BIGMOMAL - Big Data Analytics for Mobile Malware Detection

Sarah Wassermann, Pedro Casas

BIGMOMAL — Big Data Analytics for Mobile Malware Detection

Sarah Wassermann*
Inria Paris
sarah.wassermann@inria.fr

Pedro Casas
AIT Austrian Institute of Technology
pedro.casas@ait.ac.at

ABSTRACT

Mobile malware is on the rise. Indeed, due to their popularity, smartphones represent an attractive target for cybercriminals, especially because of private user data, as these devices incorporate a lot of sensitive information about users, even more than a personal computer. As a matter of fact, besides personal information such as documents, accounts, passwords, and contacts, smartphone sensors centralise other sensitive data including user location and physical activities. In this paper, we study the problem of malware detection in smartphones, relying on supervised-machine-learning models and big-data analytics frameworks. Using the SherLock dataset, a large, publicly available dataset for smartphone-data analysis, we train and benchmark tree-based models to identify running applications and to detect malware activity. We verify their accuracy, and initial results suggest that decision trees are capable of identifying running apps and malware activity with high accuracy.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; • **Security and privacy** → **Network security**;

KEYWORDS

Mobile-Malware Detection; Big-Data Analytics; Machine Learning; High-Dimensional Data.

ACM Reference format:
Sarah Wassermann and Pedro Casas. 2018. BIGMOMAL — Big Data Analytics for Mobile Malware Detection. In *Proceedings of ACM SIGCOMM 2018 Workshop on Traffic Measurements for Cybersecurity , Budapest, Hungary, August 20, 2018 (WTMC'18),* 7 pages.
https://doi.org/10.1145/3229598.3229600

The research leading to these results has been partially funded by the Vienna Science and Technology Fund (WWTF) through project ICT15-129, "BigDAMA".

*This work has been carried out while the author was an MSc. student at the University of Liège.

WTMC'18, August 20, 2018, Budapest, Hungary

ACM ISBN 978-1-4503-5910-8/18/08…$15.00
https://doi.org/10.1145/3229598.3229600

1 INTRODUCTION

Today, more and more tasks are performed on smartphones, and they see an impressive popularity: while 472 million devices were sold in 2011 [1], this number has been more than doubled in 2014 [2]. Moreover, users spend most of their time browsing on their mobile phones, more than on any other device. In the United States and the United Kingdom, for instance, smartphone users spend 2–3x more hours (87 hours per month in the US and 66 hours in the UK) on their mobile device than on desktop machines (34 hours and 29 hours, respectively) [3]. The impressive growth of mobile devices and usage has led to unprecedented increase in cellular traffic. Unfortunately, these devices increasingly become the target of malicious attacks that aim at stealing the users' private information and using it for bad-natured activities. Smart devices are an appealing target not only because of their popularity, but also because they incorporate a very large amount of sensitive information about the users, even more than a personal computer [4]. For instance, the included sensors can track the users' current location and physical activities, and the embedded cameras can take pictures and record videos, even without the users' awareness.

The first piece of malware was detected before 2004 [4] and the number of malware detections and their diversity grew with the expansion of the smart-device market. In particular, popular mobile operating systems such as Symbian, iOS, and Android accelerated the development of mobile malware: while only about 400 malware infections were reported between 2004 and 2007 [4], this number increased by 3,325% in the last seven months of 2011 for Android alone [5]. In 2016, Kaspersky Lab found approximately 8,500,000 malicious installation packages, of which 130,000 mobile banking trojans and 260,000 mobile ransomware trojans [6], and noted that a large part of malicious software is distributed via the official app stores. The year 2016 also underlined that developers of malicious apps make use of the current trends: an app called *Guide for Pokémon Go* including a trojan – Ztorg – made its appearance on Google Play and was installed more than 500,000 times [7]. Ztorg belongs to the group of rooting malware, which gains super-user rights and is thus able to do almost everything on the infected device [8]. In addition to that, the user cannot remove this kind of malware by simply performing a factory reset of her smartphone, which makes Ztorg an even more frightening threat. Even though the number of rooting-malware infections decreased in 2017, these attacks are still very popular. According to the same report, 2017 was the year of the rise of WAP and mobile banking trojans. WAP trojans steal money from the user by visiting (often without her knowing) websites with WAP subscriptions. Additionally, mobile banking trojans are now going beyond traditional financial apps and target applications designed for booking hotels, taxis, etc.

