A new tool for static and dynamic Android malware analysis

A. MARTÍN*, R. LARA-CABRERA* and D. CAMACHO*

Computer Science Department,
Universidad Autónoma de Madrid,
Madrid, Spain
**E-mails: {alejandro.martin,raul.lara,david.camacho}@uam.es*
www.uam.es

AndroPyTool is a tool for the extraction of both, static and dynamic features from Android applications. It aims to provide Android malware analysts with an integrated environment to extract multi-source features able of modelling the behaviour of a sample and that can be used to discern its nature, whether malware or goodware. AndroPyTool integrates well known tools in this field, such as AndroGuard, DroidBox, FlowDroid, AVClass, VirusTotal or Strace, which allow to obtain a wide set of features including Application Programming Interface (API) calls, permissions, labels obtained from the different antivirus engines included in VirusTotal, Source-Sinks data connections, API calls invoked in real time, accessed files, files operations and many others. AndroPyTool is an open source tool that can be used via both the source code and a Docker container, in just three stages (pre-static, static and dynamic analysis).

Keywords: Cybersecurity; Android Malware Analisys; Static Analysis; Dynamic Analysis

1. Introduction

In recent years, malware analysis has acquired great importance positioning itself as a strategic and promising research field. The continuous apparition of new shapes of malware always goes hand in hand with a deep study in order to prevent the effects of these new signatures and to mitigate the damage. Android has focused many researches in this respect, mostly due to the amount of malware designed specifically for this platform. Many of these researches have make an effort on designing malware detection tools, which typically follow the same procedural steps: firstly, a feature extraction step is carried out, to procure information able to describe the behaviour and intentions from each particular sample; secondly, to build a features space all samples can be represented, lastly, to train a classification method able to create a separation between malware and goodware (also known as benignware) samples in this space.

Focusing on the first step, many tools have been proposed. These are aimed at extracting several features, each one focused on modelling the behaviour of a sample from a different perspective. For instance, the well-known Androguard tool is designed for extracting static features from files contained in the package file. DroidBox, another widely used tool, is employed to run an application in an emulator while all the actions undertaken by the sample under analysis are captured. However, if a complete and multilevel analysis is required (*i.e.* combining the output of different static and dynamic features), it is necessary to deploy and execute separately all these tools.

In this paper, we present AndroPyTool, a framework for integrating the process of extracting varied static and dynamic features. AndroPyTool comprehends the most important Android analysis tools, performs code inspection in order to retrieve a wide set of characteristics and processes all the information gathered. The main objective of this tool is to provide those researchers interested in Android malware with an all-in-one tool that allows to perform the whole malware analysis cycle of a suspicious sample, thus making this process faster and easier. As a result, a detailed report for each sample is generated containing all the collected features. Furthermore, it provides a scalable solution and enables to easily integrate new tools. AndroPyTool is publicly available[a].

2. Background

The 2017 *State of Malware Report* from Malwarebytes LABS illustrates and evidences the current problem stated by mobile malware. Furthermore, it evidences new concerns due to the novel characteristics found in the latest Android malware samples found in the wild, which are becoming smarter, and due to the significant presence of ransomware for this platform. Recently, the increase in the number of malware detected specifically designed for the Android platform[1] has been accompanied by the corresponding efforts for tackling this problem. Thus, multiple mechanisms, tools and frameworks appeared to model the behaviour of malicious and benign samples, building classification tools for detecting malware or defining new kinds of features among many others. Specifically, all the tools proposed so far are

[a]`https://github.com/alexMyG/AndroPyTool`

typically grouped into two different categories:[2] static and dynamic analysis, although it is also possible to establish a previous stage named pre-static analysis. The *pre-static analysis* is focused on extracting meta-information, features from the compressed *apk* file which do not require code inspection. The *static analysis* step is employed to extract more detailed features, such as declared Application Programming Interface (API) calls or permissions required. Finally, the *dynamic analysis* step monitors the application when it is executed in order to capture the real behaviour and actions performed.

