On-device Anomaly Detection for Resource-limited Systems

ABSTRACT

With the rapid evolution of small scale embedded systems like Smartphones, mobile malwares are becoming more and more sophisticated and dangerous. Most existing anti-malware solutions are based on complex algorithms that require a lot of system resources. Unfortunately these solutions cannot be hosted in small-scale devices. This paper focuses on the usability of on-device anomaly detection algorithms. It proposes a detection framework for Android-based devices. The framework allows local and remote construction of normal behavior based on various anomaly detection algorithms applied to system calls traces generated by the execution of mobile applications. Traces and models are stored in compressed format using lossless compression technique that allows higher data density in order to increase storage area and reduce trace data transfer cost over the network. In addition, traces are exported to the remote server where they are analyzed and scanned using more complex algorithms. The experimental results presented in this paper provide reliable measurements that help defining the required trade-off between detection accuracy and resources consumption.

Keywords

limited-resources, profiling, anomaly detection, n-grams

1. INTRODUCTION

Intrusion Detection Systems (IDSs) are used to recognize and raise alarms for abnormal activities. There are host-based and network-based IDSs that monitor computer or network activities. IDSs are also used in small scale embedded systems (i.e. Smartphones) to detect anomalies and report suspicious activities based on the already created normal model. Small scale-embedded systems are rapidly evolving, creating several families of microprocessors that meet performance, power and cost for almost all application markets, from Beagleboard, released in 2008, with single-core processor clocked at 600 MHz and 128 MB of RAM [1], to latest powerful system on chip like Matrix TBS2910 released in 2014 with Quad-core processor clocked at 1GHZ each core, 2GB of RAM and 16GB of onboard storage [2].

Unfortunately, this popularity attracts malware authors too, which are continuing to grow at an alarming rate, from cabir in symbian phone in 2004 [3] to Android SMS malware in 2014 that infects about 1.2 Million users [4] . Android platform is today's most popular malware target, with nearly 2000 Android malware sample discovered every day [5]. Attack detection in such devices is difficult due to limited battery life, memory capacity, computing power, and bandwidth. Some companies have implemented embedded anti-malware solutions, but the majority doe not use the limited resources economically. Most security tools are based on complex detection algorithms in order to increase the detection rate and accuracy, such as AntiMalDroid [6] and DroidAPIMiner [7]. Some use machine learning [6] that requires high speed calculations, others use Data Mining [7] that involves huge data management. Although these approaches maintain high detection rate, they are not practical for small-scale devices like Smartphone. For example, Overo FE COM [8] and APC 8750 [9] have restricted resources with at most 512 MB of RAM and 800MHZ of CPU. Even the brand new Samsung Galaxy S5 with 16GB of on-board storage "has Less Than 8GB of Usable Storage" [10]. Considering these limitations, the following questions are raised:

- What is the best trade-off between security and usability? and how to adapt security mechanisms according to system resources availability?
- Can we apply more than one detection algorithm when we have enough resources?

To address these questions, we establish a base of knowledge about "normal" behavior, as models, in an initial learning phase. Then in the detection phase, once the behavior is too different from the normal behavior, it is considered abnormal. With this technique we can detect both known and unknown malwares, also it does not require either a huge signatures database or continuous availability of network. Therefore, in this paper, we adopt this technique for Host-based intrusion detection system on limited-resources devices. We construct a healthy model using traces that are collected for training purposes. During operation, the anomaly detection system attempts to detect events that deviate significantly from the expected normal profile. Although these deviations are considers as anomalous events, but they can generate large number of false alarms if appropriate thresholds are not picked for decision maker algo-

SAC'15 April 13-17, 2015, Salamanca, Spain.

http://dx.doi.org/xx.xxxx/xxxxxxx.xxxxxxx

rithm. This problem occurs when the built normal model is not representative of normal software system behavior.

After establishing a proper detection algorithm, we attempt to address the raised questions to guarantee both security and usability for resource-limited devices. We propose an energy-efficient malware detection scheme that handles this trade-off in light of multiple types of resources. The objective is to ensure good protection of a device even with limited resources, meaning establishing an adaptive detection solution based on profiling results. We develop three anomaly detection algorithms and we examine the impact of some factors such as the threshold of detection, the size of sequences and the length of patterns, on the performance of each algorithm on the device.

