Data-Aware Declarative Process Mining for Malware Detection

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

Lerina Aversano
Dept. of Engineering
University of Sannio
Benevento, Italy
aversano@unisannio.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 devices have become, in the last years, an essential tool used to perform daily activities. However, they also have become the target of continuous malware attacks usually coming out from new malware obtained as a variant of existing ones. For this reason, we suppose that by comparing the behavior of a new application with those of known malware applications it is possible to define it as malicious or trusted. According to this, the current study proposes an approach based on a data-aware declarative process mining technique to identify similarities and recurring patterns in the system call traces generated by a set of malicious mobile applications. The obtained characterization, represented by a set of declarative constraints within their data attributes, can be considered as a run-time fingerprint of a malware useful to evaluate the membership of a new application to a given malware family. The empirical validation of the proposed approach is performed on a dataset of more than 1200 trusted and malicious applications coming out from eight malware families and the obtained results show a very good discrimination ability.**

***Index Terms*—Malware detection, Declarative process mining, data-aware process mining**

I. Introduction

Mobile devices are usually used to communicate and to access to sensitive data, such as personal certificates, email or banking account. This explains the increasing number of malware attacks aimed to capture reserved information or reduce mobile service performances. New malware is continually obtained from existing malicious code [8] by using automatic tools able to perform code exchange. Consequently, new malware detection approaches are continually developed to detect the always more sophisticated and complex malicious code. This paper presents an update version of the approach presented in [4] consisting to use a process mining (PM) technique [21] for the dynamic analysis of the system call traces gathered from an application. With respect to [4], this new approach allows analyzing system call traces within their data attributes. The main hypothesis at the base of this proposal is that the study of the data attributes should improve malware detection and give further information about the malware behavior. Moreover, the proposed approach assumes that similarities and derivations between mobile application system calls can be discovered and modeled similarly to process activities in business process logs. Indeed, the PM is used to describe the behavior of trusted/malware applications from a set of system calls traces obtained in response to some activating events. The PM tool used to discover the trusted/malware application behavior model from a set of collected traces is based on MP-Declare [12], a multi-perspective version of Declare Miner [5]. The model is represented as a set of declarative constraints between system calls [17] within their correlating data and is named *Data-aware System Calls Execution Fingerprint* (DSEF). The approach is suitable for all the mobile platforms but in this study, we focus on the Android platform while it is considered the favorite target of malware attacks [1]. The novelty introduced by the proposed approach consists of a data-aware declarative PM technique that is used to model a malware behavior exploiting a higher number of properties and relations among the system calls gathered from system call traces. Moreover, the proposed approach should be very useful to support the automatic verification and the new application approval process usually performed by the commercial mobile application stores. The paper is organized into 8 sections. Section II describes the background. Section III reports the related work discussion. In Section IV the proposed approach is described. Section V reports the evaluation consisting to apply the proposed approach a dataset of 8 malware families and 1200 malicious and trusted applications. Section VI discusses the obtained results while in Section VII, the threats to validity are discussed. Finally, section VIII provides some conclusive remarks and future work discussion.

II. Background

A. Mobile Malware Families

A malware family represents a set of malware having similar behavior and properties. The specific knowledge of the structural and behavioral properties of a malware family is an effective support for malware detection since new malwares are usually obtained from existing ones through evolutionary relationships [11]. The list of malware families studied in this work are reported in Table I. The table, starting from the third column, shows a brief description of the malware family, its installation type (IT) and the known classes of events activating the malware (AE). More precisely, IT refers to the way the malicious payload is installed [27] and can assume the values 'r' (repackaging), 's' (standalone), and 'u' (update attack). Activating Events belong to different classes

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

(e.g., for the Android platform they are BOOT, BATT, SMS, SYS, NET, CALL).

The correspondence between a malware family and the classes of its activating events is reported in Table I whereas the list of considered activating events along with their classes, for the Android platform, is reported in Table II and is further discussed in [10, 4].

B. Multi-perspective Declare

The proposed approach is based on MP-Declare, a multi-perspective version of Declare language [17]. Declare is a process modeling language that describes a process as a set of constraints that must be satisfied throughout its execution. The activities sequence is specified by the constraints since all that does not violate them is allowed during the process execution.

Declare constraints are concrete instantiations of templates defining parameterized classes of properties.

The constraints inherit the graphical representation and semantics from their templates. Common semantics are based on LTL logic [5] allowing making processes verifiable and executable throughout finite traces.

A multi-perspective version of Declare is MP-Declare [12, 13]. It is based on the Metric First-Order Linear Temporal Logic (MFOTL). This logic allows to enrich the existing Declare language with a time and data perspective.

