorithm 1. Our attack starts by appending $numBytes$ random bytes to the original sample x_0 and updating them using a policy dictated by FGM. The FGM attack updates each embedding value by a user specified amount ϵ in a direction that minimizes the classification loss l on the input, as dictated by the sign of the gradient. In order to avoid the non-differentiability issue, our attack performs the gradient-based updates of the appended bytes on the embedding space, while mapping the updated value to the closest byte value representation in EMBEDDINGMAPPING using the L_2 distance metric. Unlike Gradient Append, the FGM Append applies a single update to the appended bytes, which makes

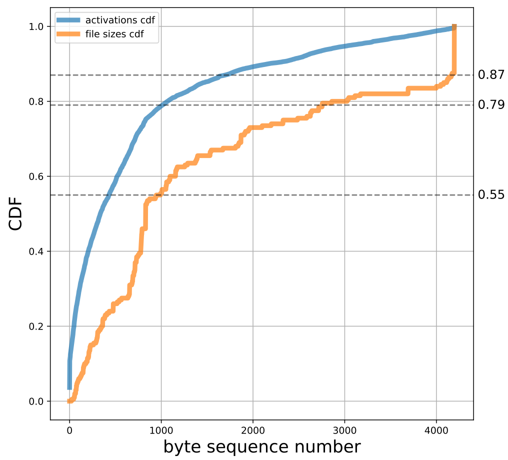

Figure 2: CDF of the file sizes and activation locations after the max-pooling layer. Each byte sequence correspond to 500 bytes from the original file and could be selected at most 128 times for each instance.

it very fast regardless of the number of appended bytes.

Slack Attacks

Besides the inability to append bytes to files that already exceed the model's maximum size (e.g., 2MB for MalConv), append-based attacks suffer from an additional limitation. Figure 2 plots the average frequency of each of the 4,195 input byte sequences as input to the fully connected layer across a random set of 200 candidate malware samples. This shows that, for example, while the first 1,000 byte sequences (0.5 MB) in binaries correspond to 79% of the actual features for the classifier, only 55% of the files are smaller than that. Additionally, 13% of the instances cannot be attacked at all because they are larger than the maximum file size for the classifier. The result shows not only that appended bytes need to offset a large fraction of the discriminative features, but also that attacking the byte sequences of these discriminative features directly will likely amplify the attack effectiveness due to their importance. Driven by this intuition, we proceed to describing an attack strategy that would exploit the existing bytes of binaries *with no side effects on the functionality of the program.*

Slack FGM. Our strategy defines a set of slack bytes where an attack algorithm is allowed to freely modify bytes in the existing binary without breaking the PE. Once identified, the slack bytes are then modified using a gradient-based approach. The SLACKATTACK function in Algorithm 2 highlights the architecture of our attack. The algorithm is independent of the strategy SLACKINDEXES employed for extracting slack bytes or the gradient-based method in GRADIENTATTACK used to update the bytes.

In our experiments we use a simple technique that empirically proves to be effective in finding sufficiently large slack regions. This strategy extracts the gaps between neighboring PE sections of an executable by parsing the executable

Algorithm 2 The Slack FGM attack

function SLACKATTACK(x_0)
  $m \leftarrow$ SLACKINDEXES(x_0)
  $e \leftarrow$ GETEMBEDDINGS(x_0)
  $e_u \leftarrow$ GRADIENTATTACK(e)
  $x_u \leftarrow$ EMBEDDINGMAPPING(e_u)
  $x^* \leftarrow x_0$
  for idx **in** m **do**
    $x^*[idx] \leftarrow x_u[idx]$
  end for
  return x^*
end function
function SLACKINDEXES(x)
  $s \leftarrow$ GETPESECTIONS(x)
  $m \leftarrow$ ARRAY(0)
  for i **in** $0...|s|$ **do**
    if $s[i].RawSize > s[i].VirtualSize$ **then**
      $r_s \leftarrow s[i].RawAddress + s[i].VirtualSize$
      $r_e \leftarrow s[i].RawSize$
      for idx **in** $r_s...r_e$ **do**
        $m \leftarrow$ APPEND(m, idx)
      end for
    end if
  end for
  return m
end function

section header. The gaps are inserted by the compiler and exist due to misalignments between the virtual addresses and the multipliers over the block sizes on disk. We compute the size of the gap between consecutive sections in a binary as $RawSize - VirtualSize$, and define its byte start index in the binary by the section's $RawAddress + VirtualSize$. By combining all the slack regions, SLACKINDEXES returns a set of indexes over the existing bytes of a file, indicating that they can be modified.

Although more complex byte update strategies are possible, potentially accounting for the limited leverage imposed by the slack regions, we use the technique introduced for the FGM Append attack in Algorithm 1, which proved to be effective. Like in the case of FGM Append, updates are performed on the embeddings of the allowed byte indexes and the updated values are mapped back to the byte values using the L_2 distance metric.