The remaining of the paper is as follows: we discuss related work in Section 2 and we describe attack scenarios in Section 3. Then we present our framework in Section 4. Experimental results and their analysis are detailed in Section 5. Section 6 concludes the paper and lists future work.

2. RELATED WORK

There are many anomaly detection algorithms for behavioral modeling like Hidden Markov Models [11, 12], Support Vector Machine [13], Neural Networks [14] and statistical profiling [15]. We chose those based on n-gram (an n-gram is a subsequence of n items from a data sequence), because they are the simplest possible algorithms that provide a lightweight and real time detection. Forrest et. al [15] built a model of normal behavior based on fixed-length sequences of system calls. Hubballi et. al [16] created a set of bins for different length of n-gram, and Kymie et. al [17] tried to find the optimal size of n-gram, which offers the best detection rate.

The implementation of these algorithms in small-scale systems with limited resources requires a good management solution. There have been a lot of work done in this field. Amontamavut et al. [18] suggest sending traces to a server via network and to perform a remote monitoring. His approach consists of dividing logging mechanism into two parts: one in the device side, which generates and collects log data; and the other in the server side, for storing, analyzing, and debugging which improves log data management system. Although this solution is able to bypass the limited log data storage in small scale systems, but it requires a continuous availability of the server, which is not guaranteed. WANG et. al [24] had used a detection algorithm based on compression in order to prevent huge size of log data, Amontamavut et. al [18] used BSF compressor in order to decrease log data collection/size overhead.

In this paper, we use lookahead model [15], n-gram tree model [22] and a modified version of varied-length n-grams model [23]. Unlike [18]'s approach, our management solution for resource-restricted devices covers local and remote host-based detection including a powerful compressor tool.

3. ATTACK SCENARIOS

The main security problem targeted by this paper is malicious update of legitimate Android applications. These updates can be performed with the knowledge and the approval of Smartphone owner or without his knowledge. In the first scenario, an application known to be safe will have news versions that include malicious code. This can be performed deliberately by some person involved in the development of the application or by repackaging it and making it available in alternative applications markets. These malicious updates can be then installed by the Smartphone owner. In the second scenario, after its installation on the device, an application downloads additional code without notifying the Smartphone owner. This can be scheduled by the developer of the application or by the providers of some external components (e.g., advertisement). In fact, these applications can be hijacked by cyber-criminals in order to display ads with fake updates for legitimate installed applications. This kind of attack was discovered in April 2013 with malicious updates for 32 applications [19].

To detect malicious update of legitimate applications on the device, it is important to perform real-time monitoring of their executions and detect the malicious updates when they happen. This is based on the assumption that these updates will inject malicious activities that will deviate from the usual behaviour of these applications before being infected.

4. FRAMEWORK DESIGN

In this section we introduce the proposed framework. It allows monitoring the execution of any application on the device. For each application, the Framework allows building a "normal" model based on the selected anomaly detection algorithm. Then each future execution of the application will be monitored to build a "detection" model that will be compared with the "normal" one. If the "detection" model is significantly different from the "normal" one, the framework raises an alert mentioning the abnormal traces. Building the normal model can be done locally or remotely. In both cases, the traces are submitted to a remote server that can build more complex models and perform heaviest anomaly detection algorithms. In addition, traces and models are stored in compressed format using lossless compression technique that allows higher data density in order to increase storage area and to reduce trace data transfer cost over a network. Traces are serialized before being exported to the remote server. As illustrated in Figure 1, the proposed framework consists of six major units: (1) data collection, (2) data processing; (3) scan and model management, (4) profiling, (5) storage, and (6) database and server connection serialization.

4.1 Data Collection

To build a representative model, we need to collect from the target system the data generated when executing the monitored applications. Collected data can be in the form of network traffic, battery consumption, memory usage statistics or system calls. In this paper we monitor system calls as they are provided by the kernel and used by programs running in user space. This is motivated by the assumption that once compromised, the observed system call traces of an application are different from those generated by its original version. This approach has been followed by various past work on detecting anomalies in servers [20]. To generate the execution traces resulting from running applications, we are using state of the art system instrumentation tools like Strace. Strace is installed by default on every Linux distribution (besides Gentoo). Once traces are generated, they are pretreated to be used as input for the chosen model. These traces can be filtered to keep only the most important ones in order to reduce the size of traces and models.

