2019 15th European Dependable Computing Conference (EDCC)

Studying the Applicability of Intrusion Detection to Multi-tenant Container Environments

José Flora, Nuno Antunes
CISUC, Department of Informatics Engineering
University of Coimbra
Polo II, 3030-290 Coimbra – Portugal
jeflora@dei.uc.pt, nmsa@dei.uc.pt

***Abstract*—The use of containers in cloud-based applications allows for rapid and scalable deployments. Containers are lightweight and appealing to be used even in business-critical systems, but their use implies great security concerns, which are exacerbated in multi-tenant environments. To mitigate these concerns, techniques such as intrusion detection are a must, however, in the containers' context, it has received limited attention. Thus, it is necessary to define an improved approach to container-level intrusion detection for multi-tenant environments. In this paper we make a preliminary feasibility analysis of host-based container-level intrusion detection. For this, we are currently focusing on achieving a stable container profile definition and the results obtained show we are following the correct path.**

***Index Terms*—Containers, Security, Intrusion Detection, Multi-tenancy, Experimental Evaluation**

I. Introduction

There has been a growth on the use of containers in cloud deployments [1]. This technology permits to share a physical infrastructure over multiple customers in a lightweight and fast manner. However, this is only possible because containers share the kernel of the underlying Operating System (OS). Since cloud environments are multi-tenant environments, relying on technologies such as *cgroups* and *namespaces* to provide container isolation, the attack surface of applications there deployed, which are frequently business-critical, increased substantially. Therefore, there is a major necessity to deploy container-focused counter-measures on these infrastructures.

Intrusion detection has been widely used across multiple contexts, however, its application to containers is still sparse. This remedy, correctly applied, is a *must-have* so that intrusions are detected and mitigation or remediation measures are put in place. As a result, we aim to develop a functional host-based and anomaly-based Intrusion Detection System (IDS), whereby containers are profiled during its normal execution periods and then monitored for anomalous events, such as *zero-day* or known attacks.

To the best of our knowledge, limited work has been done with regard to intrusion detection at container level. In 2015, Abed *et. al* [2] proposed an IDS for container level, which used Bags of System Calls (BoSC) to represent the normal behaviour database entries. In this work, an anomaly detection based IDS was proposed, capable of monitoring a container and detect possible intrusion attempts at real time. During the experimental analysis, the authors claimed to have obtained a True Positive Rate (TPR) of 100% with a False Positive Rate (FPR) of 2%. In 2019, Srinivasan *et. al* [1] proposed a real-time IDS that uses n-grams of system calls and the probability of its occurrence. The authors used the University of New Mexico (UNM) dataset to validate the approach and claim to have obtained an accuracy ranging from 87–97%.

In this paper, we are focused on understanding if intrusion detection techniques are applicable to multi-tenant container environments, using anomaly detection algorithms, such as Sequence Time-Delaying Embedding (STIDE) [3] or BoSC. Therefore, we aim to define stable profiles for the containers under monitoring using the system calls, by them executed, to train the referred algorithms. We were able to observe the applicability of both STIDE and BoSC in the context of intrusion detection, and conclude that both algorithms have a promising capability to define stable profiles for containers, either Docker or LXC.

The structure of the paper is as follows. Sec. II states background and motivation, Sec. III focuses on the scope, preliminary architecture, and presents the results and discussion, and Sec. IV outlines the next steps and concludes.

II. Background and Motivation

Cloud-based intrusion detection has seen some developments, in terms of deployment and monitoring mode, for instance the use of distributed IDSes. Despite great advances in intrusion detection for Virtual Machines based environments, container-level approaches have been neglected and improvements in this context are still sparse.

Intrusion Detection is the process of monitoring the events occurring in a computer system or network and analysing them for signs of possible incidents [4], and is accomplished by IDSes. While network-based IDSes process more data and are typically placed at the network's perimeter, thus monitoring a broader area, host-based IDSes are placed on machines, present in a network, monitoring them and therefore having a more local view of the events.

