AIMED: Evolving Malware with Genetic Programming to Evade Detection

Raphael Labaca Castro, Corinna Schmitt, and Gabi Dreo Rodosek

Research Institute CODE
Bundeswehr University Munich, Germany
{Raphael.Labaca, Corinna.Schmitt, Gabi.Dreo}@unibw.de

***Abstract*—Genetic Programming (GP) has previously proved to achieve valuable results on the fields of image processing and arcade learning. Similarly, it can be used as an adversarial learning approach to evolve malware samples until static learning classifiers are no longer able to detect it. While the implementation is relatively simple compared with other Machine Learning approaches, results proved that GP can be a competitive solution to find adversarial malware examples comparing with similar methods. Thus, AIMED – Automatic Intelligent Malware Modifications to Evade Detection – was designed and implemented using genetic algorithms to evade malware classifiers. Our experiments suggest that the time to achieve adversarial malware samples can be reduced up to 50% compared to classic *random* approaches. Moreover, we implemented AIMED to generate adversarial examples using individual malware scanners as target and tested the evasive files against further classifiers from both research and industry. The generated examples achieved up to 82% of cross-evasion rates among the classifiers.**

***Index Terms*—AIMED, Genetic Programming, Malware, Byte-level perturbations, Adversarial learning**

I. Introduction

The development of new threats is still an arms-race between criminals and researchers. The easier it is to find new ways to develop malware, the most important it is for security researchers to understand how to prevent these attacks. Detecting malicious software using machine learning techniques has been increasingly embraced by security companies for a while now [1]–[3].

Nevertheless, advances in adversarial machine learning have shown that white box learning models [4] are vulnerable to gradient-based attacks and non-differentiable algorithms [5] that report a score for detection can be compromised by genetic algorithms. Furthermore, recent approaches to attack black box learning models using Reinforcement Learning (RL) have proved to create successful mutations, though functionality is not always guaranteed for the new malware mutations [6]. Using a Generative Adversarial Networks (GAN) was also proposed [7] against black box models to bypass detection. In that approach they use a neural network surrogate model to fit the target classifier and a generator to create the evasive examples. However, the authors assume that the attackers should be fully aware of the features implemented in the model, which can be challenging for practical scenarios. While machine learning has drastically improved its accuracy and scores, it has been shown [4] that neural networks exhibit a non-reliable behavior when confronted with unexpected input, so called adversarial examples. These have been widely studied in the literature mostly in the image domain [8], whereas altering the structure of Windows Portable Executable (PE) files can compromise its functionality [9]. In addition to that, once a piece of malware is detected it is usually no longer useful for cybercriminals. Hence, committed adversaries are required to create new techniques to bypass detection.

We therefore present **AIMED** – **A**utomatic **I**ntelligent **M**alware Modifications to **E**vade **D**etection – a genetic programming (GP) approach to automatically find optimized modifications that, when injected into the previously-detected malware, will result in misclassification by malware scanners.

The underlying idea is to manipulate the malware structure in order to increase the likelihood that the modified sample will no longer be detected by a classifier while making sure its functionality is preserved. Automatically generating adversarial examples for malware scanners can be interesting from the security perspective to understand how byte-level modifications impact malicious samples without corrupting them. Furthermore, static malware detection plays an important role in security as it allows to proactively detect malicious threats prior to its execution. Therefore, we aim to provide an environment that can help researchers both in academia and the industry to further improve malware detection. To the best of our knowledge this is the first time GP is implemented to find adversarial malware examples while returning valid PE files as output. Most of the work in adversarial learning using GP in the past focused on mobile malware [10]–[12], or further related topics including malware evolving, PDF files classification and Intrusion Detection Systems [5], [13]–[16].

We run our experiments using 6880 Windows PE files and generate over 5500 malware mutations. We tested our adversarial examples against state-of-the-art classifiers and compared against similar work in the literature. AIMED demonstrates that it can converge 50% faster than random approaches and generate more robust examples that are fully functional and able to cross-evade other classifiers with up to 80% success rate. Therefore, we aim to provide evidence that motivated adversaries can use GP to be able to bypass state-of-the-art malware detection within a few minutes regardless of the classifier analyzed.

The rest of this paper is structured as follows: Section II provides the background and related work that inspired design decisions made and described in Section III. In Section IV we introduce the AIMED framework and evaluate its results in Section V. Finally, we conclude our paper in Section VI.

II. Background

This section lays ground for the design of AIMED in Section III by introducing related work on genetic programming and adversarial learning in malware classification as both topics are combined in AIMED to improve how evasive malware is achieved.