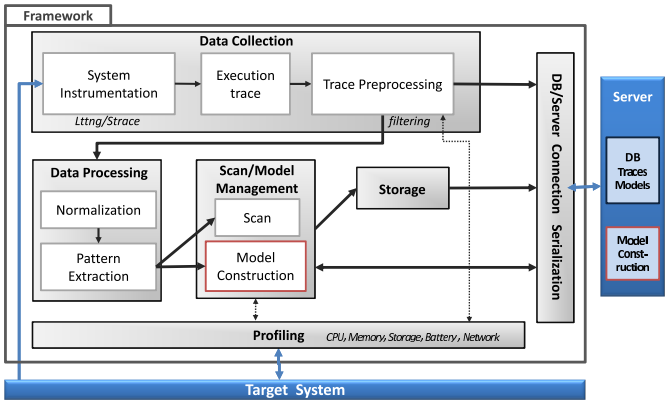

Figure 1: Framework's Architecture

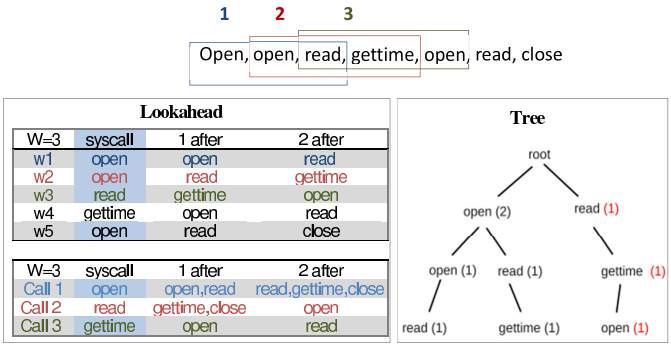

Figure 2: Lookahead and Tree Algorithms

4.2 Data Processing

To simplify the process of analyzing the data and creating normal behavioral model of the system, we preprocess the collected data and cut the generated traces to fixed size sequences. This size has an influence on scan accuracy (as we will see in section 5). For each fixed-size sequence of system calls, we extract features that are used by the anomaly detection algorithm. The extracted features are fixed and varied-length n-grams (based on temporal order of the system calls in the sequence) and the frequency of each call [21]. Then, the generated traces are converted to the appropriate format required by the used anomaly detection algorithm.

4.3 Model Management and Scan

In this unit, we create the normal behavioral representative of the system. The normal model is then used as benchmark while testing a trace to identify anomalies. To this end, the more complete the representative model is and the more accurate the anomaly detection algorithm functions, the better results we obtain in terms of true positive rate (true attack detection) as well as false positive rate (false alarms). We chose to define the normal profile using short sequences of system calls using n-gram language models. Indeed, a vast community of researchers demonstrated overs years the efficiency of analysing n-grams of system calls in detecting malicious use of legitimate applications (mainly services) [20]. As mentioned earlier, in this paper we use three different n-gram based algorithms for anomaly detection and we extract n-grams by sliding window over the training data using a static or dynamic window (regarding the size). In the following, we introduce the adopted anomaly detection algorithms and describe our approach in deploying them.

4.3.1 Lookahead

This model reproduces the work of Forrest [15]. This algorithm builds the normal behavioral profile by sliding a fixed window across the training normal traces. For example, for a window of size k=3, it records for each call the call that follows it at position 1, at position 2, and at position 3 in the window (see Figure 2). After crossing all the sequences, the resulting model contains all the normal subsequences of size 3. In detection phase, we slide a window in the same way as for the construction, and we compare the extracted pairs with those found in the normal model. A pair that is not in the normal model is considered abnormal. A trace is considered abnormal when its rate of abnormal pairs exceeds a certain threshold.

