Securing Microservices Against Password Guess Attacks using Hardware Performance Counters

Sai Praveen Kadiyala[1*], Xiaolan Li[2*], Wonjun Lee [3†], Andrew Catlin [4*]
[*]*Katz School of Science and Health, Yeshiva University*, New York
[†]*Department of Computer Science and Engineering, California State University*, California
{[1]ksp463, [2]xialoancara}@gmail.com, [3]wonjun.lee@csun.edu, [4]andrew.catlin@yu.edu

2022 IEEE 35th International System-on-Chip Conference (SOCC) | 978-1-6654-5985-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/SOCC56010.2022.9908109

***Abstract*—Modern customer-facing applications need to be easy to use, localizable, and to scale out to serve large customer bases. Microservice architectures have the potential to decentralize functionality, improve flexibility, and provide faster time to market of incremental changes. However, applications implemented as microservices also have a larger surface area, which may make them more prone to cyber attacks. Modern operating systems provide performance counters which are tamper-resistant, and can be used to track the run-time behavior of applications. In this work, we aim to detect a password guess attack on microservice using performance counter data. Our approach consists of modelling behavior of normal and attack user login requests, identification of key performance counters that effectively distinguish these requests and developing a machine learning model that classifies unknown login requests. A fully connected neural network-based classification model gave us 98.3% test accuracy in detecting the attacks with a false negative rate of 1.6%.**

***Index Terms*—microservices, hardware performance counters, password guess attack, machine learning**

I. Introduction

Designing an application system as a collection of independently deployable services with a lightweight communication mechanism is often referred as microservice architecture. These architectures offer smaller and self contained microservices as opposed to heavy monolithic architectures giving advantages in terms of scalability, error isolation, ease of development and deployment. These microservices are often deployed in containers which may share a common kernel and hardware as shown in Fig. 1. Containers keep the system more lightweight than virtual machine (VM) based deployments, in which each VM has its own operating system. Software applications like Netflix, eBay, PayPal have transitioned from using monolithic architectures to microservice architectures [1]. Owing to the large attack surface these architectures are often more prone to cyber attacks [2]. To ensure safety of microservices, there needs to be an efficient means of detecting such attacks.

Performance counters are low-level features that record the footprint of a given process using hardware features. From Fig. 1, it can be observed that the hardware information is low-level which is more difficult to tamper with compared to high-level process information like API calls [3]. Using these performance counters we can model the dynamic behaviour of given microservice over a selected duration. An attack on a microservice can cause a deviation in its normal behaviour which can be captured by observing the corresponding performance counter data. Modelling such attacks as well as normal behaviours can help security researchers efficiently classify zero day behaviour of microservices.

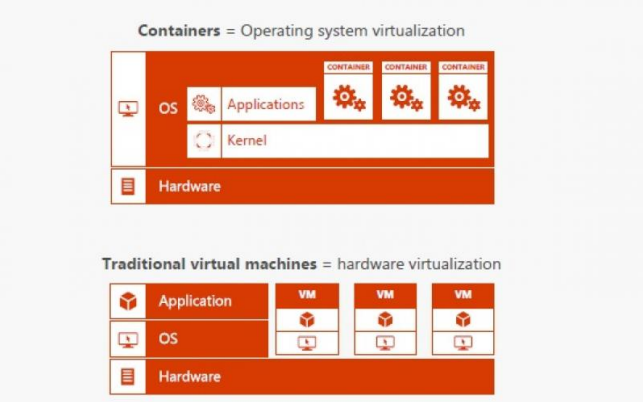


Fig. 1. Achieving Virtualization using Containers and Virtual Machines

In this work, we use process-specific data collected from performance counters to monitor the microservice behaviour, specifically to identify password-guess attacks. We generate user request traffic for a specific microservice both for normal login and password guess attack scenarios. For each of these normal and attack patterns, hardware performance counter (features) data is collected. Using statistical methods we identify the key features that can efficiently distinguish between normal and attack patterns. From data of identified key features for both normal login and password guess attack patterns we train a machine learning model that could further identify the class of an unknown or zero-day pattern. To the best of our knowledge, our work is the first attempt to identify password guess attacks on microservices using performance counter data.

