Recurrent Neural Networks for Colluded Applications Attack Detection in Android OS Devices

Igor Khokhlov, Ninad Ligade, Leon Reznik
Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, NY, USA
ixk8996@rit.edu, nl4439@rit.edu, lr@cs.rit.edu

***Abstract*—The paper presents a design and an implementation of an intelligent detector of a novel "colluded applications" attack on user's privacy in Android OS devices, which employs recurrent neural network (RNN) models. The paper reports the results of an empirical study that involved the attack research, data collection and pre-processing, the choice of the RNN model for a detector design, multiple detector implementations, their performance evaluation and analysis, and finally, an Android app realization and execution on a real device. We investigate and analyze multiple attack scenarios and the attack influence on such technological signals as memory consumption and a CPU's cores clock speed. For the attack detection, a few detectors exploring multiple RNN models are designed, implemented, and examined. The detectors employ various RNN models, such as a simple recurrent neural network, a long short-term memory, and a gated recurrent unit. Each model's performance in detecting multiple attack scenarios is evaluated and analyzed in order to compare classification models against various criteria.**

***Index Terms*—Recurrent neural networks, anomaly based attack detection, Android device security.**

I. Introduction

There are nearly 3.2 billion smartphone owners in the world right now [1], with most of them connected to the Internet. In 2019, around 52.2% of all website traffic worldwide was generated through mobile phones [1]. These smartphones provide a vast computational power and have a gamut of embedded sensors. Many applications use these sensors and the computational power to provide users with outstanding functionalities and features. For instance, consider fitness tracking applications that use the accelerometer and GPS inputs to keep track of your fitness. According to the 2019 data [1], the share of Android phones in the global mobile market is continuously rising and is more than 87% of mobile platform market share. Moreover, this Android OS platform has around 2.47 million applications available; from this, we can imagine the sheer impact of this platform and the volume of data these devices might be generating. This massive volume of generated data attracts a lot of malevolent actors who create malicious applications to steal this data, as it may have substantial monetary benefits. Android OS mobile platform becomes a major battleground in the cyber-security domain [2]. New malware gets very sophisticated [3] that requires adequate detection techniques. In this paper, we present a novel approach to the detection of the "Colluded Applications" attack on the users' privacy that involves using computational intelligence (CI) techniques and investigate their feasibility and performance.

The Android operating system uses the Linux kernel at its core. Its security architecture is a bit different, however. It has redesigned some of the underlying operating system security policies to provide better application security, protect users' private sensitive data and system resources. Android implements these privacy policies through the application sandboxing, application signing, and the permissions mechanism. In addition to this, there are various authentication mechanisms and anti-malware tools provided by Original Equipment Manufacturers (OEM). These tools mostly use some computational intelligence approaches to detect threats. The security threat can be mitigated at the user level as well by using the applications from the authenticated sources and giving them only the required permissions. Unfortunately, users do not always pay attention to these choices. On top of this, even if a user sets permissions to the applications very cautiously, it does not guarantee an absolute data leakage prevention as applications can bypass this permission model by colluding with each other, discussed in detail in Section II.

Moreover, the maliciousness of the attacks is increasing daily, and these existing tools can detect only the existing threats. Hence, even after these security measures, Android is not as secure as it seems to be, and these security measures are breachable in various ways. One of the most seemingly benevolent but very harmful breach being the "colluded applications" attack.

The cyber-security community turned their sight onto this attack in 2014, and since then, have produced a number of tools that address this issue [4], [5], [6], [7], [8], [9], [10], [11], [12]. Unfortunately, these tools are based on the system data that is not accessible in modern stock versions of Android OS [13], [14], [15], [16] or involves severe firmware modifications, which make them unusable on stock Android OS devices. Unlike previous research, we aim at finding an effective AND efficient solution implementable on resource-

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

constrained mobile platforms that would restrict the amount of data collected and analyzed with only major system parameters but would allow achieving a reasonable detection performance in real-time.

In order to achieve this goal, in this paper, we employ recurrent neural network (RNN) and analyze the performance of its various models for the attack detector design and implementation on mobile devices in real-time. The RNN architecture was chosen based on our previous research results, where it outperformed such CI techniques as feed-forward neural networks and decision trees [17], [18]. To validate our results and design, we implement multiple scenarios of a novel "colluded applications" attack and record corresponding system technological signals such as memory consumption and CPU clock speed (frequency) and use them to design CI-based attack detectors. We realize the developed attack detector as an Android application that can be executed on Android-based devices without firmware modification.

The paper is organized in the following way. Section II introduces the "colluded applications" description and a corresponding attack formalized model. We present the data collection process for technological system signals and investigate the effect of the attacks on these signals in section III. Section IV presents RNN-based attack detectors, analyzes their performance, and renders the attack detector implementation on a mobile device. Section V concludes the paper.