4.3.2 Tree n-gram

This model is a more complex version of the previous one, it is inspired by the work of Hubballi et al.[22]. It uses the same approach of sliding a fixed-length window across the sequence, but instead of regrouping them by position, this algorithm generates a tree of n-grams, where each node of a tree represents a system call in the form of a tuple $< a_i, frequency >$, where a_i denotes the system call in position i of a n-gram and *frequency* denotes the number of times the system call appeared in this position while sliding over the whole sequence in the normal model. Therefore, a branch of a tree contains a n-gram with the frequency of appearance for each system call. Figure 2 illustrates the generation of the n-gram tree for the same running example. In this algorithm, the common prefixes are merged. Therefore, this model forms an efficient storage structure.

The constructed tree represents the normal behavioral of the system. The tree is used to determine how frequent each n-gram is compared with all other n-grams appeared in the training trace. So in detection phase, we built a tree in the same way for the testing sequence. Comparing the test tree to the model tree identifies the anomalies. This is done by assigning each n-gram that appears in both trees the frequency rate previously calculated in the first tree, while a new n-gram which is not in the model tree is assigned a maximum fault rate. The maximum fault rate is presented in this algorithm. At the end, the mean rate computed from all appeared n-grams is compared to a threshold to raise an alarm for anomaly. In our improved version, we decided to classify the tree by number of occurrences so we always check in priority those nodes that appear most often. This is expected to reduce scan overhead and latency.

4.3.3 Varied-length n-gram

This algorithm is inspired by the work of Guofei et al. [23]. The idea is to extract only frequent patterns (n-gram) and to use a threshold (from 0 to 1) to control the generalization ability of the model, allowing various lengths of n-grams.

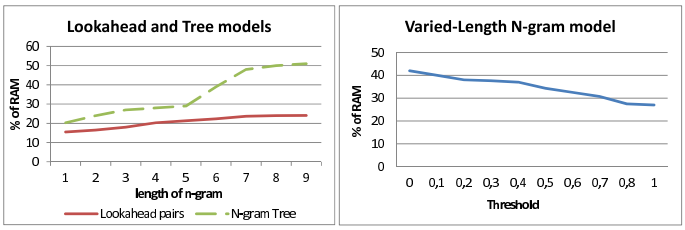

Figure 3: RAM overhead of creating a normal profile for Lookahead pairs, N-gram tree and Varied-length N-gram algorithms

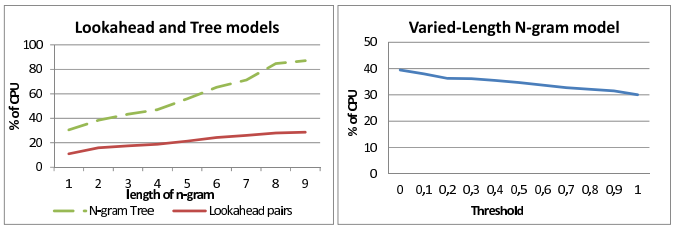

Figure 4: CPU overhead of creating a normal profile for Lookahead pairs, N-gram tree and Varied-length N-gram algorithms

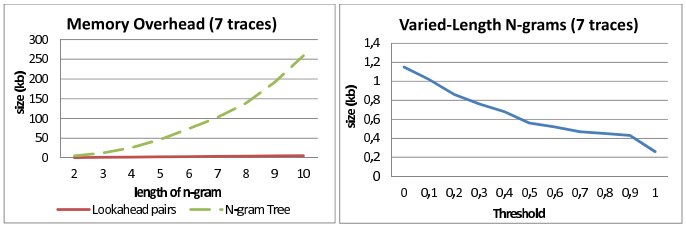

Figure 5: Storage overhead of creating a normal profile for Lookahead pairs, N-gram tree and Varied-length N-gram algorithms

This model allows to reduce the size of dataset while keeping a good detection efficiency. Normal behavioral profile is created according to "N-grams extraction algorithm" of [23]. This algorithm allows to create sets of varied-length n-grams by checking each time if $f(x_k+1) > \alpha \times min(f(x_k^i), f(x_k^j))$, while α is a fixed threshold (between 0 and 1) and $f(x_k^i)$, $f(x_k^j)$ are the frequencies of 2 n-grams of size k that constitute the n-gram x_{k+1} of size k+1. The output of this algorithm is used in [23] as input to an automaton, which represents the model, whereas we have used this algorithm to create our model in order to minimize its size. Modeling normal behavior with such algorithm makes detection phase less accurate, that is why we assigned a different abnormal threshold for each set of n-grams according to their size and we defined a new threshold from which the trace is considered as abnormal. So in the detection phase, we extract new sets of n-grams and we compare them to the model created in learning phase.

