Exploring an Artificial Arms Race for Malware Detection

Zachary Wilkins
Dalhousie University
Computer Science
Halifax, Nova Scotia, Canada
zachary.wilkins@dal.ca

Ibrahim Zincir
Izmir University of Economics
Software Engineering
Izmir, Turkey
ibrahim.zincir@ieu.edu.tr

Nur Zincir-Heywood
Dalhousie University
Computer Science
Halifax, Nova Scotia, Canada
zincir@cs.dal.ca

ABSTRACT

The Android platform commands a dramatic majority of the mobile market, and this popularity makes it an appealing target for malicious actors. Android malware is especially dangerous because of the versatility in distribution and acquisition of software on the platform. In this paper, we continue to investigate evolutionary Android malware detection systems, implementing new features in an artificial arms race, and comparing different systems' performances on three new datasets. Our evaluations show that the artificial arms race based system achieves the overall best performance on these very challenging datasets.

CCS CONCEPTS

• **Computing methodologies** → **Genetic algorithms**; • **Security and privacy** → **Malware and its mitigation**;

KEYWORDS

Cyber security, Machine learning, Malware, Mobile security, Smartphone, Android, Cyber attack, Evolution

ACM Reference Format:
Zachary Wilkins, Ibrahim Zincir, and Nur Zincir-Heywood. 2020. Exploring an Artificial Arms Race for Malware Detection. In *Genetic and Evolutionary Computation Conference Companion (GECCO '20 Companion), July 8–12, 2020, Cancún, Mexico.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3377929.3398090

1 INTRODUCTION

Cell phones have become an indispensable part of our modern world. From their humble origins as large and cumbersome telephone bricks, they now sport sleek profiles and are capable portable computers. They facilitate our communications through calling, messaging, and email services, as well as many digital conveniences such as online shopping and banking. In 2016, mobile web browsing surpassed desktop web browsing for the first time [6], cementing their place as our go-to Internet devices. The degree with which we interface with our smartphones has resulted in these devices containing large amounts of personal information. As a result, our phones have attracted the attention of various cyber criminals who would profit from such data.

While many manufacturers fabricate smartphones, two operating systems control the vast majority of the market. Apple's iOS is built especially for their devices, but Google's Android powers many different brands to command a staggering 85% of the global market [2]. The diversity of devices running Android, and the inconsistent adoption rates[1] with respect to upgrades, means that a one-size-fits-all security solution is difficult to create. Furthermore, Android users have significant flexibility in how their applications are installed, and can elect to install applications from channels outside of the primary Google Play Store. Malicious actors have leveraged this reality to create an ecosystem of malware targeting Android devices.

Security professionals have responded in-kind with many anti-malware solutions. As malware has become more adaptable to these detectors, machine learning is a popular technique for building detection models [4] [13] [10] [20]. Genetic algorithms have been especially successful, as they can evolve detectors that are specifically hardened against resilient malicious behaviours [11] [12]. Observing their success in standalone situations, some researchers have created adversarial, co-evolutionary systems [3] [18], simulating the real-world arms race that exists between attackers and defenders. We refer to this as the artificial arms race.

This paper extends our previous work in [23], where we evaluated an evolutionary, co-evolutionary, and state-of-the-art, rule-based system by training them on older, established datasets before testing them on newer, unseen datasets. This time, we evaluate the performance of an aggregate detection web service, before extending the aforementioned artificial arms race system to include certificate-based features. Finally, we benchmark all four systems on three novel datasets. The results indicate that the new datasets are challenging for all systems, but the ordering from [23] is preserved, with the evolutionary system competing with or exceeding the rule-based system, and the artificial arms race (co-evolutionary) system besting both. The new features provide a mixed result, with an increased benign accuracy, but lower malicious rates.

The rest of this paper is organized as follows. We begin by reviewing related research in Section 2. We discuss the detection systems and the datasets employed in Section 3. Results are presented and discussed in Section 4. Section 5 concludes our observations and suggests directions for future work.

GECCO '20 Companion, July 8–12, 2020, Cancún, Mexico

ACM ISBN 978-1-4503-7127-8/20/07…$15.00
https://doi.org/10.1145/3377929.3398090

[1] https://developer.android.com/about/dashboards/

2 RELATED WORK

A variety of approaches are being employed to increase malware detection rates. In this section, we explore benchmark surveys on machine learning techniques, before detailing evolutionary works.

From a general machine learning (ML) perspective, Ucci et al. conduct a survey of popular ML approaches for malware detection [20]. They identify specific trends that exist, and the current shortcomings. One area of interest that they identify is being able to predict future variants of malware, and this is an ideal task for evolutionary algorithms.

With that in mind, Wilkins et al. benchmark evolutionary, co-evolutionary, and state-of-the-art, rule-based systems [23]. The evolutionary systems are trained on older, well-established academic datasets, as well as benign samples from popular app stores. The authors then employ newer, unseen datasets so as to ascertain the relative effectiveness of each system. The co-evolutionary system significantly outperforms the other systems, especially on newer datasets.

Highlighting individual solutions that inform the benchmarking survey, Meng et al. audit anti-malware tools by developing an evolutionary malware generation system, Mystique [12]. After codifying a number of attack and evasion features, they evolve malicious models where the presence of a set bit indicates a particular chosen feature. They achieve a dramatic reduction in the mean detection rate, dropping over 60 percent points. Their follow up, Mystique-S, adds dynamically loaded malware as an evasion technique [24].

