2020 12th International Conference on Communication Software and Networks

MineRBS: Detecting Android Malware Based on Runtime Behavior Sequence

Hao Jin, Yangyang Li
National Engineering Laboratory for
Public Safety Risk Perception
and Control by Big Data (PSRPC), CAEIT
Beijing, China
e-mail: jh_cetc@163.com, liyangyang@cetc.com.cn

Ying Yang
The Third Research Institute of
Ministry of Public Security
Shanghai, China
e-mail: yangying@mcst.org.cn

***Abstract*—The runtime behaviors performed by Android applications reflect their potential characteristics. While the implementation of a malicious attack usually requires the cooperation of multiple runtime behaviors, so mining the association between runtime behavior sequences can effectively detect unknown malicious applications. Most researchers concerned the statistical properties of a single behavior, and there was little work studying the statistical properties of the association between runtime behaviors. In this paper, we present an Android malware detection system MineRBS based on a novel sequential pattern mining method, called RB AprefixSpan (PrefixSpan Abbreviated Project Mining of Runtime Behaviors), to dig out runtime behavior associations. RB AprefixSpan algorithm could discover runtime behavior sequential patterns from known malware families and build the behavior sequential pattern database to detect mal-ware. What's more, RB AprefixSpan algorithm uses abbreviated projection database instead of projection database in PrefixSpan to improve the spatial performance. Through experiments, we verity the correctness and effectiveness of our system.**

Keywords-sequential pattern mining; Android malware detection; runtime behavior sequences

I. INTRODUCTION

According to a recent report from Gartner [1], Android has been growing market share in the smartphone operating system market, which in the first quarter of 2017 is at 86.1%. Android users can download applications not only from Google's official market Google Play, but also from other third-party markets. Google takes security measures to check whether the new submitted app is benign to reduce malware, while third-party markets usually do not have sufficient malware scanning, which lead to the wide spread of malware. It is reported that, the number of Android malware has increased from hundreds of thousands to more than ten million since 2012, which indicates that Android smartphones are facing an explosive growth of malware threats [2].

This work was supported in part by Strategic Consulting Research Project of Ningxia Research Institute, Chinese Academy of Engineering (No. 2019NXZD7) and Major Special Science and Technology Project of Hainan Province (No. ZDKJ2019008).

Generally, there are two types of malware detection approaches. One is to perform static malware detection based on features like required permissions, sensitive API calls, etc. The systems leverage static information to generate signatures for detection [3, 4]. However, there is evidence that with growing popularity of static malware detection, malware increasingly conducts obfuscation and transformation attacks to break static methods. Researchers have proposed several approaches to solve the problem [5-7]. The other one is the dynamic analysis system, which has been proposed in several researches [8-10]. These systems identify malware through monitoring the behaviors of an application at runtime but suffer from great challenge that how to trigger sufficient suspicious behaviors.

According to a study [11], malicious applications with similar behaviors should be classified as being in the same malware family, such as Fake, DroidKungFu, ADRD, etc. Furthermore, survey [12] reveals that over 98% of novel malware are actually variants from existing malware families. These variants adopt sophisticated techniques to evade existing static analysis-based detection systems, while the malicious behaviors of core functionalities like privilege escalation, remain unchanged [13]. With the observation, runtime behaviors have been used extensively to detect [13] and classify [14] malware with high accuracy. As of yet, most researchers simply take the single runtime behavior into consideration, and there is little work studying the statistical properties of the associations between runtime behaviors.

Data mining-based approaches can automatically infer detection patterns from features extracted by program or dynamic analysis from malware, and have promising approaches for detecting malware. Some researchers perform static or dynamic analysis to extract features from PC applications and adopt data mining algorithms to achieve malware detection. Yang et al. focus on the permissions requested by Android applications, and propose an Android malware detection system based on frequent pattern mining algorithm [15].

In this paper, we find that the associations between runtime behavior sequences can greatly reflect the potential charac-

978-1-7281-9815-6/20/$31.00 ©2020 IEEE

teristic of the application. With the observation, we build an Android malware detection system, named MineRBS, based on sequence pattern mining of runtime behavior sequences of applications. We firstly leverage the CopperDroid dynamic analysis tool to extract runtime behaviors of known Android malware families [10], and construct the runtime behavior sequence database. Then, we propose a novel sequential pattern mining method $RB\_AprefixSpan$(Abbreviated Project Mining of Runtime Behaviors based on PrefixSpan) to discover runtime behaviors associations in each malware family. $RB\_AprefixSpan$ algorithm uses abbreviated projection database instead of projection database in PrefixSpan. After that, we can construct behavior sequential pattern database for each malware family. Lastly, we extract runtime behavior sequences of the application to be tested, and match the behavior sequential pattern databases to identify whether it is malicious. Experiments on a large number of real-world apps reveal the effectiveness of our approach on detecting malware variants and novel malware.