4.4 Storage

So far, we introduced three anomaly detection algorithms that are inspired from previous works. These algorithms are trained by normal traces and build a normal model that represents normal behavior of the system. Since we building a model per application, storing these models is an issue for limited resources devices like smartphone. To optimize the memory budget consumed for saving the generated normal models we adopted the open source higher compression tool "Zopfli" released by Google in 2013 [25]. Using "Loss less Compression" technique, "Zopfli" allows to perfectly reconstruct original data from the compressed data. It compresses around 3.7% better than any zlib-compatible compressor [25].

5. EXPERIMENTATIONS

We performed several tests on a Samsung Galaxy S3 Smartphone running Android 4.1.1 with 2GB of RAM and a dual core CPU running at 1.5 GHz. Monitoring in a real environment ensures that monitored applications are behaving as they behave on any phone and they are not likely to detect a sandbox-type environment where they can avoid to express their malicious behavior.

To start our experiments, we created normal behavioral profiles for three Android applications ("Angry birds space", "Candy Star", "Ninja Chicken") using execution traces generated by "strace". Each built model represents around 480,000 system calls. Figures 3, 4 and 5 show respectively RAM, CPU, and storage overhead of creating a normal model for each algorithm. In the left plots, the x-axis is the n-gram size which directly influences RAM, CPU, and storage overhead. The right plots illustrate performance of varied length n-gram algorithm that based on a threshold picks the common n-grams. Therefore, when increasing the threshold, the data to be investigated is also decreasing. Based on the obtained results, the Lookahead algorithm seems to be the lightest in terms of RAM and CPU consumption. The consumption even stays under 20% for n-grams of length 5 or less. At the opposite, from pairs of length 7, Tree model provides the highest RAM and CPU consumption. It reaches 51% of RAM usage and 58.4% of CPU usage while it drops down to 20% for both factors with only 2 pairs or to 30% for pairs smaller than 5.

Comparing to Tree model with pairs of length greater than 7, Varied-length N-gram model consumes less RAM and CPU that can stay under 34% with a threshold greater than 0.5 and 0.6 respectively. It can even, with a maximum threshold, drop till the value consumed by lookahead model with 10 pairs.

Figure 6 shows RAM and CPU overheads of scanning 3 applications in parallel. We note that in the worst case, using 10 pairs, we are able to scan even three applications at the same time using Lookahead (staying under 55% of RAM and CPU). However, we can perform 3 parallel scans with tree algorithm using only 2 pairs. Concerning Varied-length N-gram algorithm, it can provide up to 3 parallel executions with a threshold of 0.1 (its worst case). Theses results confirm that our Framework is able to lunch more than one detection algorithm without affecting system resources.

Both Lookahead and Varied-length N-gram models occupy a smaller disk space that stays under 5.6 KB. This seems low enough to store a model by application on the device. The size of these models is therefore proportional respectively to the depth of the model (the length of n-

gram considered to construct the normal model) and the threshold. In contrast, the size of the Tree model appears to increase polynomially reaching more than 250 KB, which is difficult to tolerate. In fact, if we store dozens of Tree models with 6 pairs, we quickly reach several MBs of data. However, if we limit to models with 2 or 3 pairs, their size remains acceptable.

According to these results, we note that the three models present different characteristics regarding resource cost. In the following, we investigate their performance for anomaly detections. For this end, we used already existing contaminated versions of the three applications. We installed these applications on the Smartphone, collected the generated execution traces and analysed them by each of the three studied algorithms. We measured the rate of detection using the following formula.