The MP-Declare semantics are shown in Table III. It is based on the concept of payload of an event. Consider, for example, the activity *receive_call* executed at a timestamp t_s having a *caller* attribute equal to *Joe*. In this case, we say that {caller=Joe, timestamp=t_s} is the *payload* of *receive_call*. Two kind of parameters are defined for MP-Declare activities: activation and target parameters. Looking for example to the response constraint between the activities *receive call* and *filter_call*, it means that the activity *receive call* is always eventually followed by activity *filter_call*. With respect to Declare, a timed semantics is here introduced corresponding to two additional conditions on data (for example an *activation condition* φ_a and a *correlation condition* φ_c). The activation condition is a relation among the variables (event logs global attributes) that must be valid when the activation occurs. If the activation condition is not valid, the constraint is not activated. Looking at the response template reported in Table III, the activation condition is represented as $p_A(x) \wedge r_a(x)$. It means that when A occurs with payload x, the relation r_a over x must be valid. So, considering our small example, when the activity *receive_call* occurs and the *(caller)* is contained in the black list, the activity *filter_call* must eventually follow. Viceversa, if the activity *receive_call* occurs and the *(caller)* is not contained in the black list, the constraint is not activated.

Moreover, we consider the correlation condition as a relation that must be valid when the target occurs. It is formalized as $p_B(y) \wedge r_c(x, y)$, where r_c represents a relation among the variables corresponding to the global attributes in the event log but, in this case, relating the payload of A and the payload of B.

In this study we use MP-Declare [13] to discover constraints that relate an activation and target attributes.

III. Related Work

Several studies about malware detection are recently presented. For briefly, here we focus on dynamic methods while a further discussion including also static methods is proposed in [4]. Dynamic methods consist to analyze a malicious application during its execution on a real device or in a controlled environment. A great number of these approaches are based on the system calls analysis [9, 18, 20, 7, 19, 24] and use customized Android kernel. For example, the approach proposed in [9] allows mining the read/write operations system to detect malicious code. The authors use a real device equipped with a customized kernel and the evaluation is performed on a synthetic dataset. Differently, our proposed method does not require a customized kernel, with consequently improved applicability for the user.

Some other studies [23, 2] propose feature network-based approaches to perform malware detection. However, these studies are more focused on analyze data-leakage. Another limit of the above approaches is that they require to recompile the kernel. This makes them time intensive and resource-consuming, with respect to our proposed approach. A recent approach also proposes the combination of both static and dynamic features to mine the malware behavior [26]. This work propoposes the abstraction of feature vectors with the procedure of decompiling the APK file. This makes this approach more time and resource consuming with respect the proposed one. Android malware detection is also the focus of the study introduced in [6]. The authors analyze the sequences of system calls to find a fingerprint of the malware obtaining promising results (the accuracy is 97%).

The concept of malware fingerprint is also presented in [4]. With respect to [6], this approach proposes the adoption of process mining techniques to obtain a model (it is called System Calls Execution Fingerprint (SEF)) of the malware behavior from a set of traces collected during the running of trusted and malware applications. The classification is performed using the distance among SEFs to discriminate malware and trusted applications on the base of their similarities. The study presented in [4] represents the starting point for our proposal. Here the concept of SEF is updated and the Data-aware System Calls Execution Fingerprint (DSEF) is introduced. The basic idea is that the addition of data attributes analysis can improve the malware detection 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. In this study, we also compare obtained results with the ones previously obtained in [4] to verify if the introduction of data attributes study can effectively improve the malware detection performances.

IV. Approach

The proposed approach uses a data-aware process mining technique to mine system call traces (logs) of an Android

TABLE I
A LIST OF SOME COMMON MALWARE FAMILIES.

Family	Samples	Description	IT	Class of Activating Event
Airpush	7845	It displays unwanted ads without any consent	r	BOOT, BATT, SMS
DroidKungFu	3943	It installs a backdoor supporting the access of attackers to the smartphone	r	BOOT, BATT, SYS
Dowgin	3894	It displays for displaying third-party advertising content and silently leaks or captures sensitive information	r	BOOT,SMS,SYS
Fusob	2205	It encrypts data and then orders victims to pay to unlock the device	r,u	BOOT, SMS, NET, BATT
FakeInst	2178	it sends SMS messages to premium-rate numbers or services	r,u	BOOT, CALL
Mecor	1822	It is a Trojan-Spy	s	BOOT
Youmi	1302	It monitors and captures user behavior and floods the device with unsolicited pop-up advertisements	r,u	BOOT, SMS, CALL
Kuguo	1201	it redirects the victim to malicious websites and continually display popup ads	r	BOOT, SMS, NET, BATT

TABLE II
SYSTEM EVENTS ACTIVATING THE MALICIOUS BEHAVIOR.

Class	Activating Event (AE)	Description
BOOT	*BOOT_COMPLETED*	Able to catch the boot completed
CALL	*PHONE_STATE*	Incoming call
	NEW_OUTGOING_CALL	Outgoing call
SYS	*INPUT_METHOD_CHANGED*	An input method has been changed
	USER_PRESENT	User unlocks the device
	SIG_STR	Listening to signal strength when the phone sleeps
	SIM_FULL	The SIM storage for SMS messages is full
SMS	*SMS_RECEIVED*	Reception of SMS
	WAP_PUSH_RECEIVED	A WAP PUSH message has been received.
BATT	*POWER_CONNECTED*	Battery status in charging
	POWER_DISCONNECTED	Battery status discharging
	BATTERY_OKAY	Battery full charged
	BATTERY_LOW	Battery status at 50%
	BATTERY_EMPTY	Battery status at 0%
	BATTERY_CHANGED	Battery status changed
NET	*PICK_WIFI_WORK*	Start the Wifi
	CONNECTIVITY_CHANGE	The state of connection has been changed.

