Safety Critical Systems Formal Verification using Execution Traces

Fabio Martinelli*, Francesco Mercaldo*, Vittoria Nardone†, Albina Orlando‡, Antonella Santone§, Gigliola Vaglini¶
*Istituto di Informatica e Telematica, Consiglio Nazionale delle Ricerche, Pisa, Italy
{fabio.martinelli, francesco.mercaldo}@iit.cnr.it
†Department of Engineering, University of Sannio, Benevento, Italy
vnardone@unisannio.it
‡Istituto per le Applicazioni del Calcolo "M. Picone", Consiglio Nazionale delle Ricerche, Napoli, Italy
a.orlando@iac.cnr.it
§Department of Bioscience and Territory, University of Molise, Pesche (IS), Italy
antonella.santone@unimol.it
¶Department of Information Engineering, University of Pisa, Pisa, Italy
gigliola.vaglini@unipi.it

***Abstract*—Data breaches usually involve financial information such as credit card or bank details. Automated formal verification of safety critical systems has been mostly focused on analysing high-level abstract models which, however, are significantly different from real implementations written in programming languages. In this paper we propose a technique that links model checking verification closer to real implementations, with particular regard to financial environment, in order to identify possible causes of data breaches. A formal model starting from execution traces is retrieved. Thus, the discovered model can be analysed to verify it respects the defined properties. A real safety critical system has been used as a case study to evaluate the proposed methodology.**

I. Introduction and Related Work

In the spring of 2016, the SecurityScorecard security company analyzed 7,111 financial institutions in order to find existing vulnerabilities within investment banks, asset management firms, and major commercial banks to determine the strongest and weakest security standards based on security hygiene and security reaction time compared to their peers. The emerging scenario is that cyber attacks on corporations and banks are really accelerating. This scenario calls for new approaches in order to push banks and financial companies to make investment in cyber security. Many insurance companies aware of the risk of cyber threats, begin to offer insurance policies for risks arising from cyber threats. This is the reason why high assurance about their correctness is highly desirable also from a cyber insurance point of view. Due to the inherent complexity, developing safety critical systems demands rigorous, mathematically based methods for reasoning about their correctness. For example, the application of formal methods, like model checking, is significant. Model checking [1] applies to a formal description of the system behaviour as a labeled transition system (LTS) and to a temporal-logic formula representing the requirement to be verified. Nevertheless, a gap still exists between these models and a real-world system and its execution. In fact, in the context of software verification, a non-trivial problem to be faced is the actual availability of an LTS description of the system. In the best cases, such models might have been obtained in the design phase of the system development process. Usually, UML dynamic diagrams are used to attempt a translation to LTS. But, even then, design models are not always consistently updated in the implementation activities.

In this preliminary paper, we propose a method, not fully validated, that reduces the gap between models and implementations, trying to extract model from execution traces. The first step extracts execution traces from a software system. The code is instrumented with instructions to generate logs containing just the necessary events for the verification of a specific set of properties. Then, using the "Mine Transition System" plugin in ProM[1] we obtain a labelled transition system, that can be easily used to prove properties. Differently from [2], our aim is to use an existing model checker such as CADP [3], which is a mature verification tool. The most widely used model checkers are by now extremely sophisticated programs that have been crafted over many years by experts in the specific techniques employed by the tool: any re-implementation of similar tools could likely yield worst performance. A real case study in the financial field has been considered as a proof of concept of our methodology. Using synthetic data obtained from bank transactions, a formal model has been automatically constructed.

Traditionally, models used in model checking are manually constructed. However, such model-construction can be extremely time-consuming, or even infeasible in the

[1] http://www.promtools.org/doku.php

case of insufficient documentation for an existing system, thus there is an increasing interest in model learning for formal verification. For example, in [4] a learning algorithm has been proposed for probabilistic systems. A limit of this approach is that the algorithm might not converge to the good model in general. It is ensured that only with sufficiently big sample set of traces, a given property will hold on the original and the learned model with the same probability. Moreover, [4] assume that learning is based on data consisting of many independent finite execution runs, each starting in a distinguished, unique initial state of the system. In many situations, it will be difficult or impossible to obtain data of this kind: we may not be able to run the system under laboratory conditions where we are free to restart it any number of times, nor may we be able to reset the system to a well-defined unique initial state. For deterministic system models it has been suggested to use Angluin's [5] approach to learning deterministic finite automata. Many modeling approaches have been proposed. They differ in the kind of properties they help reasoning about, and in the level of precision or formality of the results one may obtain through them. In [6], the authors focus on models that may be used to reason about non-functional properties of the software-to-be. In this case, models are heavily dependent on parameters that must be provided a-priori by domain experts or are extracted by other similar systems.

