DOI: 10.15439/2019F212
ISSN 2300-5963 ACSIS, Vol. 18

Standardized container virtualization approach for collecting host intrusion detection data

Martin Max Röhling, Martin Grimmer*,
Dennis Kreußel†, Jörn Hoffmann*
Leipzig University
Ritterstraße 9-13, 04109 Leipzig
Email: roehling@wifa.uni-leipzig.de

*{grimmer, jhoffmann}@informatik.uni-leipzig.de
†dnk0@protonmail.com

Bogdan Franczyk
Leipzig University
Grimmaische Straße 12, 04109 Leipzig
Uniwersytet Ekonomiczny we Wroclawiu
ul. Komandorska 118/120, 53-345 Wroclaw
Email: franczyk@wifa.uni-leipzig.de

***Abstract*—Anomaly-based Intrusion Detection Systems (IDS) can be instrumental in detecting attacks on IT systems. For evaluation and training of IDS, data sets containing samples of common security-scenarios are essential. Existing data sets are not sufficient for training modern IDS. This work introduces a new methodology for recording data that is useful in the context of intrusion detection. The approach presented is comprised of a system architecture as well as a novel framework for simulating security-related scenarios.**

I. Introduction

The current threat situation of the IT landscape makes it necessary to monitor systems and detect attacks at an early stage. Host-based Intrusion Detection Systems (HIDS) are important tools to inspect system calls and to analyze processes which are accessing systems. Especially anomaly-based HIDS are able to detect previously unknown attacks. These are trained in advance with normal behavior and detect deviant behavior in the event of an attack. The quality of an anomaly-based HIDS in relation to the detection and error rate is significantly linked to the quality of the training of these systems. In recent years, various data sets have been published to evaluate a HIDS. As it turns out all them have at least one serious problem [1]. In addition for comparability and evaluation, their metrics must be applied to a set of coherent standardized data sets. Thus, all existing data sets are not sufficiently applicable to design anomaly-based HIDS for the modern IT landscape. Especially when modern operating systems, multithreaded applications and concurrent communications are considered.

The methodology presented in this paper enables the simulation and comprehensive recording of normal and attack behavior with an high degree of detail. Further, we suggest plausible practices for implementing this approach. This includes a procedure to generate new data sets on current operating systems. The latter are suitable to develop and evaluate algorithms for today's state of the art anomaly-based HIDS.

A. Relevance in practice and research

Our methodology can be used to record normal behavior from productive systems. This generates models based only on the data actually captured from live containers. This is the opposite to today's HIDS that often rely entirely on data captured from a staged lab environment. This way, a user can simulate a security violation by implementing a custom policy. This results in a transparent process, in which an IDS can be build and evaluated with an productive system in mind.

The procedure model serves research primarily with experimenting and evaluation of new IDS algorithms. In today's research, new algorithms are evaluated with outdated and incomplete data sets. Current data sets with extensive context enable new research and better evaluation of the algorithms.

II. Background and related work

Since 1998 data sets for training and comparison of HIDS have been published. The best known are: the DARPA Intrusion Detection Evaluation Data Set (KDD), from 1998 to 2000 [2] the data set of the University of New Mexico (UNM) from 1999, [3], [4], the data sets of the Australian Defence Force Academy, the ADFA-LD from 2013 [5], [6] and the NGIDS-DS from 2017 [7]. They share at least one of the problems described by Grimmer et al. [8] as shown in table I. These data sets consist of sequences of system calls. The ADFA-LD for example is relatively up-to-date, but it does not provide thread information, parameters and return values. In addition it contains system calls of a complete system with all its processes. In particular, it is therefore not possible to learn the normal behavior of a single program from it. Short examples for all four data sets can be seen in listing 1.