II. Colluded applications attack model

"Colluded applications" attack, despite its novelty, has already been reported multiple times. In 2015, a modified version of this attack was discovered [19]. The "Moplus" SDK library that was used in 14,112 Android applications at the time of discovery created numerous backdoors and other breaches. Once the application that contains this library was installed, its exploitation allowed an attacker to: get phone details, download and upload files to/from a device, read/send text messages, get a user's geo-location, etc. In 2016, Intel identified 21 applications that may execute "colluded applications" attacks to escalate privileges, bypass system limitations, and perform malicious activities [20]. Later, in June 2016, McAfee Labs published the threats report [21] that documented 23 colluded applications. In 2017, more than twenty thousand of application pairings that may leak data were identified [22]. More applications in the wild that exploit this vulnerability were discovered in 2018 [23]. In November 2019, the security research team at Checkmarx discovered and implemented a modification of "colluded applications" attack [24] that involved vulnerability of the standard "camera app" and allows to control a smartphone's camera, take photo images, record videos, and send them to a remote server. These cases demonstrate the importance of developing an efficient and effective attack detector.

A. Colluded applications attack formalized description

Let the applications A and B belong to a set of installed applications S. A has the permission set P_A, and B has the permission set P_B. P_A consists of the normal permission subset P_{NA} and the dangerous permission subset P_{DA}. P_B consists of the normal permission subset P_{NB} subset and the dangerous permission subset P_{DB}. P_{DA} and P_{DB} represent subsets of dangerous permissions DP. P_{DA} and P_{DB} are not equal. A generates data object D. To generate D, a permission p_D is required, and to leak D to third parties permission p_L is required. A transmits D to B $(t_A(B, D_{p_D}, background))$ while A and B are in the background. If P_{DA} includes p_D and does not include p_L, P_{DB} does not include p_D but includes p_L, then A and B are colluded applications. Definition of colluded applications can be written with the following statements:

$(A, B \in S) \wedge (P_{DA}, P_{DB} \subset DP) \wedge P_{DA} \neq P_{DB} \wedge (p_D \in P_{DA}) \wedge (p_D \notin P_{DB}) \wedge (p_L \in P_{DB}) \wedge (p_L \notin P_{DA}) \wedge$
$\wedge t_A(B, D_{p_D}, background) \rightarrow A$ and B are colluded.

It is important to notice, that $P_{DA} \neq P_{DB}$. In the case $P_{DA} = P_{DB}$ there is no sense for application collusion. Applying this rule before further analysis may reduce the search space of analyzed applications. In addition, this definition is true for Intra-Library Collusion (ILC) attack [25] as well, since instances of an embedded library are part of host applications.

The attack is an action that results in unauthorized data flow from the device's source to the attacker's destination through colluded applications without a user's permission (see figure 1).

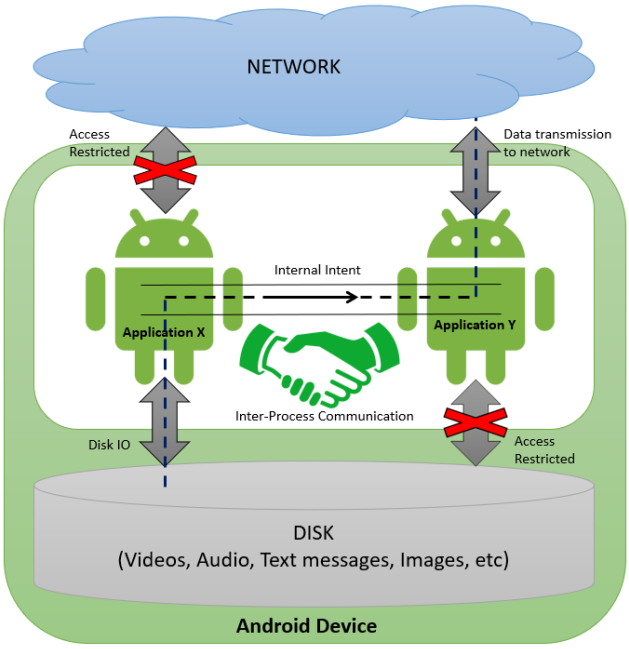

Fig. 1: Attack data flow model

III. Data Collection and Its Pre-processing for Detector Design

A. Data Collection

To employ CI techniques in detector design, we collect data representing various attack/"no attack" scenarios on real Android devices (see Table I). We have collected 14,143,997 entries for over 5,000 attacks. As we rendered in section I, most of the current research on "colluded applications" detection use data that are not accessible on the modern versions of Android OS. For our research, we chose those technological signals that are available through a standard Android API: random access memory (RAM) consumption and CPU cores clock speed (a.k.a. CPU frequency).This data collection is done using an Android application which monitors technological system signals in diverse application collusion scenarios such as:

1) **Pure Attack Scenario**: This scenario represents the attack implementation on a real device. In this scenario, we employ two Android applications (A and B). Both A and B applications have an activity's A1 and A2 and a service's S1 and S2 (the activity provides the user interface (UI), while service is a UI-less app component that runs in the background and performs long-running tasks). S1 is capable of processing incoming intents and then sending the data to the requester. S2 is capable of sending explicit intents to demand data from application A. Application A has access to contacts, SMS, audio, and image data, while application B has access to the internet. Applications B colludes with app A and steals all the data to which application A has access. The data is transmitted between applications in chunks.
2) **No Attack Scenario**: This scenario represents typical device exploitation, where users perform various operations with such types of data as images, video recordings, audio recordings, etc. In this scenario, two legitimate applications may exchange data and transfer it to the internet. For instance, images are shared by Instagram or WhatsApp performs a data backup. Using this scenario data helps us to reduce the false alarm rate.
3) **No Data Transmission Scenario**: In this scenario, no attack is executed, and no data is uploaded, accessed, or transmitted.

Figure 2 visualizes an example of the collected technological signals, with clearly visible correlation patterns between attack and CPU frequency and memory consumption that could be learned with CI techniques.

B. Data Pre-processing

Before using it in the detector design, the collected data is pre-processed and normalized to facilitate an RNN model application. The data pre-processing includes:

1) Noise reduction;
2) Stabilizing the time interval between records;
3) Data normalization;
4) Data labeling;

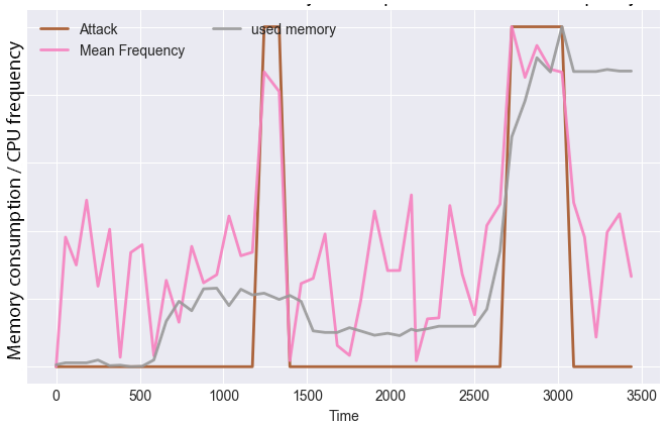

Fig. 2: Samples of the technological signals and their change during the attack and no-attack intervals.

5) Input dimensionality reduction;

Below, we provide further details on these data preprocessing operations.

The first problem that was identified in our preliminary empirical study with various devices [17], [18] is the excessive noise in the recorded CPU frequency signals. To resolve this issue, we used a band-pass filter, which is called to limit the output frequencies within the specified range of allowed frequencies. We embedded this band-pass filter into the aforementioned "signal monitoring" application. This frequency range is determined during the data collection, and it varies for each Android device.

The second problem that we identified is inconsistency in time intervals between sample records because technological system signals are not measured at regular time intervals due to the Android OS properties. We solved this problem by generating missing samples using linear interpolation.

The data was normalized within the range from 0 to 1 for each input (we employed formula 1). Thus, after performing the Exploratory Data Analysis, we made the data suitable for an RNN model.

$$normalized\,value = \frac{col\_val - col_{min}}{col_{max} - col_{min}} \tag{1}$$

The data was labeled with all samples that lay within the attack time bracket identified as "attack" and all others labeled as "no attack".

To improve the performance of an RNN model, we apply a Principle Component Analysis (PCA) to reduce the dimension of our input data. This processed data is then used as an input to RNN models, as discussed in the next section.

IV. RNN Models, their Implementation and Performance

As stated earlier, the data collected is presented as the time-series data, where many inputs or chunks of input sequences are mapped to a single output. This is one of the major

TABLE I: Android phone models and OS versions used for data collection.

Name	OS Version
One Plus 6T	9.0
One Plus 5T	8.0
Moto G6	9.0
Moto G5	8.0
Pixel XL	9.0

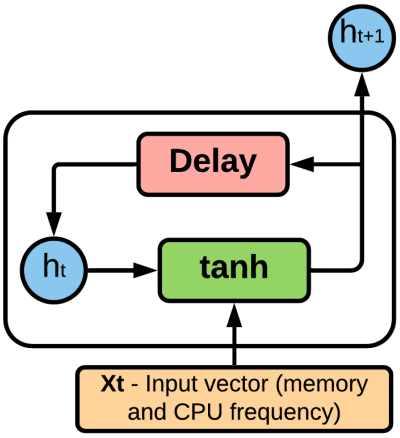

Fig. 3: Basic architecture for a simple RNN model

reasons why we considered using an RNN in our investigation. One of the goals of this research is to investigate which type of an RNN model would perform better at the limited computational cost. We analyzed such types of RNN as a simple RNN, a Long short term memory (LSTM) RNN, and a Gated Recurrent Unit (GRU) RNN.