There are different tools widely employed for analysing Android applications. In this sense, Apktool[3] is commonly used to decompress and decode the application executable. It allows to read the different files contained, such as the Android Manifest and also the code extracted from Dalvik byte-code to *smali*. The *Androguard* tool[b] is a Python library to obtain dozens of features from an *apk* file. Dex2jar[4] is used to decompile Dalvik bytecode to Java compiled code (a *.jar file*) which can be later read as Java using *jd-gui*. Other tools allow to obtain deeper characteristics. For instance, FlowDroid[5] runs taint analysis to detect information flows. In the dynamic analysis research area, DroidBox[6] connects to an Android emulator to obtain dynamically extracted information.

All these instruments try to model the behaviour of Android samples in order to, for instance, develop malware detection frameworks such as RevealDroid,[7] which combines static features like API calls and taint analysis traces obtained with FlowDroid, to later train a machine learning classifier (including the C4.5 decision-tree classifier and the 1-nearest-neighbor algorithm) to segregate samples between malware and goodware. Drebin[8] makes malware classification based on permissions requested or API calls invoked. MOCDroid[9] employs a genetic algorithm to build an Android malware classifier. Other example is ADROIT,[10] which detects Android malware analysing meta-information and where a set of machine learning classifiers are trained. Other examples such as Droid-Sec[11] include the use of deep learning to detect Android malware.

3. AndroPyTool

AndroPyTool integrates different analysis tools and Android applications processing tools, in order to deliver fine-grained reports drawing their individual behaviour and characteristics. This framework, which has been implemented as a Python project, takes the *apk* file to be analysed as input

[b]`https://github.com/androguard/androguard`

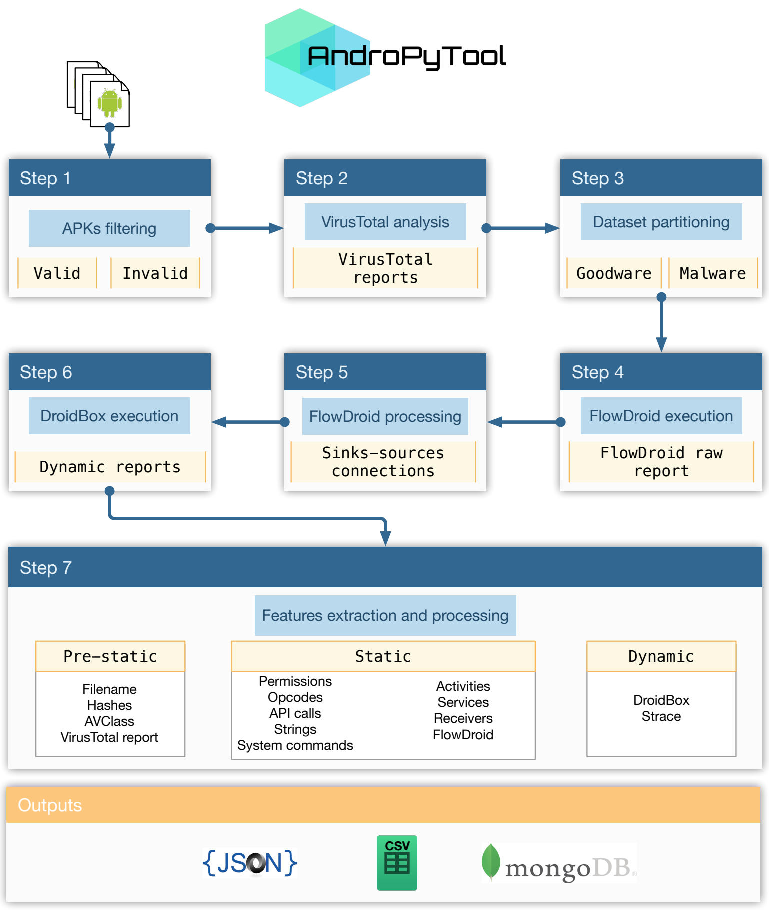

Fig. 1. AndroPytool architecture.

and follows up a seven-step process (see Fig. 1) in which different static and dynamic analysis tools are executed, the code is automatically inspected to obtain different features, and finally concludes generating both, an application independent report including all the features extracted and processed, and a secondary report including all samples analysed.

