A Heuristics Approach to Mine Behavioral Data Logs in Mobile Malware Detection System

Giang Nguyen[a,*], Binh Minh Nguyen[b], Dang Tran[b], Ladislav Hluchy[a]

[a] *Institute of Informatics, Slovak Academy of Sciences, Dubravska cesta 9, 845 07 Bratislava, Slovakia*
[b] *School of Information and Communication Technology, Hanoi University of Science and Technology, Vietnam*

Abstract

Nowadays, in the era of every "things" connect together via the Internet, mobile device number has increased exponentially up to tens billions around the world. In line with this equipment increase, generated data amount is enormous and have attracted malefactors to spoil. For hackers, one of the popular ways to threaten mobile devices is to spread malware, which is very difficult to prevent because the application installation and configuration rights are set by owners, who usually have very low knowledge or do not care about security. In this study, we aim at improving security in environment of mobile devices by proposing a novel system to detect automatically malware intrusions. Our solution thus is built based on modeling user behaviors and using heuristic analysis approach to mobile logs generated during the device operation process. Although behaviors of individual user have a significant impact upon social cyber-security but achievement of user awareness still remains one of the major challenges today. For this task, a light-weight semantic formalization in form of physical and logical taxonomy is proposed for classifying the collected raw log data. Then we use a set of techniques like sliding windows, lemmatization, feature selection, term weighting and so on to process data. Meanwhile, malware detection tasks are performed based on incremental machine learning mechanisms because tasks' complex degree. Our solution is developed in the manner of allowing scalability in which the system is composed by several component blocks that cover preprocessing raw collected logs from mobile devices, automatically creating datasets for machine learning methods, using the best selected model for detection suspicious activity surrounding malware intrusions and supporting decision making using predictive risk. The proposals are experimented cautiously and gained test results show the effectiveness and feasibility of our malware detection system in applying to large-scale mobile environment.

Keywords: mobile security, situational awareness, anomaly detection, incremental machine learning, natural language processing, scalable solution design

1. Introduction

Today, the popularity of low cost technologies as well as of mobile devices coupled with the advent of social media, social networks, and cloud computing comes into being multiple connectivity possibilities that can cause unattended, mis-configured devices and even potentially vulnerable for others. Here is a high increase of new zero-day vulnerabilities, personal records stolen or lost, phishing campaigns targeting end-users, malware attacks, fake technical support scams like url-based spreading threats and so forth Parsons et al. (2010), ThreadTrackSecurity (2015), ENISA (2017). Several new effective functions for such threats could be listed including anonymization strategies, strong encryption including HTTPS, flexible

*This is the preprint of the paper: Nguyen, G., Nguyen, B.M., Tran, D. and Hluchy, L., 2018. A heuristics approach to mine behavioural data logs in mobile malware detection system. Data & Knowledge Engineering, 115, pp.129-151, DOI 10.1016/j.datak.2018.03.002

Email addresses: giang.ui@savba.sk (Giang Nguyen), minhnb@soict.hust.edu.vn (Binh Minh Nguyen), dangtv18@gmail.com (Dang Tran), ladislav.hluchy@savba.sk (Ladislav Hluchy)

key management schemes, and obfuscation methods for payload detection. However, there is a truth that while cyber-criminals are getting better, users still are sharing more information electronically and current anti-phishing solution threat intelligence may not be sufficient.

Also at this time, the number of malware is numerous, but they could be categorized into three main groups as follows: back-door (1), security hole (2), and distributed denial of service (DDOS) (3). While malware of (2) does not generate abnormal logs, and malware of (3) often can be prevented effectively using defined rules (i.e. static analyses), malware of (1) creates many challenges for computer scientists because it lurks inside victim's devices a long time, and automatically opens communications with outside without user authorizations. In this study, we concentrate on settling security issue related to back-door malware based on mobile operation logs. The critical desired outcomes of this work is to detect precisely as much as possibly the presence of malware belonging to (1) group inside mobile devices.

Our starting point is the realization: with optimal conditions, activity posture of all mobiles operated in a certain network needs to be monitored securely in real-time. Besides, common and normal situational awareness should be provided to minimize security risks. To do this, it can be seen that modeling behaviors of individual users is one of the important factors for security issue, nonetheless this task still remains one of the major challenges today related to degree of accuracy.

In this direction of using technology to support every people life aspects, a way to improve significantly effectiveness for the security issue mentioned above is to incorporate innovative models and new technologies to better understand human behaviors on mobile devices. As a result, the consequent trigger action from detection system, at least, should be create more precisely recommendations or warnings Ricci et al. (2010) and Tam et al. (2017) to users or automatically isolate sooner infected devices from network in critical situations.

The main goal of our work is to improving cyber-resilience focused on the user behaviors on mobiles using heuristic approach. Thus, our solution enables to automatically detect and create alerts when malware attacks those devices. There are a lot of difficulties to deal with the problem. First, collected logs belong to human-generated data classes and have high potentials in volume, velocity and variety characteristics. They also contain the huge number of data features and are extremely noisy (e.g. over time duplicated information, data with evolving specific characteristics) for our detection purpose. Therefore, diverse exploratory data analysis techniques are applied to eliminate noises and inaccuracy in imbalanced data in order to enhance data quality and to find patterns that do not conform to the expected normal behaviors. Next, suspicious activities often have low occurrences that causes imbalanced data classes. So, besides data processing mechanisms, we also have to use effect ML methods covering support vector machine (SVM), logistic regression (LR) and artificial neural network (NN) to mine data. Choosing the best model for the collected data is also another important effort and contribution in our work. Last but not least, our proposed solution has scalability and can support extensive data analysis as well as model developments with data increase case in the near future.

The paper is organized as follows. In Section 2, related research works are categorized and discussed to highlight the differences and contributions of our work in comparison with existing efforts. Section 3 describes raw log properties collected from mobile devices. The logs show that there are many challenges to process data because its complexities. The architecture design for our detection system is presented in Section 4, where expresses the flexibility composition through the integrability of block components. We also present the usage of specific mechanisms like sliding windows, feature processing, lemmatization, term weighting and so forth to deal with the raw data. To find out the anomalies situation, we apply different ML methods that are also introduced here. At the end of this section, our scalable deployment is described in detail, where parallel and incremental learning techniques are exploited thoroughly. While our experiments, achieved potential results and their evaluations are depicted by Section 5 in detail, conclusion and future works are presented in the last section.

2. Related work

Cyber-security is the set of technologies and processes designed to protect computers, networks, programs, and data from attacks, unauthorized accesses and changes, or destruction Mukkamala et al. (2005) and Dua

and Du (2016). For example, anti-virus software and intrusion detection systems (IDSs) will help discover, determine, and identify unauthorized usages, duplications, alterations, and destructions for information systems. Currently, one of the attractive research topics in cyber-security field is malware detection in computing devices. To recognize threats, data analysis techniques usually are used widely and also presented in many recent works such as Buczak and Guven (2016) and Tam et al. (2017). Roughly, the techniques are exploited to detect suspicious software can be divided into the following categories: dynamic, static and heuristic.

Principle of *dynamic analysis techniques* is to test suspect software in a controlled environment (e.g. computers, mobiles or specific networks) using different tools such as debuggers, process monitors, package sniffers, sandboxes, behavior and influence monitoring. However, modern threats even have the ability to detect artificial testing environments and become dormant Tam et al. (2017) to wait for other harm opportunities.

In *static analysis techniques*, information about programs or their behaviors consists of explicit and implicit observations, which are stored in binary/source code. The most well-noted approach is misuse/signature-based mechanism that is a common and effective method used by current antivirus software. The mechanism relies on identification of unique signatures, which point out clearly suspicious pieces of software if they occur during operation process. While this approach offers the fast and effective mechanism, it still has limitations. Concretely, it is unable to detect new one, zero-day and obfuscated threats/malware. An example of obfuscated threat is metamorphic viruses, which can themselves changes their internal structure. This dangerous function has provided an effective means to avoid the signature detection solution Vokorokos et al. (2017). In general, it can be seen that the static techniques cannot cope effectively with polymorphic malware today.

Heuristic analysis techniques operates based on experience basis. They make an attempt to generalize and learn the characteristics of threats and intrusion behaviors that differentiate them from benign software. Usually, the techniques are built by integrating several methods together (i.e. hybrid method), in which malware are either determined by experts or ML data analysis models in order to detect a suspicious or benign behaviors from unknown applications. The detection processes usually cover data collection and analytics stages. In conjunction with mobile context, some studies relate to our work that are presented and discussed below.

Mobile data collection and user modeling. The authors of Woerndl et al. (2010) present a framework, which enables to collect data about user actions on mobiles. The monitored logs include call and message histories, peripheral devices, media players, Global Positioning System (GPS) sensors, networking, personal information management, web browsing, system behaviors and application resource utilization. However, although data achieved from the framework is ready for data analytics process for detecting anomalies, unfortunately, the goal of work is only to model mobile users. Using behavioral context to model users also is mentioned in Verkasalo and Salmeron (2009), where the work's authors analyze contextual behaviors of mobile users from data collected covering user movements (via GPS), time distribution of events (SMS, calls histories) and application usages. The study presented in Papliatseyeu and Mayora (2009) proposes another work-in-progress dedicated to recognize and predict mobile user activities also using positioning information. As a result, the research focuses on only optimizing routing and enhancing performance of wireless networks. Nonetheless, the works described above still did not refer to malware issue, which is dealt in this our work.

Mobile data analytics for anomaly detection issue. Recently, in line with the development and dissemination of smart-phone, there are many existing studies approaching to the anomaly detection problem for this device. In Buczak and Guven (2016), the authors introduce a careful survey in applying ML and data mining (DM) methods to the detection problem. In this way, the work describes complexity of addressed algorithms, discusses challenges in using them for mobile cyber-security. Through analyses, the authors propose recommendations on usage of ML and DM methods with strong emphasis on data diversity aspect. In addition, basic principles of anomaly detection mechanisms using deviation measurements of network, system behaviors, and identification as compared with normal situations also are presented in this paper. In summary, main contribution of the work is a data customization technique to adapt to different systems, application types, and networks. The customization creates difficulties for hackers to know which activities they can attack without detections. Finally, alerts can be generated based on collected anomaly data, which

furthermore is used to define signatures for misuse detectors.

The authors of Shamili et al. (2010) carry out experiments with SVM algorithm in order to recognize suspicious software on mobile network. In most cases, these software often sends out an SMS in a period time. Therefore, the authors use a dataset delivered by a MIT reality mining project, which consists of phone calls, short messages, and data communication logs that are collected via a special application installed on mobiles of volunteers. Achieved outcomes show effectiveness of the authors' approach with their given dataset.