Rather than extracting specifically chosen features, Martín et al. develop an evolutionary malware detector for Android applications that operates on extracted 3rd party imports [11]. These imports are indicative of the behaviours of the applications, and are difficult to obfuscate without breaking compatibility with the library. After clustering imports, models are produced for malicious and benign software, with which new samples can be compared. The resulting classifiers achieve a maximum accuracy of 95%.

Inspired by the success of Mystique, Bronfman-Nadas et al. go a step further by simulating the "arms race" that exists between the authors of malware and security software [3]. They develop companion generators to create Android malware and Android malware detectors, and combine them in a feedback loop. As the goals of these generators are in conflict, they cannot both be optimal simultaneously. Their results indicate that the evolved detectors are highly effective, and less complex than comparable solutions.

In a similar vein, Sen et al. create and combine an Android malware generator and detector into a co-evolutionary framework [18]. Instead of the basic reusable features employed in [12] and [3], their system operates on smali, the assembler language for the modern Android platform, and generates control flow graphs that are evolved. The detectors that emerge from this process are much more accurate than comparable tools, even though the evolved malware is highly evasive.

Since security practitioners always need to be working ahead of malicious threat actors, solutions that can predict future malware are especially promising. Co-evolutionary systems have demonstrated that they are capable of fulfilling this need, but a comprehensive survey has not been conducted. To further understand the advantages of co-evolutionary systems against other popular methodologies, in this work, we continue to benchmark and contrast evolutionary, co-evolutionary, state-of-the-art rule-based, and aggregate detection systems.

3 METHODOLOGY

In this section, we describe the detection systems and datasets, as well as discuss the exploratory process by which we produced new features for the co-evolutionary system.

3.1 Detectors

3.1.1 MOCDroid. MOCDroid is an evolutionary malware detector generation system [11]. It operates on the import calls made from 3rd party libraries, as this is difficult to obfuscate without breaking compatibility. In this way, malicious behaviours can be identified by common behavioural groupings. As MOCDroid operates on decompiled class files contained in APKs, it is a form of static analysis.

After the imports are extracted, they are clustered in R. MOCDroid takes a text mining approach, as it envisions each application as a document and each import as a term in that document. After applying k-Means, two matrices are produced.

These matrices represent benign and malicious app imports, and are used in a multi-objective genetic algorithm, where a classifier is trained by flipping bits indicating the presence of import clusters. The algorithm seeks to maximize accuracy while minimizing false positives.

3.1.2 ArmsRace. ArmsRace is a co-evolutionary malware and malware detector generation system that aims to replicate the adversarial relationship between attackers and defenders [3]. By evolving predictions of future malware based on current samples, the generated models can be more resilient to new threats.

The detector generator creates small, linear programs that consist of instructions for a simple virtual machine. This machine is capable of performing arithmetic operations, as well as reading from or writing to registers. The programs operate on 23 features that include relevant Android permissions, as well as behaviourly-indicative code features.

The malware generator is inspired by Mystique [12], outputting a list of aggressive and evasive features to be used as a template for the generated malware. An indicator-based evolutionary algorithm is employed to satisfy three objectives: maximize aggressiveness, minimize evasion, minimize detectability.

By combining these systems into a feedback loop, they compete against each other, aiming for optimality. Their adversarial nature ensures that both cannot be optimal simultaneously.

3.1.3 Assemblyline. Assemblyline (AL) is a modular, scalable, open-source malware analysis system from the Canadian Centre for Cyber Security [15]. The system can run in a virtual machine on a single computer, or across many computers and operate in a distributed fashion. Users can interface with Assemblyline programmatically via the API, through a Python library, or their web browser. Assemblyline is employed by cyber security professionals in Canada and developed by the "Government of Canada's centre of excellence in cyber security" [15]. Accordingly, we considered it to be a state-of-the-art system.

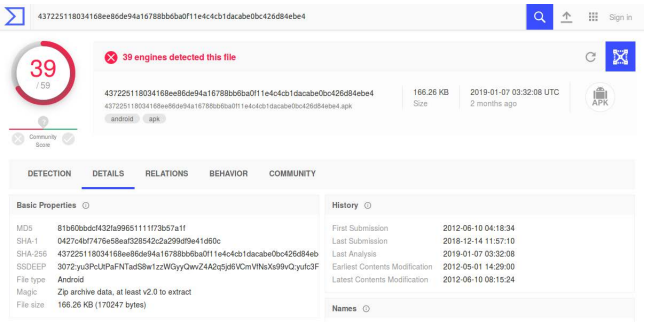

Figure 1: VirusTotal results for a malicious APK. This file is part of the Drebin dataset.

The system operates a number of services which cater to specific analysis techniques and file types. Users are encouraged to develop their own services, and consider contributing them back to the Assemblyline ecosystem. Each service has its own defined heuristics, which are summed to calculate the final score. By default, files less than -1,000 are considered benign, while files over 1,000 are malicious. The range between indicates how certain the service is about the label it is assigning. The system raises an alert for any file scoring above 500.

To assess Android malware, Assemblyline ships with APKaye. APKaye is a static analysis service, disassembling the suspicious file and assessing various permissions, code features, strings, and packaged scripts or executables.

3.1.4 VirusTotal. VirusTotal (VT) is a community-driven, corporately supported knowledge base that aggregates results from over 70 malware detection systems [21]. The detectors include well-known commercial products such as Avast, AVG, ESET NOD 32, McAfee, Kaspersky, Symantec, TrendMicro, and ZoneAlarm.

