Coevolution of Mobile Malware and Anti-Malware

Sevil Sen*, Emre Aydogan, Ahmet I. Aysan

***Abstract*—Mobile malware is one of today's greatest threats in computer security. Furthermore, new mobile malware is emerging daily that introduce new security risks. However, whilst existing security solutions generally protect mobile devices against known risks, they are vulnerable to as yet unknown risks. How anti-malware software reacts to new, unknown malicious software is generally difficult to predict. Therefore, anti-malware software is in continuous development in order to be able to detect new malware or new variants of existing malware. Similarly, as long as anti-malware software develops, malware writers also develop their malicious code by using various evasion strategies such as obfuscation and encryption. This is the lifecycle of malicious and anti-malware software. In this study, the use of evolutionary computation techniques are investigated, both for developing new variants of mobile malware which successfully evades anti-malware systems based on static analysis and for developing better security solutions against them automatically. Coevolutionary arms race mechanism has always been considered a potential candidate for developing a more robust system against new attacks and for system testing. To the best of the authors' knowledge, this study is the first application of coevolutionary computation to address this problem.**

***Index Terms*—mobile malware, automatic malware generation, static analysis, evasion, obfuscation, malware detection, evolutionary computation, coevolution, Android**

I. Introduction

Mobile devices have become an integral part of daily life. They provide many useful functions such as the ability to read and write e-mails, surf the Internet, indicate nearby facilities, video conferencing, and voice recognition, to name but a few. However, the popularity and adoption of mobile devices also attract malware writers to develop mobile malware in order to harm these devices. According to a Kaspersky security report [1], 884,774 new malware was introduced in 2015, three times more compared to 2014. Symantec also reported that one zero-day attack per week on average was discovered in 2015 [2]. Moreover, they emphasized the large increase in the volume of Android variants (40%) besides new Android malware families added in 2015 (6%) [2]. Hence, in order to protect mobile devices from such threats, researchers and security companies work to develop effective and efficient anti-malware systems.

There are some techniques available for malware analysis and detection with varying strengths and weaknesses. Two common types of malware detection techniques, according to how the code is analyzed, are static and dynamic analyses. They can also be combined to create hybrid solutions. Since dynamic analysis might not be affordable on some mobile devices due to their significant limitations in terms of power consumption, most of the proposed approaches in the literature rely on static analysis. However, these tools are known to be vulnerable to some obfuscation techniques and new attacks. Therefore, in recent years, attackers have focused on exploiting the vulnerability of static analysis tools. While the number of new Android mobile malware families has inclined to decrease over the past two years, there has been significant growth in the number of new Android mobile malware variants [2]. How anti-malware is effective against known attacks, variants of known attacks, and unknown attacks requires further investigation, and forms the primary goal of the current study. In order to be able to assess security solutions proposed for mobile devices, new variants of existing attacks were automatically generated. The generated new attacks highlighted weaknesses of the market available static analysis tools, and the need for new detection techniques suited to mobile devices. The secondary goal of the study is to explore developing an anti-malware software automatically, which is robust to both some known attacks and their variants. In order to achieve these goals, coevolutionary computation techniques were applied to the problem. The researchers believe that better anti-malware software can be developed when new malicious software is taken into account, hence the use of coevolutionary arms race mechanism is explored in this study. Experiments in this current study are grouped into three main sets.

- mobile malware evolution
- mobile anti-malware evolution
- mobile malware/anti-malware coevolution

The researchers created new malware and variants of known malware by using genetic programming (GP) in order to mimic mobile malware evolution, and thereby evaluate the performance of existing static analysis tools. The aim was to generate new malware automatically that could be used in order to also strengthen existing static analysis tools automatically. As most existing static tools update their signature databases when they encounter new/unknown malware, automating this process will ensure that detection systems are more robust against attacks. While this approach only automates the generation of new/unknown attacks, an evolution-based detection system is proposed for mobile anti-malware. The framework is extended by improving existing solutions automatically in mobile malware/anti-malware coevolution. Malware writers mainly aim to achieve their goals (*e.g.* damaging mobile devices and/or achieving financial gain) without being detected by using effective evasion strategies such as obfuscation and encryption. The increasing number of new malware variants has led to security companies improving their solutions. As new anti-malware solutions are introduced, malware writers also try to evade them, resulting in a cyclical, endless process. These cycles exactly define the coevolution arms race mechanism between malware and anti-malware, with each competing to outmaneuver the other. This current study investigates the use

S. Sen*, E. Aydogan, A. I. Aysan are with the WISE lab, Hacettepe University, Ankara, TURKEY (e-mail: {ssen, emreaydogan}@cs.hacettepe.edu.tr, aysan@hacettepe.edu.tr) * Corresponding author

of coevolutionary computation techniques in order to generate more evasive malware and more robust anti-malware against new variants of existing malware.

GP has already been applied in order to evolve new attacks and new malware in the literature [3]–[7]. However, most of these approaches are not fully automated and only proposed for a specific attack type. A security expert is generally needed to analyze the code and extract parameters that changes in different variants of malware code, so a representation of the problem can be constructed for GP. The aim of this study was to create a fully automated system by employing genetic operators on smali codes of existing malware by using GP. The results show that GP could generate effective attacks able to evade existing anti-malware systems which are considered to be among the most successful mobile security solutions [8]. Furthermore, GP shows better performance than sole or dual application obfuscation techniques as proposed in the literature [9][10].

The current study also developed a detection system based on certain static features of Android applications such as API calls and permissions. As far as the researchers are aware, there is no such GP-based mobile malware detection system in the literature. The results produced a high rate of detection on known attacks with a low false positive rate. Furthermore, coevolutionary computation techniques were applied in order to generate more robust detection systems, and as a result, more evasive attacks and robust security solutions were generated. This approach was able to produce very evasive malware comparable with the obfuscation results of Zelix KlassMaster [11], a well-known Java bytecode obfuscator. The coevolved AVs were evaluated against three datasets : PRAGuard obfuscated dataset [12], Drebin dataset [13], and Zelix dataset created by employing Zelix KlassMaster [11]. Results showed that the coevolved solution especially outperformed commercial AVs to a considerable degree for the PRAGuard and Zelix datasets.

This current study makes the following contributions :

- A fully automated model is proposed which generates evasive mobile malware from existing ones.
- A GP-based malware detection system is proposed based on static features of Android applications, which proves very effective against known attacks.
- The first application of coevolutionary computation techniques to system security is proposed in order to both generate more evasive malware and robust anti-malware software.

The remainder of the paper is organized as follows: Section II summarizes the related approaches in the literature. Section III describes the proposed method for generating new malware. Section IV details the proposed model for detecting known mobile malware. Section V contains the main framework which combines malware and anti-malware evolution under coevolution. Section VI reports on the performance of the model, with results discussed in Section VII. Section VIII is devoted to concluding remarks and future works.

II. Related Work

This section is divided into two in order to support better understanding of the current literature and this study's contribution.

A. Malware Evolution

Security companies and researchers working on security solutions for mobile devices use two main techniques to analyze apps: dynamic and static analysis. Since dynamic analysis is unsuited to resource-constrained mobile devices, static analysis techniques are mostly used to analyze mobile apps. Since many security companies already brought out their solutions for protecting mobile devices, how effective they are against known and unknown malware requires investigation.

There are two main studies in the literature that evaluate commercial anti-malware products against obfuscation techniques on the Android platform. In [9], Zheng et al. proposed a system called ADAM that evaluates the effectiveness of anti-malware systems against malware samples generated by employing some obfuscation techniques automatically, while preserving the original malicious function. Rastogi et al. [10] developed a system called DroidChameleon that evaluates Android anti-malware products against obfuscation attacks that are extended forms of the attacks in [9]; automatically mutating Android applications by using polymorphic and metamorphic techniques.

Christodorescu and Jha [14] proposed a technique based on a transformation of source code for creating test samples for desktop malware detection systems. Their technique aimed to evaluate the resilience of anti-malware systems to various obfuscation techniques. Morales et al. [15] evaluated and tested how anti-malware systems protect handheld devices against known malware. Their results indicate that all four anti-malware systems tested produce high false negative rates due to the simple signature detection algorithms employed by the products. Moser et al. [16] proposed a malware detector relying on semantic signatures and employed a model checking for detection. They showed that static analysis alone is not efficient in detecting malware, and that it should be complemented with dynamic analysis techniques.

