Coriander: A Toolset for Generating Realistic Android Digital Evidence Datasets

Irvin Homem(✉)

Department of Computer and Systems Sciences,
Stockholm University, Postbox 7003, Kista, Sweden
irvin@dsv.su.se

Abstract. Triage has been suggested as a means to prioritize and identify sources and artifacts of evidence that might be of most interest when faced with large amounts of digital evidence. Memory Forensics has long relied on simple string matching to triage evidence sources. In this paper, we describe the early developments into our study on Machine Learning-based triage for Memory Forensics. To start off, there are no large datasets of memory captures available. We thus, develop a toolset to enable the automated creation of realistic Android process memory dumps. Using our toolset we generate a dataset of 2375 process memory string dumps from both malicious and benign Android applications, classified by VirusTotal, and sourced from the AndroZoo project. Our dataset and toolset are made available online to help promote research in this field and related areas.

Keywords: Android forensics · Digital forensics · Mobile forensics
Memory forensics · Digital evidence · Datasets · Metadata · Machine learning
Triage

1 Introduction

Digital Investigations struggle with large amounts of potential evidentiary data. Triage [1] has been proposed as a means to help speed up the identification of high priority digital evidence data sources for acquisition, or sections of digital evidence that should be prioritized for analysis [2]. Triaging disk-based evidence to identify files or sections of a disk to be either ignored, or prioritized has been studied widely [3–5]. Triaging of network traffic captures has been less studied, however there are some studies such as [6, 7]. There has been a call for mobile device triage [8] however, to the best of our knowledge little has been done. We thus delve into addressing this knowledge gap, directing our efforts into the triaging of mobile device memory in an automated manner, with a focus on identifying processes in memory that may require further investigation. We propose the use of machine learning techniques as previously used with disk-based [5] and network traffic evidence [7], to aid in identification and prioritization of processes of interest within mobile memory dumps, as part of a triage procedure.

More concretely, in the early stages of this research, we aim to create a significantly large dataset of Android process memory dumps. From these memory dumps, process

P. Matoušek and M. Schmiedecker (Eds.): ICDF2C 2017, LNICST 216, pp. 228–233, 2018.
https://doi.org/10.1007/978-3-319-73697-6_18

related features (metadata) are to be extracted and used to develop machine learning predictive models to help identify particular processes in memory that warrant further investigation into their activities and interactions with the file system or network.

To aid in achieving our goals, we have developed a toolset (Coriander) in Python to automate the creation of our dataset of Android process memory dumps. Using this toolset we generated a dataset of 2375 Android process memory string dumps using a subset of APK files from the AndroZoo Project [9]. The Coriander toolset and the resulting process memory dataset are the main subject described in this early progress report on our research.

2 Background and Related Work

Some of the methods for triaging digital evidence include removal of known benign artifacts from within the digital evidence dataset. This has been done on *disk* data [2] using: (i) Focused extraction of well known artifacts (E.g. the Windows Registry, File Metadata); (ii) Using string-matching techniques [10]; (iii) Matching files with hash lists of known files or parts of known files [3]; (iv) Using fuzzy hashing to identify closely similar files [4]; and (v) recently also using machine-learning techniques [5]. With regard to triaging *network traffic* evidence, fuzzy hashing has been used to detect files [6] and machine learning methods have been used to identify network protocols within DNS tunneling network traffic [7].

Recently network-scanning tools such as Yara have been deployed to scan memory images as a triage method [11]. This used string-matching techniques, with additional capability of conditional matching of sets of strings through Yara. Visualization techniques have also been developed to help identify areas of priority in the triage process of Windows memory [12].

The triage of mobile devices is largely unexplored and so far only one study [8] has attempted to address this knowledge gap, focusing on the different forms of evidence artifacts available and indicating that there is a lack of tools and techniques for triaging mobile devices, short of "thumbing through" a live device. We thus aim to address this shortcoming by providing a technique for triaging mobile devices with a focus on the processes running in memory.