In Shabtai et al. (2012), the authors propose a framework for detecting malware on Android devices. The solution continuously monitors various features and events, then applies supervised ML to classify the collected data into normal or anomaly types. In this direction, system metrics such as CPU consumption, WiFi network overload, number of running processes, battery level and so on are employed to examine similarities with system metric patterns generated previously to detect anomalies. The approach is useful for finding out continuous attacks (e.g. DoS, worm infection), although its model needs to be trained on a broad range of examples as well as a large collected data. In comparison with this work, although the utilization metrics also are a part of our collected logs, our proposal focuses on discovering modern malware intrusions, which do not immediately cause high resource utilization over mobile devices.

Dealing with online learning techniques, the authors of Beygelzimer et al. (2015) bring forward an online boosting mechanism, which converts any weak online learners into a strong online learner. The learning algorithms described in Gama et al. (2014) use existing examples in succession to update their predictor module. The main task of this work is to propose some important weights for these examples. Based on the weights, sampling can be accepted or rejected in updating their predictor. Nevertheless, the techniques still do not apply to suspicious activity detection on mobile issue in comparison with our work.

Mobile data analytics focuses on behavioral detections. While the main interests in this research group are code analysis, CPU, memory consumptions, network overload and/or positioning related problems, the less interests are behaviors, especially in relation with human factor. Works described in Bose et al. (2008) propose a behavioral detection framework to discover suspicious software instead of the signature-based solutions on mobiles. Because behavioral detection assesses the overall effects instead of just specific payload signatures, it is more resilient than polymorphic code obfuscation. The works' authors declare that their approach has high potential in finding out new malware and preventing zero-day worms. The reason is that new malware are often constructed by replacing obsolete modules with fresh ones. Consequently, they share similar behavior patterns with existing malware. The authors of Zainuddin et al. (2014) study anomaly behaviors in mobiles through incorrect application operations, bad usability, not responding, crashed and data loss.

As compared with the existing studies discussed above, the major contributions and differences of our work include:

1. We focus on using logs generated from application/service activities to find out anomalies during mobile operation. Essentially, the activities are made by device owner, hence in other word, our solution enables to model mobile users through appliance behavior logs.
2. Because of using heuristic techniques, our approach is expected to find out modern malware that may not make unauthorized operations immediately as soon as they penetrate successfully. Instead of, they nestle inside devices for a while to wait for suitable attack opportunities.
3. We design a complete system deals with many problems related to data preprocessing. Concretely, the collected data shows that it is large-scale, complicated, and noisy. In addition, it also continuously augments by the time, however suspicious activities often have low occurrences. Our system thus must have several specific mechanisms and techniques in order to achieve satisfactory effects.
4. We propose a novel lemmatization module to process our raw logs. The component is changed to suit the collected logs with specific characteristics of Android mobile devices.
5. Evaluations and remarks in using three ML methods including SVM, LR and NN are given based on real data logs gathered from mobiles in malware detection context.
6. To resolve scaling issue of data, we develop operation model using advanced cross-industrial technologies for our detection system. Otherwise, incremental learning technique also is applied to increase

desired performance, efficient memory utilization anf data management. Entire our system is deployed and operated toward scalable solution design for large-scale mobile network.

3. Raw log description and taxonomy

In this section, we describe about log collection process and data features. All devices operate in our testbed are monitored in real-time and they sends raw logs to our Logger server in predefined time period. Every mobiles is marked by their International Mobile Equipment Identity (IMEI) and can be dynamically located by positioning longitude and latitude values (via GPS). Thus, device movements can be described by three dimensions space with x, y, and z values from device accelerometer. Each mobile user is recognized via its userID. However, at this stage of our work, we assume that only one user employs one device. Therefore, our userID also can be considered as the same as device's IMEI. Because of high expenses to equip a large number of devices for volunteers, the data collection process is done by our technological partners, which is also devices' owner.

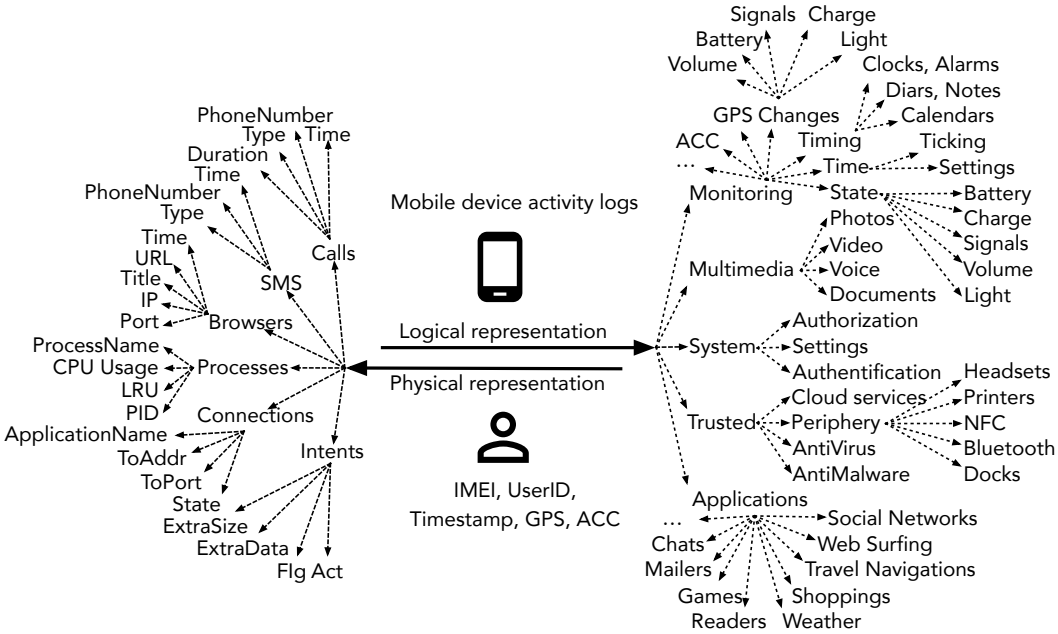

Figure 1: Simplified physical and logical taxonomy representations of mobile device logs

While our monitoring agents are deployed on Android operating system, the intrusion activities are generated by modifications of external metasploit library in order to investigate vulnerability Rapid7 (2017). The goal is to collect raw logs including threats for our work. Raw logs are recorded containing device activities as follows: history of calls, SMS and browser, intents, sampling of processes and connections. These raw data will be processed, analyzed and cleaned in the Logger server by several mechanisms, which are presented in the next section. Our collected logs from mobile devices can be categorized into groups according to physical taxonomy as illustrated by Figure 1 and paragraphs below.

- Calls: Timestamp, Time, PhoneNum, Type, Duration ...
- SMS: Timestamp, Time, PhoneNum, Type, ...
- Browser history: Timestamp, Time, URL, Title, IP, Port, ...
- Processes: Timestamp, ProcessName, CPU usage, LRU (Least Recently Used), PID, ...

- Connections: Timestamp, Application, ToADDR, ToPort, FromADDR, FromPort, State, ...
- Intents: Timestamp, Act, Flg, ExtraSize, ExtraData, ...

More details and full descriptions of raw log structures are available in Section 8. Full names of intents, processes and connection types are available in AndroidDevelopers (2017) and in vendor technical manuals (Asus, Lenovo, LG, Samsung, SonyEricson). Raw data (mobile device logs) belongs to human-generated data class, which has high potential in all 3Vs (Volume, Velocity and Variety) as introduced before but they are not so big as machine-generated data. Concretely, just a half of gigabytes (GB) per device per day without multimedia are gathered from each twenty devices of volunteer users. The data analysis also shows that the collected data from real operation environment contains high number of features (i.e. *high-dimensional* data). In addition, the data also is *extremely noisy* for our detection purpose. Two typical notable problems could be mentioned including over time duplicated information about processes as well as connections, and missing values due to difficulties in the data collection phase in real environment. From this viewpoint, to reduce features before analytics, all physical raw logs are mapped into logical classes, which cover monitoring, multimedia, system, trusted and user's applications. The logical representation taxonomy of raw logs is expressed by Figure 1.

- Monitoring: information services include timing, signal and synchronizations (WiFi, Bluetooth, NFC), battery states, battery state changes, flashlights, power states, power state changes, charging, positioning (GPS), accelerometer values, background settings, light intensity, volume, status monitoring and changes, time setting, time ticking, calendars, diaries, weather, hours, alarms, and so on.
- Multimedia: data generated from photos, video, voice (calls, music), text (SMS, documents, notes);
- System processes such as settings, authentication and authorization;
- Trusted applications are antivirus, antimalware, cloud and periphery services;
- User applications are used and installed based on user's demands for various purposes such as web surfing, sending and receiving emails, discussions, chats, social networking, collaborations, games, shopping, travel navigation, reading, and so on.

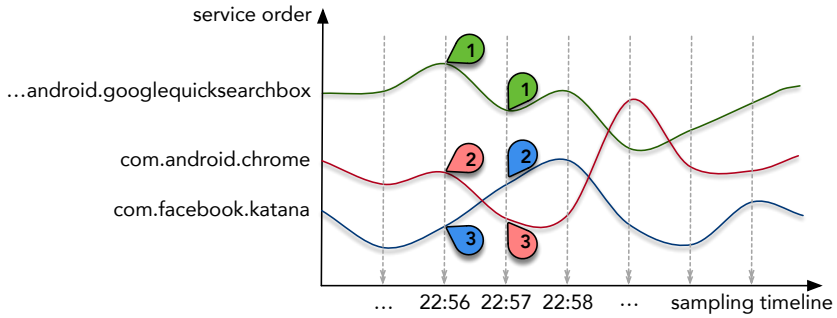

Figure 2: Simplified illustration of OS service order for three processes in relation with sampling timeline

With the goal of detecting malware intrusions on mobiles, we only focus on system and user application classes to collect and analyze logs. Main reason for the elimination is that log types monitored from the other classes do not provide sensitive data for our detection purpose. User's activities on mobiles are collected by two ways as follows:

- Browser, call, SMS histories and intents are gathered gradually as soon as they appear on devices.

- Processes and connections are collected in the manner of sampling in time periods, which are changed based on stored data amount. Because applications and processes are sampled in predefined intervals (nearly per every minute), the log data contains redundant. However, such repeated information from the sampling way denotes that processes running on mobile operation system (OS) are dynamical. This phenomenon is expressed by Figure 2.

Figure 2 illustrates a simplified situation of OS service orders in relation to sampling timeline for three processes *com.google.android.googlequicksearchbox*, *com.android.chrome* and *com.facebook.katana*. It can be seen that at the time of 22:56, the sampled process order is search, chrome and facebook from top to down. In the next sample time at 22:57, the order is search, facebook and chrome. Another observations can be made here consisting of almost all applications are running on the devices as background processes without strictly *start* and *stop* states and the relationship between each pair of applications, processes and connections are not one-to-one correspondence. For instance, the application *facebook* has several processes, namely *com.facebook.katana*, *com.facebook.katana:dash*, *com.facebook.katana:nodex*, *com.facebook.orca* and *com.facebook.pages.app*, but only processes *com.facebook.katana*, *com.facebook.orca* and *com.facebook.pages.app* relate to web connection. This concrete example shows the complexity of monitoring processes states in our testbed. The collected process logs is only one of many data types, which we gather during carrying out our project.

