Malware Phylogeny Analysis using Data-Aware Declarative Process Mining

Pasquale Ardimento
Dept. of Informatics
University of Bari Aldo Moro
Bari, Italy
pasquale.ardimento@uniba.it

Mario Luca Bernardi
"Giustino Fortunato" University
Benevento, Italy
m.bernardi@unifortunato.eu

Marta Cimitile
Unitelma Sapienza University
Rome, Italy
marta.cimitile@unitelmasapienza.it

***Abstract*—Mobile phones are currently the main targets of continuous malware attacks. Usually, new malicious code is generated conveniently changing the existing one. According to this, it becomes very useful to identify new approaches for the analysis of malware phylogeny. This paper proposes a data-aware process mining approach performing a malware dynamic analysis. The process mining is performed by using a multiperspective declarative approach allowing to model a malware family as a set of constraints (within their data attributes) among the system call traces gathered from infected applications. The models are used to detect execution patterns or other relationships among families. The obtained models can be used to verify if a checked malware is a potential member of a known malware family and its difference with respect to other malware variants of the family. The approach is implemented and applied on a dataset composed of 5648 trusted and malicious applications across 39 malware families. The obtained results show great performance in malware phylogeny generation.**

***Index Terms*—Malware phylogeny, Data-aware declarative process mining, multi-perspective declare**

I. Introduction

More and more often sensitive data and online services are accessed through mobile phones. This explains the increasing number of mobile malware attacks and their always greater aggressivity. Continuously, new malicious code is obtained as a variant of existing malware also with the support of automatic tools [1]. This encourages the birth and the study of new solutions to malware detection and malware phylogeny analysis. In particular, malware phylogeny consists to study the malware evolution, similarities, and relationships [2]. The deriving advantage is a higher comprehension of the malware useful to identify new anti-malware strategies. According to this, in this paper, a new approach to malware phylogeny is proposed. The main assumption is that the behavior of a malware can be captured and modeled mining the system calls traces generated during the execution of applications infected with that malware. In particular, the common parts of a set of different applications infected with a given malware family can be used to obtain a sort of fingerprint of the malware family. The malicious behavior model is expressed as a set of declarative constraints [3] within their data conditions and is called *Data-aware System Calls Execution Fingerprint* (DSEF). The mining of the infected application code is here performed by using the multiperspective Declare (MP-Declare) [4] miner. MP-Declare allows discovering the application models also considering the data conditions associated with each syscall. This paper starts from the approach described in [5, 6, 7] that proposes the adoption of the Declare Process Mining tool [3] to support malware analysis and malware phylogeny. With respect to [5], here we introduce a data-aware based process mining technique able to capture the constraints associated with a set of syscall traces within their data attributes. Our idea is that the addition of data attributes analysis can improve the malware modeling capability of the process mining technique since malware behavior is often activated and conditioned to the value assumed by the data associated with the syscalls.

The proposed approach is applied on a dataset composed of 39 malware families and 5468 malicious applications.

The rest of paper is organized as follows. Section 2 reports a background overview on malware families, malware phylogeny and MP-Declare language. Section 3 proposes a discussion of related work. Section 4 describes the malware phylogeny approach. Section 5 reports an experiment describing the application of the malware phylogeny approach on a real dataset. The obtained results are discussed in Section 6 while in Section 7 the Threats to validity are discussed. Finally, Section 8 gives some conclusive remarks and describes our future work.

II. Background

A. Malware phylogeny and malware family

We consider a malware family as a set of malware with similar behavior and characteristics. The knowledge of the properties of a malware family should support malware phylogeny defined as an estimation of the derivation relationships between malware variety of families [2]. Example of Android malware families are reported in the first three columns of Table I respectively containing the family name, the number of downloaded samples (#DS) and the number of samples included in the experiment (#IS).

Malware is usually downloaded and installed by the deceived users in different ways: the different installation types (IT) are described in [8]. For example, possible values of IT are repackaging (r), standalone (s), and update attack (u). Repackaging is when the first downloaded application is benign and successively it is repackaged with additional malicious

Submitted to IEEE-EAIS 2020 conference.

TABLE I
A LIST OF SOME COMMON MALWARE FAMILIES.