The reminder of the paper is organized as follows. We introduce the necessary background in Section II. After that in Section III, we present our Android malware detection system MineRBS based on $RB\_AprefixSpan$ algorithm. In Section IV, we report the evaluation of our experimental results. We discuss the conclusion in Section V.

II. Background

In this section, we introduce the essential background knowledge of the technique of runtime behavior analysis and sequence pattern mining algorithm.

A. Runtime Behavior Analysis

Researchers conclude in [14] that, the dynamic runtime behavior analysis should operate at the level of system calls where high-level application semantics are obscured, to solve the problem of obfuscation and native code. Besides, [8] points out that pure system calls cannot characterize the runtime behaviors of an Android application, as they fail to reconstruct inter-process and inter-component communications, which are essential to understanding runtime behaviors of the application.

We therefore leverage CopperDroid [10], an approach which can reconstruct the integrated runtime behaviors of Android applications. According to a single point of observation (i.e., system calls), CopperDroid can reconstruct runtime behaviors of Android applications at multiple levels (i.e., pure system calls, decoded binder communication, and abstracted behavioral patterns). What's more, the dynamic analysis of CopperDroid is agnostic to the runtime system, thus it can be applied to all Android operating systems while making no modifications to the system.

B. Sequence Pattern Mining Algorithm

Sequence pattern mining is a data mining method for temporal data, which aims at discovery of frequent episodes as patterns in a sequence database [16]. The main idea is to discover a frequently occurs sequence of events. Sequence pattern mining approaches are suggested for numerous applications, such as the analyses of customer purchase behaviors, DNA sequences, web access patterns, and other data-mining tasks.

Most of the sequence pattern mining algorithms proposed previously are based on the Apriori heuristic first illustrated in [17]. However, though Apriori-like sequential pattern mining approach reduces the search space, it bears the inherent costs of the enormous set of candidate sequences and repetitive scans of the databases. Pei et al. proposes a novel sequential pattern mining method PrefixSpan in [18]. The main idea of PrefixSpan is that, it only examines the prefixes, and only projects their postfixes into projected databases. In each projected database, it generates sequential patterns only by exploring the frequent patterns. PrefixSpan has the advantage that it runs considerably faster than both Apriori-based GSP algorithm [19] and FreeSpan [20].

III. System Design

In this section, we firstly introduce the overview of MineRBS. Next, we explain each module respectively to describe how it works for malware detection by constructing behavior sequential pattern database for multiple malware families.

A. System Overview

Fig. 1 gives an overview of MineRBS, and its dynamic analysis and data mining components. Malware families for constructing the feature base are passed to CopperDroid for runtime behavior extraction, including system calls and high-level runtime behaviors (see III-B). After constructing the runtime behavior sequence database (see III-C), MineRBS designs an improved PrefixSpan algorithm $RB\_AprefixSpan$ (see III-D) to discover frequent runtime behavior subsequences as patterns, and constructs behavior sequential pattern database (see III-E) for malware detection (see III-F).

B. Dynamic Analysis

As discussed above, the runtime behaviors of an Android application are fully described by system calls and binder communications. For a single application, MineRBS uses CopperDroid as its runtime behavior extractor, which runs the application in an unmodified Android image on the top of CopperDroid emulator. CopperDroid can perform out-of-the-box behavior analysis on arbitrary Android applications automatically, and characterize low-level OS-specific and high-level Android-specific behaviors.

Tam et al. examined the results of CopperDroid's analyses on a number of Android malware, and classified the high-level behaviors to six macro classes (see left column in Table I). Each class consists of one or more behavioral models, which can be expressed by a set of actions (see right column in Table I). The complexity of these actions varies greatly. Among them, some actions are defined as a single system call (i.e., execve), some are defined as a set of binder transactions

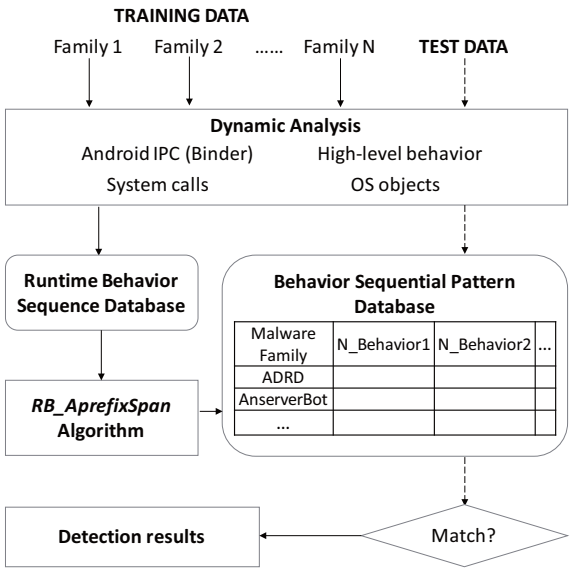

Figure 1. Architecture of MineRBS.