4. Designing system

Our proposed system architecture is built based on anomaly detection principles and focusing on behavioral aspect of mobile device users. The system creates trigger alarms as soon as detecting objects, which behave significantly differently from predefined normal one trained from history data. In this way, there are two data processing stages inside our system including training and detection.

- In the training stage, ML and nature language processing (NLP) techniques are applied to extract and select useful information from raw logs. The stage goal is to generate the best data model to distinguish normal from suspicious behaviors surrounding malware intrusions.
- In the detection stage, device activities are monitored in real-time and collected in predefined time intervals. Thus, the raw logs are transfered to the Logger server, where they furthermore are transformed to ML observation models. After that, suspicious probability will be rated using the trained model in the previous stage. At the time, our solution can effectively distinguish normal from suspicious behaviors. It also can detect last unknown intrusions under the same or similar behavior patterns.

Because such detection process is usually marked by a high rate of false alarms, hence our extraction and consequent selection of appropriate features in the training stage target to improve significantly the effectiveness of detection results. The scheme of our proposed architecture is shown in Figure 3. It contains five main components, including: *Monitoring*, *Log preprocessor*, *Feature processing*, *Learning framework* and *Decision support*.

- **Monitoring** component is taken shaped from two parts: agents are installed on every devices to collect logs and Logger server, which stores collected data and sends back alerts to mobiles.
- **Log preprocessor** component is responsible for reducing data complexities through the following techniques: sliding window, log filter, and stemming-lemmatization in order to eliminate noise and features.
- **Feature processing** component enables mechanisms to represent log terms and select data features.
- **Learning framework** component's goal is behavioral analysis using classification methods.
- **Decision support** component transfers analyzed data back to Logger server, where creates warning alerts sent to mobiles.

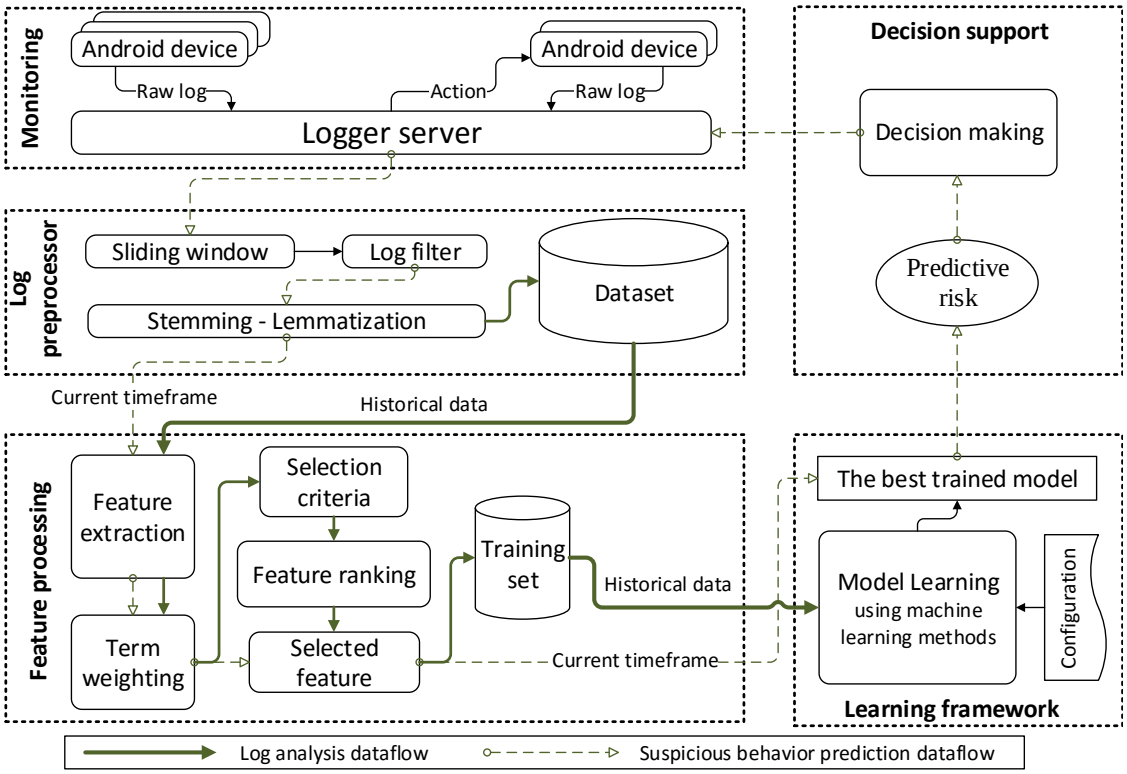

Figure 3: Overall architecture

The monitoring component and its operation mechanisms already are described in Section 3. In the following sub-sections, we concentrate on presenting the remainders.

4.1. Log preprocessor component

NLP is a field of computer science and engineering that has developed from language and computational linguistics study within artificial intelligence topics as in Hardeniya (2015). In a practical context, NLP is analogous to the way of enabling computer system to understand document meanings. Some of the most common tasks understanding words, documents or forming grammatically correct sentences that are very natural to humans. The basic idea of applying NLP to our log preprocessing component is to formulate a understandable communication way from mobiles to our system. Collected logs are considered as a kind of language statements, which contain all activities gathered from devices and users.

As presented before, the specific role of *Log preprocessor* component is to reduce data size and dimension, which are collected by the first component. Thus, log preprocessor consists of three modules:

- Sliding window
- Log filter
- Stemming-Lemmatization.

4.1.1. Sliding window

Malware, phishing and other threats have become more challenging and difficult to address ThreadTrack-Security (2015) and ENISA (2017). At the present, attackers can coordinate their attacks among various delivery venues, including email, web, social media, files, attachments, and so on. They also can remain dormant for an extended period. In this way, the current malware are less likely to be detected by many traditional anti-phishing and anti-malware solutions.

From the behavioral viewpoint, malware intrusion progress includes a sequence of activities and events that usually takes a certain time to finish. Therefore, for the behavioral distinguished purpose, the sliding window technique is opted to transform log series into separate time frame sub-collection of activities. As presented previously, in each time interval, there are six log types collected in the Logger server. The changes of logs against time reflect mobile device states that are in a high level corespondent to reality. The map of sub-collections into a six-tuple is defined by Definition 1 as follows:

Definition 1. *The representation of device state d in a time-frame (or sliding) window w is a six-tuple* $d = (a, s, b, I, C, P)$

where

$a \in \mathcal{N}_0$ *is the number of phone calls in* w
$s \in \mathcal{N}_0$ *is the number of SMS in* w
$b \in 0, 1$ *is the number of browser's use in* w
$I = \{it_1, it_2, \ldots, it_n\}$ *is a set of collected intents in* w
$C = \{p_1, p_2, \ldots, p_m\}$ *is a set of applications that use connections in* w
$P = \{p_1, p_2, \ldots, p_k\}$ *is a set of processes servicing by OS in* w
$n \neq m \neq k; \quad m \leq k; \quad n, m, k \in \mathcal{N}_0$

The mapping functions are expressed by the following formulae:

$$a = \sum_{i=t_{begin}}^{t_{end}} count_i(calls) \tag{1}$$

$$s = \sum_{i=t_{begin}}^{t_{end}} count_i(SMSs) \tag{2}$$

$$b = sign\left(\sum_{i=t_{begin}}^{t_{end}} count_i(URLs)\right) \tag{3}$$

$$I = \bigcup_{i=t_{begin}}^{t_{end}} Set_i(intent.Act) \tag{4}$$

$$C = \bigcup_{i=t_{begin}}^{t_{end}} Set_i(connection.Application) \tag{5}$$

$$P = \bigcup_{i=t_{begin}}^{t_{end}} Set_i(process.Processname) \tag{6}$$

where $count_i()$ is the number of records within timestamp i; Set_i is value set of a log field at timestamp i; and (t_{begin}, t_{end}) is the time frame of log sub-collection.

The mobile device state denotes an user's normal or suspicious behavior in one time frame. In our work, malware intrusions are generated url-based spreading threats, which is the most frequent malware intrusions on mobiles as described above in Section 1 and Section 3. Transitions of mobile device states according to time frames are described by Figure 4. After applying sliding window technique, we use the six-tuple d to data reduction with two modules as follows:

- *Log filter* module eliminates data noise.
- *Stemming-Lemmatization* module unifies and reduces features.

4.1.2. Log filter

From the logical representation aspect of mobile device logs, there are many log types, which are not related to malware intrusions, for example, activity logs, which include multimedia. In this direction, our Log

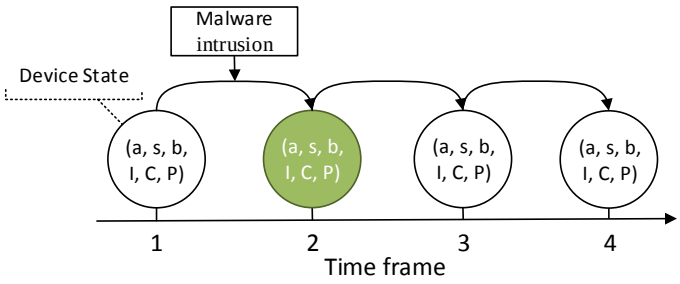

Figure 4: State transitions in a mobile device based on the physical log representation

filter involves three lists: I_{filter}, C_{filter} and P_{filter} that contain a lot of insensitive intents, connections and processes respectively belonging to multimedia and trusted categories (applications or service provided by verified vendors like Google, antivirus and antimalware) illustrated by the logical taxonomy representation (Figure 1). The *Log filter* module eliminates such data from I, C and P sets by the following way:

$$I_{fitered} = I \setminus I_{filter} \quad (7)$$
$$C_{fitered} = C \setminus C_{filter} \quad (8)$$
$$P_{fitered} = P \setminus P_{filter} \quad (9)$$

4.1.3. Stemming-Lemmatization for mobile logs

Due to diversity of mobile devices, which are manufactured by different producers, there are many differences during log processing. For instance, applications (and their processes) may have the same name, but their versions running on different devices have slightly diverse full names. As a result, a lot of morphological log variants for these processes and connections must collected. This is also one of main reasons that causes high dimensionality and duplication in the data obtained from *Sliding window* and *Log filter* modules. To resolve the issue, distinct name of the same process and applications are mapped into their shorter and common parts. In this direction, data complexity is reduced and data quality for learning process is improved significantly.

Stemming and lemmatization are two word form normalization techniques used widely in NLP to condense various forms of a word into an unique single term. While stemming employs rules to strip off suffix of a word and thus gain the stem, lemmatization maps a word into its dictionary, which is known as a lemma. In comparison with stemming, the advantage of lemmatization is that it uses lexical resources created manually by developers with concrete domain knowledge.

