AndroParse - An Android Feature Extraction Framework and Dataset

Robert Schmicker, Frank Breitinger(✉), and Ibrahim Baggili

Cyber Forensics Research and Education Group (UNHcFREG),
Tagliatela College of Engineering, University of New Haven,
West Haven, CT 06516, USA
rschm2@unh.newhaven.edu, {FBreitinger,IBaggili}@newhaven.edu

Abstract. Android malware has become a major challenge. As a consequence, practitioners and researchers spend a significant time analyzing Android applications (APK). A common procedure (especially for data scientists) is to extract features such as permissions, APIs or strings which can then be analyzed. Current state of the art tools have three major issues: (1) a single tool cannot extract all the significant features used by scientists and practitioners (2) Current tools are not designed to be extensible and (3) Existing parsers can be timely as they are not runtime efficient or scalable. Therefore, this work presents *AndroParse* which is an open-source Android parser written in Golang that currently extracts the four most common features: Permissions, APIs, Strings and Intents. AndroParse outputs JSON files as they can easily be used by most major programming languages. Constructing the parser allowed us to create an extensive feature dataset which can be accessed by our independent REST API. Our dataset currently has 67,703 benign and 46,683 malicious APK samples.

Keywords: AndroParse · Android · Malware · Dataset · Features Framework

1 Introduction

Without a doubt, smartphone malware is on the rise. As a consequence, researchers and industry spend significant resources to improve malware detection techniques, e.g., by manually analyzing applications during forensic investigations or applying machine learning techniques.

Regardless of how a practitioner analyzes applications, there are usually two essential steps. First, one acquires a single malware sample/a sample dataset; when it comes to machine learning datasets are essential. Second, one will have to parse information to gain insight into the application(s). An overarching step by step workflow for machine learning approaches is depicted in Fig. 1 which coincides with the process observed in other works [9].

F. Breitinger and I. Baggili (Eds.): ICDF2C 2018, LNICST 259, pp. 66–88, 2019.
https://doi.org/10.1007/978-3-030-05487-8_4

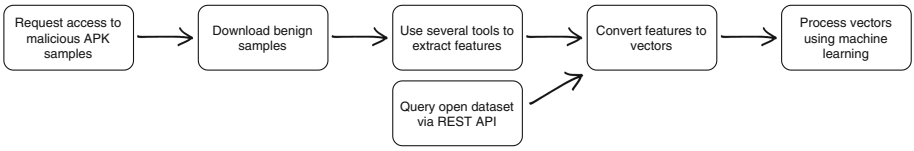

Fig. 1. Current work flow for machine learning approaches.

Malicious APK acquisition is often kept private due to ethical restrictions of freely sharing malware on the Internet; Table 2 shows some available datasets. For those that are private, users can often request access through a review process.

Benign APK acquisition involves downloading samples through public websites [4,5] such as Google Play (Google's application store).

Feature extraction defines the step of extracting relevant information from the APKs. This may include a separate library for each sought after feature. Since these libraries are often written in varying languages (e.g., C++, Java, or Python) this requires the user to be well versed in many languages, adding another layer of complexity.

Features to vector conversion transforms the raw feature data into vectors for a machine learning algorithm. One will typically write a script to massage the data into the format required for their algorithm.

Processing vectors is the final stage and allows a data scientist to test the detection rate of their algorithm.

From a forensic practitioner's perspective, the bulk of the work is related to reverse engineering the applications where one usually starts by extracting features to understand the application's code (e.g., looking for strings in the APK like IPs, hashes or URLs).

While these procedures are well established, there are some drawbacks. Downloading benign applications/requesting access to malicious applications can be time consuming, e.g., one may be tasked with writing a crawler or contacting website administrators for access to a bulk download. Sharing malware directly has downsides as well [11] and even though a review process exists, there is no guarantee that samples will only be used for research.

In this paper we present *AndroParse*[1], as well as, a freely accessible Android feature dataset which can be easily used by practitioners; it allows to download features of over 100,000 applications (benign and malicious)[2]. Specifically, this paper has two major *contributions*:

1. AndroParse is the first open source and extensible Android APK parser that allows users to quickly access features/artifacts of interest. As it is expandable, AndroParse provides a framework for plugins that can accommodate for new features/artifacts in various programing languages.

[1] https://github.com/rschmicker/AndroParse (last accessed 13-April-2018).
[2] https://64.251.61.74/ (last accessed 13-April-2018).

2. We provide a centralized, online dataset of Android APK features for examiners and data scientists[3] that can be accessed and downloaded through our web interface. Currently AndroParse's open dataset holds a total of 114,386 unique APKs - 67,703 benign and 46,683 malicious. This count is tentative as the dataset grows in size every day through the use of automated webcrawlers.

In our initial version, AndroParse supports four major features; we chose these features after analyzing state-of-the-art research: as they were the most common ones in scientific literature. To extract the features, we constructed a multi-threaded Golang plugin framework that utilizes existing applications (e.g., Android Asset Packaging Tool). This modular design allows anyone to add new feature extraction plugins if needed. Data scientists and forensic practitioners can access our platform to download the extracted features by querying a REST API and receive them in a JSON format. Note, feature extraction is performed on our server thus it consumes minimal computational resources from the user.

The rest of the paper is structured as follows: Sect. 2 summarizes existing tools for extracting information from an APK file, as well as, Android datasets and services. The extraction process is explained in granular detail in Sect. 3 which includes the implementation, tools used, features used, extending to new features, and extraction process. Statistics and an overview of the open dataset provided is presented in Sect. 4, in addition to, querying the parsed APKs contained in the open dataset in Sect. 4.3. This leads to an evaluation of AndroParse in Sect. 5. Lastly, we provide limitations, as well as, future work.

2 Background and Related Work

Given our two major contributions, we separated this section into *Feature extraction and decompilation tools* (Sect. 2.1) where we summarize existing frameworks and tools and *Malware samples and services* (Sect. 2.2) which summarizes the existing datasets we found. For a more comprehensive list of Android security resources, one may visit Ashish Bhatia's Github [12].

