Evading Machine Learning Malware Detection

Hyrum S. Anderson
Endgame, Inc.
hyrum@endgame.com

Anant Kharkar
University of Virginia
agk7uc@virginia.edu

Bobby Filar
Endgame, Inc.
bfilar@endgame.com

Phil Roth
Endgame, Inc.
proth@endgame.com

ABSTRACT

Machine learning is a popular approach to signatureless malware detection because it can generalize to never-before-seen malware families and polymorphic strains. This has resulted in its practical use for either primary detection engines or supplementary heuristic detections by anti-malware vendors. Recent work in adversarial machine learning has shown that models are susceptible to gradient-based and other attacks. In this whitepaper, we summarize the various attacks that have been proposed for machine learning models in information security, each which require the adversary to have some degree of knowledge about the model under attack. Importantly, even when applied to attacking machine learning malware classifier based on static features for Windows portable executable (PE) files, these attacks, previous attack methodologies may break the format or functionality of the malware. We investigate a more general framework for attacking static PE anti-malware engines based on reinforcement learning, which models more realistic attacker conditions, and subsequently has provides much more modest evasion rates. A reinforcement learning (RL) agent is equipped with a set of functionality-preserving operations that it may perform on the PE file. It learns through a series of games played against the anti-malware engine which sequence of operations is most likely to result in evasion for a given malware sample. Given the general framework, it is not surprising that the evasion rates are modest. However, the resulting RL agent can succinctly summarize blind spots of the anti-malware model. Additionally, evasive variants generated by the agent may be used to harden machine learning anti-malware engine via adversarial training.

Keywords

malware evasion, model hardening, reinforcement learning

Black Hat USA 2017, July 22-27, 2017, Las Vegas, NV, USA

1. INTRODUCTION

Machine learning has been an attractive tool for anti-malware vendors for either primary detection engines or as supplementary detection heuristics. Properly regularized machine learning models generalize to new samples whose features and labels follow the same distribution as the training data set. Furthermore, supervised learning models automatically summarize complex relationships among features in the training dataset that are discriminating between malicious and benign labels. This allows defenders to quickly adapt to shifts in how malware is manifest in the wild.

Unfortunately, motivated and sophisticated adversaries are intentionally seeking to evade anti-malware engines, be they signature-based or otherwise. In the context of a machine learning model, an attacker's aim is to discover a set of features that the model deems discriminating, but may not be a causal indicator of the desired malicious behavior. Additionally, the attacker attempts to camouflage the malware in feature space by inducing a feature representation that is highly correlated with, but not necessarily causal to benign behavior.

Several recent studies have demonstrated how machine learning systems can be evaded algorithmically or, ironically, by other machine learning models. Some of this work has been generally devoted to evading models that detect malware (Android, PDF malware, Windows PE) or malware behavior (detecting domain generation algorithms) [10, 1, 23, 11]. For each, the adversary has a greater or lesser degree of knowledge about the machine learning model under attack. Their applicability for evading Windows PE static malware classifiers may not be straightforward because modifying the binary PE file may destroy its format, or maim the malicious behavior. We summarize these attacks in Section 2.

The goal of this whitepaper is to report on an ambitious approach to evade static analysis anti-malware PE engine under the following conditions:

1. The attacker has no knowledge of the features, structure or parameters (weights) of the static PE malware classifier.

2. The attacker has the ability to retrieve a malicious/benign label (or score, if reported) for an arbitrary PE file submitted to the anti-malware engine.

3. The attack aims to modify a malicious Windows PE so that it is no longer flagged by the anti-malware engine.

Our intent is two-fold: provide an automated means to summarize the weaknesses of an anti-malware engine, and to produce functioning evasive malware samples that can be used to augment a machine learning model in adversarial training [9]. We focus on static Windows PE malware evasion, which presents some unique challenges for realistic implementation. We also release open source code in the form of an OpenAI gym [3], for researchers to improve upon this generic approach[1].

2. BACKGROUND

We begin with a summary of static malware detectors for Windows PE files. We provide a summary of recent work in attacking machine learning infosec models in Section .

2.1 Synopsis of static malware detection using Machine Learning

Static malware detection and prevention is an important protection layer in a security suite because when successful, it allows malicious files to be detected prior to execution, for example, when written to disk, when an existing file is modified, or when execution is requested.