A. Adversarial Learning in Malware Classification

Adversarial examples in deep learning models have been getting a lot of traction in recent years. Early contributions focused on adversarial learning techniques targeting Intrusion Detection Systems algorithms to force misclassification [17]. Further research by Szegedy et al. also conducted adversarial techniques on deep neural networks for image classification by injecting imperceptible perturbations that mislead the model to classify the images incorrectly [4]. Goodfellow et al. presented a gradient-based approach that focused on neural networks' linearity in order to generate adversarial examples [18]. Papernot et al. introduced a Jacobian Matrix that provided high evasion results with a low rate of modification of the features used [19]. It was able to identify which features should be modified to generate the adversarial examples. Following the trend on gradient based approaches, Grosse et al. trained a feed forward neural network to bypass an Android malware scanner [20]. Kolosnjaji et al. also proposed a method based on a gradient attack that is capable of evading a deep neural network classification by modifying less than 1% of the samples' bytes. In most of these cases, though, it is assumed that the attacker has full knowledge of the model's architecture and therefore the gradient based algorithm can be applied [21].

Recent approaches have shifted the attack towards black box models where normally no information is given regarding architecture, weights, training samples or confidence scores. Hu et al. proposed a GAN [22] that creates both a surrogate model that fits the black box classifier and a second model that emulates the generative network, which is trained to generate adversarial samples. Once the surrogate model is properly trained, it performs cross-evasion to the actual black box classifier and, according to the authors, reports perfect evasion for the samples analyzed [7]. Recently, Anderson et al. proposed a black box attack on static malware detection based on an OpenAI gym [23] implementation. On this approach a RL agent is trained to inject perturbations to malware samples and provide a reward to the model towards bypassing the classifier. Even though the perturbations are functionality-preserving operations the new malware mutations are not checked whether they are valid executables and thus many evasions can end up as corrupt files [6].

In [9] we proposed ARMED, which is capable of generating automatic adversarial examples of a previously-known malware by injecting random perturbations until it is misclassified. Random perturbations are useful to find single adversarial examples timely but do not keep the performance when the problem scales up to find several evasive mutations [9].

The malware domain is by nature a different problem than image classification given that the input features are no longer continuous but rather binary and unlike the image domain, injecting perturbations into malware might render the sample corrupt and no longer useful. Therefore, it adds a new challenge in order to build successful adversarial examples.

B. Genetic Programming for Malware Evasion

Artificial intelligence has gone a long way since this question attributed to Arthur Samuel was asked "*How can computers learn to solve problems without being explicitly programmed?*" [24]. To tackle that central question different approaches were proposed and among them, the use of the biological theory of evolution in computer science. Hence, Holland demonstrated how evolution in nature can be applied to solve problems using a technique called genetic algorithms (GA) [25]. GA convert a $population$ of individuals with its respective $fitness$ through operations including $crossover$ and $mutation$[1] following the Darwinian principle. Combining that approach with computer programs brings us GP, which is an approach to automated machine learning that attempts to evolve programs to solve a variety of problems. It has been extensively studied for decades [24], [26]–[28].

Genetic Programming has been used in the literature to evolve different types of malware. Noreen et al. introduced an evolutionary framework to evolve a piece of malicious software called $Bagle$ [16]. Evolving is defined as changing the behavior of the malware sample, which brings a controversial approach to the discussion in malware classification since at which extent can behavior be modified and still be considered malicious. The scope of the study is limited to only known variants of the same malware by testing it against commercial security products. The main motivation seems to be evolving malware in a way that new mutations are detected as different variants of the same malware family. GP has also been already introduced to evolve malware and test against commercial security products in the mobile domain [10]–[12]. Kayacik et al. focused on evolve buffer overflow attacks to obfuscate them [13]. Further research introduced GP to create mimicry buffer overflow attacks with the purpose of finding vulnerabilities faster than attackers [14]. In [15] is proposed a comparison between white and black box attacks in intrusion detection systems based on GP. Xu et al. introduced a method that searches for PDF adversarial examples drawing ideas from GP by using API calls from PE files and testing them against PDF malware classifiers that return a classification score [5]. Recently, Calleja et al. introduced a genetic algorithm that is able to force an Android malware classifier to misclassify the family in the dataset by changing only one feature in the

[1]Note that we differentiate two types of mutations in this work by defining genetic mutations as changes into the sequence of perturbations and malware mutations, which are the modified malware files generated by AIMED.

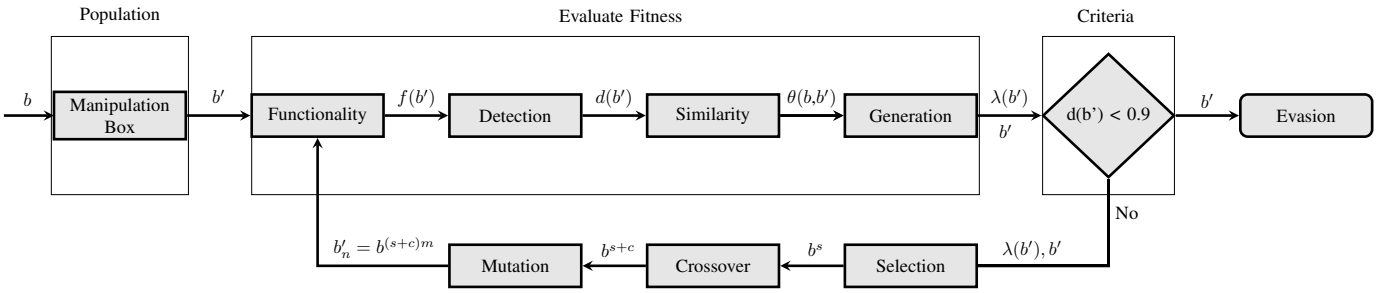

Fig. 1. AIMED workflow

original malware [29]. Nonetheless, the research is focused on targeted attacks forcing an Android malware to be categorized under a different malware signature (family) instead of bypassing the classifier and forcing it to label a malicious application as benign.