Family	#DS	#IS	Activating Events (AEs)
Asroot	168	129	BOOT, CALL, SMS
CruseWin	165	126	BOOT, NET
Tapsnake	166	125	BOOT, CALL, NET, SMS
Geinimi	173	125	MAIN
DroidKungFu4	180	123	BATT, BOOT, SYS
AdWo	163	122	BOOT, CALL, SMS
DroidKungFuSapp	172	121	BATT, BOOT, SYS
PjApps	154	119	BATT, BOOT, SMS
Benign	149	115	BOOT
AnserverBot	171	113	BOOT, CALL, NET
DroidKungFu2	173	112	BATT, BOOT, SYS
DroidKungFu3	179	112	BATT, BOOT, SYS
RogueSPPush	147	111	BOOT, NET, SMS
Zsone	152	109	BOOT
Opfake	158	107	MAIN
SndApps	154	104	BOOT, NET, SMS
GPSSMSSpy	149	104	BOOT
ADRD	144	101	BOOT, CALL
GingerMaster	154	100	BOOT
Gone60	128	99	BOOT, CALL, NET, SMS
FakeNetflix	143	98	BOOT, CALL, NET
KMin	155	98	BOOT
NickySpy	131	97	BOOT
DroidDreamLight	125	95	MAIN
BaseBridge	139	93	BATT, BOOT, NET, SMS
GoldDream	148	93	BOOT, MAIN, NET, SMS
jSMSHider	118	92	BOOT
DroidDream	145	90	BOOT
FakeInstaller	116	90	BOOT, CALL
Plankton	125	87	BOOT
DroidCupon	115	87	BOOT
AirPush	134	87	BATT, BOOT, SMS
YZHC	106	84	MAIN
Boxer	128	81	MAIN
HippoSMS	103	72	BOOT
zHash	92	67	MAIN
Bgserv	82	62	BOOT
DroidKungFu1	82	52	BATT, BOOT, SYS
BeanBot	82	52	BOOT, MAIN

payloads. Update attacks consist to inject malicious payloads in the updated versions of the application. Finally, standalone attacks are able to redirect users to download malware. We restrict our interest to repackaging and standalone malware since we need the infected package to perform model extraction. Another aspect discriminating different malware family are the activating events (AEs). They refer to the events that activate the malicious behaviour. AEs belongs to several broad classes depending on the activated functionalities: `BOOT`, `BATT`, `SMS`, `SYS`, `NET`, `CALL`.

The abbreviation `BOOT` refers to the `BOOT COMPLETED` event. `BATT` is then used to describe events representing a specific state of the battery (i.e., `battery low`, `battery ok`, `action power connected`, `action power disconnected`, `batter change action`). Moreover, `SMS` refers to `sms received` event. The abbreviation `SYS` includes `user present`, `input method changed`, `sim full`. `net` corresponds to `connectivity changed` or `pick wifi work` acitvation events. Finally, `CALL` event consists of `new outgoing call` or `phone state`.

B. Multi-perspective Declare

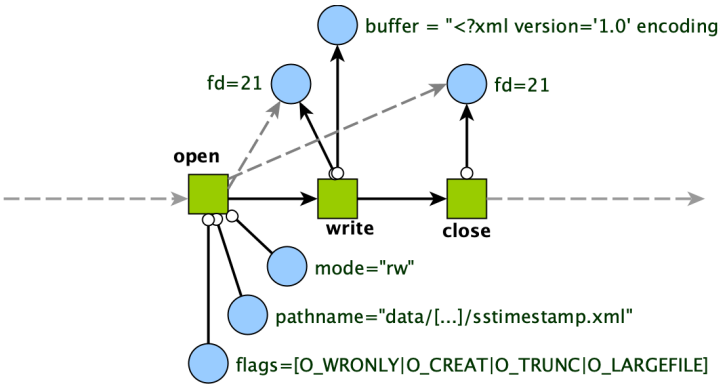
The code mining is performed in this study using a tool called MP-Declare [4, 9]. It is the multi-perspective version of the Declare tool introduced in [10]. The Declare modeling language [10, 11] represents each process as a set of constraints that must be satisfied during its execution. Specifically, all the sequences of activities that do not violate the constraints are allowed. Moreover, each constraint can be seen as a concrete instantiation of a set of templates represented as a class of properties. From the templates, the constraints inherit both their semantics and graphical representation. Declare uses LTL semantics [12] because it ensures that the processes can be verifiable and executable throughout a finite number of traces. The implementation of Declare-based process mining is obtained throughout the tools described in [13]. MP-Declare [4, 9] adds to Declare a multi-perspective vision using a Metric First-Order Linear Temporal Logic (MFOTL) that allows ensuring a new time and data perspective.

To understand the MFOTL we have to introduce the definition of event payload. Let us consider, for example, the activity *receive call*(C) executed at the timestamp ts. This activity assigns to the attribute *caller* the value *Simon* representing the *payload of C*. Considering, for example, that activities *receive call* and *filter call* are related by a response constraint, it corresponds to the fact that the activity *receive call* is always eventually followed by activity *filter call*. Additionally, a timed semantics is implied in the above model thought some additional conditions on the data: they can be an *activation condition* φ_a and a *correlation condition* φ_c. The *activation condition* can be seen as a relation among the variables (event logs global attributes) that must be valid to ensure the activation (when the activation condition is not verified, the constraint is not activated). Further details, in particular on the semantics of MP-Declare templates, are described in section 2.2. of [9].