(such as those under the class "SMS Send" and "Personal Info. Access"), others are defined as a series of system calls (such as those under the class "Network Access"). For constructing the runtime behavior sequence database, MineRBS uses the classes of high-level behaviors generated by CopperDroid.

TABLE I. Behavioral Classes Extracted by CopperDroid

Behavioral Class	Subclass
S_1: File Access	Open, Read, Write
S_2: Exec External Application	Shell, Generic, Privilege escalation, Install APK
S_3: Network Access	DNS, HTTP, Other
S_4: Personal Info. Access	SMS, Contacts, Phone Info., Location
S_5: Send SMS	
S_6: Make/Alter Call	

C. Runtime Behavior Sequence Database

The workflow of constructing runtime behavior sequence database is shown in Fig. 2. To construct the runtime behavior sequence database for each Android malware family, we firstly choose runtime behaviors in CopperDroid behavioral classes, and number each behavior to construct a suspicious runtime behavior database, which consists of all runtime behaviors that frequently occur in Android malware. Next, we number the runtime behaviors extracted by CopperDroid of malware samples in each malware family. As input for the $RB\_AprefixSpan$ algorithm, the order number of suspicious runtime behaviors of Android applications must be expressed by runtime behavior sequences. Let R denote all suspicious runtime behaviors and N denote the order numbers of all these suspicious runtime behaviors. Given an application a, let $N\_Behavior(a)$ represent its behavior sequence. Then, for each suspicious runtime behavior $r_i \in R$, the order number of which $n_i \in N$, we add n_i to $N\_Behavior(a)$ if a produce the runtime behavior r_i. Thus, the runtime behavior sequence of a can be expressed as $N\_Behavior(a) =< n_1 n_2 \cdots n_i \cdots >$. Finally, each runtime behavior sequence is stored to runtime behavior sequence database for further mining to characterize the behavior feature of each malware family.

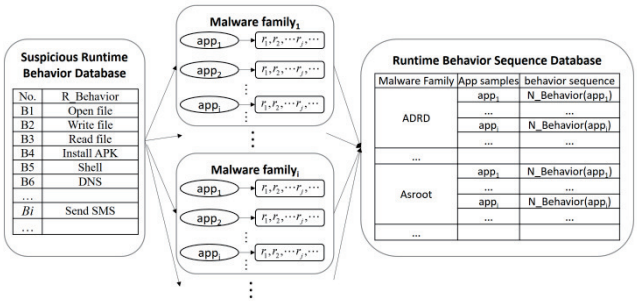

Figure 2. Workflow of constructing the runtime behavior se-quence database.

D. $RB\_AprefixSpan$ Algorithm

The $RB\_AprefixSpan$ algorithm takes the runtime behavior sequences of each app in each malware family as input, and analyzes the associations between these sequences to discover the frequently occurred sequential patterns in each malware family. In this section, we first briefly introduce several definitions in sequence pattern mining. Then, we recall the major idea of PrefixSpan. Finally, we present the improved algorithm $RB\_AprefixSpan$ that takes advantages of both the speed of PrefixSpan and the low space overhead of GSP.

