I See Syscalls by the Seashore: An Anomaly-based IDS for Containers Leveraging Sysdig Data

Anderson Frasão
Informatics Department
Federal University of Paraná
Paraná, Brazil
aacf20@inf.ufpr.br

Tiago Heinrich
Max Planck Institute for Informatics
Saarbrücken, Germany
theinric@mpi-inf.mpg.de

Vinicius Fulber-Garcia
Informatics Department
Federal University of Paraná
Paraná, Brazil
vinicius@inf.ufpr.br

Newton C. Will
Computer Science Department
Federal University of Technology - Paraná
Dois Vizinhos, Brazil
will@utfpr.edu.br

Rafael R. Obelheiro
Computer Science Department
State University of Santa Catarina
Joinville, Brazil
rafael.obelheiro@udesc.br

Carlos A. Maziero
Informatics Department
Federal University of Paraná
Paraná, Brazil
maziero@inf.ufpr.br

***Abstract*—Intrusion detection in virtualized environments is vital due to the widespread adoption of virtualization technology. A common strategy for achieving this task involves collecting data from the virtual environment and providing it to intrusion detection solutions. However, these solutions can be affected by other elements present in the virtual environment. An approach that has gained prominence is applying machine learning (ML) models to perform anomaly-based intrusion detection based on system call traces. In Linux-based environments, many tools can be used for collecting the system calls issued by processes and containers; two of the most popular are *strace* and *sysdig*. This paper introduces a dataset of system call traces collected with *sysdig* with a focus on anomaly-based intrusion detection for containerized applications and uses this dataset to compare the effectiveness of *strace* and *sysdig* data and evaluate the performance of five different ML models for anomaly detection. The results reveal that *sysdig* is an attractive option, enabling the collection of system call traces with lower overhead than *strace* while achieving good detection performance with several ML models.**

***Index Terms*—Intrusion Detection, Anomaly Detection, Security.**

I. INTRODUCTION

Virtualization technologies within cloud computing have emerged as a promising solution to address the challenges posed by computing environments that rely on dedicated hardware. Virtualization enables a single physical machine to deploy and manage multiple virtual computing environments, offering fine-grained control over available computing resources and providing high flexibility, mobility, and scalability [1]. Consequently, it is possible to provide computing as a service [2], one capable of delivering a myriad of other computing-enabled services within virtual environments [3]–[5].

There are multiple strategies for virtualization. In particular, operating system-level virtualization takes advantage of the kernel to create an isolated environment for a specific process. This virtualization strategy, also known as containerization, enables resource sharing across a computing system while isolating processes from each other and the underlying operating system [6], [7]. Thus, containers offer low consumption of computing resources, minimal processing overhead, and do not depend on a complex hypervisor.

However, the popularity of containerized environments raised security concerns given the lower degree of isolation compared to hypervisor-based virtualization, with multiple attacks having been explored and studied [8]. These attacks have exploited various misconfigurations, deliberate backdoors, and software vulnerabilities to perform attacks such as break-ins, privilege escalation, side-channel attacks, and denial of service [8].

In light of the previously presented scenario, industry and academic researchers have been exploring intrusion detection systems. These systems aim to detect attacks within containers. Such intrusion detection systems can assess both the network and the host. Therefore, by employing data classification and anomaly detection techniques, intrusion detection systems can identify unexpected patterns and those that pose a risk to the system, allowing managers to trigger mitigation and protection mechanisms [7], [9]. Recently, several works have proposed solutions for intrusion detection in virtualized environments using techniques including system call analysis [7], [10], real-time behavior monitoring [11], and rule-based or machine learning-based analysis [12].