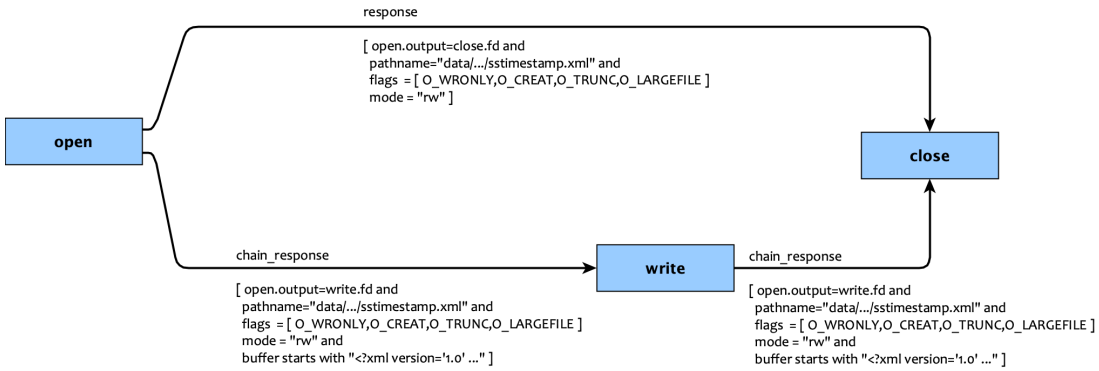
This study uses the MP-Declare tool and formalization [9] to discover the constraints (within their attributes) useful to represent the behavior of a malicious application.

III. RELATED WORK

The topic of malware phylogeny is debated in several studies in the last years. Authors in [14], proposes a method based on a set of features (called n-perms) useful to match different code permutations obtaining a malware tree description. The obtained models are suitable to support malware discovery, identify malware variants and verify the possible inconsistencies in malware naming. Differently, from our phylogeny approach, the method is static while malicious applications are never executed. Moreover, its robustness is quite reduced in case of code modifications that cannot be represented as permutations. Conversely, the proposed phylogeny approach is less sensitive to static code transformations while it is based on system call captured during malware execution. Another approach to malware evolution is reported in [15]. Authors define malware evolution relations using derivation graphs to describe path patterns. A huge limitation of this framework



(a) An excerpt of a syscalls execution trace



(b) An MP-declare process to which the trace in (a) is conformant

Fig. 1. A small running example.

is that it is unable to model automatically generated source code (a significant number of malware is generated using automatic tools). Differently, our method is suitable to analyze also automatically generated code. A taxonomy of malware is also obtained in [16]. Here authors explore the similarities and differences between malware variants using clustering algorithms to manually obtain the malware taxonomy. The limitation of this approach is the absence of a reference phylogeny model. A malware shellcode phylogeny study is also proposed in [17]. The method is mainly based on a data-centric approach allowing to perform packets inspections to automatically identify and evaluate the shellcode similarity. With respect to our approach, this method is more focused on malware behavior analysis. An approach based on the analysis of logs is reported in [18, 19]. Here authors capture possible modifications among application running obtaining a phylogeny tree. In this study, authors execute traces at APIs level. For this reason, differently from our phylogeny approach, they are affected by API version. Finally, the proposed approach represents an evolution of [5]. In particular, the approach proposed in [5] is here extended and improved with the addition of a data-aware perspective.

IV. Approach

This study proposes an approach consisting of three main steps: i) compute of the DSEFs of each analyzed malware families; ii) evaluate the similarities among the DSEFs; iii) perform a clustering analysis to obtain a phylogeny model for the analyzed families.

In the first step, for each considered malware family, we mine the system call traces of a set of infected Android applications when an AE (the list of considered AE is reported in Table I) has occurred. The mining is performed by using a data-aware process mining technique and the obtained constraints (within their attributes) are used to generate the models (or fingerprints) of the behavior of a malware family infecting all the mined applications. The malware SEFs of different families are then compared and their similarities are evaluated. Finally, clustering analysis is performed to study malware phylogeny and classify new malware basing on its membership to a given family. Starting from the logs captured during the execution of the infected applications, the proposed approach is able to analyze the system call traces to characterize the discovered malware behavior.

In the following, the three steps are further detailed.

A. Compute the SEFs of a malware family

The DSEF computation process is summarized in Figure 2. Given a set of applications (APKs) infected with a malware family M, the corresponding DSEF is built.

The DSEF construction for a malware family requires a generation process involving the usage of an android device (a real device or a sandbox) used to generate syscall traces from one or more application packages (APKs).

The DSEF computation process as provided in Figure 2 takes as input the considered APK (infected with the malware family M) and extract the syscalls traces generated in response to an AE (we are assuming that the malicious behavior is usually triggered by a set of activating events [5, 20]). All the APKs of Table I are executed on an emulator of the Android device [1]. For each APK, all the corresponding AE (Table I) are tested one for time (each event is sent more than once to the APK in order to have a huge number of samples).