Users can submit files for analysis, which are compared to existing samples, and receive detailed information. This includes the detectors that flagged the sample, the labels assigned, and various properties of the file. If possible, dynamic analysis is performed, which can highlight specific behaviours of the malware sample. Searches can also be performed using file hashes, and other identifying information, such as URLs and IP addresses.

A number of entrypoints are available to access VirusTotal services, including a web interface, shown in Figure 1, and API. Access to the Basic API is for non-commercial use only, and is rate-limited, while only returning a subset of the full report for each file, though this is sufficient for our comparisons. Expanded plans are available upon contact.

As VirusTotal does not require any setup, even an account, to use, it is the fastest of all of these detection systems. As well, having already seen many samples allows for results to be returned instantly, subject to the mentioned limitations.

To acquire information from VirusTotal, we computed the MD5 hashes of each sample in each dataset, and queried the VirusTotal API with a Python script. This is diagrammed in Figure 2, alongside the other detection systems. The values returned include the assigned labels for each sample, the detectors that assigned the labels,

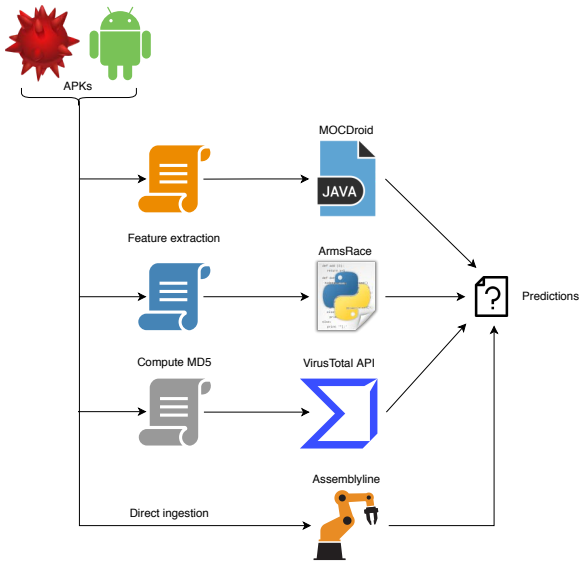

Figure 2: Detector submission process.

and the total number of detectors. From this, a detection rate was calculated.

An important point to note is that this detection rate is not directly comparable to the detection rate from other detectors, given the aggregate nature of the VirusTotal service. Detectors that are not capable of evaluating Android applications are still included, which serves to falsely lower the detection rate. It is more appropriate to consider the average, minimum, and maximum VirusTotal detection rate for each dataset, so as to surmise their relative detectability. As well, the Basic API does not allow for files greater than 10MB to be uploaded, so unseen hashes return no results at all. This typically only affects a small number of benign samples.

3.2 Datasets

In [23], we employed a number of datasets from a variety of sources to train and evaluate the detection systems, and we continue to evaluate them in this paper. These datasets include academic datasets such as the Android Malware Dataset (24,553 malware) [22], the Android Malware Genome Project (1,260 malware) [25], the Drebin Dataset (5,560 malware) [1] [19], and CICAndMal2017 (426 malware and 5,065 benign) [9]. In order to ease references to these datasets, we employ the following abbreviations, respectively: AMD, Genome, Drebin, and UNB Malware / Benign. Additionally, we use applications sourced from two popular Android app stores for [3], Google Play (1,085 benign) [8] and F-Droid (1,339 benign) [5]. We also acquired a large, crowd-sourced collection of malware from VirusShare (VS), a community-oriented malware sharing project [16].

In June 2019, VirusShare restructured how they distribute Android APK files [17]. Their previous Android malware dataset was a collection of samples from 2013 and 2014, hereafter referred to as VS1314. They have now made all APK files available by year, from 2012 to 2018. The training datasets for our models were all sourced

Detector	Accuracy
Arms Race	0/44 (0.0%)
Assemblyline	21/44 (47.73%)
MOCDroid	16/44 (36.36%)

Table 1: Detector accuracy on undetectable malware.

in the earliest years of the decade. In this paper, differing from previous work, we elected to download the three most recent years: 2016, 2017, and 2018. These three new malware dumps together contain upwards of 50,000 samples, and we hoped that their novelty could enable performance comparisons in respect to evolutionary predictions of future malware.

3.3 Deep Dive

Based on the results in [23], we concluded that ArmsRace was an overall more effective solution than MOCDroid, especially on newer datasets. For that reason, we decided to focus on improving its detection rate. UNB Malware was the most difficult dataset to predict for two of the three detectors in that work. Hence, it was the natural choice for further scrutiny.

Looking at the best performing model for ArmsRace (Drebin / Google Play), there were still 44 UNB Malware samples that failed to be detected. This is possibly because of the focus ArmsRace places on privacy leakage malware, which is not present in the UNB samples. To determine a path forward, it was necessary to dive into the composition of the undetectable malware, and find some common properties that may make their detection easier.

3.3.1 APK Properties. To determine which subset of these apps are particularly difficult, we evaluated the 44 samples in Assemblyline and MOCDroid, using MOCDroid's best model (Drebin / Google Play) [23].

These malware seem to employ more advanced evasion techniques, because the results in Table 1 are all unsatisfactory.

Of the samples missed by ArmsRace, 30 are also undetectable by Assemblyline, MOCDroid, or both. The number of samples that fail to be detected, in addition to their VirusTotal detection rate and common labels, is presented in Figure 4. Filtering it down, we end up with 21 APKs that are undetectable by all three systems. As VirusTotal employs a variety of commercial detectors, these 21 samples can be considered extremely evasive, and were subject to further scrutiny.