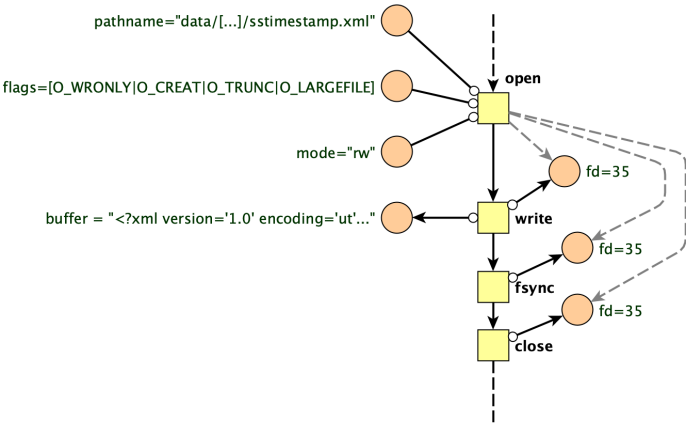

Fig. 1. Excerpt of a syscalls execution trace

TABLE III
SEMANTICS FOR MP-DECLARE CONSTRAINTS.

Template	MFOTL Semantics
existence	$\mathbf{F}_I(A \wedge \exists x.\varphi_a(x))$
absence	$\neg\mathbf{F}_I(A \wedge \exists x.\varphi_a(x))$
choice	$\mathbf{F}_I(A \wedge \exists x.\varphi_a(x)) \vee \mathbf{F}_I(B \wedge \exists x.\varphi_a(x))$
exclusive choice	$(\mathbf{F}_I(A \wedge \exists x.\varphi_a(x)) \vee \mathbf{F}_I(B \wedge \exists x.\varphi_a(x))) \wedge \neg(\mathbf{F}_I(A \wedge \exists x.\varphi_a(x)) \wedge \mathbf{F}_I(B \wedge \exists x.\varphi_a(x)))$
responded existence	$\mathbf{G}(\forall x.((A \wedge \varphi_a(x)) \rightarrow (\mathbf{O}_I(B \wedge \exists y.\varphi_c(x,y)) \vee \mathbf{F}_I(B \wedge \exists y.\varphi_c(x,y)))))$
response	$\mathbf{G}(\forall x.((A \wedge \varphi_a(x)) \rightarrow \mathbf{F}_I(B \wedge \exists y.\varphi_c(x,y))))$
alternate response	$\mathbf{G}(\forall x.((A \wedge \varphi_a(x)) \rightarrow \mathbf{X}(\neg(A \wedge \varphi_a(x))\mathbf{U}_I(B \wedge \exists y.\varphi_c(x,y)))))$
chain response	$\mathbf{G}(\forall x.((A \wedge \varphi_a(x)) \rightarrow \mathbf{X}_I(B \wedge \exists y.\varphi_c(x,y))))$
precedence	$\mathbf{G}(\forall x.((B \wedge \varphi_a(x)) \rightarrow \mathbf{O}_I(A \wedge \exists y.\varphi_c(x,y))))$
alternate precedence	$\mathbf{G}(\forall x.((B \wedge \varphi_a(x)) \rightarrow \mathbf{Y}(\neg(B \wedge \varphi_a(x))\mathbf{S}_I(A \wedge \exists y.\varphi_c(x,y)))))$
chain precedence	$\mathbf{G}(\forall x.((B \wedge \varphi_a(x)) \rightarrow \mathbf{Y}_I(A \wedge \exists y.\varphi_c(x,y))))$
not responded existence	$\mathbf{G}(\forall x.((A \wedge \varphi_a(x)) \rightarrow \neg(\mathbf{O}_I(B \wedge \exists y.\varphi_c(x,y)) \vee \mathbf{F}_I(B \wedge \exists y.\varphi_c(x,y)))))$
not response	$\mathbf{G}(\forall x.((A \wedge \varphi_a(x)) \rightarrow \neg\mathbf{F}_I(B \wedge \exists y.\varphi_c(x,y))))$
not precedence	$\mathbf{G}(\forall x.((B \wedge \varphi_a(x)) \rightarrow \neg\mathbf{O}_I(A \wedge \exists y.\varphi_c(x,y))))$
not chain response	$\mathbf{G}(\forall x.((A \wedge \varphi_a(x)) \rightarrow \neg\mathbf{X}_I(B \wedge \exists y.\varphi_c(x,y))))$
not chain precedence	$\mathbf{G}(\forall x.((B \wedge \varphi_a(x)) \rightarrow \neg\mathbf{Y}_I(A \wedge \exists y.\varphi_c(x,y))))$

application produced in response to the set of AE reported in Table II. The obtained models represent a fingerprint of the behavior of an application regarding executed system calls. It is used to classify malware or trusted applications by evaluating the membership of an infected application to a given family. Specifically, the approach is based on the analysis of system call traces (i.e., the logs) captured from the execution of a mobile application. Such logs can be used to characterize the behaviour of an application (including the possible malware behavior). We assume that the malicious behavior is usually triggered by system events broadcasted by the operating system, as also highlighted in some studies [4, 14].