You and Yim [17] analyzed the obfuscation techniques commonly used by malware writers, with the aim of understanding how malware writers use these techniques. Christodorescu et al. [18] propose a malware normalizer that reverts obfuscated malware to their original by undoing the obfuscations. Their goal being to increase the detection rate of malware detectors.

The earliest works based on malware evolution by using evolutionary computation are proposed by Kayacik et al. [3], who used GP to evolve buffer overflow attacks in order to obfuscate the true intent of the attacker. In [4], the authors also used GP to generate mimicry attacks with the objective of finding potential vulnerabilities before attackers could exploit them, with the goal to generate attacks that seem benign, hence they could evade detection. The authors extended this study in [5] by increasing the number of detectors and adding a delay parameter in order to build evasion attacks. In [6], a comparison of two approaches was employed, white-box and black-box. While the former employs the internal knowledge of the detectors to generate evasion attacks, the latter uses only the output of the detectors.

Wu et al. [19] proposed a computer malware evolution model based on Immune Genetic Algorithm, and Noreen et al. [7] created new variants of Beagle malware by applying genetic operators. They also applied this approach to Silvio malware threatening the ARM-Linux based smartphones and generated new variants of Silvio malware by using Markov models [20]. In [21], the authors aimed to generate new malware by changing the semantic of malware by extracting the abstract representation of a malware, then using GP to evolve a new malware from this representation.

The closest work to the approach of the current study in terms of generating new mobile malware using evolutionary algorithm is Mystique [22], which was published after the current study's previous work [8]. In [22], attack and evasive features of Android malware were extracted, and a meta-model for Android malware created by using these features. Then, a multi-objective evolutionary algorithm was employed on these features in order to generate more aggressive, evasive and undetectable malware variants. The approach was effective; however, analyzing malware and determining attack/evasive features, detectability and latency values of such features require considerable manual input and experience. In [23], the authors extended their work by using more attack types and generating malware using dynamic code-loading, which can be used to hide malicious activities of an application from the analysis carried out in devices and market stores [24].

The studies in the literature proposed for generating new malware by using evolutionary computation are mostly not fully automated, and focus on specific attacks like buffer overflow attacks or malware like Beagle and Silvio. Only a few approaches were found to evaluate mobile anti-malware solutions against obfuscated attacks [9][10]. The target of this current study is also to test existing anti-malware solutions, and to the best of the authors' knowledge, this is the first work that generates evolved mobile malware using GP for Android platforms [8]. GP automatically employs the best evasion strategies by reducing the size of the search space. One of the main characteristics of the proposed system is its being fully automated. The proposed approach in the current authors previous study [8] is extended with additional obfuscation techniques, and experimental results; combining it with anti-malware evolution to become coevolutionary.

B. Anti-Malware Evolution

In order to cope with the rapid increase in the number of mobile malware, many companies have introduced their own mobile security solutions, which are mainly based on static analysis. Many studies in the literature also propose mobile malware detection in academia as well. However, this current study only outlines the important detection techniques based on static analysis due to the relevance for this study.

Stowayay et al. [25] detects overprivileged apps by analyzing API calls of applications. Kirin [26] detects mobile malware by employing permission-based static analysis. Drebin [13] also employs permissions, API calls, and network addresses in order to differentiate malicious software from benign software by employing machine-learning techniques. Similarly, DroidAPIMiner [27] employs machine-learning techniques and uses API calls of applications as distinguishing features. RiskRanker [28] uses two-level static analysis. High- and medium-risk applications are determined in the first order analysis, and applications employing obfuscating, encryption or dynamic class loading techniques are extracted among these risky applications in the second-order analysis. Some researchers focused on system-level events for malware detection. Schmidt et al. [29] proposed a static analysis detection technique based on library and system function calls.

Moser et al. [16] indicated that static analysis alone is not efficient in detecting malware and should be complemented with dynamic analysis. They used semantic signatures and employed a model checking for detection. A recent study for mobile malware detection called MARVIN [30] employs both static and dynamic analysis. It is shown that they achieve better results than Drebin [13] and DroidAPIMiner [27]. The most distinguishable features are also analyzed and noted that static features show a high level of accuracy.

Recently, Martin et al. proposed a framework called MocDroid [31] that extracts import terms from Java codes obtained by decompiling apk files since they cannot be obfuscated. These import terms are then clustered by three different algorithms and the obtained clusters used as inputs to a multi-objective genetic algorithm in order to create a classifier. The performance of the classifier is then compared with commercial AVs. The authors showed that MocDroid improves results by more than 10%.

As shown in the literature, different machine-learning techniques are employed for malware detection. A recent proposal [32] for detecting unknown malicious code used genetic programming, plus comparison of GP with other well-known machine-learning methods such as Support Vector Machines, Bayesian Networks, Decision Trees, and Artificial Neural Networks. Their results showed that GP outperforms other methods on both balanced and imbalanced datasets. Such a conclusion showed promise for employing GP to mobile malware detection; hence it has been applied with different features in this current study.

The battle between computer malware and anti-malware is still as far from over as ever [33]. Although coevolution has a potential for application in the area of systems security [34], it has not yet been applied and is therefore a primary aim and intended contribution of this study. To the best of the researchers' knowledge, there has only been one work that proposes a network anomaly intrusion detection using coevolutionary computation [35]. In [35], a network behavior model based on the self-nonself space paradigm is constructed. The authors then built hyper-rectangular detectors covering nonself-space using a niching genetic algorithm. Their results showed that their proposed method was very effective against all five types of attacks in the dataset.

In the following sections, to aid understanding of the proposed approach based on coevolutionary computation, each component (malware and anti-malware evolution) will be introduced. Then, how these components are combined for coevolutionary computation will be presented.

III. Malware Evolution

This section introduces the method proposed by this current study, which employs GP in order to generate new, unseen mobile malware from that occurring in the wild. Figure 1 illustrates the conceptual schema of the proposed approach called *malware evolution*.

Primarily, the *apk* files (executable application files on Android) are converted to their source codes. Although Java source files can be obtained from *apk* files and modifications can be applied to them, this study uses smali files since they allow for the conversion of modified source files back to apk files. Smali can be seen as the assembly language for dalvix, Android's Java VM implementation. It should be noted that conversion to *apk* files is essential in order to evaluate generated malware on mobile devices/emulators.

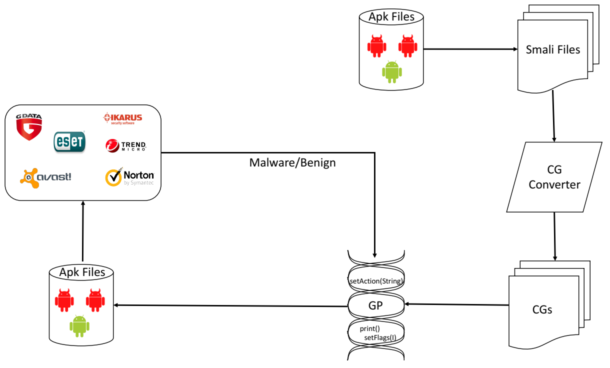

Fig. 1. The Architecture of Malware Evolution.

Smali files are similar to Java class files and consist of methods. ApkTool [36] is an open source tool used to obtain smali files from *apk* files. Call graphs (CGs) of *smali* files files were extracted by using a CG converter developed by the authors. CG is a control flow graph representing all paths that can be passed through a program during its execution. Each CG corresponds to a tree, each method is represented by nodes, and the edges indicate the flow between each method. Each application is represented by an individual in GP and each individual has different sizes of trees (CGs) as each *apk* file has a variable number of functions. In GP, crossover or mutation operators are applied to each CG. It should be noted that applications considered here do not consist of recursive functions, which would limit the study to a smaller set of malicious applications.

