ClusTheDroid[1]

Clustering Android Malware

Authors:
David Korczynski, MSc (Royal Holloway, 2014)
Lorenzo Cavallaro, ISG, Royal Holloway

Abstract

The volume of new Android malware is growing at an exponential pace. This cries out for automated tools that can aid the malware analyst in dissecting the behaviours of new malicious applications. In this article, we present ClusTheDroid, a system for clustering Android malware so as to identify malicious applications that exhibit similar behaviours. For this, ClusTheDroid extracts feature sets from profiles of reconstructed malicious Android behaviours. These feature sets serve as input to a clustering algorithm that allows clustering of thousands of malicious applications in a single day. We implemented ClusTheDroid and evaluated it on real-world malware. We furthermore evaluate the use of the cluster-validity measure C-index, which offers a more flexible approach to identifying near-optimal clusters than that of previous literature.

Introduction

Smartphones have become extremely popular computing platforms over the last few years. Gartner estimates the worldwide sales of smartphones to end users in 2013 to be 967 million installed units. Of these almost one billion smartphones, 78.4% run Android as the underlying operating system. The complexity of the operating system and the number of daily tasks for which smartphones are used have increased manifold compared to first-generation mobile phones. Moreover, as a consequence of increased functionality, more and more personal information is stored on the personal devices. All this has resulted in a large attack surface, and smartphones have become a very lucrative target for cyber criminals. The growth of malware targeting Android exploded during 2013, and SophosLabs reports an accumulated number of individual Android malware samples to be around 650,000 at the end of 2013 compared to 150,000 at the beginning of 2013.

Automatic malware detection. The threat posed by Android malware makes it clear that mitigations are needed. More specifically, rigorous, automatic and efficient tools are needed to assist the Android malware analyst. In its simplest form, automatic malware detection is a two-step process. The first step is to extract information about the program under analysis and the second step it to use this information to draw conclusions about the potential malware. The first step can be done in two ways: static or dynamic analysis. Most prominently, regular expressions have for a long time been used as static signatures to detect known sequences of bytes specifying a specific type of malware. However, static analysis has many well-known limitations. For example, it is defeated by some popular anti-detection mechanisms such as code obfuscation and polymorphism. When static analysis fails, dynamic analysis takes over. The idea behind dynamic analysis is to execute the

[1] This article is to be published online by Computer Weekly as part of the 2015 Royal Holloway info security thesis series. The full MSc thesis is published on the ISG's website.

potential malware in a controlled environment. During the execution, specific properties of the environment are monitored to create a behavioural profile of the program. While dynamic analysis is a powerful technique, it comes with the cost of a large performance overhead. In addition, dynamic analysis also has its limitations. It is notoriously difficult to reach 100% code coverage during execution. Dynamic analysis is thus limited to potentially not seeing the entire possible control flow, which makes it particularly weak against trigger-based malware such as logic bombs.

In his seminal work on computer viruses Fred Cohen showed that theoretical virus detection comprises determining whether a program will ever halt. This is the famous Halting problem which has been proven to be undecidable. Thus virus detection is at least as hard as an undecidable problem. Effectively, automatic malware detection tools are probabilistic. The aim for automatic detection mechanisms is, therefore, not to state the definitive truth about each investigated sample. Rather the aim is to maximize the number of true positives and true negatives while minimizing the number of false positives and false negatives.

Behavioural analysis. Whenever automatic detection tools lack sufficiency, the malware analyst is forced to conduct manual analysis. Manual analysis is a time consuming task and is, again, carried out by either static analysis or dynamic (behavioural) analysis. Static analysis refers to actual inspection of the code and is carried out through a reverse engineering process using disassemblers such as IDA Pro. Behavioural analysis comprises inspecting the behaviour of the malware as it executes. In particular, behavioural analysis is an automated process where a dynamic analysis tool such as CWSandbox and Anubis executes the malware and outputs a behaviour profile. These behaviour profiles contain detailed information of the execution traces of the malware, and are used by the malware analyst to draw conclusions about the behaviour and threat-level of the malware. However, the current number of malware discovered implies that the malware analyst faces up to thousands of reports everyday. Therefore a need for prioritizing the reports arises. One way of processing the reports is to group the samples into families according to their monitored behaviour. There are two main benefits of being able to automatically group malware samples according to behaviour: Firstly, given groupings of malware that exhibit the same behaviours, it becomes easier to create generalized signatures and mitigation techniques. Secondly, it becomes easier for the malware analyst to determine whether malware samples found in the wild is an instance of a well-known family or a sample of a new malware variant.

