GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN IN PUBLICA COMMODA SEIT 1737

HPS

HPCSA Seminar Report

Security Infrastructures and intrusion systems

Matthias Mildenberger

Supervisor: Artur Wachtel

Georg-August-Universität Göttingen
Institute of Computer Science

March 30, 2023

Abstract

In this paper, an overview of intrusion detection for high-performance clusters is provided. After a description of the architecture of high-performance clusters, aspects of intrusion detection that are particularly important in an HPC environment are outlined. A variety of differend IDS solutions is presented together with a more thorough description of the software wazuh, which enables comprehenseive and efficient intrusion detection. Afterwards four different practial intrusion detection use cases that are oriented on the problems of HPC are implemented with wazuh and it's results are presented. Finally, the limitations of IDS for HPC are pointed out as well as aspects for future work on the topic are given.

Contents

List of Figures

List of Abbreviations

ID Intrusion Detection

IDS Intrusion Detection System

SD Signature Based Intrusion Detection

AD Annomaly Based Intrusion Detection

DOS Denial of Service

HIDS Host-Based Intrusion Detection System

NIDS Network-Based Intrusion Detection System

HPC High-Performance Computing

HPCC High-Performance Computing Cluster

FIM File Integrity Monitoring

1 Introduction

In this term paper, an overview of intrusion detection capabilities is given. In particular, the special requirements in a high-performance computing cluster are discussed. Four different use cases are used to demonstrate how the software wazuh can be used to effectively detect intrusions.

1.1 Intrusion Detection

Intrusion detection (ID) defines the process of monitoring entities (those could be a network, system, or application) for unauthorized access and malicious activities. The goal of intrusion detection is to detect security incidents, before they cause significant damage or data loss[Bac00]. Intrusion detection is necessary because it is impossible to create a computer system that has no security vulnerabilities[Den87]. For this reason, one should at least be able to quickly detect attacks on a system in order to respond to them appropriately and in a timely manner. Intrusion detection can be distinguished from intrusion prevention (IP), which deals with how attacks can be prevented. To prevent an attack, however, it is first necessary to identify it as such. For this reason, ID and IP usually go hand in hand[Fuc05]. This work focuses mainly on detection of intrusion, although some intrusion detection systems can also contribute directly to preventing an attack. In intrusion detection, the security-relevant events of a system or network are continuously examined and checked for signs of malicious intent[Lia+13].

To detect the multitude of different possible attacks[Den87], two main-methodologies can be considered[Lia+13]: Signature based intrusion detection (SD) matches events with known signatures that have been stored in the context of attacks in the past. For this reason, Signature Based Intrusion Detection is also called knowledge based Intrusion Detection. Since SD requires that the possible security risks are known beforehand, knowledge based IDS will leave the defender always one step behind the potential attackers[MSD14]. In contrast to SD is Anomaly Based Intrusion Detection (AD). Here, a baseline is first determined from the stream of data, which represents normal behavior. Various algorithms can then find deviations from this normal behavior and detect them as intrusions. An example of AD is a Denial of Service (DOS) attack, in which an unusual amount of requests to a system is intended to prevent it from functioning normally[Lau+00].

In addition to the methodology for intrusion detection, a distinction can also be made in the objects that are examined for intrusions. Host-based intrusion detection systems (HIDS) and network-based intrusion detection systems (NIDS) are the most important of these, and are described in more detail below[Lia+13] HIDS focuses on detecting intrusions in a specific host. This could be a server or any endpoint device of the users of a bigger system[Ou+10; Lia+13]. HIDS involves collecting and examining the activity of applications, processes, or files on a host. This allows potential threats and attacks to be detected before they can do much harm to a system. For example HIDS could help by continuously examining files that are important for the operation of a system and detect and report changes that have been done to those files. NIDS describes the detection of intrusions in networks[MHL94] and can be used in various points in the network architecture. For example, it can be used to examine traffic caused by devices on a private network to the Internet. If there are unusual accesses from a certain IP address, the address could be blocked by a dynamic firewall[Fuc05]. In addition, there are also databases of known suspicious signatures that NIDS can use to scan traffic for malicious

access[Lia+13]. The techniques described can be applied in a single computing system as well as in high-performance computing environments. The particular challenges required in such contexts are described below.

