Towards resilient machine learning for ransomware detection

Li Chen, Chih-Yuan Yang, Anindya Paul, Ravi Sahita
Security and Privacy Research, Intel Labs, Hillsboro, OR 97124

ABSTRACT

There has been a surge of interest in using machine learning (ML) to automatically detect malware through their dynamic behaviors. These approaches have achieved significant improvement in detection rates and lower false positive rates at large scale compared with traditional malware analysis methods. ML in threat detection has demonstrated to be a good cop to guard platform security. However it is imperative to evaluate - is ML-powered security resilient enough?

In this paper, we juxtapose the resiliency and trustworthiness of ML algorithms for security, via a case study of evaluating the resiliency of ransomware detection via the generative adversarial network (GAN). In this case study, we propose to use GAN to automatically produce dynamic features that exhibit generalized malicious behaviors that can reduce the efficacy of black-box ransomware classifiers. We examine the quality of the GAN-generated samples by comparing the statistical similarity of these samples to real ransomware and benign software. Further we investigate the latent subspace where the GAN-generated samples lie and explore reasons why such samples cause a certain class of ransomware classifiers to degrade in performance. Our focus is to emphasize necessary defense improvement in ML-based approaches for ransomware detection before deployment in the wild. Our results and discoveries should pose relevant questions for defenders such as how ML models can be made more resilient for robust enforcement of security objectives.

KEYWORDS

Dynamic ransomware detection, generative adversarial net, adversarial quality metric.

ACM Reference Format:
Li Chen, Chih-Yuan Yang, Anindya Paul, Ravi Sahita. 2019. Towards resilient machine learning for ransomware detection. In KDD 2019 Workshop. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Ransomware is a type of malicious software (malware), which hijacks and blocks victim’s data or machine until a monetary ransom is paid. Its life cycle consists of six phases [42]: i). Distribution: the ransomware arrives at victim’s machine by an email attachment, a drive-by download or a code dropper; ii) Infection: the ransomware installs itself to survive a reboot and disables shadow copies or antivirus processes; iii). Communication: the ransomware contacts its Command and Control (C&C) server for the encryption key; iv) Preparation: the ransomware scans the user’s files, usually pdf, docx, jpg files; v). Encryption: the ransomware encrypts the selected user files; and finally vi) Extortion: a “ransom note”, asking for payment, is displayed to the user. After the ransom is paid, instructions to receive the decryption key will be sent to the user.

There are two main categories of ransomware based on attack approaches: the locker-ransomware and the crypto-ransomware [9, 26]. The locker-ransomware locks the victim’s computer without the encryption. The crypto-ransomware encrypts victim’s files which are very difficult to revert. The quick solution is to pay the extortion and hope the given key can truly decrypt the data. Thus the crypto-ransomware remains a notorious security issue today. In our case study, we focus on cryto-ransomware.

The popularity of Internet and untraceable payment methods and availability of software development tools makes ransomware an feasible weapon for remote adversaries [1]. In recent years, ransomware has posed increasingly major threats. Since 2017, ransomware attacks have increased over 59% yearly with 35% growth in Q4 2017 alone. Although the trend of devising new ransomware has declined in 2018, the occurrence of ransomware attacks is still rising [40, 41].

Dynamic analysis on malware can reveal true malicious intentions by executing malware in a contained environment. Recent research have found behavior analysis via analyzing API calls, registry accesses, I/O activities or network traffic can be effective for ransomware detection[9, 19, 26, 29, 34, 46, 53].

Faced with a tsunami of malware attacks, the security industry are employing machine learning (ML) to automatically detect threats and enhance platform security. Their confidence in ML is not ungrounded. ML algorithms have demonstrated state-of-the-art performance in the field of Computer Vision (CV), Natural language Processing (NLP), Automatic Speech Recognition (ASR). The success of ML has generated huge interest of applications on platform security domains for automated malware detection[15, 16, 51, 56]. Particularly for ransomware detection, algorithms such as naive Bayes, support vector machine, random forest, logistic regression have shown good classification efficacy [2, 47, 57]. Shallow or deep neural networks also demonstrated high effectiveness at ransomware detection [4, 18, 59].

Recent research take advantage of opaqueness of NN algorithms and generate subliminal perturbed input examples which have shown to evade ML based detection. These types of emerging attacks, where an adversary can control the decision of the ML model by small input perturbations, expose a broad attack surface. Although most of the Adversarial Machine Learning (AML) publications [7, 10, 11, 21, 31] focus on misclassification on CV and ASR domains, the proliferation of adversarial examples are spreading to generate sophisticated adversarial malware. These examples perform real-time evasive attack by camouflaging malicious behavior...
to a legitimate software while keeping maliciousness intact and fooling detection during run-time. For example, AVPASS[32] generates potent variations of existing Android malware by querying and inferring features used by malware detection systems. Additionally recent research have shown promise of ML-based approaches to thwart ransomware attack on user systems [19, 54].

The malicious use of ML motivates us to properly study the adversarial attack threat models and investigate the robustness and vulnerability of ML-powered security defense systems. In this paper, we present a case study on using deep learning to automatically bypass ML-powered dynamic ransomware detection systems. We propose a framework based on generative adversarial network [27] to generate dynamic ransomware behaviors and a set of adversarial quality metrics to justify the generated samples indeed persist maliciousness. We discover that most of the selected highly effective ransomware classifiers fail to detect the adversary-generated ransomware, indicating a broad attack surface for ML-powered security systems. We thoroughly examine the latent feature space to understand where the adversarial examples lie. We believe that our proposed framework is useful for the defender system to incorporate and minimize their detection algorithms’ blind spots. Our case study provides a true attack platform security.

The goal of our paper is to provide a framework to understand the resiliency of ransomware detectors. We do not enable a true attack on user system. As demonstrated in this paper, we advocate that a defender should fortify their ML models for ransomware detection via adversarial studies.

