A Modern and Sophisticated Host Based Intrusion Detection Data Set

Martin Grimmer[1], Martin Max Röhling[1], Dennis Kreußel[1], Simon Ganz[1]

Abstract:
Cyber attacks can do great damage. Host intrusion detection systems (HIDS) can help to detect those attacks. In order to compare different HIDS and test their effectiveness, up-to-date, correct and publicly accessible data sets are required. Since all data sets available so far have serious problems, we present a new one that solves these problems and explain how it came about.

Keywords: Host Based Intrusion Detection, Data Set

1. Introduction

The dependency of our society on IT infrastructure increases every day. At the same time attacks in cyberspace are becoming increasingly professional and complex. This is why the potential for damage is constantly increasing.

A first possible countermeasure to current and future cyber attacks is to use Intrusion Detection Systems (IDS). Unlike network intrusion detection systems (NIDS), host-based intrusion detection systems (HIDS) record data of the system to be monitored. In recent years, there have been some promising publications that have shown that system calls of those hosts have been used to successfully identify attacks [5] [6] [8] [9] [11] [16]. These methods, often based on anomaly detection, are even able to detect previously unknown attacks. In most cases, the sequences of the system calls were analyzed [8] [9] [16]. Some other methods did not consider the sequences but the frequencies of the occurring system calls [11]. Rarely, but with interesting results, the parameters of the system calls were considered [6]. There are also studies that investigate the sequences as well as the parameters in combination, which is very promising [5]. However, in the absence of a suitable and publicly accessible test data set, it is difficult to compare these methods.

Nonetheless, in the past, several data sets have been published for testing and comparing host based intrusion detection systems. The BSM (Basic Security Module) of the DARPA Intrusion Detection Evaluation Data Set 1998 and 1999 (called KDD) [3] [4], the Intrusion Detection Data Set of the University of New Mexico from 1999 (called UNM) (Engineering, 1999) [1], the newer and recently often used data set from 2013, called ADFA-LD [2] [10] and the newest data set from 2017, the NGIDS-DS [10] [12], both from the Australian Defence Force Academy. Each of these data sets has at least one critical drawback[7], which will be summarized in the following. Therefore, we are introducing a new data set to close the resulting gap and to enable future, comparable research on this topic.

[1] Leipzig University

2. The Problems

These data sets all have at least one of the following problems.

First: they are outdated. This is doubly true. On the one hand, 20-year-old software is rarely used today, and on the other hand, both the hardware and the corresponding operating systems and their system calls[2] have changed during this time.

Second: they lack thread information. Since almost all current IT systems are multithreaded, this is a critical point. Even with two runs of the same software with the same input data, the order of execution may vary with several parallel threads. Since many current work on HIDS is spend on learning and analysing n-grams of system calls this can lead to wrong results. As seen in Figure 1 on the left side a trace of a program with its system calls is shown split up by its threads and as plain trace. On the right side, another run of the same software, with the same input is shown. This time the first system calls of thread 1 and thread 2 are executed in a different order due to the multithreaded environment. Each thread has the same order as before, but the overall context has a different plain trace. To avoid such errors the thread information is necessary. Since n-grams are often used to learn normal sequences in HIDS ignoring this kind of information can lead to learning incorrect data, as shown in Figure 2.

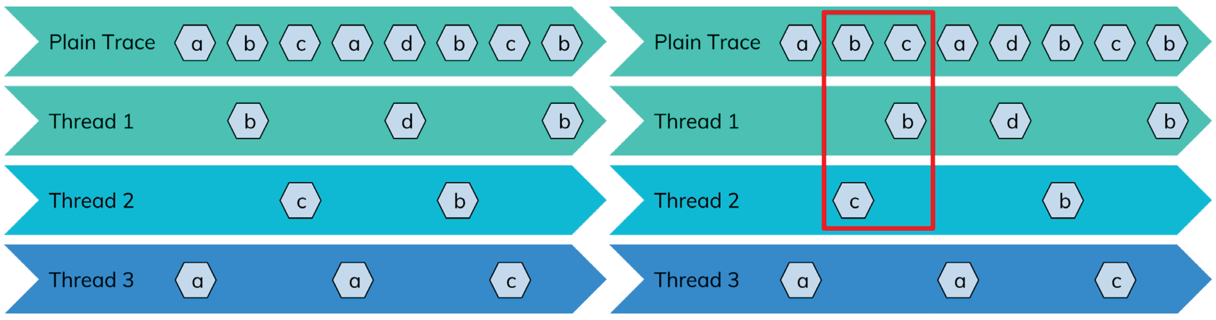

Figure 1: Two different traces resulting from two runs of the same software and input in a multithreaded environment.