Static PE malware detectors have been used since at least 2001 [20], and owing largely to the structured file format and backwards compatibility requirements, many concepts remain surpisingly the same in subsequent published studies [12, 21, 18, 6, 19]. We provide a review in temporal order of publication. (We exclude a large body of literature that includes dynamic malware detection from time-dependent sequences of system calls for analysis [6, 16, 2].)

In [20], authors assembled a dataset and generated labels by running through a McAfee virus scanner. PE files were represented by features that included imported functions, strings and byte sequences. Various machine learning models were trained and validated on a holdout set. Models included rules induced from RIPPER [5], naive Bayes and an ensemble classifier. In 2004, authors in [12] included byte-level N-grams, and employed techniques from natural language process, including tf-idf weighting of strings. In 2009, authors in [21] proposed using just seven features from the PE header, including DebugSize, user-definable ImageVersion, ResourceSize, and virtual size of the second listed section, motivated by the fact that malware samples typically exhibit those elements. Authors in [19] leveraged novel two dimensional byte entropy histograms that is fed into a multi-layer neural network for classification.

Notably, despite recent advances in deep learning that have dramatically improved the state of the art especially in object classification, machine translation and speech recognition, hand-crafted features apparently still represent the state of the art in published literature. Although the state of the art may change to end-to-end deep learning in the ensuing months or years, hand-crafted features derived from parsing the PE file may continue to be relevant indefinitely because of the structured format.

2.2 Related work: attacking machine learning infosec models

Several recent works have addressed attacking machine learning models in information security. In this section, we'll categorize these methods into three coarse bins, which are graphically portrayed in Figure 1.

1. *Direct gradient-based attacks* in which the model must be fully differentiable and the structure and weights must be known by the attacker. Given this, the attacker can essentially query the model directly to determine how best to bypass it.
2. *Attacks against models that report a score.* The attacker has no knowledge about the model structure, but has unlimited access to probe the model and may be able to learn how to decrease the score.
3. *Binary black-box attacks.* The attacker has no knowledge about the model, but has unlimited access to probe the model.

2.2.1 Direct gradient-based attacks

Gradient information about the model under attack provides extremely powerful clues to the attacker, which can be used in one of at least two ways.

The first is a perturbing the sample $\mathbf{x}$ in the direction that would most decrease the score $J(\mathbf{x};\theta)$,

$$\mathbf{x}^* = \mathbf{x} + \sigma\left(\nabla_{\mathbf{x}} J\left(\mathbf{x};\theta\right)\right) \tag{1}$$

The vector function $\sigma(\cdot)$ is a domain-specific mapping of the input back to the range of acceptible objects. For example, the fast gradient sign method uses $\sigma(\delta) = \epsilon \text{sgn}(\delta)$ so that the perturbation is imperceptible, maximally bounded by a change of ϵ to any one pixel [9].

In [10], the authors attack a deep learning Android malware model using gradient perturbation method. The feature vector $\mathbf{x} \in \{0,1\}^{545333}$ is a large sparse binary vector. It is perturbed in a way that bounds the total number of changes to a fixed number (via ℓ_1 constraint). Furthermore, $\sigma(\cdot)$ is implemented as an index set that allows features to be added (never removed), and only if they do not interfere with other features that are already present in the application. The authors report evasion efficacy from 50% to 84%, depending on the model architecture. Since the work performs the attack only in feature space, malicious files are never generated during this process.

A second class of gradient-based attacks connects the model under attack to a generator model in a generative adversarial network (GAN) [8]. Unlike perturbation methods, the generator learns to generate a completely novel sample from a random seed. Through a series of adversarial rounds, the generator learns to produce samples that appear to be drawn from the benign class-conditional distribution $\hat{p}(\mathbf{x}|y = \text{benign})$ that has been estimated by the model under attack (the *discriminator* in GAN literature).

Like the perturbation method, a mapping function is required to ensure that hallucinated samples constitute acceptable objects. For images, this step is often ignored. For malware, the mapping onto legitimate PE files that perform the desired malicious function has yet to studied in the general case.

In [1], the authors apply this GAN-based attack to a detector of domain generation algorithm (DGA) domains, which attempts to distinguish human-crafted from algorithmically-generated domain names. The only constraint on generated domain names is that they contain valid characters, which is trivially encoded into the alphabet of tokens in the neural network. As such, the mapping into "legitimate" characters is automatically encoded.

