Study of an Approach Based on the Analysis of Computer Program Execution Traces for the Detection of Vulnerabilities

Gouayon Koala[1(✉)], Didier Bassolé[1], Télesphore Tiendrébéogo[2], and Oumarou Sié[1]

[1] Laboratoire de Mathématiques et d'Informatique, Université Joseph Ki-Zerbo, Ouagadougou, Burkina Faso
gouayonkoala1@gmail.com

[2] Laboratoire d'Algèbre, de Mathématiques Discrètes et d'Informatique, Université Nazi Boni, Bobo-Dioulasso, Burkina Faso
http://www.ujkz.bf, http://www.univ-bobo.gov.bf

Abstract. Malicious attacks exploit software vulnerabilities to violate key security features in computer systems. In this paper, we review the related works of studies that propose mechanisms for detecting software vulnerabilities or ways to protect application data. The aim is to analyse how these mechanisms are exploited to detect software vulnerabilities and secure data via applications. Then, we present tracing techniques to understand the behaviour of applications. Finally, we present an approach based on the analysis of program execution traces that allows the detection of vulnerabilities.

Keywords: Detection · Vulnerabilities · Tracing · Attacks

1 Introduction

Digital technology (devices, systems, connected objects) increasingly offers multiple benefits and a variety of services to businesses and individuals. This has increased usage and apps are an important part of this digital boom. Competition between application developers has brought much innovation. However, many of these applications are increasingly vulnerable. Exploitable vulnerabilities in software can pose potential threats to the functioning of IT systems, impacting millions of users on a daily basis. To reduce malware, researchers have proposed protection and control mechanisms (firewalls, intrusion detection systems, web scanners, etc.) to secure data. Despite these efforts, cybercriminals are using more sophisticated and innovative evasion techniques that hamper efforts to secure data. No system is spared from the malicious actions of attackers. In addition, vulnerabilities in applications are increasing in number and intensity, facilitating

A. D. Mambo et al. (Eds.): InterSol 2022, LNICST 449, pp. 105–115, 2022.
https://doi.org/10.1007/978-3-031-23116-2_8

attacks in many systems [1–3]. To counter threats and attacks, several methods and techniques have been proposed by researchers [1,4–9].

Studies based on these approaches have made it possible to propose solutions for the detection of vulnerabilities on the one hand and data protection mechanisms on the other. Despite all the efforts made, the search for effective protection against threats and attacks is still ongoing. Vulnerabilities and anomalies in applications therefore constitute a threat to their users (individuals or companies). To find solutions capable of effectively improving data protection become therefore essential [10]. The first step to improve this protection is to be able to detect threats and attacks on software. This study therefore reviews approaches based on analysis methods for detecting vulnerabilities. The objective is to propose an approach to detecting vulnerabilities in applications by exploiting these existing methods.

The rest of this document is organized as follows: in the Sect. 2 we present the Background of Study and the problematic of this study. The Sect. 3 defines the concepts used in our study and the Sect. 4 deals with related work and identified shortcomings. In the Sect. 5, we present our approach for analysing execution traces. We conclude by presenting a synthesis of our contribution and our perspectives in Sect. 6.

2 Background of Study

The digital market is growing at a rapid pace. From 2016 to 2020, we have gone from 2 billion objects to 200 billion connected objects, an increase of 200% [11]. This growth is due to the presence of digital technology in all sectors of activity such as education, health, finance, entertainment, home, energy, smart cities, tourism and transport [12–15]. Users are more concerned with innovations and benefits and rarely pay attention to the safety and security of the applications that offer them services. The use of certain applications can have consequences with regard to the protection of the data that passes through them. Thus, negligence in data protection can put users at risk and disrupt computer systems. For example, cybercriminals can take control of a computer system and cause panic among citizens[1].