	aa	ab	ac	ad	ba	bb	bc	bd	ca	cb	cc	cd	da	db	dc	dd
1)	0	1	0	1	0	0	2	0	1	1	0	0	0	1	0	0
2)	1	0	1	0	0	0	0	1	0	1	0	0	0	1	0	0

Figure 2: Line 1) shows the frequency of all n-grams with n=2 in case they were determined from the plain trace. In line 2), the frequency of all n-grams is shown for the case that they were determined from the thread wise traces.

Third: they lack metadata. Arguments, timestamps and return values of system calls are promising indicators for HIDS [5][6]. An example for a vulnerability detectable by arguments of system calls is the "Zip Slip Vulnerability"[3]. It leads to overwritten files

[2] Over time some system calls were added or removed from the Linux kernel. See: http://manpages.ubuntu.com/manpages/bionic/man2/syscalls.2.html

[3] Zip Slip Vulnerability: https://snyk.io/research/zip-slip-vulnerability

by unpacking manipulated archives. This manipulation is manifested by the fact that the paths of the files to be unpacked are of the following form: `../../../../../somepath/evil.sh` and are thus unpacked into any directory on the same machine.

Fourth: a small amount of data. The relative small number of (mostly short) traces makes it hard to apply modern machine learning approaches such as deep learning.

All previously mentioned data sets contain at least traces of system calls. The BSM part of the KDD data sets from 1999 consists of complex Solaris BSM audit logs including events like system calls including arguments and return values. The UNM data from 1999 contains only process IDs and system call IDs, but no other information. Both data sets, the KDD and UNM, are outdated and therefore not suitable for evaluating modern methods for intrusion detection[7]. The ADFA-LD from 2013 is relatively up-to-date, but only contains sequences of system call IDs without their parameters, return values or other metadata. Although the newer NGIDS-DS of 2017 contains thread information, both parameters and more accurate timestamps are missing.

Figure 3 gives a first impression of the information contained in the data sets listed here.

KDD

```
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
```

UNM

```
# PID Syscall, ...
162 4, 162 2, 162 66, ...
```

ADFA-LD

```
# Syscall Syscall ...
54 175 120 ...
```

NGIDS-DS

```
DATA,       TIME,    PID,  PATH,                      Syscall, 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
```

Figure 3: Examples for the data contained in the different data sets.

3. The Solution: LID-DS

3.1 Overview

To overcome the mentioned problems the new *Leipzig Intrusion Detection - Data Set* (LID-DS) contains a multitude of data, associated with system calls, and recorded on modern operating systems with current software and different types of attacks. This recorded data should contain the system call IDs, their parameters and return values. Additionally it contains corresponding metadata such as process IDs, process names and timestamps.

LID-DS consists of several scenarios, each containing an attackable system. For each included scenario, two categories of traces were recorded. The normal behavior of the attacked system (victim) and the behavior of the victim under attack, which includes the normal behavior as well.

The whole process of generating and recording the mentioned data is configurable scripted. This makes it possible to create any number of traces with different configurations, for example, different access patterns to web applications or a different numbers of active users in the simulation.

In addition, the attack sequences should not always occur at the same position in the trace and are therefore started at a random point in time during the execution of normal behavior.

3.2 Attacks

In order to obtain a broad spectrum of attacks and the corresponding system calls LID-DS contains recordings of the following attacks:

CVE[4]-2014-0160, Heartbleed: The bug allows attackers to read memory contents from the server due to a bug in the implementation of OpenSSL and thus possibly get access to secret keys and other sensitive data.

CWE-307, Improper Restriction of Excessive Authentication Attempts: Brute force username and password guessing attempt.

CWE-89, SQL Injection: The insertion of attacker-controlled data into variables that are used to construct SQL commands.