According to the labels provided by the researchers in [9], 15 of these samples are Scareware[2]named Android Defender. In fact, there are only 17 Android Defender samples in the UNB dataset. This indicates a highly evasive family. The remaining samples consist of three SMS Malware[3], each in the Nandrobox family, and three Adware[4](two from the Mobidash family, one from the Youmi family). These types of malware were not originally considered in ArmsRace, which focuses on privacy leakage, and might explain their evasiveness.

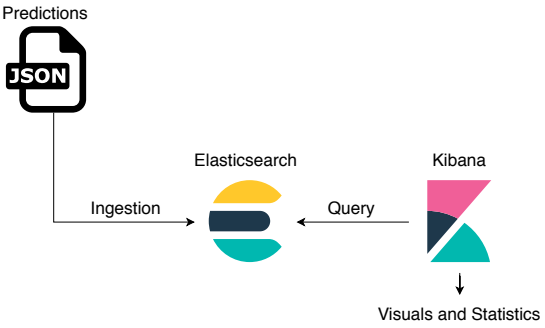

Figure 3: Elastic Stack ingestion and analysis process.

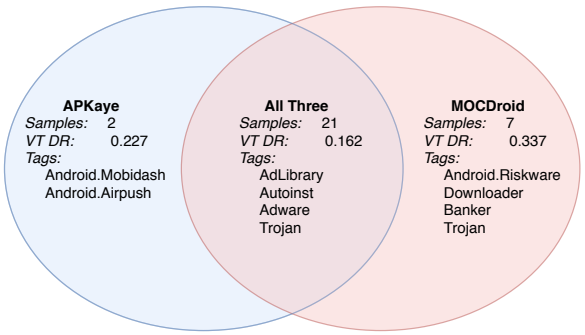

Figure 4: VirusTotal Venn diagram for UNB Malware undetectable by ArmsRace. VT DR is the detection rate.

To get a more comprehensive overview of each undetectable APK, we retrieved a number of distinguishing characteristics for each from VirusTotal and Assemblyline. The VirusTotal field of interest was the assigned labels. Extracted Assemblyline fields include the Android activity, signing certificate information, as well as minimum and target SDK. In addition, Assemblyline provides the heuristics it uses to identify suspicious characteristics of applications.

In order to simplify the analysis process, we compiled all of the aforementioned information and uploaded it to a local Elastic stack[5]cluster. The data was housed in Elasticsearch, a database and search engine, and visualized with Kibana, the accompanying front-end server. This ingestion process is depicted in Figure 3.

3.3.2 VirusTotal Labels. Calculating the average, minimum, and maximum detection rates for VirusTotal on the 21 completely undetectable samples, we end up with 16.2%, 3.4%, and 40%, respectively. This is a significant drop from the average and maximum reported

[2]This type of malware seeks to scare the user into paying the attacker by raising false alarms.
[3]This type of malware creates unwanted financial charges through premium SMS messages.
[4]This type of malware displays unwanted ads on the user's device.
[5]https://www.elastic.co/products/

Frequently Occurring VT Labels	Count
SPR/ANDR.Autoinst.A.Gen	11
a variant of Android/Autoins.C potentially unsafe	11
Trojan.Gen.2	10
Trojan/Android.TSGeneric	10
AdLibrary:MoPub	6
AdWare.AndroidOS.Autoins	6
Adware.Allinone.1.origin	6
Spr.Andr.Autoinst.A!c	5

Table 2: Labels assigned by VirusTotal for undetectable samples.

AL Heuristic	Count	Meaning
AL_APKaye_003	21	Network indicator found
AL_APKaye_004	21	Dangerous permission used
AL_APKaye_010	21	Self signed certificate
AL_APKaye_005	20	Unknown permission used
AL_APKaye_013	20	Certificate valid more then 30 years
AL_APKaye_011	15	No country in certificate owner
AL_APKaye_015	6	Non-conventional certificate name
AL_APKaye_014	3	Invalid country code in certificate owner

Table 3: Frequent heuristics employed by APKaye in Assemblyline.

for the entire dataset in Table 5. Evidently, commercial detectors also have trouble with these samples.

The assigned labels give a more detailed idea of the type of malware we are dealing with, according to the commercial detectors. The most frequently occurring labels are listed in Table 2. While these detectors do not specifically list many of the families, they seem to concur that many of these are repackaged or otherwise obscured by seemingly legitimate code.[6]

3.3.3 Assemblyline Heuristics. While metadata about the apps is plentiful here, some of the fields, such as domains or package names, are not well suited for adaptation to ArmsRace. A Boolean check or integer value, on the other hand, would work well.

With this in mind, the frequently occurring AL Heuristics in Table 3 provide much inspiration. We discarded AL Heuristics 003, 004, and 005: the app accessing some address is likely to be present in the majority of modern applications, permissions are already accounted for in ArmsRace, and defining a custom permission is not malicious, in and of itself.

The rest of these heuristics, however, relate to inconsistencies in the signing certificate. In fact, 010 is present in every undetectable APK. A self-signed certificate speaks for itself: without a chain of trust, it is impossible to determine if the certificate is actually legitimate. On further inspection, we found that 1,078 of the 1,084 applications (99.45%) in the Google Play dataset are actually self-signed. As a result, this feature was dropped from consideration.