Our contributions are summarized as follows:

(1) Although generative adversarial network (GAN) has been used to generate fake samples to resemble the true data distribution, our framework is the first one to study ML resiliency via GAN to automatically generate dynamic ransomware behaviors. Although our experiments illustrated that ML models are highly effective in combating real-world ransomware threats and can achieve high classification accuracy up to 99% accuracy with extremely low false positive rate, our results show that such ML models fail to detect the GAN-generated adversarial samples. To stabilize training and achieve convergence, we utilize data segmentation techniques and auxiliary conditional GAN architecture.

(2) We propose a set of adversarial quality metrics to validate the generated adversarial ransomware and demonstrate the GAN-generated samples via our framework maintain maliciousness verified by such metrics. Although our ML classifiers misclassify these adversarial samples as benign, the adversarial samples are statistically much closer to real ransomware samples.

(3) We emphasize that robustness against adversarial samples is an equally important metric in addition to accuracy, false positive rate, true positive rate, $F_1$ score to thoroughly evaluate ransomware detection scheme before deployment. In our experiment, only one of the seven models has the strongest resiliency on the GAN-generated samples, indicating a broad adversarial attack surface of ML algorithms. On the other hand, our experiments provide guidance for security practitioners to develop resilient ML algorithms proven to defend against adversarial attacks.

(4) We study the reasons why the highly effective models are susceptible by properly investigating in the latent feature space and provide understanding of the blind spots of these models. We present our learning to generate awareness to the security community that adversarial threat models need to be properly evaluated before deploying ML models to detect malware attacks.

The rest of the paper is organized as follows: Sec 2 briefly provides the background on ransomware analysis, adversarial machine learning and generative adversarial network. Sec. 3 describes system architecture, data collection and pre-processing. Sec. 4 presents our proposed framework and adversarial quality assessment procedure. Sec. 5 illustrates experimental results on our dataset.

2 BACKGROUND AND RELATED WORK

2.1 Ransomware Detection

Cabaj et al. [9] use HTTP message sequences and content sizes to detect ransomware. Morato et al. [46] analyzed file sharing traffic for ransomware early detection. Scaife et al. [53] provide an early detection system by monitoring user data changes including the file entropy and similarity changes, the file type changes, file deletion and file type funneling. The honeyfiles-based R-Locker system is in [26] to trap and block ransomware operations. When ransomware scans user’s file system and accesses pre-installed decoy files, the R-Locker service is triggered to apply countermeasures. The “Unveil” system introduced in [34] can detect crypto-ransomware via the I/O access patterns. A Windows kernel I/O driver is developed to collect I/O operations and buffer entropy. It provides an early detection capability on a zero-day ransomware. Continella et al. create ShieldFS [19], a custom kernel driver that collects and performs analysis of low-level file-system activity to classify ransomware activity at runtime using a multi-tier hierarchical decision tree based process monitoring model. ShieldFS also integrates file back-up to its ransomware detection system so it can able to recover files from a trusted secure storage after confirming malicious activity. Sgandurra et al. [55] proposed “EldeRan” which dynamically analyzes Windows API calls, registry key operations, file system operations, directory operations and so on in a sandboxed environment, selects relevant features and finally applies a logistic regression classifier to determine whether an application is ransomware or benignware. In contrast to monitoring system executions proposed in “EldeRan”, we have focused on collecting changes in user file events which demonstrated early indication of ransomware activity, helped processing and storing limited data and finally worked under very limited computational budget in order not to interfere with users regular computational needs. Scaife et al. proposed CryptoDrop [52], an early stage ransomware detection system which made use of file event changes rather than program execution inspection through API call monitoring. Although they have used different ransomware behavioral indicators compared to our file events, they have reached the same conclusion as ours that an union of indicators is a more effective approach in ransomware detection than any of those alone.
2.2 Adversarial Machine Learning

The first adversarial machine learning attack is used against spam filtering by generating adversarial text without affecting content readability [20]. The topic got significant attention in the security community when Szegedy et al. [58] fool a DNN based image recognition classifier by adding low-intensity perturbations to the input image which looks indistinguishable to human eyes. Adversarial attacks on CV typically add small human imperceptible perturbations to the original images and have shown to drastically alter the ML boundary decisions [28], [37], [49], [45], [12]. Beyond CV, [13] generate adversarial speech to change the output of a speech-to-text transcription engine. Adversarial malware are created to bypass ML detection while keeping maliciousness of the software intact [32].

Defense techniques including pre-processing via JPEG compression [22, 23], feature squeezing [61], architecture via regularization [33], adversarial training [38], neural fingerprinting [24] have exhibited success to mitigate the proliferating adversarial machine learning attacks.

2.3 Generative Adversarial Network

The first generative adversarial network (GAN) ever introduced is a fully connected neural network architecture for both the discriminator and the generator [27]. Ever since, abundant GAN variants are proposed. The Deep Convolutional GAN (DCGAN) [50] proposes using strided convolutions instead of fully connected multi-layer perceptrons and feature normalization to stabilize training and dealing with the poor weight initialization problem. The Conditional GAN (CGAN) [44] adds conditional setting to the generator and the discriminator by making both neural networks class-conditional. It has advantages to better represent multi-modal data generation. The Laplacian Pyramid GAN (LPGAN) [25] produces high quality generated images and uses multiple generators and discriminators in its architecture. It downsamples the input images, and during back-propagation, injects noise generated by a conditional GAN and then upsamples the images. Auxiliary Classifier GAN (ACGAN) [48] improves the training of GAN by adding more structure to the GAN’s latent space along with a specialized cost function. Wasserstein GAN (WGAN) [5] uses Wasserstein distance as the loss function to efficiently approximates the Earth Mover distance and significantly reduces the mode dropping phenomenon.

Generative adversarial networks have been widely used in creating adversarial examples to fool ML. [60] trains a conditional GAN algorithm, AdvGAN, to generate perceptually similar adversarial input images to attack state-of-the-art defense methods [39], [30] uses a method to generate adversarial malware samples using MalGAN to attack state-of-the-art black-box ML detection algorithms. [3] demonstrates creation of domain generation malware instances (DGAs) using GAN to bypass modern DGA ML classifiers such as random forest. [6] proposes using GAN to model the malicious behaviors and generate synthetic malware representation which is trained with existing malware samples for effective zero-day threat prevention on ML detectors. [35] proposes tDCGAN using unsupervised deep auto-encoding technique to generate new malware variants based on raw codes and modified features.