The rest of the paper is organized as follows. Section 2 presents some of the latest work done in the field of security for microservices. Our proposed framework for microservice security using low level performance counters is presented in Section 3. Choice of test bench, details of experiments carried

The work reported in this paper has been partially supported by USDA NIFA under the grant 2020-67021-32476.

out and analysis of obtained results are presented in Section 4. Section 5 concludes with our findings and with some possible directions to carry forward this work.

II. Related Work

Detecting anomalies using microservice calls is carried out by Jacob, et al. [4]. In this work, monitoring and logging of various functionalities of applications are carried out with the help of distributed tracing. The work considered features as microservice call sequences specific to a given function but not the entire environment. However, these are high-level features and are vulnerable to attacks. Anomaly detection approach using comparison of execution traces for microservice application is discussed in [5]. In this work Meng, et al. have used dynamic instrumentation to collect execution traces of applications and used call trees to describe these execution traces. Although the authors in this work have identified the components causing the anomalies, to extract the traces they need to go for dynamic instrumentation which needs to tamper the executable. A summary of various security aspects of microservices is presented by Mateus-Coelho, et al. [2]. A taxonomy of microservices security is presented in [1] assessing various security implications of microservices architecture and the contemporary solutions available for it. In this work, authors present important insights into Docker Swarm and Netflix security decisions in the context of microservices. Pahl, et al. [6] present IoT as Service Oriented Architecture (SOA) of microservices. As opposed to the standard policy based security for IoT, authors in this work present a graph-based access control technique that runs as a module on various IoT nodes.

In summary, these recent works in microservice security aim at providing a taxonomy of security for microservices and present the security models involved. Some works performed the anomaly detection and identification of components that cause the anomalous behavior. One work [4] characterizes the attacks on microservices. However, this work used high-level microservice calls as features which are easy to tamper. These factors motivate us to model the microservices behaviour using low-level features, namely hardware counters that are difficult to be tampered. These features can be accessed using Performance Management Unit(PMU) and a judicious choice of features is necessary to offer lightweight, yet effective attack detection approach. In the following section, we present our framework for modelling the password guess attack on microservices using the low-level performance counter data.

III. Microservice Security Framework

In this Section, we describe the methodology used for detecting password guess attack on microservices using low-level features. Approaches for generating the attack data, identifying key features that differentiate between a normal login and a password guess attack and the process used for classifying the unknown login request are elaborated. We also describe the benchmark and the password guess attack considered for validating our approach.

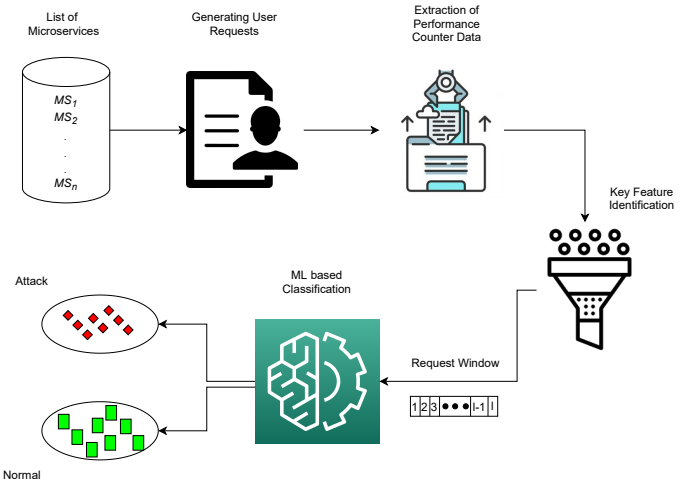


Fig. 2. Framework for securing microservices using low level performance counters

A. Generating User Requests

The proposed framework for detecting password guess attack on microservices is shown in Fig. 2. Given a large scale application that is using microservice architecture, it is divided into sub-functions each of which is carried out by a separate microservice. As shown in Fig. 2, let us assume an application can be split into n microservices $\{MS_1, MS_2, \ldots MS_n\}$. We choose a particular microservice $MS_i, i \in n$ for our analysis. Consider an attack (*att*) that is to be carried on MS_i. For the attacker to carry out *att*, he needs to provide some inputs for the execution of MS_i, that can be called from a separate database. Similarly, inputs are needed for the normal operation of MS_i which can also be called from a database. Using these inputs we generate normal and attack requests for microservice MS_i.