From 013, we were encouraged to review the validity period of each certificate. These results are shown in Table 4. Every single app in the undetectable group has a large validity period for their certificate (at least 25 years), with 19 of the 21 meeting or exceeding 50 years.

[6]See Adware, Autoins (auto-installer), Trojan, in Table 2.

Cert. Start Year	Cert. End Year	Difference in Years	Count
2015	2065	50	14
2012	2062	50	3
2016	2041	25	1
2017	2052	35	1
2011	2066	55	1
2016	2067	51	1

Table 4: Signing certificate dates from Assemblyline.

Google recommends a validity period of 25 years, and requires certificates to expire after 22 October 2033 [7]. Malware authors, desiring to remain malicious for as long as possible, seem to be inclined to sign for longer periods.

3.4 Proposed Certificate Validity Features for ArmsRace

The original implementation of ArmsRace does not consider certificates in any of its features. Addressing this blind spot could be the key to correctly identifying some of these undetectable APKs. Taking inspiration from Assemblyline, we explore the effectiveness of certificate features concerning the period, in years, that the certificate is valid.

The signing certificate file is stored inside of the `META-INF` directory of each APK, so was easily accessible as part of ArmsRace's disassembly process. Using `keytool`, a utility included as part of Java,[7] certificate start and end dates can be extracted as a String and converted into `datetime` Python objects for the purposes of date math. An example output from `keytool` is shown in Figure 5.

Furthermore, incorporating certificates of varying length was an issue that we explored. There are multiple pathways that a generated malware can take in ArmsRace's generation logic. After the call to `gradle`[8], the build system used for Android applications, the newly built APK is unsigned. If DroidChameleon [14] evasion features are selected, the APK is re-built and signed. This can occur multiple times, depending on the selected features. Otherwise, the sample proceeds and is unsigned.

To ensure that every generated app would be signed, we added a static keystore to ArmsRace. In the case that DroidChameleon is not called, the app is signed with this keystore, with a validity period of 50 years. This value was chosen based on the results reported in Table 4.

We wanted to give the generation process the opportunity to choose differing validity periods, and so created three new flags that can be passed to DroidChameleon: `-certShort`, `-certMed`, and `-certLong`. Choosing one of these features will programmatically generate a keystore with a certificate that is valid for a randomly chosen period, as defined by each feature. The periods for short, medium, and long are 20–30 years, 45–55 years, and 100–200 years, respectively. The new certificate is then used by DroidChameleon

[7]https://docs.oracle.com/javase/8/docs/technotes/tools/unix/keytool.html
[8]https://gradle.org/

```
zachary@rebel-alliance:/tmp$ keytool -printcert -file FACEBOOK.RSA
Owner: CN=Facebook Corporation, OU=Facebook, O=Facebook Mobile, L=Palo Alto, ST=CA, C=US
Issuer: CN=Facebook Corporation, OU=Facebook, O=Facebook Mobile, L=Palo Alto, ST=CA, C=US
Serial number: 4a9c4610
Valid from: Mon Aug 31 18:52:16 ADT 2009 until: Sun Sep 25 18:52:16 ADT 2050
Certificate fingerprints:
         MD5:  3F:AD:02:4F:2D:CB:E3:EE:69:3C:96:F3:50:F8:E3:76
         SHA1: 8A:3C:4B:26:2D:72:1A:CD:49:A4:BF:97:D5:21:31:99:C8:6F:A2:B9
         SHA256: E3:F9:E1:E0:CF:99:D0:E5:6A:05:5B:A6:5E:24:1B:33:99:F7:CE:A5:24:32:6B:0C:DD:
Signature algorithm name: MD5withRSA (weak)
```

Figure 5: Keytool output for Facebook certificate.

Dataset	Seen	Unseen	Avg	Min	Max
AMD	24553	0	49.50	19.30	81.97
Drebin	5560	0	66.62	5.00	86.67
F-Droid	1328	11	99.99	99.47	100.00
Genome	1260	0	71.68	57.41	81.97
Google Play	1056	29	99.99	99.77	100.00
UNB Ben	1700	0	99.99	99.84	100.00
UNB Mal	426	0	47.95	0.00	80.00
VS1314	35397	0	53.34	0.00	82.09

Table 5: VirusTotal results on previous datasets.

to sign the modified application, using `jarsigner` another Java-provided utility.[9]

4 EVALUATIONS

In this section, we benchmark the performance of VirusTotal on the datasets from [23]. We also evaluate the proposed ArmsRace features, before tasking MOCDroid, ArmsRace, Assemblyline, and VirusTotal on three new and current VirusShare datasets.

When evaluating classifiers, it is common to report the results in terms of accuracy, precision, and recall (detection rate), as defined below:

$$A = \frac{T_p + T_n}{T_p + T_n + F_p + F_n}$$

$$P = \frac{T_p}{T_p + F_p} \qquad R = \frac{T_p}{T_p + F_n}$$

As we only test these systems on malicious samples, there are no true negatives (T_n) or false positives (F_p), only true positives (T_p) and false negatives (F_n). This causes accuracy to be equal to recall, and precision to be constant at 1. Accordingly, we only report the detection rate in the result tables.

It is important to note that MOCDroid, ArmsRace, and Assemblyline decompile the malicious files as part of their feature extraction process. While this is typically not an issue, a small number of malware can cause the decompilers to fail. In these evaluations, failed decompilation is the cause of any discrepancy between the reported results, and the actual dataset size.