$$\frac{(number\ of\ abnormal\ Ngrams) \cdot 100}{total\ number\ of\ Ngrams}.$$

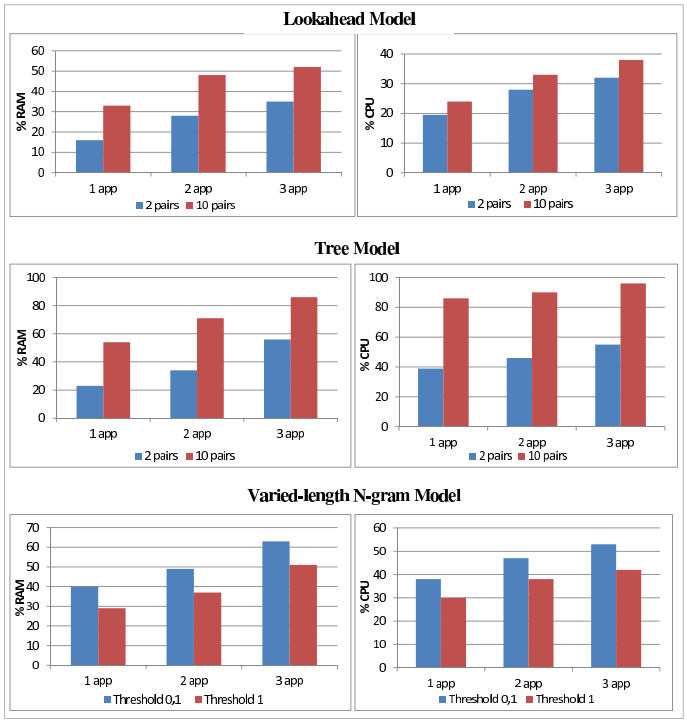

Figure 6: RAM and CPU overheads of scanning 1, 2 and 3 applications in parallel for each algorithm

The results are shown in Figure 7 per trace (rate for trace number 1, trace number 2, etc.) for Lookahead and Tree models using 400 traces, and in Figure 8 per number of used traces (rate using 1 trace, 2 traces, etc.) for Varied-length N-grams model using 100 traces. We note that there is a maximum rate in trace number 2. This reflects that this trace is abnormal as the length of used n-grams and the value of threshold influence on the consumption. For Lookahead model, the plotted curve using n-grams of length 2 is flatter than the curves plotted with n-grams of larger sizes. In addition, the peaks of anomalies are very similar between the curves drawn with 7 or more pairs. We therefore believe that limiting to 7 pairs provides approximately the same

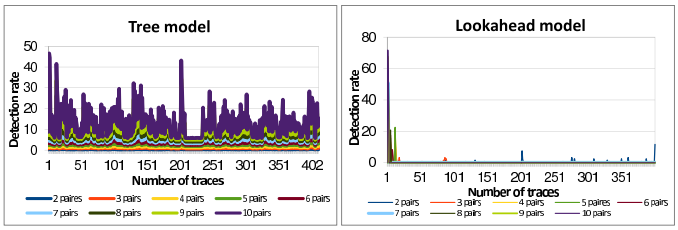

Figure 7: Detection rate of Lookahead and N-gram tree algorithms

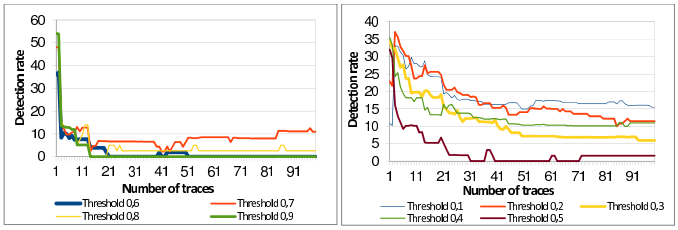

Figure 8: Detection rate of "Varied-length N-grams" algorithm

results as the use of a higher number. Lookahead model does not generate any false positive rate, which makes it more accurate.

The curves of Tree model have however almost the same gaits with some translation but with a different order of magnitude. Indeed peaks of anomalies are more visible for larger n-grams. However, the rate of false positive, described in Figure 9, increases when we increase the size of used n-grams, which makes the model less accurate. Detection rate

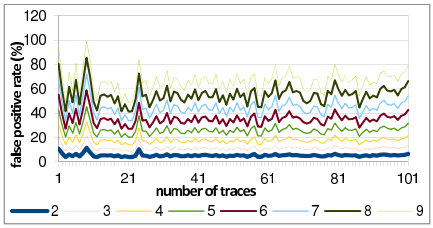

Figure 9: Percentage of false positive rate of "Tree" algorithm

of Varied-length N-grams model depends on both threshold and number of analyzed traces. Figure 10 shows how false positive rate decreases when we increase the number of traces and the threshold. In fact, the maximum threshold provides the least false positive rate that tends to 0 from the threshold 0.5. As the threshold is decreased the model generates more n-grams and then it can detect more cases, this explains why the detection curves made with smaller threshold are less uniform compared to other curves made with higher thresholds. We note that the threshold 0.5 can provide a certain detection rate with no false positive.