It should be noted that benchmarking different commercial products has been widely studied in the literature [30] [31]. AIMED does not follow this approach currently. We aim to test evolving malware mutations against robust classifiers and understand how sensitive static malware detection can be when confronted with automatically generated adversarial examples. Furthermore, how these samples can cross-evade different classifiers both in research projects as well as in the industry. GP proves to be a suitable tool as it looks for the best evasion sequence while reducing the search space. Moreover, unlike further machine learning methods, it does not require previous training time.

III. Design

A. PE File Injections

Windows Portable executable files were created from Microsoft to establish a way for Windows to run applications and to store the important data needed during execution [32]. PE files are an extension of Common Object File Format (COFF) [33], which is a format for executables introduced on Unix systems. Modifying PE files has been of interest in the past but the development of such tools are complex and expensive and thus sometimes leads to less tools shared with the community [34] [35]. Moreover, authors tend to implement their own parser linked to a specific language. Hence, LIEF [36] has been introduced to provide a cross platform library that is able to parse and modify different executable formats including PE.

We decided to implement defined [6] byte-level perturbations – modifications – that can be injected to Windows PE files. According to our experiments, we adjusted and selected nine of the perturbations that returned the highest number of valid malware mutations. Based on the number and order of perturbations injected, new malware mutations can bypass detection at the expense of its functionality, which means perturbations often lead to corrupt files that are no longer detected by malware classifiers. However, corrupt files are useless in spite of whether they are detected or not. Thus, we implement a sandbox to make sure all new mutations are functional before checking whether they are also evasive.

B. AIMED's Workflow

Our approach consists of eight main components and is inspired on the algorithm displayed in Fig. 2, which summarizes the genetic algorithm approach. The **Manipulation Box** in Fig. 1, which will be part of Step 1 in Fig. 2, where perturbations are injected to the malware sample in order to create the population.

In Step 2 each member of the population will be evaluated in terms of their fitness and receives a score. This is also highlighted on the AIMED workflow in Fig. 1. In order to perform the evaluation the new mutation runs through four different stages: (a) **Functionality** stage implements a sandbox to check whether the file is not corrupt and then sends it to the next stage. (b) **Detection** scans every generated mutation using commercial engines or research models to determine whether it is malicious. Based on the results, the new malware mutation can be: corrupt, valid but detected, or valid and evasive. Each of the states receive a score where *evasive* returns the highest value and *corrupt* the lowest. (c) A **Similarity** function that calculates the differences between any given malware mutation and the original file where it came from at a binary-level representation. (d) **Distance from origin** stage returns the number of current generation and is used to give more preference to recent members of the population. All this stages combined contribute their value to the fitness function, which will be used in the following steps of the GP algorithm.

In Step 3, **Selection**, the two fittest members will be selected to breed the next generation. During the first generation, though, two scenarios may open. Either scores are initialized in zero and the two first generated malware mutations will be selected or all of the members shall have their fitness score assigned before proceeding to the next generation, which is the approach we decided to adopt.

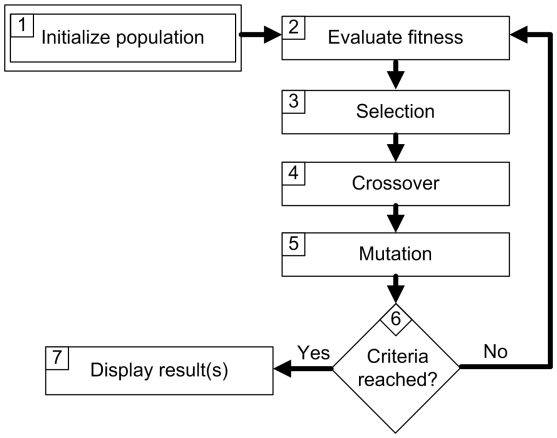

Fig. 2. Standard genetic algorithm

In Step 4, called **Crossover**, the two fittest malware mutations mate with each other and generate new children that are more prone to result in a valid evasion.

In order to avoid getting stuck in a loop because of only crossing-over the best members and producing the same offspring, random genetic mutations are added in Step 5, called **Mutation**, evolving the malicious files in unexpected ways similar to evolution in nature and all new generated malware files are sent back to the loop to assess their fitness.

The whole process is repeated over many generations until a number of adversarial samples is achieved or a threshold of generations is surpassed as pictured in Step 6, which is also highlighted as **Criteria** in Fig. 1. Finally, the evasive malware mutations are displayed, which are successful in bypassing the malware classifier.

