Sliding Window: The Impact of Trace Size in Anomaly Detection System for Containers Through Machine Learning

Gabriel Ruschel Castanhel[1], **Tiago Heinrich**[1], **Fabrício Ceschin**[1], **Carlos A. Maziero**[1]

[1] Computer Science Department
Federal University of Paraná (UFPR)

{grc15,theinrich,fjoceschin,maziero}@inf.ufpr.br

Abstract. *Anomaly intrusion detection in Host-based Intrusion Detection System (HIDS) is a process intended to monitor operations on a host to identify behaviors that differ from a "normal " system behavior. System call based HIDS uses traces of calls to represent the behavior of a system. Due to the volume of data generated by applications and the operating system, sliding windows are applied in order to asses an online environment, allowing intrusions to be detected in real time while being still executed. The respective study explores the impact that the size of the observation window has on Machine Learning (ML) one-class algorithms.*

1. Introduction

An Intrusion Detection System (IDS) is responsible to perform the identification of attacks in a computerized environment. There are different techniques used to detect intrusion, some of them are based on: (i) signatures, where a dataset with known threats are used to compare new signatures and, despite of being widely used, it is still not capable of identifying threats that are not yet classified (unknown attacks); and (ii) normal behaviors, where the anomaly detection algorithm defines a normal behavior for the environment, and new unknown behaviors are classified into two classes: normal or anomaly [Yassin et al. 2013].

Host-based Intrusion Detection System (HIDS) has the focus on the host, with the ability to identify internal attacks using data from one or more host systems [Liu et al. 2018]. This approach is not limited to one environment, could be used in a distributed infrastructure, and could harness data from the network to better understand the environment. System calls based HIDS explore data collected from traces of the operating system or individual application, which could vary according to the approach [Liu et al. 2018]. The system call is responsible to make an interface between the operating system and the kernel, each call represents a basic operation [Mitchell et al. 2001].

A variety of strategies can be used to adapt the data to the models, one of the most used is the sliding window technique, where a window of n-size is used to scan the data. This way, one trace will be split in smaller traces that may be suitable to detect intrusions. Many researchers already tried to define the right size of a detection window, which may vary from five to eleven in many cases. However, as many ML parameters, the size of the detection windows may not be the always the same, given that an optimal choice of size will depend on the environment and data type [Liu et al. 2018].

Another point to consider is that many work in the literature consider the whole trace to detect an intrusion (offline detection). This approach, however, is not real given that an attack will only be analyzed after it finished executing. Thus, the IDS would not work as expected in the real world, which is the reason that sliding window technique is important (online detection). In theory, the smaller the window size, the faster an anomaly could be detected, with the trade-off that these windows may cut malicious actions in half, affecting their detection. Finally, selecting an ideal window size is important to detect attacks as soon as they start to be executed, without the need to watch the whole trace.

Aiming to explore the impact of window size in one-class ML methods, we selected a set of traces that were evaluated with different sizes, in a docker environment where, at the best of our knowledge, was not made yet in the literature. The main idea is to identify how the size of a window will impact the process of identifying an anomaly in one environment, helping to understand how the classifier will perform with this data.

This work is structured in five Sections. The Section 2 presents related work. The Section 3 presents the proposal and background for the study. The Section 4 presents the results and the Section 5 presents the conclusions.

2. Related Work

The literature presents approaches to detect the most adequate window size for the UNM [Systems 1998] and ADFA-LD datasets [Xie and Hu 2013], concluding that a size between six and seven were the best for them. Considering virtual machines the window size could diversify between six and ten [Liu et al. 2018]. The window size will impact the n-gram size created (a n-gram will represent a sequence of system calls extracted from the trace) and is responsible for splitting the trace into groups that will be used for training and testing.

Host-based anomaly intrusion detection with system calls is an active area in computer research. [Wang et al. 2006], for example, presents *Anagram*, a content anomaly detector that models a combination of high-order n-grams (n $>$1) designed to detect anomalous and "suspicious" network packet payloads. The work provides a comparison between approaches using different sizes of n-grams (or window sizes). The authors demonstrated that a semi-supervised strategy, using Bloom Filter, obtained 100% detection with 0.006% false positive rates in some cases.