In Linux and other Unix derivatives, a tool often used for capturing system calls issued by applications is *strace* [13]. Despite having been used in the evaluation of intrusion detection systems based on system call analysis [7], [14], *strace* is not really suited for real-time data collection, since it imposes a significant performance penalty on the monitored applications. Therefore, a recent trend has been the use of *sysdig* [15], a Linux-specific tool, to collect system call data for IDS purposes. For instance, [16] have shown the feasibility of using *sysdig* data to perform anomaly-based intrusion detection for containerized applications based on Sequence Time-Delay Embedding (STIDE) and Bag of System Calls

(BoSC). Within the same scope, [17] proposes a framework for anomaly detection for containers running in Kubernetes clusters; although the proposal leverages *sysdig* data for anomaly detection using machine learning, it lacks empirical evaluation of the intrusion detection aspects. Finally, [18] uses *sysdig* data in an IDS that first classifies running containers using a clustering algorithm (DBSCAN) and then performs anomaly detection using a RandomForest classifier trained for each specific container class, achieving positive results. So far, the literature has not evaluated differences among machine learning algorithms in this context, nor directly compared *sysdig* and *strace* as data sources for anomaly detection based on system calls.

This paper aims to bridge these gaps, comparing the effectiveness of *sysdig* and *strace* data for anomaly-based IDS and evaluating five machine learning algorithms in this context. In summary, this paper presents the following contributions:

- We highlight the distinctions between using *sysdig* and *strace* for intrusion detection;
- We develop a novel, publicly available dataset containing system call traces collected with *sysdig* for 10 WordPress plugins (including malicious and non-malicious executions); and
- We propose a Machine Learning (ML) approach for intrusion detection using *sysdig* data, and evaluate it using five different algorithms.

The remainder of this paper is structured as follows. Section II provides background on anomaly-based intrusion detection and system calls. Section III proposes an IDS based on system call traces collected using the *sysdig* kernel module. Section IV presents our experimental evaluation, comparing the performance of ML algorithms with *sysdig* data as well as the effectiveness of anomaly detection using data collected with *strace* and *sysdig*. Section V reviews related work on intrusion detection in container-based virtualization environments. Finally, Section VI concludes the paper.

II. BACKGROUND

This section provides relevant background concepts in two parts. In Section II-A, we present the fundamentals of anomaly-based intrusion detection. In Section II-B, we define and explain system calls.

A. Anomaly-based Intrusion Detection

In computing, it is possible to define an anomaly as a pattern or phenomenon that deviates from the expected behavior of data, commonly referred to as normal behavior [19]. It is important to note that anomalies may arise in a computing system for various reasons, including malicious activities, software or hardware malfunctions, and incorrect configuration setups, among others.

Anomaly detection involves the process of identifying unexpected patterns and bringing them to the attention of an interested entity. Specifically, anomaly-based intrusion detection systems typically collect events of interest that occur in a target system and construct a statistical model describing the normal behavior of these data. Consequently, data collected during system operation are evaluated against this model, and any deviation from the expected outcome is flagged as an anomaly [20]. The outcome, in turn, can be presented as a label (categorical result) or a score (continuous result) [21].

However, there is no universal anomaly detection solution. Detection solutions typically focus on specific scenarios, considering their diverse and particular characteristics and behaviors as parameters to identify a range of potential anomalies. Thus, in the context of this paper, we regard anomalies as security threats capable of compromising the computing system or leaking data through the execution of malicious and undesirable operations. The process of identifying such threats is referred to as anomaly-based intrusion detection, or simply anomaly detection.

B. System Calls

Applications in the user space interact with the operating system via system calls. When an application needs resources located in the kernel space, it initiates a request through system calls to access these privileged resources. Examples of such resources relate to the process life cycle, network operations, and file management [7], [22].

Since system calls mediate the access of userspace processes to critical system resources, monitoring these calls provides rich insights into an application's behavior. As noted by [23], system calls can be categorized and classified based on their threat levels. This categorization is valuable in the context of intrusion detection processes [24].