Each model has their advantages and disadvantages. Lookahead model has the advantage of having zero false positive

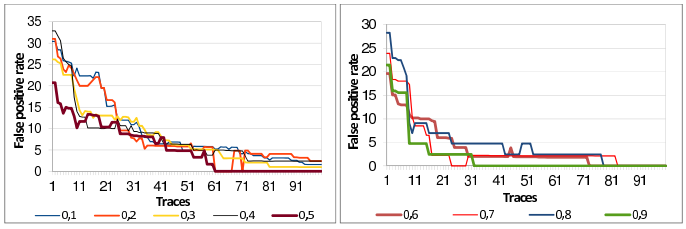

Figure 10: False positive rate of "Varied-length N-grams" algorithm

rate and the least RAM and CPU consumptions with a number of pairs lower than 5. However its detection rate increases by increasing the number of pairs, this will therefore increase its resource consumption until exceeding that of Tree model with 2 pairs. Varied-length N-grams model has the advantage of occupying very little disk space, even if a large number of n-grams is used (with lower thresholds). It also provides a good detection rate with the threshold 0.5 while consuming a considerable amount of CPU and RAM, which decreases by increasing the threshold. Moreover, the consumption of RAM/CPU and the detection rate seem to give better results for Tree model with 2 pairs, while occupying much space compared to Lookahead with 7 pairs. However, its size remains acceptable with data compression.

Based on the experimental results, a trade-off between accuracy and resource consumption can be established by varying the number of used pairs, the threshold of extracted n-grams, as well as the number of analyzed traces. So we adjust the value of detection thresholds for Lookahead with 7 peers to *'0.025'* and for Tree algorithm with 2 pairs to *'1.2'* (Figure 7). We adjust also the value of generating threshold to *'0.5'* for Varied-length N-grams algorithm. This has improved algorithms performance significantly and has provided a better accuracy (we tested detection accuracy with the three Android applications).

6. CONCLUSION AND FUTURE WORK

In this paper, we proposed an adaptive host-based anomaly detection framework that provides lightweight protection for limited-resources devices. Our approach allows detecting malicious functionalities injected silently to mobile applications without the knowledge of the device owner. We investigated Lookahead, Tree N-gram, and Varied-length N-grams algorithms. Further we tested our framework with normal Android applications and their malicious versions. The experimental results show that based on the size of n-gram, the value of threshold and the number of used traces, a good detection rate can be reached with low false positive rate while consuming a reasonable amount of resources. In future work, we plan to study other classes of anomaly detection algorithms. In addition, we will develop dynamic decisions maker that (1) monitors the state of system resources and (2) selects the detection algorithms that allow the best trade-off between detection accuracy and resource consumption. Finally, we plan to test more Android applications with various malicious repackaging techniques.

7. REFERENCES

[1] M. Frazier, *The BeagleBoard: $149 Linux System*, 2008. Available from: http://www.linuxjournal.com/content/beagleboard-149-linux-system

[2] S. Joly, *TBS2910 Mini PC ARM Matrix*, 2014. Available from: http://domotique-info.fr/2014/04/tbs2910-mini-pc-arm-matrix/

[3] E. Millard, *"Cabir: World's First Wireless Worm"*, 2004. Available from: http://www.technewsworld.com/story/34542.html

[4] J. ABHISHEK, *"Android SMS malware hosted on Google Play infects 1.2 Million users"*. Available from: http://www.hackleaks.in/2014/02/android-sms-malware-hosted-on-google.html

[5] Sophos, *"Mobile Security Threat Report 2014Ť*. Available from: http://www.sophos.com/en-us/medialibrary/PDFs/other/sophos-mobile-security-threat-report.pdf

[6] M. Zhao et al. *"AntiMalDroid: an efficient SVM-based malware detection framework for Android."* Information Computing and Applications. Springer Berlin Heidelberg, 2011. 158-166.