In this work, we develop a specific tool called LogLemmatizer to perform our stemming-lemmatization process as described in Algorithm 1. The tool analyzes data corpus of (a, s, b, I, C, P) preprocessed previously by Sliding window and Log filter. The main differences between our LogLemmatizer and general NLP stemming-lemmatizer can be listed as follows:

- A term in mobile logs such as names of applications and processes are different than a word because *"log"* is not a natural language. Usually, process names frequently have the following structure: *term* = <*name*>.<*name*>...<*main_name*><*name*>:<*subprocess_name*>.
  For example, *com.facebook.katana:nodex* or *com.android.chrome:sandboxed*. The number and order of <*name*> parts are changeable. Hence, prefix and suffix of log *terms* can not be removed using traditional information extraction methods in NLP domain and our LogLemmatizer's goal is to extract the $< main\_name >$ from the raw structure.

- In our module, two terms C and P in the corpus (I, C, P) have the same lemma if they indicate two identical processes/applications or a application group that has the same function.

As presented above, we develop the LogLemmatizer based on the lemmatization algorithm in NLP. The operation of LogLemmatizer is described in Algorithm 1, which is composed from two phases: *rule-based*

Algorithm 1: LogLemmatizer for mobile device logs

input : log term t; stemming dictionary D; lemmatization rules R
output: lemma

```
1  primary_term ← None
2  lemma ← None
3  while t.endBy(":<subprocess_name>") do
4  |  t ← t.remove(":<subprocess_name>")
5  end
6  while t.endBy(".<name>") do
7  |  t ← t.remove(".<name>")
8  |  if name ∈ D then
9  |  |  primary_term ← name
10 |  |  break
11 |  end
12 end
13 lemma ← primary_term
14 for rule ∈ R do
15 |  if primary_term satisfy rule then
16 |  |  lemma ← rule(primary_term)
17 |  end
18 end
19 return lemma
```

lemmatization (from line 3 to 12) and *lexicon lookup* (from line 13 to 18). The objective of *rule-based lemmatization* is to find the term root. While *lexicon lookup* is based on dictionary D and set of rules R that are domain-specific lexical resource developed for mobile logs. The operation of entire component *Log preprocessing* is shown in the following Procedure LogPreprocessing.

Procedure LogPreprocessing($Call$,SMS,$Browser$,$Process$,$Connection$,$Intent$)

input : Raw Logs: $Call$,SMS,$Browser$,$Process$,$Connection$,$Intent$
output: S: a set of 6-tuples (a, s, b, I, C, P)

```
1  Divide the period of logs into a set of time frame W
2  S ← ∅
3  foreach w = (t_begin, t_end) ∈ W do
4  |  Calculate a, s, b, I, C, P by equation 1, 2, 3, 4, 5, 6
5  |  Apply log filter for I, C, P by formula 7, 8, 9
6  |  foreach term t ∈ I, C, P do
7  |  |  t ← apply Algorithm 1 for t
8  |  end
9  |  S = S ∪ (a, s, b, I, C, P)
10 end
11 return S
```

4.2. Feature processing component

After transforming terms into lemmas as described in the previous Section 4.1, each data record or document in (a, s, b, I, C, P) is furthermore organized as sets of intents, connections and processes with unique names in time frames as follows.

Definition 2. *The data record d in the sliding window w after lemmatization process is a 6-tuple* $d = (a, s, b, I, C, P)$

where

$a \in \mathcal{N}_0$ *is the number of phone calls in* w

$s \in \mathcal{N}_0$ *is a number of SMS in* w

$b \in \{0, 1\}$ *indicates if a browser is used in* w

$I = \{it_1, it_2, \ldots, it_n\}$ *is a set of collected intents in* w

$it_i \neq it_j : \forall i \neq j; \quad i, j \in \{1, \ldots, n\}$

$C = \{p_1, p_2, \ldots, p_m\}$ *is a set of applications that use connections in* w

$p_i \neq p_j : \forall i \neq j; \quad i, j \in \{1, \ldots, m\}$

$P = \{p_1, p_2, \ldots, p_k\}$ *is a set of processes servicing by OS in* w

$p_i \neq p_j : \forall i \neq j; \quad i, j \in \{1, \ldots, k\}$

$n \neq m \neq k; \quad m \leq k; \quad n, m, k \in \mathcal{N}_0$

4.2.1. Feature extraction with bag-of-words representation

The bag-of-terms representation with *tf-idf* weight scheme is applied to extract features from the data corpus. In this way, each document is represented by a vector depending on the terms that they cover $d = \{w_1, w_2, \ldots, w_n\}$, where w_i is the weight of i-th term in the document d. Let the frequency f_i be the number of occurrences of i-th term in the document, and let the frequency df_i be the number of documents containing i-th term. The weight w_i of i-th term is determined by the following formula:

$$w_i = tf_i.idf_i = (1 + log(f_i)) \cdot \left(1 + \frac{|D|}{df_i}\right) \tag{10}$$

where $|D|$ is the number of document in the corpus. In our work, *"document"* is correspondent to one time frame of collected activities in a mobile device. The transformed record from a time frame is considered as one observation (or sample) in ML datasets.

4.2.2. Feature selection

As presented in Subsection 4.3, each document/data record is classified into two categories: normal or anomaly. For supervised learning, set of training examples, denoted by T, has the following form:

$$T = \{y, d\} = \{y, (t_1, t_2, t_3, \ldots, t_n)\}$$

where $y \in \{-1, 1\}$ indicates normal or suspicious data record, $(t_1, t_2, t_3, \ldots, t_n)$ are the set of terms in document d. Expected information gain (IG) of term t - a measurement of information obtained for prediction categories by knowing the presence or absence of a term in a document, is defined as follow:

$$\begin{aligned} IG(y,t) &= H(y) - H(y|t) \\ &= -\sum_{y \in \{-1,1\}} P(y) \cdot logP(y) + P(t=1) \cdot P(y|t=1) \cdot logP(y|t=1) \\ &\quad + P(t=0) \cdot P(y|t=0) \cdot logP(y|t=0) \end{aligned} \tag{11}$$

where $P(y)$ is probability of the document category, $P(t)$ is probability of presence $(t = 1)$ or absence $(t = 0)$ of a term. Usually in decision tree, IG is biased towards choosing features with a large number of values as root nodes and gain ratio (GR) is a modification that reduces this bias by taking into account the number of branches (intrinsic information - IV) that bring better result before making the separation.

$$\begin{aligned} IV(y,t) &= -P(y|t=1) \cdot logP(y|t=1) - P(y|t=0) \cdot logP(y|t=0) \\ GR(T,i) &= \frac{IG(T,i)}{IV(T,i)} \end{aligned}$$

Both IG and GR are used to rank the importance degree of extracted features (i.e. the higher value is the rank, the more important feature is Schutt and O'Neil (2013)). After ranking, all the terms have

higher values than a configured threshold are selected and hence other lower values are ignored. In this way, the data dimension is effectively reduced and the performance of training process is improved. In this work, IG is the chosen rank due to over time duplicated information related to processes and connections (as illustrated by Figure 2) and features are nominal after data transformation and reduction.

4.3. Learning framework component

ML techniques in general and supervised learning approaches in particular play the central role in many researches and commercial cases Dua and Du (2016). If the use of supervised learning is realizable, it is preferred over unsupervised learning due to the higher prediction accuracy. The objective of our model is to detect suspicious activities around malware intrusions on mobile devices as described in Section 1 and 3. This kind of malware intrusions is the most frequent one but does not affect device performance immediately. Thus, each separated activity performed by user is legal with execution permission, but a group of them after certain time may lead to unwanted situations. Consequently, the suspicious class describes groups of actions that can lead to dangerous consequences. The problem of suspicious activity detection is to find a classification function, which maps operations performed in mobiles to suspicious or normal class:

$$y = \gamma(d) \tag{12}$$

where d is state of mobile device, y = -1 if d suspicious and $y = 1$ if S is normal. In the next sections, we present our approach to determine system state d based on mobile logs and to find classifier γ. In this research, three supervised learning methods are employed to categorize states: SVM, LR and NN Tong and Koller (2001); Sebastiani (2002) to conduct the best model. These methods is selected due to their capabilities of classification and scalable realization as described below in the Section 4.6. When each state of mobile devices is represented by a document (a, s, b, I, C, P), the learned classifier is expected to map precisely each document to a label y:

$$y = \gamma(a, s, b, I, C, P) \tag{13}$$

4.3.1. Support Vector Machine

SVM is useful technique for data classification by finding the optimal hyperplane that separates training example into two types: positive and negative. The separating hyperplane is used to maximize distance (margin) between the two parallel hyperplanes. On the training set of n instance-label pairs (y_i, d_i), finding such hyperplane for text corpus can be translated into the following optimization problem:

$$\begin{aligned} \min_{w,b,\epsilon} \quad & \frac{1}{2} w^T w + C \sum_{i=1}^{n} \epsilon_i \\ \textit{subject to} \quad & y_i(w^T \Phi(d_i) + b) > 1 - \epsilon_i, \quad \epsilon_i \geq 0 \end{aligned}$$

where $w = \sum_{i=-1}^{n} \alpha_i \Phi(d_i)$. The classification function is as follows:

$$f(d) = \sum_{i=-1}^{n} \alpha_i \Phi(d_i)^T \Phi(d) + b \tag{14}$$

where $K(d_i, d) = \Phi(d_i)^T \Phi(d)$ is kernel function, which can be linear, polynomial or radial basis. To avoid overfitting, lambda values of L2 regularization is applied to a per-example basis in incremental learning.

4.3.2. Logistic regression

LR is statistical technique that tries to estimate probability that a dependent variable will have a given value, instead of estimating value for the variable. This technique classifies a sample by predicting probability

of each label, which it belongs to. For the binary classification of document d, probability of each class is defined as follows:

$$p(y=1|d) = \frac{1}{1+exp(-\theta^T d)} \quad (15)$$

$$p(y=-1|d) = \frac{1}{1+exp(\theta^T d)} \quad (16)$$

The learning goal of logistic model is to find the parameter vector θ that maximizes the log-likelihood of training set (y_i, d_i) $(i = 1, \ldots, n)$:

$$\theta = arg\max_{\theta} \sum_{n}^{i=1} logP(y_i|d_i, \theta) \quad (17)$$

4.3.3. Artificial Neural Network

The sigmoidal feedforward network is also used for our classification. Input of the network is each documents d_i and output is label y_i. The training process is to find out all the weights (w_{ij}), which minimize the classification error:

$$\xi = 1/n \sum_{i=1}^{n} (y_i' - y_i)^2 \quad (18)$$

Our neural network architecture is configured with three layers: input, hidden and output. This network is trained by back-propagation algorithm. First, each input neuron corresponds to a feature of training data, consequently the number of input neurons equals the feature number in training dataset. Next, the number of neurons in output layer corresponds to the number of classification classes. Finally, the number of neurons in hidden layer is customized to conduct models with the best performance evaluations.