3 RANSOMWARE DATA DESCRIPTION

3.1 Data Collection and Description

In our analysis, the ransomware samples are downloaded from VirusTotal, where we collect submitted ransomware between late 2017 to early 2018 based on tags from Microsoft and Kaspersky. The dataset contains various of ransomware with nine major families including Locky: a Microsoft Office macro based ransomware, and Cerber: a product of ransomware-as-a-service. The ransomware family distribution is seen in Figure 1.

The samples are executed in a regular Windows system as shown in Figure 2. The dynamic behaviors are collected via the .Net framework FileSystemWatcher (FSW) API. The callback functions bound to FSW are triggered for all file I/O operations. We collect the low-level I/O activity patterns and calculate the normalized Shannon entropy of the targeted files [53]. To catch evasive ransomware, a user activity simulation program is executed to emulate mouse clicks and key strokes. To mimic an active desktop environment, a Notepad and Office Word applications are launched before and during ransomware execution. The benign data is collected manually from installing and executing approximately a hundred applications from various categories such as office suite, browsers and file compression applications. The idle I/O activities of benign Windows system are collected for a few months from regular backups, updates, anti-virus applications and so on.

Each data collector robot, as seen in Figure 2, is pre-installed with several user files such as Windows Office, text or multimedia.
files. These files are designed to be the target of ransomware and used as decoy files to filter active ransomware samples. If these files are modified during execution, then this sample is assumed to be a “crypto”-ransomware and then collected to the malicious dataset. All behavior data are uploaded to Resilient ML platform [62] for data cleansing. The names of the decoy files are appended with time stamps before ransomware execution, so each sample will see the same set of user files but with different file names.

3.2 Feature Mapping
The collected execution log via FSW contains time stamp, event name, targeted file name and file entropy, as seen in Figure 3. We attempt the least effort of feature processing by mapping the event combined with entropy change. The four main file actions are file delete, file create, file rename and file change. The entropy level is combined with the event of file change. Hence each execution log is represented by a sequence of events. We set the length for each sample to be 3000, so that the shorter length samples will be padded with zeros towards the beginning to match the dimension. Table 1 shows the feature mapping.

![Figure 3: A screen shot of dynamic execution log collected using FileSystemWatcher (FSW).](image)

| Events | Feature encoder |
|--------|----------------|
| Padding | 0 |
| File deleted | 1 |
| File content changed and entropy ∈ [0, 0.1] | 2 |
| File content changed and entropy ∈ [0.2, 0.4] | 3 |
| File content changed and entropy ∈ [0.6, 1] | 4 |
| File created | 5 |
| File content changed and entropy ∈ [0.8, 0.9] | 6 |
| File renamed | 7 |
| File content changed and entropy ∈ [0.4, 0.6] | 8 |
| File content changed and entropy ∈ [0.6, 0.8] | 9 |

Table 1: Feature mapping. We attempt the least effort of feature processing, categorize the events into 9 categories and use zero padding maintain the same length. Our features contain various I/O events and entropy change.

The patterns from both I/O event types and entries of target file constitute our feature set. The benign programs, such as WinZip, may have change events with high entropy, but they will not have as many rename or delete I/O events as typical crypto-ransomware. On our collected dataset, combining various I/O event types and entropy as features is effective to detect ransomware.

4 SYNTHESIZING DYNAMIC FEATURES VIA GAN
GANs are mostly used in computer vision to generate images that seem real to the human eyes. Because they are typically used in CV, one can terminate the training when the generated images look like the real images. The inputs in our case study, however, are dynamic execution logs, so it is not practical to stop training GAN by merely visualizing the generated samples. Furthermore when we directly employ the typical training mechanism of GANs, mode collapsing issues constantly arise. The challenges of training an effective GAN to fool the ransomware classifier motivate us to propose a different GAN training scheme for faster convergence and better-quality sample generation.

The principle of our proposed GAN training scheme is to segment the dynamic execution logs and leverage transfer learning to accelerate training convergence. Each execution log is segmented into m subsequences and then converted 2-dimensional arrays. Then transfer learning is employed such that the parameters and neural network architectures are borrowed from existing and successfully convergent GANs used in the vision domain, while we still train from scratch on the fixed architecture. The effectiveness of employing transfer learning from computer vision to malware classification is previously demonstrated for both static and dynamic malware classification [14, 15], but not yet for adversarial malware generation.

4.1 Threat Model
We assume that the adversary has knowledge to the training data, but no knowledge at all of the underlying ransomware classifiers. This is a realistic assumption since for malware detection, anti-virus vendors obtain their training samples from VirusTotal, which allows users to download binaries or hashes.

4.2 Training Procedure
Our approach essentially consists of segmentation and reshaping as preprocessing, GAN training, quality assessment, concatenation and evaluation. An overview of our framework is seen in Figure 4.

4.2.1 Segmentation and reshaping as preprocessing. We observe that, in our initial experiments, GAN did not converge when trained on the entire logs. This motivates us to consider training a convergent GAN on log segments. After feature mapping, we divide each training execution log into sequences each of length 784. If the length of the execution log is not divisible by 784, the end of the last subsequence will be padded zero. Each subsequence is then reshaped into two-dimensional square arrays of 28 × 28.

We note that the convergence issue may be resolved through searching the space of neural network architectures and the parameters. However our preprocessing step enables transfer learning to borrow existing convergent GAN architectures, hence saving exhaustive search efforts while still achieving convergence.
4.2.2  Training. The generative adversarial networks (GAN), first introduced in [27], are paired neural networks consisting of a generator and a discriminator, which act like two players to win a game. The generator produces samples from the generated distribution \( P_G \) which is to be as close as the real data distribution \( \text{ACGAN} \) on the segmented two-dimensional arrays processed from the execution logs. The discriminator classifies whether the samples are generated by \( P_G \) or truly sampled from \( P_R \). The purpose of the generator is to fool the discriminator and the purpose of the discriminator is to separate the fake from the real. At the end of the training, the generator is supposedly and theoretically to maximize fooling the discriminator.