1.2 High-Performance Computing

High Performance Computing (HPC) describes the use of advanced computing technologies and procedures to process and analyze large and complex data sets or to perform simulations that require a significant amount of computing power[IBM]. The possibility of scaling individual components (such as the clock speed of processors) is becoming increasingly difficult and will eventually be limited by physical circumstances such as the lowest possible size of semiconductors[GI12]. It is therefore not possible to perform computations of arbitrary complexity in a classical shared-memory system. In order to further increase the performance capacity, different computers can be connected together and form an HPC cluster (HPCC) to be able to solve even more complex problems[MS11]. HPCC will be described further below.

1.2.1 HPCC-Architecture

In the beginning of the history of high-performance computing, individual computers with very high computing capacity were used (so-called mainframes). Even today, these are still used in applications where very high reliability is required. In the field of HPC, however, systems with individual computers could no longer keep up with the increasing demands of computing tasks. The architecture of HPC developed in the direction that several computers (nodes) are connected with each other and form a cluster such that the the number of processor cores that can work simultaneously on a problem can be increased significantly[SAB18]. The individual components are loosely connected via a fast network and communicate with each other with messages[Hou+20; Gew]. In such a cluster, complex problems can be split into smaller sub-problems (provided that the programming of the individual applications allows for it) which can be processed in parallel by the nodes, so that only the results of the sub-computations need to be communicated between the nodes. This means that very complex problems can be solved with an HPCC[SAB18]. The top500[Str] lists the most powerful supercomputers in the world. The underlying computers that are involved in the clusters of the supercomputers can consist of various hardware components that could involve highly specific computers that only a few manufacturers can produce and would thus be very expensive to buy and maintain. The architecture of the Beowulf cluster however provides for the use of inexpensive, commercially available components in an HPCC, on which a normal Linux operating system can be installed. These nodes are networked with each other, which makes an HPCC easier to maintain, as faulty components can be replaced more easily[Bec+95].

In current HPCCs, a variety of different nodes are used that are optimised for specific use cases[Pei17; SAB18; LUM; Hou+20]. For example, individual nodes may be equipped with powerful GPUs to run matrix multiplications for machine learning in parallel, while other nodes have a particularly large shared memory area to analyse and visualise large amounts of data[LUM]. Usually, the various computing nodes are connected to a distributed file system on which large amounts of data can be stored. Since HPCCs are also used in the scientific field, scientists all over the world can thus be granted access to large computing capacities, the HPCCs must also be accessible from the public Internet. In order to administer the supercomputers (e.g. to install a new operating system on

all compute nodes), it is also necessary to have administration nodes that can access the cluster and have increased authorisation over the cluster. The structure of an HPCC is shown schematically in Figure 1, it can be seen, that a HPCC consists of heterogeneous nodes. Due to the architecture of an HPCC, which is accessible from the Internet, used by many different users and potentially consists of different components with partly heterogeneous hardware, some security gaps arise which must be considered in comparison to a conventional computer or server. The special security requirements that arise in the HPCC are considered in the following chapter.

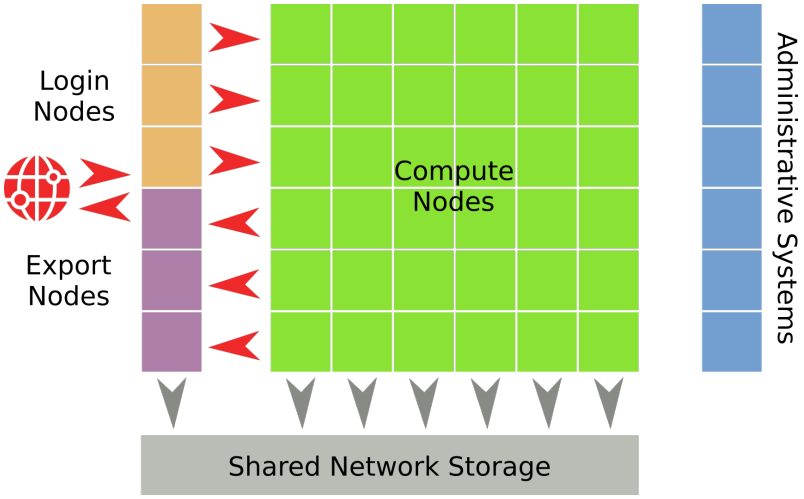

Figure 1: Exemplary architecture of a HPCC. Source:[EUD]

