Yellow Planet

Johan Donkersloot, Arthur Bulan, Geoffrey Friesen

# Abstract

With the advancement of cheaper production of faster hardware, mobile devices are able to perform tasks that only computers were able to do in the past 5 years. With the convenience that modern mobile devices can provide there are new threats to personal information on the device. Number of detected threats has been on the rise in the past 5 years. In 2015 there was 884,774 new malware detected by the Kaspersky lab (compare to 295, 539 in 2014). Much research has been done on the topic of prevention of malware on the devices like Intrusion Detection Systems (IDS). This paper describes one of the implementations of an IDS according to Majeed et. al[4] who describes an IDS that uses devices features like RAM, CPU, Network usage to generate a normal usage profile and perform real-time local analysis on the features to detect features that use unusual amounts of the devices resources. This can indicate a possible intrusion into device and unwanted behavior. The implementation of this type of IDS describes in this paper performs K-means algorithmic analysis to generate centroids of data with a threshold radius. If at any point any of the resource monitored cross the threshold the IDS will alert the user of possible intrusion. This IDS can protect users from threats such as battery depletion attacks, DDoS attacks, and other malware that may use up resources.

Table of Contents

Abstract

I. Introduction

II. Related Works

A. Behaviour Based Anomaly Detection

B. Crowdroid

C. Andromaly

D. Active Platform Security

III. Implementation

A. Overview

B. Feature Extraction

C. Feature Storage

D. Profile Creation

E. Anomaly Detection

F. Testing

G. Results

IV. Citations

# I. Introduction

Mobile devices have evolved from being able to only send SMS messages and phone calls to 64-bit computers with 3G or 4G Internet connection, Bluetooth, Global Positioning Systems (GPS) and other systems. Because of this advancement mobile phones are able to make daily tasks such banking or online shopping more convenient. They gather volumes of personal information on their users, such as passwords, credit card numbers, and personal messages. With this convenience however, the task of ensuring confidentiality, availability, and integrity on the device becomes more important.

As mobile devices become increasingly accessible around the world, security becomes a larger concern. The volume of mobile malware has tripled in 2015; 884,774 new malicious programs were detected by Kaspersky Lab, a three-fold increase on 2014 (295, 539). To address this concern, research has been done in intrusion detection systems (IDS) that will look for specific signs of intrusion such as a file signature on the device and alert the user of the possible threat.

As it stands there are two main types of Intrusion Detection Systems, rule-based, and anomaly based. The first type provides a low false positive rate however it fails to detect new types of malware as it depends on historical data like malware file signatures detection. However, the anomaly based system looks for usage anomalies on the device itself on so it is independent of historical data and is able to discover novel threats.

This paper introduces an implementation of such an IDS that incorporates machine learning in a form of a K-means analysis on the usage features of the phone like CPU, RAM, Network activity etc... to build a regular usage profile of the device and be able to to detect any outliers which indicates unusual behavior on the phone.

To design the system, the key observation that was made is that malware uses up devices resources such RAM, CPU, network packets in and out. We can use this observation to create an Intrusion Detection System which extracts devices features, stores the data, creates a usage profile, and detects any anomalies on the device. In our testing we have the algorithm to successfully detect a simulated DDoS attack which rule-based IDS would not have detected.

# II. Related Works

Due to the rise in malware in mobile operating systems, behavior-based anomaly detection and intrusion detection systems are frequently researched topics. Detecting malware on Android devices, in particular, started with signature-based misuse detection where malware is detected based on its footprint or signature and only previously discovered malware can be detected. Once the rate of new malware started increasing, it became clear that misuse detection was unsustainable and that behavior-based anomaly detection was the only clear solution to the malware problem.

## A. Behaviour Based Anomaly Detection

Majeed et. al[4] proposed a framework for anomaly detection that uses a large number features to compare a complete picture of the device’s activity at the sample time to historical data. This project was limited by the fact that the behavior profile, once created, is static. The user has the responsibility to send the first 2 weeks of data to the researchers for additional processing before being sent back to the user. If a user’s behaviors change dramatically (e.g. new job, new game, etc.), the profile may not be an accurate representation. Another limitation of this approach is that it samples 19 features for analysis. The sheer size of the data will strain a resource constrained environment such as an Android device.

## B. Crowdroid

Burguera et. al[5] introduced an anomaly detection system for android devices that relied on aggregating and processing data from multiple users of the feature collection service. Crowdroid builds a profile of user interactions with each application individually, and as such is able to identify potentially malicious apps. Due to the crowdsourced nature of Crowdroid, it is a network-based intrusion detection system.