We train an auxiliary classifier generative adversarial network (ACGAN) on the segmented two-dimensional arrays processed from the execution logs. The ACGAN architecture we employed from computer vision is shown in Figure 5. Denote each real sample as \( r \in \mathbb{R} \subset \mathbb{R}^{28 \times 28} \), where \( \mathbb{R} \) is the space containing all the real segmented execution logs. The paired data are drawn from the joint distribution \((r_1, y_1), (r_2, y_2), \ldots, (r_n, y_n) \sim \, i.i.d \, P_{R,Y} \), where \( y \in Y \) are the class labels with \( Y = 1 \) being ransomware and \( Y = 0 \) being benign.

Denote each generated sample as \( g \in \mathcal{F} \), where \( \mathcal{F} \) is the space containing all fake samples and \( g \) is drawn from the generated sample distribution \( P_G \). Let random variable \( C \) denote the label for data source where \( C = 1 \) means the data is real and \( C = 0 \) means the data is fake. The entire data denoted by \( X \) consist of both real and fake samples, i.e., \( X = \mathbb{R} \cup \mathcal{F} \).

We denote \( z \) as the noise generated by the generator \( G \), which is a function \( G : (z, y) \mapsto g \). Given the data \( X \), the discriminator \( D \) calculates two probabilities: whether the data is real or fake \( P(C|X) \) and the class label of the sample \( P(Y|X) \). The loss function of ACGAN comes into two parts:

\[
L_C = \mathbb{E}(\log P(C = 1|R)) + \mathbb{E}(\log P(C = 0|\mathcal{F})),
\]

and

\[
L_Y = \mathbb{E}(\log P(Y = y|R)) + \mathbb{E}(\log P(Y = y|\mathcal{F})).
\]

The generator is trained to maximize \( L_Y - L_C \) and the discriminator is trained to minimize \( L_Y + L_C \). Adding the above auxiliary classifier to the discriminator in AC-GAN stabilizes training.

Because our threat model assumes the adversary has no knowledge of the underlying classifier, the stopping criterion for training our proposed mechanism only relies on the discriminator loss. However in a white-box attack where the adversary has knowledge of the ransomware detector, the goal of the attacker is to cause the generated samples from the malicious class to be misclassified as benign. Hence we can include a third term, with respect to the ransomware detector, to the loss function as follows:

\[
L_{detector} = \mathbb{E}(\log P_G(\hat{Y} = 0|Y = 1, C = 0)).
\]

The stopping criterion for training is the loss of the discriminator. After training, we can generate both fake malicious samples \( G_m \) and fake benign samples \( G_b \). From an attacker’s perspective, it is more desirable to generate malicious samples, bypass detection and increase false negative rate. Hence we focus on \( G_m \) for subsequent analysis and experiments. Each generated sample is of size \( 28 \times 28 \), so we flatten the sample to 1-dimensional segments of length 784 and round the generated sample to the closest integer value. For abuse of notation, we denote this set as \( G_m \).

![Figure 5: The ACGAN architecture for generating reshaped execution segments. Left table: the architecture of the generator, where the input \( z \) is latent variable and the output \( g \) is a generated 2-D execution log segment. Right table: the architecture of the discriminator, where the inputs are the 2-D execution log segments, the output \( y \in Y \) is predicted as benign or malicious via the auxiliary classifier, and the output \( c \in C \) is predicted as real or fake.](image)

4.3  Quality Assessment on the Generated Malicious Samples

Unlike in computer vision where the quality of the generated samples can be evaluated by visual inspection, evaluating the quality on dynamic execution logs requires a quantifiable metric. We propose a sample-based adversarial quality metric \( q_i \), where for each sample \( i \in G_m \)

\[
q_i = \frac{n_1(i)}{n_2(i)},
\]

where \( n_1(i) = |N_{i\cap m}| - |N_{i\cap m\cap b}| \), and \( n_2(i) = |N_{i\cap b}| - |N_{i\cap m\cap b}| \). Here, \(| \cdot |\) denotes the cardinality, \( N_{i\cap m} \) is the set of matched \( n \)-grams between the sample \( i \) and the malicious test set, \( N_{i\cap b} \) is the set of matched \( n \)-grams between the sample \( i \) and the benign test set and \( N_{i\cap m\cap b} \) is the set of matched \( n \)-grams among the sample \( i \), the test malicious set and the test benign set. Passing the quality check means that the generated samples contain more unique malicious samples than the unique benign samples. Since the real test data was not used for training the ACGAN, the proposed metric evaluates the generalized malicious properties that may not be found from the training set.

For a generated set \( G_m \), we calculate the quality metrics for each sample and filter the samples whose quality metric is below a pre-specified threshold \( r \). Suppose we expect to generate \( K \) malicious samples and \( K_0 \) samples have \( q < r \). Then we regenerate a smaller set of \( G'_m \) and repeat the process until we obtain \( K \) desired quality samples.

Similarly for the entire set \( G_m \), we propose a batch-based adversarial quality metric \( Q \) to statistically summarize the set of \( q_i \) for all \( i \in G_m \). The summary statistics are minimum, first quartile, median, lower quartile, minimum and outliers.

We summarize the adversarial quality assessment procedure in Algorithm 1.

4.4  Log Generation and Evaluation

The number of ways to concatenate the generated segments from \( G_m \) is approximately \( m^{784} \). In our experiment, since all the segments in \( G_m \) pass quality assessment, we can randomly concatenate the
individual segments. We note that for even stronger attacks, the attacker can optimize the concatenation based on some optimization objective, and this is one of our next research steps.

The generated malicious samples, after quality assessment in Sec. 4.3, are fed into the ransomware classifier. The adversarial detection rate is defined as the number of correctly predicted adversarial samples divided by the total number of adversarial samples. From a defender’s perspective, we can use the adversarial detection rate as another metric to quantify how resilient the malware detector is against adversarial attacks.

### 4.5 Summary of Proposed Methodology

In Algorithm 2, we summarize our framework of training ACGAN to generalize dynamic ransomware features and using a set of quality metrics to statistically evaluate the maliciousness of the generated samples.