4.3.4. Model Learning

The operation of Learning framework module is introduced by Algorithm 2. The module output is to bring forward the best ML model under performance and stability criteria, which can distinguish effectively normal from suspicious behaviors surrounding malware intrusions in the detection stage. This algorithm denotes main parts of the training stage in our design, including feature weighting (from line 3 to line 7), feature ranking and selection (from line 8 to line 12), generating training configurations with various time frames, datasets and the number of parameters (from line 16 to line 19), training models with setting configurations for SVM, LR and NN methods (from line 20 to line 33) and the best model selection. The learning process is realized in the manner of batch-incremental learning as described in Read et al. (2012). Thus, our models can be updated continually and based on that bring significant efficiency for memory utilization and data management.

4.4. Decision support component

Suspicious probability will be rated using trained model in the training stage. In this direction, our solution detects suspicious activities surrounding malware intrusions as expressed by Algorithm 3. It can also deal with previously unknown intrusions under assumption of the same behavior intrusion patterns.

4.5. Algorithm analyses and time constraints

In this subsection, we analyze the complexity and time constraints of our proposed model to validate its implementability. The model covers two main workflows: *Log analysis* and *Suspicious behavior prediction* as shown in Figure 3 by continuous and intermittent arrow respectively.

- The operation of *Log analysis* workflow is manifested by Algorithm 2, which takes all the historical raw logs as input data and returns the best trained model.

Algorithm 2: Incremental learning framework from mobile device log history

input : raw logs, timeframe
output: best trained model

Init time interval I for training datasets
Init training set $T \leftarrow \emptyset$
D = LogPreprocessing(raw logs in I)
foreach *document* $d \in D$ **do**
    calculate $d = \{w_1, w_2, \ldots, w_n\}$ by equation 10
    $y \leftarrow$ label(d)
    $T \leftarrow T \cup (y, d)$
end
foreach *term* $t \in T$ **do**
    calculate $IG(t)$ by equation 11
    if $IG(t) < threshold(t)$ **then**
        remove term t in T
    end
end
Divide T into batches $\{T_1, T_2, \ldots, T_j\}$
$CF_{SVM} \leftarrow$ generated set of SVM configurations (method, settings)
$CF_{LR} \leftarrow$ generated set of LR configurations (method, settings)
$CF_{NN} \leftarrow$ generated set of NN configurations (method, settings)
$CF_{ML} \leftarrow CF_{SVM} \cup CF_{LR} \cup CF_{NN}$
$Models \leftarrow \emptyset$
foreach $cf \in CF_{ML}$ **do**
    $cf.method \leftarrow$ method from cf
    $m \leftarrow null$
    foreach $T_i \in T$ **do**
        if m *is null* **then**
            $m \leftarrow$ model trained by $cf.method$ with T_i and configuration cf
        end
        $m \leftarrow$ update m using $cf.method$ with T_i and configuration cf
    end
    $Models \leftarrow Models \cup m$
end
best_model $\leftarrow$ the best model in $Models$
return *best_model*

Algorithm 3: Suspicious activity detection

input : current log, threshold
output: trigger alert

d = LogPreprocessing(current log)
calculate $d = \{w_1, w_2, \ldots, w_n\}$ with selected terms by equation 10
calculate probability y based on d using the trained model from Algorithm 2
if $y \geq threshold$ **then**
    set trigger alert for the given mobile device in the Logger server
end

- The *Suspicious behavior prediction* workflow, which is presented by Algorithm 3 takes raw logs in only one time frame as input data to make prediction and corresponding decisions.

In *Log analysis* workflow, historical raw logs collected in the Monitoring component are firstly processed by the *Log preprocessor component* and then stored as a preprocessing dataset. After that the dataset are passed to *Feature processing* for feature ranking (Section 4.2.2) and consequently a set of selected features is created from it as a training dataset for Learning framework. The final result of *Log analysis* workflow is the best trained model (SVM, LR, and NN) for suspicious activity detection. This model is used in the detection stage. From the view of Log analysis dataflow, its time complexity includes the time complexity of each component (i.e. *Log preprocessor*, *Feature preprocessing* and training process of *Learning framework*).

Assume that N is the number of terms in collected raw logs, and n is distinct transformed log terms that are process names, connections or intents. The *Log preprocessor* component has linear complexity $O(N)$ in relation with the of terms in collected raw logs because of the fact that all its three modules has linear complexity. The *Sliding windows* and *Log filter* module explores all logs once and transform them into time frames according to formulae from 1 to 9. The complexity of *Stemming-Lemmatization* module is also $O(N)$ due to the fact that each term is processed by this module only one times (Algorithm 1) to transform raw terms into unique and distinct terms.

In *Feature processing* component, the *Feature extraction* module processes in all terms and records, therefore its time complexity is $O(n.|D|)$, where $|D|$ is the number of records in dataset. The *Feature selection* calculates IG of each term by equation 11, and rank all of them by IG order. As a result, it takes

$$O\left(n.|D| + n.log(n)\right) \tag{19}$$

to finish.

Run time of *Learning model* in *Learning framework* denoted by $T_{training}$, which depends on many factors such as the complexity of chosen ML method, sizes of training and testing datasets, parameter settings specified in configuration set and termination criteria and so on. Because ML approach is incremental learning, the run time of Learning model can be considered as near linear complexity in relation with sizes of training and testing datasets. In summarization, the total time complexity of *Log analysis* workflow as denoted in Algorithm 2 is:

$$T_{log\_analysis} = O(N) + O(n.|D|) + O\left(n.log(n)\right) + T_{training} \tag{20}$$

In *Suspicious behavior prediction* workflow, logs in current time frame of each mobile is used as input data for detection using the best ML model obtained from *Log analysis* workflow. In *Log preprocessor* and *Feature processing*, dataflow (current logs), which corresponds to line 1 and 2 in Algorithm 3 is similar to the flow in *Log analysis* workflow.

Assume that k is log size in current time frame from a device, M is the number of mobiles to be analyzed in order to warn if suspicious behavioral activities are in the time frame. Time complexity of these two components is $O(k.M)$. The lines from 3 to 7 are some simple computations with the complexity $O(k.M)$. The total time complexity of *Suspicious behavior prediction* workflow is $O(k.M)$.

Starting from analyses above and based on the fact that $k \leq n$, it is clear that run time of *Suspicious behavior prediction* workflow is insignificant if M is not very large, especially with supports from modern technological hardware. Otherwise, Algorithm 3 should be planned with adequate time interval $T_{detection}$. The scalable solution for huge number of M is presented in data-centric strategy at Section 4.6, which requires certain up-to-date high-performance technological support for fast real-time prediction realization.

In order to update ML model with new incoming data, updated time interval T_{update} is required to be planned under the following condition $T_{update} \geq T_{log\_analysis}$. In practice, the value of T_{update} can be set in month or year interval, which is much longer than $T_{log\_analysis}$. However, to reduce $T_{log\_anlysis}$ for massive ML data processed inside our scalable solution, a computational efficient learning framework was proposed in our previous work Hluchý et al. (2016).

4.6. Scalability of the system

As discussed before, our raw logs of mobiles have high potential in all 3Vs and suspicious activity detection function is developed to operate in a long-time production concept with acceptable performance. For this reason, we try to design system at early stage in the way of allowing scalability and adaptability with increase data amount and complexity in the future. Several techniques thus are exploited in our system, involving parallel processing and incremental learning supported by modern technologies.

Firstly, parallelization is a powerful mechanism to increase effectiveness in processing large data amount. There are many ML usecases that can be applied parallel techniques such as model selection with grid search, model evaluation with cross validation, bagging models (random forest), and (ensemble) average models. The common characteristic of these typical examples is that their training and testing models are independent with input data. The essential of parallelizing approach is when input data for model learning is large-scale, it significantly shortens required time for the training stage. In our work, parallel model learning is operated in worker nodes. Each worker process initially takes an input data from the central data repository located in Network File System (NFS) then it trains and tests models independently. The results of all separated parallel trainings covering quality metrics and settings models are returned back to the central data repository for the best model selection. In addition, time needed for the best model selection is negligible in comparison with the time required for the training stage of large-scale data. More details and experiments about this approach can be found in our work Hluchý et al. (2016).

Secondly, while parallelization deals well with multi-model or repeated model training in ML training stage, the efficiency of distributed ML has faced with challenges of data-centric computing platform (e.g. Hadoop), where Message Passing Interface (MPI) operations are not supported fully. That means when data storage is distributed under data-centric parallel processing manner, it is much more desirable to process data in a distributed fashion and avoid the bottleneck of data transfer to a single powerful server. In order to resolve the problem, Vowpal Wabbit (VW) Langford and et al. (2017) ML system is implemented with its own Hadoop-compatible computational model called AllReduce with similar functionality to MPI AllReduce Agarwal et al. (2014). Then, in its daemon mode, VW initializes a spanning tree for servers from gateway nodes to whole Hadoop cluster in order to empower fast real-time prediction response, which is especially desirable and necessarily for the detection stage. More discussion about ML in data-centric and compute-intensive strategy including various considerations about suitable technological realizations based on 3Vs of data can be found in another our recent work Nguyen et al. (2016).

Furthermore, the incremental learning technique helps solve near-linear complexity issue and support one-pass learning over data without memory limitation as introduced in Leskovec et al. (2014) with the possibility to adapt to changes in data by incremental updating, which is described in Section 4.5. The realization of ML models thus is done through VW library, which is scalable extremely fast learning system. Training ML models processes with that library does not require loading whole data into memory at once but incrementally through time frames. Correspondingly, input training dataset can be large-scale without the memory limitation. VW library also runs properly in single machine, HPC cluster and specially in our case is in Hadoop cluster, which is introduced in Hluchý et al. (2016).

Based on the design presented above in conjunction with underlying modern technologies, our system gains scalability, robustness and execution speeds up completion. In the case of ML dataset is massive, the needed time for parallel training process can be significantly shorten; nevertheless, thank to our used preprocessing data mechanisms, this case may occur very rarely in our testbed. Conversely, in the case of collected raw logs are large-scale with 3Vs potential characteristics, the combination of data preprocessing implementation, in-memory operations and Hadoop clusters is more practical. When the fast out-of-core incremental learning system is integrated in such environment, our system can face challenge of monitoring large-scale mobile network in near real-time. Note that these above mentioned advanced technologies do not have to be always coupled together, but the alliance is essential for cross-industry data mining as well as cyber security.

5. Experiments and evaluations

In this section, we present outcomes achieved from experiments with our collected data in practice to demonstrate the feasibility and effectiveness of our proposed system in detecting malware intrusions mainly against the back-door malware group. As presented before, our desired system result is figure out precisely as much as possible that there is malware or not on each test device. The experiments are divided into two groups, including:

- *Data preprocessing and feature-related techniques* is done to manifest positive performance of *Log preprocessor* and *Feature processing* components in terms of reducing data size as well as dimensionality.
- *Learning process* is carried out to select the best model to detect suspicious activities. Thus, three different ML techniques cover LR, SVM and NN are deployed to measure their effectiveness.