B. Extracting Hardware Performance Counter Data

After creating the user requests, we need to build a system that can profile a given microservice when the requests are generated. We do this profiling using low-level hardware performance counters. A number of low-level hardware performance counters exist for a given environment. As explained in Section I, these hardware performance counter data correspond to the dynamic behaviour of a given process. The performance counter values are extracted using the standard *perf* tool as described in [7]. Consider there exist m hardware performance counter features in a given environment. We represent these as $f_1, f_2, \ldots f_m$. We consider each of these performance counters as separate feature for our analysis. Each performance counter measures a particular type of activity corresponding to a given process. When an attack is carried out, owing to the resources affected by the attack (e.g. network, memory, I/O operations etc.) a set of performance counters show deviation. Hence, for our work, there is a need to identify the set of key features that are helpful in detecting password guess attack. A list of hardware and hardware cache performance counters present

in Linux and Windows environments are listed in [3] and [8] respectively. In the following subsection we elaborate the method we used for identifying the key features.

C. Feature Pruning

As mentioned in Section III-B, owing to the nature of password guess attack, there will be deviations in certain hardware performance counters. Hence we need to identify k key features from the available set of m overall features, that show significant deviation when an attack is carried out. To achieve this objective, we conduct a statistical test which computes the correlation of performance counter values for various features collected over a set of *r* samples in which *p* samples belong to the normal class (*N*) and *q* samples belong to the attack class (*A*) as shown in Table I

In Table I, θ^i_j represents the performance counter value of i^{th} feature for j^{th} sample. The total number of considered samples for this test are $r = p + q$. We compute the Pearson correlation coefficient [9], between data vector collected from each feature f_i and the label vector L as per the formula mentioned in Equation (1).

$$\rho_i = \frac{\sum(\theta^i_j - \bar{\theta^i})(L_j - \bar{L})}{\sqrt{\sum(\theta^i_j - \bar{\theta^i})^2 \sum(L_j - \bar{L})^2}} \tag{1}$$

In Equation (1), $\bar{\theta^i}$ represents the mean value of all j observations from feature f_i and $\bar{L}$ represents the mean value of all the elements in label vector L. After computing the correlation coefficient ρ_i for each feature f_i and the label vector L we arrange them in descending order and choose the top k features for our further analysis.

TABLE I
KEY FEATURE IDENTIFICATION

Counter 1 (f_1)	**Counter 2** (f_2)	...	**Counter m** (f_m)	**Label**
θ^1_1	θ^2_1	...	θ^m_1	N
θ^1_2	θ^2_2	...	θ^m_2	N
...	...	...	...	N
θ^1_p	θ^2_p	...	θ^m_p	N
θ^1_{p+1}	θ^2_{p+1}	...	θ^m_{p+1}	A
θ^1_{p+2}	θ^2_{p+2}	...	θ^m_{p+2}	A
...	...	...	...	A
θ^1_{p+q}	θ^2_{p+q}	...	θ^m_{p+q}	A

D. Machine Learning based Classification

We collect the performance counter data for the shortlisted k features. Next, we define a collection of successive user requests as a window (*W*), because a password guess attack is carried out by the attacker using a set of requests. We call this set of requests as attack window. If the number of requests in the set are *l*, then we consider the size of window as *l*. The nature in which the password guess is carried out by the attacker can vary. We translate this as variation in window composition.

The password guess attack (*A*), is specifically carried out on a microservice called *user login*. Whenever a user request is sent to this microservice, it will be a correct login if the correct credentials are provided. Otherwise it will result in an incorrect login. It is to be noted that due to human error, at times even normal users login incorrectly. It is important to include these incorrect logins in normal behaviour. On the other hand, most requests sent by an attacker will largely result in an incorrect login. In this work, as a comprehensive exploration, we consider both normal window and attack window have varying composition of correct and incorrect login requests. As shown in Fig. 2, we feed such windows instances to train our machine learning model. A separate set of instances are used for testing the model's efficiency to classify unknown windows. Our model will classify these test instances as either attack or normal. Details of variation in composition of windows, variation of window size, classification models that are explored are elaborated in Section IV

IV. EXPERIMENTAL RESULTS

In this Section, we elaborate on the benchmark we have chosen to validate our approach. We also present the details of various tools that we have used to generate the user request traffic, extract the performance counter data and implement the classification algorithms. We present and analyse the results of various experiments conducted in the course of our work followed by a comparative study.