#### Algorithm 2 Generate dynamic adversarial logs to bypass ransomware detector.

**Input:** Desired number of generated malicious samples \( K \), quality threshold \( \tau \), training data

**Step 1:** Segmentation and dimension conversion.

**Step 2:** Train AC-GAN.

**Step 3:** Generate \( G_m \) such that \( |G_m| = K \).

**Step 4:** Apply quality assessment procedure on \( G_m \) as in Algorithm 1.

if \( K - K_0 = 0 \) then

Stop

else

Generate \( G_m' \) with \( |G_m'| = K - K_0 \). Repeat until all generated segments pass quality assessment.

end if

**Step 5:** Concatenation.

**Step 6:** Feed the logs into ransomware detectors.

We consider several popular classifiers including Text-CNN[36], XGBoost[17], linear discriminant analysis (LDA), Random Forest[8], Naive Bayes[43], support vector machine with linear kernel (SVM-linear), and support vector machine with radial kernel (SVM-radial). For fair comparison, all classifiers are trained on the same sequences and no further feature extraction such as \( n \)-gram is performed prior to the classification algorithms. We note that the raw features are not \( 1 \)-gram modeling, which counts event occurrences. We report the classification accuracy, false positive rate (FPR), true positive rate (TPR), \( F_1 \)-score and area under the ROC curve (AUC) for all selected classifiers.

As seen in Table 2, Text-CNN achieves the highest accuracy at 0.9890, low false positive rate at 0.03, highest true positive rate at 0.9989, highest F-score at 0.9796 and highest AUC at 0.9950 among all other selected classifiers. XGB performs second best with accuracy at 0.931 and lowest false positive rate at 0.03. All other classifiers either suffer from low accuracy or high false positive rate. However, we expect \( n \)-gram feature extraction will greatly improve the other classifiers’ performance.

Due to Text-CNN’s superior performance, we naturally use it as a feature extractor via the last pooling layer and retrain all the other classifiers on the embedding extracted via Text-CNN. We observe significant improvement of other classifiers composed with Text-CNN, as seen in Table 3.

### 5 EXPERIMENT RESULTS

#### 5.1 Ransomware Classification on Real Data

Machine learning can be efficient, scalable and accurate at recognizing malicious attacks. We first demonstrate its benefits for highly effective ransomware detection. The training and testing ratio is set at 80% : 20%, where the training set contains 1292 benign samples and 3736 malicious samples, and the test set contains 324 benign samples and 934 malicious samples. After feature mapping, each execution log is represented as a sequence of events, and the sequence length is set to be 3000, where shorter sequences are padded with zeros to ensure same length.
Figure 6: Class-conditional density plot for each dimension in Text-CNN feature space. Red denotes the malicious class and blue denotes the benign class. Text-CNN as a feature extractor helps separate the samples from two classes, as indicated by the density plots. The features extracted from Text-CNN are in $\mathbb{R}^{32}$.

Figure 7: ROC curves of XGB, LDA, SVM compared with XGB $\circ$ Text-CNN, LDA $\circ$ Text-CNN and SVM $\circ$ Text-CNN. When using Text-CNN as a feature extractor and retraining XGB, LDA, SVM in the Text-CNN embedding subspace, all the composed classifiers possess significantly higher classification efficacy measured by AUC, $F_1$-score, accuracy FPR and TPR.

SVM $\circ$ Text-CNN. In our experiment results, Text-CNN, whether as a classifier on its own or as a feature extractor, is most likely to be selected by a security defender. Although knowledge of the defender’s ransomware classifier is not needed by our analysis methodology, we evaluate the adversarial detection rate against Text-CNN based classifiers.

5.2 Generate Adversarial Segments

We follow the steps in Section 4.2.2 to train an AC-GAN [48], where we set the batch size to be 100, the latent dimension to be 100, and the training is stopped at the 80-th epoch. After training, we obtain 5029 segments from the malicious class $Y = 1$. We round the segments to the nearest integer and denote this set as $G_m$.

5.3 Adversarial Quality Assessment

A successful evasion means the generated malicious samples not only fool ransomware classifier, but also persist maliciousness based on certain metrics. Following Section 4.3, we compute the adversarial quality metric $q_i$ of each GAN-generated sample for $n$-grams with $n \in \{3, 4, \ldots, 7\}$. Figure 8 shows the quality metric in $y$-axis against each generated segment in $x$-axis for 4-, 5-, 6-grams. We set the quality threshold to be $\tau = 1.5$, which means a qualified generated segment with statistically measured maliciousness would need to match over 50% of the unique malicious $n$-grams than the unique benign $n$-grams.

We also plot the batch-based quality metric $Q$ for $n \in \{3, 4, \ldots, 7\}$-grams, as represented in boxplots in Figure 9. As shown in the box-plots, all the generated segments are statistically much closer to the real malicious class with $q_i \geq \tau$ and $\min(q_i) = 1.9$.

All the generated and qualified segments are concatenated randomly to produce 1257 execution logs.

5.4 Evasion

The highly performing ransomware detectors Text-CNN, XGB $\circ$ Text-CNN, LDA $\circ$ Text-CNN, Random forest $\circ$ Text-CNN, Naive Bayes $\circ$ Text-CNN, SVM-linear $\circ$ Text-CNN, SVM-radial $\circ$ Text-CNN are applied on the adversary-generated logs. We report the number of detected samples and the detection rate in Table 4. Most of the classifiers significantly degrade in detection performance, where Text-CNN, LDA $\circ$ Text-CNN, Naive Bayes $\circ$ Text-CNN, SVM-linear $\circ$ Text-CNN fail to detect any generated malicious samples, while XGB $\circ$ Text-CNN detects 12.73% correctly and Random forest $\circ$ Text-CNN detects 36.35% correctly. The most robust classifier turns out to be SVM-radial $\circ$ Text-CNN in this experiment with 100% detection rate. This can be due to its nonlinear boundary in the Text-CNN latent feature space. However only one classifier out of all seven highly effective classifiers is resilient to our bypass scheme. Our adversarial detection result clearly indicates that this is a potential vulnerability for ML-based ransomware detection systems.
We investigate why most of the highly effective classifiers fail to predict the adversarially generated samples correctly. We use the last pooling layer from Text-CNN as a feature extractor and refer to the space of features extracted by Text-CNN as the latent feature subspace. The classifiers that achieve effective and competitive classification performance are XGB, LDA, Random Forest, Naive Bayes and SVM trained in the latent feature subspace. Text-CNN classifier itself has linear boundaries via the fully connected layer in the latent feature subspace. Hence one natural investigation starts at how the generated samples and the real samples relate in the latent feature subspace induced by Text-CNN, in comparison with the density of the original sample distribution.