To test the system, in our experiments, url-based malware was created using Rapid7 (2017). Normally, this malware type spreads very quickly on mobiles because users frequently do not pay attention to opening links. With its characteristics, the malware tries to open back-doors from inside mobile devices and cause threats related to data and control right.

5.1. Data preprocessing and feature-related techniques

This is the first test group experimented with our system. Table 1 depicts realization step-by-step (i.e it involves four steps) and gained results with different classes using our data preprocessing and feature-related techniques modules. It can make two important observations via the test. First, the number of features is quite high. Concretely, the collected data has more than thousand features as shown through step 1. Second, the data is imbalanced with less than 0.2% of the minority positive class with suspicious behavior indication. This issue also is expressed by step 1 in that Table.

step	description	process	connection	intent	browser	features	records	positive
1	raw data	781	293	73	1	1 148	63 072	125
2	after filtering	383	217	27	1	628	63 072	125
3	after feature selection	48	20	7	1	76	63 004	82
4	after data cleaning	48	20	7	1	76	6 802	76

Table 1: Data cleaning, feature selection and reduction

After data preprocessing and feature-related techniques are run, it could be seen that the size and dimensionality of collected data are reduced approximately to 10% with impurities and duplications already have removed. The achieved outcomes are also described by step 2, 3 and 4 in Table 1. However, the data is still imbalanced with around 1% of positive records for ML as shown by step 4.

The next phase in our system as well as testing process is to evaluate the feature selection task. With that goal, the top features of processes, connections and intents are ranked using entropy as illustrated by Figure 5, Figure 6 and Figure 7 respectively. The approach is theoretically described in Section 4.2 before. Note that the higher value of IG indicates: corresponding features in process, connection and intent group are more valuable for supervised ML methods in classifying suspicious behaviors from normal one based on gathered logs.

The feature selection process is a crucial step to improve data quality for ML and accelerate creation of model without losing the potential predictive power of the data. In this study, it is done by threshold choice step for IG. The thresholds thus are set based on the mean indicator of ranking entropy values as they are graphical denoted by Figure 5, 6 and 7. They are also achieved through repeating evaluations with various levels. By this way, we find out the threshold values as follows: $threshold_{intent} = 0.0005, threshold_{connection} = 0.0002, threshold_{process} = 0.001$. Also based on experimental results with IG, we recognize that the number of calls a and SMSs s do not have significant worth in comparison with remainders covering process, connection and intent groups. This can be explained as follows: modern malware

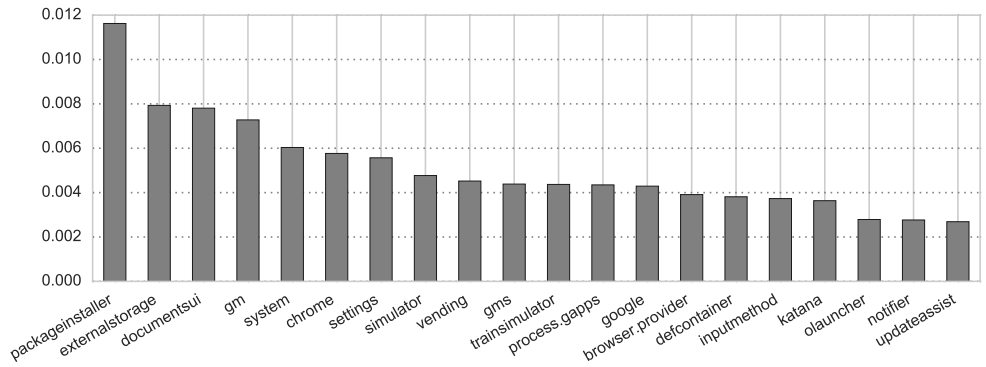

Figure 5: Top processes according to entropy ranking

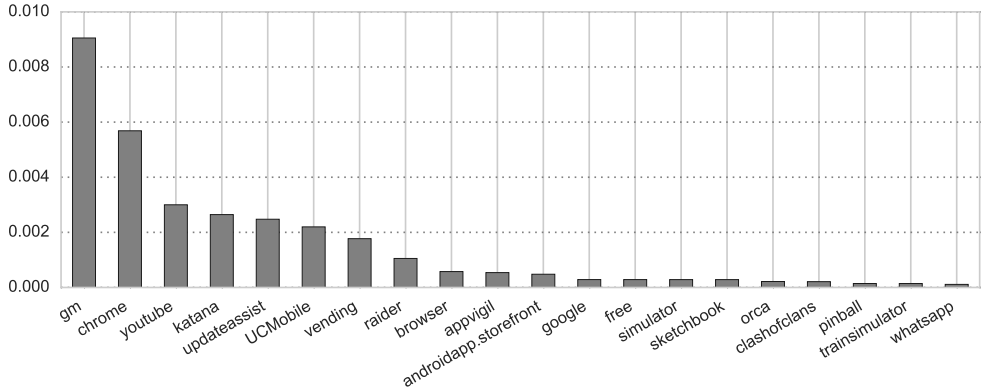

Figure 6: Top connections according to entropy ranking

intrusions usually do not create a lot of calls nor SMS(s) immediately. For this reason, the data does not belong to selected features.

However, after feature-related techniques and data cleaning operations, our obtained data is still quite high dimensional with total 76 features as shown by step 4 in Table 1. The bright point here is that the number of negative records after data cleaning and removing duplications are significantly reduced. But the transformed data for ML module is still imbalanced.

5.2. Model selection for suspicious activity detection

In our second group test, several well-known measurements are applied to evaluate quality of used models consisting of LR, SVM and NN in suspicious activity detection around malware intrusion problem. As we known, the goal of this phase is to select a model with the best prediction performance. Otherwise, the model has to behave in the most stable way to an independent dataset such as real data coming in the detection stage.

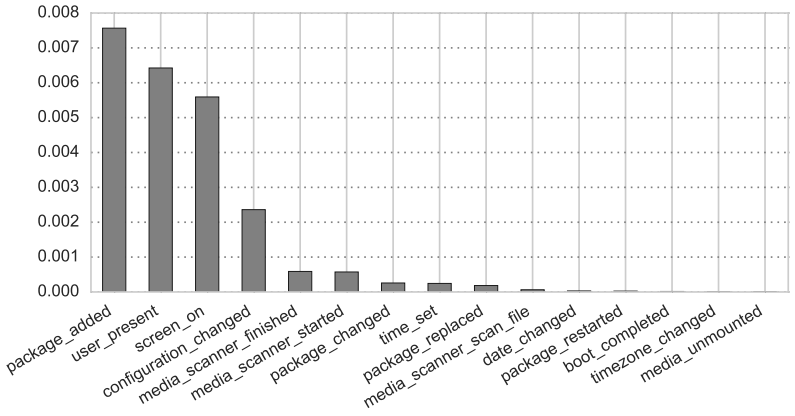

Figure 7: Top intents according to entropy ranking

5.2.1. Performance measurements for machine learning models

Popular ML metrics are measured including: accuracy (ACC), precision (Prec), recall (Rec), F1, Matthews correlation coefficient (MCC), RMSE. Those measurements are calculated as follows:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{21}$$

$$Prec = \frac{TP}{TP + FP} \tag{22}$$

$$Recall = \frac{TP}{TP + FN} \tag{23}$$

$$F1 = 2 \cdot \frac{Prec \cdot Recall}{Prec + Recall} \tag{24}$$

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FN)(FP + TN)(TP + FP)(FN + TN)}} \tag{25}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(p_i - y_i)^2} \tag{26}$$

$$\tag{27}$$

where TP is true positives, FP is false positives, TN is true negatives, TN is false negatives, p represents predicted values, y represents label and n is the number of data records.

While there is no perfect measurement for the model evaluation, MCC is generally considered as one of the best when data is imbalanced. In our testbed, the other measures (especially ACC) are mostly not useful when the differences among two classes are large. However, those others such as F1, RMSE still can provide a complementary view into model performances. Based on the obtained results, we can better choice the most suitable one.

5.2.2. Best model selection

In general, ML model is built relied to a training dataset and it needs to be calibrated through many parameters. The most popular technique to calculate those parameters is to minimize models error with Gradient Descent, which estimates model weight with many iterations. In other word, the operation sets learning rate (well-known also as λ) to an appropriate value. This parameter determines how fast or slow ML model moves towards optimal weight. If the learning rate is very large, the model may skip the optimal solution. If it is too small the model converge to the best values slowly. So using a good learning rate

is extremely important. Although there are many settings (e.g. regularization values and types, adaptive learning rate and/or early termination) and other measurements such as MAE, ROC_AUC, and so on are calculated, but to clearly present, only ACC, Prec, recall, F1, MCC and RMSE are described in our result graphs.

Figure 8 and Figure 9 depict performance measurements of ACC, Precision, Recall, F1, MCC (on the left side) and RMSE (on the right side) with models built based on RL and SVM against various learning rate values. An observation can be made here: ACC and Precision always have stable values. The reason is stated in Section 5.2.1: such measurements are not very suitable for our imbalanced data.

In contrast, measurement values of MCC, F1, Recall vary in the range of *(0.90,1.00)* and RMSE values are below *0.04* when learning rates are greater than *0.2*. This value expresses that our system achieves good data quality with *Log preprocessor* and *Feature processing* components.