1.2.2 Intrusion detection for HPC

Since an HPCC consists of several individual computers, there are some parallels to security requirements that are also necessary on individual machines[Bul+18; Hou+20]. This includes decent hardening of the operating system and the infrastructure, such as setting up firewalls, enforcing secure user passwords, physically protecting the infrastructure or educating users about social-engineering attacks. In contrast to regular computers, HPCCs require additional security considerations or greater emphasis on certain aspects.

The first thing to mention here is the accessibility of an HPCC. Usually, a large number of users have access to an HPCC, which is significantly higher than with regular systems[Ada11; Pei17]. Due to the sheer number of users, it is difficult to find those with malicious intent among the many genuine users. In a HPCC, the regular users also have wide permissions and are able to allocate many resources, which makes it difficult to distinguish between a normal user and one who wants to abuse the hardware[Ada11]. Since the users of an HPCC can be widely distributed geographically, it is also important that the cluster is accessible from the public Internet, which makes it possible for malicious users to gain access to a cluster from outside in the first place. Malicious users can be distinguished between insiders and outsiders[MSD14]: Insiders have legitimate access to the HPCC which they misuse, outsiders gain access to the cluster without being authorised to do so (for example via stolen credentials). Since many users have legitimate access to an HPCC and their identities are not always carefully checked, such systems are particularly vulnerable to insider threats[Hou+20]. After a user has gained access to a computer, many security mechanisms such as firewalls do not have an effect anymore[MSD14] and

it is possible that they can use exploits or bugs to gain privileged access to the entire HPCC[MV13; EUD]. To manage the security risks posed by the large number of users of an HPCC, access control lists must be rigorously and carefully maintained and user credentials must be protected[Hou+20]. A last issue that the amount of users poses is the sheer number of security related alerts that a sysadmin might be getting from an IDS. To decrease the rate of false positives, proper filtering mechanisms should be considered.

Besides the amount of users, the heterogeneity of hardware and software is also a problem in an HPCC[Pei17; Ada11; Hou+20]. As the hardware is sometimes highly specialised[Hou+20], it can become a problem to replace obsolete hardware, as hardware manufacturers might stop supporting it or do not keep up with the delivery of security-related spare parts. This could expose vulnerabilities for attacks. An HPCC should also have a high level of availability, so it is particularly difficult to install security-critical software updates quickly and consistently on the entire cluster[Pei17]. The lack of important updates makes such an HPCC vulnerable not only to zero-day exploits, but also to those that have been known for some time and would actually already be fixed in a newer version of operating systems or softwares. Especially in such contexts where there is a greater risk due to potentially outdated hardware and software, it is very important to have a functioning intrusion detection system, as it's impossible to prevent all attacks[Den87]. With proper intrusion detection, the attacks that do take place can at least be identified quickly and the attackers can be deprived of access to the system again before greater damage can be done.

Several weaknesses in security systems have caused attacks on HPCCs to be successful in the past. In a forensic analysis of an attack on an HPCC in Germany, it turned out that the attacker was able to gain access to a user's ssh credentials and, due to an outdated operating system, was able to utilize a known kernel exploit to gain root privileges on the system and from there also gain access to other nodes in the HPCC[EUD]. Similar attacks are often carried out with misused ssh credentials[Laf20]. Once access to a system has been established, various flaws in hardware and software are often exploited to gain elevated privileges in the system[Hou+20; Rob04; onl17]. Thus, an important part of an IDS for HPCC is to quickly detect illegitimate privilege escalation of users to respond appropriately, and after an attack was carried out to find files manipulated by attackers that can cause further damage even after an attack. The Kobalos malware for example targets HPCC and modifies the ssh service of the machine that it infected in a way that the ssh credentials of the users who use the ssh service after the infection will be stolen[LS21].