2.1 Feature Extraction and Decompilation Tools

The following tools have been developed to ease the process of extracting desired features from Android applications.

Android Asset Packaging Tool (AAPT, [14]*)* is part of Google's Android SDK and has been utilized by several researchers. This command-line tool decodes and parses the AndroidManifest.xml and allows users to query certain information about an APK. AAPT has been used "[...]to extract and decrypt the data from the AndroidManifest.xml file[...]" to access the APKs' permissions [31]. Written

[3] A prominent example that these services are valuable for the community is the UCI Machine Learning Repository [25] which includes a multitude of data and repositories and is frequently referenced in literature.

in C++, it is a fast tool as it provides the AndroidManifest.xml without having to decode the entire APK file.

apk_parse [36] is a Python library written to parse information from the AndroidManifest.xml. Unfortunately, it limits itself to the manifest and meta data of an APK for feature extraction. A more comprehensive tool is *Androguard* [13] which is an open source Python tool for extracting features from an APK's AndroidManifest.xml and DEX files. For instance, it has been used to test Android APK code obfuscation techniques [15]. Although *Androguard* is extensive and capable, it is time consuming to process an APK. In addition, it does not parse intents from an APK used by several works (see Table 4). Rapid Android Parser for Investigating DEX files is an open source Java based library for parsing DEX files [42]. It minimizes the time it takes to parse an APK by having an in-memory representation of the data that allows queries. The problem is that it is limited to strings and APIs and scientists still need to understand the structure/APIs in order to query it. Besides the actual malware dataset (mentioned in the previous section), *Drebin* provides "all features extracted from each of the 123,453 benign applications and 5,560 malicious applications" [7,38]. However, the *Drebin* feature extraction tool seems to be closed source. This hinders open performance reviews and comparison to open source tools.

While the previous tools focused on feature extraction, *APKTool* [40] disassembles the APK file into smali form as well as decompresses the AndroidManifest.xml. Smali files are text files (one per java class) which are simpler to understand than DEX files. However, these files then need to be parsed again in order to be used by data scientists [29].

2.2 Malware Samples and Services

While searching for malware samples, we identified that there were two kinds of sets which we will refer to as *services* and *sample sets*.

Malware services are online applications that possess or allow the uploading of samples but only share secondary information. For instance, these services examine an APK file and detect if it is malicious or provide other information such as extracted strings or permissions. `VirusTotal.com` is one popular example [37]. Although convenient, VirusTotal has a major limitation of being signature based and therefore it cannot be fully aware of the intents of an application. Payload Security [27] does offer an online searchable dataset of malware. Although highly informative, it is limited to metadata, permissions, and extracted strings for a given malware sample but does not include APIs and other strings. Other sources such as AndroTotal [21] and NVISO APK Scan [23] exist but the user must first have the APK samples to analyze. To sum it up, these third party services are convenient for small applications in small quantities and are not suitable for large-scale detailed APK file analysis. Secondly, AndroTotal and NVISO APK Scan offer some features to be viewed online but they do not offer a download option of the features. Payload Security offers an API except the user must sign up for access and is given a quota per API key.

Malware sample sets are repositories which are available for download; an overview is shown in Table 2. Most datasets are kept password protected and only through a review process can a researcher gain access.

One frequently utilized dataset is Drebin [7]. This dataset consists of 5,560 samples from 179 different malware families collected from 08/2010 to 10/2012 and is available for researchers in academia as well as industry after 'registration' (sending an email). Another example dataset was the Malware Genome Project[4] [43]. However, according to the website this dataset is no longer being maintained. Contagio Mobile [26] contains a smaller amount of APKs, but are referenced extensively in research articles. Works have used the repository to analyze the effectiveness of permissions as the sole feature for malware detection [32]. Das Malwerk [35] and theZoo [24] are examples of datasets that are open to the public. They not only contain Android malware, but Windows and OS X executables as well. The malware samples vary from cryptolockers to ransomeware, and trojans.

3 AndroParse

AndroParse is a feature extraction framework that is developed for digital forensic practitioners and data scientists. It allows users to parse features out of Android applications which can then be manually analyzed (e.g., using elasticsearch) or used as input for machine learning approaches. A complete overview is depicted in Fig. 2.

Although popular tools for Android APK reverse engineering have been previously written in Python [13] and Java [1], AndroParse is written in *Golang*. We chose Golang as it provides authentic multi-threading, unlike Python[5], and a runtime plugin interface, unlike Java. Both of these programming language features are heavily relied upon in the framework. A detailed comparison is provided in Table 1.

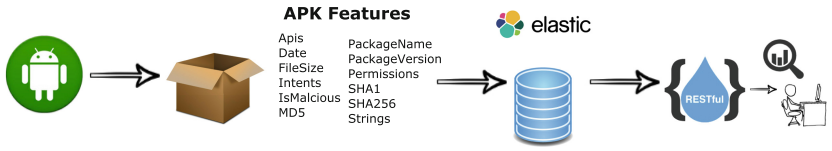

Fig. 2. Extraction & querying workflow

3.1 Installation and Usage

AndroParse is a command-line driven tool that parses four different features from APK files and outputs results in a commonly accepted JSON format. Before running it, it requires some preparation:

[4] http://www.malgenomeproject.org (last accessed 13-April-2018).
[5] https://wiki.python.org/moin/GlobalInterpreterLock (last accessed 13-April-2018).

Table 1. Reverse engineering tool comparison.

Tool	AndroParse	php_apk_parser [34]	Androguard [13]	Apktool [1]
Open Source	✓	✓	✓	✓
License	GPL-3.0	None	Apache-2.0	Apache-2.0
Expansion	✓			
Language	Golang	PHP	Python	Java
Manifest	✓	✓	✓	✓
Dex	✓		✓	✓
Export Format	JSON	XML	Python	XML/Smali

Dependencies. While some dependencies are included in the repository, others must be downloaded and installed manually. Particularly, our implementation requires the RAPID JAR [42] (included), Glide package manager, and Google's AAPT [14].