[7] Aafer, Yousra, Wenliang Du, and Heng Yin. *"DroidAPIMiner: Mining API-level features for robust malware detection in android."* Security and Privacy in Communication Networks. Springer International Publishing, 2013. 86-103.

[8] gumstix.com, *"Overoő FE COM"*, 2014. Available from: https://store.gumstix.com/index.php/products/256/

[9] apc.io, *"APC 8750"*, 2014. Available from: http://apc.io/products/8750a/

[10] Gary Ng, *"The 16GB Samsung Galaxy S5 Has Less Than 8GB of Usable Storage"*, 2014. Available from: http://www.iphoneincanada.ca/news/galaxy-s5-8gb-usable-storage/

[11] Warrender, C., Forrest, S., & Pearlmutter, B. (1999). *"Detecting intrusions using system calls: Alternative data models"*. In Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on (pp. 133-145). IEEE.

[12] Sultana, A., Hamou-Lhadj, A., & Couture, M. (2012, June). *"An improved Hidden Markov Model for anomaly detection using frequent common patterns"*. In Communications (ICC), 2012 IEEE International Conference on (pp. 1113-1117). IEEE.

[13] Jain, R., & Abouzakhar, N. S. (2013). *"Comparative Study of Hidden Markov Model and Support Vector Machine in Anomaly Intrusion Detection"*.

[14] Li, W., & Meng, Y. (2013). *"Improving the performance of neural networks with random forest in detecting network intrusions"*. In Advances in Neural NetworksŰISNN 2013 (pp. 622-629). Springer Berlin Heidelberg.

[15] S. Forrest, SA. Hofmeyr, and A. Somayaji. *"A sense of self for Unix process"*. In Proceedings of the 1996 IEEE symposium on research in security and privacy, Oakland California, pp. 120Ű128, 1996.

[16] N. Hubballi, S. Biswas, and S. Nandi. (2010). *"Layered Higher Order N-grams for Hardening Payload Based Anomaly Intrusion Detection"*. Availability, Reliability, and Security, 2010. ARES '10 International Conference on , vol., no., pp.321,326.

[17] M. C. T. Kymie and A. M Roy. 2002. *"Why 6?" Defining the Operational Limits of Stide, an Anomaly-Based Intrusion Detector*. In Proceedings of the 2002 IEEE Symposium on Security and Privacy (SP '02). IEEE Computer Society, Washington, DC, USA, 188-.

[18] P. Amontamavut, Y. Nakagawa, and E. Hayakawa. *"Separated Linux Process Logging Mechanism for Embedded Systems"* Embedded and Real-Time Computing Systems and Applications (RTCSA), 2012 IEEE 18th International Conference on , vol., no., pp.411,414, 19-22 Aug. 2012

[19] Panda labs. *"Panda Security Annual Report PandaLabs 2013 Summary"*, 2013. Available from: http://m.itcafe.hu/dl/cnt/2014-03/107032/pandalabs-annual-report-2013.pdf

[20] S. Forrest, S. Hofmeyr, and A. Somayaji. *"The evolution of system-call monitoring"* Computer Security Applications Conference, 2008. ACSAC 2008. Annual. IEEE, 2008.

[21] A. Amamra, C. Talhi, and J-M Robert. *"Impact of Dataset Representation on Smartphone Malware Detection Performance"* Trust Management VII. Springer Berlin Heidelberg, 2013. 166-176.

[22] H. Neminath, B. Santosh, and N. Sukumar. *"Sequencegram: n-gram modeling of system calls for program based anomaly detection"*. In Communication Systems and Networks (COMSNETS), pp. 1Ű10, Jan 2011

[23] J. Guofei, Chen. Haifeng, C. Ungureanu, and K. Yoshihira. *"Multi-resolution Abnormal Trace Detection Using Varied-length N-grams and Automata"*. International Conference on Autonomic Computing (ICAC 2005), pp.111,122, 13-16 June 2005

[24] N. WANG, J. HAN, and J. FANG. *"Anomaly Sequences Detection from Logs Based on Compression"*. arXiv preprint arXiv:1109.1729, 2011.

[25] J. Alakuijala and V. Lode. *"Data compression using Zopfli"*. Tech. rep. Google Inc., Feb.