Monitoring system-level calls for security purposes is a well-established technique in the literature [25]–[27]. Furthermore, other proposals have explored similar resources, such as system calls for detecting security leaks and malicious intruding processes, such as Binder calls in Android systems [28]–[30] and WASI calls in WebAssembly applications [31], [32].

III. PROPOSAL

There are several tools and techniques for collecting and monitoring information from running applications. For example, tools such as *strace* [13] and *sysdig* [15] are commonly used in Linux to record the system calls issued by applications. However, while both tools serve the same purpose, they employ different strategies.

strace uses the *ptrace* mechanism [33]: the kernel interrupts every system call issued by the monitored process twice to allow *strace* to collect data about the call (at entry to record the arguments, and at exit to save the return value). Since the system call is interrupted by the kernel and *strace* runs in userspace, each of these interruptions involves at least two context switches (kernel $\rightarrow$ user, user $\rightarrow$ kernel). The result is that applications run much slower when monitored with *strace*, and may even behave differently if they are timing-sensitive.

sysdig relies on a kernel probe (*sysdig-probe*) that, when a system call is issued, gathers a small amount of data about the call and writes it to a memory buffer. This buffer is mapped in userspace, from where the data are read asynchronously by the *sysdig* binary. Therefore, *sysdig* has a much smaller overhead compared to *strace*, since system calls are only interrupted while the necessary bits are copied to the buffer in the kernel, with no context switches. These two approaches are shown in Figure 1.

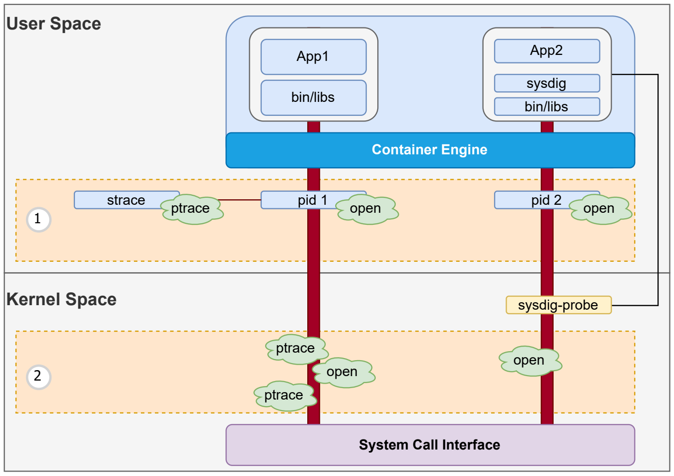

Fig. 1. Capturing system call traces using *strace* vs *sysdig*.

Our previous work [7] explored anomaly-based intrusion detection for containerized applications using system calls collected with *strace*, achieving positive results. The superior performance of *sysdig* prompted us to investigate the effectiveness of performing anomaly detection using system calls collected with *sysdig*. The basic idea is to use *sysdig* to collect system call traces from benign and malicious containerized applications, and use these traces to train and test machine learning models that perform trace classification. Our experimental evaluation is presented in Section IV.

IV. EXPERIMENTAL EVALUATION

This section presents the experimental evaluation of our proposal. Section IV-A describes the experimental methodology and environment, and explains the *strace* dataset and how we generated the new *sysdig* dataset. Section IV-B presents and discusses our results.

A. Methodology

Assessing the effectiveness of *sysdig* data for anomaly-based intrusion detection using machine learning classifiers involves four basic steps:

1) Selecting suitable classification algorithms;
2) Finding or generating an appropriate dataset;
3) Training and testing the classifiers using this dataset; and
4) Evaluating the results.

We evaluated the following classification algorithms:

- RandomForest;
- XGBoost;
- Nu-Support Vector;
- MultiLayer Perceptron; and
- AdaBoost.

We selected algorithms that performed well in previous work, enabling a comparison with other studies [7], [34].