In the forensic analysis of live memory, an important artifact of interest is the running processes, their interaction with other resources, such as other processes, data in memory, the filesystem, network adapters, the kernel and other peripheral devices drivers. The activities carried out by a process may be malicious or normal benign activity. Within the context of an investigation of live memory, it would be helpful to provide a forensic analyst with a quick method of identifying and differentiating potentially interesting malicious processes from other benign normal process activities. Thus, in this paper, we begin our study towards achieving this triage process on Android mobile device memory.

The identification of malicious Android applications has been studied as malware detection prior to the app running (Static Analysis) with static APK files [13]; while it is running (Dynamic Analysis) [14], or a combination of both (Hybrid Analysis) [15]. These methods are not perfect, and there are ways to beat them [16]. Essentially mobile

malware variants can beat detection mechanisms and continue to run undetected. Memory forensics tools such as Volatility and Rekall provide in depth structured analysis, however identification of miscreant processes is left to the discretion of the analyst. With potentially numerous processes in memory it may be a difficult task to identify which processes require further analysis. Whitelisting certain processes based on their name might be one way, however even well known processes may be hijacked to perform malicious tasks. This gives rise to a need for providing more robust, automated techniques for identifying malicious processes on memory images after an incident. To the best of our knowledge the automated identification of malicious processes on memory dumps has not been performed. Thus, we aim to use machine learning techniques to characterize the memory footprint of malicious and benign applications, so as to automate distinguishing between the two classes, and hence provide an automated classifier for triaging malicious processes within memory dumps.

3 The Coriander Toolset

To develop our technique for classifying Android process memory instances we needed a dataset of process memory dumps to identify relevant features. As there is no such dataset available, we set out to generate one. We developed the *Coriander Toolset* to automate this generation of realistic Android process memory dumps from real world APK files. The Coriander Toolset is composed of two major components: The *Coriander* application[1] and the *AndroMemDump* application[2]. The Coriander application coordinates the running of APK files within an Android Emulator and initiates the memory dumping procedure. The AndroMemDump application enables the actual dumping of a given running process' memory space. The functionality of these two applications is described in the following subsections:

3.1 Coriander

The Coriander Python application is made up of 3 main components: *SDK Tools*, *APK Tools* and the *Cookbook*.

1. The SDK Tools package consists of wrappers for the Android Debug Bridge (ADB), the Android Emulator and a class for managing SDK location configurations. It provides a logical abstraction of components of the Android SDK that allow for running, querying and controlling various parameters of an Android device or emulator.
2. The APK Tools package is comprised of two main abstractions: The *APK Store* and the *APK File*. The APK Store serves to maintain the location configurations and metadata extracted from a repository of APK files. The location can be a remote network path, or a local directory on the device running Coriander. The specific parameters are stored in a JSON file within the 'config' directory. The APK File

[1] Source code available at: https://github.com/irvinhomem/Coriander.

[2] Source code available at: https://github.com/irvinhomem/AndroMemDumpBeta.

class holds the metadata of a specific APK file, as well as functions for extracting specific metadata out of an APK file. The metadata stored include the *package name*, the *activities* and *permissions*. Other metadata could be captured, but these few are the important ones required to get an Android application to run.

3. The Cookbook package has a single class (*Recipe*) containing the instructions that the Coriander Toolset should run in a given session. There are 2 categories of instructions: Emulator instructions, and ADB/APK instructions. The emulator instructions revolve around the lifecycle of an emulator, that is, setting up an emulator instance, running the instance, resetting the instance, and killing the emulator instance. The ADB/APK instructions involve downloading APK files from an APK Store, installing apps, running app activities, initiating memory dumps, closing apps and uninstalling apps. To achieve these functionalities, the Cookbook calls methods from all other packages (SDK Tools, APK Tools) and the AndroMemdump application.