[Forrest et al. 1996] introduced the use of system calls for anomaly detection years ago, where they proposed an approach using a look-ahead pair based technique, where each entry in the database represented a system call, and an immediate subsequence of system calls in a window of size n. The sliding window then moves by one position at a time to form the database. In their extended work, they proposed the Sequence Time Delay Embedding (STIDE) approach, based on the analysis of short sequence of the system calls, of which it was observed that fixed-length contiguous sequences have better-discriminating power than look ahead pairs. A sliding window of fixed size is used to produce the short sequences of system calls.

In a cloud environment focusing on virtual machines, the sequence of system calls can be used to anomaly detection across approaches like "Bag of system calls" [Alarifi and Wolthusen 2012]. The data is collected from a KVM virtualization

environment by trace tools. A window size of six is used, with 11.1% of false positives. Some problems with the dataset could be raised as the data collection period assume that the normal behavior will not be impacted by attackers.

3. Proposal

The literature presents limitations for anomaly detection in a container environment [Liu et al. 2018]. We try to explore the impact that a partial view of information could have in the results. Looking at how this data can be retrieved and applied in ML methods, we focus on how the window size will impact the detection of anomalies. The container is a technique of virtualization for applications, and are very popular nowadays because of tools like Docker. The virtualization happens in Operational System (OS) level, allowing the load and run of applications that have hardware managed made by the container [Merkel 2014]. This is possible because the container is responsible to gather all the necessary components in a single image.

Our dataset[1] was developed capturing data from a container environment, noting that the collection happens from the operating system point of view. This way, there is not partial view and limitations for the data capture [Castanhel et al. 2020]. The container was running WordPress 4.9.14, with Docker 19.03.11-ce, under the Linux 5.4.44-1-MANJARO. The data wrote on disk consists of system calls and signals issued by the container, containing the normal behaviors of the environment and anomaly behaviors that exploits Remote Code Execution (RCE) vulnerabilities [NVD 2020].

This defines a dataset with normal and anomaly behaviors that represent a container environment. This data is used to train and test two One-Class ML methods. The One-Class Classification (OCC) learn from only one class, different from other traditional methods that aim to learn two or more classes and may be difficult for the classifier considering the dataset used (in our case, a dataset with a lot of class imbalance: $367,342$ *system calls* representing normal behaviors and only $1,628$ *system calls* representing malicious ones)[Zhang et al. 2015].

The evaluation process reflects how window size growth will contribute to anomaly detection. The experiment consists of two study cases. In the first case, we explore how using a small portion of the trace will impact the anomaly detection. This case consists of using just 5% of the trace size to train and test the models. In the second case, the window growth will be not limited for just 5% of the trace size, instead, the growth will happen each time with 5% of the trace size until all the trace will be used.

Considering that the dataset has a set of traces with a great variety of sizes and we need a fixed window size for the ML methods to be trained and tested, the smallest trace size was considered for the size evaluation, i.e., 100% of the traces represents a trace size of $1,628$.

4. Experiments

The experiments used two types of dataset: one with raw data where no filter technique is applied and all the system calls sequence is used without any changes, and another where system calls classified as low level of threat were discarded. The system calls

[1]The dataset can be found at: `https://github.com/gabrielruschel/hids-docker`.

classification was based on [Bernaschi et al. 2002] work, and allow us to improve the results as harmless system calls are removed from the dataset.

Considering the two data approaches, the window evaluation aims to explain how windows with a small portion of the trace will impact the anomaly identification and how its grow will impact the classifier. Thus, two cases were defined for training and testing, which could represent these two types of window perspective.

For the smallest window size, a portion between zero and five percent of the total trace size was used, with a growth of 0.5% of the trace size. This investigation will represent most of the cases found in the literature [Liu et al. 2018], considering that 5% of the trace size will represent 81 system calls. Our goal here is not to stay limited by this size, but to check how a bigger trace size will behave in the second approach considering the window growth until the maximum size of the trace. The trace size will grow 5% of the total size until it reaches 100% of the trace size.