While we were able to reuse the *strace* dataset from [7], we did not find a *sysdig* dataset that was appropriate for our evaluation, and so had to build a new dataset. For this dataset, we conducted experiments on a system running Linux Mint 21.2 (with kernel 5.15.0) and Docker (version 20.10.21) as the container engine. As a target application, we used WordPress[1] version 4.9.2 with ten different plugins[2] with known vulnerabilities, as shown in Table I. Three of these plugins—Social Warfare, File Manager, and Simple File List—were the same evaluated in [7], enabling us to compare results between *sysdig* and *strace* for these plugins. The application traces were collected by manually interacting with each plugin in isolation, performing malicious and non-malicious interactions. Each trace contains all system calls issued by the application during execution. The created *sysdig* dataset encompasses 10 distinct attack types (three of which overlap with those in [7]) and 10 normal patterns. In total, we collected 100 traces, obtained by executing each pattern five times. Our dataset is publicly available[3].

The classification algorithms were trained with 50% of the available data and tested with the remaining 50%. We used the *scikit-learn* library [35], with default parameters for each model.

B. Results and Discussion

Table II shows the results obtained by each classifier model when applied to our *sysdig* dataset and the best results from the *strace* dataset for three models—RandomForest, MultilayerPerceptron, and AdaBoost. As mentioned earlier, the *sysdig* tracer provides greater flexibility in managing a large number of system calls. Therefore, our goal here is to better understand how this monitoring perspective of *sysdig* can be advantageous for intrusion detection.

The Receiver Operating Characteristic (ROC) curve indicates that three of the classifiers achieve satisfactory results, and only two classifier performs below 94%. Even without delving into other metrics, it is evident that we have obtained classifiers capable of effectively identifying threats. The values describe a viable potential of using *sysdig* data for anomaly detection, being also beneficial considering the use of a drive for the data collection.

Precision analysis reveals that our models were affected by false positives (which represent benign samples that are misclassified as malicious), with AdaBoost and Multilayer Perceptron (MLP) obtaining the best values. Additionally, we observed an impact on false negatives (malicious samples that are misclassified as benign) in terms of recall, with the biggest impact in the XGBoost model. The overall effect on the negative classes is reflected in F1Score, with MLP emerging as the best classifier and only one model falling below the 80% threshold. Both the Balanced Accuracy (BAC) and Brier

[1] https://wordpress.org/
[2] https://wordpress.org/plugins/
[3] https://github.com/Carmofrasao/hids-docker

TABLE I
WORDPRESS PLUGINS USED IN OUR EXPERIMENTS.

Plugin	Version	Vulnerability
Social Warfare	3.5.2	stored cross-site scripting (CVE-2019-9978)
File Manager	6.8	upload and execution of arbitrary PHP code (CVE-2020-25213)
Simmple File List	4.2.2	upload and execution of arbitrary PHP code
Payments forms	2.4.6	arbitrary code injection
NEX-Forms	7.9.6	SQL injection by authenticated users (CVE-2022-3142)
Mail Masta	1.0	local file inclusion
Really Simple Guest Post	1.0.6	upload and execution of arbitrary PHP code
Paypal Currency Converter Basic for WooCommerce	1.3	read arbitrary files
LeagueManager	3.9.10	SQL code injection
CodeArt Google MP3 Player	1.0.11	server file disclosure

TABLE II
PERFORMANCE OF THE CLASSIFIERS FOR ANOMALY DETECTION.

Classifier	ROC	Precision	Recall	F1Score	Accuracy	BAC	Brier
Our proposal using *Sysdig* data							
RandomForest	90.84%	80.39%	83.63%	82.00%	82.00%	82.03%	18.00%
XGBoost	91.20%	80.85%	77.55%	79.17%	80.00%	79.95%	20.00%
Nu-Support Vector	94.14%	86.00%	87.76%	86.87%	87.00%	87.01%	13.00%
MultilayerPerceptron	95.88%	93.33%	85.71%	89.36%	90.00%	89.92%	10.00%
AdaBoost	95.84%	83.67%	83.67%	83.67%	84.00%	83.99%	16.00%
Best result for *strace* from [7] (window size 7, 10 executions)							
RandomForest	-	98.7%	73.0%	83.9%	94.6%	-	-
MultilayerPerceptron	-	90.7%	67.0%	77.1%	92.2%	-	-
AdaBoost	-	98.7%	73.0%	83.9%	94.6%	-	-

Score exhibit similar characteristics to the previous metrics, indicating a minor impact from negative classes.