Given the relevance and sensitivity of this topic, we study the problem of malware and running-application detection in Android

smartphones relying on supervised-machine-learning models. Using a large, publicly available dataset for smartphone data analysis (the SherLock data collection[1]), we train and benchmark different supervised-machine-learning models to detect malware activity. We evaluate three different concepts, including (*i*) overall model performance, (*ii*) generalisation of the learned models across different users, and (*iii*) detection accuracy drift along time. Initial results suggest that tree-based models are capable of identifying malware activity with high accuracy and extremely low computational time.

The remainder of this paper is organised as follows: Section 2 discusses related work. In Section 3, we present and describe the dataset used in our study. In Section 4, we take a first approach to tackle the problem, by investigating the possibility of automatically recognising the apps running on a given device from low-level footprints, and apply the learnings and approaches later in Section 5 to automatically detect malware. Finally, Section 6 concludes this work.

2 RELATED WORK

Detecting mobile malware is a very challenging task and subject to significant research efforts. Zhou et al. found that the best antivirus software merely detects 79.6% of malware [9], which is a rather disappointing result. It is worth noting that the wide range of touch commands (such as swiping and tapping on the display) adds another layer of complexity to the process of malware detection, as this makes the exploration of all execution paths almost impossible [10]. Nevertheless, the authors of Malton [11] developed a system which promises to be a non-invasive Android-malware-detection tool outperforming multiple other proposals.

According to [4], Android OS relies more on *platform protection* than on *market protection*. In other words, Android users can download applications even from unofficial markets, but the operating system attempts to protect the users from malicious behaviour by, for instance, applying a permission-based system (i.e., an application can only perform the actions for which the permission has been granted) or letting applications run in a sandbox (i.e., an environment isolating the app). Non-negligible efforts have been spent to detect the overprivilege problem, i.e., an application requests more permissions than it needs to work properly. Projects such as Stowaway [12], its successor PScout [13], and DroidRisk [14] try to identify applications incorporating this kind of security threat. The work presented in [15] uses permission-based heuristics to determine whether an application is malicious or not.

In [10, 16], Arshad et al. and Tam et al. provide an extensive survey about malware detection and protection on Android. They present two approaches for detecting malware on Android phones: static and dynamic ones. While the static techniques are mainly based on source code analysis, dynamic methods analyse applications during their execution. For example, static techniques involve signature and permission analyses, whereas dynamic ones use anomaly detection approaches and emulation. The aforementioned works [12–15] fall into the category of static-analysis systems. TaintDroid [17] and Maline [18] are two examples of systems making use of dynamic techniques. TaintDroid relies on captured network data to analyse Android applications, while Maline is a malware-detection tool which is based on machine-learning techniques to classify system calls. Another tool relying on a dynamic approach has been proposed by Afonso et al. [19]: it investigates the system calls and API uses of an application to feed extracted features to a machine-learning algorithm.

To help the cybersecurity research community, several researchers make their mobile malware datasets publicly available. A good example is the SherLock dataset [20], a massive time-series dataset containing more than 600 billion samples. More precisely, they analysed the activities of Android applications in a very fine-grained fashion and simulated malicious behaviours on the monitored smartphones. We use this dataset in our work and describe it in more details in Section 3. Other examples are the Device Analyzer dataset [21], a very large dataset including data instances collected from nearly 900 different devices showing the behavioural diversity among users, the LiveLab Project [22], a dataset investigating the usage of iPhone 3GS smartphones, and the Android Malware Genome Project [9], a dataset encompassing more than 1,200 different Android malware samples from nearly 50 distinct malware families.