3.2 AndroMemDump

AndroMemDump is an Android application whose main function is to capture the process memory of a given process. The application is written in Java (using the Android API) and Native C code. The Native C code provides low-level access to the *ptrace* system call, which is used to capture process memory on Linux based systems [17]. When cross-compiled using the Android NDK, we get several flavours of our executable (*memdump*) for multiple process architectures i.e. *x86*, *x64*, *armeabi* and *mips*.

The Java based part harnesses the Android API to provide a simple, portable means of carrying, installing and calling native C executables within an Android ecosystem. The *memdump* binary is carried as an 'asset' within an APK, and is placed in the 'files' directory of the AndroMemDump app on first-run, after which 'execute' permissions are applied on the binary. Using our *memdump* executable AndroMemDump, we can capture process memory and save it onto the device internal memory, the SD Card, or transfer it over the network to a remote location. In conjunction with *Coriander*, process memory dumps can also be stored on the device hosting the emulator. Overall, this enables automating the capture of process memory from numerous APKs allowing us to create a large dataset within a reasonable amount of time.

4 Experiment Results and Discussion

Using the Coriander Toolset, we set out to generate a dataset of process memory dumps. This further required customizing an Android OS image to contain the AndroMemDump app and to avail root permissions. This involved modifying the '*/system'* partition of a stock Android ROM image to install our app as well as the '*su*' binary and the "Superuser" app by Chainfire (Jorrit Jongma). This was done such that after each run of our customized Android ROM on the emulator, we could wipe the *user* partition, to ensure APKs were completely gone, to avoid different malicious APK's interacting. The assumption made here was that malicious applications would

not bypass the Superuser app authorization to gain root privileges to modify the */system* partition. This decision was made as a tradeoff to having to install AndroMemDump and the 'su' binary on every round, which would slow the process down. Having these in the system partition and protected by the Superuser app was a good enough tradeoff.

Having all these in place we used the AndroZoo APK repository as our APK Store, extracting only a subset out of the over 5 million APKs available. The reason for using only a subset was due to time limitations and the size of each process memory dump. Each process memory dump took about 3–5 min to capture and store. The first few app dumps ranged between 0.8–1.5 GB in size each, thus we decided to capture only strings from each process memory dump as an initial feature set to reduce the size. The AndroZoo project classified many APKs as malicious or benign using VirusTotal, however not all were classified. Our aim was to achieve around 1000 malicious and 1000 benign process memory dumps. We ran our toolset sequentially through the repository and eventually attained 1187 benign samples and 1188 malicious samples[3]. Numerous apps had problems preventing them from executing, and were this skipped automatically by Coriander. The problems included corrupt manifest files, bugs within the code preventing installation, API level incompatibility and specially compiled native libraries that would not run on our customized ROM. We did not have the time to debug other app developer's code, nor to develop multiple ROMs to cater for the wide variation of compatibility issues in the Android ecosystem. Thus, it took 2321 and 7479 sampling rounds, respectively for benign and malicious classes, in order to achieve the 1187 and 1188 respective samples of process memory dumps from *working* APKs.

We discovered that process memory dumps in Android devices can be significantly large, thus we resorted to extracting only strings. This is acceptable since analyzing strings is one of the initial methods of performing memory forensics. We also saw the large amount of incompatibility issues that plague android apps between different versions of the Android API hindered our dataset collection efforts. Our Android ROM was customized from a stock Android 5.1 (API 22) which was the version commanding about 24% of the market share of Android devices - the 2nd highest at the time. We see the need for our toolset to have other ROMs available to allow for different flavours of the Android OS and thus increase compatibility; however, this comes at a cost of time and effort to maintain the different flavours.

5 Conclusions and Future Work

In this study, we delved into the first stage of our project to perform Machine Learning based triage on Android Memory Dumps. The first stage required a large dataset of realistic Android process memory dumps upon which we could extract features to develop machine learning models. We thus set out to create this dataset in this study. We developed the *Coriander Toolset* in order to help automate the creation of our dataset. From this toolset, we were able to create dataset of 2375 realistic Android process memory dumps, to further our own research and contribute to the larger research area.