## C. Andromaly

Shabtai et. al[6] developed an anomaly detection system for Android that set itself apart from other systems in that has an active response to any perceived malware. All other related systems in this article, including Yellow Planet, merely notified the user of the anomalous behavior. The active response performed by Andromaly include: uninstalling the malicious application, stopping a process, turning off radios, encrypting data, and changing firewall policies among others. One other unique facet of Andromaly is that each set of extracted features is analyzed by different processors for each category of threat. A feature set will be analyzed by a processor focused on detecting worms separately from being analyzed for viruses.

## D. Active Platform Security

Sebyala et. al[7] proposed an Android intrusion detection system using naive bayesian networks for anomaly detection. The use of bayesian networks sets this solution apart from others discussed in this article as all of the others use some variation of K-Means to compare the extracted features to previous extractions. The main drawback of this system is that the bulk of the calculations of the bayesian network is handled in Matlab. While it would be possible to perform the calculations on-device, this type of analysis is currently not suited for the resource constrained environment in question.

# III. Implementation

## A. Overview

In our anomaly detection system, we attempt host-based anomaly detection. This means that extracted feature data is stored on the phone as well as processed on the phone. This approach allows us to not rely on server processing of extracted features.

Removing the need for a server to process data, Yellow Planet allows users that are not connected to the internet to still enjoy the protection of anomaly detection. Additionally, removing the server removes some attack vectors from hackers, such as tampering with the data sent to and from the server. Host based detection makes anomaly detection more secure and more accurate.

## B. Feature Extraction

Feature extraction is retrieved from system files that are continually updated by the kernel during system operations. Specifically, the files in question are the proc system files. These files are updated in real time, such that when we extract data from these files, the information is as up to date as possible. The following files are queried for system information: Proc/Stat, Proc/Mem, and Proc/Net. The battery feature is queried from the android API.

Feature extraction is completed every 5 minutes. We chose 5 minutes intervals due to the limited battery power on the intended devices. Monitoring features too often would result in a noticeable loss of battery life. Additionally, more feature extractions mean more data that must be saved on the device, depleting the device’s limited hard disk space. On the other hand, monitoring features too infrequently would result in even bigger problems.

If features were monitored infrequently, for example every hour, the anomaly detection would lose such an amount of precision that the false positive and false negative rates would likely skyrocket. If an attacker issues a command that may take 5 minutes to complete, the traces of the attack in the feature extraction may be lost among the sheer volume of data collected in one hour long feature extraction.

Another problem with monitoring data too infrequently, is that a weaker profile is built for the user. While most phone users check their phone 85 times a day, most do not use it for an extended period of time[8]. Because of this usage pattern, by making feature extraction infrequent, an unrealistic profile is created for the user. Perhaps a user only used their phone for 3 minutes out of the hour extraction, however the profile would lose that precision, and would not be able to tell between a user using their phone for a high processing application for a short time, or a user using their phone a little for an entire hour.

The final problem with a infrequent feature extraction is the time that it would take for Yellow Planet to warn the user of a possible anomaly. In the worst case, with an hour long feature extraction time, Yellow Planet would warn the user of an anomaly a full hour after the anomaly had taken place. While it is still good that a user is warned of any anomaly, even ones that happened previously, an hour may be too long for the user to respond to, or mitigate the current attack.

It is for the above reasons that we have chosen a 5 minute interval of feature extraction. We feel 5 minutes is an appropriate balance between preserving system resources while still painting an accurate picture of user usage to compare anomalies against.

## C. Feature Storage

Features are stored in an SQLite database on the host device. Features must be kept in an SQL database in order to preserve state when a phone is powered down, or the application is closed. Features are saved with two other attributes, a timestamp as well as a time slice.

The timestamp attribute of the feature SQL entry denotes the time that the feature set was extracted. The timestamp is in an epoch format, that is to say the timestamp is the number of milliseconds since January 1st 1970. The timestamp is important in order to clear out feature sets that are too old. This concept will be explained in more detail in the profile creation section of this paper. Additionally, this timestamp allows us to calculate the day of the week, which may be used for more precise profile creation in the future.

The time slice is an attribute in order to easily get all entries from the same time of day. An timeslice is calculated by dividing the day into equal time chunks, equal in length to the feature collection interval. Since Yellow Planet uses a 5 minute feature collection interval, each time slice would be 5 minutes long. Therefore, time slice zero would be 12:00 am, timeslice 1 would be 12:05 am, slice 2 would be 12:10 am, and so on. This allows us to query the database for a time slice that correlated time slice and retrieve all elements for that time of day.