3 THE SHERLOCK DATASET

In the context of this work, we rely on the SherLock dataset, published by the BGU Cyber Security Research Center [20]. The SherLock dataset has been collected from January 2015 until December 2017 and includes very valuable information about the usage of smartphones. The collection was done during a long-term field trial on 50 smartphones used as primary device for different participants. More precisely, Mirsky et al. handed a Samsung Galaxy S5 to 50 clients. On these smartphones, the authors installed two applications, namely SherLock and Moriarty: while SherLock is responsible for collecting data about the smartphone (SMS, call logs, etc.), Moriarty periodically performs malicious activities, simulating the behaviour of malware found in the wild. SherLock monitors two kinds of sensors: PULL sensors which are analysed regularly (to collect, for instance, information about the installed and currently running applications, and the values of accelerometers) and PUSH sensors which are triggered when an event occurs (for example, receiving / issuing a call or a text message). The malicious actions of Moriarty are diverse and encompass activities such as contact, photo and SMS theft, phishing, and ransomware: its payload is updated every few weeks. Data collection is done at very high sample rate (as low as 5 seconds). The complete labelled dataset contains more than 600 billion data records, with a total of more than 4 TB of data. It is worth noting that the SherLock team cares a lot about the users' privacy: network identifiers such as cell tower IDs, SSIDs, and MAC addresses have been hashed; geographical locations have been anonymised through clustering techniques.

In this paper, our evaluation focuses on the data collected during the second quarter of 2016. From the complete dataset, we keep two specific feature categories: all those features related to the network traffic generated by the apps, and all those features corresponding to the footprint of the app on the CPU and internal running processes (e.g., statistics on CPUs, memory usage, Linux-level process information). The rationale is that some malware activity would be more visible at the network-traffic level, whereas some others would be better identified at the local-process level.

[1] http://bigdata.ise.bgu.ac.il/sherlock/

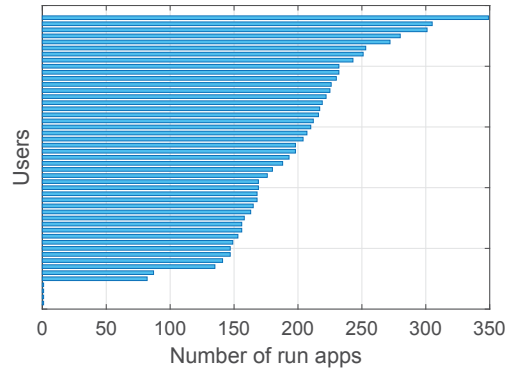


Figure 1: Number of different applications launched during Q2 2016 by each user.

Given the size of the data we study, we rely on big-data platforms to conduct the study. In particular, we use the Big-DAMA framework [23] to process the dataset, build machine-learning models, and evaluate their performance. In a nutshell, Big-DAMA uses Apache Spark Streaming for stream-based analysis, Spark for batch analysis, and Apache Cassandra for query and storage. Within the Big-DAMA platform, we have conceived different algorithms for network security and anomaly detection using supervised- and unsupervised-machine-learning models.

We start by doing a brief characterisation of the considered dataset snippet. During these three months, we found more than 600,000,000 records from 48 different users. Figure 1 depicts the number of different Android applications launched by each user. We can easily see that not all the users use their smartphone in the same way: while nearly 50% launched more than 200 apps, there are also clients who used less than 10 different applications. For the latter, we suppose that they did not use the Samsung smartphone on a daily basis, but only periodically for the SherLock campaign. Next, we check which applications were the most popular among our users. We consider an app as popular if the dataset contains a large number of data instances for it. Table 1 shows the ten most popular apps for Q2 2016. We note that SherLock was the app yielding by far the most samples, which comes as no surprise because it is the application responsible for the information gathering. Through this table, we can also point out that a non-negligible part of the popular applications are Android services which are automatically launched in the background by the Android OS and not by the users. For instance, SmartcardService ensures that applications can communicate with the SIM card and Google Play services allows the installed applications to access the newest features published by Google. In Table 2, we summarise the ten most popular apps excluding internal Android services and the SherLock application. According to our findings, WhatsApp and Hangouts seem to have been the favourite social applications of our users from April to June 2016.