Unfortunately, the authors of the data sets omit many details about their implementation. There is little information about the general conditions under which the recordings have been made. This refers to information on the recording process, the tools and software versions used or the attack vectors engineered into the system. Moreover, the normal behavior and its origin are either not described at all or only insufficiently described. Pendleton and Xu describe in [1] an architecture

This work was partly funded by the German Federal Ministry of Education and Research within the project "Explicit Privacy-Preserving Host Intrusion Detection System" (EXPLOIDS) (BMBF federation code 16KIS0522K) and "Competence Center for Scalable Data Services and Solutions" (ScaDS) Dresden/Leipzig (BMBF federation code 01IS14014B).

based on a syscall collector for data generation. For instrumentation the Intel tool "Pin" is used[1]. It allows applications to be extended with their own source code at runtime and to profile the application. This allows thread-based system call sequences and context information to be captured. The authors show this with the software example Firefox. Abed et al. describe in [9] a real-time IDS for passive monitoring of Linux containers using the tool strace. The evaluation takes place with the example of the database application MySQL in the normal and malicious behavior under consideration of the frequencies of system calls. The data set was not published. In [10] older and new algorithmic approaches were compared that evaluate sequences of system calls. It was observed that both the detection and the false alarm rates of the different approaches could not be improved beyond a certain value. The authors' thesis is that the quality of HIDS can be improved if the algorithms also take into account context information such as parameters and return values for system calls. To pursue this thesis, a new data set containing such information is needed.

```
# Structure of KDD BSM data.
open(2): read
system call    open(2)
event-ID       72 AUE_OPEN_R
event class    fr(0x00000001)
audit record:  header token, path token, [attr token], subject token, return token

# Extract from the UNM data set: PID SystemcallID, PID SystemcallID, ...
162 4, 162 2, 162 66, ...

# Extract from the ADFA-LD data set: SystemcallID SystemcallID ...
54 175 120 ...

# Extract from the NGID-DS data set
DATA, TIME, PID, PATH, SystemcallID, Event ID, Categ., Subcat, Label
11/03/2016, 2:45:01, 1830, /sbin/upstart-dbus-bridge, 142, 45354, normal, normal,0
11/03/2016, 2:45:06, 1804, /bin/dbus-daemon, 256, 45352, normal, normal, 0
```

Listing 1. Fragments of commonly used IDS data sets

III. Requirements

Based on the weaknesses of the previous data sets [8], the following requirements apply to the new data set and the method of producing it: Over time, the number, syntax and semantics of system calls of operating systems have changed. For this reason and to solve the lack of topicality, the system calls of today's systems and current software must be considered. To ensure that the generated data sets can be kept up-to-date in the future, the simulation process should be replicable.To fix the lack of thread information, the recorded system calls must contain process and thread information. This allows the data set to correctly represent normal and attack behavior in today's multithreaded environments. In addition, the recorded system calls must include metadata such as their time stamps, parameter and return values to solve the mentioned lack of meta information. The size of the data set, i.e. the number of contained sequences and their system calls can be selected as required in order to carry out procedures with large training requirements, such as the training of a neural network. This solves the lack of data volume. Normal and attack behavior shall be recorded according to the same procedure. The basic conditions such as operating system/kernel, the software used and versions should be identical. The only difference between normal and attack procedures is the attack carried out during the simulation. The process must be customizable in order to be able to adapt the collected data to the respective application area by implementing own scenarios.

IV. Method

Our method considers the previously established requirements to collect suitable data sets for training and evaluation of a HIDS. By using this method, scenarios can be defined and both normal behaviour and attack behavior can be simulated and recorded. Simulation in this matter means the staged execution of benign and attacker behavior on an actual machine. The method basically defines the following procedure pattern: (1) The acquisition of events at kernel level (system calls). (2) The use of a container-virtualized environment. (3) A framework for instrumentation and configuration of scenarios.

The procedure is based on a system model in which three actors *Victim Unit*, *Normal Behavior Unit* and *Control Unit* are related. The development of a system model represents an application scenario. The Leipzig Intrusion Detection Data Set (LID-DS) framework was developed and provided as a reference implementation.

With it, scenarios that include or exclude vulnerabilities can be defined, simulated and recorded. In recent years, Microservice architecture has become more and more established. With such an architecture, complex software is composed of many loosely coupled services. Due to this development, the whole process is adapted for use in container virtualized environments. Therefore, the resulting dataset no longer describes an entire complex system but a single component, e. g. a web server. This is referred as a scenario.

The result is a set of captured instructions executed by the system in the form of system calls related to single application. This shifts the scope towards attack vectors based on behaviour and network communication. However, many choices pointed out here only require a little adjustment to support other scenarios.