2.2.2 Attacks against models that report a score

[1] https://github.com/drhyrum/gym-malware

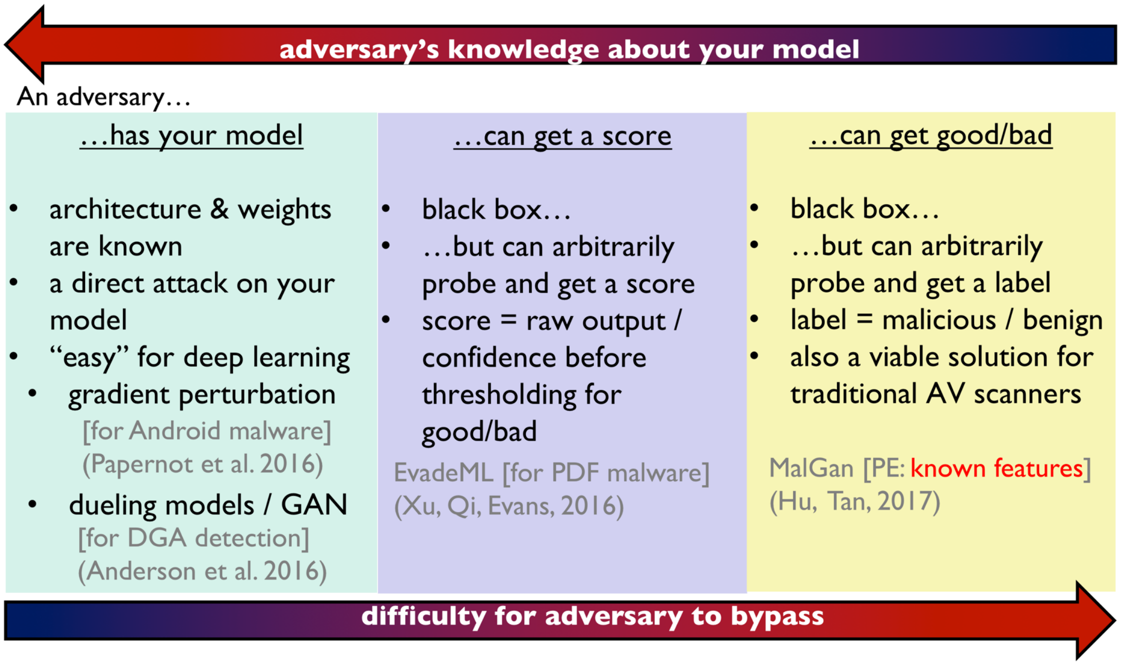
adversary's knowledge about your model
An adversary…
…has your model
architecture & weights are known
a direct attack on your model
"easy" for deep learning
gradient perturbation
[for Android malware]
(Papernot et al. 2016)
dueling models / GAN
[for DGA detection]
(Anderson et al. 2016)
…can get a score
black box…
…but can arbitrarily probe and get a score
score = raw output / confidence before thresholding for good/bad
EvadeML [for PDF malware]
(Xu, Qi, Evans, 2016)
…can get good/bad
black box…
…but can arbitrarily probe and get a label
label = malicious / benign
also a viable solution for traditional AV scanners
MalGan [PE: known features]
(Hu, Tan, 2017)
difficulty for adversary to bypass

Figure 1: Attack categories, categorized into three coarse bins.

Attacks against black-box models represent a more generic attack. If the black-box model produces a score for any query, an attacker can directly measure (myopically) the efficacy of any perturbation.

In [23], authors leverage this reduction in score reported by PDF malware classifiers as a fitness function in a genetic algorithm framework. To ensure that mutations preserve the desired malicious behavior, an oracle is used to compare the runtime behavior with that of the original seed. The authors utilize the Cuckoo sandbox as an oracle, and note how it is computationally expensive. After approximately one week of execution, the genetic algorithm found nearly 17K evasive variants from 500 malicious seeds, and achieved 100% evasion rate of the PDFrate classifier. After two days of compute time, the algorithm found nearly 3K evasive samples from 500 initial seeds, all bypassing the PDF malware model.

2.2.3 *Binary black-box attacks*

Finally, in the most generic attack, the anti-malware engine reports only malicious or benign for an input.

Recently, [11] introduced MalGAN to generate PE malware to bypass a black-box static PE malware engine. The idea is simple: instead of attacking the black box directly, the attacker creates a substitute model trained to reproduce outputs observed by probing the target model with corresponding inputs. Then, the substitute model is used for gradient computation in a modified GAN to produce evasive malware variants. The authors report 100% efficacy in bypassing the target model, and furthermore, demonstrate that retraining with the adversarial examples has limited efficacy.