To better understand the behaviour of Moriarty as compared to benign applications, we compare WhatsApp – the most popular social app in the dataset – against Moriarty, the malware-emulation app. In Figure 2, we plot several characteristics of both of them. In

Rank	Application name
#1	SherLock
#2	Google Play Services
#3	Chrome
#4	SmartcardService
#5	Unified Deamon
#6	SecurityManagerService
#7	WhatsApp
#8	S Finder
#9	Samsung Push Service
#10	Context Service

Table 1: Top 10 popular apps in Q2 2016.

Rank	Application name
#1	Chrome
#2	Google App
#3	WhatsApp
#4	S Finder
#5	S Health
#6	Hangouts
#7	Samsung text-to-speech engine
#8	Geo News
#9	Peel Smart Remote
#10	Contacts

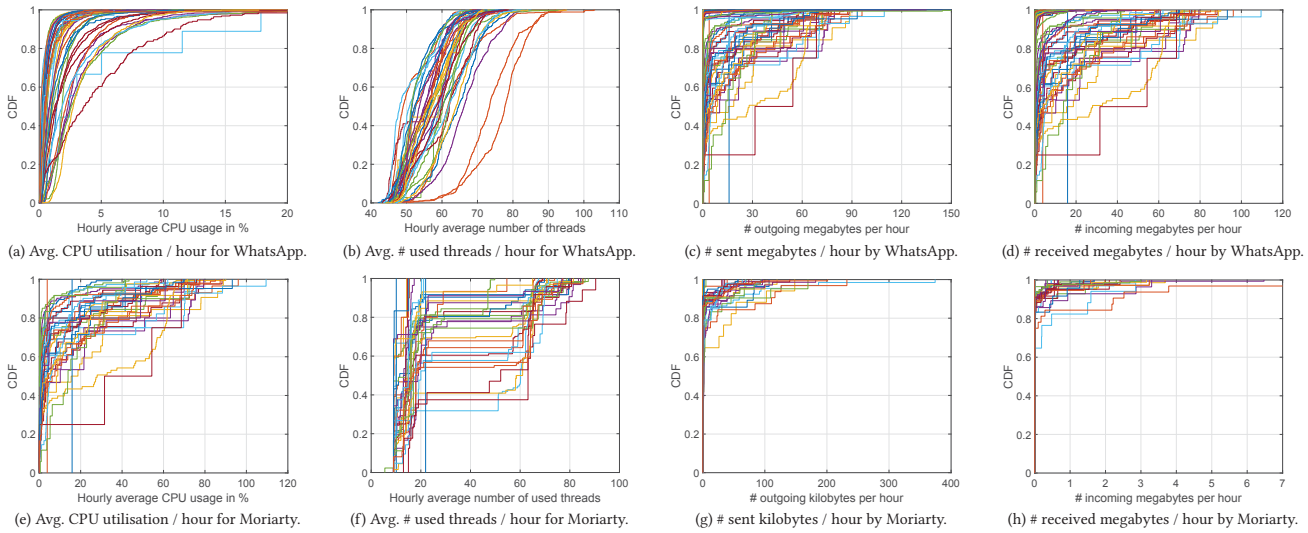
Table 2: Top 10 popular apps in Q2 2016 (Android services and SherLock excluded).

particular, we focus on the average CPU utilisation per hour, the average number of used threads per hour, and the sent / received traffic on a hourly basis. We observe that Moriarty uses much more CPU resources than WhatsApp: while WhatsApp used most of the time only 5% of the CPU on the monitored smartphones, Moriarty mostly needed at least 20%, which is a significant difference. However, WhatsApp seems to need more threads to run properly compared to Moriarty. When it comes to network traffic, Moriarty sent only a couple of kilobytes per hour, whereas WhatsApp sent multiple megabytes. We have a similar behaviour for the received traffic. While this is expected, as the users of WhatsApp constantly exchange messages and multimedia files, this simple comparison makes the point in the statistical differences in some of the aforementioned characteristics, which shall form the basis for the input features to the machine-learning models detecting malware activity.