Consequently, software security became an important strategic issue [1]. Thus, several research projects have been carried out and solutions proposed to solve these security problems. Despite these research efforts, the number of vulnerabilities in computer systems continues to increase [3]. Also, malicious actions to exploit these vulnerabilities continue to increase and some actions can cause serious problems such as loss of revenue, disruption of critical operations within an organisation, ... Despite the complexity of today's IT systems, hackers are adapting to the evolution to succeed in their attacks. Recent years have shown that attack scenarios are evolving and becoming more and more complex. These attacks target both hardware components and applications. Indeed, many

[1] https://www.voaafrique.com/a/une-cyberattaque-cause-des-p%C3%A9nuries-de-carburant-aux-usa/5888344.html.

applications contain multiple vulnerabilities that can be exploited by hackers. Most of these attacks rely on software-related attack vectors, in particular on use of software (privilege escalation, information leakage, denial of service ...) or on exploiting software implementation errors (buffer overflows).

Although proposals for solutions exist in the literature, it is difficult to propose security solutions applicable to all applications in particular. In order to reduce security vulnerabilities in software, several methods and analysis techniques are used to detect their vulnerabilities. When the source code of an application is accessible, we can obtain information that can help improve the detection of vulnerabilities and propose solutions to protect data. Indeed, such analysis is only possible if the source code is available and understandable. Hence the need to explore application tracing techniques in this study. These techniques have been used in the literature on monitored machines to detect anomalies in systems. Few works have focused on application traces, which can be valuable for improving the quality of vulnerability detection. With this study, we wish to analyse the traces to examine and even detect precursor behaviours of attacks. Thus, it is essential to know how programs work through their traces, and to ensure their correct behaviour. This problem constitutes the main subject of our work.

We will begin by recalling some basic terminology and concepts associated with computer security. This then allows us to understand the vulnerabilities, threats and attacks that undermine data protection.

3 Concepts

3.1 Vulnerability

A vulnerability is an accidental or intentional fault (with or without intent to harm) in the specification, design or configuration of the system, or in the way it is used [8,16]. Applications may have security holes. This is all the more serious as these applications sometimes handle confidential data (passwords, bank card numbers) and are generally exposed to the public. These security flaws exist on all operating systems because several flaws are due to programming errors in the application [17,18].

Thus, the increasing complexity of the technologies used for application development coupled with the neglect of security by application developers can largely explain the presence of recurring vulnerabilities. There is a wide variety of vulnerabilities targeting applications. However, some are more well-known and dangerous than others. Several databases list these vulnerabilities with statistics indicating their relative importance. For example, databases such as CVE[2] (Common Vulnerabilities and Exposures), NVD[3] (National Vulnerability Database) or VUPEN[4] (Vulnerability Penetration testing) list all types of vulnerabilities, including those targeting applications.

[2] https://www.cvedetails.com/.
[3] https://nvd.nist.gov.
[4] http://www.vupen.com.

The increase in vulnerabilities and attacks has led many researchers to focus more on security in applications in order to improve data protection [1,4,6,7,10,19,20]. Work in this context has resulted in the proposal of taxonomies and classifications for the most common vulnerabilities and attacks [3,17,21–23]. Also, the vulnerability can be exploited to create an intrusion.

3.2 Attack and Intrusion

Attack is a malicious interaction to violate one or more security properties. It is an external fault created with the intention to harm. An attack may or may not be carried out by automatic tools [17,23].

Intrusion is an internal, but externally generated malicious act resulting from an attack that successfully exploited a vulnerability. Intrusion is any penetration of a computer system with the aim of undermining its confidentiality, integrity or availability. Intrusion detection brings together all the techniques implemented to alert the users of the computer system targeted by an attacker [1,17,21].

3.3 Execution Trace

A trace is a constitution of imprints left in an environment as a result of a process [24]. Any process can produce more or less persistent footprints. Just as a fingerprint is linked to something, a trace is always associated with an activity. Thus, the digital trace is a constitution of digital footprints left in a computer environment on the occasion of computer processes [18,24]. Over the years, the computer trace has become an object to be protected like other resources available in the computerised environment. Through web applications, many digital traces are constructed via the digital footprints left behind. We consider an execution trace as a reference to a mechanism that mainly collects information. This data can be analysed to detect sources of errors in software systems [25,26]. Thus, the execution traces of applications provide relevant data on the internal state of these applications.