To better clarify the approach we refer to the running example shown in Figure 1 and Figure 2. The starting point is the parsing of a set of syscall traces. Figure 2 reports an example of process to which the trace of Figure 1 is conformant. Figure

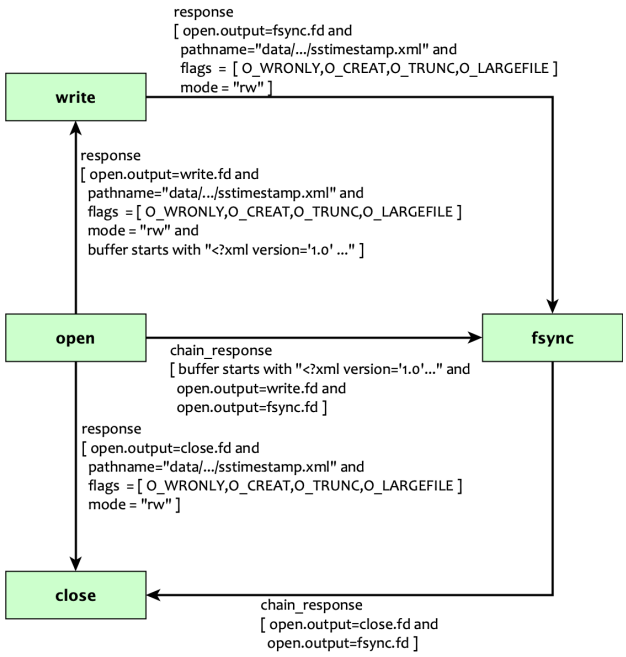

Fig. 2. A process to which the trace in Figure 1 is conformant

1 shows a small sequence of syscall execution events generated by an application that opens a file, write some data and close it. The syscall executions in the trace are analyzed to generate the events retaining information on both parameters and return values. Each syscall is associated to 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.

The behavioral model extracted from syscalls logs is called DSEF — Data-aware Syscalls Execution Fingerprint. With respect to the original SEF notion [4] which is based on Declare language, DSEF is represented using the MP-Declare notation [13] in order to represent correlation conditions among syscall taking also into account data parameters obtained from the application execution traces.

A. Data-aware Syscalls Execution Fingerprint (DSEF)

In this section we report the definitions for the DSEF of a mobile application and the DSEF of a malware family.

Definition 1 (MP-Declare model)**.** *The MP-Declare model associated to a set T of system call traces is defined as:*

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

where $C_h = (S_A, S_T, P)$ is a constraint (unary or binary) specifying a condition P as reported in the second column of Table III. 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.*

MP-Declare models are the building blocks for the definition of DSEF for both applications and malware families. For the following definitions, let be:

- A, a set of m applications infected with the malware family M;
- E, the set of the n system events sent to each application.

Definition 2 (DSEF of an application a)**.** *The $DSEF_a$ of the application $a \in A$ is the set of the MP-declare models, defined as:*

$$DSEF_a = \{MPD_{a_1}, \ldots, MPD_{a_n}\}$$

where:

- *MPD_{a_j} is the MP-Declare model of the application a for the system event j, mined from the set of traces $\{t_{j_1}, \ldots, t_{j_r}\}$.*
- *n is the number of the system events sent to an application;*
- *r is the number of performed runs of the application a;*
- *t_{j_k} is the k-th system calls trace generated by the application a, in response to the event j, with $k \in [1, r]$;*

Definition 3 (DSEF of a malware family M)**.** *The DSEF of a malware family M can be defined as the set:*

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

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 $i \in [1, m], j \in [1, n]$;*

Observe that the kind and number of traces used for Definition 2 and Definition 3 are different. For the application, the DSEF is obtained, for each system event $e \in E$, by a set of r traces executing the same application several times. In this case, we are mining the models, for each event e_j, of the application and hence several traces are required to model its behavior using MP-Declare rules (the application behavior, that is very similar from trace to trace since the application is the same, is reflected as a proper set of MP-Declare constraints).

Differently, the DSEF of a malware M is considered as the model of the malicious behaviour that is shared by an entire set of different applications infected by the same malware M. In this case a single run for each application of the set it is sufficient to mine the rules characterizing the common (i.e. malicious) behavior shared by all applications in A. Specifically, in this case each trace, generated by a different application, contains different behaviors that do not determine any constraints in the mined process. Only the common parts, i.e. those generated by the malware portion of the application, determine a set of constraints that can be considered a almost unique fingerprint of the malware.

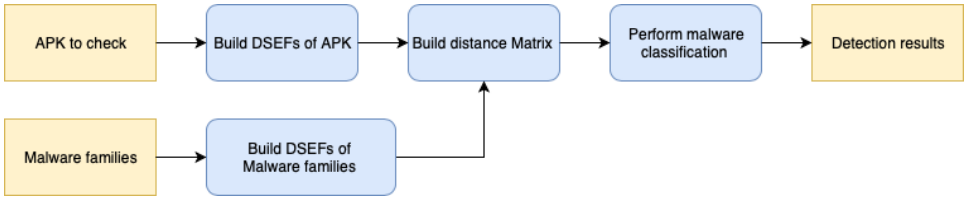

Fig. 3. The malware detection process.