4 IDENTIFYING RUNNING APPLICATIONS

Before setting the focus on the malware detection, we investigate the possibility of automatically identifying the different running Android applications from the set of input features available in the dataset. The goal is to blindly differentiate among different running apps (i.e., without gathering the process name), so as to better understand if it would be possible to later on identify malware activity using the same approach.

We consider as input a set of 45 SherLock-monitored smartphone features, reflecting the behaviour of the different apps, and build



(a) Avg. CPU utilisation / hour for WhatsApp.
(b) Avg. # used threads / hour for WhatsApp.
(c) # sent megabytes / hour by WhatsApp.
(d) # received megabytes / hour by WhatsApp.
(e) Avg. CPU utilisation / hour for Moriarty.
(f) Avg. # used threads / hour for Moriarty.
(g) # sent kilobytes / hour by Moriarty.
(h) # received megabytes / hour by Moriarty.

Figure 2: Characterisation of the WhatsApp and Moriarty Android applications. Each curve on the CDFs corresponds to one user and each point of the curve to one hour.

supervised-machine-learning models to automatically classify them. We particularly focus on the usage of decision trees, based on their a-priory excellent performance in similar network-measurement analysis problems [24]. We proceed as follows: for each user, we randomly choose 20% of the application samples as training set and the remaining 80% for testing. We obtain almost perfect results for the entire set of Q2 users: for all of them, our model correctly identified the running applications in about 99% of the cases.

4.1 Feature Selection

Next, we investigate which features are the most discriminative ones when it comes to the identification of applications. We rely on an embedded method, based on decision trees and the notion of variable importance [25]. With this kind of method, the most relevant features are determined while building the prediction model. The ten most important features are reported in Table 3.

We note that the most discriminative features are related to the CPU, threads, and memory. Moreover, the information related to the importance of the Android application – i.e., whether it is running as a service, in the background, or in the foreground – helps for the identification. Overall, our results show that low-level information is very useful for this kind of prediction task. As an additional analysis, we evaluate the application-identification power of a decision tree for one user when relying only on the five and ten most important features. For this, we split the dataset as previously, i.e. 20% of the data is considered as the training set and the remaining 80% as testing set. Our results reveal that the decision-tree model still works extremely well even when using only 5 or 10 features instead of 45: in both cases, the identification was correct for about 99% of the testing records. It thus seems that only a small amount of data is required to obtain highly accurate prediction performance, which has a paramount advantage in case the app identification needs to be done in an online context. Indeed, when working with a restricted set of features, classifiers can work more efficiently and provide an answer quickly. Furthermore, gathering only little data implies that we do not have to overload the smartphone for data collection – only a few system calls are necessary.

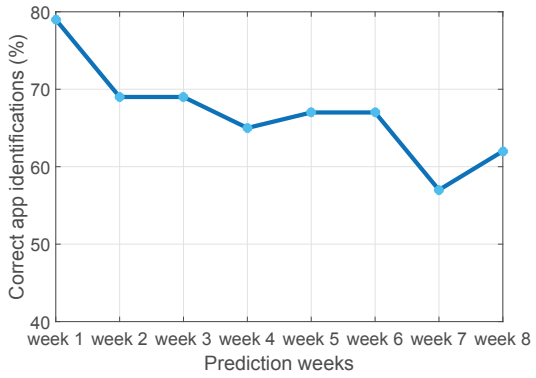


Figure 3: App identification accuracy when training and testing on a weekly basis.

4.2 Temporal App Identification

Finally, we are interested in the quality of the application identification on a temporal basis. More precisely, we are curious about the accuracy of the prediction model in case the training phase takes place in a different time span. In particular, for each user, we train a decision tree on data collected during the first week and try to identify the applications run during the following weeks. Figure 3, depicting the results for one user, reveals that the overall identification accuracy decreases with respect to our previous evaluation, suggesting that restricting the training set to the samples collected