The evaluation considered two OCC algorithms: Isolation Forest and One-Class SVM. Both of them are adaptations of the multi-class classifiers Random Forest and SVM, respectively, trained only over the majority class (normal behavior). Thus, these classifiers generates decision boundaries that define what is normal, i.e., every sample that is out of these boundaries are considered as anomaly. Isolation forest is slightly faster to train than one-class SVM, given that it builds an ensemble with a set of random trees, which makes it more suitable to real applications, once it could react faster to changes [Liu et al. 2008]. One-class SVM, in the other hand, estimates the support vectors (instances that are near the decision boundary) of the training data, generating a hyper-plane that contains the normal behavior (which in practice takes much more time than generation random forests) [Zhang et al. 2015].

The Figure 1 presents the results for the Isolation Forest algorithm. Both raw and filter data reach reasonable results with small window size and variate over the growth size until reaching F1Score next to 100% after half of the trace percentage is used. The window size variation will be responsible for the 1% fall in the results, considering that this is not a huge impact in the results, we assume that would be possible to avoid this behavior with a more complex dataset. For the first 5% window size, the instant growth is more clearer on the partial graph view, where we can see that, with a window size of 1% of the trace size, we already achieve an F1Score of 95%.

Although the filter data suffer more with the growth window size, it is the first one to obtain an F1Score over 97%, with a window size of 1.5% of the trace (this represents a number of 24 system calls). The raw data will not suffer the same impact with the growth, being more stable with the size change.

The Figure 2 presents the results for the One-Class SVM algorithm. The graph shows a small variation among raw and filter data for a window size with 10% of the trace size, that will impact an F1Score between 98% and 99%, never reaching an F1score of 100%. Considering the first 5%, we can see a faster reach for the F1score where a window smaller than 0.5% of the trace size already reaches an F1score of 98%.

Comparing both results, it is interesting to appoint that a small window size tends to reach an F1score quickly, but better results will be impacted by the algorithm used and approaches still could have better results using a bigger portion of the trace instead

of small window size. Also, one-class SVM perform slightly better than isolation forest with smaller traces, but as the window size increases, isolation forest becomes much better, outperforming SVM.

In our study case, the raw and filter data present variations over the window growth, not showing a bigger difference between the use of one specific data type. It is possible to appoint that the filter approach was the first one to obtain an F1Score above 98% in both evaluations and this data type could help small window size techniques for anomaly detection.

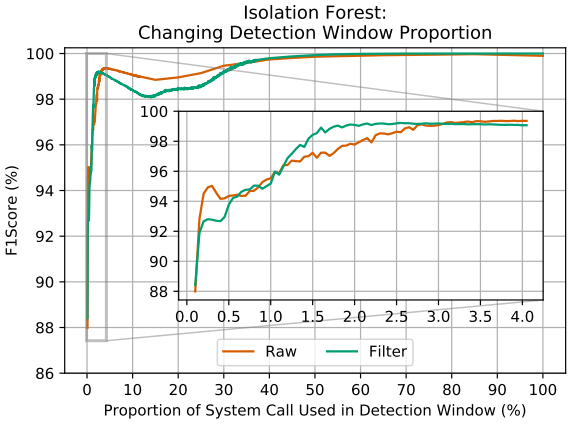

Figure 1. Window growth for Isolation Forest algorithm.

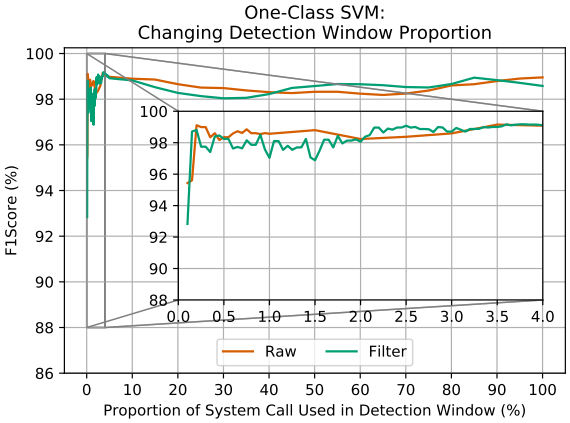

Figure 2. Window growth for One-Class SVM algorithm.