CWE[5]-434, Unrestricted Upload of File with Dangerous Type (PHP): The attacker is allowed to upload files of dangerous type, like php scripts, that can be processed within the victims environment.

CVE-2014-3120, Arbitrary code execution: The default configuration in Elasticsearch enables dynamic scripting, which allows remote attackers to execute arbitrary Java code.

CVE-2015-1427, Arbitrary code execution: The Elasticsearch scripting engine makes it possible to bypass the sandbox protection mechanism and execute arbitrary shell commands.

CVE-2014-6271, Shellshock: In Bash, the value of a string variable can contain a function definition. This vulnerability allows unchecked program code to be executed after such a variable has been evaluated.

CVE-2016-6515, Denial-of-service: The password authentication in OpenSSH does not limit password lengths, which allows attackers to cause a denial of service via a long string.

[4] CVE: Common Vulnerabilities and Exposures, a public database for information-security vulnerabilities and exposures.

[5] CWE: Common Weakness Enumeration, a list of for software weaknesses and vulnerabilities categories.

CVE-2015-5602, Local Privilege Escalation: Allows local users to gain privileges via a symlink attack on a file whose full path is defined using multiple wildcards in /etc/sudoers.

Zip Slip: Zip Slip is a widespread arbitrary file overwrite critical vulnerability, which typically results in remote command execution. Was in Projects like: Hadoop and Maven. It is known under several CVEs.

CWE-434, Unrestricted Upload of File with Dangerous Type (eps): A service is converting images from different image formats to the svg file format. One supported format, the Encapsulated PostScript actually is a scripting language. With eps it is possible to embed malicious code in the images.

3.3 Normal Behavior

As mentioned before LID-DS contains a normal behaviour simulations for each included attack scenario. In contrast to the recorded attacks, it is difficult to model normal behavior. It differs greatly from application to application and in different operational areas. Nevertheless, we have modeled a normal behavior for each scenario to generate recordable system calls. Scenarios included in LID-DS are presented in the next section.

3.4 Description of the scenarios

This section briefly describes each LID-DS scenario.

3.4.1 CVE-2014-0160 (Heartbleed) and CWE-307 (brute force login)

The normal behaviour for the CVE-2014-0160 (Heartbleed) and CWE-307 (brute force login) scenario looks like this: An Apache web server is deployed as victim with a vulnerable OpenSSL version. It serves different kinds of http requests to a web application and heartbeat protocol. The normal behaviour simulation for these scenarios involves multiple users using Selenium[6] to request random documents from the web server or upload files to a basic access authentication protected area of the web application. In addition, each of the clients ensures that the server can still be reached by means of valid heartbeats. The attacking behaviour for CVE-2014-0160 adds a client who sends a malicious heartbeat request message. The attacking behaviour for CWE-307 adds a user who uses the http_login module from Metasploit[7] to start a brute force login attempt.

3.4.2 CWE-89 (SQL Injection) and CWE-434 (PHP File Upload)

The normal behaviour for the SQL injection and file upload scenarios is modeled as follows. Multiple clients log into the DVWA[8] web application and navigate by means of a random walk. When they encounter the SQL Injection page during their random walk, they submit non malicious form input. When they hit the File

[6] Selenium, a browser automation tool. https://www.seleniumhq.org/

[7] Metasploit, a pentesting framework. https://www.metasploit.com/

[8] Damn Vulnerable Web App, a web application with numerous security flaws. http://www.dvwa.co.uk/

Upload page, they submit random files. The timings of the user actions are modeled after the distributions given in the work of Deng[13] and visualized in Figure 4. Simply put, there are two phases of user behaviour: the active phase in which the user makes requests and the inactive phase in which the user processes the information received. The attacking behaviour for the SQL Injection adds an additional client that uses the sqlmap[9] tool to read the contents of the entire DVWA database. The attacking behavior for the file upload vulnerability adds a user who uploads a reverse shell php script which is then executed by requesting it from the server.