a) Definitions: Let $I = \{i_1, i_2, \cdots, i_n\}$ represent a set of behavior items. A behavior itemset is a subset of I. A behavior sequece is an ordered list of behavior itemsets. A behavior sequence s is donated by $< s_1 s_2 \cdots s_i >$, where s_j is a behavior itemset, i.e., $s_j \subseteq I$ for $1 \leq j \leq l$. s_j is also called a behavior element of the behavior sequence, and denoted as $(x_1 x_2 \cdots x_m)$, where x_k is a behavior item, i.e., $x_k \in I$ for $1 \leq k \leq m$. A behavior item can occur at most once in an element of a behavior sequence, but can occur multiple times in different elements of a behavior sequence. The number of instances of items in a behavior sequence is called the length of the sequence. A behavior sequence with length l is called a l-sequence. A behavior sequence $\alpha =< a_1 a_2 \cdots a_m >$ is called a subsequence of another behavior sequence $\beta =< b_1 b_2 \cdots b_n >$ and β is a super sequence of α, donated as $\alpha \subseteq \beta$, if there exist intergers $1 \leq j_1 < j_2 < \cdots < j_m \leq n$ such that $a_1 \subseteq b_{j_1}, a_2 \subseteq b_{j_2}, \cdots, a_m \subseteq b_{j_m}$. The support of a behavior sequence α in a behavior sequence database S is the number of tuples $< sid, s >$ containing α, i.e., $support_s(\alpha) =\mid \{< sid, s >\mid (< sid, s >\in S) \wedge (\alpha \subseteq s)\} \mid$. Given a positive integer ϵ as the support threshold, a behavior sequence α

is called a frequent behavior sequential pattern in behavior sequence database S if the behavior sequence is contained by at least ϵ tuples in the database, i.e., $support_s(\alpha) \geq \epsilon$. A behavior sequential pattern with length l is called a l-pattern.

b) PrefixSpan: Researchers introduced PrefixSpan in [18]. We can see the major idea of PrefixSpan algorithm is that, the projecting operation is based purely on frequent prefixes, instead of frequent subsequences, as the latter can always be found by growing the former. The algorithm has the efficiency that it does not need to generate candidate sequences and the projected databases keep shrinking. Thus, compared to GSP, PrefixSpan searches a much smaller space, and takes a much better shrinking factor comparing with FreeSpan. The detailed algorithm of PrefixSpan is illustrated in [18]. Due to limited space, no more tautology here.

c) $RB\_AprefixSpan$: The projected database constructed by PrefixSpan contains several infrequent items, which need to be scanned in every step. For the problem, we consider GSP [19], an Apriori-like method which states the fact that any super-pattern of a nonfrequent pattern should not be frequent. By combining PrefixSpan and GSP, we propose an improved algorithm $RB\_AprefixSpan$. The workflow of $RB\_AprefixSpan$ is mainly based on PrefixSpan, while adding the pruning phase of GSM before the projection phase to construt a abbreviated projection database to replace the original one.

- Abbreviated projection database

Let the behavior sequence database S given in Table II be an example database and we set min_support to be 4. We can see that the behavior items set of S is {B1, B2, B3, B4, B5, B6}.

TABLE II. A Behavior Sequence Database

Sequence id	Behavior Sequence
1	$< B1(B1,B2)(B1,B3)B4(B3,B6) >$
2	$< (B1,B4)B3(B2,B3) >$
3	$< (B5,B6)(B1,B2)(B4,B5)B3 >$
4	$< (B1,B6)B3,B2,B3 >$

$RB\_AprefixSpan$ firstly scans S and collects the support for each behavior item. The items are listed in support descending order as: $b\_item =< B1 >: 4, < B2 >: 4, < B3 >: 4, < B4 >: 3, < B6 >: 3, < B5 >: 1$. As $min\_support = 4$, the length-1 sequential patterns include $< B1 >, < B2 >$ and $< B3 >$. According to these three prefixes, we can partition the set of sequential patterns into three subsets respectively. In this way, the projected databases constructed by PrefixSpan are listed in the third column in Table III. As $< B4 >: 3, < B6 >: 3$, and $< B5 >: 1$ don't meet the criteria, i.e. $min\_support = 4$, we can infer that the sequential patterns cannot contain $< B4 >, < B5 >$ and $< B6 >$. Consequently, we can prune $< B4 >, < B5 >, < B6 >$ items in the original projected databases to construct abbreviated projected databases, shown in the fourth column in Table III.

TABLE III. Projected Databases and Abbreviated Projected Databases

Prefix	$S \mid_p$	Projected database	Abbreviated projected database
$< B1 >$	$S \mid< B1 >$	$< (B1,B2)(B1,B3)B4(B3,B6) >$ $< (\_,B4)B3(B2,B3) >$ $< (\_,B2)(B4,B5)B3 >$ $< (\_,B6)B3,B2,B3 >$	$< (B1,B2)(B1,B3)\_(B3,\_) >$ $< (\_,\_)B3(B2,B3) >$ $< (\_,B2)(\_,\_)B3 >$ $< (\_,\_)B3,B2,B3 >$
$< B2 >$	$S \mid< B2 >$	$< (B1,B3)B4(B3,B6) >$ $< (\_,B3) >$ $< (B4,B5)B3 >$ $< B3 >$	$< (B1,B3)\_(B3,\_) >$ $< (\_,B3) >$ $< (\_,\_)B3 >$ $< B3 >$
$< B3 >$	$S \mid< B3 >$	$< B4(B3,B6) >$ $< (B2,B3) >$ $< B2,B3 >$	$< (\_(B3,\_) >$ $< (B2,B3) >$ $< (B2,B3) >$