5. Conclusion

The respective study focus on the impact of size window in a docker environment. Our approach aims to clarify, through a growth window size experiment, how the size will impact on One-Class Classification (OCC). Our contribution is for the anomaly detection system turned to Docker environments, and helps with new security insights, due to the popularization of containers in the last decade.

The One-Class evaluation present allow us to better understand the anomaly detection scenario in containers. With two distinct groups of data, we can claim that the smallest window is able to achieve reasonable F1Score results, even with filtered data, where this result can be reached with smaller windows in comparison to raw data.

According to the experiments, it is possible to assume that a small window size already presents an acceptable F1Score for anomaly detection and it's more appropriate to an online detection system. But the size could be affected by the classifier used to identify anomalies.

Two open topics that will be approached in future works: (i) data generation, given that our dataset is limited to one application behavior and we hope to develop a more complete dataset, with more applications that will create more real investigations, and (ii) explore new classifiers and techniques of anomaly detection considering the window size shown in this research.

References

[Alarifi and Wolthusen 2012] Alarifi, S. S. and Wolthusen, S. D. (2012). Detecting anomalies in iaas environments through virtual machine host system call analysis. In *2012 International Conference for Internet Technology and Secured Transactions*. IEEE.

[Bernaschi et al. 2002] Bernaschi, M., Gabrielli, E., and Mancini, L. V. (2002). Remus: a security-enhanced operating system. *ACM Transactions on Information and System Security (TISSEC)*.

[Castanhel et al. 2020] Castanhel, G. R., Heinrich, T., Ceschin, F., and Maziero, C. A. (2020). Detecção de anomalias: Estudo de técnicas de identificação de ataques em um ambiente de contêiner. Undergraduate Research Workshop - Brazilian Security Symposium (WTICG - SBSeg).

[Forrest et al. 1996] Forrest, S., Hofmeyr, S. A., Somayaji, A., and Longstaff, T. A. (1996). A sense of self for unix processes. In *IEEE Symposium on Security and Privacy*.

[Liu et al. 2008] Liu, F. T., Ting, K. M., and Zhou, Z.-H. (2008). Isolation forest. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, page 413–422, USA. IEEE Computer Society.

[Liu et al. 2018] Liu, M., Xue, Z., Xu, X., Zhong, C., and Chen, J. (2018). Host-based intrusion detection system with system calls: Review and future trends. *ACM Computing Surveys (CSUR)*.

[Merkel 2014] Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux journal*.

[Mitchell et al. 2001] Mitchell, M., Oldham, J., and Samuel, A. (2001). *Advanced linux programming*. New Riders Publishing.

[NVD 2020] NVD (2020). National vulnerability database: Rce wordpress. `https://nvd.nist.gov/vuln/search/results?form_type=Basic&results_type=overview&query=RCE+wordpress&search_type=all`.

[Systems 1998] Systems, C. I. (1998). Sequence-based intrusion detection. `http://www.cs.unm.edu/~immsec/systemcalls.htm`.

[Wang et al. 2006] Wang, K., Parekh, J. J., and Stolfo, S. J. (2006). Anagram: A content anomaly detector resistant to mimicry attack. In *International workshop on recent advances in intrusion detection*, pages 226–248. Springer.

[Xie and Hu 2013] Xie, M. and Hu, J. (2013). Evaluating host-based anomaly detection systems: A preliminary analysis of adfa-ld. In *2013 6th International Congress on Image and Signal Processing (CISP)*, volume 03, pages 1711–1716.

[Yassin et al. 2013] Yassin, W., Udzir, N. I., Muda, Z., Sulaiman, M. N., et al. (2013). Anomaly-based intrusion detection through k-means clustering and naives bayes classification. In *Proc. 4th Int. Conf. Comput. Informatics, ICOCI*, number 49.

[Zhang et al. 2015] Zhang, M., Xu, B., and Gong, J. (2015). An anomaly detection model based on one-class svm to detect network intrusions.