1.3 Tools for Intrusion detection

There are various tools on the market that can be used for Intrusion Detection. There are commercial providers as well as open source software. In the case of NIDS, it primarily requires a log analyzer that can efficiently examine the network activity. For knowledge based intrusion detection a detailed rule set is important to detect as many suspicious signatures as possible. Freely available rule sets may be shared between different vendors[Sie21]. The Snort[Cisa] software was one of the first to make open source NIDS available to a wide audience. With Snort, as with many other vendors, it is possible to subscribe to rule sets in addition to the public rule sets, which differ mainly in that they are more up-to-date and integrate new vulnerabilities into the rule set more quickly[Cisb]. Suricata[Fou] is a NIDS, which is newer than Snort and has native support for multithreading, making it more efficient than Snort. In addition, Suricata can examine

network traffic at the application level as well, providing better ways to filter the traffic. Suricata, while being a free open source software, also provides the ability to subscribe to advanced rule sets for a monthly fee. The results of the analysis of Suricata can be stored in another logfile, which can be summarized and visualized by further analysis programs. The program Zeek[Pro] is another NIDS which, in contrast to Suricata and Snort, focuses more on anomaly-based intrusion detection.

On the HIDS side, Tripwire[For] can efficiently monitor the integrity of files and exists in a limited open-source version and an enterprise version that offers extended functionality and can be deployed on different platforms. The OSSEC[Tea] software can be deployed on different platforms and has a structure in which a central manager collects and analyzes the log information of the different agents (nodes monitored by tripwire). OSSEC provides multiple HIDS functionality starting from file integrity monitoring and rootkit detection. The software Wazuh[Inc] is a fork of OSSEC, because OSSEC is no longer actively developed. Wazuh thus provides a similar architecture to OSSEC and it can be helpful to look also into the documentation of OSSEC for some use cases. Wazuh also offers a graphical user interface (dashboard) in which the events of different systems can be conveniently displayed. Thus, it is also possible to integrate a NIDS like Suricata into the interface of Wazuh and thus to access the advantages of both NIDS and HIDS. Wazuh also provides for horizontal scaling of the management node making it a good choice for big systems like in an HPC context. In this thesis wazuh is used as a central IDS platform and on the basis of different use cases it is examined how Wazuh can efficiently detect intrusions.

1.3.1 Wazuh

Wazuh is an open-source security platform that provides intrusion detection, threat hunting, log management, and compliance monitoring capabilities. Its architecture is depicted in figure 2. As shown in the diagram, wazuh consists of the following components:

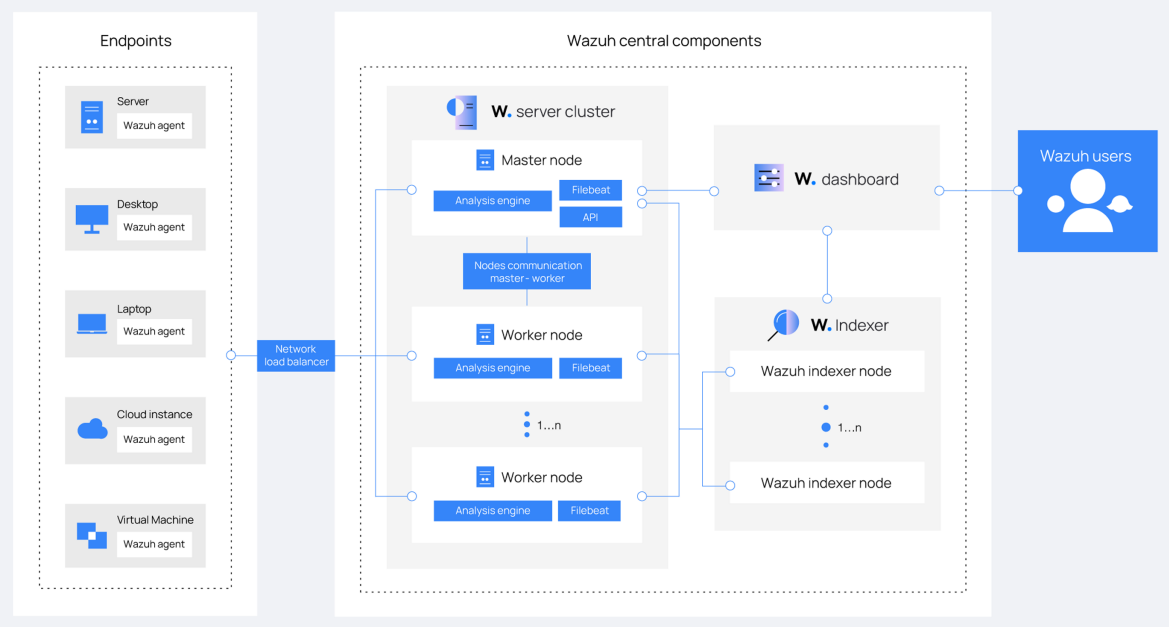

Figure 2: Architecture of Wazuh. Source:[Inc]

Agents: The wazuh agent is a lightweight software package that is installed on various

hosts (endpoints) that are supposed to be monitored. The wazuh agent collects data such as system logs, configuration files and system events that are specified in the configuration file that is located in `/var/ossec/etc/ossec.conf`. Wazuh agents are sending the collected security relevant events continuously to the wazuh server where they are analyzed. The messages of the agents are sent to the server with a secure channel (TCP or UDP) using TLS encryption. The wazuh agents can also be configured to automatically engage in an active response when threats are detected (such as stopping a process or blocking network connections).

Server: The wazuh server keeps track of all the agents and can configure and update them. The server collects and analyzes the data received from the agents. It can for example check if the incoming security events trigger pre-configured rules that indicate a threat. The server itself can be seperated into different master and worker nodes to be able to balance the load from the incoming agents and scale to a big cluster. The alert and event data that the server receives is then again sent to the wazuh indexer. The wazuh server also runs the wazuh REST API, that can be used to programatically interact with the wazuh infrastructure to configure settings and rules for agents or monitor the overall health of the cluster.

Indexer: The wazuh indexer is a scalable, full-text search and analytics engine. The security events from the server are stored by the indexer and can afterwards be queried efficiently to provide for a near real-time search. The data is stored as JSON documents that can be distributed across different shards, providing for a scalable solution. The data stored by the indexer can then be presented and observed in the wazuh dashboard or fetched via a REST API.

Dashboard: The wazuh dashboard is the central graphical user-interface of wazuh. In the dashboard, the collected data is presented in a structured and clear manner. The status and configuration of wazuh can be monitored as well as the summarized security events in the cluster. The dashboard also provides an API console that facilitates communication with the wazuh REST API.

Furthermore the documentation of wazuh provides a list of use cases and predefined proof of concept implementations for important use cases that could arise during the operation of a HPC.

2 Experiments

In the following section, wazuh is tested in a virtual environment. For this purpose, various security-relevant use cases derived from the literature are evaluated in wazuh. In each case it is stated how the detection of the threat can be configured in wazuh and how well actual threats can be detected.

2.1 Testing Environment

Two virtual machines are created in Virtualbox for the test environment. Both virtual machines (nodes) have Ubuntu 22.04.1 LTS as their operating system. One node will act as a wazuh server and the other as a wazuh agent. The wazuh server has 4GB memory allocated, the agent 1GB. Both nodes have access to the internet and are connected in a virtual local network, which allows them to communicate with each other. On both nodes

wazuh version 4.3.10 was installed, on the agent node wazuh runs in agent mode, on the server node it runs in server mode.

2.2 Use Cases

2.2.1 Use Case 1: File integrity monitoring

File Integrity Monitoring (FIM) is an important aspect of HIDS, it allows for the detection of unauthorized changes or modifications to critical system files and directories. Changes to critical binaries such as `/usr/bin/ls` or global configuration files like `/etc/bash.bashrc` that might add malicious aliases for common commands could be detected quickly using proper FIM. The exact names of the files that should be checked for integrity may be different for several linux distributions or even agents that run another operating system like Windows.