We can find that the abbreviated projected database is smaller than the original projected database. In reality, the reduction coefficient is considerable as the following reasons: 1) With the growth of prefixes, the number of behavior sequential patterns in abbreviated projected database will become much smaller. 2) Comparing with PrefixSpan, the abbreviated projected database constructed by $RB\_AprefixSpan$ contains only length-1 sequential pattern. Thus, the reduction coefficient is much more significant than PrefixSpan.

With the abbreviated projected database, we can reserve critical behavior items, and prune those unconsidered ones. As we just remove infrequent behavior items, the abbreviated projected database would not loss important information on identifying malware.

- $RB\_AprefixSpan$ algorithm

Based on the lemmas of PrefixSpan discussed in [18], we have the algorithm of $RB\_AprefixSpan$ as follows.

Algorithm 1 RB_AprefixSpan

Input: S: a behavior sequence database
$min\_support$: the minimum support threshold
Output: The complete set of behavior sequential patterns
Method: $abbreviated\_pb(S, p)$
$PrefixSpan(S, 0, <>)$
Subroutine: $PrefixSpan(S \mid_p, l, p)$
Parameters:
p: a behavior sequential pattern
l: the length of p
$S \mid_p$: the p-abbreviated projected database, if $p \neq <>$; otherwise, S
Method:
1. Scan $S \mid_p$ to find frequent behavior items b_item which satisfies one of the conditions that:
(a) b_item can be tacked to the last behavior element of p, and generate a behavior sequential pattern;
(b) $< b\_item >$ can be assembled to p, and generate a behavior sequential pattern.
2. For each frequent behavior item b_item, append it to p, and generate a behavior sequential pattern p', then output p';
3. For each p', construct p'-abbreviated projected database $S \mid_{p'}$, and call $PrefixSpan(S \mid_{p'}, l+1, p')$.

Take the behavior sequence database S shown in Table

II as example, we describe how $RB\_AprefixSpan$ finds the complete set of sequential patterns. We get the length-1 sequential patterns $< B1 >, < B2 >$ and $< B3 >$, and construct their abbreviated projected database respectively in Table III. We then scan $S \mid< B1 >$ and collects the support for each behavior item. The items are listed in support descending order as: $b\_item =< B2 >: 4, < B3 >: 4, < B1 >: 1$. As $min\_support = 4$, we can prune $< B1 >$, and get the length-2 sequential patterns $< B1, B2 >$ and $< B1, B3 >$. According to these two prefixes, we can partition $S \mid< B1 >$ into two subsets respectively, shown in Table IV.

TABLE IV. Abbreviated Projected Database $S \mid< B1, B2 >$ and $S \mid< B1, B3 >$

Prefix	$S \mid_p$	**Abbreviated projected database**
$< B2 >$	$S \mid< B1, B2 >$	$< (\_, B3)\_(B3, \_) >< (\_, B3) >$ $< (\_, \_)B3 >< B3 >$
$< B3 >$	$S \mid< B1, B3 >$	$< \_(B3, \_) >< (B2, B3) >< B2, B3 >$

Repeat the procedure for $S \mid< B2 >$, we can prune $< B1 >$ and $< B2 >$, and get the subset $S \mid< B2, B3 >$, shown in Table V.

TABLE V. Abbreviated Projected Database $S \mid< B2, B3 >$

Prefix	$S \mid_p$	**Abbreviated projected database**
$< B3 >$	$S \mid< B2, B3 >$	$< \_(B3, \_) >$

Similarly, repeat the procedure for $S \mid< B3 >$, the items are listed in support descending order as: $< B3 >: 3, < B2 >: 2, < B1 >: 0$, all of which don't satisfy the min_support. Thus, the prefix $< B3 >$ cannot grow.

Repeat the procedure for $S \mid< B1, B2 >, S \mid< B1, B3 >, S \mid< B2, B3 >$ respectively, we find that only $S \mid< B1, B2 >$ can generate a new behavior sequential pattern $< B1, B2, B3 >$. The abbreviated projected database $S \mid< B1, B2, B3 >$ is shown in Table VI.