ClusTheDroid. Machine learning techniques have been widely applied in the field of malware analysis. Mainly, supervised learning has been used as a means of malware detection, while unsupervised learning has been used to identify shared attributes between malware. However, the focus of these techniques have been on PC malware. As the threat of Android malware increases and the damages it causes become more severe, we need to focus more attention on mitigation techniques targeting Android malware. In this article we present ClusTheDroid, a tool for unsupervised learning on data extracted by behavioural analysis of Android malware. To the best of our knowledge, ClusTheDroid is the first approach to focus on clustering Android malware into families based on profiles of reconstructed malicious Android behaviours. We implemented the system comprising 1,700 lines of C code. We then evaluated it on real-world malware and compared our results to previous work. Our evaluation shows fair accuracy and efficient performance that can easily handle the current volume of Android malware. We also propose the use of C-index for determining near-optimal clusters when clustering unknown malware samples, which suggests a more flexible approach than that of previous work.

Malicious Android Applications

In 2012, Zhou and Jiang published a detailed analysis of 1,260 individual Android malware samples originating from 49 different malware families. They characterized each sample according to how it got installed, how the malware was activated, and the malicious payload carried by the malware. The entire dataset is published in the Android Malware Genome (Malgenome) Project. We will here present a description of the various types of payloads classified by Zhou and Jiang.

Privilege Escalation. Privilege Escalation refers to malware that exploit platform-level vulnerabilities to gain access to resources normally protected from an application. Such an example is malware from the GingerMaster family, which escalate privileges using the GingerBreak exploit.

Remote Control. Remote Control refers to malware that turn the infected phones into bots controlled by a command and control (C&C) server. Zhou and Jiang found that 1,172 of the 1,260 infected applications became part of a botnet and of these 1,172 samples, all but one used the HTTP protocol to receive commands from C&C servers. NickyBot is the only malware family which uses SMS messages to communicate with its C&C server. Some of the malware with this type of payload use encryption schemes to hide the URL of its C&C servers and also to hide the communication with its C&C servers.

Financial Charge. Financial Charge refers to malware that causes financial charges to the user. This type of payload is most frequently carried out by subscribing to premium-rate services as a means of gaining financial benefit. Amongst the 49 families in the data set, four actually call premium-rate call services and twenty subscribes to premium-rate SMS services.

Information Collection. Information Collection comprises malware that collects user information to share with remote servers. The information they collect can be both user credentials and other meta-information usually used by advertising companies. An example is malware from the FakeNetflix family, which collects the user's Netflix account and credit card information, and sends it to a remote server in Russia.

ClusTheDroid

The purpose of our system is to cluster Android malware samples into malware families. Our goal is to find a partition of an arbitrary set of malware samples so that the subsets share common malicious behaviours. Clustering malware is a multi-step process and consists of four individual components. The first two steps in the process are to conduct analysis of the application to create a behavioural profile. We use CopperDroid for these two steps. The third and fourth step is to extract feature sets from the behaviour profiles and perform a clustering of these. These two steps are carried out by ClusTheDroid. Figure 1 shows how ClusTheDroid and CopperDroid fits into the entire process.

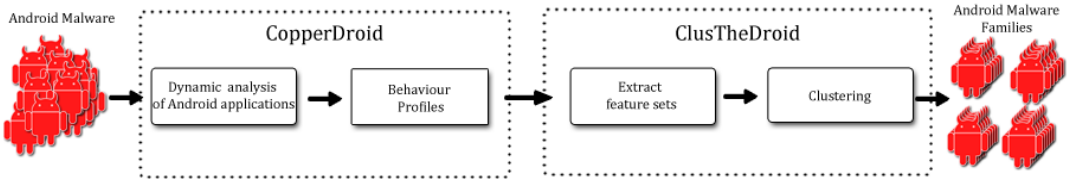

Figure 1 Overview of the entire process of clustering Android malware into families.

CopperDroid

CopperDroid (http://s2lab.isg.rhul.ac.uk/papers/files/ndss2015.pdf) performs out-of-the box behavioural analysis of Android applications to reconstruct malicious Android behaviours. CopperDroid consists of an emulator and an analysis component. The emulator executes the Android system and the analysis component conducts analysis of data from the emulator. As many malicious behaviours are guarded by the IPC and RPC mechanisms of Android, CopperDroid effectively tracks and dissects communication happening over these channels. As such, CopperDroid is able to reconstruct both low-level OS- (e.g., file creation, networking operations) and high-level Android-specific (e.g., sending an SMS, placing a phone call) behaviours, through a system call-centric analysis. CopperDroid furthermore proposes a solution to the code-coverage problem apparent in dynamic analysis tools, by allowing injection of artificial Android events. These events are both 'high-level' events such as sending SMS and performing incoming calls, to 'lower-level' events such as location updates.