The paper is organized as follows. In Section II, after having recalled preliminary concepts of the selective mu-calculus temporal logic, the proposed method is described. The approach has been applied to a simple real case study in Section III. Finally, concluding remarks are given in Section IV.

II. The Method

We assume the reader familiar with the basic concepts of model checking and of the selective mu-calculus logic. For details the reader can refer to [1], [7]. We have chosen to use the selective mu-calculus logic since, as demonstrated in [8], it is useful to combat the state explosion problem, typical of the model checking technique and equivalence checking [9]. We propose a formal based methodology to verify critical systems starting from execution traces. Figure 1 shows the work-flow of the complete methodology. The methodology is mainly based on the creation of a formal model from traces. It consists of three steps: **First step:** The first step, see Figure 1 (1), consists of obtaining execution traces from programs. The execution traces, recovered during a software system execution, include both static and dynamic information. Static information regards, for instance, class structure in terms of methods and fields. Dynamic information refers to method calls, field access in read or write mode and synchronization on objects. For obtaining execution traces, software systems need to be instrumented adding some instructions called *instrumentation code*. Different executions of the same software systems will generally produce different traces depending on the execution order of the different instructions due to thread scheduling or other software external context. Text files are used for storing the execution traces. Then, we generate XES-based Event Streams from execution traces. The process aims to clean, filter and convert the execution traces collected in textual format in the previous process into eXtensible Event Stream (XES)-compliant log format (IEEE XML-based standard for event logs). During this conversion, all the unnecessary information is filtered out. **Second step:** The second step, see Figure 1 (2), consists in creating the model from the traces. For the purpose of this work, we have chosen the XES format since we use Process Mining Workbench (ProM)[2]. Process Mining research is concerned with the extraction of knowledge about a process from its process execution logs. From the XES Event Log, using the "Mine Transition System" plugin in ProM developed by H.M.W. Verbeek, we obtain a labelled transition system, where the transitions correspond to the events in the log, whereas a state corresponds to a situation in between two events. **Third step:** The third step, see Figure 1 (3), consists in applying the model checking technique. Once the formal model has been retrieved we can easily use it to prove properties using model checking. This step checks the sets of logic properties against the formal model obtained starting from the feature set, as described above. In our approach, we invoke the Construction and Analysis of Distributed Processes (CADP) tool [3] as formal verification environment. In order to apply CADP, we have to convert the transition system obtained into the input format of CADP. This is obtained by parsing the automaton ProM file. Moreover, the property, written in selective mu-calculus, can be equivalently transformed in the syntax of the logic used by the CADP environment.

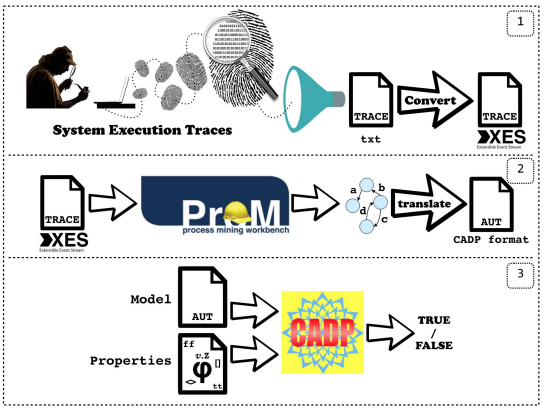

Figure 1: The Work-flow of Our Approach

[2]http://www.promtools.org/

III. Experimental Evaluation

For the evaluation we considered an example of a real system taken from ProM website[3] that describes a realistic transaction process within a banking context. In this preliminary evaluation, we have jumped the first step since we have considered a real case study already developed and available from the repository of the ProM database. The analysed process contains all sort of monetary checks, authority notifications, and logging mechanisms responding to the new degree of responsibility and accountability that current economic environments demand. As stated in [10], "the banking regulation states that serial numbers must be compared with an external database governed by a recognized international authority ("Check Authority Serial Numbers CASN"). In addition, the bank of the case example decided to incorporate two complementary checks to its policy: an internal bank check ("Check Bank Serial Numbers CBSN"), and a check among the databases of the bank consortium this bank belongs to ("Check Inter-Bank Serial Numbers CIBSN"). At a given point, due to technical reasons (i.e., peak hour network congestion, malfunction of the software, deliberated blocking attack, etc.), the external check CASN is no longer performed, contradicting the modeled process, i.e., all the running instances of the process involving cash payment can proceed without the required check".