Genetic programming [37] is a population-based search algorithm inspired by natural evolution . It starts with generating a population of individuals (usually at random) which are candidate solutions for the target problem. Then, each individual is evaluated and assigned with a fitness value that indicates how well this candidate solves or comes close to solving the problem at hand. Until a termination criterion is satisfied, new populations are generated iteratively by using selection, crossover, and mutation operators, as in natural evolution. These genetic operators are used to provide better solutions in the new population. Each individual represents an Android application in GP. The initial population consists of malware in the wild. The main aim is to generate more evasive malware at each generation by applying genetic operators on them. These operators are applied to CGs, then *smali* and *apk* files are recreated from them. Newly generated apk files are signed by using *keytool* and *jarsigner* tools, then sent to emulators in order to calculate their fitness values, which shows the degree to how evasive newly generated malware is.

In order to evaluate newly generated malware, six emulators with different anti-malware systems installed and a detection system based on machine-learning (ML) are used in this study. While anti-malware systems work on static attributes of applications, the ML-based detection system extracts dynamic features of applications by running them on an emulator for a period of ten minutes. A newly generated malware is run on each emulator separately and tagged as either malicious or benign by anti-malware systems. The aim of this process is to generate malware that evade anti-malware systems by decreasing the fitness value in each generation. The fitness value is calculated as the ratio of the number of systems detecting the generated application as malware to the total number of systems. Since it is hard to obtain the ideal fitness value by generating malware that evades all anti-malware systems, the GP algorithm terminates when the predefined generation size is reached.

In the experiments, the ECJ v.20 [38] tool is used. The GP parameters used are shown in Table I. Parameters not shown here are the default parameters of ECJ. The tree depth (17) is also the default parameter of ECJ. GP generally has a code bloating problem due to its long tree depth. Since this problem is not desired in many GP applications, the tree depth parameter is often decreased. However, bloating has a positive effect in the method used in this study by causing more complex, hence more evasive malware to be generated. Therefore, the default value of the tree depth parameter is employed in this study.

A. Genetic Operators

1) Crossover: This operation exchanges sub-trees of individuals. In order to obtain executable programs, sub-trees must be compatible with each other. Only methods with the same declarations having the same return type and an equal number of parameters with the same types may be exchanged. This allows for the creation of new malware from existing malware. However, since the high crossover rate increases the number of non-executable individuals, the crossover operation is assigned a low rate (0.1) in the experiments. Crossover provides the change of malware signatures by code exchange, and helps generate more evasive malware.

TABLE I
GP Parameters for Malware Evolution

Parameter	**Value**
Goal	Generating evasive malware
Population Size	15
Generation Size	200
Crossover Rate	0.1
Mutation Rate	0.9
Tournament Size	2
Tree Size	17

When the crossover is applied to applications, it is required to see whether or not the newly generated application runs properly and exposes malicious behaviors. In order to be able to check this automatically, a ML-based detection system based on dynamic analysis is employed [39]. The tool runs each application for a period of ten minutes, extracting dynamic features collected from the outputs of DroidBox [40] such as messaging information, services used etc. in order to distinguish malicious applications from the benign using machine-learning. Monkey [41] is used as an UI exerciser as it generates random events. Although Monkey [41] is one of the tools to achieve the best code coverage [42], triggering malware is one of the main limitations of any dynamic analysis tool. Although the developed detection system has a high rate of accuracy (97.91%), it might not detect malicious applications not triggered during the ten minute period as would any other dynamic analysis-based detection system. If the application is neither runnable or malicious, the worst fitness value (1.0) is assigned in order to eliminate it from the next generation.

2) Mutation: These operator changes selected individuals in order to introduce diversity to the population. Six obfuscation techniques are employed on sub-trees of randomly chosen individuals. These techniques aim to generate different variants of malware while preserving the original malicious function. The mutation rate was set to a high value (0.9). Source codes are shuffled as much as possible in order to make it less recognizable by detection systems. The obfuscation techniques used in this study are as follows; although further techniques could be added in the future.

- Rename Local Identifier
- Junk Code Insertion
- Data Encryption
- Two-fold Code Reordering
- Three-fold Code Reordering
- Register Realignment

TABLE II
ANTI-MALWARE SYSTEMS AND THEIR PROTECTION SCORES.

Manufacturer	Product	Version	Protection Score 2014	Protection Score 2017
Avast	Avast Mobile Security	3.0.7550	5.5	6.0
Ikarus	Ikarus mobile.security	1.7.20	4.5	5.0
Norton	Norton Mobile Security	3.8.6.1533	5.5	6.0
TrendMicro	Trend Micro Mobile Security	5.0.0.1255	6.0	5.5
Eset	Eset Mobile Security	3.0.882.0-16	6.0	5.5
Gdata	G Data Internet Security	25.0.0	6.0	5.5

B. Fitness Function

Since the fitness function defines how individuals' solve problems or come close to a solution, defining a well-representative fitness function is very important in any GP application. First an evaluation takes place to ascertain whether or not the evolved malware is runnable and showing malicious behavior to the ML-based detection system. If not, the worst fitness value is assigned to the individual. Then, the fitness function uses the output of six anti-malware systems, selected according to their protection score given in AV-TEST [43]. The second criteria in choosing the anti-malware systems is being able to use the anti-malware output automatically to run on an emulator. Hence, the solutions used are those which produce log files. Table II lists the anti-malware systems employed in this study's experiments. Current protection scores of the related products are shown in Table II in addition to the protection scores of the version of each AV system used in the training. Each anti-malware is executed on different emulators (GenyMotion [44], a powerful Android emulation platform) to simulate the execution of evolved malware and the anti-malware' response against them. Each individual is run for a period of one minute on emulators in order to generate results. This time was determined experimentally. Anti-malware systems generally return their results in less than one minute, although analysis of larger applications could take longer. If an output cannot be obtained from an anti-malware system for an individual, then the specific anti-malware is not taken into account in the fitness function for that individual of that particular generation. The fitness value, which is aimed to be minimized, is defined as follows :

$$\text{Fitness Value} = \frac{\text{\# of anti-malware systems detecting the malware}}{\text{total \# of anti-malware systems}} \tag{1}$$

The fitness value is between 0 and 1.0, with 0 being the best fitness value being aimed at, and equates to evolved malware having deceived all anti-malware solutions. When the fitness value is 1.0, it means that either the malware is detected by all solutions, or it is not runnable or malicious.

C. Dataset

The malware dataset used in this study, MalGenome, is the first mobile malware dataset used in the research community, having been generated by the Android Malware Genome project [45] in 2011. Most of the literature used this dataset to compare their results to other works. It consists of 1,260 Android malware from 49 different malware families collected between August 2010 and October 2011. Five of the malware families are found only within the official Android market and 35 are found in the alternative Android markets. The remaining nine malware families are found in both the official and alternative Android markets. It is stated that 1,083 malware is in the form of packed malware in the wild. This study used the same malware families from MalGenome as employed in the current authors' previous work [8]; extending it with additional families obtained through their CGs. The malware families used in this study are AnserverBot, Asroot, BaseBridge, DroidKungFu1, DroidKungFu2, FakeNetFlix, GPSSMSSpy, HippoSMS, and NickSpy. Their CGs were extracted and used as input to the GP algorithm as the first population. Each was detected by at least some of the anti-malware systems given in Table II.

IV. ANTI-MALWARE EVOLUTION

This section details the approach developed in this study for evolving more robust malware detection systems. The simplified schema of the approach is illustrated in Figure 2. Firstly, reverse-engineered techniques were used for applications collected from the MalGenome Project [45] and Google Play [46]. They were then analyzed and the distinguishing features of malicious applications were noted. Then, a malware detection system based on these features was evolved using genetic programming (GP).

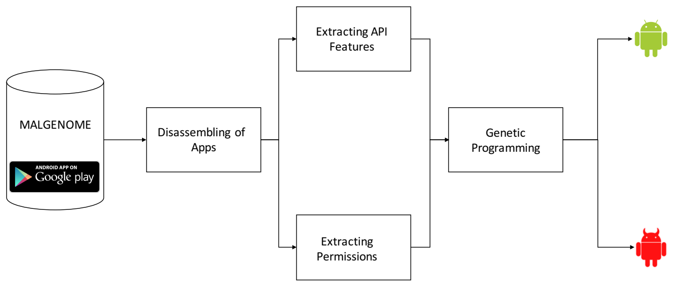

Fig. 2. The Conceptual Schema for Anti-Malware Evolution

A. Features

First, API calls and permissions for each application were extracted, as these are the most used features in static analysis for detecting mobile malware [47]. The difference between the number of malicious and benign applications of each API call was determined and sorted in descending order, and the top 100 API calls were selected for use in training. The most discriminative 40 permissions were also selected in a similar way. In addition to these 140 features, six static attributes of applications were also used in the training. A recent study showed that structural attributes such as the following, as employed in this study, could detect new malware better than solely applying API calls-based features [48]. These attributes were also observed in the evolved anti-malware.