- **Step 1: APKs filtering.** This first step is in charge of inspecting each sample with Androguard in order to check if the sample is a true and a valid Android application.
- **Step 2: VirusTotal analysis.** A report for each sample is re-

trieved from the VirusTotal portal. It contains the scan result and analysis date from more than 60 different antivirus engines.

- **Step 3: Dataset partitioning.** A categorisation based on the VirusTotal report is used to split the set of samples into two sets: malware samples (at least one antivirus tests for positive) and goodware samples.
- **Step 4: FlowDroid execution.** The taint analysis based FlowDroid framework is executed for each sample.
- **Step 5: FlowDroid outputs processing.** The output delivered by FlowDroid for each sample is processed in order to extract the connections between sources and sinks found.
- **Step 6: DroidBox execution.** In this step, the Android dynamic analysis tool DroidBox is executed for each sample. The version executed of DroidBox includes some modifications as integrating the *Strace*.
- **Step 7: Features extraction and processing.** Finally, the rest of the features are extracted and all the information is processed and handled in order to put it together. A report for each sample and global output including all the samples analysed is generated in three different formats: *Comma-separated Values (CSV)*, *JavaScript Object Notation (JSON)* and as a MongoDB database.

3.1. *Features description*

Through the seven-step process, AndroPyTool executes different tools in order to extract a wide set of features from an input set of Android samples. All these features, and the information that they represented, are organised in three different categories (pre-static, static and dynamic), both the features and how they are extracted are described below:

3.1.1. *Pre-static analysis*

It involves extracting information without code inspection and allows to identify and to track the sample. This category includes: the file name, MD5, SHA1 and SHA256 hashes, a report obtained from the VirusTotal[c] portal with the scan result from more than 60 antivirus engines, and consensual label determined by the AVClass.[12] It also includes the package name and the main activity name, which are obtained with *Androguard*.

[c]`https://www.virustotal.com/`

3.1.2. *Static analysis*

This category includes those features that are retrieved by analysing the application at the code level. They can be used to build a representation about the expected behaviour of the sample. In this category, features such as API calls, activities, opcodes or permissions can be found. More specifically the features extracted are:

- **API calls.** They allow to model the expected behaviour of the sample. Certain API calls are commonly found among malware samples. These API calls are extracted by parsing the *smali* files and searching for *invoke* opcodes.
- **Opcodes.** Collecting a counter for each Dalvik opcode that appears in the *smali* files can help to understand the complexity and behaviour of the sample. Again, the *smali* files are parsed in order to retrieve all the opcodes declared.
- **Permissions.** Android permissions shape barriers between the application and the use of certain device functionalities and must be declared in Manifest. They are obtained using *Androguard*.
- **Intents.** An intent expresses an action to be performed. In many cases, malware often listens to some specified intents. The *AndroidManifest.xml* file is manually analysed to obtain this feature.
- **Receivers.** This application component allows applications to receive intents broadcast by the system or other applications. We focus on Receivers declared in the Android Manifest. A list of Intents declared for each Receiver is included.
- **Services.** This component allows performing operations without the user interaction or exposes functionalities to other applications to use. This feature is obtained by parsing the Manifest XML file. A list of Intents declared for each Service is included.
- **Activities.** Activities implement interfaces that allow the user to interact with the application. All activities must be declared in the *AndroidManifest.xml*, which can be parsed using *Androguard*. A list of Intents declared for each Activity is included.
- **Strings.** An analysis over the strings, which appear in the code as constants, becomes really interesting, as they can contain code to be loaded in runtime, network addresses or threatening messages. String variables are declared in the *smali* files as Dalvik opcodes.
- **System commands.** System commands are indicators of certain activities which must be controlled, like privilege escalation or the

process of rooting the device. These system commands can be stored as strings.

FlowDroid has also been integrated in AndroPyTool. It is a taint analysis based framework that detects potential data leaks within the life cycle of an application, which can be later used to differentiate malicious behaviours. Basically, it detects information flow that starts in a source (*i.e.* invoking an API call that returns certain data) and that end in a sink (*i.e.* another API call that receives this data). According to the authors, FlowDroid[5] is "*fully context, flow, field and object-sensitive*". It considers the whole Android application life-cycle as well as the UI widgets of the application. As result of executing FlowDroid, a list counting the number of connections between each source and sink is obtained.