A. System model of an application scenario

Scenarios implemented in the LID-DS are based on software using the client-server model. However, also centralized alternatives like the mainframe architecture or peer-to-peer applications are within the scope of the recording framework. An example is the web server scenario, which consists of one web server as well as *n* units which request it. Figure 1 list the schematic model in normal and attack behavior. The *Victim Container* describes the *Server* in the network. This is monitored by a *Container Sensor* that extracts normal and attack behavior. The Sensor works introspectively and records the events on kernel level in the form of system calls with minimal influencing the behaviour. *Normal Behavior Unit(s)* resemble the clients in the network. In a web server scenario, they would execute requests to call a web page. As with reality,

[1] https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool

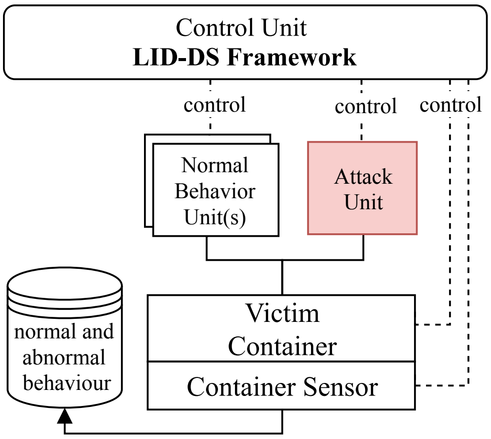

Fig. 1. system architecture to capture attack behavior

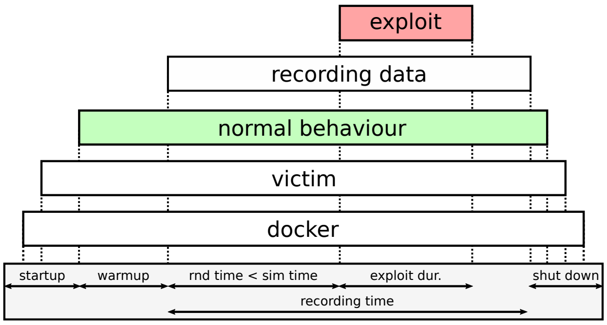

Fig. 2. Simulation procedure of LID-DS

several of these actors can exist to create a realistic normal behavior, including occasional load peaks. The *Attack Unit* also is a client in the network, which however executes an attack on the victim.

Normal behavior and attack behavior are caused by the influence of other entities like containers or processes. In a server-client application for example it is possible to map one or more webclients to a container. A separate *control unit* executes the LID-DS framework and implements the application scenario. LID-DS allows controlling the lifecycle of containers, which includes the initialization and configuration of the actors as well as the control and monitoring of the entire simulation process. Use cases are defined in the form of scenarios and executed by the process.

In this way, simple as well as more complex realistic scenarios, such as multi-step attacks, can be evaluated.

B. Simulation procedure

The simulation process is shown in figure 2 and consists of the four consecutive phases *startup*, *warmup*, *simulation of normal program behavior*, *shutdown* and potentially the *exploit*. Basically, each implemented scenario is executed according to this procedure. At the beginning, the container is initialized and the Victim Container is configured and executed. At this point, benign user behavior is started to be executed with respect to the victim container. The subsequent *warmup* phase includes a delay so that the system is in a steady (non-transient) state when recording starts. After the *warmup* delay has passed, the recording of the normal behavior begins. Granted, a malicious user pattern is supplied, the attack behavior gets executed at a randomly chosen time within the recording window. After the specified recording time has passed, the monitoring tool together with all containers gets shut down in reverse order that they started.

C. Instructional System Call Tracer

Historically, strace[2] was used to monitor interactions between processes and the Linux kernel. Strace interrupts the traces process every time a system call is invoked, captures the system call, decodes it and then resumes the execution of the monitored process. It is obvious that while this behavior allows for easy recording and tampering of system instructions, for the purpose of recording system activity this is not very efficient. Sysdig [3] on the other hand, loads a small driver in the kernel that makes it possible to handle different events related to system calls. This event collection is, in contrast to strace, non-blocking. Furthermore, Sysdig pre-processes the data collected, combining information on system call executions with data from tcpdump or information on referenced files. Our approach settled on using Sysdig for recording system calls since it provides a pragmatical way of achieving the requirements identified here. This is best displayed by Sysdig providing export functionality, pre-processing of many file descriptors and rich filtering functionality, allowing for efficient prototyping. This choice, however, does not limit the approach's capabilities since Strace would allow for the extraction of data in a similar manner.