B. Distance among DSEFs

To discriminate if an application a is infected with a malware M we need to compare the MP-Declare model of the application a obtained for an activating event with the one of the malware M for the same event.

To accomplish this we need to define a distance between MP-Declare process definitions.

Referring to the definitions 2 and 3, we can define the distance $\sigma(MPD_i, MPD_j)$ between two MP-Declare models MPD_i and MPD_j as follows:

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

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;

Specifically, the tree edit distance among the expression trees P_{ih} and P_{jh} has been evaluated using the approach proposed in [16] and normalized using the approach suggested in [25].

The defined distance represents a measure of similarity among two MP-Declare models: distance of models having same constraints with the same correlation conditions is equal to zero, viceversa models with different constraints or with completely different correlation conditions have a maximum distance equals to one.

Finally, the distance between two DSEFs, namely A and B, can be defined as:

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

that is the average of the distances between the models A and B of all the activating events. The distance among the DSEF of an application can be effectively used to perform malware detection by evaluating its similarity with the DSEFs of known malware families.

C. Malware detection

The proposed malware detection approach is summarized in Figure 3. For each checked Android application (APK) and for all the considered malware families, the corresponding DSEF is built. Successively, the matrix of the DSEF distances between the application and the malware families is computed. The matrix is the input of a malware classifier allowing to indicate if the APK is infected with one or more considered malware family. Several classification approaches have been implemented and tested using Scikit-learn and Keras toolkits[1].

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

Further details about the process for computing the DSEF of an APK are provided in Figure 4 and discussed in the remainder of the section. The process starts with the *traces extraction* step allowing to capture, for the considered APK, the syscalls traces generated in response to the AE reported in Table II. The traces are collected in a textual format. In this step, an Android device emulator[2] is used. Here the considered APK is installed and started. During the APK running, a system event is sent to the emulator and the subsequent system calls are captured (when the APK state became stable the syscall capture is stopped). Each event is sent more than once in order to have a huge number of syscall traces to generalize the APK behavior. After a syscall trace is captured, the emulator is stopped and its disk is cleaned. However, the APK is reinstalled to each run in order to always have the same initial conditions. All the AE of Table II are scanned using ad hoc build shell scripts [4].

In the *CSV Generation*, the syscall traces collected in the previous step are converted into the CSV log format as requested by the MP-Declare Miner [13]. The conversion step is necessary because the syscall traces are in a textual format and need to be adequately processed. The conversion step retains only useful information including the attributes of the entire session (e.g., the UUID for the application run and the application id) and to each system call occurrence (e.g., the executed system call, its timestamp, its ordered list of arguments, and the process id requesting the system call). This information is used to generate the event payload for each log event and is needed during the following process mining step to generate correlation conditions for the subsequently mined MP-declare process. For data correlation, we adopted a reference-based correlation approach as implemented in [4] that defines a correlation function exploiting the process id

[1] See https://scikit-learn.org/ and https://keras.io for reference.
[2] https://developer.android.com/studio/run/emulator.html

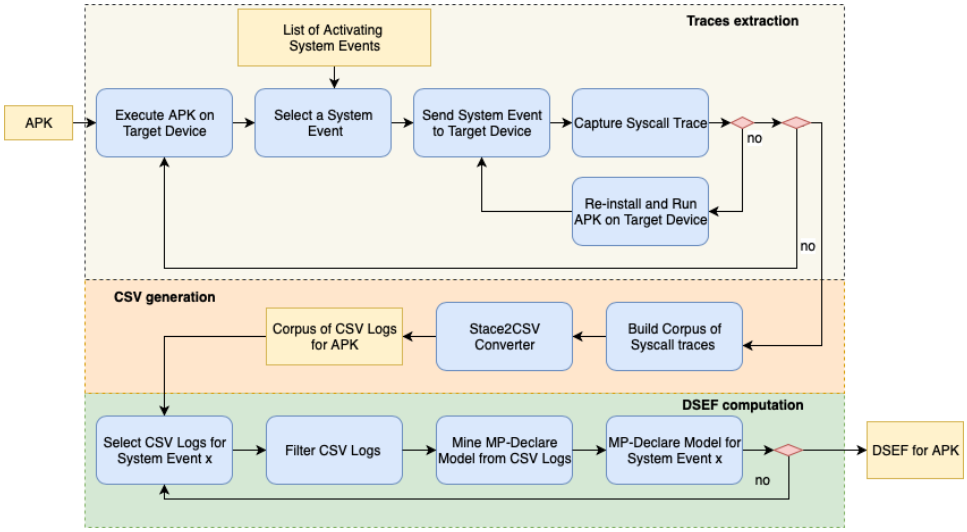

Fig. 4. The process for computing the DSEF of an application.

and the event timestamps attributes.

The *DSEF computation* step starts from the CSV logs generated in the previous step and builds the DSEF of the APK. The CSV logs are firstly filtered (e.g., basing on the Gaussian distribution of the sizes of the logs, the logs that are outside the 80th percentiles are removed since they have a very high probability to be not correct - for example when generated by an application that is not working as intended and hence is terminated by the OS - or not relevant). The relevant logs are then mined using the MP-Declare tool obtaining an MP-Declare model that represents the DSEF of the APK. Each model can be seen as a set of MP-Declare constraints allowing to describe the relationships among system call holding in all the traces, for a specific activating event among those reported in Table II. The set of mined models (one for each activating event) represents the DSEF of the application.