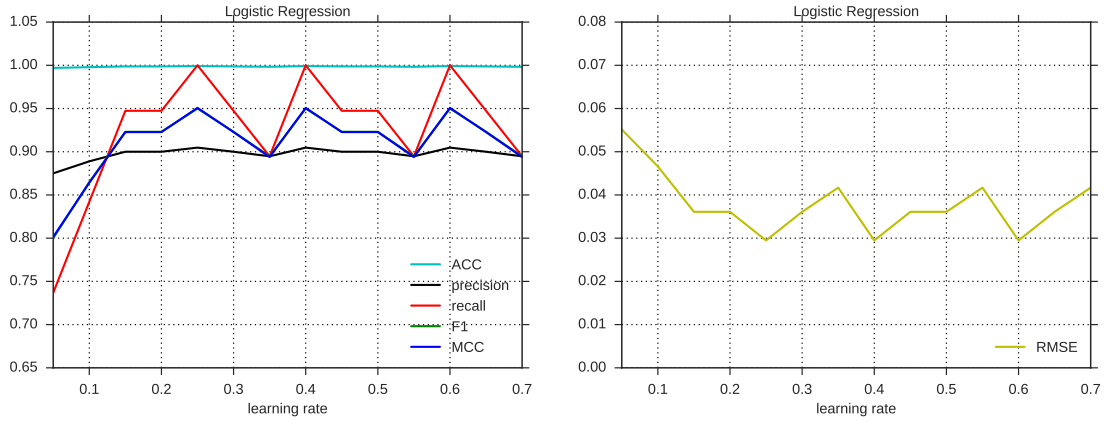

Figure 8: Logistic Regression: model performance against learning rate

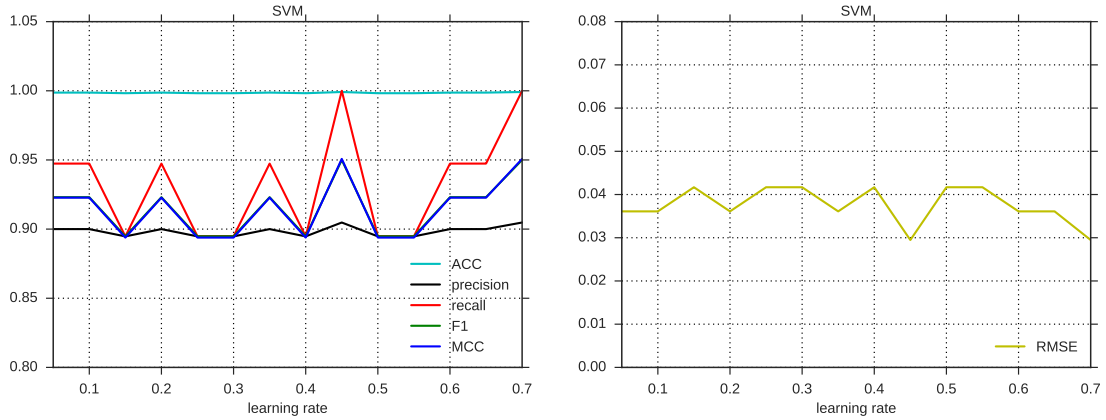

Figure 9: SVM: model performance against learning rate

Imbalanced data classes for ML may generate poor model performance or otherwise, over-fitting effect. To prevent over-fitting, L2 regularization is set as written in Section 4.3.1. To enhance model performance, over-sampling technique introduced in Chawla et al. (2002) and Zheng et al. (2015) is used in our design. The technique looks at positive records with I weight, which is also understood as multiply copies of the minority class. Thus, variable I indicates the relative importance of positive records over negative. By this

way, the proportion $P_{positive}$ of data classes for ML is denoted as follow:

$$P_{positive} = \frac{I.n_{positive}}{n_{negative} + I.n_{positive}} \tag{28}$$

Impact of importance weight I to classification effectiveness also is experimented in our work. The gained outcomes are illustrated by Figure 10 and Figure 11. The bigger value of weight I, the more balanced is. Because the balance is artificial, ML model more easily tends to over-fitting and the model could less generalized as compared with using independent datasets. Increasing I values from 1 to 5, almost all the measurements have a slightly improvement (LR) or stable values (SVM). But when I is greater than 5, the result is less stable and the accuracy of both LR and SVM decline quickly. Those outcomes prove that our models do not need to be too much artificially turned up by importance weight to get acceptable performance in a good level.

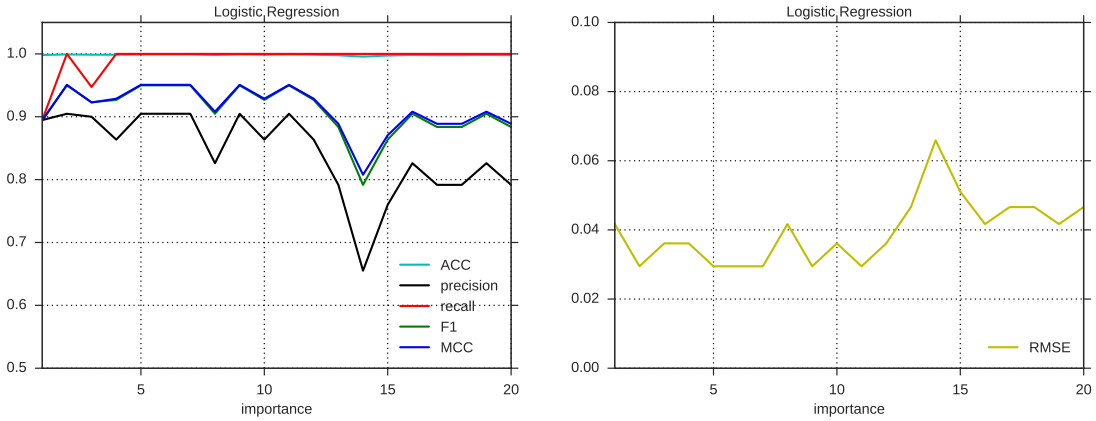

Figure 10: LR model performance against importance weight for imbalanced dataset

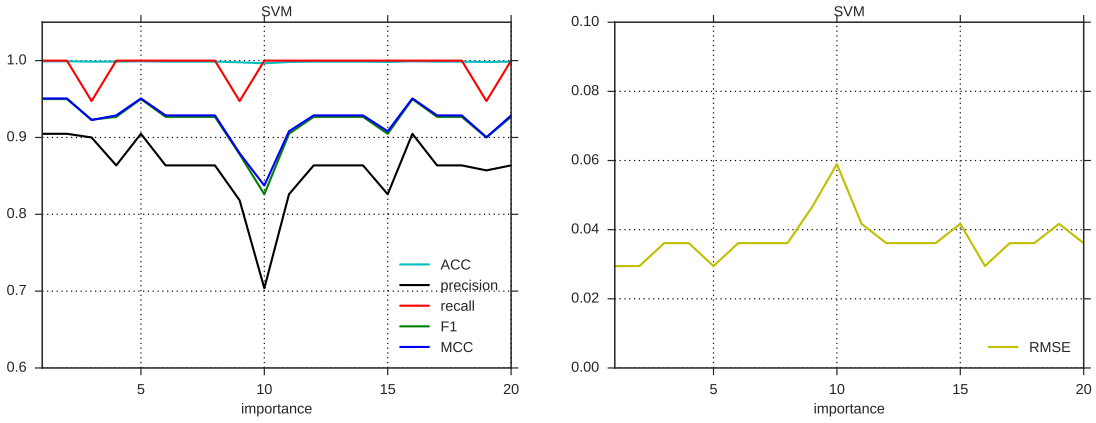

Figure 11: SVM model performance against importance weight for imbalanced dataset

NN has other specific parameters in comparison with SVM and LR as described in Section 4.3.3. Figure 12 illustrates performance NN model against number of neurons in the hidden layer. It is noticeable that the more number of hidden neurons makes all measurements are more variable. Thus almost all of them are stable in the cases of hidden neuron number is smaller than 5 or greater than 25. Similar to LR and SVM, ACC values for NN are almost not changed. This confirms that ACC measurement could not be used to evaluate classification effectiveness with highly imbalanced dataset.

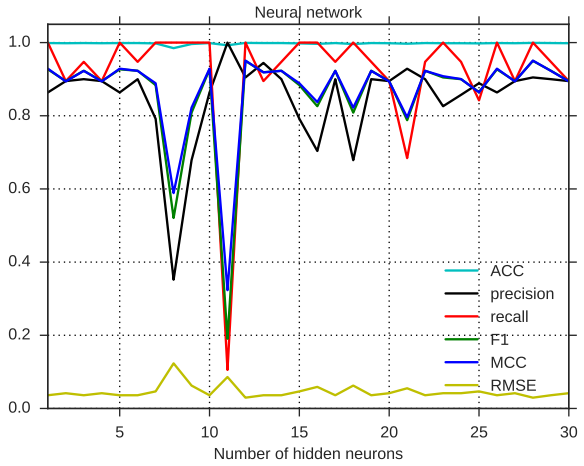

Figure 12: NN: model performance against number of neurons in hidden layer

Performance evaluations of the best models from our three selected ML methods are shown by Table 2 with very similar overall results. That means all of three methods are suitable for our binary classification problem solution: detection of suspicious activities surrounding malware intrusions.

	ACC	Precision	Recall	F1	MCC	RMSE
Logistic regresion	0.998	0.900	0.947	0.923	0.922	0.036
Support vector machine	0.998	0.900	0.947	0.923	0.922	0.036
Artificial neural network	0.998	0.863	1.000	0.926	0.928	0.036

Table 2: Performance evalutions of the best models

However, it can be seen that the stability of NN models varies quite significantly in relation with the number of neurons in hidden layers. Therefore, NN approach is eliminated. To continue to select the best model, we employ k-fold test with LR and SVM model. The goal of this test is to limit problems like over-fitting, and give an insight on model's generalized behaviors over an independent dataset. The k-fold (with k=5) is carried out repeatedly with average results shown in Table 3, which also depicts relations of imbalanced importance weight (over sampling of positive observations). These outcomes also confirm that our results expressed in the Figure 10 and Figure 11 are precise. At present, we can conduct that SVM appears as the most suitable ML method in our domain.

	weight	ACC	Precision	Recall	F1	MCC	RMSE
Logistic regresion	5	0.996	0.812	0.931	0.864	0.866	0.055
	3	0.996	0.826	0.906	0.862	0.862	0.055
Support vector machine	5	0.997	0.884	0.919	0.897	0.898	0.054
	3	0.998	0.900	0.976	0.936	0.936	0.037

Table 3: Model stability with k-fold (k=5) and average performance evaluations

6. Conclusion and Perspective

Research interest presented in this work is to detect suspicious behaviors based on collected logs from mobile devices. The characteristics of this problem include: (1) collected raw logs are extremely noisy for specific detection purpose; (2) the data contains over time duplicated and continuous information about

processes and connections, and (3) suspicious behaviors have low occurrence ratio and logs contain uncertain feature because applications can be installed on mobile devices according to users demands without any restriction. In addition, (4) mobile devices product a huge amount of log data continuously and dynamically.

Dealing with the difficulties described above, our work focuses on improving cyber-resilience by analyzing applications/services activity logs that are represented user behaviors on mobile devices. The obtained results bring the following contributions:

1. Proposing a novel approach to detecting malware run on mobiles using logs generated from applications/services operations, which occur under user authorizations.
2. Heuristic analysis technique is taken full advantage to find out modern malware type that may be unknown before.
3. Due to data complexity, physical and logical taxonomy are proposed to map the collected raw data into a light-weight semantic formalization based on specific domain knowledge. Furthermore several selected NLP techniques are used in combination with the taxonomy mapping to remove noises and inaccuracy inside collected log data. The aim is to find feasible patterns that do not conform to the expected normal behaviors. Although these methods are quite popular but applying appropriately to specific gathered data in this domain is the critical point to help resolve the similar problems.
4. Another our good effort presented in the work is to design a novel lemmatization module for the guide-guard system. This ingredient is changed according to characteristics of Android environment and also to serve data preprocessing task.
5. Experiments and evaluations in choosing a suitable ML methods (i.e. SVM, LR, NN) in such a way that the chosen method should bring effectiveness as desired with the collected data. This achievement can help developers save testing time in the case they face with the same security problem in the future.
6. Proposing a scalable solution for the malware detection problem is the last our contribution. The scalability is manifested via two points: the system is built based on high-performance operation fashion using up-to-date open-source technologies like Apache Spark, fast out-of-core parallel learning system VW deployed on a Hadoop cluster with MPI support. Otherwise, incremental ML techniques are employed to our dynamic proposed data processing model. The scalability is key to the question: can the system extend and operate in practice with the huge number of devices?