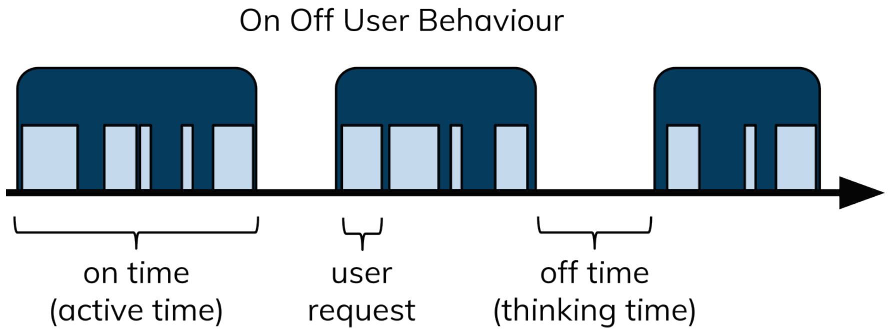

Figure 4: Visualisation of the SQL Injection scenario and the "on off" user behaviour used for scheduling user interactions according to Deng.

3.4.3 CVE-2014-3120 and CVE-2015-1427 (Arbitrary code execution)

The normal behavior for these scenarios involves a user doing essentially two things: sending documents for indexing to Elasticsearch and requesting documents from an Elasticsearch index. The attack behavior sends manipulated requests to Elasticsearch to exploit the specific vulnerabilities.

3.4.4 CVE-2014-6271 (Shellshock)

The scenario for the Shellshock vulnerability contains users who request documents from an Apache web server similar to the the users in 3.4.1. The attacker send a manipulated request with an environment variable in the user-agent header which is erroneously evaluated by the server.

3.4.5 CVE-2016-6515 (DOS) and CVE-2015-5602 (Local Privilege Escalation)

For these scenarios, multiple users connect to the victim via ssh and create, move, or edit files. In case of the DOS scenario the attacker starts its attack via a long password in its request for a connection. In the privilege escalation attack the attacker creates a special symlink to a file he wants to edit or read without permission and edits its target via the symlink.

[9] Sqlmap, a sql injection penetration testing tool. http://sqlmap.org/

3.4.6 Zip Slip and CWE-434 (PS)

Each of these scenarios represents a service that processes files of a specific type. In the Zip Slip scenario the service accepts a folder as input and unpacks all zip files one after the other. In case of an attack, there is a manipulated zip file in the files to be processed which breaks out of the current folder and overwrites existing files on the system. In the second scenario a service converts all postscript files in the target folder to svg files one after the other. In the attack case, one of the ps files is manipulated in such a way that malicious code is executed on the system by the conversion.

3.5 Recording Process

To record the resulting system call traces, the victim is executed within the Docker[10] container virtualization software. This makes it possible to have a defined initial state of a victim and to jump back to it after each attack. The attack is executed by the host or other containers during network-based attacks. In the case of local attacks where the malicious code must have reached the victim beforehand, for example by email or social engineering, the victim itself runs the attack. The data is recorded from the host using the Sysdig[11] software, which directly supports Docker container inspection. Another advantage of Sysdig is that it automatically records the binary data buffer of events that have one, such as `read()` or `recvfrom()`, up to a specified length. We used the standard bufferlength of 80 from Sysdig.

For the recordings we use the LID-DS framework[12] developed by us. The central point of it is the control script. First, it launches the docker container, which hosts the victim. Then, depending on the scenario, initialization tasks are executed and the simulation of normal behavior is started. After that a short time is waited before Sysdig is activated. This warm-up time should prevent Sysdig from recording any startup effects of the victims software. In case of a recording of attack behaviour, the exploit is started after a random time. After the recording has run for the desired time, it is stopped by the control script. It also stops and removes the simulation of normal behavior and the used docker containers. A simplified representation of this process can be seen in Figure 5.