The approach is based on a similar idea for attacking black box models, presented in [15] for computer-vision models. But the latter work leverages a more straightforward gradient perturbation method to generate samples adversarial to the substitute model. These evaded the target models with high probability. One could reasonably apply the same approach to PE malware evasion.

A notable limitation of [11] is that the attacker must know the complete feature space of the target model. The substitute model is trained and GAN attack is carried out in this feature space. The authors argue that the feature space may be discovered by the attacker, and use only imported functions in their evaluations. While this paper is among the first to attempt attacking PE malware models, it is quite limited in the fact that the attacker knows and shares the features space: features are limited to only imported functions, which is insufficient for a modern static malware model. Manipulations are made in feature space, and a malicious binary file is actually never created, instead only passed to the models under attack as a feature vector. These issues make it an unrealistic attack in practice.

2.2.4 *How our approach differs*

Our approach presents further limitations on the information available to an attacker.

1. Output from the target classifier is strictly Boolean, declaring only whether a sample is deemed benign or malicious by the classifier.

2. The feature space and structure of the target classifier are completely unknown.

3. There does not exist an external party (such as an oracle) to guarantee that a sample is valid. Thus, there is no mapping function to the space of legitimate PE files.

These restrictions presents what we believe is the most difficult black-box evasion scenario from an attacker's perspective. As a result of the limited information available to the attacker, evasion rates are significantly lower than those of the approaches above.

2.3 Reinforcement learning

We implement our black-box attack using a reinforcement learning approach [22]. A reinforcement learning model consists of an agent and an environment. For each turn, an agent may choose one from a set of actions $\mathcal{A}$. The selected action may emit a change in the environment described by the state space $\mathcal{S}$. A reward function produces a scalar award for the new state. The reward and observed state of the environment are fed back to the agent, which may determine the estimated value each possible action. The agent follows a policy based on these values to select it's next action. The agent learns incrementally through a tradeoff of exploration and exploitation which actions to produce given the environment's state. The reward provides the key objective for learning, and notably, may be zero for many turns until a target state is reached through a relatively long series of actions/states. Early actions/states that produce no immediate reward but are important to the final outcome are promoted via a value function that predicts the long-term reward for a given action/state (Q-learning).

Deep reinforcement learning was introduced as a framework to play Atari games by reinforcement agents that often exceeed human performance [13, 14]. Among the key contributions of the deep reinforcement learning framework was its ability, as in deep learning, for the agent to learn a value function in an end-to-end way: it takes raw pixels as input, and outputs predicted rewards for each action. This learned value function is the basis for so-called deep Q-learning, where the Q-function is learned and refined over hundreds of games.

In the context of malware evasion, we apply deep Q-learning in a reinforcement learning framework, as shown in Fig. 2. In the Markov decision process shown, agent gets an estimate of the environment's state $\mathbf{s} \in \mathcal{S}$, represented by a feature vector $\mathbf{s}$ of the malware sample (which need not correspond to any internal representation of the malware by the anti-malware engine). The Q-function and action policy determine what action to take. In our framework, the actions space $\mathcal{A}$ consists of a set of modifications to the PE file that (a) don't break the PE file format, and (b) don't alter the intended functionality of the malware sample. The reward function is measured by the anti-malware engine, which is converted to a reward: 0 if the modified malware sample is judged to be benign, and 1 if it is deemed to be malicious. The reward and state are then fed back into the agent.

3. OUR APPROACH

With an aim to engage the broader community, we implement our malware evasion environment as an extensible OpenAI `gym` [4], which we release at https://github.com/drhyrum/gym-malware. The `gym` framework has become popular for training RL agents because it provides a standardized environment to produce benchmarks (e.g. like playing Atari games). We adopt some game-playing terminology in some of our description below. In addition, we release a default agent using `keras-rl` [17].