Results

Here, we evaluate the attacks described in the previous section in the same adversarial settings using both our Mini and Full datasets. Our evaluation seeks to answer the following three questions:

- How do existing attacks generalize to classifiers trained on larger datasets?
- How vulnerable is a robust MalConv architecture to adversarial samples?
- Are slack-based attacks more effective than append attacks?

In an attempt to reproduce prior work, we select candidate instances from the test set set if they have a file size smaller

# Bytes	Random		Benign		FGM	
	Mini	Full	Mini	Full	Mini	Full
500	0%	0%	4%	0%	1%	13%
2000	0%	0%	5%	0%	2%	30%
5000	0%	0%	6%	1%	2%	52%
10000	0%	0%	9%	1%	1%	71%

Table 1: Success Rate of the Append attacks for increasing number of bytes on both the Mini and Full datasets.

than 990,000 bytes and are correctly classified as malware by the victim. We randomly pick 400 candidates and test the effectiveness of the attacks using the Success Rate (SR): the percentage of adversarial samples that successfully evaded detection.

Append Attacks. We evaluate the append-based attacks on both the Mini and the Full datasets by varying the number of appended bytes. Table 1 summarizes these results.

We observe that the Random Append attack fails on both datasets, regardless of the number of appended bytes. This result is in line with our expectations, demonstrating that the MalConv model is immune to random noise and that the input size is not among the learned features. However, our results do not reinforce previously reported success rates of up to 15% in (Kolosnjaji et al. 2018).

The SR of the Benign Append attack seems to progressively increase with the number of added bytes on the Mini dataset, but fails to show the same behavior on the Full dataset. Conversely, on the FGM Append attack we observe that the attack fails on the Mini dataset, while reaching up to 71% SR on the Full dataset. This paradoxical behavior highlights the importance of large, robust datasets in evaluating adversarial attacks. One reason for the discrepancy in attack behaviors is that the MalConv model trained using the Mini dataset (modeled after the dataset used by Kolosnjaji et al.) has a severe overfitting problem. In particular, the success of appending specific benign byte sequences from the Mini dataset could be indicative of poor generalizability and this is further supported by the disconnect between the model's capacity and the number of samples in the Mini dataset. When we consider the one-shot FGM Attack's success on the Full dataset and failure on the Mini dataset, this can also be explained by poor generalizability in the Mini model; the single gradient evaluation does not provide enough information for the sequence of byte changes made in the attack. Recomputing the gradient after each individual byte change is expected to result in a higher attack success rate.

Aside from the methodological issues surrounding dataset size and composition, our results also show that even a robustly trained MalConv classifier is vulnerable to append attacks when given a sufficiently large degree of freedom. Indeed, the architecture uses 500 byte convolutional kernels with a stride size of 500 and a single max pool layer for the entire file, which means that not only is it looking at a limited set of relatively coarse features, but it also selects the best 128 activations locations irrespective of location. That is, once a sufficiently large number of appended bytes are added in the FGM attack, they quickly replace legitimate

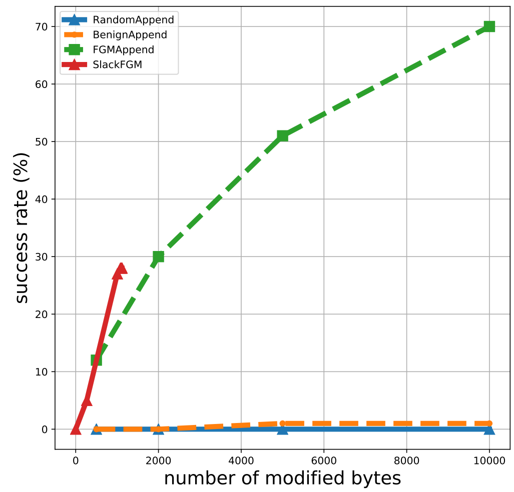

Figure 3: The success rate of the attacks as a function of the average number of modified bytes on the Full dataset.

features from the original binary in the max pool operation. Therefore, the architecture does not encode positional information, which is a significant vulnerability that we demonstrate can be exploited.

Additionally, we implemented the Gradient Append attack proposed by Kolosnjaji et al., but failed to reproduce the reported results. We aimed to follow the original description, with one difference: our implementation, in line with the original MalConv architecture, uses a special token for padding, while Kolosnjaji et al. use the byte value 0 instead. We evaluated our implementation under the same settings as the other attacks, but none of the generated adversarial samples were successful. One limitation of the Gradient Append attack that we identified is the necessity to update the value of each appended byte at each iteration. However, different byte indexes might converge to their optimal value after a varying number of iterations. Therefore, successive and unnecessary updates may even lead to divergence of some of the byte values. Indeed, empirically investigating individual byte updates across iterations revealed an interesting oscillating pattern, where some bytes receive the same sequence of byte values cyclically in later iterations.