The generated traces have a textual format. In particular, each APK is installed and started and during its running, a system event is sent to the emulator. The generated system calls are then captured until the APK state return to be stable, and the APK running is stopped. After each stop, the emulator is cleaned and restarted. The APK is newly installed to ensure that each run has similar conditions. The scanning of all the AE is obtained using some ad hoc shell scripts. Successively, the CSV generation phase is started. The collected syscall traces are now converted from text to CSV format in order to be processed by the MP-Declare Miner [9]. In the conversion step, all the useless information is filtered out while it is retained information like session attributes, application id, system call occurrence (its timestamp, the id of its requesting process and the ordered list of arguments). This information allows generating, for each log event, the event payload. The data correlation is performed by using a reference-based correlation approach [5] using a correlation function that considers both the process id and the event timestamps attributes. The events payloads are used by subsequent process mining step to infer activation and correlation conditions for activities. The set of DSEF computation activities starts by taking the CSV logs for each malware family and builds the corresponding DSEF. The CSV logs associated with each event are filtered using the Gaussian distribution of the sizes of the logs (all logs outside the 80th percentiles are filtered out having a high probability to be incorrect).

The remaining logs are processed by the MP-Declare tool obtaining the model for each system event. In this step, we are assuming that each model (DSEF of the malware family) is a set of constraints (and their attributes) characterizing the behavior of the shared malware part while the parts that are specific to the various applications (they are different from trace to trace) are discarded.

To further explain the concepts of DSEF, we introduce a running example synthesized in Figure 1. Figure 1-(a) reports an excerpt of a syscall execution trace obtained from an Android application. A possible process to which this trace is conformant is then reported in Figure 1-(b). The syscall execution events reported in Figure 1-(a) are generated by an application able to opens a file, write some data and close it. The figures show that each syscall in the execution trace is associated with an activity in the process that defines the event payload structure in terms of attributes and their types. During the syscall trace parsing, events and the values of their attributes are created and added to the process log used for a subsequent mining step. From the syscall logs, a behavioral model called Data-aware Syscalls Execution Fingerprint (DSEF) is then extracted. The DSEF is obtained using the MP-Declare notation [9] in order to represent correlation conditions among syscall taking also into account data parameters obtained from the application execution traces. In the following we introduce the definitions of the MP-Declare model and of DSEF of a malware family.

Definition 1 (MP-Declare model)**.** *The MP-Declare model (MPD) for a set of system call traces T is obtained as:*

$$MPD = \{C_1, \ldots, C_n\}$$

$C_h = (S_A, S_T, P)$ is a unary or binary constraint specifying a condition P. The constraints are satisfied if P holds over traces in T:

- *on the occurrences of each system calls S_A, for unary constraints;*
- *on each couple $(S_A,\ S_T)$ of activating and target system calls, for binary constraints.*

Definition 2 (DSEF of a malware family M)**.** *The DSEF of a malware family M is defined as:*

$$DSEF(M) = \{MPD_{M_1}, \ldots, MPD_{M_n}\}$$

where:

- *MPD_{M_j} is the MP-Declare model for event j mined from the set of traces $\{t_{j1},\ \ldots\ ,\ t_{jm}\}$*
- *t_{ji} is the execution trace generated by the i-th application of the set A in response to the j-th system event, with $i \in [1, m], j \in [1, n]$;*

Notice that in the above definition we use A to describe the set of m applications infected with M while n is the number of system events sent to each application.

Basing on the above definition, the DSEF of a malware family M is computed and modeled as the shared malicious behavior of the entire set of applications infected by M. According to this, it is necessary a single run for each application included in A to extract the set of constraints characterizing the shared malicious behavior of all the applications. Specifically, on the base of the results of [5] we can assume that the common parts of a set of different applications infected with the malware family M determine the unique fingerprint (expressed as a set of constraints) of M.