In addition, IDSes can follow two main different approaches to achieve its purpose. The **signature based** approach consists of known malicious patterns identification when analysing new events, such as network traffic or application data [5]. This approach results in a low false positive rate but is unable to detect newly crafted attacks. While **anomaly based** implies

978-1-7281-3929-6/19/$41.00 ©2019 IEEE
DOI 10.1109/EDCC.2019.00033

two phases, a **training phase**, to build the profile, and a **detection phase**, where new events are evaluated based on the defined profile [6]. Therefore, this approach is able to detect new attacks, although, it may produce a higher rate of false positives.

This typically implies the use of algorithms that range from statistical methods to machine learning and data mining approaches, whose usage has proved to be helpful in the field of intrusion detection. Artificial Neural Networks (ANNs) is a methodology that has the ability to generalise data, thus being able to detect intrusions from incomplete training data [7]. Hidden Markov Models (HMM) are used to describe the system being modelled through a set of weighted-connections between each pair of states, which represent the probability of transitioning from one to the other [8]. The application of K-Nearest Neighbour (KNN) to intrusion detection has also seen some positive results, it classifies events based on distance heuristics [9]. The STIDE method is based upon a window sliding over a sequence of system calls [3]. Support Vector Machines (SVM) is a method that on its essence divides events into two different classes [7]. Usually, in anomaly detection, it is utilised a SVM variation called One-Class Support Vector Machines (OCSVM), that classifies data into only one class, thus removing the need to provide anomalous data to the classifier when in training phase.

Currently, the main challenge is the fact that **various containers, with different owners, cohabit in the same machine** which raises the possibility that some tenants, with malicious intentions, may try to compromise the normal execution, of either neighbours or the host machine. Thus,we assume that containers with malicious intentions, *initially acquire resources in a lawfully and non-abusive manner*. That is, only after obtaining access to the container, its owner utilises their resources to perform mischievous activities, such as trying compromise other tenants' execution. The fact that cloud computing services assume their consumers are trustworthy, allows an attacker to allocate resources, as any other customer, and use them. So, in this work, we assume that an agent obtains access to computing resources within a cloud service lawfully and, only after, uses them to carry out illegal and malicious endeavours.

Therefore, our concerns rely upon the threats presented by these legally allocated resources, that originally are considered trustworthy despite the possibility of having malicious intention towards other tenants or the cloud service infrastructure. Which in fact stands a great menace to the underlying layers of the cloud service stack and to the normal execution of other honestly obtained containers without wicked motivations.

III. Intrusion Detection in Containers Environment

This work emerges as a result of imminent threats to container-based cloud deployments, which are depicted in Fig. 1, considered the threat model in this work.

In this figure, it is possible to observe a malicious container, sharing the infrastructure with non-malicious one, and the

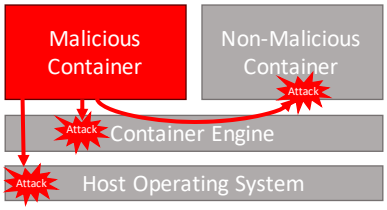

Fig. 1. Container-based cloud deployments' threat model.

attack venues the malicious one can take as a manner of compromising the infrastructure or other containers. Generally, every attack aims to compromise the infrastructure or its components, and gain some type of reward, such as more resources or unauthorised access to information.

Regarding the attack to the host machine, a malicious container may try to exploit vulnerabilities present in the cloud's stack in order to attain access or gain control of privileged resources. Typically, these attacks are motivated by the ambition of collecting information or controlling the manner by which the resources are allocated.

Containers are commonly managed by an orchestration middleware, called a container engine. This also acts as an attack venue to achieve the OS layer, thereby this piece of software may also be compromised in cloud deployments, by rogue containers, assuming a pathway role to achieve both attacks, that is to the host machine and to other tenants. Hence, a malicious container would need to compromise the container engine in order to make damage reach other containers or the host itself. The attacks to the engine are a great concern owing to the damage that can come from it being compromised.

Therefore, the main focus of this work is to detect intrusion attempts against containers, which are deployed in a multi-tenancy environment thus being subjected to the inherent risks of sharing physical resources and not having the control of them. This work aims to act as an improvement to the state of the art in intrusion detection at the container level, due to its use growth on cloud services and also to aid with isolation guarantees.