A. Benchmark

To validate our proposed approach, we chose the DeathStarBench [10], an open-source benchmark suite. It consists of several applications which are based on microservice architecture. Some of the applications are **Social Networking**, where users can register, login and post messages; **eCommerce Site**, **Banking System** etc. The **Social Networking** application is chosen to validate our proposed approach.

B. Tools

We used the Linux perf tool [7], to extract the performance counters for the microservices of **Social Networking** application, specifically *user login* microservice. We used Python based http traffic generator to generate user requests for the *user login* microservice. A database of correct and incorrect login profiles are built and are used by these user requests so that the login attempts be classified as successful or unsuccessful. The number of login requests considered for training and testing are shown in Table II. We used Python 3.7 and PyTorch framework for implementing our machine learning and neural network models.

TABLE II
DATA SET

		Number of Requests
Training	Correct Login	35,000
	Incorrect Login	35,000
Testing	Correct Login	15,000
	Incorrect Login	15,000

C. Results and Analysis

In this section, we present the results of various experiments conducted and analyse them.

1) Key Features: A total of 18 hardware and hardware cache performance counters are available for extraction in our chosen environment [7]. In Section III-C, we have suggested a correlation based gradation for various features that are being considered. To perform this gradation, we have built normal and attack windows of size 10 requests each. The normal window consist of 8 correct login and 2 incorrect login requests and the attack window consists of 2 correct and 8 incorrect login requests. The position of correct and incorrect login request, in the window is set as random. We build 500 attack and normal windows. We collect the performance counter values for each of the 10 requests, with one counter at a time. The mean of performance counter values for all 10 requests of a given window is stored for further analysis. We get 1,000 such mean values (for 500 attack and 500 normal windows) for each of the counters and term these as mean vectors. We perform the correlation of this mean vector of each of the counters with the corresponding label vector (attack class is given label '0' and normal class is given label '1') and obtain *Pearson correlation coefficient* ρ. Correlation values for all 18 counters are listed in Fig. 3, in descending order. From the figure, we can observe that there is clear difference in the correlation values of the first five entries namely *L1-dcache-loads*, *branch-instructions*, *branch-loads*, *L1-dcache-stores*, *instructions* and the values for remaining 13 counters. The high correlation value indicate the effectiveness of each feature in correctly determining the label of the window. We chose the top 5 features for further analysis.

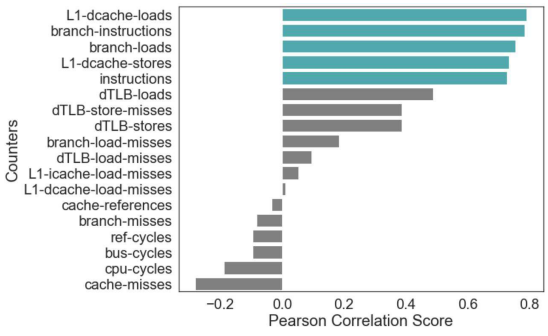
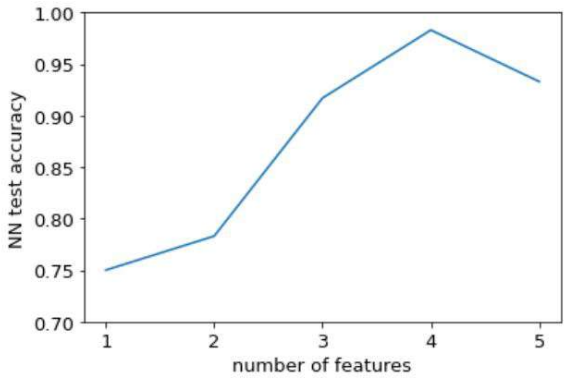
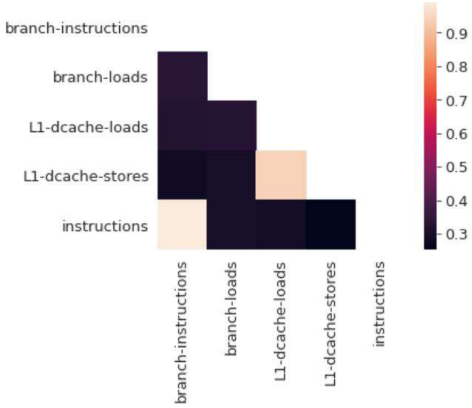