As we can observe in Fig. 1, we adjust the termination clause right after the fitness evaluation, specifically after the detection stage is tested. That is basically because looking for an adversarial example relies exclusively on having a successful evasive mutation and hence it may provide a more efficient result when checking earlier throughout the flow.

We implemented as a malware classifier four different options, three top commercial scanners and one research model: Gradient Boosted Decision Tree (GBDT), Sophos, ESET, and Kaspersky. Given that Kaspersky proved to give a good performance by detecting a high number of initially sampled malware mutations, we decided to test AIMED using it extensively in order to obtain more robust adversarial examples to eventually cross-evade the other three classifiers in what could be a more realistic scenario. Nonetheless, we set holdout samples for each classifier and tested against each other to compare the impact of evasive mutations among different black box classifiers.

IV. AIMED Framework

A modified version of a malware analysis environment [37] is implemented to act as the local sandbox. In a first implementation, we started tagging new malware mutations as functional very cautiously to avoid false positives. The malware analysis environment tracks how the sample behaves and gives us different structure metrics including a score based on malware behavior and time processing. We determined that corrupt files usually return shorter than average process times and malicious scores close to zero. However, in a later implementation we removed this condition because it was leaving out potential candidates and improved the functionality test by implementing a dedicated system without the need to run all sandbox's components.

As malware classifiers, both top commercial [1]–[3] and research machine learning models [6] are used. According to the authors of the latter, the research model was trained using 100.000 benign and malicious files and reports a ROC-AUC score of 0.993. We implemented a threshold of 0.9 that corresponds to a 90% true positive rate while keeping a 1% false positive rate.

Additional API modules are also available in AIMED to implement malware engine aggregators (e.g., VirusTotal, Metadefender) in case that a pool of engines is meant to be tested simultaneously. Note that these options may require premium services and can increase the processing time to several minutes per generation and thus were not adopted in these experiments.

The whole process including generating a new malware mutation and sending it through the sandbox and detection stages take currently between 15 – 30 seconds. That means, the GP algorithm needs to be implemented in a way where only the most relevant mutations need to be processed.

A. Genetic Programming

In order to optimize the efficiency of our approach, different steps need to be taken: (1) Search for an optimal population number. Small populations are faster in the first generation but provide only a limited number of genes to start with, whereas for larger population numbers a more diverse first generation can increase the probability of a strong generation afterwards. (2) While duplication is not allowed in the first generation, it is later in the process, which means diversity decreases. In our approach, change is not implemented adiabatic as new members are created while older are not always killed or replaced. Instead, the fittest members along with its offspring pass through the next generation. (3) Next generation's offspring can lead to repeated genes as the fittest members might continue for several generations as parents. Thus, to avoid duplicated members to perpetually remain as the fittest candidates, we added a control mechanism to swap those with the same fitness value but different sequence of perturbations. That will help to increase diversity in the population of every generation.

Moreover, genetic mutation rates should be further studied to determine optimal values along with the size of population. Our experiments have shown that relatively low mutation rates (e.g., 10%) are enough to outperform random algorithms convergence rates. In the current implementation this parameter can be specified at the beginning and depending on its value it will bring further diversity to the next generations.

Algorithm 1 Finding evasive malware mutations with GP

Functions:
S: Selection;
C: Crossover;
M: Mutation;
Input:
Input malware $b \in B^*$;
Byte-level perturbations δ;

$b_i \leftarrow B^*$
$b'_i \leftarrow b_i + \delta$
while $True$ **do**
  for $f(b'_i) = 1$ **do**
    if $d(b'_i) < 0.9$ **then**
      return b'_i
    end if
  end for
  $b'_s \leftarrow S(\lambda(b'_i))$
  $b'_{i+1} \leftarrow M(C(b'_s))$
  if $g > \gamma$ **then**
    return $\emptyset$
  end if
end while

However, the higher the mutation rate is the closer to a random approach the GP results will be.

B. Formalization

As per [38] we formulate the problem and formally define the notation as follows. A PE is composed by a sequence of bytes b with length n where $b = (b_1, b_2, ..., b_n)$ and $b_i \in B$ for all $1 \leq i \leq n$. B^* is defined as finite sequences of B and thus $b \in B^*$. The classifier, or malware scanner, is used to label an input file as benign or malicious and is defined as $d : b \in B^* = \{0, 1\}$. The output is considered malicious, labeled as 1, in case it's greater than 0.9 for the GBDT model. We define as $f : b \in B^* = \{0, 1\}$ the function to determine if the functionality of b is preserved. The sandbox returns 0 for corrupt and 1 for valid malware mutations.

A $perturbation$, denoted by δ, is a specific modification injected into any given malware resulting in a modified sample b' called $malware\ mutation$. In order to be successful, or evasive, a malware mutation needs to be classified as benign so that $d(b') < 0.9$. Thus, misclassification is achieved.