1) Number of API calls related to DexClassLoader used to load codes at runtime.
2) Number of API calls related to Crypto used to encrypt the code.
3) Number of classes in the application.
4) Number of *goto* statements used in the applications.
5) Number of methods in the application.
6) Number of permissions in the application.

An individual in GP, which represents an anti-malware solution, is a tree consisting of all 146 features defined above as terminal nodes and some operators as non-terminal nodes. Mathematical operators such as addition, multiplication and subtraction, together with the logical operators such as or, xor were employed to form a GP tree, as illustrated in Figure 3. Each individual produces an *if statement* in determining the maliciousness of an application being analyzed. An example *if statement* is given below for the GP tree given in Figure 3. In crossover operation, new offspring *if statement(s)* are created for the new population by exchanging randomly chosen parts of two selected parent GP trees. For example, the rightmost subtree F71-&&-F34 could be exchanged with another subtree selected in another individual. In mutation operator, one new offspring *if statement* is generated for the new population by randomly mutating a randomly chosen part of one selected GP tree. For example, the node && could be mutated to the || operator. Please see the tutorial in [49] for further information on genetic programming.

$$if(\frac{(F45*F5)-F74}{F2}*F25) \leq (\frac{\frac{F59}{F43}-(F57||F142)}{F1}*(F71\&\&F34)) \quad (2)$$

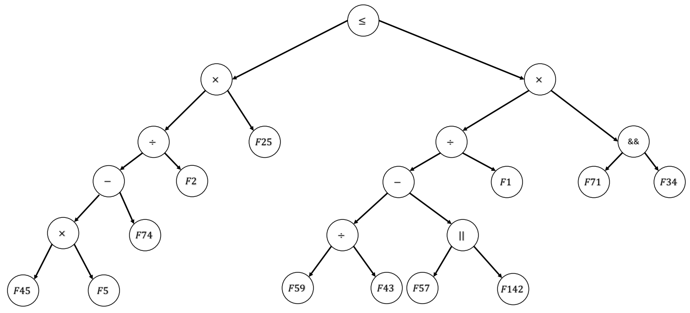

Fig. 3. The GP Tree of the Equation 2

B. Fitness Function

Fitness value is calculated based on true positive and false positive rates, as defined in Equation 3. The true positive rate is the ratio of applications labeled malicious to all malicious applications; whereas, the false positive rate is the ratio of applications labeled malicious to all benign applications. Since the evolved malware was observed in preliminary results to mostly have the perfect detection rate (100%) with a high false positive rate, the weight of the false positive rate was increased for the fitness function. Having a low false positive rate is important as much as having a high detection rate, hence the constant (k=4) in the fitness function is used for decreasing the false positive rate. The parameters used in GP are given in Table III (note that the first population is generated randomly).

$$Fitness\ Value = True\ Positive\ Rate\ -\ k\ *\ False\ Positive\ Rate \quad (3)$$

TABLE III
GP PARAMETERS FOR ANTI-MALWARE EVOLUTION

Parameter	**Value**
Goal	Developing more resistance and robust detection method against malware
Function Set	$+, *, -, >, <, \geq, \leq, \&\&, \|\|$
Terminal Set	top 100 API calls, top 40 permissions, 6 static features
Population Size	15
Generation Size	200
Crossover Rate	0.1
Mutation Rate	0.9
Tournament Size	7
Tree Size	17

C. Dataset

This study's experiments used the MalGenome dataset created by Zhou et al. [45] in which malware is classified by their common features. However, only the following 11 malware families were used in the training: AnserverBot, BaseBridge, DroidDreamLight, DroidKungFu1, DroidKungFu2, DroidKungFu3, DroidKungFu4, Geinimi, GoldDream, Kmin, and Pjapps. The main reason for using these families was to ensure a sufficient number of malware in each family for both training and testing. For benign applications, the most popular applications were downloaded from GooglePlay. Normally, Google Play does not allow the download of applications, only allowing installation on a device. Therefore, a Java program using Android Market API was developed to download applications from GooglePlay automatically. The critical feature of the applications is that they have been downloaded at least five million times, and it is therefore assumed that these applications are benign. A total of 500 applications (250 malicious, 250 benign) were employed in the training.

V. Mobile malware/anti-malware Coevolution

Coevolution is evolving better individuals among multiple species affecting each other. When one species evolves, the relationship of this species with other species also changes. Species affect each other in order to evolve better through each generation. When this philosophy is applied to computer science, coevolution is used against problems that are aiming to improve multiple systems simultaneously. These problems can be cooperative or competitive; however, most problems in the security field are competitive problems known as the *arms race*. There are limited resources in competitive problems and species compete with each other in order to use more resources. In a competitive problem, while the fitness value of one species increases, the fitness value of its rivals decreases, or vice versa. In the experiment for this study, malicious software competes with anti-malware. While malware tries to survive against anti-malware software through evasion, the anti-malware software aims to detect both known and new kinds or variants of malware.

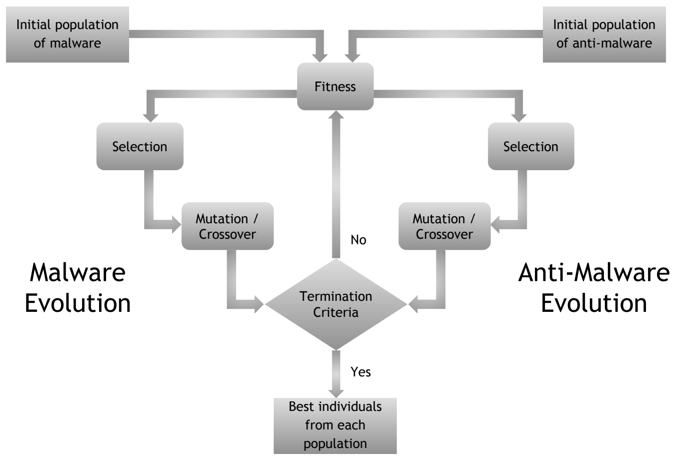

Fig. 4. The Architecture of Malware/Anti-Malware CoEvolution

The conceptual scheme of mobile malware/anti-malware coevolution is illustrated in Figure 4. The coevolution framework is based on malware and anti-malware evolutions as previously described in Sections III and IV. Two subpopulations were used in this model; with the first consisting of malware and the other of anti-malware. The representations of malware and anti-malware are used as in the malware and anti-malware evolutions, respectively. There is a variance on how fitness values are calculated. Two fitness functions were used; one for malware evolution and the other for anti-malware evolution. However, in this experiment the fitness function is not evaluated on a static dataset. The population of each system is used as an input for evaluating the fitness value of the other system. Hence, coevolution is carried out on dynamic datasets in each generation.

The coevolutionary computation algorithm is executed until the best individual is obtained or a defined number of generations is reached. Since the former is difficult to obtain in a timely manner, the termination criterion, which is mainly employed in GP applications, was employed in this study. As in the previous experiments, a generation size of 200 was applied.

Fitness function of malware evolution: This function evaluates the performance of evolved malware. Besides the outputs of commercial anti-malware systems along with the ML-based detection system as previously described, the output of anti-malware evolution at each generation is also used as input to the malware evolution. Hence, more evasive malware could be evolved against more powerful anti-malware systems. However, there is no guarantee that anti-malware evolved in the early generations can detect malware successfully; hence, this could lead to a negative effect on the malware population. Therefore, a threshold (75%) was applied for the fitness value of the individuals in the anti-malware subpopulation. If the individual in the anti-malware subpopulation detects malware effectively, then it is included in the fitness function of malware evolution.