The process described in Figure 4 is similar to the one executed to compute the DSEF of a malware family. Concerning trace extraction process, in this case the initial dataset is composed of different applications (several APKs) all infected with the same malware of the family under study. The results are a set of syscall traces generated from malicious applications belonging to a specific malware family and stimulated with one of the considered system events. Since the applications are different, the only common part of syscall traces is the one generated by the shared malicious portion. For this reason, the mined model contains a set of constraints that characterize the behavior of the shared malware part while discards the ones that are specific to the various applications (since they are different from trace to trace). The obtained set of mined models (one for each activating event) represents the DSEF of the considered malware family.

D. Building of the neural network malware classifier

The malware classifier, used to check if an application is infected or not, is trained on the distances between the application DSEF and the malware models DSEFs.

The classification process starts by taking as a input the DSEFs of Malware families, the DSEFs of training applications and the DSEFs of the testing applications. Respectively, the DSEFs of training applications and the DSEFs of the testing applications are computed. The training evaluates the dissimilarity matrices between DSEFs of training applications and malware family and is repeated (increasing the number of training applications) until the best values for precision and recall are obtained (using the set of testing applications for assessment).

V. The Experiment

A. Dataset description

An overview of the dataset used to evaluate the proposed approach is shown in Table I. The dataset includes a selected set of malware and trusted applications. The malware applications are downloaded from known datasets like Genoma [28], Drebin [3] and ADM [22]. The trusted applications, instead, are selected among the most downloaded applications of the Google Play store. Both trusted and malware applications have different application domains (i.e., education, traveling, internet navigation, lifestyle, news, productivity, business, communication, health) and the considered malware applications differ also for their malicious behavior. The label of trusted or malware is assigned by each considered applications directly from the dataset producers. Moreover, we performed an additional analysis to confirm that the application labels are correctly assigned. For the trusted applications the labels are checked by using the Google Bouncer [15]. For the malware applications, the check is performed using 57 anti-malware (running on VirusTotal service[3]). In this check, we filtered out from the infected applications list all the applications that were

[3]https://www.virustotal.com/gui/home/upload

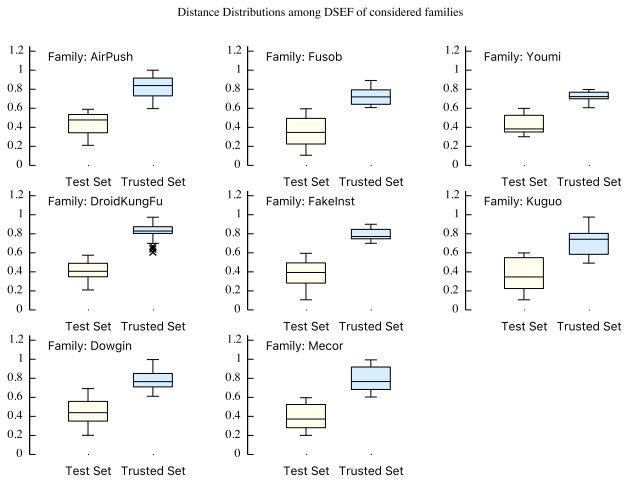

Fig. 5. Distance Distributions among DSEF of considered families

labeled as not infected by at least five antimalware over the 57 ones.

B. Experiment setting

The experiment consists to evaluate the performances of two different classifiers. The first single multinomial classifier is trained by using samples of all the eight families of Table I. It allows classifying each analyzed application as infected or not infected. Moreover, the infected applications are classified as belonging to one of the eight considered families. The second classifier is a binary classifier able to identify for each malware family of Table I if the analyzed application is infected with that malware. The classification has been always performed using different algorithms: DNN (Deep Neural Network) 5L, CNN (Computational Neural Network) 5L, and CNN 6L. Classifiers were trained using the distance matrices among DSEFs of applications and reference malware families.

The classifiers' performances are evaluated using the Precision and Recall. Precision is the fraction of the correctly classified samples to all samples while Recall is the fraction of correctly classified instances to all correct samples. Finally, we also evaluate the ROC Area, that is the probability of correctly classifying a random pair of occurrences as infected or not.

VI. Discussion of Results

Table IV shows the obtained results. The second column reports the name of the adopted algorithms (they are colored in black), while the columns 3-5 report, respectively, the values of Precision, Recall and ROC AUC. Concerning the results, the first row (All families) shows the results obtained by the multinomial classifier while the rows from two (AirPush) to nine (Youmi) report the performance of the binary classifier for each considered malware family.