An RNN is a type of neural network that is generally used when the data has certain temporal structures, and the current state depends not only on the current input but also on the previous state. An RNN model can be considered as a modified Feed-Forward Neural Network (FFNN) with an internal memory component. As seen in figure 3, it processes X_0 input from the data to produce the output h_0. This output, along with the X_1, the next input, produces the output h_1, then used as an input to its next state. This way, it keeps remembering the context while training. The formula for the current state is

$$h_t = f(h_{t-1}, X_t)$$

This network uses the $tanh$ squashing function, as shown in figure 3.

In one of our early classifier designs, we employed an RNN architecture with a single input layer, multiple hidden layers with the feedback loop, and an output layer. The results we got were not impressive, even for smaller sizes of sliding windows, and they deteriorated even further for larger window sizes. The accuracy we achieved was 62%. The primary reason for this low performance was the vanishing gradient problem that RNN faces. We managed to overcome this issue by employing an LSTM-based classifier.

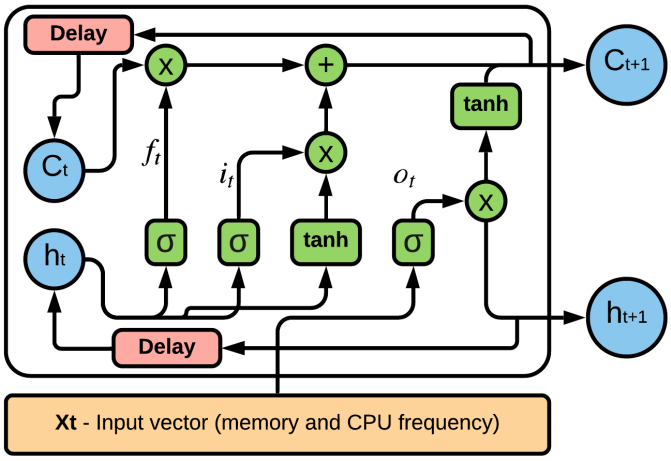

Fig. 4: Basic architecture for LSTM model

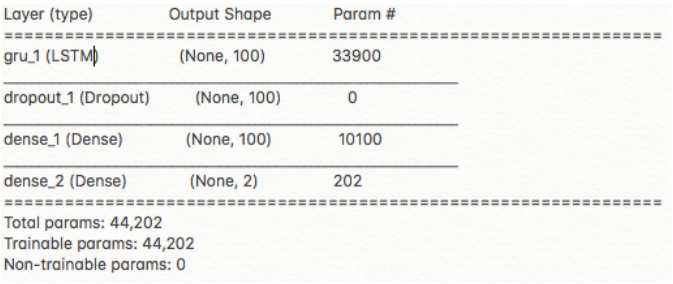

Fig. 5: LSTM model parameters generated by TensorFlow

The LSTM is a modified version of an RNN architecture with three gates that facilitates remembering past data in memory (see figure 4). The LSTM model is trained by using back-propagation. The first gate is the input gate, which decides which input should be allowed to modify the memory and to what extent. The second gate is the "forget gate", which decides what to discard from the memory block, and the third is the output gate [26]. We implemented an LSTM model using the architecture presented in figure 5.

The model we created using the LSTM model achieves a very good result with a detection accuracy of 93.48% for the non-interpolated data set and 96.27% for the interpolated one. However, an LSTM model faces another issue, which is due to the additional gates, the computational resources required to run this model, and the time needed for the training are high if compared with a simple-RNN model. Moreover, the size of the generated model is significantly bigger. Since our ultimate goal is to create a stand-alone Android OS application that is going to be executed on an Android device with limited resources, employing a large and computationally intensive model is not the best solution. These problems can be solved by utilizing a GRU-based model instead of the LSTM-based model.

GRU is another variation of the RNN architecture, which provides good accuracy at a lower computational cost if compared to the LSTM architecture [27]. It consists of one gate called an "update gate", which decides whether to pass the