Table II also shows the best results for intrusion detection using *strace* data from a previous study [7]. The higher recall indicates that detection using *sysdig* presents a lower number of false negatives in comparison to *strace*. However, the lower values for the *sysdig* precision reflect a higher impact of false positives in the models. The F1Score demonstrates that both data sources have acceptable results for an anomaly detection strategy, with MLP having the best performance for the *sysdig* dataset. Detection using *sysdig* traces performs worse in terms of accuracy than the *strace* data, but with better recall and F1Score. Overall, both *strace* and *sysdig* data can be effective for anomaly detection, without a clear winner in terms of classifier performance.

Despite the models exhibiting a margin of error, it is crucial to emphasize that we are exploring the potential of our proposal. The results for *sysdig* data come from unoptimized classifiers, meaning that there is room for improving the models. The results obtained so far are promising for leveraging *sysdig* for anomaly-based intrusion detection.

V. RELATED WORK

The literature encompasses numerous works focused on system call-based anomaly detection in the context of containers, as highlighted in Table III. These proposals have received significant attention in recent years, creating solutions that employ various techniques for anomaly detection via system calls. The outcomes of these solutions vary widely, but most of them demonstrate promising potential to enhance the security of computing systems.

TABLE III
RELATED WORK USING *sysdig* AND/OR *strace*.

Reference	Tool	Description
[14]	Strace	From the point of view of the host, the BoSC technique is applied for anomaly detection.
[16]	Strace/ Sysdig	STIDE and BoSC technique for profiling Docker containers.
[7]	Strace	Machine learning techniques were used to detect anomalies in the Docker container from a host perspective view. Also, a discussion of different perspectives of view of virtualized environments is presented.
[17]	Sysdig	A framework for intrusion detection in Kubernetes clusters.
[18]	Sysdig	A clustering strategy for anomaly detection in containers is presented.

The main differences between our research and previous work are *(i)* a comparison of the effectiveness of *strace* and *sysdig* data for anomaly detection based on the analysis of system calls, *(ii)* the development of a publicly available *sysdig* dataset, and *(iii)* an evaluation of the detection performance of five different ML models using *sysdig* data.

VI. CONCLUSION

This paper presented an intrusion detection strategy based on data collected by *sysdig*. We show a comparison between our *sysdig* proposal and previous work using *strace* [7]. To evaluate the strategy we built a dataset based on *sysdig* trace collection, which was evaluated with different ML algorithms and shown to be feasible to detect anomalies in containerized virtual environments. This monitoring approach is less intrusive to the system and allows collecting the data with lower overhead when compared with *strace*.

For future work we will evaluate whether results can be improved using data categorization and risk assessment of system calls [23], [31]. As *sysdig* can provide additional data about the execution of system calls, we will also investigate if leveraging these data enhances our results.

ACKNOWLEDGEMENTS

This work was partially supported by the C3SL - Center for Scientific Computing and Free Software (https://www.c3sl.ufpr.br/).

This study was financed in part by the *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior — Brasil (CAPES)* — Finance Code 001 and *Fundação de Amparo à Pesquisa e Inovação do Estado de Santa Catarina (FAPESC)*.

The authors also thank the UDESC, UFPR and UTFPR Computer Science departments.