Fig. 3. Correlation between label vector and mean value vector for all counters

2) Choice of Features: In Section IV-C1, we identified five features that are individually effective in distinguishing between normal and attack windows. We used a fully-connected neural network model to classify the label of unknown window of requests. We carried out the training and testing of this model considering various combinations of the 5 key features identified earlier. In Fig. 4(a), we present the test accuracy of the fully connected neural network model using various number of features. We observe that the maximum accuracy 98.3% is achieved when the number of features is four for the combination *L1-dcache-loads*, *branch-instructions*, *branch-loads*, *L1-dcache-stores*. When considering a smaller number of features, the amount of information used for training the model might be less resulting in lower test accuracy. When more than four features are considered, there is a dip in the test accuracy. The reason can be ascertained to the high internal correlation between the features *instructions* and *branch-instructions*. The same can be seen from Fig. 4(b). Hence, we chose to use the top four features from Fig. 3, namely *L1-dcache-loads*, *branch-instructions*, *branch-loads*, *L1-dcache-stores* features only.



(a) Variation of Test Accuracy with Feature Count



(b) Internal correlation between the key features

Fig. 4. Analysis for choosing the optimum features from the key features

3) Effect of Window Size and Composition: After selecting the final features for our classification we explored various classifiers to get the best accuracy. We used Logistic Regression, Random Forest, Naive Bayes, k-nearest neighbours, Support Vector Machine and fully connected neural network models. We obtained windows using the data mentioned in Table II. Out of these, 70% are used for training and 30% are set aside for testing each model. We varied both the composition of the window (number of correct and incorrect logins) and the size of the window (number of user request per window) for attack and normal labels. We considered window size 1, 4, 6, 8, 10, 20 and varied the composition for each window size. The test accuracies for the each of the above mentioned scenarios are shown in Fig. 5. As shown in Fig.