[1] https://developer.android.com/studio/run/emulator.html

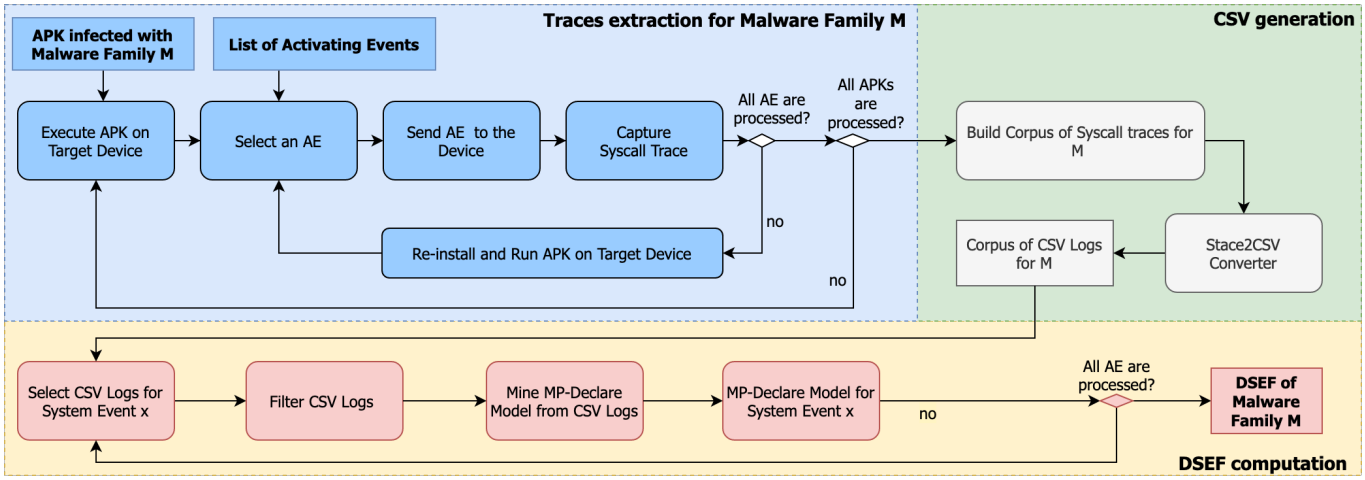


Fig. 2. Computing the DSEF of a Malware family M.

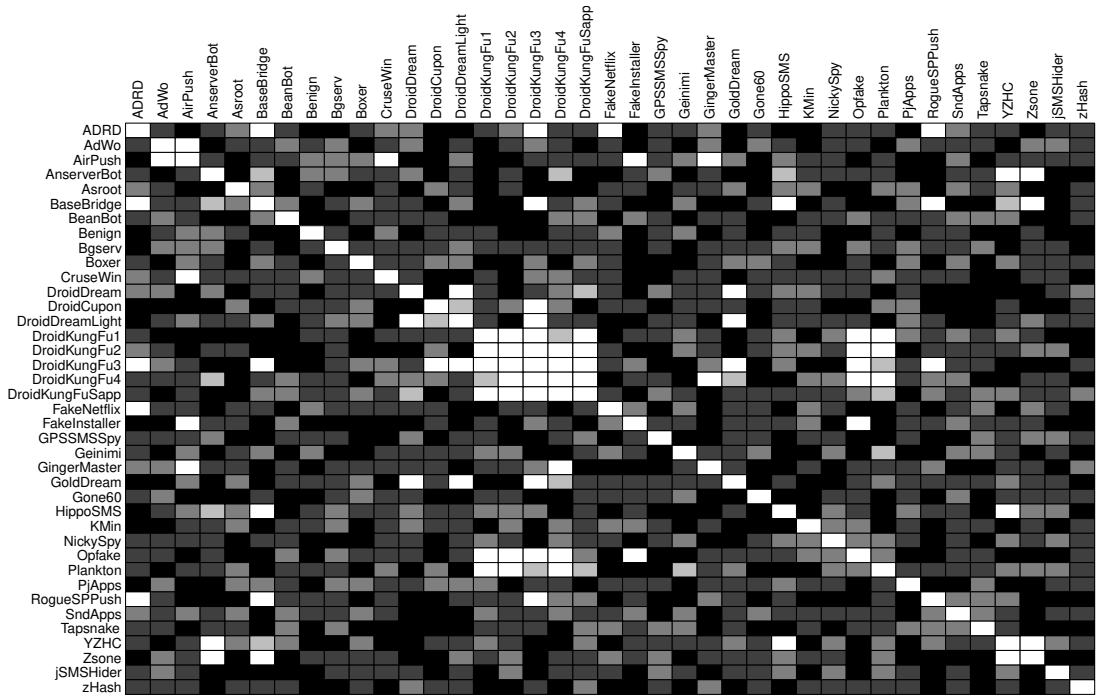


Fig. 3. The dissimilarity matrix obtained evaluating the distances among MP-declare models on the dataset.

B. Evaluate the similarities among the DSEFs

To evaluate the similarities among malware families we calculate the distance among their SEFs. To this aim, we introduce the concept of distance between two MP-Declare models MPDi and MPDj.

Definition 3 (Distance between two MP-Declare models)**.** *We define the distance $\sigma(MPD_i, MPD_j)$ between two MP-Declare models MPD_i and MPD_j as:*

$$\sigma(MPD_i, MPD_j) = \frac{\sum_{h=1}^{k} \sigma(C_{ih}, C_{jh})}{|E_i| + |E_j| + \sum_{h=1}^{k} \sigma(C_{ih}, C_{jh})}$$

where:

- *$C_{ih} = (T, P_{ih})$ and $C_{jh} = (T, P_{jh})$ are the k constraints with the same template present in both models and $\sigma(C_{ih}, C_{jh})$ is the tree edit distance among the expression trees of their predicates (P_{ih} and P_{jh});*
- *E_i and E_j are the sets of constraints present, respectively, only in model MPD_i and in model MPD_j;*

The distance between the expression trees P_{ih} and P_{jh} is computed as described in [21] and normalized as described in [22].