Fitness function of anti-malware evolution: The fitness value of individuals in the anti-malware subpopulation is evaluated by using the same malware and benign dataset given in Section IV. Additionally, malware from the malware subpopulation is also included in order to evolve more robust anti-malware systems against newly unseen malware. It should be noted that if the crossover operator is applied on malware evolution, only executable, malicious malware assured by the ML-based detection are used in the fitness function.

VI. Results

A. Performance of Coevolved Malware

This section discusses the results of the coevolution approach. First, outputs were analyzed from the best (n=64) individuals from coevolved malware after 150 runs that showed improvement in their fitness value. Only the mutation operator was applied in the malware evolution in order to be able to run the system as many times as possible. When crossover is applied, the evolved applications should be evaluated on dynamic analysis, which increases the simulation time considerably. For further information on the effect of each operator on malware evolution, see the authors' previous study [8].

Table IV shows the detection rate of commercial anti-malware on coevolved malware. Since some aspects of evolutionary computation is stochastic, the algorithm is generally run n number of times, and the best individual of these runs returned as the output. Therefore, the best coevolved malware in each malware family is taken into account in the results.

It was seen that many of the coevolved malware could successfully evade commercial anti-malware solutions. The IKARUS anti-malware solution was the least effective against unknown malware, besides its high detection performance against known malware. The most robust anti-malware solution against all types of malware was GData, despite its low detection rate; even identifying evolving or obfuscating malware. Commercial anti-malware solutions were also evaluated against obfuscated malware by using Zelix KlassMaster [11] for comparison. The anti-malware solutions were seen to be very ineffective against obfuscated malware by Zelix KlassMaster [11]. While this tool obfuscates two methods in each class by using various obfuscation techniques like string encryption, flow obfuscation and name obfuscation, this experiment obfuscated one method of an application with 0.9 mutation probability per iteration.

TABLE IV
THE PERFORMANCE OF COMMERCIAL AVS AGAINST COEVOLVED MALWARE

Commercial AV	Original	Coevolved Malware	Zelix K.	Top 100 Apps (FPR)
AVAST	77.78%	55.56%	22.22%	0.00%
ESET	88.89%	66.67%	88.89%	2.04%
GDATA	66.67%	66.67%	66.67%	0.00%
IKARUS	100.00%	55.56%	55.56%	3.06%
NORTON	100.00%	33.33%	22.22%	0.00%
TRENDMICRO	88.89%	66.67%	66.67%	0.00%

The best coevolved malware in each family against commercial AVs is shown in Table V. It should be noted that the anti-malware evaluated is some of the most powerful solutions in the market. In the table, O indicates the original malware, and C indicates the coevolved version of the same malware, whilst ✓denotes malware detected by the AV, otherwise it evaded detection. The results demonstrate that AVs could be robust to different types of malware. The difference should be based on their detection method and their signature database, which are not specified publicly. DroidChameleon [10] shows that the combinations of some obfuscation techniques should be applied for a particular malware family in order to evade detection from a particular anti-malware tool. For example, in order to evade detection from Trendmicro, two transformations should be applied to the BaseBridge family: EE (encrypt native exploit) and RF (rename files). Similarly, at least EE should also be applied in order to evade detection from ESET. However, this technique is not fully automatable as stated in [10]; therefore, this technique was not applied in this study. Hence, results showed that BaseBridge could not evade detection from these solutions. BaseBridge, which downloads its malicious code at runtime, is one of the most evasive malware families. Other than BaseBridge, all families evades one or more AVs. The evasiveness of all coevolved malware can be seen in Figure 5.

TABLE V
THE PERFORMANCE OF AVS AGAINST THE BEST COEVOLVED MALWARE IN EACH FAMILY

COMM. AV / FAMILY	AnserverBot		Asroot		BaseBridge		DroidKungFu1		DroidKungFu2		FakeNetflix		GPSSMSSpy		HippoSMS		NickySpy	
	O	C	O	C	O	C	O	C	O	C	O	C	O	C	O	C	O	C
AVAST	✓	×	✓	✓	×	×	✓	✓	✓	✓	✓	×	×	×	✓	✓	✓	✓
ESET	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	×	×	×	✓	×	✓	✓
GDATA	✓	✓	×	×	✓	✓	✓	✓	×	×	✓	✓	×	×	✓	✓	✓	✓
IKARUS	✓	✓	✓	✓	✓	✓	✓	×	✓	✓	✓	×	✓	×	✓	×	✓	✓
NORTON	✓	×	✓	×	✓	✓	✓	✓	✓	×	✓	×	✓	×	✓	×	✓	✓
TRENDMICRO	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	×	×	✓	✓	✓	×	✓	×

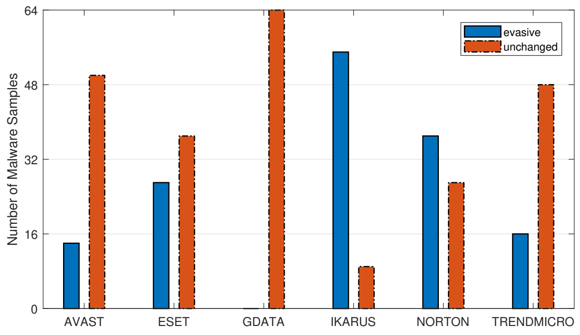

Fig. 5. The Performance of AVs against All Coevolved Malware

The results in Table V could be improved by adding more obfuscation techniques; such as file and package renaming which are known to be very effective techniques to evade detection. When these techniques are simply applied to the evolved malware, more evasions from the AV tools are noticed as given in Table VI. For example, it is observed that the NickSpy and HippoSMS families are weak against both techniques. Again, by applying package renaming to the Asroot family, the AVAST solution was deceived. Furthermore, by applying package renaming technique, all families evade detection from GDATA, the most robust anti-malware solution in Tables IV and V. This result supports that different AV tools are vulnerable to different obfuscation techniques. However, the application of these techniques could result in modification of more than one node in the GP tree. On the other hand, other obfuscation techniques used in the mutation operator change only one node at a time. Even when applicable to GP, the researchers elected to apply more suitable obfuscation techniques for GP at first. Therefore, more obfuscation techniques such as file, method, and package renaming, which could affect more nodes in one mutation operator, could be applied to all nodes in the evolved malware as applied in previous studies [9][10], which is also an automatable step.

TABLE VI
THE PERFORMANCE OF AVS AGAINST THE BEST COEVOLVED MALWARE IN EACH FAMILY AFTER APPLYING MORE OBFUSCATION TECHNIQUES

COMM. AV / FAMILY	AnserverBot		Asroot		BaseBridge		DroidKungFu1		DroidKungFu2		FakeNetflix		GPSSMSSpy		HippoSMS		NickySpy	
	O	C	O	C	O	C	O	C	O	C	O	C	O	C	O	C	O	C
AVAST	✓	×	✓	×	×	×	✓	✓	✓	✓	✓	×	×	×	✓	×	✓	×
ESET	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	×	×	×	✓	×	✓	✓
GDATA	✓	×	×	×	✓	×	✓	×	×	×	✓	×	×	×	✓	×	✓	×
IKARUS	✓	✓	✓	✓	✓	✓	✓	×	✓	✓	✓	×	✓	×	✓	×	✓	✓
NORTON	✓	×	✓	×	✓	✓	✓	✓	✓	×	✓	×	✓	×	✓	×	✓	✓
TRENDMICRO	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	×	×	✓	✓	✓	×	✓	×

In order to see the capability of the proposed system against Zelix KlassMaster, the best coevolved malware in each malware family in Table V was compared with its obfuscated version by Zelix KlassMaster in Table VII. *Original* indicates the number of AV solutions (out of six) that detect the original malware. With comparable results, it is hard to state one method as being superior over another. However, when more obfuscation techniques are applied, coevolved malware provides slightly better results than Zelix KlassMaster (5 more evasions). It is observed that Zelix KlassMaster does not change the package names in the manifest file, hence it becomes less effective than package renaming technique applied in this study.