4 Related Work

In this section we review the work relating vulnerabilities detection and proposed solutions. This allows us to understand the vulnerabilities, threats and attacks that compromise data protection and to discuss different ways of classifying them.

The widespread use of software is visible worldwide. Users are being targeted by cyber attacks that have exposed critical flaws in IT systems. Information security has become a major concern [4,11,27]. The danger posed by vulnerable applications affects information security and threatens the entire digital world by exploiting vulnerabilities. The scanning techniques used for vulnerability detection can be categorised into static, dynamic and hybrid approaches.

Static approach techniques (rule-based or model-based analysis, code similarity detection, etc.) rely on source code analysis [19,20]. The techniques of the dynamic approach (fuzz tests, analysis of spots or alterations, etc.) are performed during the execution of the programme [9,28]. Limitations in accessibility and understanding of the source code and poor code coverage have led to the hybrid analysis approach. The techniques of this approach combine static and dynamic analysis techniques to reduce the limitations of both approaches [5,6,21,29].

In the field of connected objects, various techniques have been proposed for vulnerability detection. Several papers have provided reviews on software vulnerability detection from different perspectives. For example, Chaabouni et al. [27], compared several techniques in the literature to detect and prevent new attacks in connected objects. In their review, they classified the threats and security challenges in connected objects (Fig. 1).

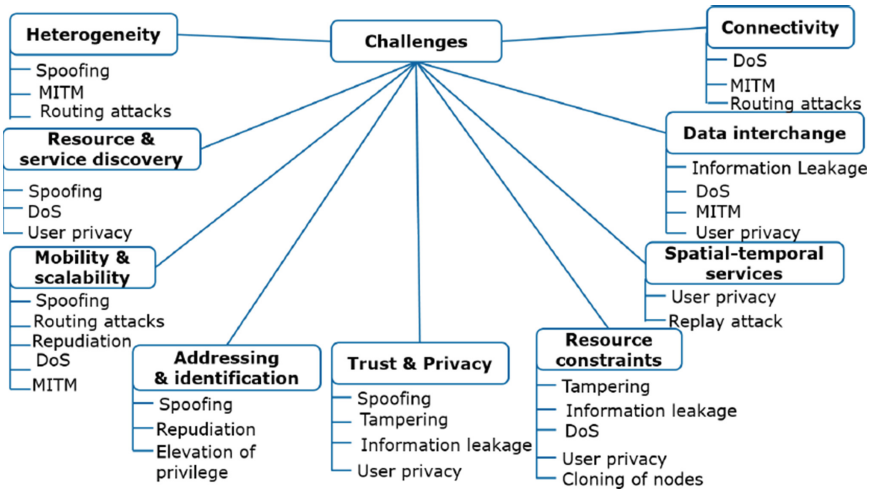

Fig. 1. Classification of threats in computer systems (Chaabouni et al. [27])

Their study reveals DDoS attacks that have compromised data availability. Their analysis notes that security attack vectors have evolved in terms of bot complexity and diversity. Although their work focuses on the characteristics of the attacks. They concentrate on network intrusion detection systems (NIDS). Nevertheless, we deduce that from these sequences of possible attacks in a computer system, there are attacks related to the programs that interest us in this work.

In [4], authors focused on the efforts made for intrusion detection in connected objects. In this review, Benkhelifa et al. made a set of proposals on the architecture of connected objects in order to improve data protection.