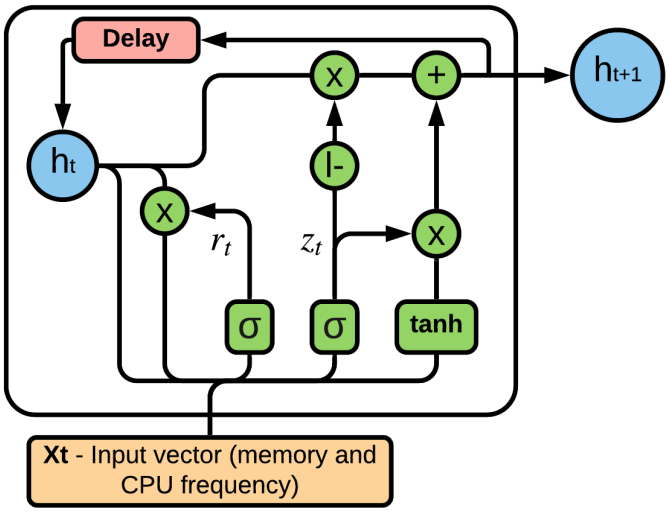

Fig. 6: Basic Architecture for GRU model

Layer (type)	Output Shape	Param #
gru_1 (GRU)	(None, 100)	33900
dropout_1 (Dropout)	(None, 100)	0
dense_1 (Dense)	(None, 100)	10100
dense_2 (Dense)	(None, 2)	202

Total params: 44,202
Trainable params: 44,202
Non-trainable params: 0

Fig. 7: GRU model parameters generated by TensorFlow

output of the previous state to the next state or not (see figure 6). We implemented a GRU-based model using the architecture presented in figure 7. This GRU-based model trains faster (see figures 9, 11), has better accuracy (see figures 8, 10), and the size of the output model is 17% less if compared to an LSTM-based design.

We performed various experiments using these models for finding the right hyper-parameters. We used the interpolated and non-interpolated datasets to compare the results for different window sizes, as seen in figure 12 and figure 13.

A. Implementing an attack detector on a real Android smartphone

Since initial classifier implementations were developed in Keras with TensorFlow backend, we converted the developed GRU-based model to a TensorFlow-Lite model. The conversion was performed by the TensorFlow-Lite converter and allowed us to embed the developed GRU model into

TABLE II: SQLite table: attack_info (dataset example)

Attack Time	Active Apps
03-12-2019 5:39	Mail, App A, App B
03-12-2019 5:40	Photos, App A, App B

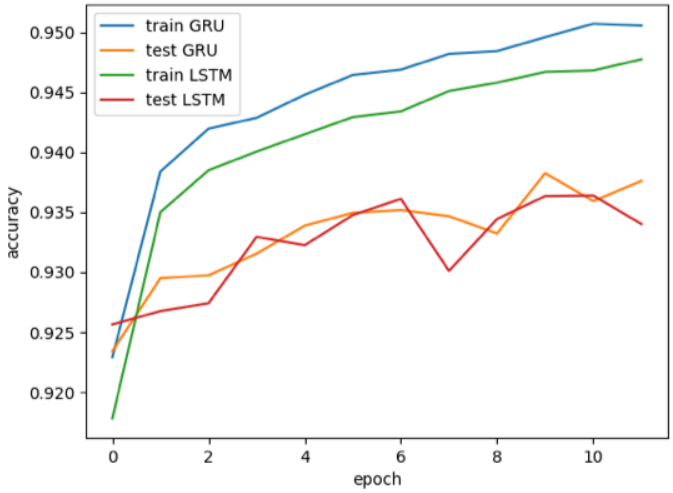

Fig. 8: Detection accuracy of both GRU and LSTM models that use raw dataset.

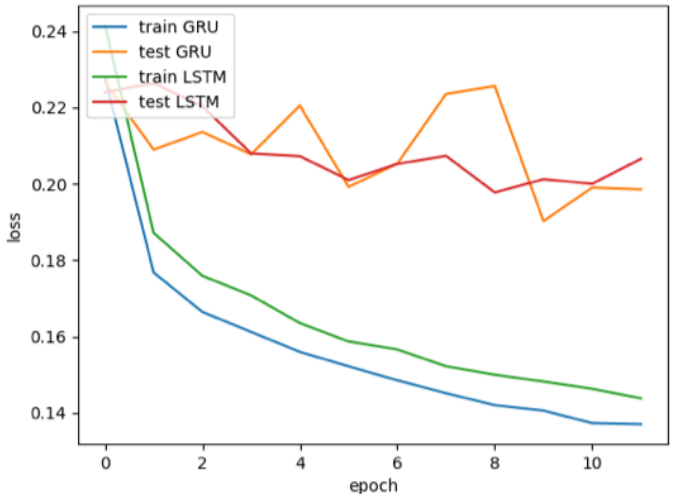

Fig. 9: Loss function plot from GRU vs LSTM using raw dataset.

the Android OS stand-alone application. The final Android application has three parts, a Service component, and two Activity components (see figure 14). The service component runs in the background and continually collects the system data and feeds it to the TensorFlow-lite model. If an attack is detected, we find out all the applications that are active on the host phone at that point of time and save them into the SQLite database table called attack_info (see table II).

We display the attack timings column in the first activity as a RecyclerView, and upon selecting an attack row, the second activity is triggered, which displays all the applications which were active at that moment. The user can then check the permissions granted to this application subset and easily find the "colluded applications".