4.1 Evaluations Using VirusTotal On Older Datasets

To put our research in this paper into context, the results in Table 5 are fairly consistent with the results for the detectors reported in [23]. The benign training / testing datasets, Google Play and F-Droid, are marked as harmless by 99.99% of the detectors on average, with even the most 'malicious' sample marked as benign by 99.77% of the detectors.

Genome and Drebin are also well defined, with 71.68% and 66.62% of detectors labelling the sample malicious on average. A notable distinction between these datasets, however, is their minimum detection rate. There is at least one sample in Drebin that is only marked by 5% of the detectors, whereas the least identifiable Genome sample still achieves a score of 57.41%.

Considering AMD, UNB Malware, and VirusShare 2013 / 2014 (VS1314), we can see that the unknown malicious datasets are surprisingly consistent. All three hover around 50%. VS1314 is the most detectable on average at 53.34%, and UNB Malware is the least at 47.95%. AMD is in the middle at 49.50%. This ordering is identical with the results for MOCDroid in [23], and very similar to ArmsRace and Assemblyline in the same work.

Their minimum detection rates show that every sample in AMD is detected by about 20% of the detectors, but UNB Malware and VS_1314 contain at least one sample that is entirely undetectable by every VirusTotal detector. This is reasonable, considering that AMD is an older, established academic dataset, whereas the other two are not as well known.

UNB Benign follows a similar narrative to the training / testing benign datasets, in that every sample is very well detected as being harmless, even in the worst case, averaging 99.99%.

4.2 Evaluations Using Certificate Validity Features

In [23], ArmsRace detection programs that were trained on Drebin and Google Play were consistently rated as the best detectors for unseen malicious samples. Accordingly, this configuration was a natural choice to gauge the performance of the certificate validity features. After re-extracting features from those datasets for training / testing, it was time to evaluate the proposed features. The best generated program achieved a respectable 95.56% overall accuracy on the testing set, a 4.17 percentage point increase over the same configuration in [23]. This was due to an improved benign detection rate (168/180, 93.33%), and correspondingly lower false positive rate (6.66%). Malware detection was nearly unchanged (176/180, 97.78%). This did not translate into an improved detection rate for

[9] https://docs.oracle.com/javase/8/docs/technotes/tools/unix/jarsigner.html

Dataset	Detection Rate
VS16	8689/12694 (68.45)
VS17	6151/9855 (62.42)
VS18	22082/28474 (77.55)

Table 6: ArmsRace (using certificate validity features) results on new VirusShare datasets.

Testing Dataset	Training Datasets	Detection Rate
VS16	Genome / F-Droid	4537/12730 (35.64)
VS16	**Drebin / F-Droid**	**9680/12730 (76.04)**
VS16	Drebin / Google Play	8337/12730 (65.49)
VS16	Genome / Google Play	7345/12730 (57.69)
VS17	Genome / F-Droid	3290/9944 (33.09)
VS17	**Drebin / F-Droid**	**7724/9944 (77.67)**
VS17	Drebin / Google Play	6379/9944 (64.15)
VS17	Genome / Google Play	5314/9944 (53.44)
VS18	Genome / F-Droid	3913/28509 (13.73)
VS18	**Drebin / F-Droid**	**8290/28509 (29.08)**
VS18	Drebin / Google Play	7319/28509 (25.67)
VS18	Genome / Google Play	6214/28509 (21.80)

Table 7: MOCDroid results on new VirusShare datasets.

Testing Dataset	Training Datasets	Detection Rate
VS16	Drebin / F-Droid	7779/12694 (61.28)
VS16	Genome / F-Droid	8214/12694 (64.71)
VS16	**Drebin / Google Play**	**10420/12694 (82.09)**
VS16	Genome / Google Play	6556/12694 (51.65)
VS17	Drebin / F-Droid	4840/9855 (49.11)
VS17	Genome / F-Droid	5566/9855 (56.48)
VS17	**Drebin / Google Play**	**7629/9855 (77.41)**
VS17	Genome / Google Play	4583/9855 (46.50)
VS18	Drebin / F-Droid	19145/28474 (67.24)
VS18	Genome / F-Droid	20100/28474 (70.59)
VS18	**Drebin / Google Play**	**23862/28474 (83.80)**
VS18	Genome / Google Play	18597/28474 (65.31)

Table 8: ArmsRace results on new VirusShare datasets.

Dataset	Detection Rate
VS16	6712/12306 (54.54)
VS17	4878/9875 (49.40)
VS18	22802/28523 (79.94)

Table 9: Assemblyline results on new VirusShare datasets.

the undetectable malware, however. All 21 are predicted as benign apps by the aforementioned program.

This lacklustre performance also extended to the new VirusShare datasets. The aforementioned program scored lower than its corresponding configuration for each of VS16, VS17, and VS18, as shown in Table 6. The differences are significant compared to the baseline results in Table 8. VS16 and VS17 drop 13.64 and 14.99 percent points, respectively. VS18 is closer, but still loses 6.25 points. As a result, we did not evaluate any further configurations.

4.3 Evaluations Using New Datasets

4.3.1 MOCDroid. As in [23], we chose the best detectors based on the possible combinations of training datasets as follows: Genome / F-Droid (0.95, 180), Drebin / F-Droid (0.96, 120), Drebin / Google Play (0.96, 120), and Genome / Google Play (0.97, 140), where the first number in parentheses represents the sparse parameter and the second number represents the cluster size.