In [30], Braiek et al. examined fault detection in data and/or machine learning models in their review. The work of Zhang et al. in [2] provide a survey of machine

learning testing by bringing together aspects of work that have dealt specifically with software testing. In their review, authors have simultaneously covered all the types of machine learning approaches that have so far been addressed using testing. They thus identify the problems and challenges associated with software testing techniques and machine learning testing problems. This review provides a comprehensive study with a primary focus on machine learning testing.

Lin et al. [1], in their literature review discussed deep learning/neural network approaches to vulnerable code learning and neural networks for software vulnerability detection. In this review, authors examined neural techniques for learning and understanding code semantics to facilitate vulnerability discovery.

For mobile apps, a recent study on vulnerability detection techniques in Android was done by Qamar et al. [31]. Authors have provided taxonomies of malware detection approaches based on the analysis techniques used, platforms and data. In addition, this review presents the different attacks on mobile applications by providing a taxonomy on malware attack vectors. This taxonomy allowed them to examine the threat groups and vulnerabilities. This allowed them to identify their impact on users as presented in Fig. 2.

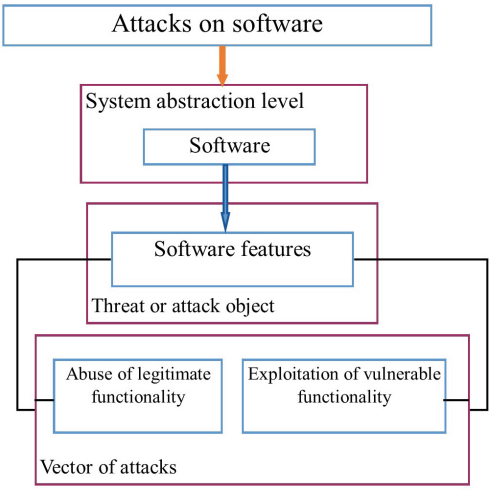

Fig. 2. Diagram of software attacks in a computer system (Qamar et al. [31])

All these techniques have shortcomings and some are sometimes ineffective in practice [5]. So far, the rapid increase in the number of vulnerabilities disclosed after the release of software products suggests that current software vulnerability detection techniques need to be improved in terms of efficiency and effectiveness [3,21]. Thus, an approach based on program execution traces can be used in research to study the behaviour of applications. Despite the problem of some application traces being very large (compression problems), the interest in traces is gaining importance as more and more researchers study the dynamic activities

of computer systems. Unfortunately, approaches based on execution traces are specific, hence a global approach.

We will provide an up-to-date overview of existing methods used for vulnerability detection, which includes a description of each method, its strengths and weaknesses, and its resistance to malware evasion techniques. In addition, we include an overview of studies on machine learning techniques used to improve vulnerability detection in software. Although various approaches and/or techniques for vulnerability detection are proposed in the literature, the changing digital environment with cloud computing, connected objects leads to new software vulnerabilities and thus new exploits such as ransomware.

Malware authors can use techniques such as code obfuscation, dynamic code loading, encryption or packaging to evade static analysis and even signature-based antivirus tools. Furthermore, static scanning techniques are limited by the understanding of the programming language used for application development.

Dynamic scanning techniques, on the other hand, can provide a greater understanding of the code being scanned. It thus provide better results in detecting vulnerabilities in software. It can be deduced that dynamic scanning is more robust than static scanning. However, existing dynamic analysis tools and techniques are imperfect. Moreover, no single tool can cover all aspects of malware behaviour and thus effectively detect vulnerabilities. This has led to the analysis of applications with methods of a hybrid approach that combines both approaches (static and dynamic). These techniques use machine learning algorithms to detect vulnerabilities and reduce malware.

Unfortunately, these techniques are used separately and lack a public reference data set for exchange. Such a set could allow a reliable and practical comparison of different machine learning detection techniques. This shortcoming may allow some vulnerabilities to escape detection.

These limitations show the importance of implementing experiments to evaluate the effectiveness of software vulnerability detection techniques by subjecting applications to different analysis approaches and different attack scenario assumptions. The knowledge gained from these experiments will also be useful in identifying ways to improve the design of systems for data protection.