The Glide package manager provides an easy to use interface for installing Golang dependencies. Installation instructions for Glide can be found on their GitHub repository[6]. Once Glide has been installed, users can download the AndroParse source code and put it in any directory. Next, the user must run the command `make update && make configure` to download, install, and prepare all Golang related dependencies AndroParse requires to compile. The second manual dependency, AAPT, can be installed as a system wide package in Debian, CentOS, and Mac OSX based distributions of UNIX/BSD. Once dependencies are installed, AndroParse can be compiled and executed using `make && androparse`.

Command-Line Options. Before running the application, the user is required to create a YAML configuration file:

```
apkDir: "/home/myuser/apks"
codeDir: "/home/myuser/src/github.com/AndroParse/androparse"
outputDir: "/home/myuser/output"
vtapikey: "My VirusTotal API Key"
```

The field `apkDir` contains the directory of the user's APK dataset. `codeDir` provides the source code directory to access the RAPID JAR file, as well as, feature extraction plugins at runtime. `outputDir` specifies the directory the user would like to store the resulting JSON for parsed APKs. Lastly, `vtapikey` is optional, however, is required under the condition that the user requests each sample to be validated with VirusTotal using the `vt` flag. Then our implementation can be executed by

[6] https://github.com/Masterminds/glide (last accessed 13-April-2018).

```
androparse -config ~/myconfig.yaml -vt -clean -append -parser Permissions
```

The `config` flag is the only required command line option as it provides a relative or absolute path to a user's configuration file. `vt` specifies that the user wants to validate the APK samples with VirusTotal (this requires an API key in the configuration file). In the absence of this flag and should the user execute the *IsMalicious* plugin, the user must separate their APKs into `benign/` and `malicious/` directories. The flag, `clean`, renames all files stored in the targeted APK directory to their SHA256 values. This removes any duplicates from the dataset and reduces disk usage (note, the original file name is not captured as often APKs were renamed beforehand, e.g., most of the malicious datasets). `append` allows users to add a new feature into existing JSON output files from a previous extraction run, or skip over already parsed APKs. Lastly, `parser` permits the user to specify which feature extraction plugin(s) to run explicitly. In the absence of this flag, all feature extraction plugins are ran.

3.2 Extracting and Adding New Features

The following paragraphs highlight how AndroParse extracts the features from every APK file:

Deduplication (a.k.a. clean). As a first step, AndroParse will rename every APK to its corresponding SHA256 hash value. This mitigates any duplicate APKs in the dataset and decreases the necessary runtime of the extraction process.

Feature Extraction. To extract various features, we utilize existing tools:

MD5, SHA1, SHA256, Date, File Size are generated using Golang's standard libraries. The file size of an APK is stored in bytes. Furthermore, we capture the timestamp (format `"yyyy-mm-dd HH:MM:SS"`) when the APK is processed which allows to have standardized sets, e.g., the detection rates can be compared by the standardized corpus before 'date'.

Permissions, Intents, Package Name and Version are extracted using Google's *Android Asset Packaging Tool* (AAPT, [14]). AAPT can extract an APK's AndroidManifest.xml without having to decompress an entire APK's content. The feature extraction plugins *Permissions, Intents, PackageName, and PackageVersion* each call AAPT to decompress the AndroidManifest.xml file to parse a given feature.

APIs, Strings are analyzed by the RAPID library [42]. Therefore, we invoke RAPID's Java jar library through an operating system *exec* call[7].

[7] This portion of code must be performed sequentially as there is a low-level JVM memory error when multiple threads access the library at once.

Adding New Features/Extending AndroParse. One of the key strengths of AndroParse is extensibility (adding new feature extraction methods) which is implemented using Golang's runtime plugin interface[8]. The interface provides three main benefits to a developer creating new plugins for feature extraction. (1) The developer does not need to have a working knowledge of the framework and can purely focus on extracting desired features, (2) It does not require recompilation of the entire framework, and (3) It allows plugins to be written in other programing languages such as C and C++[9]. More details can be found in the documentation[10]. The remainder of this section details the development of a plugin.

The sample structure for plugins is highlighted in Listing 1. Each plugin is considered as its own package and therefore must label itself as *main* (line 1). Furthermore, the plugin must import AndroParse utils (line 4) package as it contains a necessary configuration data structure so that the plugin can access information from the included YAML file on execution. The actual functionality is implemented in three functions:

1. `NeedLock()` returns true or false depending on if the parser needs to be locked from other threads accessing the same parser at the same time. For instance, the RADIP JAR library currently cannot run in multiple threads and therefore this function should return true (See Sect. 6 for more details).
2. `GetKey()` only returns a key (string) that indexes the given plugin's value in the resulting JSON output for a given APK. A user may choose this to be the plugin's name for example.
3. `GetValue(string, utils.ConfigData)` accepts a path to an APK (the framework iterates over each APK) and a struct containing configuration data from the user created YAML file (See Sect. 3.1). Note, the first return type is `interface{}`[11] which means the plugin can return any type and the AndroParse framework will correctly handle its type to be displayed in the resulting JSON file. In addition, the plugin must also return an error value should an error occur or *nil* when all has processed correctly.

Once completed, the parser needs to be stored in the folder `./androparse/plugins/MyPluginName/`. Following this, the plugin's `.go` file needs to be added in the Makefile (`./androparse/plugins/Makefile`):

```
PLUGINS := Apis/Apis.go Intents/Intents.go [...] MyPluginName/MyPluginName.go
```

Subsequently, the developer can compile their plugin using the `make` command, and finally, invoke their plugin by executing the command below where `myPluginName` is the file name:

```
androparse -config ~/myconfig.yaml -parser MyPluginName
```

[8] https://golang.org/pkg/plugin/ (last accessed 13-April-2018).