Represented in 2-D visualization, Figure 10 shows that the generated samples, in dark red, lie close to a linear boundary but much closer to the real benign samples in the Text-CNN latent feature subspace. However as shown in Section 5, most of the generated samples match more than twice of the unique ransomware signatures than the unique benign signatures. This motivates us to explore the $L_2$ distance between the real malicious samples and real benign samples, as well as between the generated samples and the real samples in both the latent feature subspace and the original feature space.

Denote the latent features of the generated malicious logs as $F_g$, the latent features of the training malicious logs as $F_{tr,m}$ and the latent features of the training benign logs as $F_{tr,b}$. Similarly, for the test data, the latent malicious and benign features are denoted as $F_{te,m}$ and $F_{te,b}$ respectively.

We plot the density of the $L_2$-distances between test malicious data and training data, both of which are real samples. The left figure in Figure 11 shows, in the original feature space, the density of the $L_2$ distance $D_{tr,te,m}$ between the malicious test logs and the training malicious logs in red and the density of the $L_2$ distance $D_{tr,te,b}$ between the malicious test logs and the training benign logs in blue. The dashed red and blue vertical lines represent the means of $D_{tr,te,m}$ and $D_{tr,te,b}$ respectively. On average, the malicious test logs are closer to the training malicious logs than to the training benign logs. However in the original data space, the distributions of distances are not very well-separated and this is also reflected in the algorithm performance on the original data space as shown in Table 2.

The right figure in Figure 11 plots the density of the $L_2$ distance $d_{tr,te,m}$ between $F_{te,m}$ and $F_{tr,m}$ in red and the density of the $L_2$ distance $d_{tr,te,b}$ between $F_{te,m}$ and $F_{tr,b}$ in blue. The dashed red and blue vertical lines represent the means of $d_{tr,te,m}$ and $d_{tr,te,b}$ respectively. $F_{te,m}$ is much closer to $F_{tr,m}$ than to $F_{tr,b}$. The distances are consistent across original feature space and the latent feature subspace. This observation is expected since the malicious samples should be close together in either feature space.

Next we investigate whether the observed phenomenon extends to the generated samples and real samples. The left figure in Figure 12 plots, in the original feature space, the density of the $L_2$-distance $D_{tr,g,m}$ between the generated logs and the training malicious logs in red and the density of the $L_2$ distance $D_{tr,g,b}$ between the generated logs and the training benign logs in blue. The dashed red and blue vertical lines represent the means of $D_{tr,g,m}$ and $D_{tr,g,b}$ respectively. The generated malicious logs are much closer to the real malicious logs than to the real benign logs in the original feature space.

The right figure in Figure 12 plots, in the latent feature space, the density of the $L_2$-distance $D_{tr,g,m}$ between $F_g$ and $F_{tr,m}$ in red and the density of the $L_2$ distance $D_{tr,g,b}$ between $F_g$ and $F_{tr,b}$ in blue. The dashed red and blue vertical lines represent the means of $D_{tr,g,m}$ and $D_{tr,g,b}$ respectively. $F_g$ is much closer to $F_{tr,b}$ than to $F_{tr,m}$. Figure 12 shows that in the Text-CNN feature subspace, the generated logs are closer to the benign logs, while in the original feature space, the generated logs are closer to the malicious logs. This phenomenon indicates that the generated adversarial samples lie in the blind spot of the Text-CNN algorithm.

### Table 4: Adversarial detection rate on the generated malicious samples

| Classifier                  | No. detected | Detection rate (%) |
|-----------------------------|--------------|--------------------|
| Text-CNN                    | 0            | 0                  |
| XGB $\circ$ Text-CNN        | 16           | 12.73              |
| LDA $\circ$ Text-CNN        | 0            | 0                  |
| Random forest $\circ$ Text-CNN | 457         | 36.35              |
| Naive Bayes $\circ$ Text-CNN | 0            | 0                  |
| SVM-linear $\circ$ Text-CNN  | 0            | 0                  |
| SVM-radial $\circ$ Text-CNN  | 1257         | 100%               |

5.5 Latent Feature Space Investigation

We investigate why most of the highly effective classifiers fail to predict the adversarially generated samples correctly. We use the last pooling layer from Text-CNN as a feature extractor and refer to the space of features extracted by Text-CNN as the latent feature subspace. The classifiers that achieve effective and competitive classification performance are XGB, LDA, Random Forest, Naive Bayes and SVM trained in the latent feature subspace. The non-linear boundary of SVM-radial $\circ$ Text-CNN effectively detects the adversarial samples.

![Figure 10: Visualization of the Text-CNN extracted features for (left) PC-dimension 1 vs PC-dimension 3; (middle) PC-dimension 2 vs PC-dimension 3; (right) PC-dimension 2 vs PC-dimension 3. The generated malicious samples are colored in dark red, and lie closer to the benign set in Text-CNN subspace. We draw the 95% data eclipse around the scattered points.](image1)

![Figure 11: Density plot of the distances between real benign and real malicious logs in both original feature space and Text-CNN latent feature space.](image2)
We describe a framework via generative adversarial network to synthesize ransomware samples and propose a set of adversarial quality metrics via statistical similarity to quantify the maliciousness of the GAN-generated samples. We demonstrate that the unsupervised ransomware detection algorithms were not considered in this paper. As many unsupervised machine learning algorithms have been developed, how to properly access their resiliency and trustworthiness is also important. In this work, we propose and develop a ransomware simulation program to demonstrate how to extend our adversarial generation framework to semi-supervised malware classification tasks.