C. Genetic Operations

On AIMED we have implemented genetic operations using the Darwinian principle of natural selection. Crossover is according to the biological model when genetic material is exchanged to form a new recombinant. In our case we select the fittest members and define a point in the sequence to swap them and create two new children. Mutations change the genetic code and are used in nature to improve diversity. Occasionally they could be helpful to achieve fittest members. We randomly exchange some perturbations for others in order to affect the sequence. Reproduction, also cloning, is making one member survive over the next generation. Although it is recommended [39] to allow a small number of the population to reproduce, it can have a significant impact on the time required for the GP as fitness score function does not need to be recalculated. In our scenario, we allow it for between 10% – 50% of the population, depending on the population size, to speed up the process without compromising diversity. Selection is implemented using fitness-proportionate style, where members are selected based on fitness compared with the entire population.

D. Fitness Function

The fitness function uses four components to determine the best members of each generation and it can be formally defined as λ in Equation 1:

$$\lambda(b') = f(b') + d(b') + \theta(b, b') + g_{b'} \tag{1}$$

where $\theta(b_i, b'_i)$ is a function that compares the new malware mutation with its original sample and compute a similarity number. The greater the number is, the better we expect the diversity between the two samples will be when searching for an adversarial example. Though, this might not always be preferred as for some cases we could want to have a close relative of another adversarial example instead of a more different one. Nevertheless, this method will give preference to find faster valid over corrupt mutations while searching thoroughly for evasions afterwards. Finally, the current number of generation is denoted with g and is added into the fitness function to give priority to newly generated members on the selection process.

Therefore, finding a potential evasive mutation $e_i \in B^*$ can be approached by maximizing a function (Eq. 2) that receives as input a malware file, $b_i \in B^*$, and maps it to an optimal sequence of δ, which satisfies $b'_i = b_i + \delta$:

$$e = \arg\max \sum_{i=0}^{\gamma} \lambda(b'), \forall b' \in B^* \tag{2}$$

where γ is the threshold of generations expected to run until the number of adversarial malware examples is found.

V. Evaluation

In the evaluation we discuss the results by comparing the efficiency of AIMED with a random perturbation injection system and an OpenAI gym model implementing a RL agent.

As input we used 6880 malware files and deployed them into the system, which is randomly picking a sample for each round and injecting a sequence of perturbations. The idea is to start creating a pool of four mutations with 10 perturbations added each. That will be the initial population. Then we evaluate the fitness score using a combination of the four previously described methods: functionality, detection, similarity, and distance from origin or generation. The sum of these components constitutes the fitness function (Eq. 1) that

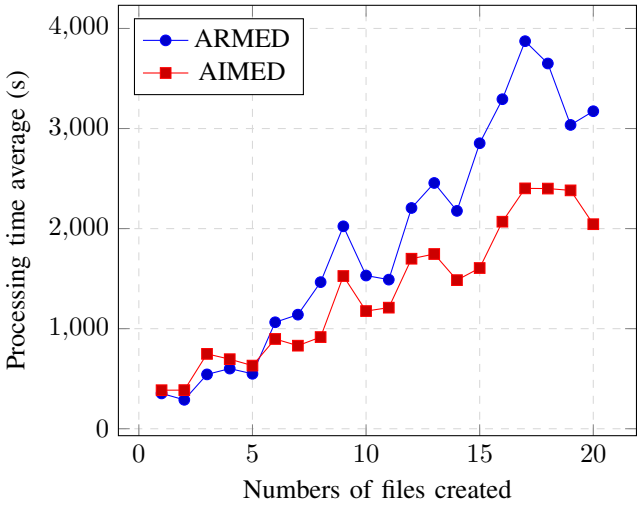

Fig. 3. ARMED vs AIMED processing times

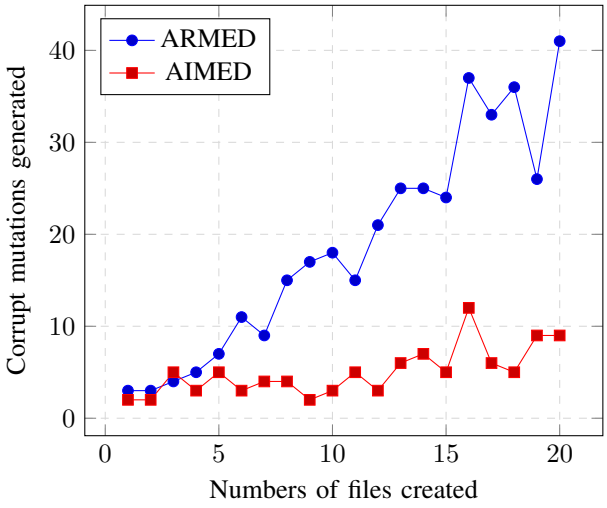

Fig. 4. ARMED vs AIMED corrupt files generated

we want to maximize as presented in section IV-D. We will be running individual original samples as input every round and expect between one and two evasions for every run. We set a limit of 10 generations, which usually takes between 1 – 4 minutes depending on how fast the algorithm converges mostly based on the classifier used. That means, if we don't find any evasion after that number of generations we declare the original sample as 'unmodifiable' and move on to the next.

A. Comparing to OpenAI Gym Model

OpenAI gym is a framework that became widely used to train RL agents mostly in the gaming environment. Anderson et al. implemented it as a malware evasion environment and adjusted the framework to be able to generate malware mutations using a RL agent [6].