## D. Profile Creation

User usage profiles are created from feature extraction sets. The usage profile is used to compare new feature sets against in order to detect anomalous behavior that deviates heavily from the usage profile. Usage profiles are created from the past two weeks of feature collection sets. Only feature sets that are not considered anomalous update the user’s usage profile. If there is less than a week worth of data to create a profile, all feature sets are treated as non-anomalous and automatically update the usage profile.

Usage profiles only take into account the past two weeks of feature sets in order to accommodate a continually updating profile. While Yellow Planet does not expect radical changes to occur in user behavior, we do expect slight changes in usage over time. By throwing away feature sets that are more than two weeks old, we can allow the user’s profile to change over time, while remaining relevant to the user’s current usage. New feature sets would have a larger impact on overall usage data with a two week rolling profile, than with an profile that never throws away data. If Yellow Planet never threw away feature sets, a change in user behavior may never truly impact the usage profile due to the sheer amount of data. Additionally, by only keeping two weeks of data at a time, Yellow Planet limits the amount of storage required from the device in order to store feature sets. For an entire feature set collection of every 5 minutes for two weeks, the features would only take 209 kb of storage.

All features extracted within the first week of extraction are all treated as non anomalous data sets, and automatically update the usage profile. This is required so that a usage profile can be created in the first place. When features are first collected, there is nothing to compare them to, and therefore no way to tell if they are anomalous or not.

## E. Anomaly Detection

Anomaly detection is done using K-means clustering and centroid calculation.

A distance is calculated from the centroid in order to find a threshold distance. New feature sets are compared against the threshold. If a feature set is above the threshold, it is treated as an anomaly, otherwise it is incorporated into the usage profile.

K-means is a algorithm in which to take data and plot the data into clusters on a graph. Since k-means needs points with both an X and Y values, features are paired for k-means points. The following Feature pairs are used: (CPU,Memory), (Battery, CPU), and (Network In, Network out). All feature pairs for the current time slice are plotted on a graph, then k-means calculates the centroid.

Feature sets were paired in order to create depth to the points on a graph. Time could not be used as an X axis points, since the time slice for all data points would be about the same. This would simply create a line of points, which would inhibit the extraction of the centroid of data. Additionally, by pairing features together, the number of times k-means must be calculated is cut in half. Instead of having to calculate k-means 5 times(once for each feature), we only need to calculate it 3 times per feature collection. This saves some battery usage by saving some calculations the CPU would otherwise need to make.

Features are paired based on features that go together. For example, CPU usage and memory usage features are paired together because when one is high, you would expect the other to be high as well. Since a centroid is calculated from the center of the data, it is important that points are not polar opposite due to the fact of the average being less accurate.

In order to calculate a distance from the centroid to act as a threshold, Yellow Planet uses 150% of the standard deviation. For example, if an feature pair’s centroid was (20,30) with an x and y standard deviation of 5, then the threshold point would be (27.5,37.5). A distance is calculated from the threshold point by connecting the two points with a line, creating a right triangle and calculating the hypotenuse.

We chose 150% of the standard deviation for the threshold for two reasons. First, we wanted to give the user a little leeway in order to slightly change their usage habits. Due to the evolving profiles that Yellow Planet employs, a chance for users to slightly change their usage behavior is a good thing. Secondly, we want to limit the amount of false positives that Yellow Planet reports. Anomaly detectors tend to struggle with false positive rates, so by giving new data points a good chance to not be considered anomalies, we hope to cut the false positive rates.

With a potentially lower false positive rates, there is also the chance for a higher false negative rate. However, since Yellow Planet specifically focuses on botnet style instructions, most zombie devices will heavily deviate from normal behavior when used by the bot herder. Because of this likely heavy deviation, we feel confident that Yellow Planet will have a lower false positive rate while still correctly identifying anomalous behavior.

## F. Testing

Testing of Yellow Planet proved to be difficult to test for several reasons. First, the time to create a full weeklong profile was not possible with the time that was available for testing. Secondly, due to limitations of available hardware, Yellow Planet has only been tested on an android emulator thus far. Finally, an legitimate malicious application was not available to us in order to test Yellow Planet.

The profile creation time was a large obstacle to overcome in testing. A weeklong profile creation was not possible, and as such, Yellow Planet would have to tested by simulating a week’s worth of feature collections.

In order to simulate the weeks worth of feature collections, we needed to collect features much faster than intended. In our testing, we will map the time slices that usually span one day to instead span one hour. This means that instead of time slices being 5 minutes long, they instead became 20 seconds long. Additionally, we needed to sample 7 times per time slice to simulate a full week of data. This means features were actually extracted every 3 seconds in our test.