D. Container Virtualization Engine

Docker[4] as a container virtualization engine (LXC) was chosen because it is a commonly used standard in practice. LXC is used to run multiple instances of the operating system isolated on a single host. In contrast to full virtual machine environments, guests share the kernel with each other. This level of virtualization allows a sufficient isolated environment to be created at the application level. To monitor one or more containers on the host, it is only necessary to inject the Sensor on the host level. This is resource-friendly and allows us to record application behavior with little impact to the system calls. It also gives us a high degree of flexibility in creating scenarios.

E. LID-DS Framework

The LID-DS framework implements this procedure with minimal development effort. It covers all steps necessary for the simulation and recording of HIDS data. In detail, it takes care of the following steps: handling victim virtualization via Docker, System Call Tracing via Sysdig and communication between user behavior and victim via a bridge network. To

[2] https://linux.die.net/man/1/strace
[3] https://github.com/draios/sysdig
[4] https://www.docker.com

schedule user actions the distribution introduced by Deng [11] has been chosen.

To record data of a scenario the following information must be defined: A Docker image, specifying the configuration of the victim environment, a set of benign user actions, an script exploiting a vulnerability of the victim and a metric to check for correct and finished initialization of the victim environment.

The LID-DS framework makes it possible to define several normal behaviors for a scenario and the associated victim. All of the passed behaviors are executed in parallel, each in its own thread. This makes it possible to simulate multiple parallel user sessions accessing the victim. This implementation opens up the possibility to mirror real-world network traffic instead of simulating staged user actions to the victim.

Within the scope of real world applications, many different scenarios can hopefully be defined by using a single user simulation definition. Furthermore, many malicious actions, especially actions of reconnaissance are indifferent to many victim configurations. For example, consider a TCP-SYN Scan using the nmap[5] tool.

V. Evaluation and results

To evaluate how the proposed framework can be used to record host data consider CVE-2012-2122[6], a tragically comedic security flaw in MariaDB/MySQL. A new data record is created by monitoring a vulnerable MySQL instance according to the procedure shown earlier. The setup consists of a Ubuntu Xenial in version 16.04 with the Docker version 18.09.6 and Sysdig in version 0.24.1. MySQL is used in version 5.5.23, which contains the vulnerability. The simulation time is 5 minutes, the time exploit is 120 seconds. As recording time we have chosen 5 minutes because Sysdig in its report[7] has surveyed an average running time of 5-10 minutes for containers. Two runs are performed. One run only generates normal behavior, a second run contains the attack on the vulnerability in addition to the normal behavior. As a result, the content and technical requirements from III are compared to this data set. The resulting data set is compared to the commonly used IDS data sets on basis of the categorized requirements (Table I).

A. Comparison

The focus of this comparison is on the resulting artifact, the data set and the features it contains. Concerns related to efficiency during the recording phase are not considered in this paper. The central question is whether this procedure, measured against the result, can solve the criticism and problems of the previous data sets. The evaluation resulted in two data sets. Data set 1 contains normal behavior and includes 46839 system calls. Data set 2 contains normal and attack behavior and includes 128799 system calls. Listing 2 shows an excerpt of data set 2. For the interpretation of the results, the classified requirements are compared below.

TABLE I
Feature comparison LID-DS with Others

feature	**LID-DS**	**NGIDDS**	**ADFA-LD**	**UNM**	**KDD**
topicality	+	+	+	-	-
thread info	+	+	-	+	+
metadata	+	-	-	-	+
data volume	+	+	-	-	+
reproduceability	+	-	-	-	-

1) Lack of topicality: The data was recorded on a modern Linux system, which has over 370 different system calls. The containerized environment and the LID-DS framework makes it easy to repeat such runs for different versions of operating systems. This makes sense because the number of system calls varies from operating system to operating system. The data set can be updated simply by adjusting the configuration and running it again.