A. Preliminary Architecture for Intrusion Detection

In this section, we provide an overview of the preliminary architecture for the intrusion detection approach, depicted in Fig. 2, which consists of three main components: the system under monitoring, a data collection tool and the IDS analyser.

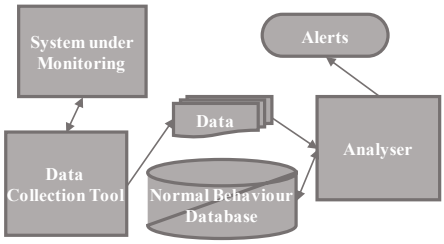

Fig. 2. Preliminary architecture proposed for the intrusion detection approach.

With respect to the system under monitoring, our focus remains on containers, that is, the intrusion detector is expected to build a profile of a container based on the system calls issued by it to the underlying OS.

There are various containerisation technologies. For now, we are focusing on Docker and LXC, however, we aim to have a platform that is able to work in an agnostic manner to containers technology.

To fulfil the data collection component, we selected two tools. Firstly, we have elicited *strace* [10], which is able to attach to processes running on a machine, monitoring and collecting the system calls by them invoked along with the arguments passed to it. Secondly, we selected the *sysdig* [11] utility, since it is a container monitoring and auditing specialised tool. It allows the collection of traces from containers containing data such as system calls and other OS events.

In sum, both tools are capable of collecting system calls and other OS events, however, *strace* is process-oriented while *sysdig* is container-oriented, which means that in our case *sysdig* might be the best choice.

The analyser component is the main unit of the IDS we aim to construct. Consequently, we want to select the algorithms with better performance for the container-level context, which may be used independently or combined as an ensemble. For this, we have either collected or implemented algorithms used for intrusion detection. In this paper, we focused on the algorithms we have implemented, STIDE and BoSC, due to its high adoption and proven effectiveness [3] [12].

Moreover, we aim to evaluate the intrusion detection state-of-the-art algorithms, using multiple realistic scenarios, in the context of container-level intrusion detection. These experiments are still in its infancy since there are no available data-sets for container intrusion detection. Therefore, we need to produce a representative and meaningful data-set to perform a valid set of experiments.

B. Preliminary Results

In this section, we present the preliminary results regarding the training of the STIDE algorithm used for intrusion detection. Firstly, using its usual data representation, a window extracted from a sequence of system calls, and using the frequency-based BoSC representation.

We collected benign traces from Docker and LXC containers running MySQL server application which received the workload produced by an implementation of the TPCC On-Line Transaction Processing Benchmark [13], configured with 100 warehouses and using 50 clients during workload execution. We conducted two runs of collections for 10 and 24 hours of collection time period.

After this collection was completed, we were able to analyse the traces produced for both LXC and Docker container. The observation of Table I allows to conclude that LXC containers produce longer traces based on a greater sets of unique system calls *per* dataset.

Afterwards, we trained classifiers of the STIDE and BoSC methods, with window size ranging from 3 to 6 system calls

TABLE I
ANALYSIS OF COLLECTED TRACES.

Training Time	Platform	Run 1		Run 2	
		Unique	Total	Unique	Total
10H	Docker	37	1,303,036,648	29	684,693,069
	LXC	116	3,279,400,309	127	2,962,761,415
24H	Docker	37	1,907,494,529	37	3,533,706,363
	LXC	129	9,617,350,813	129	10,305,731,175

in each. We computed the slope value of the growth curve, which, in this context, represents the rate of new combinations (windows) of system calls added to the classifiers' normal behaviour database during a period of time.

For this, we used the formulae $0 \leqslant \frac{S_{t_{s2}} - S_{t_{s1}}}{t_{s2} - t_{s1}} \leqslant \sigma$ [14] to decide whether an interval is at learning steady-state. In addition, we consider that the steady-state of the learning procedure is achieved when the inequality above is satisfied 5 times in a row for $\sigma = 0.15$, based on previous experiments conducted on [14]. The results from these experiments are presented in Tab. II, Docker on the left and LXC on the right.