The environment consists of an initial malware sample (1 malware sample per "game"), and a customizable anti-malware engine (the attack target). Each step or turn provides the following feedback to the agent:

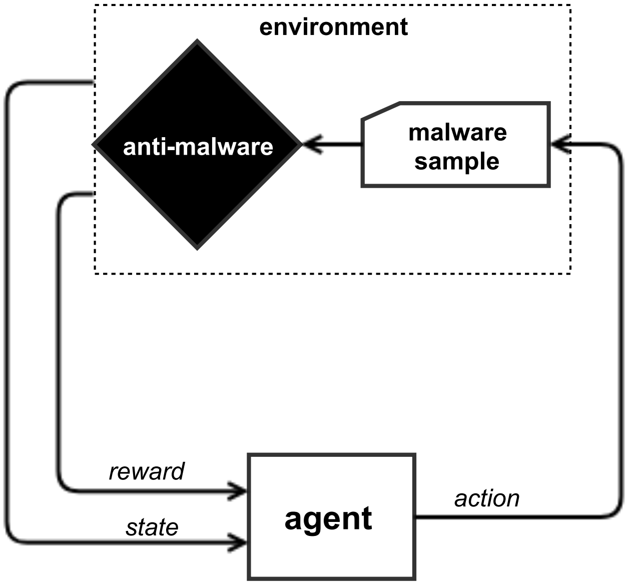

Figure 2: Markov decision process formulation of the malware evasion reinforcement learning problem.

- reward*(float)*: value of reward scored by the previous action. 10.0 *(pass)*, 0.0 *(fail)*;
- observation space*(object)*: feature vector summarizing the composition of the malware sample;
- done*(bool)*: Determines whether environment needs to be reset; True means episode was successful;
- info*(bool)*: Provides diagnostic information about the environment for debugging purposes;

Based on this feedback, the agent chooses from a set of mutations *(actions)* that preserve the format and function of the PE file. We describe our initial implementation of each of these components below. It may be helpful to refer to Figure 2.

3.1 Environment state

The malware sample exists as raw bytes in the game environment. However, in order to more concisely represent the current state of the malware sample, the environment emits the state in the form of a feature vector. In our experiments, the 2350-dimensional feature vector comprised of the following general categories of features:

- PE header metadata
- Section metadata: section name, size and characteristics
- Import & Export Table metadata
- Counts of human readable strings(e.g. file paths, URLs, and registry key names)

- Byte histogram
- 2D byte-entropy histogram as reported in [19]

For feature sets which are countably infinite (section names, imported function names, etc.), we use the hashing trick to collapse into them into a vector of fixed size.

3.2 Action Space

As mentioned above, the file mutations represent the actions or moves available to the agent within the environment. There are a modest number of modifications that can be made to a PE file that do not break the PE file format and do not alter code execution. Some of these include:

- add a function to the import address table that is never used (note, this is the sole manipulation explored in [11]
- manipulate existing section names
- create a new (unused) sections
- append bytes to extra space at the end of sections
- create a new entry point which immediately jumps to the original entry point
- manipulate (break) signature
- manipulate debug info
- pack or unpack the file
- modify (break) header checksum
- append bytes to the overlay (end of PE file)

Note that most of these functions are stochastic in nature. For example, when renaming a section, a new section name is drawn uniformly from a list of section names found in benign files. When appending bytes to the end of a section or file, the length and entropy of of the appended bytes can be specified, but for simplicity, are chosen at random by the agent. Likewise, the compression level used by the packer is chosen at random.

The stochastic nature of the manipulations was chosen for simplicity and presents a learning challenge because the actions are not exactly repeatable. Instead, actions modify broad elements of a PE file that are generally used by static machine learning malware models. An alternative is to unroll the limited number of actions into hundreds of specific actions (e.g., rename section to `.blah`, instead of renaming randomly). However, reinforcement learning with extremely large action spaces is a subject of ongoing research in the research community [7].

4. EXPERIMENTS

In our experiments, we attack a gradient boosted decision tree model trained on 100,000 malicious and benign samples, and which achieves an area under the receiver operating characteristic score (ROC AUC) of 0.96. This model is included in the code that we release. Although not necessary, for convenience, we train it on the features used to represent the state of the environment. We expect this to produce more generous results than can be expected in practice. However, as our intent is to release a toolkit for learning malware manipulating agents, this proof of concept suffices for our purposes.

Random mutations	13%
Black box attack	16%
Score-based attack	14%

Table 1: Evasion rate on 200 holdout samples. Random mutations were averaged over ten runs.

Our preliminary experiments involved our basic keras-rl agents tested in our OpenAI gym. We examined two scenarios of information feedback from the target classifier: a realistic black box (with only Boolean output) and a continuous score. Both agents utilize a Boltzman exploration / exploitation strategy, in which mutations are drawn proportionally to their expected Q-value. Both agents are allowed to perform up to twenty mutations before declaring failure. Rounds terminate early should the agent bypass the malware model prior to the twenty allotted rounds. We allow for up to 100,000 rounds (unique malware seeds) to train each model.