According to the above definition, we can observe that the distance between models having the same constraints and correlation conditions is equal to zero while the distance between models having different constraints or completely different correlation conditions is equals to one (it is the maximum distance).

Similarly, the distance between two DSEFs A and B, is computed as the average of the distances between the models A and B of all the considered AE:

$$\sigma(DSEF_A, DSEF_B) = \frac{\sum_{i=1}^{n} \sigma(MPD_{A_i}, MPD_{B_i})}{n}$$

Starting from the DSEFs distance definition, a dissimilarity matrix is computed. In the matrix, each entry i, j reports the dissimilarity between the DSEF model of the family i and that of the family j.

C. Clustering Analysis

This step is aimed to recover a phylogeny model using the Hierarchical Agglomerative Clustering (HAC) algorithm reported in [23] on the DSE dissimilarity matrix. HAC algorithm is very used for the construction of the phylogeny model both in security and biology domains. It provides impressive results with the best trade-off with respect to performances. However, the limit of the obtained phylogeny model is that it is unable to represent multiple inheritances: the lineage is always a linear path. As a consequence, if a malware derives from more than one family, the model exclusively identifies the closest parent.

V. Experiment Description

A. Dataset description

We assess our approach on a dataset built on a large set of the 39 infected applications families reported in Table I. The dataset includes samples from Drebin [5] and Genoma Dataset [26] and other specifically collected for this work. The dataset reliability is ensured by submitting the infected application to well-known anti-malware services [2]. Using several different anti-malware on the entire dataset, all the samples not detected as infected from at least ten different anti-malware were discarded. Table I reports, in the third column, the samples included in our study after quality checks. Specifically, for each family, the table reports the number of samples downloaded from the internet (#DS) and the number of samples included in the study (#IS) that pass the filtering step. As the table highlights, the initial downloaded dataset is composed of few more than 5,000 infected samples and, from these, almost 4,000 samples are selected to be used in our experiment. Each selected sample consists of an infected android package (APK) that can be installed on an Android smartphone. This allows to perform all the sequences of activities that are described in IV (i.e., syscall trace setting, SEF models generation, distance among the DSEFs evaluation and the phylogeny model recovery).

B. Experiment setting

The experiment consists in building the dendrogram of the 39 families of Table I. The infected applications are executed and traces are collected and analyzed as specified in the process of Figure 2 to build the process models for each malware family. The models are used to build a dissimilarity matrix on which to perform the hierarchical clustering step to derive a dendrogram showing the relationships among the considered families.

VI. Discussion of Results

The adoption of the agglomerative clustering applied to the dissimilarity matrix of Figure 3 resulted in the dendrogram of Figure 4 that groups together the 39 malware families under study for the behavioural similarity of the syscall they execute. The Figure 4 groups the cluster of families having stronger relationships. The families BaseBridge, ADRD and RogueSPPush show a very high membership degree. This suggests that there is a direct lineage among these three malware families. This is also true for the families Zsone, YZHC and HippoSMS since they exhibit quite similar behaviour. Another almost isolated cluster in terms of behavioural similarity is the one made up by DroidDream, DroidDreamLight and GoldDream. Moreover, these families have very low similarity to other families and, at the same time, the other families have negligible levels of similarity to them. Another group that exhibits strong membership levels is the one comprised by DroidKungFu families, Plakton and Opfake: this suggests that the DroidKungFu variants, along with Opfake and Plankton, are very close and hence have common roots. This group also confirms the well-known lineage among the six DroidKungFu versions (this strong heritage is confirmed by several works [8], [12], [15]). Looking at the dendrogram it can also be seen that there is also a strong relationships among Geimini, Plakton and Opfake families. This confirms what reported in several studies [24]: infact the use of network resources to steal sensitive information that avoids sending SMS is a common behavior of both Geinimi and Plankton payloads.