[9] One can use any language as long as the code can be compiled into a *shared object* file.

[10] https://github.com/rschmicker/AndroParse/wiki/Develop-Plugins (last accessed 13-April-2018).

[11] https://golang.org/doc/effective_go.html#interfaces (last accessed 13-April-2018).

```go
package main

import (
    "AndroParse/androparse/utils"
    "os"
)

func NeedLock() bool { return false }

func GetKey() string { return "FileSize" }

func GetValue(path string, config utils.ConfigData)
            (interface{}, error) {
    file, err := os.Open(path)
    if err != nil {
        return nil, err
    }
    fi, err := file.Stat()
    if err != nil {
        return nil, err
    }
    return fi.Size(), nil
}
```

Listing 1. Example AndroParse Plugin.

3.3 Storage Schematic/Accessing Features

The features of each APK are stored in a JSON file. An example of this output is shown in Appendix A Listing 3. We decided for JSON due to the widespread support across most programming languages and its compatibility with may tools, e.g., Elasticsearch (details below). An example use case of using AndroParse's JSON output can be seen in our repository under `analysis/train_oa.py`. The script showcases several machine learning algorithms compared against each other using the permissions of an APK as a feature vector.

Elasticsearch is a textual indexing engine used for searching for the features by our backend which can be used for JSON files. Elasticsearch requires a *mapping* to be used which "defines how a [JSON] document, and the fields it contains, are stored and indexed"[12]. The mapping used by AndroParse can be seen in Appendix A Listing 5. As shown, this JSON structure identifies which data type should be used for each field (it can be updated to accommodate a new feature). Once the mapping of the dataset is updated, a new feature can be appended onto existing documents. Using Elasticsearch in our backend provides AndroParse a scalable solution not only as the number of APKs grows, but also as the number of new features increases.

[12] https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping.html (last accessed 13-April-2018).

Verification of Elasticsearch (See Sect. 5.1) verifies the process of extracting the features from an APK, however, we still found it necessary to verify that the JSON documents Elasticsearch indexes are not modified in any way when queried. To verify Elasticsearch's output, a query is performed on a given APK downloading all of its key value pairs. Then for every value contained in the given APK's JSON file produced by AndroParse, it is searched and matched to Elasticsearch's output. After 100 successful trials, it was concluded that the data is valid.

4 Dataset and Parsed Features

We will summarize the dataset we collected, as well as, motivate the features that are currently available.

4.1 Dataset Contents

In order to provide a comprehensive feature dataset for researchers, we collected available malware datasets and downloaded benign samples as listed in Table 2 (note, a few malware datasets contained duplicates). While collecting the samples, we found that the average size per application differs; malicious applications seem to be smaller than benign (See Table 3).

Table 2. Malicious APKs in AndroParse's dataset.

Source	# of APKs	Private	Reference
AMD	24,553	✓	[39]
PRAGuard	10,479	✓	[22]
Third Party Stores	9,587		
Drebin	5,560	✓	[7]
Contagio Mobile	818		[26]
theZoo	100		[24]
GitHub	73		[12]
Das Malwerk	55		[35]
Total Before Dedup	51,270		
Total After Dedup	46,683		

Table 3. AndroParse dataset statistics.

	APK count	APK total size	APK avg. size
Benign	67,703	583.67 GB	8.28 MB
Malicious	46,683	70.26 GB	1.54 MB
Total	114,386	653.93 GB	5.85 MB

The count of benign APKs is rapidly changing due to routine web-scrappers that continuously crawl third party websites such as *apk-downloaders.com, apkapps.com, apkfiles.com, apkleecher.com, apkmirror.com, fdroid.org, slideme.org* and the Google Play App store. These web-scrappers are public and open to the community to use in our source code repository under `./webscrappers/`. Note, third party sites constantly change their HTML structure and one may have to adjust them. Furthermore, we would like to ask the community to consider contributing their samples our repository.

4.2 Identifying Relevant Features

To gain insight into commonly used features for Android malware detection, we analyzed state-of-the-art literature by searching databases for key terms like 'Android, Malware, and/or Machine Learning'. We then selected the 15 articles listed in Appendix A Table 7 because they were the most recent (2012 or newer) and are frequently cited. A summary of the utilized features is provided in Table 4.

Table 4. Number of features used across papers.

Feature	# of occurrences
Permissions	13
APIs	11
Strings	7
Intents	6
Components	3
Graphs	1
Signatures	1
Meta Data	1
Opcodes	1

Note, the total number of features exceeds the total number of analyzed articles as most references use several features. For example, a single article could use hardware/application components, permissions, intents, APIs, and network addresses (strings) [7]. Each of these are counted individually leading to a sum of features larger than the amount of papers.

Our findings are similar to [16] who analyzed 100 papers in the Android malware domain and permissions were also the most referenced static feature followed by APIs as the second most used feature (e.g., APIs are used to discover any use of network connectivity, encryption, or obfuscation [41]). Our third feature is strings which can include label names, text shown in the application but also contain URLs, phone numbers, and IP addresses. Lastly, intents provide an effective method for understanding how an APK may operate. They

are often times combined with permissions for accurate malware detection [10]. Each feature is expressed broadly. For example, *APIs* includes the use of APIs in general as well as special API's, network API's, encryption API's, etc. Since all APIs in a given APK are extracted, any of these *sub-features* can be utilized. The same applies to *Strings*, i.e., network addresses, native system commands, phone numbers, etc.

Based on these findings, AndroParse currently supports parsing the top four features: (1) Permissions, (2) APIs, (3) Strings, and (4) Intents. As discussed earlier, the framework can easily be extended (discussed in Sect. 3.2).