For the black box attack, rewards of 10.0 / 0.0 are provided for evasion / failed-evasion, respectively. We set a threshold of 0.9 for the static malware model, which corresponds to a conservative false positive rate.

For the attack with continuous score, an immediate reward is given by initial$_s$core − reported$_s$core, and provide a reward of 10.0 if the agent successfully bypasses the model. Note that this can result in *negative* rewards should the mutations actually increase the original score.

For comparison, we also demonstrated an attack in the same environment using randomized action with no RL agent.

5. RESULTS

Results were modest, and are summarized in Table 1.

Surprising to us is that the black box attack had a higher evasion rate than the score-based attack (in which the machine learning model returns a score). We postulate that this is because the small rewards provided by the continuous rewards can cause the agent to become myopic, sacrificing larger gains for short-term wins. However, since this is merely a proof of concept, we leave this line of research to future work.

We uploaded twenty samples produced by the reinforcement learning agent to VirusTotal, and found that the median detection ratio was 18 / 63, down from 31 / 63. We note that VirusTotal does not represent the full detection platform for vendors in VirusTotal; nevertheless, this does demonstrate that by bypassing a relatively simple machine learning model, cross-evasion of commercial products may be possible.

6. DISCUSSION

We believe that machine learning is a useful tool to generalize to never-before-seen samples. We note that even after 100,000 rounds of being trained specifically to bypass it, our agent can bypass a toy model with relatively small efficacy. Although the reinforcement learning models and manipulations are relatively rudimentary, the modest evasion rate demonstrates that black box machine learning models for malware detection can be evaded.

What can be done to prevent such an attack on infosec machine learning models? First, machine learning models can be hardened by precisely the kind of techniques presented here. By generating malicious samples that are known to bypass your machine learning model, these samples can be folded back into the training set to attempt to "patch" these blind spots. Furthermore, the set of manipulations can be inpsected to understand general tendancies of the machine learning model, in order to inform data science teams and engineers.

7. REFERENCES

[1] H. S. Anderson, J. Woodbridge, and B. Filar. DeepDGA: Adversarially-tuned domain generation and detection. In *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security*, pages 13–21. ACM, 2016.

[2] B. Athiwaratkun and J. W. Stokes. Malware classification with LSTM and GRU language models and a character-level CNN. In *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*, pages 2482–2486. IEEE, 2017.

[3] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI gym. *arXiv preprint arXiv:1606.01540*, 2016.

[4] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym, 2016.

[5] W. W. Cohen. Fast effective rule induction. In *Proceedings of the twelfth international conference on machine learning*, pages 115–123, 1995.

[6] G. E. Dahl, J. W. Stokes, L. Deng, and D. Yu. Large-scale malware classification using random projections and neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 3422–3426. IEEE, 2013.

[7] G. Dulac-Arnold, R. Evans, H. van Hasselt, P. Sunehag, T. Lillicrap, J. Hunt, T. Mann, T. Weber, T. Degris, and B. Coppin. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*, 2015.

[8] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

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

[10] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel. Adversarial perturbations against deep neural networks for malware classification. *arXiv preprint arXiv:1606.04435*, 2016.

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

[12] J. Z. Kolter and M. A. Maloof. Learning to detect malicious executables in the wild. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 470–478. ACM, 2004.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

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

[16] R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas. Malware classification with recurrent networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 1916–1920. IEEE, 2015.

[17] M. Plappert. keras-rl. https://github.com/matthiasplappert/keras-rl, 2016.

[18] K. Raman et al. Selecting features to classify malware. *InfoSec Southwest*, 2012, 2012.

[19] J. Saxe and K. Berlin. Deep neural network based malware detection using two dimensional binary program features. In *Malicious and Unwanted Software (MALWARE), 2015 10th International Conference on*, pages 11–20. IEEE, 2015.

[20] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo. Data mining methods for detection of new malicious executables. In *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on*, pages 38–49. IEEE, 2001.

[21] M. Z. Shafiq, S. M. Tabish, F. Mirza, and M. Farooq. A framework for efficient mining of structural information to detect zero-day malicious portable executables. Technical report, Technical Report, TR-nexGINRC-2009-21, January, 2009, available at http://www. nexginrc. org/papers/tr21-zubair. pdf, 2009.

[22] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

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