5 Methodology

One of the most important needs for the security of the data to be processed is the improvement of the software vulnerability detection strategy. Although tracing techniques have already been used to detect anomalies on different systems, especially with the help of system calls made on a monitored machine [18,25,32]. Few works have focused on the set of information available in the events. The arguments of the system calls can prove invaluable in improving the quality of vulnerability detection. The Fig. 3 shows our model for studying application execution traces.

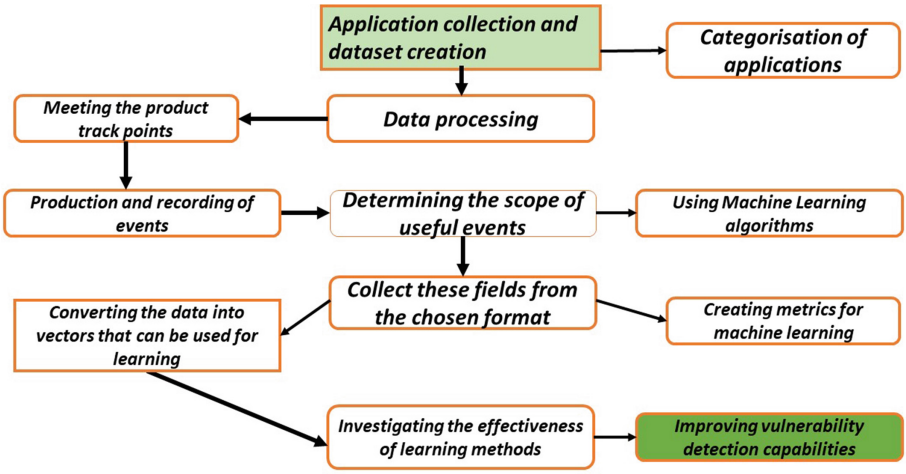

Fig. 3. The stages of transformation of the events collected with the tracing techniques

A tracepoint is a piece of code added to the execution of the traced program. When it is executed, it makes a call to the plotter so that the latter records the event associated with the tracepoint. An event contains information about the system to time of execution when the associated trace point was encountered. It can contain various information with a time stamp and contains information about the system when it was generated, such as the name of function where the tracepoint was present.

The purpose of tracing an application is to obtain accurate information about its state of operation. To obtain this information, tracing relies on events that are captured when the system reaches certain states. The collection is fast and does not change the behaviour of the system very much. The events are generated using tracepoints. These tracepoints can be added statically to an application or dynamically during its execution. Tracepoints are small pieces of code that have the function of making a call, containing information about the state of the system, at the time they are encountered by the processor. We will use all the information gathered by the tracer points to improve the quality of vulnerability detection. The tracer is the software responsible for collecting events as they occur and storing them in an orderly fashion in the trace. Therefore, it must be less disruptive to the system while it is running so that the information collected is relevant and valid. We will then analyse this information in detail to identify vulnerabilities which we will then classify. Thus, we will identify the challenges of our analysis with a view to proposing software countermeasures to improve data protection. Then, we will make a choice of implementation that could improve the results of vulnerability detection and thus reduce the risks related to the exploitation of such an attack vector.

Traces as computer objects are managed by the computer environment in a trace base. This base is a digital device known to collect, organize and provide

services on the traces thus managed. In the perspectives of our study, we plan to further refine our analysis approach. We are interested in logging, which refers to a mechanism for collecting information to support the normal operation of a system. To analyse the execution traces of programs, several analysis tools or tracers are proposed. Thus, we will comprehensively evaluate the tracer tools for the implementation of our approach. This evaluation assumes a set of test data that we have to choose. We will also discuss runtime trace analysis tools as a means of protection against attacks. In addition, we will compare our results with previous studies on software vulnerabilities. Finally, we will reconstruct the semantics of a program from its execution trace.