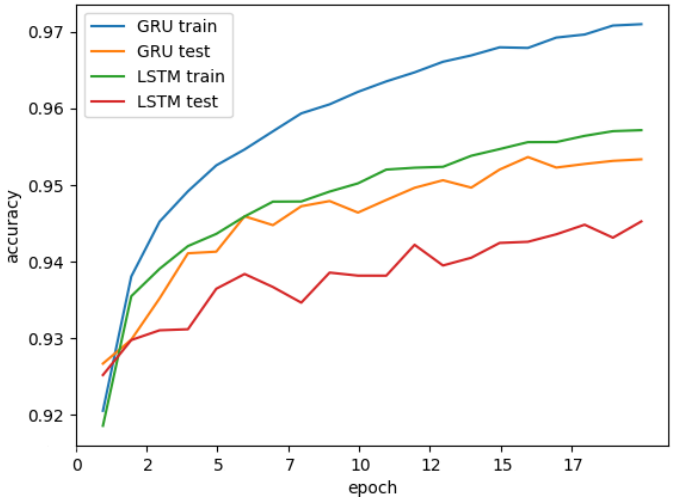

Fig. 10: Detection accuracy of both GRU and LSTM models that use pre-processed dataset.

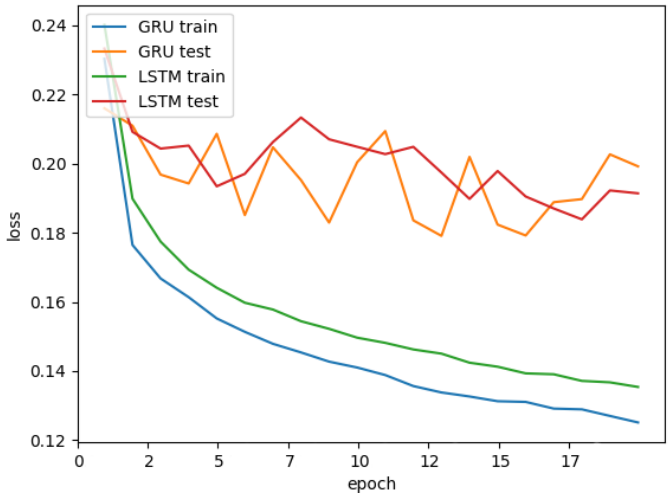

Fig. 11: Loss function plot from GRU vs LSTM using pre-processed dataset.

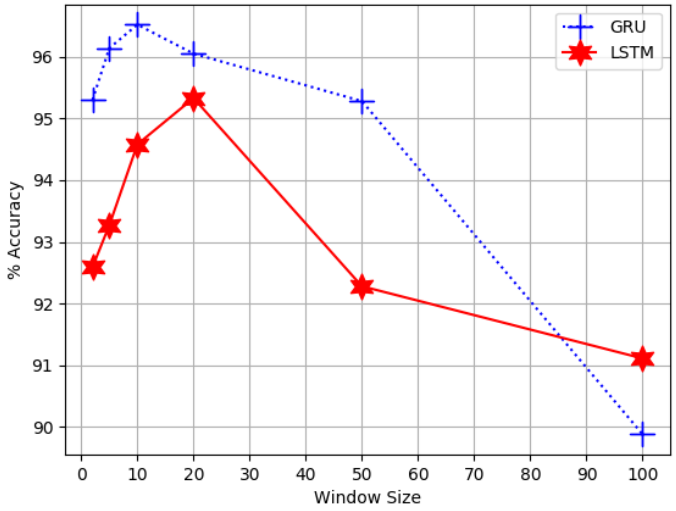

Fig. 12: Detection accuracy vs. window size for the GRU model (20 epochs) for the raw dataset.

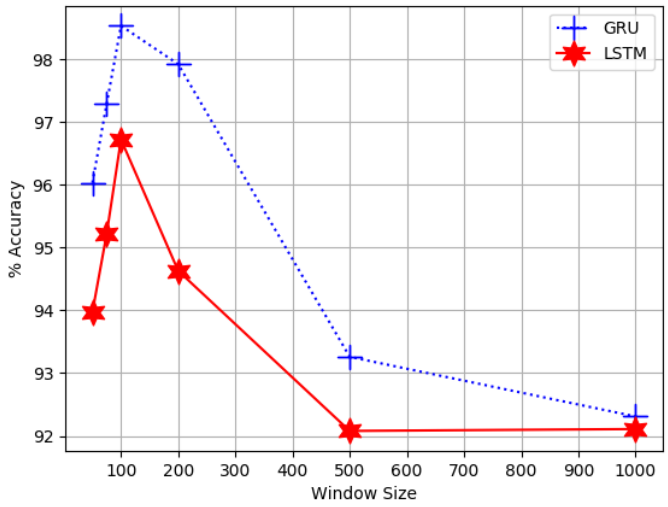

Fig. 13: Detection accuracy vs. window size for the GRU model (20 epochs) for the pre-processed dataset.