[10] Docker, software for operating system level virtualization. https://www.docker.com/
[11] Sysdig, a “Linux system exploration and troubleshooting tool”. https://github.com/draios/sysdig
[12] LID-DS framework for recording HIDS data. https://github.com/LID-DS/LID-DS

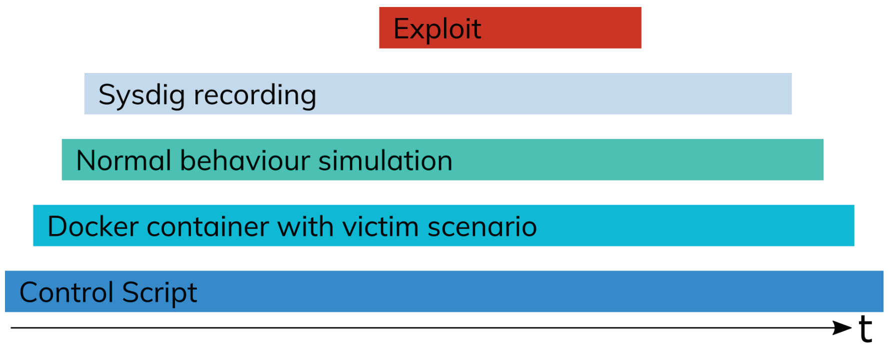
Exploit
Sysdig recording
Normal behaviour simulation
Docker container with victim scenario
Control Script
t

Figure 5: shows all steps done by the control script in order to record an attack scenario.

We recorded 1000 runs for each of the normal behaviour simulations and 100 runs for each attack described before. Each of these runs recorded about 30 seconds of system calls. Taken together, this results in 7000 normal traces and 1100 attack traces worth almost three days of recording. Each individual trace can contain up to several tens of thousands of system calls.

4. Results

Before we summarize the results, let us shortly discuss two observations. Figure 6 shows a small excerpt from LID-DS. As you can easily see in it, these records allow you to track the data flow between participating processes to a certain extent. Additionally we would like to pay special attention to the following observation. In LID-DS are sequences like the one shown in Figure 7. Obviously, these are two parallel sequences in different threads. Previous approaches based on the old data sets would interpret this as one sequence. This confirms the statements made in section 2 regarding the threadwise consideration of traces.

These two new features could not be extracted from the previous data sets.

```
TIME CPU PROCESS (PROCESS ID) ENTER(>)/EXIT(<) SYSCALL ARGUMENTS
...801301340 1 mysqld (5940) > recvfrom fd=41(<4t>127.0.0.1:54926->127.0.0.1:3306) size=102
...801302094 1 mysqld (5940) < recvfrom res=102 data=...mysql_native_pass... tuple=NULL
...801305987 0 apache2 (6264) < sendto res=106 data=...mysql_native_passw...
...801309746 1 mysqld (5940) > sendto fd=41(<4t>127.0.0.1:54926->127.0.0.1:3306) size=48 tuple=NULL
...801309909 0 apache2 (6264) > poll fds=12:431 timeout=1471228928
...801320710 1 mysqld (5940) < sendto res=48 data=,....mysql_native_password.jrz1$^/lVGCWk5cwJL![.
```

Figure 6: shows an shortened extract from a recorded trace including the metadata like arguments, data buffers and return values.

```
select t0, epoll_wait t1, accept t1, semop t1, semop t0, getsockname t1, epoll_wait t0, open t1

t0: select, semop, epoll_wait
t1: epoll_wait, accept, semop, getsockname, open
```

Figure 7: shows a sequence of system calls with their thread id and its interpretation as one or two sequences.

ith LID-DS we have created a HIDS dataset that solves the problems of older datasets mentioned in section 2 of this paper.

First: we have taken records of recent security vulnerabilities on modern Linux systems to be as up to date and realistic as possible.

Second: we have recorded the traces in such a way that the underlying thread information does not disappear, but can be used to evaluate new types of HIDS.

Third: the data of LID-DS contains the arguments of the system calls, their return values, time stamps, the corresponding process names, as well as a small excerpt of their data buffers. In particular, as far as we know, the data buffers have not been used in any HIDS so far.

Fourth: LID-DS also contains millions of data points to train modern machine learning techniques.

Finally, Figure 8 shows a simplified tabular comparison of the data sets mentioned in this paper. It shows at a glance which dataset contains which discussed feature.

	LID-DS	NGIDDS	ADFA-LD	UNM	KDD99
Process IDs	x	x		x	x
Arguments	x				x
Return values	x				x
Timestamps	x	(x)			x
Not outdated	x	x	x		
Data buffers	x				
Amount of data	x	x			x

Figure 8: Comparison of the HIDS data sets.