Hardware problems also hampered the testing of Yellow Planet. We only had a Pantech Element tablet running android 4.0.4 for testing. This android tablet appeared to be incompatible with the intents we were using to start feature extraction once the phone has been turned on. Due to this problem, as well as having limited testing time, Yellow Planet has only been tested on an android emulator through android studio.

Yellow Planet is focused around detecting mobile botnet activities, however we were unable to find a legitimate mobile botnet to infect a test device with in order to test Yellow Planet. As a substitute we created a program that simulated what network traffic might look like if an device was forced to participate in a DDoS attack by a botnet. In order to do this, we simply created an application in which the device would send a large amount of data in a never ending loop to a non-existent address. The target of the devices data needed to be not real in order to prevent actually sending unwanted data to a real server. This application is what we used to see if anomalous system behavior could be caught by Yellow Planet. If Yellow Planet surfaces a warning to the user on the next feature extraction the test would be considered successful. If a warning is not surfaced, Yellow Planet has failed and the test is unsuccessful.

## G. Results

The results of the Yellow Planet testing was based off of the testing outlined in the previous section. A warning was successfully surfaced to the user on the next feature extraction once the pseudo DDoS attack had been started. There are several considerations to make about the positive results of Yellow Planet.

First, the standard deviation of the points used in testing is unrealistically small. Due to the fact that features were being extracted 3 seconds from each other, instead of an entire day, feature extraction points were very very similar to each other. This created a smaller range in which a centroid could be, which in turn made the anomaly detection more sensitive. Although we believe that Yellow Planet would still be successful with data points a day apart, we nevertheless believe that the frequent sampling did aid in detection of the pseudo DDoS attack.

Secondly, there was much less data being extracted in the test 3 second feature extractions in comparison to a real 5 minute

extraction. Since 5 minutes gives a much shallower picture of system resources when compared to 20 seconds, it is possible that the DDoS attack may be loss among the data from the entire 5 minutes of extraction. This is especially true if the DDoS attack does not continue for an extended period of time. Again, we feel that Yellow Planet would have also succeeded under normal circumstances, but these points still need to be considered.

Thirdly, the pseudo DDoS attack that we simulated in order to test Yellow Planet was not a real world test. Although we believe our simulated test would be close to the sort of system resources that would be consumed by a device participating in an DDoS attack, we cannot say that they would be exactly the same. Due to this, we cannot say with 100% certainty that Yellow Planet would successfully catch a real DDoS attack launched from a real botnet

# IV. Citations

1. Aubrey-Derrick Schmidt, Frank Peters, Florian Lamour, Christian Scheel, Seyit Ahmet Çamtepe, and Sahin Albayrak. 2009. Monitoring smartphones for anomaly detection. *Mob. Netw. Appl.* 14, 1 (February 2009), 92-106. DOI=<http://dx.doi.org.offcampus.lib.washington.edu/10.1007/s11036-008-0113-x>
2. A. Saracino; D. Sgandurra; G. Dini; F. Martinelli, "MADAM: Effective and Efficient Behavior-based Android Malware Detection and Prevention," in *IEEE Transactions on Dependable and Secure Computing* , vol.PP, no.99, pp.1-1. doi: 10.1109/TDSC.2016.2536605 URL: <http://ieeexplore.ieee.org.offcampus.lib.washington.edu/stamp/stamp.jsp?tp=&arnumber=7422770&isnumber=4358699>
3. Pedro Martins. 2015. K Means Clustering Java Code. URL: <http://www.dataonfocus.com/k-means-clustering-java-code/>
4. Khurram Majeed. Behaviour based anomaly detection system for smartphones using machine learning algorithm. Diss. London Metropolitan University, 2015.
5. Burguera, Iker, Urko Zurutuza, and Simin Nadjm-Tehrani. "Crowdroid: behavior-based malware detection system for android." Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices. ACM, 2011.
6. Shabtai, Asaf, et al. "“Andromaly”: a behavioral malware detection framework for android devices." Journal of Intelligent Information Systems 38.1 (2012): 161-190.
7. Sebyala, Abdallah Abbey, et al. "Active platform security through intrusion detection using naive bayesian network for anomaly detection." London Communications Symposium. 2002.
8. MailOnline, Victoria Woollaston for. "How Often Do YOU Check Your Phone? Average User Picks up Their Device 85 times a Day." *Mail Online*. Associated Newspapers, 29 Oct. 2015. Web. 06 June 2016.
9. http://www.kaspersky.com/about/news/virus/2016/The\_Volume\_of\_New\_Mobile\_Malware\_Tripled\_in\_2015