They add modifications sequentially to each new sample in a round until an evasion occurs or the threshold of 10 perturbations is achieved. According to the study, with 50.000 perturbations 2085 evasions are achieved. Moreover, an evasion rate is reported of 24% for the 200 holdout group.

Given the nature of GP and that we are not adding each perturbation sequentially one after the other, we do not use a budget of perturbations as metric. Instead, we take the number of malware mutations generated based on the files sampled, ergo, in our experiments we used 1253 files as original malware b_i. After more than hundred generations, AIMED created 5551 new malware mutations. From those, 2954, more than half, were corrupted and thus dismissed. 2076 were built correctly and valid but detected by the given classifier, and finally 521 were adversarial samples, which means valid and evasive. This experiment points out how important it is to make sure evasive files are valid as 53% turned out to be corrupt. Furthermore, in 299 cases AIMED successfully delivered valid and adversarial examples, close to 24% of all the input files, which is similar to OpenAI gym's RL agent rates but with the addition of having fully functional adversarial examples and the fact that the result combines detections of four scanners. This proves AIMED's robustness to bypass detection regardless of the malware used as input or the classifier set as target.

Note that the OpenAI gym methods using the RL agent are tested against a different set of samples than AIMED as the training data has not been published. However, we also focused on Windows PE malware files and would expect similar results in case we have access to the same pool of samples.

B. Comparing to ARMED

ARMED is able to generate evasive malware mutations by injecting random binary-level perturbations. However, random perturbations are useful to find single adversarial examples but limited if the output of samples required increases as it is brute-forcing the system in order to find a suitable sequence of perturbations to bypass the classifier.

We calculate in Table I what is the ratio of corrupt files in malware mutations generation. Based on the percentages reported, we observe that generating batches of, for instance, 9 mutations are in average one of the best ratios as less than 23% of the malware mutations generated are corrupt whereas for a batch of 10 around 30%. Conversely, for both cases AIMED is generating 70% and more of valid malware mutations. Therefore, we chose 10 as the size of our batch to start generating adversarial examples.

Functionality and detection stages times were reduced by half as one round in ARMED used to take around 60 s and one generation in AIMED takes in average at most 30 s. Our experiments also showed that using pre-trained classifier models such as GBDT reduce the processing time to milliseconds and the adversarial rates are comparable to commercial classifiers, which cuts the whole generation processing time to less than 15 s per generation. A committed adversary can then leverage these adversarial examples and look for cross-evasion against

TABLE I
RATIO OF CORRUPT MALWARE MUTATIONS (%)

Mutations	ARMED	AIMED
1	3,000	2,000
2	1,500	1,000
3	1,333	1,667
4	1,250	0,750
5	1,400	1,000
6	1,833	0,500
7	1,286	0,571
8	1,875	0,500
9	1,889	0,222
10	1,800	0,300
11	1,364	0,455
12	1,750	0,250
13	1,923	0,462
14	1,786	0,500
15	1,600	0,333
16	2,313	0,750
17	1,941	0,353
18	2,000	0,278
19	1,368	0,474
20	2,050	0,450

commercial scanners as showed in Tablet II without the need to target them specifically.

C. Cross-evasions

Based on the detection rates we chose top performing malware classifiers both from the industry and academia. We performed tests using one classifier to find evasive malware mutations and then scanned the latter with other malware detectors in order to evaluate cross-evasion properties of the adversarial examples generated.

As presented in Table II, we grouped the adversarial examples found by AIMED for each of the classifiers and run them against each other. Our experiments show that cross-evasion is achieved with relatively high rates among commercial classifiers. For example, running AIMED with ESET will generate adversarial examples that bypass Kaspersky 66% of the time. Moreover, using the GBDT model, it can create adversarial malware that will be misclassified more than 80% of the time by Sophos. Conversely, more than 42% of the samples generated against Sophos will also bypass the GBDT model. This proves that AIMED can efficiently find evasions for Windows PE files that are particularly successful among different classifiers.

All scanners were tested using static detection with default configurations in the following versions: Kaspersky 19, ESET NOD32 Antivirus 11, and Sophos Endpoint Security 10.8 up-to-date at the moment when experiments were performed.

D. Limitations

Even though having valid adversarial examples against a variety of classifiers proves to be feasible within just a few minutes, the number of input files targeted as 'unmodifiable' is still high, around 76%. An adversary will still need to face a fundamental tradeoff between increasing the time spent waiting for a successful evasion automatically or finding it manually. This is certainly an interesting area to gain a better understanding about how to convert unmodifiable files into modifiable ones.

TABLE II
CROSS-EVASION RATE AMONG CLASSIFIERS

	Kaspersky	ESET	Sophos	GBDT
Kaspersky	**100%**	18,39%	49,69%	25,47%
ESET	66,02%	**100%**	69,90%	38,83%
Sophos	41,74%	42,15%	**100%**	42,71%
GBDT	38,46%	48,07%	82,69%	**100%**

While this approach focus on generating adversarial examples for static malware classifiers, bypassing dynamic classification by automatically modifying the samples' behavior still remains an open issue. A potential line of research is to explore further perturbations that preserve the semantics of malware files while covering malicious behavior by understanding each of the actions. These perturbations combined with the presented techniques can be further studied when generating adversarial examples that attempt to bypass more security layers.