4.3 Front End for Accessing AndroParse Sample Feature Dataset

To access the dataset, we developed a REST API which can be queried using three GET parameters and allows users to download needed features:

fields= returns only the features specified. If left empty or missing, all features of each APK are returned.
to= returns APKs up until the provided timestamp in the form: `yyyy-mm-ddTHH:MM:SS`.
from= return APKs starting from the provided timestamp onward in the form: `yyyy-mm-ddTHH:MM:SS`.
/**all** returns the entire dataset AndroParse has to offer at the time of querying.

Once queried, a ZIP file is created and stored into a directory shared by an anonymous read-only FTP server. Due to the potential for a large query, using an FTP server is more flexible (e.g., the end user can resume a download in case of connectivity problems; no additional query is needed). To free space, the server will delete queries after a certain time. Once the desired information is downloaded and extracted (JSON file), a user can manipulate the format to be used in a wide range of applications and languages such as WEKA [19] or Python.

Remark: Since the dataset is constantly growing, it is important to use the *to* and *from* GET parameters. Thus, future researchers can compare the their malware detection technique with previous approaches. For example, data scientist A uses the dataset prior to (to=)2018-03-03. This allows data scientist B to download the same set later even though the entire dataset may haven grown.

4.4 Accessing the Server

Our server has the IP 64.251.61.74 and is using a self-signed certificate with the SHA1 hash-value `7FE9AE1503BBA19F248E203F74A38D80DC849588`. You can access the server's REST API on port 443, e.g., https://64.251.61.74/api?fields=Permissions. To access the JSON file, connect Anonymous to the same IP on port 21.

5 Evaluation

In this section, we evaluate the forensic soundness, performance of the AndroParse framework, and the front end.

5.1 Verification of the Feature Extraction Process

As mentioned in Sect. 3.2 we rely on existing tools/implementations to extract features. These tools are well documented, have been previously tested by other works and are found to be accurate in their extraction process. AndroParse does however place some overhead on the tools used to structure the data properly for JSON.

To verify the integrity of the feature extraction process, Golang's built in command *go test* was used to perform unit tests on each of the feature extraction plugins. Using the APK *Facebook Lite* version 70.0.0.9.116 as a test APK file, each of the related plugin's expected values were manually extracted using the same underlying tool AndroParse uses. For example, `aapt` was executed to extract permissions which then were placed in a Golang unit test. This process was repeated for each of the other provided feature extraction plugins. Finally, each unit test was constructed to test for its given feature. *go test* successfully showed that each of the plugins created extracted its given feature and matched the expected value. For further description of how we constructed and executed our unit tests, please review our documentation on our wiki[13].

5.2 Application (APK) Validation

When receiving APKs from other researchers/sources, an APK is only labeled as malicious or benign by word of mouth. To further verify a given APK as malicious or benign we compared the hash of each APK to the VirusTotal [37] API. Using the VirusTotal service, we were able to more accurately identify APKs that are malicious or benign.

Interestingly we discovered that 761 previously labeled malicious samples were found to be benign. On the other hand, 9,587 benign samples were found to be malicious. This finding of mislabeled benign APKs parallels a previous study [44] where the authors discovered that third party Android APK stores often host malware.

In addition, we handled reducing potential false positives by only relabeling an application from benign to malicious if more than 4 anti-virus scanners (provided by VirusTotal) found an application to be malicious. Moreover, we relabeled a malicious application to benign if none of the VirusTotal anti-virus scanners reported a virus. Ultimately, the results VirusTotal provides are taken with a grain of salt, however, we feel this labeling technique to be more accurate than labeling by word of mouth.

[13] https://github.com/rschmicker/AndroParse/wiki/Develop-Plugins (last accessed 13-April-2018).

5.3 Runtime Efficiency of Tool Kit and API

For completeness sake, as well as, a comparative benchmark to other commonly used feature extraction tools, we provide the runtime efficiencies of extracting permissions from APKs. To measure and compare the runtime efficiency of AndroParse, we used an Ubuntu Server 16.04 VM using 8x Intel Xeon CPUs E5-2640 v3 @ 2.6 GHz with 64 GB of memory. To time the extraction, UNIX's built in time command was used.

For testing, we randomly selected 1000 APKs and compared the runtime against multiple other tools which were chosen due to their popularity in the community (e.g., highly cited or featured on GitHub). The results are shown in Table 5. It is important to mention that we only extracted permissions as none of the tools can parse the same features as AndroParse's framework. The exact methodology was as follows:

1. Randomly select 500 benign APK files
2. Randomly select 500 malicious APK files
3. Execute each tool extracting permissions from each APK
   - Note in the case of Apktool, only the *AndroidManifest.xml* is parsed as this tool does not provide permissions directly.
4. Log the time taken for the process to execute

Table 5. Extraction runtime efficiency of permissions.

Tool	Time (s)
AndroParse	6.291
php_apk_parser	13.173
Androguard	88.738
Apktool	733.928

5.4 Usability Based on Previous Works

In the following we highlight how existing work could have benefited from AndroParse. Therefore, we will briefly summarize what researchers did to extract the features, and then we will show how the identical feature vector can be created using our framework.