6 Conclusion

The IT environment is constantly changing with a proliferation of malware. The level of complexity of malware is increasing every day, prompting the exploration of new analysis methods, machine learning techniques, etc. This paper explored the notion of modelled digital traces. It developed the possibilities of exploiting them to build a vulnerability detection model on the one hand and to improve the detection of application vulnerabilities on the other. We have studied techniques and methods for detecting software vulnerabilities. Research efforts have been made with different approaches to detect software vulnerabilities. We have presented a state of art of studies on vulnerability detection and data protection in applications. The focus is on detection methodologies and threats addressed, results and shortcomings. In addition, we presented our approach to analysis based on program execution traces. We have shown that the analysis of execution traces of application can provide relevant and valuable information about the impact of this application on data security according to the predefined rules of the method. From the terminologies associated with traces and the study of digital traces, we have presented the strategies adopted and identified the strengths of this approach.

The continuation of our work can be divided into three main areas. Firstly, we will study the tools for analysing execution traces in order to select those best suited to our approach, depending on the platforms. Second, we will analyse the application traces of our dataset and evaluate our results. Finally, we will reconstruct the semantics of a program from its execution trace.

References

1. Lin, G., Wen, S., Han, Q-L., Zhang, J., Xiang, Y.: Software vulnerability detection using deep neural networks: a survey. In: Proceedings of the IEEE, May 2000. https://doi.org/10.1109/JPROC.2020.2993293
2. Zhang, J.M., Harman, M., Ma, L., Liu, Y.: Machine learning testing: survey, landscapes and horizons. IEEE Trans. Softw. Eng. **48**, 1 –36 (2022). https://doi.org/10.1109/TSE.2019.2962027

3. Chakkaravarthy, S.S., Sangeetha, D., Vaidehi, V.: A Survey on malware analysis and mitigation techniques. Comput. Sci. Rev. **32**, 1–23 (2019). https://doi.org/10.1016/j.cosrev.2019.01.002.,
4. Benkhelifa, E., Welsh, T., Hamouda, W.: A Critical review of practices and challenges in intrusion detection systems for IoT: towards universal and resilient systems. IEEE Commun. Surv. Tutor. PP(99), 1 (2018)
5. Yamaguchi, F., Golde, N., Arp, D., Rieck, K.: Modeling and discovering vulnerabilities with code property graphs. In: IEEE Symposium on Security and Privacy, pp. 590–604, May 2014
6. Liu, L., De Vel, Q., Han, Q.-L., Zhang, J., Xiang, Y.: Detecting and preventing cyber insider threats: a survey. IEEE Commun. Surv. Tuts. **20**(2), 1397–1417, 2nd Quart. (2018)
7. Sun, N., Zhang, J., Rimba, P., Gao, S., Zhang, L.Y., Xiang, Y.: Data-driven cybersecurity incident prediction: a survey. IEEE Commun. Surveys Tuts. **21**(2), 1744–1772 (2019)
8. Ghaffarian, S.M., Shahriari, H.R.: Software vulnerability analysis and discovery using machine-learning and data-mining techniques: a survey. ACM Comput. Surv. **50**(4), 1–36 (2017)
9. Newsome, J., Song, D.X.: Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In: Proceedings of NDSS, pp. 3–4 (2005)
10. Li, Z., et al.: Vuldeepecker: a deep learning-based system for vulnerability detection. In: Proceedings of NDSS, pp. 1–15 (2018)
11. U. N.: IDC, Intel, "A Guide to the Internet of Things Infographic." February 2015. https://www.intel.com/content/www/us/en/internet-of-things/infographics/guide-to-iot:html
12. Vermesan, Q., Friess, P.: Internet of Things Applications - From Research and Innovation to Market Deployment Book. River Publishers, Jun. 2014. http://www.internet-of-thingsresearch.eu/pdf/IERC_Cluster_Book_2014_Ch.3_SRIA_WEB.pdf
13. Chen, C., Seff, A., Kornhauser, A., Xiao, J.: Deepdriving: learning affordance for direct perception in autonomous driving. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2722–2730 (2015)
14. Litjens, G., et al.: A survey on deep learning in medical image analysis. Med. Image Anal. **42**, 60–88 (2017)
15. Pei, K., Cao, Y., Yang, J., Jana. S.: Deepxplore: automated whitebox testing of deep learning systems. In: Proceedings of the 26th Symposium on Operating Systems Principles, pp. 1–18. ACM (2017)
16. Sestili, C.D., Snavely, W.S., VanHoudnos, N.M.: Towards security defect prediction with AI (2018). arXiv:1808.09897. http://arxiv.org/abs/1808.09897
17. Akrout, R.: Analyse de vulnérabilités et évaluation de systèmes de détection d'intrusions pour les applications Web. Thesis, Institut National des Sciences Appliquées de Toulouse (INSA Toulouse) (2013)
18. Meresse, S., Muratet, M., Yessad, A.: Analyse de traces d'exécution de programmes informatiques : application au jeu sérieux Prog&Play", ORPHEE-RDV, atelier: Méthodologies et outils pour le recueil, l'analyse et la visualisation des traces d'interaction, January 2017, Font-Romeu, France. hal-01515783
19. Kim, S., Woo, S., Lee, H., Oh, H.: VUDDY: a scalable approach for vulnerable code clone discovery. In: Proceedings of Symposium on Security and Privacy, pp. 595–614, May 2017