TABLE VII
THE COMPARISON BETWEEN COEVOLUTION AND ZELIX KLASSMASTER

Family	Malware	Original	Coevolved	Zelix K.
AnserverBot	1de1d5f21a4afefb7c345187cd391ab83589f85a	6	4	4
Asroot	0c059ad62b9dbccf8943fe4697f2a6b0cb917548	5	4	3
BaseBridge	6d5ed5ca8434a571f21fc0770f4f8ba689b20fed	5	5	4
DroidKungFu1	02d2e109d16d160f77a645f44314fedcdbcd6e18	6	5	6
DroidKungFu2	1fce3240ea70b77e0d11c009788ae7cea5c87f77	5	4	4
FakeNetflix	0936b366cbc39a9a60e254a05671088c84bd847e	5	1	2
GPSSMSSpy	af727f5e23e69bfe2321f5d556c63f741dae8283	3	1	0
HippoSMS	bd7e85f5a0c39a9aeecc05dbc99a9e5c52150ba6	6	2	2
NickySpy	1ce27fa92a313da39f1e31e97d3ac05a8d6ffe78	6	5	4

(Co)evolved malware has evasive features added in some parts of the code that helps it to evade anti-malware systems. Since coevolved malware is evaluated against both commercial AVs and coevolved anti-malware, the coevolution mechanism has a potential in developing more evasive malware than the evolution mechanism naturally. This output was also observed in our experiments.

It was also analyzed as to whether or not these (co)evolved malware could be detected by the detection systems developed by (co)evolutionary computation. While GpAV shows the best individual evolved by anti-malware evolution in Table VIII, CoAV represents the best anti-malware solutions evolved by malware/anti-malware coevolution. The best individuals were selected among 150 runs. While GpAV shows very high accuracy on known attacks, it is ineffectual against evolved malware. CoAV-1 and CoAV-2 show the perfect detection rate. It should be noted that coevolved anti-malware are only tested on coevolved malware if they did not coevolve in the same running. However, the (co)evolved malware used in training and testing could show a degree of similarity, since they are obfuscated through the same tool. Therefore, the coevolved anti-malware systems are also evaluated against variations of malware generated by other tools as shown in the subsequent section.

It is difficult to have a detection system with both a perfect detection rate and a perfect false positive rate as there is a clear trade-off between detection rate and false positive rate. However, the authors believe that the consequences of letting a malware be installed on a device is more critical than preventing the installation of benign applications. Furthermore, the proposed static analysis-based detection system can be complemented with other solutions such as sending suspicious applications for dynamic analysis. Whitelisting, where commercial anti-malware solutions mainly apply not to detect well-known applications as grayware/malware, can also be applied in order to decrease the false positive rate. In choosing distinguished features of malicious applications, DroidAPIMiner [27] and SAFEDroid [48] were referred to as they achieve a high detection rate and a low false positive rate. The authors believe that accuracy of coevolved detectors could be increased by way of enlarging the training size [50]. Moreover, additional features could be introduced for a lower false positive rate by accepting lower detection rate, as seen in [13].

TABLE VIII
THE PERFORMANCE OF (CO)EVOLVED ANTI-MALWARE SYSTEMS

	Detection Rate		False Positive Rate	
Anti-Malware	Evolved Malware	Coevolved Malware	Top 100	Top 1000
GpAV	48.44%	42.86%	0.00%	1.58%
CoAV-1	100.00%	100.00%	5.00%	5.59%
CoAV-2	100.00%	100.00%	7.00%	12.09%

Finally, periodic updates to commercial anti-malware solutions were applied in order to see their effect on coevolved malware over time. Updates were applied that were released between 2014 and September 2017 for each anti-malware in Table II. Even though GDATA was updated, their recent updates were not received and were therefore excluded from the results. When an original malware is out in the wild, anti-malware solutions also take precautions in detecting it, as shown in Figure 6. Their detection rates on coevolved malware also increase in time as shown in Figure 7. It was interesting to note that their solutions develop over time for malware similar to coevolved malware in this study.

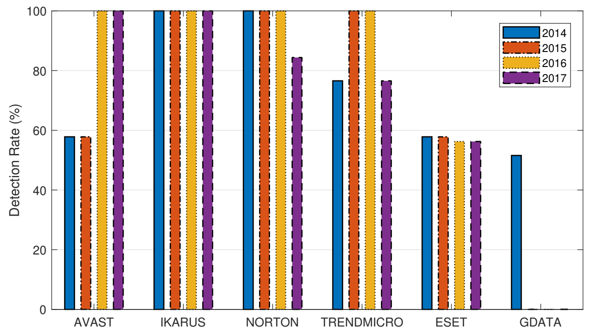

Fig. 6. The Detection Rate of Original Malware in Time

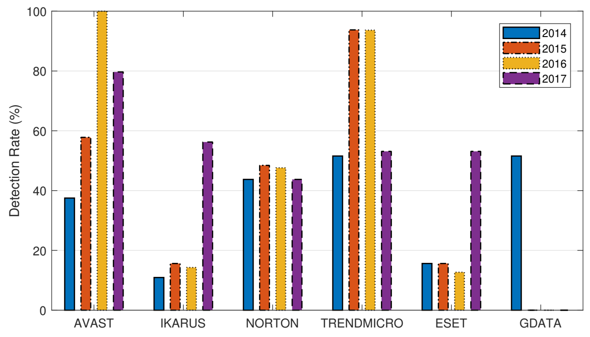

Fig. 7. The Detection Rate of Coevolved Malware in Time

B. Performance of Coevolved Anti-Malware

The proposed detection system was also evaluated against new variants of existing malware (see Table IX). A new dataset of obfuscated malware called PRAGuard dataset was released in a recent study [12]. PRAGuard contains malware from the MalGenome [45] and the Contagio datasets [51], and then different obfuscation techniques were applied. There are trivial techniques (TRI) which only affect strings such as renaming of classes/methods, and non-trivial techniques that both affect string and bytecode of executables. The non-trivial obfuscation techniques used in the PRAGuard dataset are string encryption (SE), reflection (REF), and class encryption (CE). While some techniques applied in PRAGuard and the current study are common, there are also different techniques as well, and combinations of different obfuscation techniques are also applied in the experiments. The study tested the proposed detection system on the malware obfuscated by all techniques in PRAGuard in order to check the system. The PRAGuard dataset contained 608 malware, whose families are used in the malware (co)evolution. Results show that coevolved anti-malware systems perform superior on obfuscated malware; almost achieving a perfect detection rate, while commercial anti-malware solutions other than GDATA missed most of the obfuscated malware. Only some malware (n=17) from PRAGuard was able to be installed on emulators in order to evaluate them against commercial anti-malware.

TABLE IX
THE DETECTION RATE OF ANTI-MALWARE ON NEW VARIANTS OF MALWARE

Anti-Malware	Original	PRAGuard (All)	Zelix K.	Drebin
CoAV-1	94.08%	98.85%	94.08%	86.58%
CoAV-2	96.38%	100.00%	98.16%	95.95%
AVAST	95.35%	0.00%	75.92%	99.22%
ESET	99.00%	11.76%	94.49%	97.67%
GDATA	66.61%	64.71%	60.00%	86.56%
IKARUS	100.00%	17.65%	74.29%	100.00%
NORTON	73.09%	11.76%	18.37%	95.87%
TRENDMICRO	98.84%	11.76%	83.27%	97.16%

The effect of each obfuscation technique employed in PRAGuard dataset can be seen in Table X. Coevolved anti-malware detects most malware obfuscated by each technique including class encryption. Class encryption, which completely encrypts and compresses each class, is claimed to be the most powerful and advanced technique employed in the PRAGuard dataset [12].

TABLE X
THE DETECTION RATE OF COEVOLVED ANTI-MALWARE ON PRAGUARD DATASET

Anti-Malware	CE	REF	SE	TRI	TRI+SE	TRI+SE+REF	TRI+SE+REF+CE
CoAV-1	98.85%	93.91%	93.91%	93.91%	94.24%	94.24%	98.85%
CoAV-2	93.09%	96.55%	96.55%	96.55%	96.55%	96.38%	100.00%

The proposed anti-malware systems were also tested on malware obfuscated by Zelix KlassMaster, a well-known Java obfuscator [11] as shown in Table IX. The proposed system was also found to be very effective against obfuscated malware, while commercial anti-malware other than ESET missed many malware obfuscated by Zelix KlassMaster, and did not show any improvement over time. Only AVAST showed mild improvement (6.25%) against malware obfuscated by Zelix KlassMaster over time.