Permission Based Approach by [30]. In their work, the authors cross compare the standard permissions found in the AndroidManifest.xml (i.e., any permission starting with android.permission) with all standard permissions offered in Android[14]. If the APK requests a standard permission, it generates a 1 in the vector and 0 otherwise. To do so, the users extracted permissions of an APK

[14] https://developer.android.com/reference/android/Manifest.permission.html (last accessed 13-April-2018).

using Androguard which is a timely process when scaled to thousands of APKs (Compare Table 5). On the other hand, AndroParse can provide this information (used permissions of an APK) by running the following query

```
https://hostname/api/?fields=Malicious,Permissions
```

which returns the (list of all) permissions and a true/false malicious indicator for each APK formatted in a list of key value pairs as described in Appendix A Listing 4. Next, the output of AndroParse's REST API conversion into feature vectors can be done with a short Python script and does not require sophisticated programming skills (See Listing 2 or in our repository under `./analysis/perms.py`). Lines 3–7 load in the downloaded JSON file into a dictionary and line 9 creates a list of all standard Android permissions to compare against. Continuing along, lines 11–16 create a permission's binary vector. Lastly, lines 22–25 loop through the Android APKs and generate a permission's binary vector for each, as well as, determine if the given APK is benign or malicious.

```
1  import json
2
3  def get_apk_json(filepath):
4    d = {}
5    with open(filepath) as json_data:
6      d = json.load(json_data)
7    return d
8
9  PERMISSIONS = [<standard Android permissions>]
10
11 def get_permissions(apk):
12   perms = []
13   for permission in PERMISSIONS:
14     status = 1 if permission in apk['Permissions'] else 0
15     perms.append(status)
16    return perms
17
18 feature_vector = []
19 target_vector = []
20 apks = get_apk_json("perms.json")
21 apks = apks['data']
22 for apk in apks:
23    feature_vector.append(get_permissions(apk))
24    target_type = 1 if apk['Malicious'] == 'true' else 0
25    target_vector.append(target_type)
```

Listing 2. Excerpt from perms.py.

Permission Based Approach by [20]. Another work extracted the permission list from an APK and from that list, a count of the total permissions. This work could use the identical query to the one in our prior example. Once downloaded, they would need to iterate through and determine a count of permissions for each

APK. A script demonstrating this parsing of output and counting of permissions can be found at `./analysis/permscount.py`.

Permissions, APIs, and Strings Approach by [41]. In particular, this approach was only concerned with strings that contained a system command (e.g., `chown` or `mount`) [41] as well as Permissions and APIs. To collect the necessary data for this approach, the authors could have queried:

```
https://hostname/api/?fields=Malicious,Permissions,APIs,Strings
```

A script is provided under `./analysis/permstringsapis.py` which parses the downloaded JSON data, as well as, build the feature vectors for permissions, APIs, and system commands.

In summary, AndroParse directly provides the needed information and does not require a sophisticated APK parser. It is important to note that since the code for each of these three works were not made publicly available, the scripts were constructed to match as close as possible to the description in each of the respective papers. The scripts are located in our public source code repository and can be taken advantage of in future work.

5.5 Runtime Efficiency Assessment

In this section we briefly discuss the runtime efficiency for AndroParse as a stand alone tool as well as querying our web front end.

AndroParse. To test the feature extraction performance, we selected several sample sizes (from 250 to 2000 APKs) that were randomly chosen from our dataset. The results are summarized in Table 6; focus on *Extract* columns. As can be seen, the time for extracting features can be time consuming (over 1 h for 2000; not including validation against VirusTotal). The limiting factor here is the thread lock when extracting strings and APIs using RAPID JAR.

Web Front End. For testing the front end, we downloaded all data using an *all* query (`https://hostname/all`) which pulls all fields from each APK. Again, the results are listed in Table 6 (focus on *query* columns). Of course, downloading the required data is much more space and time efficient.

Table 6. Query vs. extraction performance.

APKs	Size (MB)		Time		Performance (MB/s)	
	Query	Extract	Query	Extract	Query	Extract
2000	1561	11,888	53.5 s	64 m16.2 s	29.17	3.08
1000	765	6,522	28.4 s	36 m28.2 s	27.98	2.98
750	592	5,017	21.9 s	28 m9.9 s	27.04	2.97
500	407	3,481	16.5 s	19 m18.1 s	24.68	3.01
250	199	1,945	7.7 s	9 m54.8 s	25.90	3.27

6 Limitations

In its current form, AndroParse has two main limitations. First, as discussed in Sect. 1, the extraction process is only concerned with static analysis of APK files. This was an initial design choice to focus on the usability of such a platform. Dynamic analysis can expand on features such as but not limited to: file operations, commands, network traffic, system properties, etc. [38]. Second, AndroParse has been multi-threaded as much as possible to reduce the time taken to extract features. In its current form, the *RAPID* JAR file must be ran with only one instance at a time using the resource. This is due to a low level unsafe memory access exception thrown from the JVM. Until this bug can be resolved, the strings and APIs must be ran sequentially, significantly slowing down the extraction process.

7 Conclusion

In this paper we presented AndroParse, a feature extraction tool for data scientists and forensic examiners, as well as, a feature dataset that can be accessed through a REST API.

AndroParse is a general framework that allows users to extract features/forensic artifacts in a rapid and scalable manner. It is written in Golang and can easily be extended. In its current version, the tool can extract package name, package version, MD5, SHA1, SHA256, date extracted, file size, permissions, APIs, strings and intents. Due to the usage of the JSON format for the output files, the features can be further processed using any language (e.g., for machine learning purposes). For instance, a user can utilize Elasticsearch.

Feature dataset was created using AndroParse and is an online dataset that currently contains the features of approximately 114,386 Android applications – 67,703 benign and 46,683 malicious. Compared to previous approaches, we do not share the malware samples directly but only the features which comes with two benefits. First, the malware samples are not shared and thus cannot be misused. Second, researchers do not have to extract the features on their side which saves time and processing power.

Acknowledgements. We like to thank the University of New Haven's Summer Undergraduate Research Fellowship (SURF) program who supported this research.