TABLE VI. Abbreviated Projected Database $S \mid< B1, B2, B3 >$

Prefix	$S \mid_p$	**Abbreviated projected database**
$< B3 >$	$S \mid< B1, B2, B3 >$	$< \_(B3, \_) >$

In summary, we can get the length-1 sequential patterns $< B1 >, < B2 >$ and $< B3 >$; length-2 sequential patterns $< B1, B2 >, < B1, B3 >$ and $< B2, B3 >$; and length-3 sequential patterns $< B1, B2, B3 >$.

E. Behavior Sequential Pattern Database

We have constructed runtime behavior sequence database in III-C, in which each malware family maintains several applications with several runtime behavior sequences. Let F donate all malware families, expressed as $F = \{f_1, f_2, \cdots, f_n\}$. Given a malware family f_i (i.e., $f_i \subseteq F$ for $1 \leq i \leq n$) which holds a set of applications, donated as $\{a_1, a_2, \cdots, a_m\}$. Then the runtime behavior sequences maintained by f_i can be donated as $N\_Behavior(f_i) = \{N\_Behavior(a_1), \cdots, N\_Behavior(a_m)\}$. We input $N\_Behavior(f_i)$ into the $RB\_AprefixSpan$ algorithm and analyze the association between these sequences to discover the frequently occurred sequential patterns, donated as $S\_Pattern(f_i)$. $S\_Pattern(f_i)$ is stored to construct the behavior sequential pattern database.

F. Detection

Once the package file of an unknown application is submitted, MineRBS firstly leverages CopperDroid to extract its runtime behavior sequences. And then MineRBS matches the behavior sequential pattern database to identify whether the application is malicious.

IV. EVALUATION

In this section, we perform several experiments to evaluate MineRBS. We firstly introduce the experimental samples (IV-A), and then present the evaluations on the correctness and performance of $RB\_AprefixSpan$ comparing with PrefixSpan (IV-B). Finally, we evaluate MineRBS and find that our approach can achieve near-perfect effectiveness in detecting malware variants and promising effectiveness in detecting novel malware (IV-C).

A. Samples

The construction and evaluation of MineRBS are mainly based on a sizable dataset of real-world Android malware samples, which we split into two parts. The first part was collected and characterized by Jiang et al. originally [11], and extended by Arp et al. into the Drebin dataset later [21]. The dataset contains 5560 malware samples from which we extract runtime behaviors for 5246. For the remaining 314 ones, we find most are not valid. The second dataset consists of 90 novel malware samples, in which some are recently gathered from security vendors, and others are developed by ourselves. We classify these novel malware samples based on their runtime behaviors. The category of the novel malware and their number in each category are listed in Table VII. In addition, we employ 500 benign samples to evaluate the capability of MineRBS on identifying benign applications.

TABLE VII. Novel Malware Category

Malware Category	**No. of Malware in Each Category**
S_1: File Access	13
S_2: Exec External Application	18
S_3: Network Access	14
S_4: Personal Info. Access	25
S_5: Send SMS	11
S_6: Make/Alter Call	9

B. Evaluation of $RB\_AprefixSpan$ compared with PrefixSpan

a) Correctness: In the experiment, we pick 500 malware covering several malware families to construct the runtime behavior sequence database. The database is one of the input of PrefixSpan and our $RB\_AprefixSpan$ algorithm. The other input is $min\_support$. Through repeated experiments, we set $min\_support$ to be 3. Finally, we construct two behavior sequential pattern databases for the selected malware families by using PrefixSpan and $RB\_AprefixSpan$ respectively. Table VIII shows the result set of length-k patterns, we only give parts of patterns here. We can get from Table VIII that, the result set of length-k patterns mined by $RB\_AprefixSpan$ and PrefixSpan is the same. Thus, we can prove the correctness of $RB\_AprefixSpan$.

TABLE VIII. Length-k patterns mined by PrefixSpan and $RB\_AprefixSpan$

Length-k patterns	PrefixSpan	RB_AprefixSpan
Length-1 patterns	$< B2 >< B6 >$ $< B7 >< B9 >$ $< B10 >< B13 >$ $< B16 >$	$< B2 >< B6 >$ $< B7 >< B9 >$ $< B10 >< B13 >$ $< B16 >$
Length-2 patterns	$< B2, B6 >< B2, B9 >$ $< B2, B13 >< B6, B10 >$ $< B9, B10 >< B10, B16 >$	$< B2, B6 >< B2, B9 >$ $< B2, B13 >< B6, B10 >$ $< B9, B10 >< B10, B16 >$
Length-3 patterns	$< B2, B6, B10 >$ $< B2, B9, B10 >$ $< B6, B10, B16 >$ $< B9, B10, B16 >$	$< B2, B6, B10 >$ $< B2, B9, B10 >$ $< B6, B10, B16 >$ $< B9, B10, B16 >$

b) Performance: Fig. 3 compares $RB\_AprefixSpan$ with PrefixSpan on the time performance. We set the size of runtime behavior sequence database to be 200, 500, 1000 and 2000, $min\_support$ to be 3, and then evaluate the running time of the two algorithms respectively. We can get from Fig. 3 that, as the size increases, $RB\_AprefixSpan$ outperforms PrefixSpan. In pratical test, we find our $RB\_AprefixSpan$ algorithm has a much better space-time performance in detecting.