The proposed systems were also evaluated against the new variants of malware and 0-day attacks in the Drebin dataset [13]. The six families in Drebin; BaseBridge, DroidKungFu, DroidDream, Geimini, GoldDream, and Kmin have many more samples than the MalGenome dataset. Since these samples (n=490) are not used in the coevolution training, they are unknown to the proposed system. The results demonstrate that the coevolved anti-malware system effectively detected new variants of known attacks. Commercial anti-malware solutions are also very effective against the Drebin dataset. This was an expected result, since malware in the dataset is quite well known since its release in 2014. The performance of coevolved malware on each family in the Drebin dataset can be seen in Table XI. The tested (co)evolved anti-malware systems were the least effective against the BaseBridge, DroidKungFu and GoldDream families. It should be noted that the BaseBridge and DroidKungFu families are known to be difficult to detect due to their loading of malicious code at runtime [52]. GoldDream might also require dynamic analysis for detection in order to observe its bot behavior.

TABLE XI
THE PERFORMANCE OF COEVOLVED ANTI-MALWARE SYSTEMS ON DREBIN DATASET

Anti-Malware	BaseBridge	DroidDream	DroidKungFu	Geinimi	GoldDream	Kmin
CoAV-1	81.81%	97.05%	81.03%	96.00%	56.52%	100.00%
CoAV-2	86.36%	100.00%	95.90%	96.00%	91.30%	97.92%

To sum up, in this study, new variants of malware and new anti-malware systems are generated automatically by using coevolutionary-based computation techniques. One of the most important characteristics of the coevolution process is that it is fully automated, so long as only mutation operator is applied. It is shown that coevolved malware are more evasive than their original versions, hence they could evade from commercial anti-malware systems. The results show that these security solutions could be ineffective against different obfuscation techniques. Hence, coevolved malware could be more evasive by adding more obfuscation techniques into the coevolution system. Furthermore, it is shown that over time, coevolved malware could be detected by such solutions. Therefore, coevolved malware shows similarity to malware in the wild, and security companies need to develop their anti-malware solutions against such attacks. The results also show that developed anti-malware systems are very effective against new variants of malware and obfuscated malware, and much better than commercial anti-malware systems. Since there is a significant growth in the number of new Android malware variants [2], automatically improving anti-malware systems as performed in the current study becomes vital.

VII. DISCUSSION

This study investigated the use of coevolutionary computation techniques for the development of malicious and anti-malware software. It is believed to be the first application of coevolutionary computation to systems security. Although the proposed approach produced promising results from the point of malware and anti-malware evolution, the system also has some limitations which are discussed in the following.

Fully automated: The aim was to develop a fully automated system. GP has already been applied for malware evolution in some approaches in the literature. However, these approaches are not fully automated and only proposed for specific types of attacks. A security expert is generally needed to analyze the code and to extract parameters that changes in different variants of malware' codes, so a representation of the problem could be constructed for GP. On the other hand, the current study used the CGs of malicious applications, obtained automatically, as inputs to the GP algorithm in the malware evolution. Then, new applications were evolved by applying genetic operators such as crossover and mutation. Here, the mutation operator does not change the functionality of the code, hence its maliciousness. On the other hand, the crossover operator could change the malicious code in an application. In order to assure the maliciousness of an application, a ML-based detection system was developed. However, this detection system does not provide perfect detection, the same as any system developed for detection of malware either in academia or the security industry. It also has certain limitations such as triggering malicious code and running for a limited time period. There are a few precautions that could be taken in order not to have to analyze evolved malware manually. More than one detector could be used for labeling applications as malicious or benign, which would increase the credibility of the system. Excluding the crossover operator would solve the problem of manually analyzing some evolved malware. In the authors' previous study [8], malware was evolved using only the mutation operator or the mutation-crossover operators together, which shows that the mutation operator is sufficiently powerful to generate evasive malware. Furthermore, GP only outputs legal applications with respect to its ability to eliminate non-compiled and non-executable malware.

Sufficient number of malware: This study used a limited number of malware families existing in the MalGenome dataset [45]. In order to have adequate sample numbers both for training and testing, the larger malware families were chosen in the dataset. Samples were also selected that did not include recursive methods, since GP accepts trees as inputs, not graphs. This could limit the applicability of the proposed approach to certain kinds of malware, but not all. However, this could be overcome by replacing recursive with iterative methods before obtaining the CGs.

VIII. Conclusion

Mobile malware is one of today's biggest security issues. Malware writers have become more attracted to mobile devices in recent years since these devices have become a widespread, integral part of daily life. Security firms also release solutions for mobile devices. Since mobile devices have certain power limitations, most proposed anti-malware solutions in the market rely on static analysis techniques. However, these techniques could be more open to new attacks or even new variants of known attacks than dynamic analysis techniques. Therefore, these techniques need to be evaluated against unseen attacks, which is one of the aims of this current study. New mobile malware was successfully generated from known malware by using GP. These attacks are seen to be quite effective against the popular security solutions in the market. One of the most powerful features of the proposed approach is its applicability to any mobile malware. In order to evolve new variants of malware, analysis of the malware or knowledge of a security expert is no longer needed, with this approach able to work as fully automated. In the future, functionality could be extended by adding dynamic code-loading features [23], or adding method renaming technique in order to generate more evasive attacks. Moreover, the similarity of evolved malware to the original malware could be taken into account in the fitness function as in [7].

The main aim of this study was to investigate the use of coevolutionary computation techniques on the development of mobile malware and anti-malware. The researchers are aware of no other approach in the literature that can do this. By using coevolutionary computation techniques, the proposed system evolved more evasive malware and more robust anti-malware against unseen attacks. The anti-malware system developed shows a superior performance on new datasets, which reveals its robustness to new attacks. In the future, work could be undertaken to decrease its false positive rate by running the algorithm against a larger training set, and exploring more features for detection.

The possibility of malware/anti-malware coevolution has always been a point of academic interest [33]. The authors believe that this has been successfully achieved in this study. Researchers could also apply these techniques to new areas to be explored in other security areas such as intrusion detection and prevention, and employ generated intrusions in penetration testing.

Acknowledgment

This study is supported by the Scientific and Technological Research Council of Turkey (TUBITAK-112E354). We would like to thank TUBITAK for its support.

References

[1] K. Lab. (March 2016) The volume of new mobile malware tripled in 2015. http://www.kaspersky.com/about/news/virus/2016/The_Volume_of_New_Mobile_Malware_Tripled_in_2015.

[2] Symantec. (April 2016) Internet security threat report. https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf.

[3] H. G. Kayacık, M. Heywood, and N. Zincir-Heywood, "On evolving buffer overflow attacks using genetic programming," in *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '06, 2006, pp. 1667–1674.

[4] H. G. Kayacık, A. N. Zincir-Heywood, M. I. Heywood, and S. Burschka, "Generating mimicry attacks using genetic programming: A benchmarking study."

[5] H. G. Kayacık, A. N. Zincir-Heywood, and M. I. Heywood, "Can a good offense be a good defense? vulnerability testing of anomaly detectors through an artificial arms race," *Applied Soft Computing*, vol. 11, no. 7, pp. 4366–4383, Oct. 2011.

[6] H. G. Kayacık, A. N. Zincir-Heywood, and M. Heywood, "Evolutionary computation as an artificial attacker: generating evasion attacks for detector vulnerability testing," *Evolutionary Intelligence*, vol. 4, pp. 243–266, 2011.

[7] S. Noreen, S. Murtaza, M. Z. Shafiq, and M. Farooq, "Evolvable malware," in *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '09, 2009, pp. 1569–1576.

[8] E. Aydogan and S. Sen, "Automatic generation of mobile malwares using genetic programming," in *European conference on the applications of evolutionary computation*. Springer, 2015, pp. 745–756.

[9] M. Zheng, P. P. C. Lee, and J. C. S. Lui, “Adam: An automatic and extensible platform to stress test android anti-virus systems,” in *Detection of Intrusions and Malware, and Vulnerability Assessment*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013.