We can observe that for all the considered algorithms, we obtained very good and similar results. The best performances

Family	Algorithm	Precision	Recall	ROC AUC
All families	DSEF-CNN 6L	0.965	0.958	0.95
	DSEF-DNN 5L	0.953	0.978	0.94
	DSEF-CNN 5L	0.962	0.958	0.95
	SEF-DNN 5L	0.922	0.931	0.91
AirPush	DSEF-CNN 6L	0.935	0.932	0.92
	DSEF-DNN 5L	0.933	0.934	0.93
	DSEF-CNN 5L	0.942	0.942	0.93
	SEF-DNN 5L	0.906	0.921	0.90
DroidKungFu	DSEF-CNN 6L	0.975	0.979	0.97
	DSEF-DNN 5L	0.969	0.978	0.96
	DSEF-CNN 5L	0.972	0.981	0.97
	SEF-DNN 5L	0.956	0.951	0.95
Dowgin	DSEF-CNN 6L	0.925	0.918	0.92
	DSEF-DNN 5L	0.913	0.908	0.90
	DSEF-CNN 5L	0.922	0.928	0.92
	SEF-DNN 5L	0.866	0.821	0.85
Fusob	DSEF-CNN 6L	0.961	0.953	0.95
	DSEF-DNN 5L	0.953	0.955	0.93
	DSEF-CNN 5L	0.958	0.952	0.94
	SEF-DNN 5L	0.906	0.921	0.91
FakeInst	DSEF-CNN 6L	0.945	0.965	0.94
	DSEF-DNN 5L	0.933	0.953	0.94
	DSEF-CNN 5L	0.942	0.964	0.95
	SEF-DNN 5L	0.926	0.931	0.90
Mecor	DSEF-CNN 6L	0.920	0.912	0.91
	DSEF-DNN 5L	0.921	0.932	0.91
	DSEF-CNN 5L	0.922	0.938	0.92
	SEF-DNN 5L	0.896	0.881	0.89
Kuguo	DSEF-CNN 6L	0.945	0.952	0.94
	DSEF-DNN 5L	0.923	0.918	0.92
	DSEF-CNN 5L	0.932	0.937	0.93
	SEF-DNN 5L	0.796	0.801	0.80
Youmi	DSEF-CNN 6L	0.915	0.941	0.92
	DSEF-DNN 5L	0.903	0.908	0.90
	DSEF-CNN 5L	0.922	0.934	0.93
	SEF-DNN 5L	0.862	0.913	0.89

TABLE IV
PERFORMANCE OF THE CLASSIFIERS.

are obtained using the CNN 6L algorithm on the DroidKungFu family (precision is equal to 0.975 and recall is equal to 0.979). All the classifiers and all the algorithms always give recall greater than 0.908. Moreover, for each considered malware family, Figure 5 shows a statistical comparison among distances of the trusted and infected applications. The boxplots reveal that the distances are always well separated and can be effectively used as a discriminant signature of the malicious behavior.

Finally, we perform a comparison between the results obtained using a similar approach proposed in [4]. As explained in Section III, the proposed approach updates the concept of SEF [4] in DSEF. To evaluate the different performances of these two approaches, we replicate the experiments described in the previous subsection using the SEF distance as defined in [4] in the place of DSEF distance. Table IV reports (they are highlighted in blue) for each considered family the performance of classification performing by using a DNN 5L algorithm on classifiers trained using SEF distances. The table shows that in all the cases (with the exception of the DSEF-DNN 5L in Youmi family), the proposed approach ensures better performances.

VII. Threats to validity

Looking to the *construct validity*, a possible threat is that the process of syscall traces extraction is affected by some

possible imprecisions. However, the script for trace capturing allows recording the trace when an AE and stops when the application reaches a new stable state. This automatic cut may cause the collection of incomplete traces. To mitigate this problem, we perform a trace validation step consisting to filter the incomplete and incorrect traces.

Another construct validity threat is the assumption that the applications labeled as "malicious" are really infected. To mitigate this risk each application is checked on a consistent number of antimalware and the applications labeled as "infected" are the only ones that are recognized by a number of antimalware greater than five.

Finally, looking to the *external validity* threats and the generalization of the obtained results, we have to highlight that our experimentation requires an extension. To this aim, new malware families and applications will be evaluated in the future.

VIII. Conclusions and future work

We presented an extension of an approach, introduced in [4], consisting in using a process mining technique for the dynamic analysis of the system call traces gathered from an application. In detail, the approach identifies similarities and recurring patterns in the system call traces generated by a set of malicious mobile applications. The extension proposed in this work allows analyzing system call traces within their data attributes building a behavioral model called DSEF. We evaluated the approach on a data set of more than 1200 infected applications from eight malware families. Overall, the validation shows the effectiveness of the approach in malware detection and the best performance of DSEF models to represent malware behaviors in comparison to the SEF models (proposed in [4]). A possible avenue for future work is to carry out further experimentations with a greater number of malware families and applications to determine if the obtained results could be generalized.

References

[1] Mobile threat report. https://www.f-secure.com/documents/996508/1030743/Threat_Report_H1_2014.pdf, last visit 26 February 2016.

[2] A. Arora, S. Garg, and S. K. Peddoju. Malware detection using network traffic analysis in android based mobile devices. In *Next Generation Mobile Apps, Services and Technologies (NGMAST), 2014 Eighth International Conference on*, pages 66–71, Sept 2014.

[3] Daniel Arp, Michael Spreitzenbarth, Malte Huebner, Hugo Gascon, and Konrad Rieck. Drebin: Efficient and explainable detection of android malware in your pocket. In *Proceedings of 21th Annual Network and Distributed System Security Symposium (NDSS)*, 2014.