V. Conclusion

In this paper, we presented the developed effective and efficient implementation of the "colluded applications" attack detection based on the analysis of the major accessible Android OS system technological signals of mobile devices. The detector was developed by employing RNN and its variations. The attack detector is designed to perform in real-time on a stock Android smartphone with no firmware modification required. "Colluded applications" attack poses a severe threat to the user's data privacy and safety on the ever-growing, most popular mobile platform. Even though this attack has been researched for several years, no tools have been developed that are capable of detection and mitigation of this attack in real-time on a standard device. We analyzed available accessible data, which resulted in choosing to employ overall memory consumption and CPU cores' clock speeds as the inputs of the developed attack detectors.

In order to evaluate the effectiveness and efficiency of our attack detectors, we conducted an empirical study that exploited this novel attack. In this paper, several scenarios that represent restricted data transmission of various types such as contacts, text messages, audio, video, and images were implemented on the real Android smartphones. During the attacks, overall memory consumption and CPU cores clock speed (frequency) were recorded. We compiled a comprehensive dataset and performed exploratory data analysis on it to find the most relevant data points that we can use to train our RNN based model. In addition, in order to use the collected data in RNN models, it was processed by filtering out the excessive noise and restoring the missed data points. We made the dataset available for public use (link: http://bit.ly/2k3M5Ny).

We implemented attack detectors that are based on simple

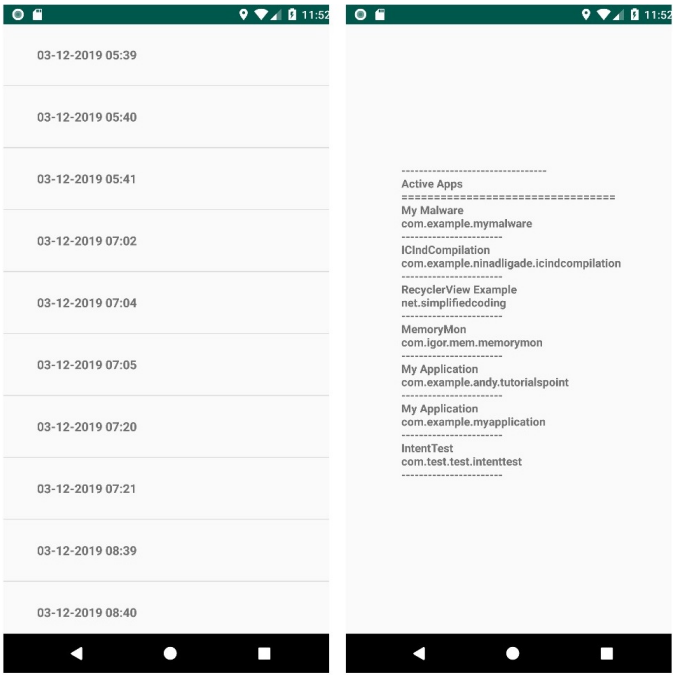

Fig. 14: Android application screenshots

RNN, LSTM, and GRU models. All models were trained and tested on the collected data, and their performance was compared and analyzed. Our analysis has rendered that the most effective and efficient design is based on the GRU model, which achieved more than 95% accuracy in the attack detection task. The developed GRU model was then converted into a stand-alone Android application that detects attacks in real-time and makes the attack logs available to a device owner.

In our future work, we are aiming at collecting more data from a broader set of smartphones and more comprehensive attack and device usage scenarios. In addition to Android OS, we are planning to cover iOS as well, which will allow us to cover an even bigger percentage (close to 100%) of mobile devices.

REFERENCES

[1] (2019, Dec.) Statista. . Accessed: Dec 09, 2019. [Online]. Available: https://www.statista.com/search/?q=Android&qKat=search

[2] (2019) Malware statistics, trends and facts in 2019. . Accessed: January 21, 2019. [Online]. Available: https://www.safetydetective.com/blog/malware-statistics/

[3] D. Goodin. (2019) Google play malware used phones' motion sensors to conceal itself. . Accessed: January 18, 2019. [Online]. Available: https://arstechnica.com/information-technology/2019/01/google-play-malware-used-phones-motion-sensors-to-conceal-itself/

[4] I. M. Asavoae, J. Blasco, T. M. Chen, H. K. Kalutarage, I. Muttik, H. N. Nguyen, M. Roggenbach, and S. A. Shaikh, "Towards automated android app collusion detection," *arXiv preprint arXiv:1603.02308*, 2016.

[5] D. Wu, Y. Cheng, D. Gao, Y. Li, and R. H. Deng, "Sclib: A practical and lightweight defense against component hijacking in android applications," *arXiv preprint arXiv:1801.04372*, 2018.