The results of running the best MOCDroid detectors on the new VirusShare datasets can be seen in Table 7. The best detectors for VS16 and VS17 are low compared to VS1314 in [23], but still reasonably competitive. The staggering difference is the variability of the detection rate depending on the training datasets. VS16 and VS17 each have a standard deviation of approximately 15, owing to the minimum and maximum detectors for each ranging from the low 30s to the high 70s.

VS18 is another story altogether. It is much more 'stable', in that it avoids the variability of the previous years, with a standard deviation of 5.72. However, the best MOCDroid models completely collapse on this dataset, peaking at a dismal 29%.

4.3.2 ArmsRace. The benchmarks for ArmsRace on VS16, VS17, and VS18 are enumerated in Table 8. The best detectors are much more successful than MOCDroid, averaging a detection rate of 81%, but still suffer from variability based on the training datasets, with an average standard deviation of 10.

The ordering of the datasets in terms of difficulty deviates from MOCDroid, where VS18 and VS16 are the easiest, with VS17 lagging slightly behind.

An interesting contrast can be seen between the results of MOCDroid and ArmsRace on VS18. While MOCDroid collapses, ArmsRace actually achieves its highest detection rate across the three new datasets. This is true regardless of the training datasets. Genome, whose samples are nearing a decade of life, still produces detectors that correctly classify two thirds of the VS18 dataset. This demonstrates the resiliency of co-evolutionary frameworks.

4.3.3 Assemblyline. Assemblyline's detection rates for the new datasets are summarized in Table 9. The system does poorly on VS16 and VS17, around 50%, but shockingly well on VS18 (80%).

As with ArmsRace, Assemblyline has the most trouble with VS17, and the best results on VS18.

Compared to the unknown datasets in [23], the new VirusShare sets drop over 10 points in their detection rate mean, from 73% to 61%. This dive would be even more substantial without the inclusion of VS18. This suggests that Android malware authors have increased the evasiveness of their malware between 2014 and 2016, the time period between the datasets.

Dataset	Seen	Unseen	Avg	Min	Max
VS16	12848	0	36.27	0.0	82.81
VS17	10152	0	27.23	0.0	84.13
VS18	28630	2	37.17	0.0	80.00

Table 10: VirusTotal results on new VirusShare datasets.

4.3.4 VirusTotal. Finally, we can observe the average, minimum, and maximum detection rates for VirusTotal systems in Table 10. We can see that all three datasets are significantly more evasive than the datasets in Table 5. UNB Malware, the most difficult Android dataset in previous evaluations, has an average detection rate of 48%, over 20 points higher than the least detectable VirusShare dataset: VS17.

All of the new datasets also contain samples that are undetectable by VirusTotal systems. In fact, VS18 actually contains two samples that were completely unseen by VirusTotal at time of evaluation, probably due to the recent release time of these datasets.

As with ArmsRace and Assemblyline, VS17 is the most difficult dataset, with an average detection rate of 27%. VS16 and VS18 are in near lockstep 10 points higher. As mentioned in sub-subsection 4.3.3, the difficulty in identifying these samples as malicious could be a result of increased focus on evasive techniques on the part of malicious actors. It is also a testament to the effectiveness of the co-evolutionary nature of ArmsRace that so many commercial detectors fail to detect many of the VirusShare samples.

5 CONCLUSION AND FUTURE WORK

In this paper, we investigated evolutionary Android malware solutions, with a focus on an artificial arms race system. Our major contribution in this work is the testing of four Android malware solutions on three new datasets. In addition, we explore the development and effectiveness of the certificate-related features to extend the best performing system.

ArmsRace is consistently successful across all three new datasets, averaging 81%. MOCDroid, Assemblyline, and VirusTotal all have signficant difficulty, each dropping below 40% average accuracy on at least one of the new datasets. As in [23], these results agree with the hypothesis in [20] that co-evolutionary (arms race) solutions are effective frameworks with which to evolve and analyze malware detection models.

Looking at the performance of MOCDroid and ArmsRace on the new datasets, it becomes apparent that Genome is very much out-of-date as a training dataset. The performance jumps on the new VirusShare datasets when using Drebin instead of Genome are significant. The MOCDroid classifiers are better across the board, with gains ranging from 4 percent points to 40. ArmsRace is overall more successful, so these gains are tempered, but still significant.

As expected, older datasets are detected with much higher accuracy by the VirusTotal engines. While UNB Malware was previously considered to be the most difficult Android dataset, VS16, VS18, and especially VS17 are now the high bar in our investigations.

The certificate features, relating to validity period of the APK signing certificate, do not seem to positively affect detectability of the newest Android datasets. They do, however, lead to an increase in the detection of benign applications. More analysis and evaluations along this line is an area for future work.

Future work could include consideration of dynamically loaded malware in the arms race framework. Further study is necessary, given that these malware may well obscure the static analysis processes that ArmsRace, MOCDroid, and Assemblyline employ.

ACKNOWLEDGEMENTS

This research is supported partly by the Natural Science and Engineering Research Council of Canada (NSERC). This research is conducted as part of the Dalhousie NIMS Lab at: https://projects.cs.dal.ca/projectx/.

REFERENCES

[1] Daniel Arp, Michael Spreitzenbarth, Malte Huebner, Hugo Gascon, and Konrad Rieck. 2014. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In *21st Annual Network and Distributed System Security Symposium (NDSS)*.