20. Jang, J., Agrawal, A., Brumley, D.: ReDeBug: finding unpatched code clones in entire OS distributions. In: IEEE Symposium on Security and Privacy, pp. 48–62, May 2012
21. Votipka, D., Stevens, R., Redmiles, E., Hu, J., Mazurek, M.: Hackers vs. testers: a comparison of software vulnerability discovery processes. In: Proceedings of IEEE Symposium on Security and Privacy (SP), pp. 374–391, May 2018
22. Sang, F.L.: Protection des systèmes informatiques contre les attaques par entrées-sorties. Thesis, Institut National des Sciences Appliquées de Toulouse(INSA Toulouse) (2013)
23. Benali, F.: Modélisation et classification automatique des informations de sécurité" (2009)
24. Mille, A.: Des traces à l'ère du Web. Intellectica **59**, 7–28 (2013)
25. Galli, T., Chiclana, F., Siewe, F.: Quality properties of execution tracing, an empirical study. Appl. Syst. Innov. **4**, 20 (2021). https://doi.org/10.3390/asi4010020
26. Savary, A.: "Détection de vulnérabilités appliquée à la vérification de code intermédiaire de Java Card", Université de Limoges (2016)
27. Chaabouni, N., Mosbah, M., Zemmari, A., Sauvignac, C., Faruki, P.: Network intrusion detection for IoT security based on learning techniques. IEEE Commun. Surv. tutor. **21**, 2671 –2701 (2018)
28. Pewny, J., Schuster, F., Bernhard, L., Holz, T., Rossow, C.: Leveraging semantic signatures for bug search in binary programs. In: Proceedings of the 30th Annual Computer Security Applications Conference (ACSAC), pp. 406–415 (2014)
29. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Yu, P.S.: A comprehensive survey on graph neural networks (2019). arXiv:1901.00596. http://arxiv.org/abs/1901.00596,
30. Braiek, H., Khomh, F.: On testing machine learning programs (2018). arXiv preprint arXiv:1812.02257
31. Qamar, A., Karim, A., Chang, V.: Mobile malware attacks: review, taxonomy & future directions. Futur. Gener. Comput. Syst. **97**, 887–909 (2019). https://doi.org/10.1016/j.future.2019.03.007
32. Hojaji, F., Mayerhofer, T., Zamani, B., Hamou-Lhadj, A., Bousse, E.: Model execution tracing: a systematic mapping study. Softw. Syst. Model **18**, 3461–3485 (2019)