5. Conclusion

With the Leipzig Intrusion Detection – Data Set we make a significant contribution to future research, evaluation and comparability of Host Based Intrusion Detection Systems. With it known algorithms can be enhanced or new ones, based on the various included features, can be developed to improve such systems.

It is the first HIDS data set which contains system calls and their timestamps, thread ids, process names, arguments, return values and excerpts of their data buffers from traces of normal and attack behaviour of several recent, multi process, multi threaded scenarios. Many of the included features can not be extracted from previous data sets.

The link to the Leipzig Intrusion Detection – Data Set itself can be found at https://github.com/LID-DS/LID-DS.

Further information on the development of LID-DS and initial analyses can be found in the works of Ganz[14] and Kreußel[15].

Acknowledgements

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 16KIS0522K).

Special thanks to Martin Georgi from sprd.net AG for his feedback and his suggestion for one of the scenarios.

References

[1] University of New Mexico Computer Science Department Farris Engineering Center. *Computer Immune Systems - Data Sets and Software*. https://www.cs.unm.edu/~immsec/systemcalls.htm. [Online; accessed 12-May-2018]. (1999).

[2] Gideon Creech und Jiankun Hu. „Generation of a new IDS test dataset: Time to retire the KDD collection". In: *Wireless Communications and Networking Conference (WCNC), 2013 IEEE*.

[3] MASSACHUSETTS INSTITUTE OF TECHNOLOGY LINCOLN LABORATORY. *1998 DARPA Intrusion Detection Evaluation Data Set*. https://www.ll.mit.edu/ideval/data/1998data.html. [Online; accessed 12-May-2018]. (1998).

[4] MASSACHUSETTS INSTITUTE OF TECHNOLOGY LINCOLN LABORATORY. *1999 DARPA Intrusion Detection Evaluation Data Set*. https://www.ll.mit.edu/ideval/data/1999data.html. [Online; accessed 12-may-2018]. (1999).

[5] Federico Maggi, Matteo Matteucci und Stefano Zanero. „Detecting intrusions through system call sequence and argument analysis". In: *IEEE Transactions on Dependable and Secure Computing* 7.4 (2010).

[6] Darren Mutz et.al. „Anomalous system call detection". In: *ACM Transactions on Information and System Security (TISSEC)* 9.1 (2006).

[7] Marcus Pendleton und Shouhuai Xu. „A dataset generator for next generation system call host intrusion detection systems". In: *Military Communications Conference (MILCOM), MILCOM 2017-2017 IEEE*. (2017).

[8] Steven A Hofmeyr, Stephanie Forrest und Anil Somayaji. „Intrusion detection using sequences of system calls". In: *Journal of computer security* 6.3 (1998).

[9] Gideon Creech und Jiankun Hu. „A semantic approach to host-based intrusion detection systems using contiguousand discontiguous system call patterns". In: *IEEE Transactions on Computers* 63.4 (2014).

[10] Australian Center for Cyber Security (ACCS). *The ADFA Intrusion Detection Datasets*. https://www.unsw.adfa.edu.au/australian-centre-for-cybersecurity/cybersecurity/ADFA-IDS-Datasets/. [Online; accessed 12-May-2018]. (2013).

[11] Abed, Amr S., Charles Clancy, and David S. Levy. "Intrusion detection system for applications using linux containers." International Workshop on Security and Trust Management. (2015).

[12] Haider, W., et al. "Generating realistic intrusion detection system dataset based on fuzzy qualitative modeling." Journal of Network and Computer Applications 87 (2017).

[13] Deng, Shuang. "Empirical model of WWW document arrivals at access link." IEEE International Conference on Communications. INSTITUTE OF ELECTRICAL ENGINEERS INC (IEE), (1996).

[14] Simon Ganz. “Ein moderner Host Intrusion Detection Datensatz”. Master thesis, Leipzig University (2019)

[15] Dennis Kreußel. “Simulation and analysis of system call traces for adversial anomaly detection.”. Bachelor thesis, Leipzig University (2019)

[16] Martin Grimmer; Martin Max Röhling; Matthias Kricke; Bogdan Franczyk; Erhard Rahm, “Intrusion Detection on System Call Graphs“, 25. DFN-Konferenz “Sicherheit in vernetzten Systemen”, 2018.