Top 10 features	Application identification	Malware detection
#1 feature	virtual memory size	shared dirty pages used by Dalvik heap
#2 feature	number of threads	number of minor page faults
#3 feature	CPU utilisation	time process scheduled in user mode
#4 feature	process priority level (foreground, background, service, sleeping, etc.)	proportional set size for Dalvik heap
#5 feature	ordering within a particular priority category	time process-children scheduled in user mode
#6 feature	process life time	virtual memory size
#7 feature	time process scheduled in user mode	time process scheduled in kernel mode
#8 feature	time process-children scheduled in user mode	process ID
#9 feature	number of minor page faults	process life time
#10 feature	number of private dirty pages used by everything else than the native heap	number of private dirty pages used by the native heap

Table 3: Most relevant features for identifying running Android applications and detecting malware.

during the first week does not capture as well the users' smartphone habits as when the training set is randomly sampled. Furthermore, the plot shows that the precision of our model decreases over time: while almost 80% of the applications were correctly identified in the first prediction week, this was the case for only about 62% in the last one. The outcome of this analysis for the other users is very similar.

5 MALWARE DETECTION

Our next goal is to detect whether an application is malicious or not, still working with the data collected during the second quarter of 2016. As presented in Section 3, Moriarty simulates bad-natured behaviour and is thus the target of the detection model. We are now trying to answer the following question: *"Is the running application Moriarty or not?"*, which boils down to a binary classification problem. We use the same feature set and prediction model as for the application-identification task. However, as the execution of the Moriarty application can be considered as a rare event (the dataset contains millions of non-Moriarty samples versus only a few thousands of Moriarty records), we randomly choose 40% of samples for the testing set compared to 20% in the context of the application identification. Figure 4 depicts the obtained recall for the Moriarty detection for each user. The recall is defined as $\frac{\#true\ positives}{\#true\ positives + \#false\ negatives}$, and a high value thus reveals that the number of missed malware instances (the false negatives) is low. This is a very important property in our context, as false negatives (i.e., considering an application as benign although it is malicious) are a considerable security threat for the user. Moreover, we achieve an extremely low false positive rate (below 1% for each user), indicating that we almost never classify a benign application as malicious. For the sake of illustration, Table 4 shows the confusion matrix obtained for one of our users.

5.1 Feature selection

Similar to the identification of applications, we are interested in the most important features for detecting malicious apps. The ten most relevant features (determined using the same technique as in Section 4) are summarised in Table 3. We note that, compared to the application-identification task, a lot of these features are related to the memory (five out of ten). Once again, low-level metrics are very useful indicators. As a final step, and similarly to Section 4, we evaluate our decision-tree model for this detection problem with the top 5 and top 10 features. The results are excellent even with these reduced feature sets. Indeed, in both cases, the recall is over 99%.

5.2 Temporal Malware Detection

Next, we carry out the temporal analysis for the Moriarty detection. We proceed in the same way as for our previous task, i.e., we train for each user the detection model on data gathered during the first week of the second quarter and test it in the remaining weeks. Given the major imbalance in classes between benign and Moriarty instances, we resort to standard bootstrapping to rebalance classes for learning purposes. In particular, we randomly oversample the Moriarty class.

Figure 5 depicts the recall obtained per week for one user. As we can easily observe, the results are very poor and detection performance rapidly degrades after the 3rd week. Two issues could be the cause of this performance: firstly, the extremely low number of gathered Moriarty samples in the training week, and secondly, a concept drift; as stated in Section 3, the Moriarty application periodically changes its behaviour, which makes its detection very challenging when the training set does not include samples of all the patterns. Indeed, during the second quarter of 2016, Moriarty changed twice its malicious behaviour: it started with SMS / bank-theft malware activity, moved on to phishing, and finished with simulating adware [20]. This prevents the decision tree from accurately grasping the behaviour of the malicious Android application. Again, the results for the other users are highly similar. We tested several other models for this specific task, without any notable detection improvements.

5.3 Malware Detection across Multiple Users

Finally, we evaluate our detection model on data collected from multiple users. More precisely, we train our decision tree on data gathered from three Q2 users and test it on data belonging to three other ones. The resulting confusion matrix is shown in Table 5.

	Predicted positive	Predicted negative
True positive	14,275,309	155
True negative	117	35,076

Table 4: Confusion matrix obtained while detecting Moriarty for one Q2 user.