2) lack of thread information: For each system call in the recording period the thread ID on which the process is running is recorded, as shown in listing 2. This ensures that the multithreading information that is important today is not lost. The most recent data set ADFA-LD lacks this information.

3) lack of metadata: For each system call, the data record contains extensive meta information such as high-precision time stamps, process name, transfer parameters and a section of the data buffer. The time stamps in the NGIDDS are only accurate to the second, which can lead to errors in the sequence.

4) lack of volume: Over 100 000 system calls were recorded during the survey period. For example, ADFA-LD, as well as the highly obsolete UNM data set, provides a smaller, fixed data set of system calls. The data collection period can be configured in LID-DS so that larger or smaller data records can be generated according to individual requirements.

5) lack of reproduceablity: By using LXC, the entire simulation can be defined with the LID-DS framework and stored. Including software versions, parameters or configurations. The performed evaluation is stored in Github as *example*[8] and can be viewed and performed by anyone. The framework itself is published under the GNU General Public License. As we know, this is the first time that a process for generating HIDS datasets is available to the public in a fully reproducible form.

B. Results

Overall, there is a significant superiority of the approach to generating modern HIDS data sets, as shown in table I. By using the LID-DS Framework, all technical and content requirements are fulfilled. The framework allows modern and operating system specific data sets to be generated, which is important to avoid training neural networks on the basis of outdated or incorrect system calls or nowadays uncommon

[5] https://nmap.org

[6] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-2122

[7] https://sysdig.com/blog/2018-docker-usage-report/

[8] https://github.com/LID-DS/LID-DS

```
TIME CPU PROCESS PROCESS_ID ENTER(>)/EXIT(<) SYSCALL ARGUMENTS
t0 0 apache2 25426 > open
t1 0 apache2 25426 < open fd=13(<f>/etc/apache2/.htpasswd) name=/etc/apache2/.
      htpasswd flags=4097(O_RDONLY|O_CLOEXEC) mode=0
t2 0 apache2 25426 > fstat fd=13(<f>/etc/apache2/.htpasswd)
t3 0 apache2 25426 < fstat res=0
t4 0 apache2 25426 > read fd=13(<f>/etc/apache2/.htpasswd) size=4096
t5 0 apache2 25426 < read res=91 data=QUEU75:$apr1$X0JgPVeW$xCKOGdUp2tNNs0t6RqB...
t6 0 apache2 25426 > close fd=13(<f>/etc/apache2/.htpasswd)
t7 0 apache2 25426 < close res=0
```

Listing 2. Short excerpt of data from a recorded trace collected with the LID-DS Framework including thread information and metadata

TABLE II
IMPLEMENTED AND PUBLISHED SCENARIOS BY LID-DS

Scenario	CVE / CWE
Heartbleed	CVE-2014-0160
PHP file upload	CWE-434
Bruteforce login	CWE-307
Rails Disclosure of content	CVE-2019-5418
ZipSlip	various
EPS file upload	CWE-434
MySQL auth bypass	CVE-2012-2122
Nginx int. overflow	CVE-2017-7529
Sprockets info. leak	CVE-2018-3760
SQL injection with sqlmap	CWE-89

operating systems. The amount of data is also important for neural networks, which can be adapted by LID-DS. Multi-threaded information is valuable to view the behavior of a system down to the application and thread level. In this context, the metadata and parameters are also relevant. These can also contain application-specific information and support the correct interpretation of the application behavior. The overall approach provides the basis to effectively compare and evaluate HIDS in the future and to develop new classification features based on thread information, metadata and parameters in order to significantly increase the recognition rate and accuracy of HIDS.