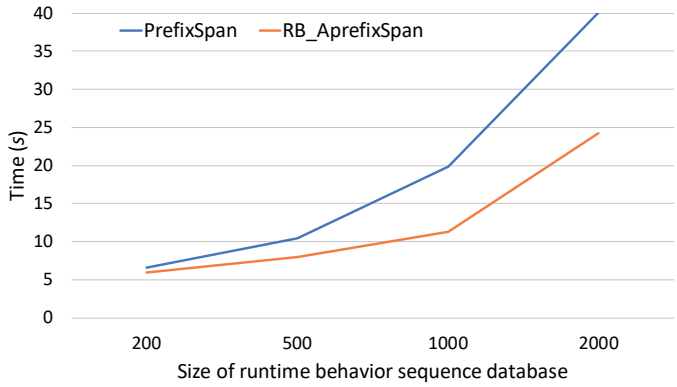

Figure 3. Time performance of RB_AprefixSpan and PrefixSpan.

C. Evaluation of MineRBS in Detecting Malware

a) Effectiveness on detecting malware variants: To evaluate the effectiveness of MineRBS on detecting malware variants, we take one famous malware family in Drebin database (i.e., DroidKungfu) as example in the following analysis. The DroidKungfu malware family consists of four variants (i.e., DKF1, DKF2, DKF3 and DKF4), in which DKF1 and DKF2 perform malicious behaviors, e.g., reading/writing file, executing root exploit, etc; DKF3 and DKF4 leverage various methods (like code obfuscation) to evade static detections.

We perform experiments to evaluate the effectiveness of MineRBS in applying behavior sequential patterns mined from several malware (e.g., DKF1, DKF2 and DKF3) to detect malware variant within the same malware family (e.g., DKF4) and benign samples. The results are shown in Table IX.

TABLE IX. Detection Results of MineRBS

Malware Variants	No. of Samples	TPR	FNR	TNR	FPR	ACC
DKF1	30	1.00	0.00	1.00	0.00	100.0%
DKF2	30	0.97	0.03	1.00	0.00	99.8%
DKF1	295	0.93	0.07	1.00	0.00	97.5%
DKF1	90	0.91	0.09	1.00	0.00	98.6%
DKF1	445	0.93	0.07	1.00	0.00	98.8%

We set $min\_support$ to be 3 in our experiment. For example, we first use runtime behavior sequences of all DKF1, DKF2 and DKF3 samples to construct the behavior sequential pattern database. Then, we match runtime behavior sequences of all DKF4 and benign samples to the database. We find that, 82 out of 90 are identified as malware variants, so TPR is 82/90. There are 8 DKF4 samples which are not detected, so FNR is 8/90. What's more, all 500 benign samples are identified as non-malicious correctly, so TNR is 1 and FPR is 0. Besides, we can get from Table IX that, the detection accuracy of DKF3 and DKF4 is a little lower than that of DKF1 and DKF2. The reason may be that some DKF3 and DKF4 variants change runtime behavior sequences in interacting with the command and control server. In summary, the detection accuracy of MineRBS on detecting malware variants is above 95%.

b) Effectiveness on detecting novel malware: In this experiment, we use 90 samples discussed above to test MineRBS's effectiveness on detecting novel malware. We first use runtime behavior sequences of all 5246 malware samples to construct the behavior sequential pattern database. Then, we match runtime behavior sequences of these novel malware to the database, and gain a detection accuracy of nearly 87.8%. The result reveals our $RB\_AprefixSpan$ algorithm is promising in detecting novel malware, as most novel malware are in fact variants from existing malware families, of which the core runtime behavior sequences are similar.