REFERENCES

[1] S. Salagrama and V. Bibhu, "Study of IT and data center virtualization," in *Proceedings of the 2nd International Conference on Innovative Practices in Technology and Management*, vol. 2. Gautam Buddha Nagar, India: IEEE, 2022, pp. 274–278.

[2] M. Singh, "Virtualization in cloud computing – a study," in *Proceedings of the International Conference on Advances in Computing, Communication Control and Networking*. Greater Noida, India: IEEE, 2018, pp. 64–67.

[3] V. Varadharajan and U. Tupakula, "Security as a service model for cloud environment," *IEEE Transactions on Network and Service Management*, vol. 11, no. 1, pp. 60–75, 2014.

[4] M. Abourezq and A. Idrissi, "Database-as-a-service for big data: An overview," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 1, 2016.

[5] V. Fulber-Garcia, A. Huff, C. R. P. dos Santos, and E. P. Duarte Jr, "Network service topology: Formalization, taxonomy and the CUSTOM specification model," *Computer Networks*, vol. 178, p. 107337, 2020.

[6] M. Eder, "Hypervisor- vs. container-based virtualization," *Future Internet (FI) and Innovative Internet Technologies and Mobile Communications (IITM)*, vol. 1, 2016.

[7] G. R. Castanhel, T. Heinrich, F. Ceschin, and C. Maziero, "Taking a peek: An evaluation of anomaly detection using system calls for containers," in *Proceedings of the Symposium on Computers and Communications*. Athens, Greece: IEEE, 2021, pp. 1–6.

[8] S. Sultan, I. Ahmad, and T. Dimitriou, "Container security: Issues, challenges, and the road ahead," *IEEE Access*, vol. 7, pp. 52 976–52 996, 2019.

[9] W. Yassin, N. I. Udzir, Z. Muda, M. N. Sulaiman *et al.*, "Anomaly-based intrusion detection through k-means clustering and Naive Bayes classification," in *Proceedings of the 4th International Conference on Computing and Informatics, ICOCI*, Kuching, Sarawak, Malaysia, 2013.

[10] A. El Khairi, M. Caselli, C. Knierim, A. Peter, and A. Continella, "Contextualizing system calls in containers for anomaly-based intrusion detection," in *Proceedings of the Cloud Computing Security Workshop*. Los Angeles, CA, USA: ACM, 2022, pp. 9–21.

[11] A. S. Abed, M. Azab, C. Clancy, and M. S. Kashkoush, "Resilient intrusion detection system for cloud containers," *International Journal of Communication Networks and Distributed Systems*, vol. 24, no. 1, pp. 1–22, 2020.

[12] R. Jolak, T. Rosenstatter, M. Mohamad, K. Strandberg, B. Sangchoolie, N. Nowdehi, and R. Scandariato, "CONSERVE: A framework for the selection of techniques for monitoring containers security," *Journal of Systems and Software*, vol. 186, p. 111158, 2022.

[13] *strace(1) – Linux manual page*, Linux, 2024, https://man7.org/linux/man-pages/man1/strace.1.html.

[14] A. S. Abed, T. C. Clancy, and D. S. Levy, "Applying bag of system calls for anomalous behavior detection of applications in Linux containers," in *Proceedings of the GLOBECOM Workshops*. San Diego, CA, USA: IEEE, 2015, pp. 1–5.

[15] *sysdig*, Sysdig, 2024, https://github.com/draios/sysdig.

[16] J. Flora and N. Antunes, "Studying the applicability of intrusion detection to multi-tenant container environments," in *Proceedings of the 15th European Dependable Computing Conference*. Naples, Italy: IEEE, 2019, pp. 133–136.

[17] S. L. Rocha, G. D. A. Nze, and F. L. L. de Mendonça, "Intrusion detection in container orchestration clusters: A framework proposal based on real-time system call analysis with machine learning for anomaly detection," in *Proceedings of the 17th Iberian Conference on Information Systems and Technologies*. Madrid, Spain: IEEE, 2022, pp. 1–4.