Adopting our methodology, we formulate this anomaly in the mu-calculus logic formula φ (we have omitted the formal definition for lack of space). The model checker returns "false" when evaluating φ, stating that the anomalous situation is immediately detected, identifying the anomalous subprocess (process cash payment), and eventually taking the necessary countermeasures. The advantage is that it is better to discover the error as soon as possible. Detecting such an error only in forensic analysis performed months after the incident are severe and difficult to recover from. It is worth noting that when a property does not hold, the model checking algorithm generates a counter-example, i.e., an execution trace leading to a state in which the property is violated. This ability to generate counter-examples, which can be exploited to pinpoint the cause of an error, is the main advantage of model checking, as compared to other well-known techniques for software verification, as abstract interpretation-based static analysis.

In the used dataset there are six different scenarios: (i) 2000-all-noise; (ii) 2000-all-nonoise; (iii) 2000-scen1; (iv) 2000-scen2; (v) 10000-all-noise; and (vi) 10000-all-nonoise.

The first item of the string is the number of traces in the XES event stream file. "noise" (resp. "nonoise") specifies if the considered traces are (resp. are not) affected by the noise. Furthermore, there are two files used in [10] which present two possible scenarios: *Serial Number Check* and *Receiver Preliminary Profiling*, i.e., "scen1" and "scen2", respectively. The results of the verification of φ formula are: "True" in 2000-all-nonoise, 2000-scen2 and 1000-all-nonoise, "False" in the other cases. In particular, anomalous situations are detected in the presence of noise which could be due for different reasons, i.e., deliberate blocking attack, peak hour network congestion or malfunction of the software.

[3]http://data.4tu.nl/repository/uuid:c1d1fdbb-72df-470d-9315-d6f97e1d7c7c

Table I: φ Model Size

Model \ Size	States	Transition
2000-all-noise	94,803	96,801
2000-all-nonoise	89,810	91,808
2000-scen1	88,863	90,861
2000-scen2	81,792	83,790
10000-all-noise	480,361	490,359
10000-all-nonoise	434,073	444,071

The sizes of models used in the experimental evaluation are shown in Table I. The size of a model is expressed in terms of states and transitions. In order to better analyze the results obtained by the φ formula, we defined additional eight formulae able to check every single trace belonging to a specific scenario. In particular, these formulae investigate the cause of φ failure. The specified properties are expressed by selective mu-calculus formulae $\varphi_i,\ i \in [1..8]$. We have omitted the formal specification for lack of space. In the following, we only report their informal meaning.

φ_1 checks if all the three actions (CASN, CIBSN and CBSN) are performed.

φ_2 checks if all CASN, CIBSN and CBSN are not performed.

φ_3 checks if CIBSN and CBSN actions are performed and the CASN action is not performed.

φ_4 checks if CASN and CBSN actions are performed and the CIBSN action is not performed.

φ_5 checks if CIBSN and CASN actions are performed and the CBSN action is not performed.

φ_6 checks if CIBSN and CBSN actions are not performed and the CASN action is performed.

φ_7 checks if CBSN and CASN actions are not performed and the CIBSN action is performed.

φ_8 checks if CIBSN and CASN actions are not performed and the CBSN action is performed.

Table II shows the results obtained during the verification of the formulae specified above. In particular, Table II is organized as follow: the above specified formulae are described in the rows, while the scenarios in the columns. Each single model represents a single realistic banking transaction trace. The first four sets have 2000 different traces, so 2000 formal models. The last two have 10000 traces corresponding to 10000 different formal models. The last row is the total number of the analyzed traces resulting true to the formulae. This value is obtained by adding to

each other the values in the corresponding column. The table shows the number of true achieved by every type of analyzed model. As the results shown and according to the "True" values of the φ formula, the files with no noise and the files of second scenario have all the traces of transactions correct, i.e., whenever a client executed a payment in cash, the three required actions have been performed. This result is highlighted by positive values of φ_1 and φ_2 and the values equal to zero achieved by the other formulae. In the "False" cases the anomalous situations are caused by several reasons. In the "scen1" scenario bad and unsafe transactions occur because only the action CASN has not been performed. Finally, in the scenarios affected by noise the causes of failure occur because one or two required actions are not performed during a payment in cash. The proposed method is also able to localize potentially fraudulent bank transactions. As we stated into the introduction, the proposed method can be considered as back-end tool for a financial organization (for instance, a bank) but also for cyber-insurance companies in order to be certain that the company to ensure have effective protection systems from cyber security threats.