Feature Set

The output of CopperDroid provides a detailed report with the reconstructed malicious behaviours of an application. We need to extract features from these reports and structure the data into a suitable format for clustering, which usually operates on vectorial data. We define an algorithm that will take as input a behaviour profile from CopperDroid, and return a feature vector as output.

We use all the malicious behaviours that CopperDroid can reconstruct as an entry in our feature set. Furthermore, for each of the sub-behaviours that have attached invocated methods as attributes we define the specific methods as features also. There are in the entire data set, 19 different reconstructed method invocations. Our feature set can be viewed as a hierarchy comprising three layers. At the top layer are all the high-level behaviours such as Access Personal Info and Network Access. At the middle layer are behaviours that are a subclass of a high-level behaviour such as SMS and Phone, and at the bottom layer are all the IPC and RPC methods that are invoked through Binder. For further information about the specific features we refer to the entire thesis which will be published on the Royal Holloway, Information Security Group's website.

Evaluation and Results

Setup. For our experiments, we used a data set of behaviour profiles produced by CopperDroid on the malware samples from the Malgenome project. The data set was given by the authors of CopperDroid, and there were a total number of 1,137 samples divided into 49 families. In the data set, 152 samples did not have any reconstructed malicious behaviours, effectively reducing the reference set to 985 samples. Also, when conducting our cluster analysis we required that each sample must have a feature vector with an

accumulated weight of at least 5. After this reduction, our effective data set comprises a total of 939 samples divided into 47 different families.

Measurements. We start with the set of 939 individual samples and put them into clusters as we go along. At each point we can measure how well we group the samples into families using the following measures:

- **Jaccard similarity.** As similarity measure between feature sets we use the Jaccard similarity, $J(A,B)=\frac{A \cap B}{A \cup B}$. For example, a Jaccard similarity score of 0.63 between two applications roughly means that they share 63% of their exhibited behaviours.
- **Precision and recall.** These validate the quality of our cluster algorithm. Precision measures how well malware from different families are put into different clusters, and recall measures how well malware from the same family are put into the same cluster. A precision of 1 means that there is no cluster with malware samples from two different families, and a recall of 1 means that every malware sample from a given family is placed in a single cluster. Initially, the precision will be 1.0 because all the applications reside in clusters containing only a single application, so there is no cluster with mixed families. The recall value will be close to 0 since there is only a single application in each cluster i.e. every family is split amongst many clusters. At the other end, if all the samples are put in one cluster then the recall value will be 1.0 since no family is split, and precision will be minimal since all the families are in one cluster. The aim is therefore to reach a point in the cluster run where *both* precision and recall are maximized. However, in a real-world scenario we are not able to compute precision and recall because we do not know the ground-truth of which family a specific application belongs to. This is exactly why we need a way to determine near-optimal clusters without knowledge about precision and recall, and as we show in our experiments, the C-index works surprisingly well at this.
- **C-index.** The C-index tells us when to stop clustering. It is an internal index that uses the overall distances between points in the data set and the distances between points internally in the clusters, to determine near-optimal clusters. It is a value between 0 and 1 where lower values indicate near-optimal clusters. It allows a flexible approach to determining the optimal cut when clustering unknown malware.

Experiment 1. Our first experiment was simply to run our clustering algorithm on the entire 939 samples. We calculate the C-index, precision and recall for every possible cut during the cluster run. Figure 2 shows the values of the three measures as a value of the combination similarity.

There are two interesting cuts in the cluster run: the cut where precision and recall intersect and also the cut where the C-index indicates near-optimal clusters. At the point where precision and recall intersect, when the similarity threshold (combination similarity) is 0.63, our cluster algorithm achieves a precision of 0.74 and recall of 0.73 and produces 81 clusters. This means that approximately 700 of the malicious applications are put well into clusters that resemble their families identified by Zhou and Jiang. The combination similarity of 0.63 means that any two applications in the same cluster have a Jaccard similarity score of at least 0.63. Therefore, a malware analyst who was to analyse these 939 applications would know that the applications in a cluster exhibit similar behaviours, potentially reducing

analysis to 81 applications (as we have 81 clusters) rather than 939. Also, it is possible, through minor analysis of the clusters, to extract the behaviours that the applications of each cluster have in common. This will be of great aid to the malware analyst during manual inspection of the applications, as the analyst will have a strong indication of what to look for in the malicious application.