	Predicted positive	Predicted negative
True positive	79,775,923	16,784
True negative	35,401	3,472

Table 5: Confusion matrix obtained while detecting Moriarty for a couple of Q2 users when training on data instances from other clients.

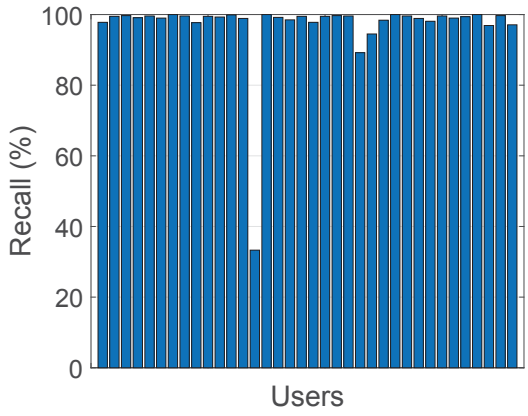


Figure 4: Recall obtained while detecting Moriarty for each Q2 user.

Results are again quite disappointing, suggesting that the particularities of the ways users use their smartphone play a more important role than the set of features used as input by the model. Indeed, each and every user uses her smartphone differently and it is thus very challenging to build a model from multiple users that can be accurately applied to other ones. In other words, malware detection based on behavioural features from a community, i.e., features describing how the smartphone is used, is a very difficult task.

6 CONCLUDING REMARKS AND PERSPECTIVES

In this paper, we tackled two prediction tasks in the domain of smartphone security by relying on the SherLock dataset: the identification of running Android-applications and the detection of malware. For both of these classification targets, we obtained highly promising results in case the training sets are generated randomly. Nevertheless, the Android application identification and malware detection on a temporal basis as well as when relying on data instances from multiple users remains a challenging task. There are two main takeaways from this study: firstly, our results suggest

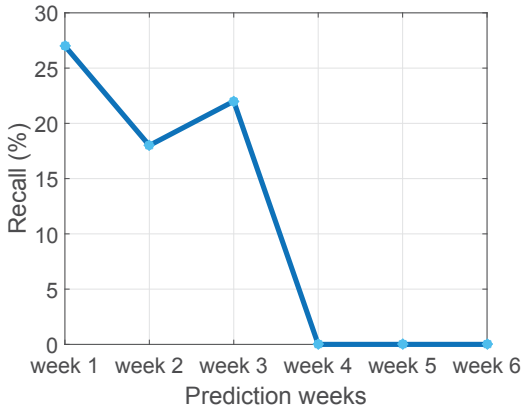


Figure 5: Recall obtained while detecting Moriarty on a weekly basis for one Q2 user.

that malware-detection models using the set of features provided by the SherLock dataset can operate with very high performance only in those cases where models are tailored on a per-user basis, taking into account the statistical behaviour of each user. Secondly, it seems hard to build a model that generalises well and can cover multiple users and longer time spans, additionally due to concept drift occurrences. We are therefore currently investigating stream-based machine-learning approaches for the malware-detection task, in which we periodically retrain the underlying model when a concept drift and / or a detection performance degradation is detected.