[3] The dataset is available online at: https://doi.org/10.17045/sthlmuni.4989773.

This study only provides the initial progress into this work and has some limitations. Firstly, more customized ROMs need to be realized to get a better variety of process memory dumps and reduce the APKs skipped. Only strings of process memory dumps were captured; other memory metadata can be captured in future by extending the Coriander Toolset. This will also aid in the eventual feature selection process for the Machine Learning-based Triage goals that we intend to achieve in the future. Algorithms such as k-NN, decision trees, SVM's, neural networks, association rule mining, time series analysis and graph mining techniques are candidates for the classification task for our future work.

References

1. Rogers, M.K., Goldman, J., Mislan, R., Wedge, T., Debrota, S.: Computer forensics field triage process model. J. Digital Forensics, Secur. Law **1**, 19–38 (2006)
2. Roussev, V., Quates, C., Martell, R.: Real-time digital forensics and triage. Digital Invest. **10**, 158–167 (2013)
3. Mead, S.: Unique file identification in the national software reference library. Digital Invest. **3**, 138–150 (2006)
4. Kornblum, J.: Identifying almost identical files using context triggered piecewise hashing. Digital Invest. **3**, 91–97 (2006)
5. Marturana, F., Tacconi, S.: A machine learning-based triage methodology for automated categorization of digital media. Digital Invest. **10**, 193–204 (2013)
6. Breitinger, F., Baggili, I.: File detection in network traffic using approximate matching. J. Digital Forensics, Secur. Law **9**, 23–36 (2014)
7. Homem, I., Papapetrou, P.: Harnessing predictive models for assisting network forensic investigations of DNS tunnels. In: ADFSL Conference on Digital Forensics, Security and Law, Daytona Beach (2017)
8. Mislan, R.P., Casey, E., Kessler, G.C.: The growing need for on-scene triage of mobile devices. Digital Invest. **6**, 112–124 (2010)
9. Allix, K., Bissyandé, T.F., Klein, J., Le Traon, Y.: AndroZoo: collecting millions of android apps for the research community. In: 13th International Workshop on Mining Software Repositories - MSR 2016, Austin, TX, pp. 468–471 (2016)
10. Koopmans, M.B., James, J.I.: Automated network triage. Digital Invest. **10**, 129–137 (2013)
11. Cohen, M.: Scanning memory with Yara. Digital Invest. **20**, 34–43 (2017)
12. Lapso, J.A., Peterson, G.L., Okolica, J.S.: Whitelisting system state in windows forensic memory visualizations. Digital Invest. **20**, 2–15 (2016)
13. Karbab, E.B., Debbabi, M., Mouheb, D.: Fingerprinting android packaging: generating DNAs for malware detection. Digital Invest. **18**, 33–45 (2016)
14. Tam, K., Khan, S.J., Fattori, A., Cavallaro, L.: CopperDroid: automatic reconstruction of android malware behaviors. In: NDSS, pp. 8–11 (2015)
15. Lindorfer, M., Neugschwandtner, M., Weichselbaum, L., Fratantonio, Y., Van Der Veen, V., Platzer, C.: ANDRUBIS-1,000,000 apps later: a view on current android malware behaviors. In: 3rd International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security, pp. 3–17 (2014)
16. Petsas, T., Voyatzis, G., Athanasopoulos, E., Polychronakis, M., Ioannidis, S.: Rage against the virtual machine: hindering dynamic analysis of android malware. In: 7th European Workshop on System Security, pp. 5:1–5:6 (2014)
17. Thing, V.L.L., Ng, K.Y., Chang, E.C.: Live memory forensics of mobile phones. Digital Invest. **7**, S74–S82 (2010)