Our proposed framework should be utilized as a defense capability for developing a resilient model for detecting ransomware in the field. As described in Section 4.4, a defender can use the adversarial detection rate as a metric to quantify the resilience of the ransomware detector against adversarial attacks. The defender can use the GAN-generated samples as part of the training procedure to update the defender’s classifier.

Our proposed quality assessment approach can be leveraged even when the model is deployed and is in use in the field to track the changes in distance between generated and real samples. These robustness mechanisms must be considered as an integral part of an adversary-resilient malware classifier.

Our case study for evaluating a broad range of ransomware classifiers also demonstrates the pitfalls in selecting classifiers based on high accuracy and low false-positives which is typical today in malware detection. After a deeper analysis of generating quality adversarial samples, the most robust classifier is verified to be SVM-radial=Text-CNN in our experiment. This analysis may form the basis of selecting multi-classifier ensemble-based approaches to act as a defense-in-depth against adversarial probing attacks once the ransomware classifiers are deployed in the field. In our specific case study, a weighted score between the XGB=Text-CNN classifier and the SVM-radial=Text-CNN classifier gives the defender much more coverage in the space of execution logs for ransomware.

It is important to note that our framework is still useful to enforce the resiliency of the ransomware detection model even when the model is deployed on a platform using software and hardware-based Trusted Execution Environments (TEEs) that protect the run-time confidentiality and integrity of the classifier(s) while in-use - providing the defender with an additional tool to continue to enforce the security objectives consistently even post the training stages.

6 DEPLOYMENT OPPORTUNITY

We develop a ransomware simulation program to demonstrate how to generate actual malicious file I/O operations based on the GAN generated feature sequences, as seen in Figure 13. When the program sees feature sequences such as “File rename”, “File created” and “File change with high entropy”, it will rename an arbitrary file, create the other arbitrary file and then encrypt a victim file. The I/O events collected when executing the simulated ransomware is very close to the GAN generated feature sequence, such that the simulated ransomware can bypass ML detection while maintaining malicious encryption behaviors. We make sure the I/O operations are completed in a short time window to minimize noisy I/O events added from other benign processes.

7 DISCUSSION

We describe a framework via generative adversarial network to synthesize dynamic ransomware samples and propose a set of adversarial quality metrics via statistical similarity to quantify the maliciousness of the GAN-generated samples. We demonstrate that six of the seven highly effective ransomware classifiers fail to detect most of the GAN-generated samples.

In our next steps, we will continue developing an automatic tool for monitoring applications and harvesting more benign logs. The addition of benign samples can augment our training set to better reflect the practical scenarios of more benign-ware than malware. We also plan to test the real-world efficacy of our proposed adversarial log generation system against the machine learning based anti-malware solutions in the market. Additionally, while we hope to extend our adversarial generation framework to semi-supervised or unsupervised malware classification tasks.

REFERENCES

[1] Bander Ali Saleh Al-rimy, Mohd Aizaini Maarof, and Syed Zainuddin Mohd Shaid. Ransomware threat success factors, taxonomy, and countermeasures: A survey and research directions. Computers & Security, 2018.
[2] Omar Mk Alhawi, James Baldwin, and Ali Dehghantanha. Leveraging machine learning techniques for windows ransomware network traffic detection. Cyber Threat Intelligence, pages 93–106, 2018.
[3] Hyrum S Anderson, Jonathan Woodbridge, and Bobby 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.
[4] Tseng Aragorn, Chen YunChun, Kao YiHsiang, and Lin Tsungnan. Deep learning for ransomware detection. IEICE Technical Report; IEICE Tech. Rep., 116(282):87–92, 2016.
[5] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. arXiv preprint arXiv:1701.07875, 2017.
[6] Vineeth S Bhaskara and Debanjan Bhattacharyya. Emulating malware authors for proactive protection using gan over a distributed image visualization of the dynamic file behavior. arXiv preprint arXiv:1807.07525, 2018.
[7] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. Pattern Recognition, 84:317–331, 2018.
[8] Leo Breiman. Random forests. Machine learning, 45(1):5–32, 2001.
[9] Krzysztof Cabr´, Marcin Gregorczyk, and Wojciech Mazureczyk. Software-defined networking-based crypto ransomware detection using http traffic characteristics. Computers & Electrical Engineering, 66:353–368, 2018.
[10] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, pages 3–14. ACM, 2017.
Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. Black-box adversarial examples. In Advances in Neural Information Processing Systems (NeurIPS), 2017.

Nicholas Carlini and David Wagner. Audio adversarial examples: Targeted attacks on speech-to-text. arXiv preprint arXiv:1801.01944, 2018.

Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. In 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017. pages 39–57, 2017.

Nicholas Carlini and David A. Wagner. Audio adversarial examples: Targeted attacks on speech-to-text. CoRR, abs/1801.01944, 2018.

Li Chen. Deep transfer learning for static malware classification. arXiv preprint arXiv:1812.07606, 2018.

Li Chen, Salmin Sultana, and Ravi Saita. Heten: A deep learning approach on intel® processor trace for effective exploit detection. In 2018 IEEE Security and Privacy Workshops (SPW), pages 109–115. IEEE, 2018.

Li Chen, Mengwei Zhang, Chih-Yuan Yang, and Ravi Saita. Semi-supervised classification for dynamic android malware detection. arXiv preprint arXiv:1704.05947, 2018.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining, pages 785–794. ACM, 2016.

Yun-Chun Chen, Yu-He Li, Aragon Tseng, and Tsungnan Lin. Deep learning for malicious flow detection. In Personal, Indoor, and Mobile Radio Communications (PIMRC), 2017 IEEE 28th Annual International Symposium on, pages 1–7. IEEE, 2017.

Andrea Contiella, Alessandro Guagnelli, Giovanni Zingaro, Giulio De Pasquale, Alessandro Barenghi, Stefano Zanero, and Federico Maggi. Shieldfs: a self-healing, ransomware-aware filesystem. In Proceedings of the 32nd Annual Conference on Computer Security Applications, pages 336–347. ACM, 2016.