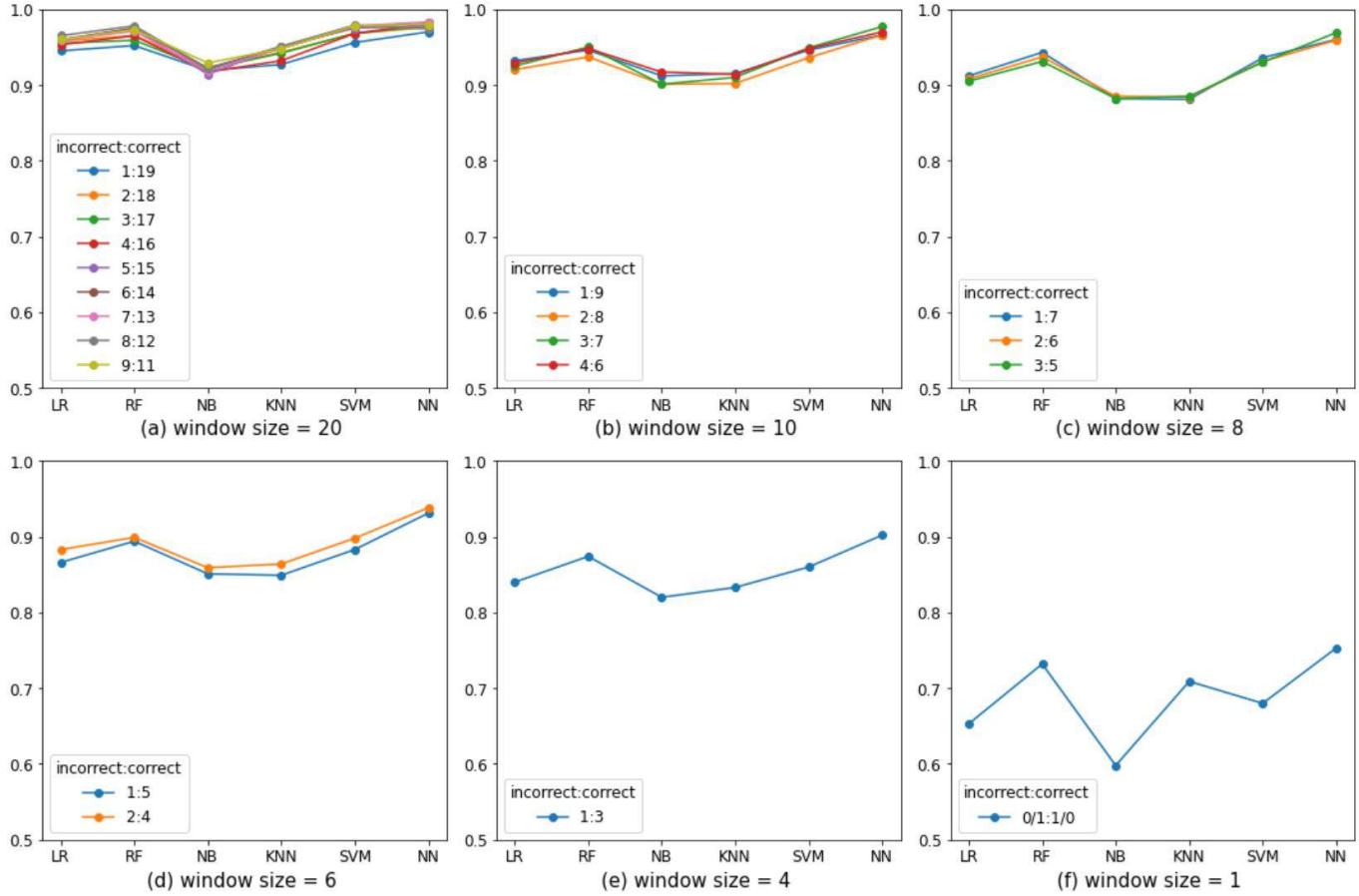


Fig. 5. Test Accuracy variation with windows size and window composition for the machine learning models Logistic Regression (LR), Random Forest (RF), Naive Bayes (NB), K Nearest Neighbours (KNN), Support Vector Machine (SVM), Neural Networks (NN)

5(f), the window size = 1 corresponds to a single user request (incorrect login:correct login is either 0:1 or 1:0). For this case, we can observe that test accuracy is lower than all other window sizes. This reconfirms our choice of modelling attack as a window as opposed to a single request. The maximum test accuracy reported is 98.3% in Fig. 5(a), for fully connected neural network with the window size 20 with 13 correct logins and 7 incorrect logins for the normal window and vice versa for attack window. From the Fig. 5, it can be observed that the testing accuracy increases with increase in window size. For smaller window size the amount of information fed to the model is low. Hence, this accuracy is maximum for window size 20. Also, when the composition of correct and incorrect login in the window is more balanced the test accuracy seems to be better.

D. Estimating the Overhead

As described in [3], [7] extraction of performance counter data for a given process involves a minimal overhead because these counters are already present in the system. The perf tools can be configured to collect the process-specific performance counter data in such a way that the collected data will not be effected by concurrently running programs in the system. When performance counter data are collected for multiple programs simultaneously there may be a degradation in data quality due to multiplexing among counters. However, in our case we collect the performance counter data for one process at a time. The 1.3%CPU usage and 2.3%MEM usage are collected using the Linux top command, to estimate the overhead of deployed classification algorithm. An increment of 35.7%CPU usage and 3.2% MEM usage were reported. While the 0.9%memory usage increased minimally, the increase in CPU usage is considerable. This can be due to the heavy weight nature of ML algorithms in general.

E. Comparative Study

In Table III, we present a theoretical comparative study of our work with some of the existing works on microservice security. Since the usage of microservice architecture is gaining popularity, we found a limited number of works in this domain. Many of these works present taxonomy of various advantages and potential threats when using the microservice architecture. Some of the other works discuss anomaly detection in microservices. They present different frameworks for microservice security against possible cyber attacks as shown in Table III. While various works used high-level features,

TABLE III
COMPARATIVE STUDY WITH STATE-OF-ART METHODS ON MICROSERVICE SECURITY

Approach	Feature Information	Key Points
Yargina *et al.* [1]	Token based Authentication	Mutual Authentication using MTLS, Principal Propagation using JWT
Jacob *et al.* [4]	Microservice API Call Frequency	High-level feature, Only statistical analysis
Meng *et al.* [5]	Execution Traces	High-level feature, Call Tree representation, Binary Instrumentation
Liebald *et al.*[6]	Graph based access control	High level Feature, IoT specific, Firewall inter service communication
Our approach	Hardware Performance Counters	Feature Pruning, Fully connected Neural Network, Determinism issues

machine learning based classification etc. as shown in Table III, in our work, we used low-level features which are difficult to tamper with as explained in Section I. Additionally, we have used a statistical method to identify key features that can effectively distinguish normal and attack scenarios. Finally our work uses a fully connected neural network which gives us a high test accuracy (98.3%) with considerable false positive rate (1.9%) and false negative rate (1.6%).

