POSTER: A Data Life Cycle Modeling Proposal by means of Formal Methods

Madalina G. Ciobanu
madalina.ciobanu@unimol.it
Department of Medicine and Health Sciences "Vincenzo Tiberio", University of Molise
Campobasso, Italy

Fausto Fasano
fausto.fasano@unimol.it
Department of Bioscience and Territory, University of Molise
Pesche (IS), Italy

Fabio Martinelli
fabio.martinelli@iit.cnr.it
IIT-CNR
Pisa, Italy

Francesco Mercaldo
francesco.mercaldo@iit.cnr.it
IIT-CNR & Department of Bioscience and Territory, University of Molise
Pisa, Italy

Antonella Santone
antonella.santone@unimol.it
Department of Bioscience and Territory, University of Molise
Pesche (IS), Italy

ABSTRACT

Data usually evolve according to specific processes, with the consequent possibility to identify a profile of evolution: the values it may assume, the frequencies at which it changes, the temporal variation in relation to other data, or other constraints that are directly connected to the reference domain. A violation of these conditions could be the signal of different menaces that threat the system, as well as: attempts of a tampering or a cyber attack, a failure in the system operation, a bug in the applications which manage the life cycle of data. To detect such violations is not straightforward as processes could be unknown or hard to extract. In this paper we propose an approach to model the data life cycle by observing the data evolution in its life cycle. Thus, we represent users able to alter data through timed automata. Through model checking, the obtained profile of evolution can be used to detect anomalies in relational database, data warehouse and big data.

CCS CONCEPTS

• **Security and privacy → Formal methods and theory of security**; *Database activity monitoring*; • **Theory of computation → Logic and verification**; **Modal and temporal logics**; *Logic and databases.*

KEYWORDS

Data modeling, data life cycle, database, data warehouse, big data, timed automata, formal methods

ACM Reference Format:
Madalina G. Ciobanu, Fausto Fasano, Fabio Martinelli, Francesco Mercaldo, and Antonella Santone. 2019. POSTER: A Data Life Cycle Modeling Proposal by means of Formal Methods. In *ACM Asia Conference on Computer and Communications Security (AsiaCCS '19), July 9–12, 2019, Auckland, New Zealand.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3321705.3331003

1 INTRODUCTION AND RELATED WORK

Data fields within databases usually vary according to specific processes which reflect the steps accomplished by the users. These processes are typically tacit, but they are reflected in the data variations. These variations, for instance, can refer to:

- groups of data fields which vary contextually: for instance, data fields *a*, *b*, and *c* vary every time together;
- temporal intervals in which the data field varies; for instance, data field *a* varies once in a month;
- users which usually have access to the data field and how; for instance, an user writes every two hours on data field *a*.

When a data field variation violates the correspondent process, it can be the case of a "data anomaly" [4]. In fact, data usually evolves according to specific processes. An anomaly, i.e., an illegal or unexpected alteration of data field, could be determined by several likely causes:

(1) tampering of the database;
(2) a cyber attack to the system of which the database is a part;
(3) a failure in the operating system of the infrastructure;
(4) a bug in the applications which update the database.

Thus, the detection of a data anomaly could lead to: reveal an improper behaviour of a clerk who has access to data (1); discover an intrusion or a cyber attack (2); verify the quality of software and hardware infrastructure (3,4). The problem is that the data variations could be not made explicit and consequently they cannot be verified, because they are not known by the organization responsible of the database. This happens for several reasons:

- the data variation can be the result of several different processes;
- the data variation could derive from not formalized processes;
- the data variation could be due to incidental or circumstances' factors, as well as: the behaviour of a specific class of users, data are available according to specific constraints,

different configurations of the systems may influence the order in which updates are made.

Data fields can be modified by recurrent or by non-deterministic processes. Unlikely recurrent processes, the non-deterministic processes have not the characteristic of occurring always in the same way, since they are the result of stochastic events. For their intrinsically unpredictable nature, non-deterministic processes must not be used for deriving data evolution, as this would entail false positives proliferation. Conversely, recurrent processes must be used for defining data life cycle. Recurrent processes may be explicit or latent: explicit processes are known, and could be completely or partially formalized, while latent processes are not known to the organization that owns/uses the database.

In general, a database administrator may have a partial view of the processes that change the data fields, with missing or incomplete information on:

- which applications modify which data field;
- what is the order of the different application execution when changing the data field;
- what is the frequency of execution of each process that modifies a data field.

Current literature focus on traditional static outlier detection [4, 5] widely applied in data mining problem, the correlation from variations, the dynamic outlier detection, was analyzed only in log mining problem [9]. Several studies in literature propose algorithms without applying it in real situations [6, 7]. Simonet et al. [10] propose a resource data life cycle formalisation using Petri nets and a mechanism to execute some tasks whether the data is in a certain life cycle stage, differently the proposed method is focused on the information data life cycle and not to the one related to resource files.