VI. CONCLUSION

Here we present AIMED, which aims to generate valid adversarial examples applying GP algorithms to previously detected malicious software in order to find optimal sequences of perturbations to bypass a malware classifier.

As comparison, results from the GP algorithm are compared with those of a previous model that only generates random sequences of perturbations to be injected into detected malware. Evolving them to bypass static malware detection is possible and, depending on the parameters and classifiers used, it can take only a few minutes to be successful.

Experimental results proved that while for small number of evasive mutations the random algorithm performs successfully, the GP algorithm displays more competitive results for larger numbers. Investigations also showed that if 10 valid evasive malware files are to be generated, AIMED returns results in average at least twice as fast as its random-counterpart. Evasion rates are similar to other frameworks while making sure all adversarial examples are valid PE files. Furthermore, adversarial examples generated with AIMED using a given malware classifier can be successfully leveraged against other models with high cross-evasion rates.

Therefore, GP seems to be a relatively simple yet powerful option to create adversarial examples of malware files that are able to evolve in such a way inspired by the Darwinian process and able to bypassing malware classifiers.

Finally, some lines of future work will include a deeper understanding on how to make more input files modifiable as well as extending the machine learning approach beyond GP. Moreover, AIMED can be used to test and assess new adversarial learning defenses in malware classifiers.

ACKNOWLEDGMENT

The authors would like to thank VirusTotal for sharing samples and the Chair for Communication Systems and Network Security as well as the Research Institute CODE for their comments and improvements. Research supported, in parts, by EC H2020 Project CONCORDIA GA 830927.

REFERENCES

[1] Sophos, "ML: How to Build a Better Threat Detection Model," https://www.sophos.com/en-us/medialibrary/PDFs/technical-papers/machine-learning-how-to-build-a-better-threat-detection-model.pdf, 2017, last access Feb. 21, 2019.

[2] ESET, "Is Machine Learning Cybersecurity's Silver Bullet?" https://www.welivesecurity.com/wp-content/uploads/2017/08/NextGen_ML.pdf, 2017, last access Feb. 10, 2019.

[3] Kaspersky, "Machine Learning for Malware Detection," https://media.kaspersky.com/en/enterprise-security/Kaspersky-Lab-Whitepaper-Machine-Learning.pdf, 2017, last access February 11, 2019.

[4] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing Properties of Neural Networks," *Computing Research Repository*, February 19, 2014.

[5] W. Xu, Y. Qi, and D. Evans, "Automatically Evading Classifiers," in *Network and Distributed Systems Symposium*, ser. NDSS. Reston, VA. USA: Internet Society, February 2016, pp. 1–15.

[6] H. S. Anderson, A. Kharkar, B. Filar, D. Evans, and P. Roth, "Learning to Evade Static PE Machine Learning Malware Models via Reinforcement Learning," *Computing Research Repository*, pp. 1–9, January 2018.

[7] W. Hu and Y. Tan, "Generating Adversarial Malware Examples for Black-box Attacks based on GAN," *Computing Research Repository*, February 20, 2017.

[8] B. Biggio and F. Roli, "Wild Patterns: Ten Years After the Rise of Adversarial Machine Learning," *Pattern Recognition*, vol. 84, pp. 317–331, December 2018.

[9] R. Labaca Castro, C. Schmitt, and G. D. Rodosek, "ARMED: How Automatic Malware Modifications Can Evade Static Detection?" in *5th International Conference on Information Management (ICIM)*. New York, NY, USA: IEEE, 02 2019, pp. 1–x, Acceptance notification on February 22, 2019.

[10] M. Zheng, P. P. Lee, and J. C. Lui, "Adam: an automatic and extensible platform to stress test android anti-virus systems," in *International conference on detection of intrusions and malware, and vulnerability assessment*. Springer, 2012, pp. 82–101.

[11] V. Rastogi, Y. Chen, and X. Jiang, "Droidchameleon: Evaluating Android Anti-Malware Against Transformation Attacks," in *8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, ser. ASIACCS. New York, NY, USA: ACM, May 2013, pp. 329–334.

[12] E. Aydogan and S. Sen, "Automatic Generation of Mobile Malwares Using Genetic Programming," in *European Conference on the Applications of Evolutionary Computation*, ser. EvoApplications. Heidelberg, Germany: Springer, March 2015, pp. 745–756.

[13] H. G. Kayacik, M. Heywood, and N. Zincir-Heywood, "On Evolving Buffer Overflow Attacks Using Genetic Programming," in *8th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO. New York, NY, USA: ACM, July 2006, pp. 1667–1674.

[14] H. G. Kayacik, A. N. Zincir-Heywood, M. I. Heywood, and S. Burschka, "Generating Mimicry Attacks Using genetic Programming: A Benchmarking Study," in *IEEE Symposium on Computational Intelligence in Cyber Security*, ser. CISDA. New York, NY, USA: IEEE, July 2009, pp. 136–143.