A Identifying Relevant Features Used

```
"Md5":"66bd8...3557ea2",
"Sha1":"h5k7...fhl65t",
"Sha256":"b277...2f443",
"Malicious":true,
"Apis":[
   "void android.app.Activity.<init>()",
   ...],
"PackageName":"bubei.pureman",
"Version":"1.0.1",
"Intents":[
    "android.intent.action.MAIN",
    "android.intent.category.LAUNCHER",
    ...],
"Permissions":[
    "android.permission.WRITE_SMS",
    ...],
"Date":"2017-12-07 16:41:51",
"FileSize":1699930,
"Strings":[
    "",
     "",
     "\u00d0",
    " ",
    "    ",
    " Build/",
    ...]
```

Listing 3. JSON output of AndroParse of a single malicious application.

```
{
    [
        "Malicious":true,
        "Permissions":[
            "android.permission.WRITE_SMS",
            ...]
    ],
}
```

Listing 4. JSON output of AndroParse's REST API querying for permissions and malicious status.

```
{
  "apks": {
    "mappings": {
      "apk": {
        "properties": {
          "Apis": {
            "type": "text"
          },
          "Date": {
            "type": "date",
            "format": "YYYY-MM-dd'T'HH:mm:ss"
          },
          "FileSize": {
            "type": "integer"
          },
          "Intents": {
            "type": "text"
          },
          "Malicious": {
            "type": "text"
          },
          "Md5": {
            "type": "text"
          },
          "PackageName": {
            "type": "text"
          },
          "PackageVersion": {
            "type": "text"
          },
          "Sha1": {
            "type": "text"
          },
          "Sha256": {
            "type": "text"
          },
          "Strings": {
            "type": "text"
          },
          "Permissions": {
            "type": "text",
            "fields": {
              "keyword": {
                "type": "keyword",
                "ignore_above": 256
              }
            }
          }
        }
      }
    }
  }
}
```

Listing 5. JSON mapping used by Elasticsearch.

Table 7. Overview of articles including their features utilized for our work.

Ref.	Features	Citation
[30]	Permissions, Control Flow Graphs	"In this article, we present a machine learning based system for the detection of malware on Android devices."
[11]	Permissions, APIs, Strings, Meta Data, Opcodes, Intents	"This study summarizes the evolution of malware detection techniques based on machine learning algorithms focused on the Android OS."
[38]	Signatures, Permissions, Application Components, APIs	"[...]we propose a novel hybrid detection system based on a new open-source framework CuckooDroid[...]"
[41]	APIs, Permissions, System Commands	"This paper proposes and investigates a parallel machine learning based classification approach for early detection of Android malware."
[16]	Permissions, Smali Code, Intents, Strings, Components	"In this paper, we studied 100 research works published between 2010 and 2014 with the perspective of feature selection in mobile malware detection."
[33]	Permissions, Intents, Services and Receivers, SDK version APIs, Strings	"In this paper, we present Mobile-Sandbox, a system designed to automatically analyze Android applications in novel ways[...]"
[10]	Permissions, APIs, URI Calls	"This paper presents an approach which extracts various features from Android Application Package file (APK) using static analysis and subsequently classifies using machine learning techniques."
[7]	Components, Permissions, Intents APIs, Strings	"In this paper, we propose DREBIN, a lightweight method for detection of Android malware that enables identifying malicious applications directlyon the smartphone."
[17]	Intents, Permissions, System Commands, APIs	"In this chapter, we propose a machine learning based malware detection and classification methodology,with the use of static analysis as feature extraction method."
[6]	File Properties, APIs, System Calls, JavaScript, Strings	"To discover such new malware, the SherlockDroid framework filters masses of applications and only keeps the most likely to be malicious for future inspection by anti-virus teams."
[2]	APIs, Permissions	"In this paper, we aim to mitigate Android malware installation through providing robust and lightweight classifiers."
[18]	Permissions, APIs	"In this paper, we present a feasibility analysis for enhancing the detection accuracy on Android malware for approaches relying on machine learning classifiers and Android applications' static features."
[28]	Permissions	"In the present study, we analyze two major aspects of permission-based malware detection in Android applications: Feature selection methods and classification algorithms."
[8]	Permissions, URI Calls, Intents	"In this paper, we perform an analysis of the permission system of the Android smartphone OS[...]"
[3]	APIs, Smali Code	Used the decompiled smali code to "[...] link APIs to their components."

References

1. apktool (2010). http://ibotpeaches.github.io/Apktool/
2. Aafer, Y., Du, W., Yin, H.: DroidAPIMiner: mining API-level features for robust malware detection in android. In: Zia, T., Zomaya, A., Varadharajan, V., Mao, M. (eds.) SecureComm 2013. LNICST, vol. 127, pp. 86–103. Springer, Cham (2013). https://doi.org/10.1007/978-3-319-04283-1_6
3. Anonymous. CAPIL: Component-API linkage for android malware detection (2016, unpublished)
4. APK-DL. Apk downloader (2016). http://apk-dl.com. Accessed 13 Apr 2018
5. APKPure. Download APK free online (2016). https://apkpure.com. Accessed 13 Apr 2018
6. Apvrille, L., Apvrille, A.: Identifying unknown android malware with feature extractions and classification techniques. In: 2015 IEEE Trustcom/BigDataSE/ISPA, vol. 1, pp. 182–189. IEEE (2015)
7. Arp, D., Spreitzenbarth, M., Hübner, M., Gascon, H., Rieck, K., CERT Siemens: DREBIN: effective and explainable detection of android malware in your pocket. In: Proceedings of the Annual Symposium on Network and Distributed System Security (NDSS) (2014). https://www.sec.cs.tu-bs.de/~danarp/drebin/. Accessed 13 Apr 2018
8. Au, K.W.Y., Zhou, Y.F., Huang, Z., Lie, D.: PScout: analyzing the android permission specification. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security, pp. 217–228. ACM (2012)
9. Aung, Z., Zaw, W.: Permission-based android malware detection. Int. J. Sci. Technol. Res. **2**(3), 228–234 (2013)
10. Babu Rajesh, V., Reddy, P., Himanshu, P., Patil, M.U.: Droidswan: detecting malicious android applications based on static feature analysis. Comput. Sci. Inf. Technol., 163 (2015)
11. Baskaran, B., Ralescu, A.: A study of android malware detection techniques and machine learning. University of Cincinnati (2016)
12. Bhatia, A.: Android-security-awesome, February 2017. https://github.com/ashishb/android-security-awesome. Accessed 13 Apr 2018
13. Desnos, A.: Androguard-reverse engineering, malware and goodware analysis of android applications. URL code. google.com/p/androguard (2013)
14. eLinux. Android AAPT, June 2010. http://www.elinux.org/android_aapt. Accessed 13 Apr 2018
15. Faruki, P., Bharmal, A., Laxmi, V., Gaur, M.S., Conti, M., Rajarajan, M.: Evaluation of android anti-malware techniques against Dalvik bytecode obfuscation. In: 2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications, pp. 414–421. IEEE (2014)
16. Feizollah, A., Anuar, N.B., Salleh, R., Wahab, A.W.A.: A review on feature selection in mobile malware detection. Digit. Invest. **13**, 22–37 (2015)
17. Fereidooni, H., Moonsamy, V., Conti, M., Batina, L.: Efficient classification of android malware in the wild using robust static features (2016)
18. Geneiatakis, D., Satta, R., Fovino, I.N., Neisse, R.: On the efficacy of static features to detect malicious applications in android. In: Fischer-Hübner, S., Lambrinoudakis, C., Lopez, J. (eds.) TrustBus 2015. LNCS, vol. 9264, pp. 87–98. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-22906-5_7
19. Holmes, G., Donkin, A., Witten, I.H.: WEKA: a machine learning workbench. In: Proceedings of the 1994 Second Australian and New Zealand Conference on Intelligent Information Systems, pp. 357–361. IEEE (1994)