[2] Dieter Bonn. 2018. Android at 10: the worldâĂŹs most dominant technology. (Sept. 2018). https://www.theverge.com/2018/9/26/17903788/google-android-history-dominance-marketshare-apple

[3] R. Bronfman-Nadas, N. Zincir-Heywood, and J. T. Jacobs. 2018. An Artificial Arms Race: Could it Improve Mobile Malware Detectors?. In *Network Traffic Measurement and Analysis (TMA) Conference*.

[4] E. David and N. Netanyahu. 2015. DeepSign: Deep Learning for Automatic Malware Signature Generation and Classification. In *International Joint Conference on Neural Networks (IJCNN)*. Killarney, Ireland, 1–8.

[5] F-Droid Limited. [n. d.]. F-Droid. ([n. d.]). https://f-droid.org/

[6] Samuel Gibbs. 2016. Mobile web browsing overtakes desktop for the first time. (Nov. 2016). https://www.theguardian.com/technology/2016/nov/02/mobile-web-browsing-desktop-smartphones-tablets

[7] Google Developers. 2019. Sign your app - Android Developers. (2019). https://developer.android.com/studio/publish/app-signing#considerations

[8] Google LLC. [n. d.]. Google Play. ([n. d.]). https://play.google.com/store

[9] Arash Habibi Lashkari, Andi Fitriah A.Kadir, Laya Taheri, and Ali A. Ghorbani. 2018. Toward Developing a Systematic Approach to Generate Benchmark Android Malware Datasets and Classification. In *52nd IEEE International Carnahan Conference on Security Technology (ICCST)*.

[10] Liu Liu, Bao-sheng Wang, Bo Yu, and Qiu-xi Zhong. 2017. Automatic malware classification and new malware detection using machine learning. *Frontiers of Information Technology & Electronic Engineering* 18, 9 (01 Sep 2017), 1336–1347. https://doi.org/10.1631/FITEE.1601325

[11] A. Martín, H. Menéndez, and D. Camacho. 2017. MOCDroid: Multi-objective evolutionary classifier for Android malware detection. *Soft Computing* 21, 24 (2017), 7405–7415.

[12] G. Meng, Y. Xue, C. Mahinthan, A. Narayanan, Y. Liu, J. Zhang, and T. Chen. 2016. Mystique: Evolving Android Malware for Auditing Anti-Malware Tools. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security (ASIA CCS '16)*. ACM, New York, NY, USA, 365–376. https://doi.org/10.1145/2897845.2897856

[13] N. Milosevic, A. Dehghantanha, and K.-K.R. Choo. 2017. Machine learning aided Android malware classification. *Computers & Electrical Engineering* 61 (July 2017), 266–274. http://eprints.whiterose.ac.uk/128366/

[14] Vaibhav Rastogi, Yan Chen, and Xuxian Jiang. 2013. DroidChameleon: Evaluating Android Anti-malware Against Transformation Attacks. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security (ASIA CCS '13)*. ACM, New York, NY, USA, 329–334. https://doi.org/10.1145/2484313.2484355

[15] Canadian Centre for Cyber Security. [n. d.]. Assemblyline. ([n. d.]). https://cyber.gc.ca/en/assemblyline

[16] J-Michael Roberts. [n. d.]. VirusShare. ([n. d.]). https://virusshare.com

[17] J-Michael Roberts. 2019. VirusShare on Twitter: "Just posted 7 custom torrents: Collections of all Android APK files for each year 2012 through 2018. 116,196 samples totaling 598GB. Users login for links. #malware". (2019). https://twitter.com/VXShare/status/1138895363727925248

[18] S. Sen, E. Aydoğan, and A. I. Aysan. 2018. Coevolution of Mobile Malware and Anti-Malware. *IEEE Transactions on Information Forensics and Security* 13, 10 (Oct. 2018), 2563–2574. https://doi.org/10.1109/TIFS.2018.2824250

[19] Michael Spreitzenbarth, Florian Echtler, Thomas Schreck, Felix C. Freling, and Johannes Hoffmann. 2013. MobileSandbox: Looking Deeper into Android Applications. In *28th International ACM Symposium on Applied Computing (SAC)*.

[20] Daniele Ucci, Leonardo Aniello, and Roberto Baldoni. 2019. Survey of machine learning techniques for malware analysis. *Computers & Security* 81 (2019), 123–147. https://doi.org/10.1016/j.cose.2018.11.001

[21] VirusTotal. 2019. How it works - VirusTotal. (2019). https://support.virustotal.com/hc/en-us/articles/115002126889-How-it-works

[22] Fengguo Wei, Yuping Li, Sankardas Roy, Xinming Ou, and Wu Zhou. 2017. Deep Ground Truth Analysis of Current Android Malware. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA'17)*. Springer, Bonn, Germany, 252–276.

[23] Zachary Wilkins and Nur Zincir-Heywood. 2019. Darwinian Malware Detectors: A Comparison of Evolutionary Solutions to Android Malware. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '19)*. ACM, New York, NY, USA, 1651–1658. https://doi.org/10.1145/3319619.3326818

[24] Y. Xue, G. Meng, Y. Liu, T. H. Tan, H. Chen, J. Sun, and J. Zhang. 2017. Auditing Anti-Malware Tools by Evolving Android Malware and Dynamic Loading Technique. *IEEE Transactions on Information Forensics and Security* 12, 7 (July 2017), 1529–1544.

[25] Y. Zhou and X. Jiang. 2012. Dissecting Android Malware: Characterization and Evolution. In *2012 IEEE Symposium on Security and Privacy*. 95–109. https://doi.org/10.1109/SP.2012.16