VII. Threats to validity

Construct validity. Performing outliers analysis allowed us to reveal that some traces captured are incomplete. An example is provided by a situation where an application is shut down due to illegal behavior and a trace, even if incomplete, is captured. The problem lies in the way the automatic script works, not considering what happens during the capturing time. The script starts to capture the trace when a system event occurs (i.e., when it is sent by the sandbox to the application) and stops to capture when a new stable state is reached. Another threat could be represented by traces captured from malware applications that do not trigger malicious behavior in response to some system events. However, this kind of issue is well addressed by our approach by an effective detection during the training process looking at the distribution of distances among test and trusted applications. For example, our approaches

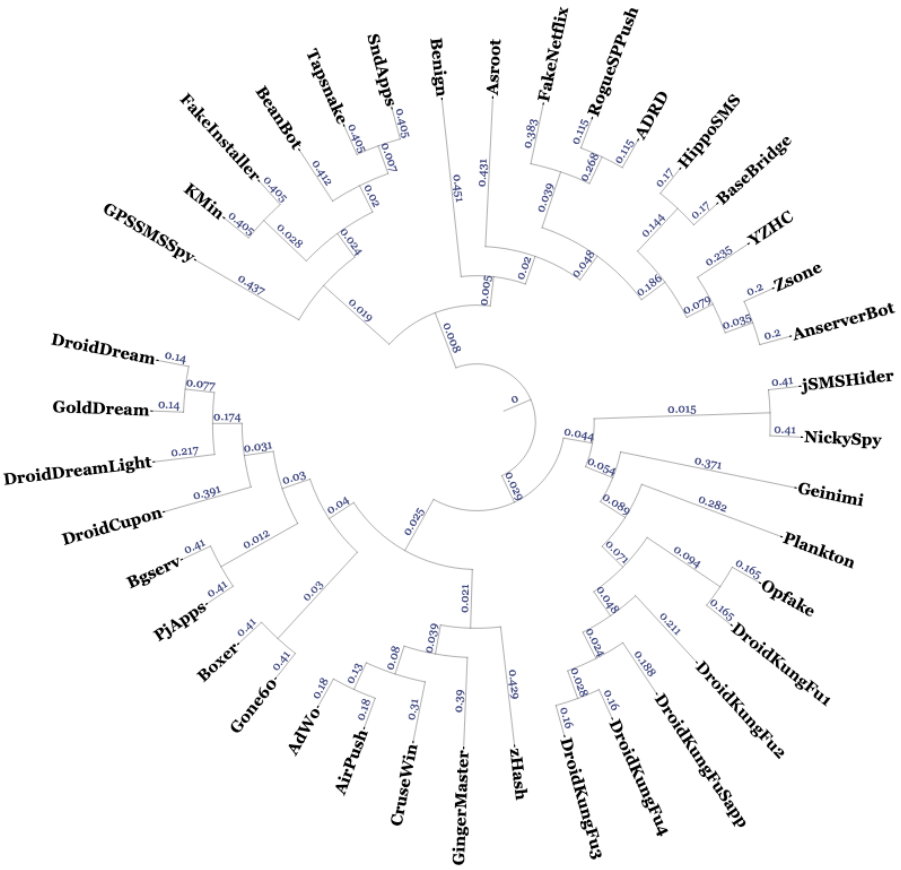


Fig. 4. The dendrogram showing the lineage relationships among families based on the behavior similarity of the invoked system calls.

recognize that DroidKungFu malware is triggering the malicious behavior for BOOT_COMPLETED, BATTERY_LOW and INPUT_MEDIA_CHANGED events allowing an effective detection. Anyway, for malware not triggering malicious behaviour for any of the system events, our approach is not able to perform well but this eventuality can be revealed during the model creation step.

The way the dataset is obtained could represent a threat. The assumption is that the considered applications are malicious on the base of the response of some antivirus software that does not provide any assurance. To reduce the number of false positives, i.e. legitimate applications identified as illegitimate, a combination of several anti-malware was used. We established that an application had to be considered as infected only when at least five different anti-malware recognized it for what it is.

External validity. The proposed approach has been evaluated on a very large number of applications, 5648 applications belonging to 39 malware families. However, to generalize the results, it is desirable to increase both the number of applications and the number of malware families.

VIII. Conclusions and future work

This paper proposes a process mining approach aimed at studying android malware phylogeny. The MP-declare language is used to mine a data-aware declarative process from the system calls traces of the malware applications. The process models represents the malware family fingerprint used to detect behaviour similarity. Specifically, the malware families similarities and the corresponding malware variants characterization are identified using a hierarchical clustering algorithm on the base of the distances between the mined models. The proposed approach is tested on more than 5,000 infected applications involving 39 malware families. The results highlight that the clustering approach is effective at comparing families basing on the similarity of their behavior executed by the malicious application at run-time. We plan to perform a wider assessment increasing the size of the dataset and the considered platforms.

References

[1] J. Jang, D. Brumley, and S. Venkataraman, "Bitshred: Feature hashing malware for scalable triage and semantic analysis," in *Proceedings of the 18th ACM Conference on*

Computer and Communications Security, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 309–320.

[2] M. Hayes, A. Walenstein, and A. Lakhotia, "Evaluation of malware phylogeny modelling systems using automated variant generation," *Journal in Computer Virology*, vol. 5, no. 4, pp. 335–343, 2008.

[3] W. van der Aalst, *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer Berlin Heidelberg, 2011.