Fig. 3 presents a visual display for the procedure executed. In this figure, the blue dots represent the moments were the slope is below σ, green dots represent the four preceding steady-state intervals to the interval were the classifiers reaches the learning steady-state, represented by the red dot.

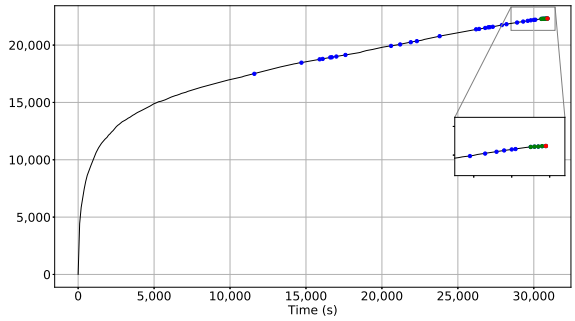

Fig. 3. Training procedure for STIDE with window 4 of run 1 of 24h collection for Docker container.

The analysis of the results produced, permits to conclude that in the case of STIDE, we were able to achieve a learning steady-state using window size of 3 and 4, with STIDE, and 3 to 6 when using BoSC. This means, that the best configuration in this case remains in the use of windows size of 3 and 4. Moreover, the fact that STIDE did not achieve steady-state with windows of size 5 and 6, in most cases, is due to the fact that is a sequence-based method. Therefore, the number of possible combinations for different windows is higher than for BoSC, which means that our threshold value might require some adjustment for this case

IV. LESSONS LEARNED AND FUTURE RESEARCH

Our following steps reside in the evaluation of state-of-the-art algorithms for intrusion detection. For this, we are focused on eliciting attacks and exploits, which will be used

TABLE II
DOCKER & LXC RESULTS FOR REACHING LEARNING STEADY-STATE.

Training Time	Algorithm	Window Size	Run 1 $T_{max}(s)$	Run 1 DB Size	Run 2 $T_{max}(s)$	Run 2 DB Size
10H	STIDE	3	3,000	2,582	3,000	2,345
		4	20,700	21,952	24,100	18,625
		5	-	-	-	-
		6	-	-	-	-
	BoSC	3	1,400	1,486	1,900	1,406
		4	6,800	6,091	6,400	5,280
		5	20,700	19,066	20,800	16,273
		6	-	-	-	-
24H	STIDE	3	3,000	2,674	2,700	2,428
		4	30,900	22,317	29,800	22,091
		5	-	-	-	-
		6	-	-	-	-
	BoSC	3	1,700	1,649	1,500	1,460
		4	5,800	5,898	6,800	5,801
		5	26,400	19,214	23,300	18,544
		6	34,200	43,942	54,900	49,151

Training Time	Algorithm	Window Size	Run 1 $T_{max}(s)$	Run 1 DB Size	Run 2 $T_{max}(s)$	Run 2 DB Size
10H	STIDE	3	3,900	3,177	6,800	3,283
		4	10,800	15,692	9,600	15,106
		5	-	-	-	-
		6	-	-	-	-
	BoSC	3	3,900	2,376	6,800	2,487
		4	5,500	5,554	6,900	5,526
		5	13,800	13,979	19,300	19,783
		6	-	-	28,200	39,513
24H	STIDE	3	6,800	4,271	4,600	3,490
		4	12,100	19,563	24,300	26,312
		5	77,700	152,366	-	-
		6	-	-	-	-
	BoSC	3	4,800	2,857	4,600	2,644
		4	6,800	6,970	17,400	10,728
		5	12,100	16,867	24,300	24,037
		6	35,600	42,830	25,200	42,019

$T_{max}(s)$ represents when learning steady-state is reached, DB Size is the number of entries of the normal database at that moment.

to construct the evaluation data-sets. Such attacks may follow different attack venues, some of the ones specified in Fig. 1. Therefore, evaluating the anomaly detection algorithms performance on the container-level intrusion detection context.