VI. CONCLUSION

The need for modern, uniform and metadata enhanced data sets can be satisfied by implementing this approach. This way, LID-DS is a significant contribution to future research, evaluation and comparability of Host-Based Intrusion Detection Systems. Additionally, this approach only needs slight adaptations to be functional in production environments. The major advantage of this approach is that it provides a high degree of flexibility in the form of scenarios that can be adapted to individual technical as well as policy requirements. For example, the evaluation MySQL example[9] from chapter V can easily be adapted using real network data. With this approach we have created a new data set. It was published as "Leipzig Intrusion Detection - Data Set (LID-DS)" in [8] which contains different use cases shown in table II. LID-DS framework and ready to use data sets are free to use and published on GitHub[9]. It is the first HIDS data set which contains normal and abnormal behavior, system calls and their timestamps, thread ids, process names, arguments, return values and excerpts of their data buffers from traces of normal and attack behavior of several recent, multi-process, multi-threaded scenarios. Many of the included features cannot be extracted from previous data sets. With it, known algorithms can be enhanced or new algorithms, based on the various included features, can be explored. Additionally, staged scenarios based on internal expert knowledge can give a practical prediction on the performance of a algorithm. The approach outlined in this work focuses on giving every actor the possibility to build their own model from their own experienced traffic. Further information on the development of LID-DS and initial analyses can be found in the works of [12] and [13]. An extension of the procedure to include network sensors is planned. Furthermore, an updated version of the data set is scheduled to be released once a year. Every version should expand the data set by including recordings of the latest commonly used software systems as well as disclosed vulnerabilities. Additionally, recordings based on new versions of the Linux kernel and configurations according to new techniques used in development, hosting and pentesting will be the target of these extensions. We anticipate feedback to allow for the progressive development of a data set that finally allows for reproducible IDS research.

REFERENCES

[1] M. Pendleton and S. Xu. A dataset generator for next generation system call host intrusion detection systems. In *Proceedings - IEEE Military Communications Conference MILCOM*, volume 2017-Octob, 2017. DOI: 10.1109/MILCOM.2017.8170835.

[2] Lincoln Laboratory; MIT. DARPA Intrusion Detection Evaluation Data Set. https://www.ll.mit.edu/r-d/datasets, 1998-2000.

[3] Computer Science Department Farris Engineering Center; University of New Mexico. Computer Immune Systems - Data Sets and Software. https://www.cs.unm.edu/˜immsec/systemcalls.htm, 1999.

[4] C. Warrender, S. Forrest, and B. Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *Proceedings - IEEE Symposium on Security and Privacy*, 1999. DOI: 10.1109/SECPRI.1999.766910.

[5] Australian Center for Cyber Security (ACCS). The ADFA Intrusion Detection Datasets. https://www.unsw.adfa.edu.au/australian-centre-for-cyber-security/cybersecurity/ADFA-IDS-Datasets/, 2013.

[6] G. Creech and J. Hu. Generation of a new IDS test dataset: Time to retire the KDD collection. In *IEEE Wireless Communications and Networking Conference, WCNC*, 2013. DOI: 10.1109/WCNC.2013.6555301.

[7] W. Haider, J. Hu, J. Slay, B.P. Turnbull, and Y. Xie. Generating realistic intrusion detection system dataset based on fuzzy qualitative modeling. *Journal of Network and Computer Applications*, 87:185–192, 6 2017. DOI: 10.1016/J.JNCA.2017.03.018.

[8] M. Grimmer, M. M. Röhling, D. Kreusel, and S. Ganz. A modern and sophisticated host based intrusion detection data set. In *IT-Sicherheit als Voraussetzung für eine erfolgreiche Digitalisierung*, pages 135–145, 2019. ISBN: 978-3-922746-82-9.

[9] A. S. Abed, C. Clancy, and D. S. Levy. Intrusion detection system for applications using linux containers. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9331, pages 123–135, 11 2015. DOI: 10.1007/978-3-319-24858-5_8.

[10] M. Grimmer, M. M. Röhling, M. Kricke, B. Franczyk, and E. Rahm. Intrusion Detection on System Call Graphs. In *Sicherheit in vernetzten Systemen*, pages G1–G18, 2018. ISBN: 978-3-3-7460-8637-8.

[11] Deng, S. Empirical model of WWW document arrivals at access link. In *Proceedings of ICC/SUPERCOMM '96 - International Conference on Communications*, volume 3, pages 1797–1802. IEEE. DOI: 10.1109/ICC.1996.535600.

[12] S. Ganz. Ein moderner Host Intrusion Detection Datensatz, 2019.

[13] D. Kreußel. Simulation and analysis of system call traces for adversial anomaly detection, 2019.

[9] https://github.com/LID-DS/LID-DS