[10] V. Rastogi, Y. Chen, and X. Jiang, “Droidchameleon: Evaluating android anti-malware against transformation attacks,” in *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, ser. ASIA CCS ’13, 2013, pp. 329–334.

[11] AV-TEST. (February 2016) Zelix klassmaster: Java obfuscator - zelix klassmaster. http://www.zelix.com/.

[12] D. Maiorca, D. Ariu, I. Corona, M. Aresu, and G. Giacinto, “Stealth attacks: An extended insight into the obfuscation effects on android malware,” *Computers & Security*, vol. 51, pp. 16–31, 2015.

[13] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, “Drebin: Effective and explainable detection of android malware in your pocket.” in *NDSS*, 2014.

[14] M. Christodorescu and S. Jha, “Testing malware detectors,” in *Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2004)*. Boston, MA, USA: ACM Press, Jul. 2004, pp. 34–44.

[15] J. A. Morales, P. J. Clarke, Y. Deng, and B. Golam Kibria, “Testing and evaluating virus detectors for handheld devices,” *Journal in Computer Virology*, vol. 2, pp. 135–147, 2006.

[16] A. Moser, C. Kruegel, and E. Kirda, “Limits of static analysis for malware detection,” in *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, Dec 2007, pp. 421–430.

[17] I. You and K. Yim, “Malware obfuscation techniques: A brief survey,” in *Broadband, Wireless Computing, Communication and Applications (BWCCA), 2010 International Conference on*, Nov 2010, pp. 297–300.

[18] M. Christodorescu, J. Kinder, S. Jha, K. S., V. H., and T. U. Mnchen, “Malware normalization,” Tech. Rep., 2005.

[19] L. Wu and Y. Zhang, “Research of the computer virus evolution model based on immune genetic algorithm,” in *Proceedings of the 2011 10th IEEE/ACIS International Conference on Computer and Information Science*, ser. ICIS ’11, 2011, pp. 9–13.

[20] F. Shahzad, M. Saleem, and M. Farooq, “A hybrid framework for malware detection on smartphones using elf structural & pcb runtime traces,” Tech. Report TR-58 FAST-National University, Pakistan, Tech. Rep., 2012.

[21] S. Noreen, S. Murtaza, M. Z. Shafiq, and M. Farooq, “Using formal grammar and genetic operators to evolve malware,” in *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection*, ser. RAID ’09, 2009, pp. 374–375.

[22] G. Meng, Y. Xue, C. Mahinthan, A. Narayanan, Y. Liu, J. Zhang, and T. Chen, “Mystique: Evolving android malware for auditing anti-malware tools,” in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*. ACM, 2016, pp. 365–376.

[23] Y. Xue, G. Meng, Y. Liu, T. H. Tan, H. Chen, J. Sun, and J. Zhang, “Auditing anti-malware tools by evolving android malware and dynamic loading technique,” *IEEE Transactions on Information Forensics and Security*, 2017.

[24] A. I. Aysan and S. Sen, “Do you want to install an update of this application? a rigorous analysis of updated android applications,” in *Cyber Security and Cloud Computing (CSCloud), 2015 IEEE 2nd International Conference on*. IEEE, 2015, pp. 181–186.

[25] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, “Android permissions demystified,” in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 627–638.

[26] W. Enck, M. Ongtang, and P. McDaniel, “On lightweight mobile phone application certification,” in *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 2009, pp. 235–245.

[27] Y. Aafer, W. Du, and H. Yin, “Droidapiminer: Mining api-level features for robust malware detection in android,” in *Security and Privacy in Communication Networks*. Springer, 2013, pp. 86–103.

[28] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, “Riskranker: scalable and accurate zero-day android malware detection,” in *Proceedings of the 10th international conference on Mobile systems, applications, and services*. ACM, 2012, pp. 281–294.

[29] A.-D. Schmidt, R. Bye, H.-G. Schmidt, J. Clausen, O. Kiraz, K. A. Yüksel, S. A. Camtepe, and S. Albayrak, “Static analysis of executables for collaborative malware detection on android,” in *Communications, 2009. ICC’09. IEEE International Conference on*. IEEE, 2009, pp. 1–5.

[30] M. Lindorfer, M. Neugschwandtner, and C. Platzer, “Marvin: Efficient and comprehensive mobile app classification through static and dynamic analysis,” in *Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual*, vol. 2. IEEE, 2015, pp. 422–433.

[31] A. Martín, H. D. Menéndez, and D. Camacho, “Mocdroid: multi-objective evolutionary classifier for android malware detection,” *Soft Computing*, pp. 1–11, 2016.

[32] T. A. Le, T. H. Chu, Q. U. Nguyen, and X. H. Nguyen, “Malware detection using genetic programming,” in *Computational Intelligence for Security and Defense Applications (CISDA), 2014 Seventh IEEE Symposium on*, Dec 2014, pp. 1–6.

[33] C. Nachenberg, “Computer virus-coevolution,” *Communications of the ACM*, vol. 50, no. 1, pp. 46–51, 1997.

[34] S. Sen, “A survey of intrusion detection systems using evolutionary computation,” *Bio-Inspired Computation in Telecommunications*, pp. 73–94, 2015.

[35] M. Ostaszewski, F. Seredynski, and P. Bouvry, “Coevolutionary-based mechanisms for network anomaly detection,” *Journal of Mathematical Modelling and Algorithms*, vol. 6, no. 3, pp. 411–431, 2007. [Online]. Available: http://dx.doi.org/10.1007/s10852-007-9061-x

[36] A. A. tool for reverse engineering Android apk files. (March 2016) https://code.google.com/p/android-apktool/.

[37] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.

[38] ECJ. (March 2016) A java-based evolutionary computation research system. http://cs.gmu.edu/ eclab/projects/ecj/.

[39] H. B. Ozkan, E. Aydogan, and S. Sen, “An ensemble learning approach to mobile malware detection,” Hacettepe University, Department of Computer Engineering, Tech. Rep., 2014, http://eprints.cs.hacettepe.edu.tr/7/.

[40] Droidbox. (Visited July 2017) [Online]. Available: https://code.google.com/p/droidbox/.

[41] Monkey. (Visited July 2017) [Online]. Available: https://developer.android.com/studio/test/monkey.html.

[42] S. R. Choudhary, A. Gorla, and A. Orso, “Automated test input generation for android: Are we there yet?(e),” in *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*. IEEE, 2015, pp. 429–440.

[43] AV-TEST. (August 2014) The independent it-security institute. http://www.av-test.org/en/home/.

[44] GenyMotion. (March 2017) Genymotion android emulator fast easy anywhere. https://www.genymotion.com/.

[45] Y. Zhou and X. Jiang, “Dissecting android malware: Characterization and evolution,” in *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, ser. SP ’12, 2012, pp. 95–109.

[46] Google. (March 2016) Google play. https://play.google.com/store.

[47] A. Feizollah, N. B. Anuar, R. Salleh, and A. W. A. Wahab, “A review on feature selection in mobile malware detection,” *Digital Investigation*, vol. 13, pp. 22–37, 2015.

[48] S. Sen, A. I. Aysan, and J. A. Clark, “SAFEDroid: Using structural features for detecting android malwares,” in *Proceedings of the 13th EAI International Conference on Security and Privacy in Communication Networks (SecureComm 2017 Workshops), LNICST 239*. Springer, 2018, pp. 255–270.

[49] R. Poli, W. B. Langdon, N. F. McPhee, and J. R. Koza, “Genetic programming: An introductory tutorial and a survey of techniques and applications,” *University of Essex, UK, Tech. Rep. CES-475*, 2007.

[50] S. Y. Yerima, S. Sezer, G. McWilliams, and I. Muttik, “A new android malware detection approach using bayesian classification,” in *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on*. IEEE, 2013, pp. 121–128.

[51] Contagio. (March 2016) contagio. http://contagiodump.blogspot.com.tr/.

[52] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu, “Droidmat: Android malware detection through manifest and api calls tracing,” in *Information Security (Asia JCIS), 2012 Seventh Asia Joint Conference on*. IEEE, 2012, pp. 62–69.