F. Threats to Validity

Inherent non determinism existing Hardware Performance Counters [11], their capability of identifying the minute deviations between normal and attack behaviours [12] are some concerns expressed in literature. From the works [13], [14], we understand that training models using performance counter data collected from multiple runs can help overcome the determinism related challenge. Also, from Fig. 3, we could understand the effectiveness of the collected performance counter data in distinguishing the normal and attack samples. This supports our idea of considering performance counters as a feature for modelling the password guess attacks microservices. In our work, we explored only the password guess attack. However, microservices are prone to a number of other cyber attacks like SQL injection, Denial of Service, spoofing etc. Effectiveness of Hardware Performance Counter data in modelling these attacks is yet to be verified and can be a part of future work.

V. CONCLUDING REMARKS

Security for microservices is necessary since they have more attack surface compared to monolithic services. Due to the increase in usage of microservice software architectures by various applications, our work is relevant to current trends. In this work, we have used the low-level, difficult to tamper performance counter data as features to model the dynamic behaviour of user login microservice. We detect the password guess attack using a fully connected neural network model and key hardware performance counters as features. Our best case resulted in 98.3% test accuracy with 1.9% false positive rate and 1.6% false negative rate. We would like to extend this analysis using a moving window average metric which can accurately represent the real world scenario. Also, validating our approach for other cyber attacks like Denial of Service, SQL injection attack and some spoofing attacks, usage of LSTM models for classification could be directions for future work.

REFERENCES

[1] T. Yarygina and A. H. Bagge, "Overcoming security challenges in microservice architectures," in *2018 IEEE Symposium on Service-Oriented System Engineering (SOSE)*. IEEE, 2018, pp. 11–20.

[2] N. Mateus-Coelho, M. Cruz-Cunha, and L. G. Ferreira, "Security in microservices architectures," *Procedia Computer Science*, vol. 181, pp. 1225–1236, 2021.

[3] S. P. Kadiyala, P. Jadhav, S.-K. Lam, and T. Srikanthan, "Hardware performance counter-based fine-grained malware detection," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 19, no. 5, pp. 1–17, 2020.

[4] S. Jacob, Y. Qiao, and B. A. Lee, "Detecting cyber security attacks against a microservices application using distributed tracing." in *ICISSP*, 2021, pp. 588–595.

[5] L. Meng, F. Ji, Y. Sun, and T. Wang, "Detecting anomalies in microservices with execution trace comparison," *Future Generation Computer Systems*, vol. 116, pp. 291–301, 2021.

[6] M.-O. Pahl, F.-X. Aubet, and S. Liebald, "Graph-based iot microservice security," in *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2018, pp. 1–3.

[7] A. C. De Melo, "The new linux'perf'tools," in *Slides from Linux Kongress*, vol. 18, 2010, pp. 1–42.

[8] S. P. Kadiyala, A. Kartheek, and T. Truong-Huu, "Program behavior analysis and clustering using performance counters," in *DYnamic and Novel Advances in Machine Learning and Intelligent Cyber Security (DYNAMICS) Workshop @ ACSAC*, 2020.

[9] J. Benesty, J. Chen, Y. Huang, and I. Cohen, "Pearson correlation coefficient," in *Noise reduction in speech processing*. Springer, 2009, pp. 1–4.

[10] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, A. Bruno, J. Hu, B. Ritchken, B. Jackson *et al.*, "An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 3–18.

[11] S. Das, J. Werner, M. Antonakakis, M. Polychronakis, and F. Monrose, "Sok: The challenges, pitfalls, and perils of using hardware performance counters for security," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 20–38.

[12] B. Zhou, A. Gupta, R. Jahanshahi, M. Egele, and A. Joshi, "Hardware performance counters can detect malware: Myth or fact?" in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, 2018, pp. 457–468.

[13] N. Patel, A. Sasan, and H. Homayoun, "Analyzing hardware based malware detectors," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2017, pp. 1–6.

[14] S. P. Kadiyala, M. Alam, Y. Shrivastava, S. Patranabis, M. F. B. Abbas, A. K. Biswas, D. Mukhopadhyay, and T. Srikanthan, "Lambda: Lightweight assessment of malware for embedded architectures," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 19, no. 4, pp. 1–31, 2020.