A few cuts before precision and recall intersect, the C-index shows a significant increase of value, jumping from 0.1 to 0.2. Interestingly, there is also a significant decrease in the value of precision as the C-index increases. Table 1 shows the exact values of similarity threshold, precision, recall and C-index at these two particular cuts.

Table 1

Similarity threshold	Precision	Recall	C-index	No. Of clusters
0.6428	0.7859	0.6911	0.1006	89
0.6415	0.7529	0.6986	0.2093	88

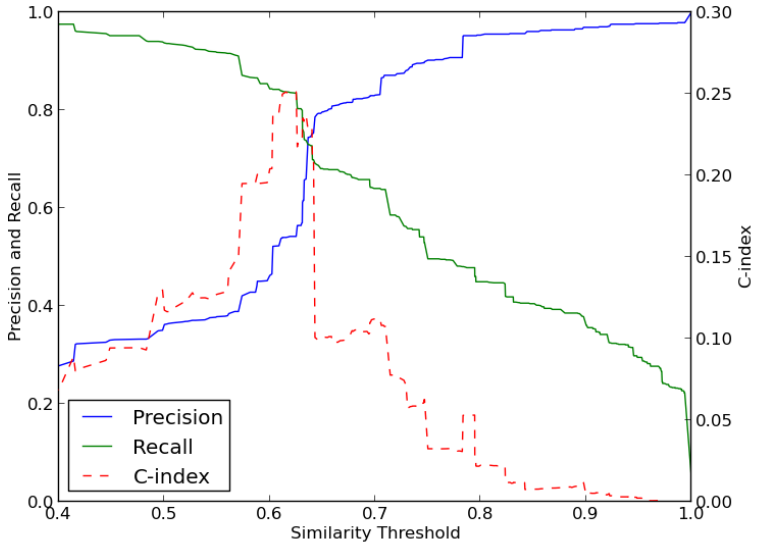

Figure 2 The values of precision, recall and C-index plotted as values of the similarity threshold for the possible cuts in experiment 1.

Experiment 2. In our second experiment we performed a division of the samples in the data set, which is agnostic to any knowledge about the malware samples, in order to increase the accuracy of our clustering algorithm. We observed that the there is a large interval of file sizes in the output of CopperDroid, also between inter-family samples. We therefore split the data set into two groups: samples that have a behaviour profile of less than 1MB and samples which have a behaviour profile of size equal to or larger than 1MB. The results from our cluster algorithm with the set of smaller files are shown in figure 3. As in Experiment 1, we look at the two interesting points. Our cluster algorithm produces 70 clusters with a precision of 0.77 and recall of 0.78 when the similarity threshold is 0.63. Again, the C-index indicates near-optimal clusters are to be found a few merges before precision and recall intersects. Table 2 shows the measurement values at the point of the sudden increase of the C-index.

Table 2

Similarity Threshold	Precision	Recall	C-index	No. Of clusters
0.642857	0.839080	0.725575	0.096968	85
0.641509	0.794540	0.725575	0.204316	84

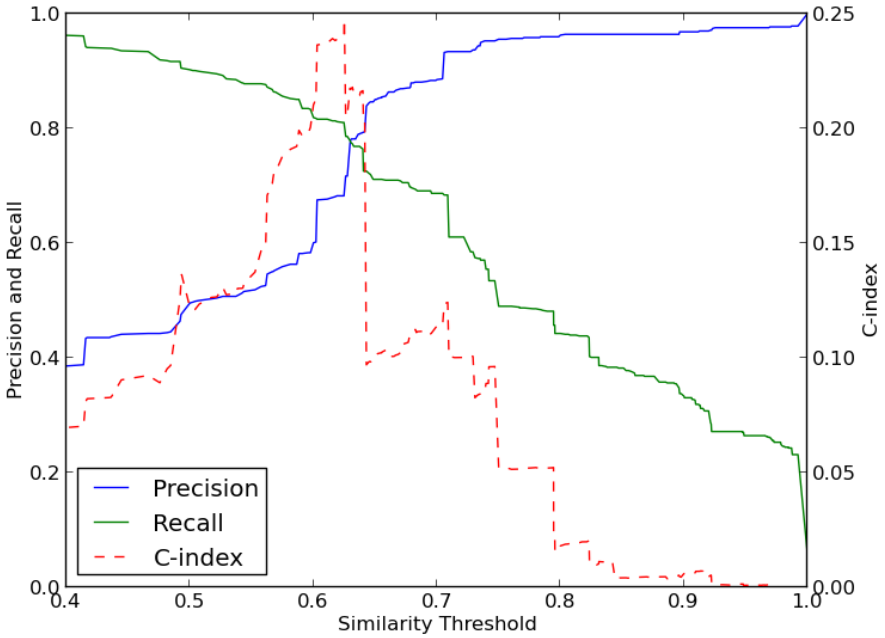

Figure 3 The values of precision, recall and C-index plotted as values of the similarity threshold for the possible cuts in experiment 2 with small files.