[4] V. Leno, M. Dumas, and F. M. Maggi, "Correlating activation and target conditions in data-aware declarative process discovery," in *Business Process Management*, M. Weske, M. Montali, I. Weber, and J. vom Brocke, Eds. Cham: Springer International Publishing, 2018, pp. 176–193.

[5] M. L. Bernardi, M. Cimitile, D. Distante, F. Martinelli, and F. Mercaldo, "Dynamic malware detection and phylogeny analysis using process mining," *International Journal of Information Security*, vol. 18, no. 3, pp. 257–284, Jun 2019.

[6] G. Acampora, M. L. Bernardi, M. Cimitile, G. Tortora, and A. Vitiello, "A fuzzy clustering-based approach to study malware phylogeny," in *2018 IEEE International Conference on Fuzzy Systems, FUZZ-IEEE 2018, Rio de Janeiro, Brazil, July 8-13, 2018*. IEEE, 2018, pp. 1–8.

[7] A. Burattin, M. Cimitile, and F. M. Maggi, "Lights, camera, action! business process movies for online process discovery," in *Business Process Management Workshops*, F. Fournier and J. Mendling, Eds. Cham: Springer International Publishing, 2015, pp. 408–419.

[8] X. Jiang and Y. Zhou, *Android Malware*. Springer Publishing Company, Incorporated, 2013.

[9] V. Leno, M. Dumas, F. M. Maggi, M. L. Rosa, and A. Polyvyanyy, "Automated discovery of declarative process models with correlated data conditions," *Information Systems*, vol. 89, p. 101482, 2020.

[10] M. Pesic, H. Schonenberg, and W. M. P. van der Aalst, "Declare: Full support for loosely-structured processes," in *EDOC 2007*, 2007, pp. 287–300.

[11] M. L. Bernardi, M. Cimitile, C. Di Francescomarino, and F. M. Maggi, "Using discriminative rule mining to discover declarative process models with non-atomic activities," in *Rules on the Web. From Theory to Applications*, A. Bikakis, P. Fodor, and D. Roman, Eds. Cham: Springer International Publishing, 2014, pp. 281–295.

[12] M. L. Bernardi, M. Cimitile, C. D. Francescomarino, and F. M. Maggi, "Using discriminative rule mining to discover declarative process models with non-atomic activities," in *Rules on the Web. From Theory to Applications - 8th International Symposium, RuleML, Prague, Czech Republic, August. Proceedings*, 2014, pp. 281–295.

[13] F. M. Maggi, "Declarative process mining with the declare component of prom," in *Proceedings of the BPM Demo sessions 2013, Beijing, China, August 26-30, 2013*, 2013.

[14] M. E. Karim, A. Walenstein, A. Lakhotia, and L. Parida, "Malware phylogeny generation using permutations of code," *Journal in Computer Virology*, vol. 1, no. 1-2, pp. 13–23, 2005.

[15] A. Walenstein and A. Lakhotia, "A transformation-based model of malware derivation," in *Malicious and Unwanted Software (MALWARE), 2012 7th International Conference on*, Oct 2012, pp. 17–25.

[16] E. Carrera and G. Erdélyi, "Digital genome mapping–advanced binary malware analysis," in *Virus bulletin conference*, vol. 11, 2004.

[17] J. Ma, J. Dunagan, H. J. Wang, S. Savage, and G. M. Voelker, "Finding diversity in remote code injection exploits," in *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '06. New York, NY, USA: ACM, 2006, pp. 53–64.

[18] W. M. Khoo and P. Lió, "Unity in diversity: Phylogenetic-inspired techniques for reverse engineering and detection of malware families," in *SysSec Workshop (SysSec), 2011 First*. IEEE, 2011, pp. 3–10.

[19] B. Fazzinga, S. Flesca, F. Furfaro, and L. Pontieri, "Online and offline classification of traces of event logs on the basis of security risks," *J. Intell. Inf. Syst.*, vol. 50, no. 1, pp. 195–230, 2018.

[20] G. Meng, R. Feng, G. Bai, K. Chen, and Y. Liu, "Droidecho: an in-depth dissection of malicious behaviors in android applications," *Cybersecurity*, vol. 1, pp. 1–17, 2018.

[21] M. Pawlik and N. Augsten, "Tree edit distance: Robust and memory-efficient," *Inf. Syst.*, vol. 56, pp. 157–173, 2016.

[22] C. Z. Yujian LI, "A metric normalization of tree edit distance," *Frontiers of Computer Science*, vol. 5, no. 1, p. 119, 2011.

[23] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: A review," *ACM Comput. Surv.*, vol. 31, no. 3, pp. 264–323, Sep. 1999.

[24] M. Salehi, M. Amini, and B. Crispo, "Detecting malicious applications using system services request behavior," in *Proceedings of the 16th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, ser. MobiQuitous '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 200–209.