Furthermore, to continue the design of an intrusion detection approach, we are to produce a more detailed architecture than the one presented in Fig. 2. This architecture will contain specifications of its components as well as of their interactions, which will serve as basis for its implementation into a functional IDS. Regarding this implementation, it will be subject of evaluation so that we are able to extract meaningful metrics and compare its performance with other IDSes used in the same context.

The accomplishment of this work will allow to push forward the current state of the art of intrusion detection in container based cloud deployments. Moreover, the importance of this work is high as a result of the increasing presence of cloud service in the population quotidian. Also, the increase of cloud use for end-users as well as for enterprises, commonly for business-critical systems, which use third-party infrastructures to host their applications, makes it appealing to malicious users. These malevolent agents can steal confidential information of cause damage to businesses by simply prevent them from being reached by their costumers. Therefore, the increase in the quality of counter measures available for these systems are very welcomed as well as any advance in this context of information security is of great importance.

ACKNOWLEDGMENT

This work was partially supported by the **ATMOSPHERE** project (atmosphere-eubrazil.eu), funded by the Brazilian Ministry of Science, Technology and Innovation (Project 51119 - MCTI/RNP 4th Coordinated Call) and by the European Commission under the Cooperation Programme, Horizon 2020 grant agreement no 777154. It is also partially supported by the project **METRICS**, funded by the Portuguese Foundation for Science and Technology (FCT) – agreement no POCI-01-0145-FEDER-032504.

REFERENCES

[1] S. Srinivasan, A. Kumar, M. Mahajan, D. Sitaram, and S. Gupta, "Probabilistic Real-Time Intrusion Detection System for Docker Containers," in *Security in Computing and Communications*. Springer Singapore, 2019, pp. 336–347.

[2] A. S. Abed, C. Clancy, and D. S. Levy, "Intrusion Detection System for Applications using Linux Containers," *arXiv:1611.03056 [cs]*, vol. 9331, pp. 123–135, 2015.

[3] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, "A sense of self for unix processes," in *In Proceedings of the 1996 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 1996, pp. 120–128.

[4] K. A. Scarfone and P. M. Mell, "Guide to Intrusion Detection and Prevention Systems (IDPS)," National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep. NIST SP 800-94, 2007.

[5] A. Patcha and J.-M. Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," *Computer Networks*, vol. 51, no. 12, pp. 3448–3470, Aug. 2007.

[6] S. A. Hofmeyr, S. Forrest, and A. Somayaji, "Intrusion detection using sequences of system calls," *Journal of Computer Security*, vol. 6, no. 3, pp. 151–180, Jul. 1998.

[7] W.-H. Chen, S.-H. Hsu, and H.-P. Shen, "Application of SVM and ANN for intrusion detection," *Computers & Operations Research*, vol. 32, no. 10, pp. 2617–2634, Oct. 2005.

[8] C. Warrender, S. Forrest, and B. Pearlmutter, "Detecting intrusions using system calls: alternative data models," in *Proceedings of the 1999 IEEE Symposium on Security and Privacy (Cat. No.99CB36344)*. IEEE Comput. Soc, 1999.

[9] L. Li, H. Zhang, H. Peng, and Y. Yang, "Nearest neighbors based density peaks approach to intrusion detection," *Chaos, Solitons & Fractals*, vol. 110, pp. 33–40, May 2018.

[10] Michael Kerrisk, "strace - trace system calls and signals," http://man7.org/linux/man-pages/man1/strace.1.html, Oct. 2018, [Accessed: 2019-01-24]. [Online]. Available: http://man7.org/linux/man-pages/man1/strace.1.html

[11] Sysdig, Inc, "sysdig," https://sysdig.com/, 2019, [Accessed: 2019-01-24]. [Online]. Available: https://sysdig.com/

[12] 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 *2015 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2015, pp. 1–5.

[13] "Tpc-c benchmark," http://www.tpc.org/tpcc/, accessed: 2019-04-10.

[14] A. Milenkoski, B. D. Payne, N. Antunes, M. Vieira, S. Kounev, A. Avritzer, and M. Luft, "Evaluation of intrusion detection systems in virtualized environments using attack injection," in *International Symposium on Recent Advances in Intrusion Detection*. Springer, 2015, pp. 471–492.