We also ran our clustering algorithm on the set of large files. The results from this run are shown in figure 4. Our clustering algorithm produces 39 clusters with a precision of 0.8 and recall of 0.8 at the cut when the similarity threshold is 0.75. In this cluster run we also see a strong correlation between the C-index and optimal values for precision and recall. Again there is a sudden increase of the C-index between two merges when the similarity threshold is around 0.58. Table 3 shows the two potential cuts of which there is more than a doubling of the value in the C-index.

Table 3

Similarity Threshold	Precision	Recall	C-index	No. Of clusters
0.581395	0.7720	0.9116	0.0136	21
0.574074	0.5721	0.9116	0.0290	20

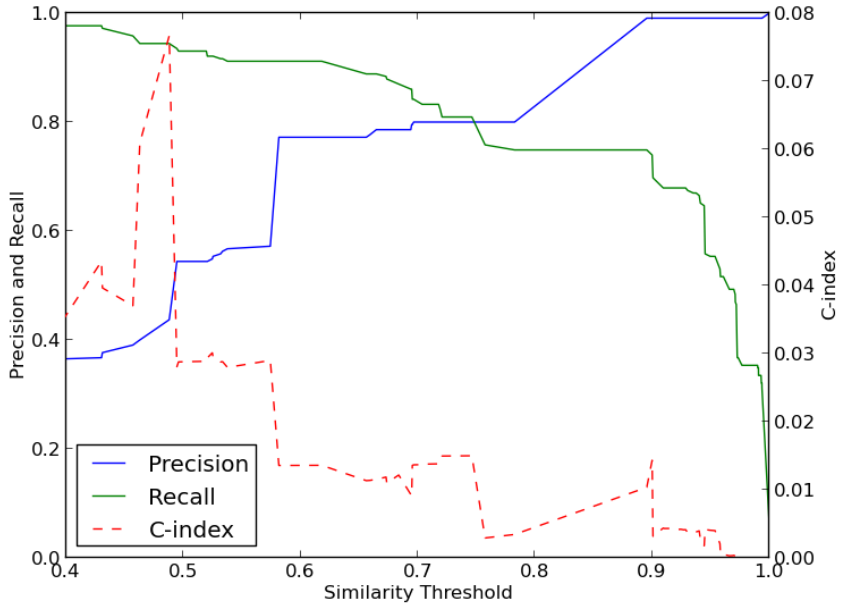

Figure 4 The values of precision, recall and C-index plotted as values of the similarity threshold for the possible cuts in experiment 2 with large files.

Conclusions

In this article, we present ClusTheDroid, a system for clustering Android malware into families based on profiles of reconstructed malicious Android behaviours. The input to our system is a set of behaviour profiles delivered by the dynamic analysis tool CopperDroid, and the output is a partition of the set so that subsets share common behaviours. Specifically, our system begins with an extraction of feature sets from the provided behaviour profiles and continues with a hierarchical agglomerative clustering of these.

The authors believe that the key to improving the accuracy of ClusTheDroid is a more fine-grained feature set. Therefore, the main part of our ongoing and future work is research into development of this. Furthermore, the authors are working to extending the implementation of ClusTheDroid, in order to make it capable of automatically extracting information about the exact behaviours that cause samples in a cluster to be similar. Ideally, this would make ClusTheDroid capable of automatically generating signatures that can be used for Android malware detection.

Biographies

David Korczynski graduated in 2014 from Royal Holloway University of London with an MSc in Information Security and received the prize for the most outstanding MSc project within the Information Security Group (ISG) that year. He is currently a DPhil student at the University of Oxford where the focus of his doctorate is to investigate how formal methods can be used to reason about malware, with the aim of developing novel malware detection techniques.

Lorenzo Cavallaro is a Senior Lecturer of Information Security in the ISG at Royal Holloway University of London. His research focuses largely on systems security. He has founded and is leading the recently-established Systems Security Research Lab (S2Lab) within the ISG, which focuses on devising novel techniques to protect systems from a broad range of threats, with the ultimate aim of building practical tools and providing security services to the community at large.