Nilesh Dalvi, Pedro Domingos, Manmohan Singh, Sanghui, and Deepak Verma. Adversarial classification. In Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD ’04, pages 99–108, New York, NY, USA, 2004. ACM.

Nilesh Dalvi, Pedro Domingos, Sumit Sanghai, Deepak Verma, et al. Adversarial classification. In Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, pages 99–108. ACM, 2004.

Nilaksh Das, Madhuri Shanbhogue, Shang-Tse Chen, Fred Holden, Li Chen, Michael E Kounavis, and Duen Horng Chau. Shield: fast, practical defense and vaccination for deep learning using JPEG compression. arXiv preprint arXiv:1705.02900, 2017.

Nilaksh Das, Madhuri Shanbhogue, Shang-Tse Chen, Fred Holden, Siwei Li, Li Chen, Michael E Kounavis, and Duen Horng Chau. SHIELD: fast, practical defense and vaccination for deep learning using JPEG compression. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018. pages 196–204, 2018.

Sumanth Dathathri, Stephan Zheng, Richard M. Murray, and Yisong Yue. Deep learning for automated dynamic analysis of ransomware: Benefits, limitations and use for detection. arXiv preprint arXiv:1806.03496, 2018.

Emily L. Denton, Soumith Chintala, Rob Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In Advances in neural information processing systems, pages 1486–1494, 2015.

J A Gómez-Hernández, L Álvarez-González, and Pedro García-Teodoro. R-locker: Automatic dynamic analysis of ransomware: Benefits, limitations and use for detection. arXiv preprint arXiv:1708.03870, 2018.

Andrew Ilyas, Logan Engstrom, Amish Alhalye, and Jesey Lin. Black box adversarial attacks with limited queries and information. arXiv preprint arXiv:1804.08598, 2018.

Jinho Jung, Chanil Jeon, Max Wolotsky, Insu Yum, and Taesoon Kim. AVPASS: Leaking and Bypassing Antivirus Detection Model Automatically. In Black Hat USA Briefings (Black Hat USA), Las Vegas, NV, July 2017.

Harini Kannan, Alexey Kurakin, and Ian J. Goodfellow. Adversarial logit pairing. CoRR, abs/1803.06373, 2018.

Amir Khazaee, Seok-Jong Oh, and Christian Szegedy. Explaining and harnessing adversarial examples. arXiv, abs/1412.6572, 2014.

Nikola Hampton, Zubair Baig, and Sherali Zeadally. Ransomware behavioural analysis on windows platforms. Journal of information security and applications, 40:44–51, 2018.

Weimei Hu and Ying Tan. Generating adversarial malware examples for black-box attacks based on gan. arXiv preprint arXiv:1702.05983, 2017.

Andrew Ilyas, Logan Engstrom, Amish Alhalye, and Jesey Lin. Black-box adversarial attacks with limited queries and information. arXiv preprint arXiv:1804.08598, 2018.

Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. CoRR, abs/1607.02533, 2016.

Aleksandr Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. CoRR, abs/1706.06083, 2017.

Aleksandr Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. arXiv preprint arXiv:1706.06083, 2017.

Mafee. McAfee Labs Threats Report March 2018. https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-mar-2018.pdf, 2018.

McAfee. McAfee Labs Threats Report September 2018. https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-sep-2018.pdf, 2018.

McAfee. Understanding Ransomware and Strategies to Defeat it. https://www.mcafee.com/enterprise/en-us/assets/white-papers/wp-understanding-ransomware-strategies-defeat.pdf, 2018.

Augustus Odena, Christopher Olah, and Jonathan Shlens. Conditional image synthesis with auxiliary classifier gans. arXiv preprint arXiv:1610.06958, 2016.

Nicola Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In IEEE European Symposium on Security and Privacy, EuroSP 2016, Saarbrücken, Germany, March 21-24, 2016. pages 372–387, 2016.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434, 2015.

Konrad Rieck, Philipp Trinin, Carsten Willems, and Thorsten Holz. Automatic analysis of malware behavior using machine learning. Journal of Computer Security, 19(4):639–668, 2011.

Nolan Scaife, Henry Carter, Patrick Traynor, and Kevin Butler. Cryptolocker (and drop it): Stopping ransomware attacks on user data. IEEE 36th International Conference on Distributed Computing Systems, 2016.

Nolen Scaife, Henry Carter, Patrick Traynor, and Kevin RB Butler. Cryptolocker (and drop it): Stopping ransomware attacks on user data. In Distributed Computing Systems (ICDCS), 2016 IEEE 36th International Conference on, pages 303–312. IEEE, 2016.

Daniele Scaglione, Psachos Katsaros, and Eleni Diamanti. Automated dynamic analysis of ransomware: Benefits, limitations and use for detection. arXiv preprint arXiv:1809.03020, 2016.

Daniele Scaglione, Psachos Katsaros, and Eleni Diamanti. Automated dynamic analysis of ransomware: Benefits, limitations and use for detection. arXiv preprint arXiv:1809.03020, 2016.

Ashkan Shariari Shamil, Christian Bauchkang, and Tansu Alpcan. Malware detection on mobile devices using distributed machine learning. In Pattern Recognition (ICPR), 2010 20th International Conference on, pages 4348–4351, IEEE, 2010.

S. Sato and J. Zadeh. Automated prevention of ransomware with machine learning and gops. In Presented as the RSA Conference 2017, San Francisco, CA, 2017.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. CoRR, abs/1312.6193, 2013.

R Vinayakumar, KP Soman, KK Senthil Velan, and Shaunak Ganorkar. Evaluating shallow and deep networks for ransomware detection and classification. In Advances in Computing, Communications and Informatics (ICACCI), 2017 International Conference on, pages 259–265. IEEE, 2017.

Chaowei Xiao, Bo Li, Jun-Yan Zhu, Warren He, Mingyan Liu, and Dawn Song. Generating adversarial examples with adversarial networks. arXiv preprint arXiv:1701.02610, 2018.

Wei Lin, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. In 25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 26-28, 2018. 2018.

Chihyuan Yang and Li Chen. GitHub: Resilient-ml-research-platform. https://github.com/Resilient-ML-Research-Platform, 2018.