In this paper we propose a data-driven approach aimed to extract the data variation directly by observing the database while modifications occur. We focus on recurrent processes that take place at regular time intervals and, each time they are enacted, modify the same sets of data fields. Our aim is to capture these recurring characteristics of data fields variation by mining the variations happening on the database using formal methods. The use of a formal system model is important to explicitly specify the dependency between the processes and the various stages of data. The data life cycle for each user is modelled through a timed automata [1]. The data anomalies can be detected through logic properties describing the expected legitimate behaviour.

2 DATA LIFE CYCLE MODELING

In this section we describe our approach to model data life cycle.

The method we propose is data-driven: it works with homogeneous data (stored in relational databases or data warehouses) but also with heterogeneous ones (i.e., big data).

Figure 1 depicts the proposed approach.

Data variation can be performed by users (*u1*, *u2*, *u3*, *u4* and *u5* in Figure 1) able to access to data. For each user data operation following information are stored:

- *timestamp*: date and time when the transaction took place;
- *user@host*: host and user who performed the transaction;
- *operation*: type of transaction performed;
- *table*: table name;
- *id*: row;
- *column*: column name.

Through the timestamp, a series of time window fixed clusters is defined: the users with correspondent data operations falling in each temporal cluster are retrieved from the log.

Then, each user behaviour is modeled through a timed automata (*a1*, *a2*, *a3*, *a4* and *a5* in Figure 1): the timed automata network is composed by all the user. The synchronization between the timed automata happens when a new temporal cluster is starting, while the data operations (on the specific field of a table) are represented by the state transitions. Once modelled the user behaviours directly from the log, exploiting model checking technique, it is possible to verify whether a set of properties (aimed to verify the legitimate user behaviour) is satisfied on the model.

Thus, the approach can be exploited to detect anomalies in the data alteration processes. For instance, considering the following legitimate behaviours for three users:

- *behaviour 1*: "Antonella" performs only *read* operations on the *a* field of the "Budget" table and on the *b* field of the "Project" table;
- *behaviour 2*: "Francesco" performs only *read* operations on the field *a* of the "Budget" table;
- *behaviour 3*: "Madalina" performs *write* operations on the *a* field of the "Budget" table and on the *b* field of the "Project" table;
- *behaviour 4*: "Francesco" should read the field *a* of the "Budget" table before the *write* operations performed by "Madalina", while "Antonella" needs to access the *a* field after the "Madalina" update;
- *behaviour 5*: "Antonella" should read the field *b* of the "Project" table before the *write* operations performed by "Madalina" on the same field.

An example of anomalies can be the following:

(1) as expected by *behaviour 1*, "Antonella" should always perform *read* operation, but from the log it seems that she is performing *write* operations on "Registry" table;
(2) as expected by *behaviour 2*, "Francesco" should always read the "Budget" table, but from the log it seems that he is performing *read* operations on the "Project"table;
(3) in addition "Francesco" is performing *write* operations on the *b* field of the "Project" table, with the consequent inconsistency in the *read* operation of "Antonella";
(4) as expected by *behaviour 3*, "Madalina" should always perform *write* operations on the "Budget" and "Project" tables, but from the logs it seems that she is performing also *read* operations on "Registry" table;
(5) as expected from *behavior 4*, "Francesco" should read the *a* fields before the update of "Madalina", but from the log it seems that he is reading the date after the update. Contrarily, "Antonella" is reading the same field before the "Madalina" update, violating the expected behavior;
(6) as expected from *behavior 5*, "Antonella" should read the *b* field of the "Project" table before the update of "Madalina", but the log shows that she is reading the data after the "Madalina" update.