3.1.3. *Dynamic analysis*

Finally, the last set of features are extracted by monitoring the application execution in a controlled environment (*i.e.* an emulator). We have chosen the DroidBox tool for this purpose, which allows to dynamically obtain diverse information in real time. This enables to make a fine-grained analysis of the application behaviour actually exhibited when it is executed. Furthermore, we have built a modification of the DroidBox framework to integrate Strace. The information gathered by the DroidBox tool includes: the use of *cryptographic* functions, loaded *DEX classes* in run time, *files accessed* and the kind of *operation*, *network connections*, *SMS*, *phone calls*, *started services*, *enforced permissions* and information leaks detected.

4. Conclusions

With AndroPyTool, our main goal is to provide researchers and malware analysts with a powerful and integrated tool for extracting multi source features from Android applications. Thus, this tool puts together the most important and employed tools in this domain, in order to build a framework that automates the process of extracting meta-information, executing static and dynamic analysis, processing all the data collected, to finally generating formatted outputs for later use. In future work, we aim to integrate more analysis tools into AndroPyTool and to improve the data processing stages, in order to give more functionalities to the users.

Acknowledgments

This work has been co-funded by the following research projects: Comunidad Autonoma de Madrid (CAM) grant S2013/ICE-3095 (CIBERDINE: Cybersecurity, Data and Risks), EphemeCH (TIN2014-56494-C4-4-P) and DeepBio (TIN2017-85727-C4-3-P) by Spanish Ministry of Economy and Competitivity, under the European Regional Development Fund FEDER, and Justice Programme of the European Union (2014-2020) 723180 – RiskTrack – JUST-2015-JCOO-AG/JUST-2015-JCOO-AG-1. The contents of this publication are the sole responsibility of their authors and can in no way be taken to reflect the views of the European Commission.

References

1. M. Lindorfer, M. Neugschwandtner, L. Weichselbaum, Y. Fratantonio, V. Van Der Veen and C. Platzer, Andrubis–1,000,000 apps later: A view on current android malware behaviors, in *Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*, (Wroclaw, Poland, 2014).
2. K. Tam, A. Feizollah, N. B. Anuar, R. Salleh and L. Cavallaro, *ACM Computing Surveys (CSUR)* **49**, p. 76 (2017).
3. R. Winsniewski, Android–apktool: A tool for reverse engineering android apk files (2012).
4. B. Alll and C. Tumbleson, *Octeau D, Jha S, McDaniel R Retargeting* .
5. S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau and P. McDaniel, *Acm Sigplan Notices* **49**, 259 (2014).
6. P. Lantz, A. Desnos and K. Yang, Droidbox: Android application sandbox (2012).
7. J. Garcia, M. Hammad, B. Pedrood, A. Bagheri-Khaligh and S. Malek, *Obfuscation-resilient, efficient, and accurate detection and family identification of android malware*, tech. rep., Department of Computer Science, George Mason University (2015).
8. D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon and K. Rieck, Drebin: Effective and explainable detection of android malware in your pocket., in *NDSS*, (San Diego, USA, 2014).
9. A. Martín, H. D. Menéndez and D. Camacho, *Soft Computing* **21**, 7405 (2017).
10. A. Martín, A. Calleja, H. D. Menéndez, J. Tapiador and D. Camacho, Adroit: Android malware detection using meta-information, in *Computational Intelligence (SSCI), 2016 IEEE Symposium Series on*, December 2016.
11. Z. Yuan, Y. Lu, Z. Wang and Y. Xue, Droid-sec: Deep learning in android malware detection, in *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14 (ACM, New York, NY, USA, August 2014).
12. M. Sebastián, R. Rivera, P. Kotzias and J. Caballero, Avclass: A tool for massive malware labeling, in *International Symposium on Research in Attacks, Intrusions, and Defenses*, (Evry, France, 2016).