[4] Mario Luca Bernardi, Marta Cimitile, Damiano Distante, Fabio Martinelli, and Francesco Mercaldo. Dynamic malware detection and phylogeny analysis using process mining. *International Journal of Information Security*, 18(3):257–284, Jun 2019.

[5] Mario Luca Bernardi, Marta Cimitile, Chiara Di Francescomarino, and Fabrizio Maria 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*, pages 281–295, 2014.

[6] Gerardo Canfora, Eric Medvet, Francesco Mercaldo, and Corrado Aaron Visaggio. Detecting android malware using sequences of system calls. In *Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile*, DeMobile 2015, pages 13–20, New York, NY, USA, 2015. ACM.

[7] Takamasa Isohara, Keisuke Takemori, and Ayumu Kubota. Kernel-based behavior analysis for android malware detection. In *Proceedings of the 2011 Seventh International Conference on Computational Intelligence and Security*, CIS '11, pages 1011–1015, Washington, DC, USA, 2011. IEEE Computer Society.

[8] Jiyong Jang, David Brumley, and Shobha Venkataraman. Bitshred: Feature hashing malware for scalable triage and semantic analysis. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 309–320, New York, NY, USA, 2011. ACM.

[9] Youn-sik Jeong, Hwan-taek Lee, Seong-je Cho, Sangchul Han, and Minkyu Park. A kernel-based monitoring approach for analyzing malicious behavior on android. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, SAC '14, pages 1737–1738, New York, NY, USA, 2014. ACM.

[10] X. Jiang and Y. Zhou. *Android Malware*. SpringerBriefs in Computer Science. Springer New York, 2013.

[11] Md Enamul Karim, Andrew Walenstein, Arun Lakhotia, and Laxmi Parida. Malware phylogeny generation using permutations of code. *Journal in Computer Virology*, 1(1-2):13–23, 2005.

[12] Volodymyr Leno, Marlon Dumas, and Fabrizio Maria Maggi. Correlating activation and target conditions in data-aware declarative process discovery. In Mathias Weske, Marco Montali, Ingo Weber, and Jan vom Brocke, editors, *Business Process Management*, pages 176–193, Cham, 2018. Springer International Publishing.

[13] Volodymyr Leno, Marlon Dumas, Fabrizio Maria Maggi, Marcello La Rosa, and Artem Polyvyanyy. Automated discovery of declarative process models with correlated data conditions. *Information Systems*, 89:101482, 2020.

[14] Guozhu Meng, Ruitao Feng, Guangdong Bai, Kai Chen, and Yang Liu. Droidecho: an in-depth dissection of malicious behaviors in android applications. *Cybersecurity*, 1(1):4, Jun 2018.

[15] J. Oberheide and C. Mille. Dissecting the android bouncer. In *SummerCon*, 2012.

[16] Mateusz Pawlik and Nikolaus Augsten. Tree edit distance: Robust and memory-efficient. *Inf. Syst.*, 56:157–173, 2016.

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

[18] A. Reina, A. Fattori, and L. Cavallaro. A system call-centric analysis and stimulation technique to automatically reconstruct android malware behaviors. In *Proceedings of EuroSec*, 2013.

[19] A.-D. Schmidt, H.-G. Schmidt, J. Clausen, K. A. Yuksel, O. Kiraz, A. Camtepe, and S. Albayrak. Enhancing security of linux-based android devices. In *Proceedings of 15th International Linux Kongress*, 2008.

[20] F. Tchakount and P. Dayang. System calls analysis of malwares on android. In *International Journal of Science and Tecnology (IJST) Volume, 2 No. 9*, 2013.

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

[22] Fengguo Wei, Yuping Li, Sankardas Roy, Xinming Ou, and Wu Zhou. Deep ground truth analysis of current android malware. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA'17)*, pages 252–276, Bonn, Germany, 2017. Springer.

[23] Te-En Wei, Ching-Hao Mao, Albert B. Jeng, Hahn-Ming Lee, Horng-Tzer Wang, and Dong-Jie Wu. Android malware detection via a latent network behavior analysis. In *Proceedings of the 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, TRUSTCOM '12, pages 1251–1258, Washington, DC, USA, 2012. IEEE Computer Society.

[24] Lok Kwong Yan and Heng 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*, Security'12, pages 29–29, Berkeley, CA, USA, 2012. USENIX Association.

[25] Chenguang ZHANG Yujian LI. A metric normalization of tree edit distance. *Frontiers of Computer Science*, 5(1):119, 2011.

[26] J. Zhang, F. Zou, and J. Zhu. Android malware detection based on deep learning. In *2018 IEEE 4th International Conference on Computer and Communications (ICCC)*, pages 2190–2194, Dec 2018.

[27] Y. Zhou and X. Jiang. Dissecting android malware: Characterization and evolution. In *2012 IEEE Symposium on Security and Privacy*, pages 95–109, May 2012.

[28] Yajin Zhou and Xuxian Jiang. Dissecting android malware: Characterization and evolution. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 95–109, Washington, DC, USA, 2012. IEEE Computer Society.