V. Conclusion

In this paper, we present the design of an approach, named MineRBS, which detects Android malware based on sequence pattern mining of runtime behavior sequences of

applications. The novel sequential pattern mining method $RB\_AprefixSpan$ proposed by MineRBS can discover runtime behavior sequential patterns from known malware families and build the behavior sequential pattern database to detect malware. In addition, $RB\_AprefixSpan$ algorithm uses abbreviated projection database to replace the projection database in PrefixSpan to improve the spatial performance. The experimental results show that $RB\_AprefixSpan$ can effectively dig out frequent behavior sequential patterns for malware detection, with better space-time performance than existing sequence pattern mining algorithms. What's more, experiments verify the effectiveness of MineRBS on detecting both malware variants and novel malware.

REFERENCES

[1] Gartner, Gartner says Android has been growing market share in the smartphone operating system market, which in the first quarter of 2017 is at 86.1%, http://www.gartner.com/newsroom/id/3725117, 2017.

[2] Mobile security risk report, 2017 q1, http://bbs.360.cn/thread-14972358-1-1.html.

[3] T. Debiaze, "Detecting malicious behavior for android applications by static analysis," Online: https://github.com/maaaaz/androwarn, 2015.

[4] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "RiskRanker: scalable and accurate zero-day android malware detection," in International Conference on Mobile Systems, Applications, and Services, 2012, pp. 281-294.

[5] V. Afonso, A. Bianchi, Y. Fratantonio, A. Doupé, M. Polino, and P. D. Geus, "Going Native: Using a Large-Scale Analysis of Android Apps to Create a Practical Native-Code Sandboxing Policy," in Symposium on Network and Distributed System Security (NDSS'16), 2016.

[6] G. Suarez-Tangil, S. K. Dash, M. Ahmadi, J. Kinder, G. Giacinto, and L. Cavallaro, "DroidSieve: Fast and Accurate Classification of Obfuscated Android Malware," in ACM on Conference on Data and Application Security and Privacy, pp. 309-320, 2017.

[7] J. Seo, D. Kim, D. Cho, T. Kim, I. Shin, and J. Seo, "FLEXDROID: Enforcing In-App Privilege Separation in Android," in Symposium on Network and Distributed System Security (NDSS'16), 2016.

[15] H. Yang, Y. Q. Zhang, Y. P. Hu and Q. X. Liu, "Android Malware Detection Method Based on Permission Sequential Pattern Mining Algorithm," in Journal on Communications [J], S1, 2013, pp. 106-115.

[8] L. K. Yan, and H. Yin, "DroidScope: Seamlessly Reconstructing the OS and Dalvik Semantic Views for Dynamic Android Malware Analysis," in Proceedings of the 21st USENIX conference on Security symposium, 2013, pp. 29-29.

[9] V. Rastogi, Y. Chen, and W. Enck, "AppsPlayground: automatic security analysis of smartphone applications," in ACM Conference on Data and Application Security and Privacy, 2013, pp. 209-220.

[10] K. Tam, S. J. Khan, A. Fattori, and L. Cavallaro, "CopperDroid: Automatic Reconstruction of Android Malware Behaviors," in Network and Distributed System Security Symposium, 2015.

[11] Y. Zhou, and X. Jiang, "Dissecting android malware: Characterization and evolution," in 2012 IEEE Symposium on Security and Privacy, 2012, pp. 95-109.

[12] Symantec. The future of Mobile Malware, accessed on Nov. 2016.

[13] P. Feng, J. Ma, and C. Sun, "Selecting Critical Data Flows in Android Applications for Abnormal Behavior Detection," in Mobile Information Systems [J], 2017, in press.

[14] S. K. Dash, G. Suareztangil, S. Khan, K. Tam, M. Ahmadi, and J. Kinder,"DroidScribe: Classifying Android Malware Based on Runtime Behavior," in Mobile Security Technologies, 2016, pp. 252-261.

[16] H. Mannila, H. Toivonen, and A. Inkeri Verkamo, "Discovery of frequent episodes in event sequences," in Data Mining & Knowledge Discovery [J], 1997, 1(3), pp. 259-289, in press.

[17] R. Agrawal, and R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases," in International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc., 1994, pp. 487-499.

[18] J. Pei, J. Han, B. Mortazavi-Asl, and H. Pinto, "PrefixSpan: mining sequential patterns efficiently by prefix-projected pattern growth," in International Conference on Data Engineering, 2002, pp.215-224.

[19] R. Srikant and R. Agrawal, "Mining sequential pattems: Generalizations and performance improvements," in International Conference on Extending Database Technology (EDBT'96), 1996, pp. 3-17.

[20] J. Han, J. Pei, M. C. Hsu, "FreeSpan: frequent pattern-projected sequential pattern mining," in ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2000, pp.355-359.

[21] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket," in Network and Distributed System Security Symposium, 2014.