[18] J. Shen, F. Zeng, W. Zhang, Y. Tao, and S. Tao, "A clustered learning framework for host based intrusion detection in container environment," in *Proceedings of the International Conference on Communications Workshops*. Seoul, Republic of Korea: IEEE, 2022, pp. 409–414.

[19] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, pp. 1–58, 2009.

[20] V. Jyothsna, R. Prasad, and K. M. Prasad, "A review of anomaly based intrusion detection systems," *International Journal of Computer Applications*, vol. 28, no. 7, pp. 26–35, 2011.

[21] M. Ahmed, A. N. Mahmood, and J. Hu, "A survey of network anomaly detection techniques," *Journal of Network and Computer Applications*, vol. 60, pp. 19–31, 2016.

[22] M. Mitchell, J. Oldham, and A. Samuel, *Advanced Linux Programming*. New riders Berkeley, 2001.

[23] M. Bernaschi, E. Gabrielli, and L. V. Mancini, "REMUS: A security-enhanced operating system," *ACM Transactions on Information and System Security (TISSEC)*, vol. 5, no. 1, pp. 36–61, 2002.

[24] M. Rajagopalan, M. A. Hiltunen, T. Jim, and R. D. Schlichting, "System call monitoring using authenticated system calls," *IEEE Transactions on Dependable and Secure Computing*, 2006.

[25] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, "A sense of self for Unix processes," in *Proceedings 1996 IEEE Symposium on Security and Privacy*. IEEE, 1996, pp. 120–128.

[26] M. Liu, Z. Xue, X. Xu, C. Zhong, and J. Chen, "Host-based intrusion detection system with system calls: Review and future trends," *ACM Computing Surveys*, vol. 51, no. 5, p. 98, 2018.

[27] M. Ozkan-Okay, R. Samet, Ö. Aslan, and D. Gupta, "A comprehensive systematic literature review on intrusion detection systems," *IEEE Access*, vol. 9, pp. 157 727–157 760, 2021.

[28] R. Lemos, T. Heinrich, C. A. Maziero, and N. C. Will, "Is it safe? identifying malicious apps through the use of metadata and inter-process communication," in *Proceedings of the 16th Annual IEEE International Systems Conference*. Montreal, BC, Canada: IEEE, 2022.

[29] R. Lemos, T. Heinrich, N. C. Will, R. R. Obelheiro, and C. A. Maziero, "Inspecting Binder transactions to detect anomalies in Android," in *Proceedings of the 17th Annual IEEE International Systems Conference*. Vancouver, BC, Canada: IEEE, 2023.

[30] A. Armando, A. Merlo, and L. Verderame, "An empirical evaluation of the Android security framework," in *Proceedings of the 28th International Information Security Conference*. Auckland, New Zealand: Springer, 2013, pp. 176–189.

[31] T. Heinrich, N. C. Will, R. R. Obelheiro, and C. A. Maziero, "A categorical data approach for anomaly detection in WebAssembly applications," in *Proceedings of the International Conference on Information Systems Security and Privacy*. Rome, Italy: SCITEPRESS, 2024.

[32] E. Johnson, E. Laufer, Z. Zhao, D. Gohman, S. Narayan, S. Savage, D. Stefan, and F. Brown, "WaVe: a verifiably secure WebAssembly sandboxing runtime," in *Proceedings of the Symposium on Security and Privacy*. San Francisco, CA, USA: IEEE, 2023, pp. 2940–2955.

[33] *ptrace(2) – Linux manual page*, Linux, 2024, https://man7.org/linux/man-pages/man2/ptrace.2.html.

[34] F. Ceschin, H. M. Gomes, M. Botacin, A. Bifet, B. Pfahringer, L. S. Oliveira, and A. Grégio, "Machine learning (in) security: A stream of problems," *arXiv preprint arXiv:2010.16045*, 2020.

[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.