[15] H. G. Kayacık, A. N. Zincir-Heywood, and M. I. Heywood, "Evolutionary Computation as an Artificial Attacker: Generating Evasion Attacks for Detector Vulnerability Testing," *Evolutionary Intelligence*, vol. 4, no. 4, pp. 243–266, October 2011.

[16] S. Noreen, S. Murtaza, M. Z. Shafiq, and M. Farooq, "Evolvable Malware," in *11th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO. New York, NY, USA: ACM, July 2009, pp. 1569–1576.

[17] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar, "Can Machine Learning be Secure?" in *ACM Symposium on Information, Computer and Communications Security*. New York, NY, USA: ACM, March 2006, pp. 16–25.

[18] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," *arXiv preprint arXiv:1412.6572*, December 2014.

[19] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The Limitations of Deep Learning in Adversarial Settings," in *IEEE European Symposium on Security and Privacy*, ser. EuroS&P. New York, NY, USA: IEEE, March 2016, pp. 372–387.

[20] 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*, June 2016.

[21] B. Kolosnjaji, A. Demontis, B. Biggio, D. Maiorca, G. Giacinto, C. Eckert, and F. Roli, "Adversarial Malware Binaries: Evading Deep Learning for Malware Detection in Executables," in *26th European Signal Processing Conference*, ser. EUSIPCO. New York, NY, USA: IEEE, September 2018, pp. 533–537.

[22] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Networks," pp. 2672–2680, June 2014.

[23] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai Gym," *arXiv preprint arXiv:1606.01540*, pp. 1–4, June 2016.

[24] Koza, J. R., "Survey of Genetic Algorithms and Genetic Programming," in *Wescon Conference Record*, ser. WESCON. New York, NY, USA: IEEE, November 1995, pp. 589–594.

[25] J. H. Holland, *Adaptation in Natural and Artificial Systems*. MIT Press, April 1992.

[26] J. R. Koza, "Evolving a Computer Program to Generate Random Numbers Using the Genetic Programming Paradigm," in *International Computer Games Association (ICGA)*, ser. September. Citeseer, 1991, pp. 37–44.

[27] Koza, J. R., "Using Biology to Solve a Problem in Automated Machine Learning," in *Models of Action: Mechanisms for Adaptive Behavior*. Hillsdale, NJ, USA: Lawrence Erlbaum Associates, 1998, pp. 157–199.

[28] S. Forrest, "Genetic Algorithms: Principles of Natural Selection Applied to Computation," *Science*, vol. 261, no. 5123, pp. 872–878, 1993.

[29] A. Calleja, A. Martín, H. D. Menéndez, J. Tapiador, and D. Clark, "Picking on the Family: Disrupting Android Malware Triage by Forcing Misclassification," *Expert Systems with Applications*, vol. 95, pp. 113–126, April 2018.

[30] M. Christodorescu and S. Jha, "Testing Malware Detectors," *ACM SIGSOFT Software Engineering Notes*, vol. 29, no. 4, pp. 34–44, July 2004.

[31] J. A. Morales, P. J. Clarke, Y. Deng, and B. G. Kibria, "Testing and Evaluating Virus Detectors for Handheld Devices," *Journal in Computer Virology*, vol. 2, no. 2, pp. 135–147, September 2006.

[32] M. Pietrek, "Peering Inside the PE: A Tour of the Win32 Portable Executable File Format," https://msdn.microsoft.com/en-us/library/ms809762.aspx, March 1994, last access Aug. 10, 2018.

[33] M. Pietrek, "PE Format," https://docs.microsoft.com/en-us/windows/desktop/Debug/pe-format, 2018, last access Aug. 28, 2018.

[34] J. R. Larus and E. Schnarr, "EEL: Machine-independent Executable Editing," in *SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI, vol. 30, no. 6. New York, NY, USA: ACM, June 1995, pp. 291–300.

[35] A. Srivastava and D. W. Wall, "A Practical System for Intermodule Code Optimization at Link-Time," in *Journal of Programming Languages*, vol. 1, no. 1. Noida, India: STM Publication, December 1992, pp. 1–18.

[36] Quarkslab, "Library to Instrument Executable Formats," https://lief.quarkslab.com, 2017, last access Feb. 23, 2019.

[37] C. Guarnieri, A. Tanasi, J. Bremer, and M. Schloesser, "Cuckoo Sandbox - Automated Malware Analysis," https://cuckoosandbox.org/, 2012, last access March 13, 2019).

[38] F. Kreuk, A. Barak, S. Aviv-Reuven, M. Baruch, B. Pinkas, and J. Keshet, "Deceiving End-to-End Deep Learning Malware Detectors Using Adversarial Examples," *arXiv preprint arXiv:1802.04528*, January 2019.

[39] Koza, J. R., "Genetic Crogramming: on the Programming of Computers by Means of Natural Selection," in *Statistics and Computing*, vol. 4, no. 2. Heidelberg, Germany: Springer, June 1992, pp. 87–112.