[6] A. Bosu, F. Liu, D. D. Yao, and G. Wang. (2018, Jun.) Android collusive data leaks with flow-sensitive dialdroid dataset. . Accessed: January 17, 2019. [Online]. Available: http://people.cs.vt.edu/gangwang/DIALDroid-poster.pdf

[7] A. Bosu, F. Liu, D. D. Yao, and G. Wang, "Collusive data leak and more: Large-scale threat analysis of inter-app communications," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ACM, 2017, pp. 71–85.

[8] Z. Zhao and F. C. C. Osono, ""trustdroid^{TM}": Preventing the use of smartphones for information leaking in corporate networks through the used of static analysis taint tracking," in *Malicious and Unwanted Software (MALWARE), 2012 7th International Conference on*. IEEE, 2012, pp. 135–143.

[9] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Transactions on Computer Systems (TOCS)*, vol. 32, no. 2, p. 5, 2014.

[10] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, and A.-R. Sadeghi, "Xmandroid: A new android evolution to mitigate privilege escalation attacks," *Technische Universität Darmstadt, Technical Report TR-2011-04*, 2011.

[11] R. Hay, O. Tripp, and M. Pistoia, "Dynamic detection of inter-application communication vulnerabilities in android," in *Proceedings of the 2015 International Symposium on Software Testing and Analysis*. ACM, 2015, pp. 118–128.

[12] M. Y. Wong and D. Lie, "Intellidroid: A targeted input generator for the dynamic analysis of android malware." in *NDSS*, vol. 16, 2016, pp. 21–24.

[13] I. Khokhlov and L. Reznik, "Data security evaluation for mobile android devices," in *2017 20th Conference of Open Innovations Association (FRUCT)*. IEEE, 2017, pp. 154–160.

[14] I. Khokhlov and L. Reznik, "Colluded applications vulnerabilities in android devices," in *2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*. IEEE, 2017, pp. 462–469.

[15] I. Khokhlov and L. Reznik, "Android system security evaluation," in *2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 2018, pp. 1–2.

[16] I. Khokhlov, Q. Li, and L. Reznik, "D.i.f.e.n.s.e.: Distributed intelligent framework for expendable android security evaluation," in *The 14th Annual Symposium on Information Assurance (ASIA '19)*. University at Albany, SUNY, 2019, pp. 18–27.

[17] I. Khokhlov, M. Perez, and L. Reznik, "Machine learning in anomaly detection: Example of colluded applications attack in android devices," in *18th IEEE International Conference on Machine Learning and Applications - ICMLA 2019*. IEEE, 2019.

[18] I. Khokhlov, M. Perez, and L. Reznik, "System signals monitoring and processing for colluded application attacks detection in android os," in *2019 IEEE Western New York Image and Signal Processing Workshop (WNYISPW)*. IEEE, 2019, pp. 1–5.

[19] C. Cimpanu. (2015, Nov.) 100 million android users may have a backdoor on their device thanks to the baidu sdk. . Accessed: January 17, 2019. [Online]. Available: http://news.softpedia.com/news/100-million-android-users-may-have-a-backdoor-on-their-device-thanks-to-the-baidu-sdk-495673.shtml

[20] C. Cimpanu. (2016, Jun.) 21 android apps spotted using app collusion attacks. . Accessed: January 17, 2019. [Online]. Available: http://news.softpedia.com/news/21-android-apps-spotted-using-app-collusion-attacks-505252.shtml

[21] (2016, Jun.) McAfee labs threats report. . Accessed: January 17, 2020. [Online]. Available: https://www.mcafee.com/us/resources/reports/rp-quarterly-threats-may-2016.pdf

[22] K. Waddell. (2017, Apr.) When apps secretly team up to steal your data. . Accessed: January 17, 2019. [Online]. Available: https://www.theatlantic.com/technology/archive/2017/04/when-apps-collude-to-steal-your-data/522177/

[23] J. Blasco, T. M. Chen, I. Muttik, and M. Roggenbach, "Detection of app collusion potential using logic programming," *Journal of Network and Computer Applications*, vol. 105, pp. 88–104, 2018.

[24] (2019, Nov.) Google confirms android camera security threat: 'hundreds of millions' of users affected. . Accessed: January 06, 2020. [Online]. Available: https://www.forbes.com/sites/daveywinder/2019/11/19/google-confirms-android-camera-security-threat-hundreds-of-millions-of-users-affected/#29116e604f4e

[25] V. F. Taylor, A. R. Beresford, and I. Martinovic, "Intra-library collusion: A potential privacy nightmare on smartphones," *arXiv preprint arXiv:1708.03520*, 2017.

[26] (2019, Dec.) The unreasonable effectiveness of recurrent neural networks. . Accessed: Dec 09, 2019. [Online]. Available: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

[27] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.