The experimental outcomes described in this document show that our approach provides high satisfaction for detection quality as desired. The solution thus can help reduce risky behaviors occurring on user mobiles and increase automatically cyber-risk cognition.

The main drawback of this work is that we assume detection and possible consequent alarm are done after a certain time interval. In our tests, this value is set to thirty minutes and it is expressed as longer than the optimal (i. e. from five to ten minutes of activities surrounding the intrusion). This is our main task, which has to resolve in the near future. On the other hand, the detection presented in the document belongs to the anomaly discovery class exploiting ML techniques. Consequently, the extension is expected to be more exact on suspicious behaviors classification in the manner of multiple data classes.

7. Acknowledgements

This work is supported by the projects Slovak VEGA 2/0167/16 “Methods and algorithms for the semantic processing of Big Data in distributed computing environment”, Vietnamese MOETs No. B2017-BKA-32 “Research on developing software framework to integrate IoT gateways for fog computing deployed on multi-cloud environment”, and DEEP-HybridDataCloud EU H2020-777435 “Designing and Enabling E-infrastructures for intensive Processing in a Hybrid DataCloud”.

Our special thanks goes to IBM Slovakia for supports and to the IISAS colleague RNDr. Viera Šipková for the language editing.

8. Appendix

8.1. Call log structure

Timestamp TIMESTAMP , Time TIMESTAMP , PhoneNum VARCHAR2 (30) , Type VARCHAR2 (100) , Duration NUMBER (5) , LongLat VARCHAR2 (100) , Longitude NUMBER (16,14) , Latitude NUMBER (16,14) , AccXYZ VARCHAR2 (100) , AccX NUMBER (12,6) , AccY NUMBER (12,6) , AccZ NUMBER (12,6) , RowID NUMBER (20) , IMEI VARCHAR2 (16)

8.2. SMS log structure

Timestamp TIMESTAMP , Time TIMESTAMP , PhoneNum VARCHAR2 (30) , Type VARCHAR2 (100) , LongLat VARCHAR2 (100) , Longitude NUMBER (16,14) , Latitude NUMBER (16,14) , AccXYZ VARCHAR2 (100) , AccX NUMBER (12,6) , AccY NUMBER (12,6) , AccZ NUMBER (12,6) , RowID NUMBER (20) , IMEI VARCHAR2 (16)

8.3. Browser log structure

Timestamp TIMESTAMP , Time TIMESTAMP , URL VARCHAR2 (4000) , Title VARCHAR2 (4000) , IP VARCHAR2 (4000) , Port NUMBER (5) , Proto VARCHAR2 (100) , LongLat VARCHAR2 (100) , Longitude NUMBER (16,14) , Latitude NUMBER (16,14) , AccXYZ VARCHAR2 (100) , AccX NUMBER (12,6) , AccY NUMBER (12,6) , AccZ NUMBER (12,6) , RowID NUMBER (20) , IMEI VARCHAR2 (16)

8.4. Process log structure

Timestamp TIMES TAMP , Processname VARCHAR2 (100) , CPUUsage VARCHAR2 (20) , LRU NUMBER (7) , ImportanceReasonPID NUMBER (7) , Importance NUMBER (7) , ImportanceReasonCode NUMBER (7) , dalvikPrivateDirty VARCHAR2 (20) , dalvikSharedDirty VARCHAR2 (20) , dalvikPss VARCHAR2 (20) , nativePrivateDirty VARCHAR2 (20) , nativeSharedDirty VARCHAR2 (20) , nativePss VARCHAR2 (20) , otherPrivateDirty VARCHAR2 (20) , otherSharedDirty VARCHAR2 (20) , otherPss VARCHAR2 (20) , TotalPrivateDirty VARCHAR2 (20) , TotaISharedDirty VARCHAR2 (20) , LongLat VARCHAR2 (100) , Longitude NUMBER (16,14) , Latitude NUMBER (16,14) , AccXYZ VARCHAR2 (100) , AccX NUMBER (12,6) , AccY NUMBER (12,6) , AccZ NUMBER (12,6) , RowID NUMBER (20) , IMEI VARCHAR2 (16) , UserID VARCHAR2 (6) , PID NUMBER (15)

8.5. Connection log structure

Timestamp TIMESTAMP , Application VARCHAR2 (100) , ToADDR VARCHAR2 (100) , ToPort NUMBER (5) , FromADDR VARCHAR2 (100) , FromPort NUMBER (5) , State VARCHAR2 (2) , LongLat VARCHAR2 (100) , Longitude NUMBER (16,14) , Latitude NUMBER (16,14) , AccXYZ VARCHAR2 (100) , AccX NUMBER (12,6) , AccY NUMBER (12,6) , AccZ NUMBER (12,6) , RowID NUMBER (20) , IMEI VARCHAR2 (16) , UserID VARCHAR2 (6)

8.6. Intent log structure

Timestamp TIMESTAMP , Act VARCHAR2 (500) , Flg VARCHAR2 (100) , ExtraSize VARCHAR2 (100) , ExtraData VARCHAR2 (4000) , LongLat VARCHAR2 (100) , Longitude NUMBER (16,14) , Latitude NUMBER (16,14) , AccXYZ VARCHAR2 (100) , AccX NUMBER (12,6) , AccY NUMBER (12,6) , AccZ NUMBER (12,6) , RowID NUMBER (20) , IMEI VARCHAR2 (16) , UserID VARCHAR2 (6)

References

Agarwal, A., Chapelle, O., Dudík, M., Langford, J., 2014. A reliable effective terascale linear learning system. Journal of Machine Learning Research 15, 1111–1133.

AndroidDevelopers, 2017. Android developers. URL: https://developer.android.com/index.html.

Beygelzimer, A., Kale, S., Luo, H., 2015. Optimal and adaptive algorithms for online boosting., in: ICML, pp. 2323–2331.

Bose, A., Hu, X., Shin, K.G., Park, T., 2008. Behavioral detection of malware on mobile handsets, in: Proceedings of the 6th international conference on Mobile systems, applications, and services, ACM. pp. 225–238.

Buczak, A.L., Guven, E., 2016. A survey of data mining and machine learning methods for cyber security intrusion detection. IEEE Communications Surveys & Tutorials 18, 1153–1176.

Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P., 2002. Smote: synthetic minority over-sampling technique. Journal of artificial intelligence research 16, 321–357.

Dua, S., Du, X., 2016. Data mining and machine learning in cybersecurity. CRC press.

ENISA, 2017. ENISA Threat Landscape Report 2016. Technical Report. The European Union Agency for Network and Information Security (ENISA).

Gama, J.a., Žliobaitė, I., Bifet, A., Pechenizkiy, M., Bouchachia, A., 2014. A survey on concept drift adaptation. ACM Comput. Surv. 46, 44:1–44:37. URL: http://doi.acm.org/10.1145/2523813, doi:10.1145/2523813.

Hardeniya, N., 2015. NLTK essentials. Packt Publishing Ltd.

Hluchý, L., Nguyen, G., Astaloš, J., Tran, V., Šipková, V., Nguyen, B.M., 2016. Effective computation resilience in high performance and distributed environments. Computing and Informatics 35.

Langford, J., et al., 2017. Vowpal wabbit. URL: https://github.com/JohnLangford/vowpal_wabbit/wiki.

Leskovec, J., Rajaraman, A., Ullman, J.D., 2014. Mining of massive datasets. Cambridge University Press.

Mukkamala, S., Sung, A., Abraham, A., 2005. Cyber security challenges: Designing efficient intrusion detection systems and antivirus tools. Vemuri, V. Rao, Enhancing Computer Security with Smart Technology.(Auerbach, 2006) , 125–163.

Nguyen, G., Astaloš, J., Hluchý, L., 2016. Considerations about data processing, machine learning, hpc, apache spark and gpu, in: 11th Workshop on Intelligent and Knowledge Oriented Technologies in conjunction with 35th conference Data and Knowledge, pp. 241–247.

Papliatseyeu, A., Mayora, O., 2009. Mobile habits: Inferring and predicting user activities with a location-aware smartphone, in: 3rd Symposium of Ubiquitous Computing and Ambient Intelligence 2008, Springer. pp. 343–352.

Parsons, K., McCormac, A., Butavicius, M., Ferguson, L., 2010. Human factors and information security: individual, culture and security environment. Technical Report.

Rapid7, 2017. Rapid7: Accelerate security, vuln management, compliance. URL: https://www.rapid7.com/.

Read, J., Bifet, A., Pfahringer, B., Holmes, G., 2012. Batch-incremental versus instance-incremental learning in dynamic and evolving data, in: International Symposium on Intelligent Data Analysis, Springer. pp. 313–323.

Ricci, F., Rokach, L., Shapira, B., Kantor, P.B., 2010. Recommender Systems Handbook. 1st ed., Springer-Verlag New York, Inc., New York, NY, USA.

Schutt, R., O'Neil, C., 2013. Doing data science: Straight talk from the frontline. " O'Reilly Media, Inc.".

Sebastiani, F., 2002. Machine learning in automated text categorization. ACM Comput. Surv. 34, 1–47. URL: http://doi.acm.org/10.1145/505282.505283, doi:10.1145/505282.505283.

Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C., Weiss, Y., 2012. "andromaly": a behavioral malware detection framework for android devices. Journal of Intelligent Information Systems 38, 161–190.

Shamili, A.S., Bauckhage, C., Alpcan, T., 2010. Malware detection on mobile devices using distributed machine learning, in: Pattern Recognition (ICPR), 2010 20th International Conference on, IEEE. pp. 4348–4351.

Tam, K., Feizollah, A., Anuar, N.B., Salleh, R., Cavallaro, L., 2017. The evolution of android malware and android analysis techniques. ACM Computing Surveys (CSUR) 49, 76.

ThreadTrackSecurity, 2015. Best Practices for Dealing with Phishing and Next-Generation Malware. An Osterman Research White Paper. Osterman Researc.

Tong, S., Koller, D., 2001. Support vector machine active learning with applications to text classification. Journal of machine learning research 2, 45–66.

Verkasalo, H., Salmeron, B.J., 2009. Analysis of the contextual behaviour of mobile subscribers, in: International Conference on Communications Infrastructure. Systems and Applications in Europe, Springer. pp. 252–266.

Vokorokos, L., Dankovičová, Z., Leščišin, L., 2017. Using of the forensic analyzing tools, code obfuscation, in: Applied Machine Intelligence and Informatics (SAMI), 2017 IEEE 15th International Symposium on, IEEE. pp. 000033–000036.

Woerndl, W., Manhardt, A., Prinz, V., 2010. A framework for mobile user activity logging. Mining Ubiquitous and Social Environments (MUSE 2010) , 39.

Zainuddin, N.B., Abdollah, M.F.B., Yusof, R.B., Sahib, S.B., 2014. A study on abnormal behaviour in mobile application. Open Access Library Journal 1, 1.

Zheng, Z., Yunpeng, C., Ye, L., 2015. Oversampling method for imbalanced classification. Computing and Informatics 34.