Slack Attacks. We evaluate the Slack FGM attack over the Full dataset for the same experimental settings as above. In order to control the amount of adversarial noise added in the slack bytes, we use the ϵ parameter to define an L_2 ball around the original byte value in the embedding space. Only those values provided by the FGM attack that fall within the ϵ ball are considered for the slack attack, otherwise the original byte value will remain. The upper bound for the SR is 28% for $\epsilon = 1.0$, where we observed all the available slack bytes being modified according to the gradient. In order to compare it with the append attacks, in Figure 3 we plot the SR as a function of the number of modified bytes. The results show that, while the FGM Append attack could achieve a higher SR, it also requires much larger number of extra byte modifications. The Slack FGM attack achieves a

SR of 28% for an average of 1005 modified bytes, while the SR of the FGM Append lies around 20% for the same setting. This results confirms our initial intuition that the coarse nature of MalConv's features requires consideration of the surrounding contextual bytes within the convolutional window. In the slack attack, we make use of existing contextual bytes to amplify the power of our FGM attack without having to generate a full 500-byte convolutional window using appended bytes.

Related Work

The work by Barreno et al. (Barreno et al. 2010) was among the first to systematize attack vectors against machine learning, where they distinguished evasion as a type of test-time attack. Since then, several evasion attacks have been proposed against malware detectors. Many of these attacks focus on additive techniques for evasion, where new capabilities or features are added to cause misclassification. For instance, Biggio et al. (Biggio et al. 2013) use a gradient-based approach to evade malware detectors by adding new features to PDFs, while Grosse et al. (Grosse et al. 2017) and Hu et al. (Hu and Tan 2018) add new API calls to evade detection. More recently, Anderson et al. (Anderson et al. 2018) used reinforcement learning to evade detectors by selecting from a pre-defined list of semantics-preserving transformations. Similarly, Xu et al. (Xu, Qi, and Evans 2016) propose a genetic algorithm for manipulating PDFs while maintaining necessary syntax. Closest to our work is the gradient-based append attack by (Kolosnjaji et al. 2018) against the CNN-based MalConv architecture. In comparison to earlier work, our slack-based attack operates on the raw bytes of the binary, and modifies them without requiring the expensive feedback loop from the reinforcement learning agent and has the potential to outperform append-based attacks.

Conclusion

In this paper, we explored the space of adversarial examples against deep learning-based malware detectors. Our experiments indicate that the effectiveness of adversarial attacks on models trained using small datasets does not always generalize to robust models. We also observe that the MalConv architecture does not encode positional information about the input features and is therefore vulnerable to append-based attacks. Finally, we proposed the Slack FGM attack, which modifies existing bytes without affecting semantics, with greater efficacy than append-based attacks.

References

Anderson, H. S.; Kharkar, A.; Filar, B.; Evans, D.; and Roth, P. 2018. Learning to evade static pe machine learning malware models via reinforcement learning. *arXiv preprint arXiv:1801.08917*.

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

Biggio, B.; Corona, I.; Maiorca, D.; Nelson, B.; Šrndić, N.; Laskov, P.; Giacinto, G.; and Roli, F. 2013. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, 387–402. Springer.

Carlini, N., and Wagner, D. 2017. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, 39–57. IEEE.

Goodfellow, I.; Shlens, J.; and Szegedy, C. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

Grosse, K.; Papernot, N.; Manoharan, P.; Backes, M.; and McDaniel, P. 2017. Adversarial examples for malware detection. In *European Symposium on Research in Computer Security*, 62–79. Springer.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Hu, W., and Tan, Y. 2018. Black-box attacks against RNN based malware detection algorithms. In *The Workshops of the The Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 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. *26th European Signal Processing Conference (EUSIPCO '18)*.

Krčál, M.; Švec, O.; Bálek, M.; and Jašek, O. 2018. Deep convolutional malware classifiers can learn from raw executables and labels only. *International Conference on Learning Representations*.

Microsoft. 2018. Pe format. `https://docs.microsoft.com/en-us/windows/desktop/debug/pe-format`.

Papernot, N.; McDaniel, P.; Goodfellow, I.; Jha, S.; Celik, Z. B.; and Swami, A. 2017. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, 506–519. ACM.

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*.

Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; and Fergus, R. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.

Ugarte-Pedrero, X.; Balzarotti, D.; Santos, I.; and Bringas, P. G. 2015. Sok: Deep packer inspection: A longitudinal study of the complexity of run-time packers. In *2015 IEEE Symposium on Security and Privacy (SP)*, 659–673. IEEE.

Xu, W.; Qi, Y.; and Evans, D. 2016. Automatically evading classifiers. In *Proceedings of the 2016 Network and Distributed Systems Symposium*.

Zhang, X.; Zhao, J.; and LeCun, Y. 2015. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, 649–657.