20. Kaushik, P., Jain, A.: Malware detection techniques in android. Int. J. Comput. Appl. **122**(17), 22–26 (2015)
21. Maggi, F., Valdi, A., Zanero, S.: Andrototal: a flexible, scalable toolbox and service for testing mobile malware detectors. In: Proceedings of the Third ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, pp. 49–54. ACM (2013)
22. Maiorca, D., Ariu, D., Corona, I., Aresu, M., Giacinto, G.: Stealth attacks: an extended insight into the obfuscation effects on android malware. Comput. Secur. **51**, 16–31 (2015)
23. Malik, S., Khatter, K.: AndroData: a tool for static & dynamic feature extraction of android apps. Int. J. Appl. Eng. Res. **10**(94), 98–102 (2015)
24. Nativ, Y.T., Shalev, S.: Thezoo (2015). http://thezoo.morirt.com. Accessed 13 Apr 2018
25. Newman, D.J., Hettich, S., Blake, C.L., Merz, C.J.: UCI repository of machine learning databases (1998). http://mlearn.ics.uci.edu/MLRepository.html. Accessed 13 Apr 2018
26. Parkour, M.: Contagio mobile. Mobile malware mini dump (2013). https://contagiominidump.blogspot.ca/. Accessed 13 Apr 2018
27. Payload Security. Learn more about the standalone version or purchase a private web service (2016). https://www.hybrid-analysis.com/. Accessed 13 Apr 2018
28. Pehlivan, U., Baltaci, N., Acartürk, C., Baykal, N.: The analysis of feature selection methods and classification algorithms in permission based android malware detection. In: 2014 IEEE Symposium on Computational Intelligence in Cyber Security (CICS), pp. 1–8. IEEE (2014)
29. Rami, K., Desai, V.: Performance base static analysis of malware on android (2013)
30. Sahs, J., Khan, L.: A machine learning approach to android malware detection. In: 2012 European Intelligence and Security Informatics Conference (EISIC), pp. 141–147. IEEE (2012)
31. Sanz, B., Santos, I., Laorden, C., Ugarte-Pedrero, X., Bringas, P.G., Álvarez, G.: PUMA: permission usage to detect malware in android. In: Herrero, Á., et al. (eds.) International Joint Conference CISIS'12-ICEUTE' 12-SOCO' 12. AISC, vol. 189, pp. 289–298. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-33018-6_30
32. Seth, R., Kaushal, R.: Permission based malware analysis & detection in android (2014)
33. Spreitzenbarth, M., Schreck, T., Echtler, F., Arp, D., Hoffmann, J.: Mobile-sandbox: combining static and dynamic analysis with machine-learning techniques. Int. J. Inf. Secur. **14**(2), 141–153 (2015)
34. SunFeith. php_apk_parser (2013). https://github.com/iwinmin/php_apk_parser. Accessed 13 Apr 2018
35. Svensson, R.: Das malwerk (2016). http://dasmalwerk.eu. Accessed 13 Apr 2018
36. Tdoly. tdoly/apk_parse. GitHub (2015). https://github.com/tdoly/apk_parse. Accessed 13 Apr 2018
37. VirusTotalTeam. Virustotal-free online virus, malware and url scanner (2013). https://www.virustotal.com/. Accessed 13 Apr 2018
38. Wang, X., Yang, Y., Zeng, Y.: Accurate mobile malware detection and classification in the cloud. SpringerPlus **4**(1), 1 (2015)
39. Wei, F., Li, Y., Roy, S., Ou, X., Zhou, W.: Deep ground truth analysis of current android malware. In: Polychronakis, M., Meier, M. (eds.) DIMVA 2017. LNCS, vol. 10327, pp. 252–276. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-60876-1_12

40. Winsniewski, R.: Android–apktool: a tool for reverse engineering android APK files (2012)
41. Yerima, S.Y., Sezer, S., Muttik, I.: Android malware detection using parallel machine learning classifiers. In: 2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies, pp. 37–42. IEEE (2014)
42. Zhang, X., Breitinger, F., Baggili, I.: Rapid android parser for investigating dex files (RAPID). Digit. Invest. **17**, 28–39 (2016)
43. Zhou, Y., Jiang, X.: Android malware genome project. Disponibile a (2012). http://www.malgenomeproject.org
44. Zhou, Y., Wang, Z., Zhou, W., Jiang, X.: Hey, you, get off of my market: detecting malicious apps in official and alternative android markets. In: NDSS, vol. 25, pp. 50–52 (2012)