REFERENCES

[1] L. Goasduff and C. Pettey, "Gartner says worldwide smartphone sales soared in fourth quarter of 2011 with 47 percent growth," 2012. [Online]. Available: http://www.gartner.com/it/page.jsp?id=1924314
[2] L. Goasduff, "Gartner says smartphone sales surpassed one billion units in 2014," 2015. [Online]. Available: http://www.gartner.com/newsroom/id/2996817
[3] Ofcom, "The Communications Market Report," December 2016. [Online]. Available: https://www.ofcom.org.uk/__data/assets/pdf_file/0026/95642/ICMR-Full.pdf
[4] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, and A. Ribagorda, "Evolution, Detection and Analysis of Malware for Smart Devices," *IEEE Communications Surveys Tutorials*, vol. 16, no. 2, pp. 961–987, February 2014.
[5] Juniper, "2011 mobile threats report," 2011. [Online]. Available: https://www.juniper.net/us/en/local/pdf/additional-resources/jnpr-2011-mobile-threats-report.pdf
[6] Kaspersky Lab, "Mobile Malware Evolution 2016," 2016. [Online]. Available: https://securelist.com/mobile-malware-evolution-2016/77681/
[7] R. Unuchek, "Rooting Pokémons in Google Play Store," Septembre 2016. [Online]. Available: https://securelist.com/rooting-pokemons-in-google-play-store/76081/
[8] ——, "Mobile Malware Evolution 2017," March 2018. [Online]. Available: https://securelist.com/mobile-malware-review-2017/84139/
[9] Y. Zhou and X. Jiang, "Dissecting Android Malware: Characterization and Evolution," in *2012 IEEE Symposium on Security and Privacy*, May 2012.
[10] K. Tam, A. Feizollah, N. B. Anuar, R. Salleh, and L. Cavallaro, "The Evolution of Android Malware and Android Analysis Techniques," *ACM Computing Surveys*, vol. 49, no. 4, pp. 76:1–76:41, January 2017.
[11] L. Xue, Y. Zhou, T. Chen, X. Luo, and G. Gu, "Malton: Towards on-device non-invasive mobile malware analysis for ART," in *26th USENIX Security Symposium (USENIX Security 17)*, August 2017.
[12] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android Permissions Demystified," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, October 2011.
[13] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, "PScout: Analyzing the Android Permission Specification," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, October 2012.
[14] Y. Wang, J. Zheng, C. Sun, and S. Mukkamala, "Quantitative Security Risk Assessment of Android Permissions and Applications," in *Proceedings of the 27th Data*

and Applications Security and Privacy (DBSec), July 2013.

[15] Y. Zhou, Z. Wang, W. Zhou, and J. Xuxian, "Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets," in *Proceedings of the 19th Network and Distributed System Security Symposium*, February 2012.

[16] S. Arshad, M. A. Shah, A. Khan, and M. Ahmed, "Android Malware Detection & Protection: A Survey," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 2, February 2016.

[17] W. Enck, P. Gilbert, B. Chun, L. Cox, J. Jung, P. McDaniel, and A. Sheth, "TaintDroid: An Information-flow Tracking System for Realtime Privacy Monitoring on Smartphones," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, October 2010.

[18] M. Dimjašević, S. Atzeni, I. Ugrina, and Z. Rakamaric, "Evaluation of Android Malware Detection Based on System Calls," in *Proceedings of the 2016 ACM on International Workshop on Security And Privacy Analytics*, March 2016.

[19] V. Afonso, M. F., A. R., G. B., and P. L., "Identifying Android malware using dynamically obtained features," *Journal of Computer Virology and Hacking Techniques*, vol. 11, no. 1, pp. 9–17, February 2015.

[20] Y. Mirsky, A. Shabtai, L. Rokach, B. Shapira, and Y. Elovici, "Sherlock vs Moriarty: A Smartphone Dataset for Cybersecurity Research," in *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security*, Octobre 2016.

[21] D. T. Wagner, A. Rice, and A. R. Beresford, "Device analyzer: Large-scale mobile data collection," *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, no. 4, pp. 53–56, April 2014.

[22] C. Shepard, A. Rahmati, C. Tossell, L. Zhong, and P. Kortum, "Livelab: Measuring wireless networks and smartphone users in the field," *ACM SIGMETRICS Performance Evaluation Review*, vol. 38, no. 3, pp. 15–20, January 2011.

[23] P. Casas, F. Soro, J. Vanerio, G. Settanni, and A. D'Alconzo, "Network security and anomaly detection with Big-DAMA, a big data analytics framework," in *2017 IEEE 6th International Conference on Cloud Networking (CloudNet)*, Sept 2017, pp. 1–7.

[24] S. Wassermann, P. Casas, T. Cuvelier, and B. Donnet, "NETPerfTrace: Predicting Internet Path Dynamics and Performance with Machine Learning," in *Proceedings of the Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*, August 2017.

[25] P. Geurts, A. Irrthum, and L. Wehenkel, "Supervised learning with decision tree-based methods in computational and systems biology," *Molecular BioSystems*, vol. 5, no. 12, pp. 1593–1605, December 2009.