Table II: Detailed Properties

	Traces					
Formulae	**all-noise**	**all-nonoise**	**scen1**	**scen2**	**all-noise**	**all-nonoise**
φ_1	531	708	327	701	2478	3326
φ_2	1293	1292	1348	1299	6678	6674
φ_3	67	0	325	0	249	0
φ_4	50	0	0	0	259	0
φ_5	49	0	0	0	260	0
φ_6	5	0	0	0	28	0
φ_7	3	0	0	0	18	0
φ_8	2	0	0	0	30	0
# of Traces	2000	2000	2000	2000	10000	10000

IV. Conclusion and Future Work

In last years data breaches are plaguing financial companies, thus the cyber insurance market is emerging. Considering that financial transactions are supported by safety critical systems, in this paper we have proposed a method to retrieve formal models starting from execution traces. In this way, we try to the fill the gap still exists between formal models and a real-world system and its execution. A real system has been used as a case study to evaluate the proposed methodology, obtaining encouraging results. Since the obtained formal models tend to be large and complex, our aim is to use also efficient model checking techniques to reduce the state explosion problem, as for example those technique based on the use of selective mu-calculus logic [11], [8]. Another research direction is represented by the application of the proposed method to cyber physical systems [12], [13], [14]. Furthermore, as future work we plan to extend our approach as a runtime verification approach [15], in order to apply our method at runtime to all bank transactions.

Acknowledgment

This work has been partially supported by H2020 EU-funded projects NeCS and C3ISP and EIT-Digital Project HII and PRIN "Governing Adaptive and Unplanned Systems of Systems" and the EU project CyberSure 734815.

References

[1] Clarke, E.M., Grumberg, O., Peled, D.: Model checking. MIT Press (2001)

[2] Awad, A., Decker, G., Weske, M.: Efficient compliance checking using BPMN-Q and temporal logic. In: BPM. Volume 5240 of LNCS., Springer (2008) 326–341

[3] Garavel, H., Lang, F., Mateescu, R., Serwe, W.: CADP 2011: a toolbox for the construction and analysis of distributed processes. STTT **15**(2) (2013) 89–107

[4] Mao, H., Chen, Y., Jaeger, M., Nielsen, T.D., Larsen, K.G., Nielsen, B.: Learning probabilistic automata for model checking. In: QEST 2011, IEEE Computer Society (2011) 111–120

[5] Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **75**(2) (1987) 87–106

[6] Epifani, I., Ghezzi, C., Mirandola, R., Tamburrelli, G.: Model evolution by run-time parameter adaptation. In: 31st International Conference on Software Engineering, ICSE, IEEE (2009) 111–121

[7] Barbuti, R., Francesco, N.D., Santone, A., Vaglini, G.: Selective mu-calculus and formula-based equivalence of transition systems. J. Comput. Syst. Sci. **59**(3) (1999) 537–556

[8] Barbuti, R., Francesco, N.D., Santone, A., Vaglini, G.: Reduced models for efficient CCS verification. Formal Methods in System Design **26**(3) (2005) 319–350

[9] De Francesco, N., Lettieri, G., Santone, A., Vaglini, G.: Heuristic search for equivalence checking. Software and Systems Modeling **15**(2) (2016) 513–530 cited By 10.

[10] Munoz-Gama, J.: Conformance Checking and Diagnosis in Process Mining - Comparing Observed and Modeled Processes. Volume 270 of Lecture Notes in Business Information Processing. Springer (2016)

[11] Santone, A., Vaglini, G.: Abstract reduction in directed model checking CCS processes. Acta Inf. **49**(5) (2012) 313–341

[12] Martinelli, F., Mercaldo, F., Orlando, A., Nardone, V., Santone, A., Sangaiah, A.K.: Human behavior characterization for driving style recognition in vehicle system. Computers & Electrical Engineering (2018)

[13] Martinelli, F., Mercaldo, F., Nardone, V., Santone, A.: Car hacking identification through fuzzy logic algorithms. In: Fuzzy Systems (FUZZ-IEEE), 2017 IEEE International Conference on, IEEE (2017) 1–7

[14] Martinelli, F., Mercaldo, F., Nardone, V., Orlando, A., Santone, A.: Who's driving my car? a machine learning based approach to driver identification. In: ICISSP. (2018)

[15] Leucker, M., Schallhart, C.: A brief account of runtime verification. The Journal of Logic and Algebraic Programming **78**(5) (2009) 293 – 303 FLACOS07.