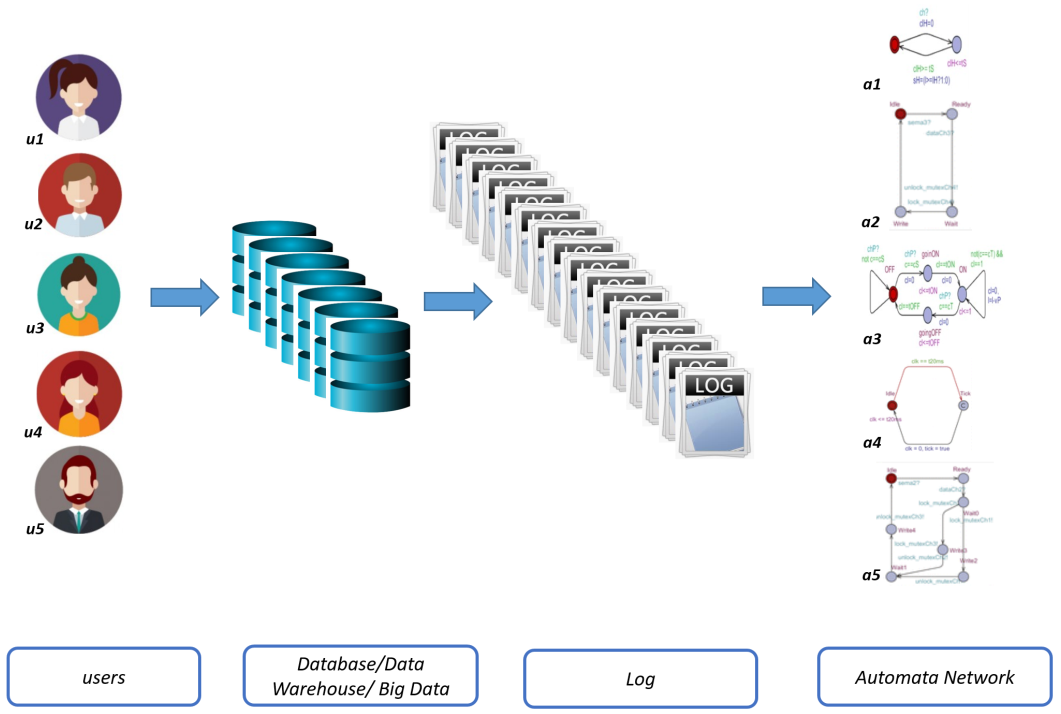

Figure 1: The proposed method.

These anomalies can be detected exploiting the proposed timed automata network and a set of temporal logic properties (provided by domain experts) aimed to check the user trustworthiness.

It is important to observe that the extended number of possible legitimate behaviours (and of the data anomalies) would make the analysis impossible without the support of automatic formal verification environments. In fact, formal analysis tools such as model checkers automatically consider all possible execution paths through the modeled user behaviours. Whether there is any possibility of a data anomaly, the model checker will be able to find it. In a parallel system where concurrency must be managed (as highlighted, for instance, from 4 and 5 behaviours), formal analysis can explore all possible interleaving and event orderings. This level of coverage is not possible to achieve through testing, from the other side formal methods have the property of completeness i.e., they cover all aspects of the system under analysis.

3 CONCLUSION AND FUTURE WORK

In this paper a method aimed to mine data anomalies is proposed. The main idea is to model data life cycle by exploiting timed automata networks. We plan to exhaustively evaluate the proposed method using a real-world dataset related, for instance, to a security domain [2, 8] or a biology one [3]. Furthermore, a research direction could be focused on the semantic meaning of change, i.e. when the change of data is not consistent with its semantic identity.

ACKNOWLEDGMENTS

This work has been partially supported by H2020 EU-funded projects SPARTA contract 830892 and C3ISP and EIT-Digital Project HII and PRIN "Governing Adaptive and Unplanned Systems of Systems" and the EU project CyberSure 734815.

REFERENCES

[1] Johan Bengtsson and Wang Yi. 2003. Timed automata: Semantics, algorithms and tools. In *Advanced Course on Petri Nets*. Springer, 87–124.

[2] Gerardo Canfora, Francesco Mercaldo, Giovanni Moriano, and Corrado Aaron Visaggio. 2015. Composition-malware: building android malware at run time. In *2015 10th International Conference on Availability, Reliability and Security*. IEEE, 318–326.

[3] Michele Ceccarelli, Luigi Cerulo, and Antonella Santone. 2014. De novo reconstruction of gene regulatory networks from time series data, an approach based on formal methods. *Methods* 69, 3 (2014), 298–305.

[4] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *ACM computing surveys (CSUR)* 41, 3 (2009), 15.

[5] Prasanta Gogoi, DK Bhattacharyya, Bhogeswar Borah, and Jugal K Kalita. 2011. A survey of outlier detection methods in network anomaly identification. *Comput. J.* 54, 4 (2011), 570–588.

[6] Mingwei Leng, Xiaoyun Chen, and Longjie Li. 2008. Variable length methods for detecting anomaly patterns in time series. In *Computational Intelligence and Design, 2008. ISCID'08. International Symposium on*, Vol. 2. IEEE, 52–56.

[7] Mingwei Leng, Xinsheng Lai, Guolv Tan, and Xiaohui Xu. 2009. Time series representation for anomaly detection. In *Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on*. IEEE, 628–632.

[8] Fabio Martinelli, Francesco Mercaldo, Vittoria Nardone, and Antonella Santone. 2017. Car hacking identification through fuzzy logic algorithms. In *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. IEEE, 1–7.

[9] Michal Munk, Jozef Kapusta, and Peter Švec. 2010. Data preprocessing evaluation for web log mining: reconstruction of activities of a web visitor. *Procedia Computer Science* 1, 1 (2010), 2273–2280.

[10] Anthony Simonet, Gilles Fedak, and Matei Ripeanu. 2015. Active Data: A programming model to manage data life cycle across heterogeneous systems and infrastructures. *Future Generation Computer Systems* 53 (2015), 25–42.