Wazuh has a FIM component that can generate alerts when changes occur in the file system. For FIM, wazuh uses `syscheck` which stores cryptographic checksums and periodically compares them with the current files. If a checksum changes, wazuh sends an alert to the wazuh server. By default each agent has `syscheck` enabled and preconfigured with sane defaults, but custom changes can easily be made in `/var/ossec/etc/ossec.conf` within the `<syscheck>` block. The configuration can be done on the server centralized for all agents as well as on a single agent. Not only single files but also whole folders can be specified in the configuration. If a folder or file is specified in the configuration of both the server and an agent, the rule from the server has priority over the one on the agent and overwrites the local configuration. Beside the content of a file also the permissions, hard-links or checks-sums can be monitored. In an FIM event of wazuh it can not only reported that a file has changed, but also information about which user or process caused the change (who-data), this is provided by the linux audit subsystem. The `syscheck` process can be assigned a nice-value, which can give the process a higher priority in the operating system, so that the syscheck is given priority over the other processes on the machine. The nice-value can take values between -20 (highest priority) and 19 (lowest priority), by default it is set to 10. It is also possible, in addition to periodically scheduled checks (e.g. once a week), to monitor directories in real time to be informed about changes as soon as possible.

To monitor the integrity of the files in the `/home/matthias/check_dir` folder, the following statement can be added to the `sysscheck` block of `/var/ossec/etc/ossec.conf` on either the wazuh manager or a node:

```
<directories
    check_all="yes"
    report_changes="yes"
    realtime="yes"
    whodata="yes">
  /home/matthias/check_dir
</directories>
```

The different options have the following effects:

- `check_all`: Integrity check is performed with all possible parameters (file size, permissions, owner, modification date, checksums)

- `report_changes`: The changes made to the file are indicated in the alert (sensitive information about the file contents may be stored in wazuh).
- `realtime`: The monitoring is done in real time, as soon as a change is made to a file it is immediately reported to wazuh.
- `whodata`: Information about the user who modified the file (e.g. the username and primary group of the user who modified the file or the username and group with which the user originally logged in) is also provided in the wazuh alert. Here the audit subsystem of Linux is used.

Once the new instructions have been added to the config of wazuh and the wazuh agent or manager (depending on where the instructions have been added) has been restarted with `systemctl restart`, the specified folder will be monitored. As soon as a file in the folder is changed, wazuh sends an alert. The raw data of the alerts can be found on the manager in `/var/ossec/logs/alerts/alerts.log`. To cause the generated alert in the following example log, the file `/home/matthias/check_dir/test.txt` was modified by the user `matthias` who has the primary group `hpc_user`. In doing so, the user used `sudo`, so the effective username that changed the file is `root`. The user added the string `change with sudo` to the file. All the corresponding information can be seen in the log that was produced by wazuh. Both a json and a text version of the alerts are generated. The online version of the dashboard uses the json file to clearly display the alerts

```
** Alert 1679584603.96315: - ossec,syscheck,syscheck_entry_modified,
    syscheck_file,pci_dss_11.5,gpg13_4.11,gdpr_II_5.
1.f,hipaa_164.312.c.1,hipaa_164.312.c.2,nist_800_53_SI.7,tsc_PI1.4,
    tsc_PI1.5,tsc_CC6.1,tsc_CC6.8,tsc_CC7.2,tsc_CC7.3
,
2023 Mar 23 16:16:43 (hpcsanode) any->syscheck
Rule: 550 (level 7) -> 'Integrity checksum changed.'
File '/home/matthias/check_dir/test.txt' modified
Mode: whodata
Changed attributes: size,mtime,sha256
Size changed from '246' to '263'
Old modification time was: '1679576782', now it is '1679584603'
Old sha256sum was: 'd417768c3677cd7[...]b529e80ddc'
New sha256sum is : '53dc75e160495[...]11feddd7fbf6fed2bcc47a1bb'

Attributes:
 - Size: 263
 - Permissions: rwxrwxrwx
 - Date: Thu Mar 23 16:16:43 2023
 - Inode: 51
 - User: matthias (1000)
 - Group: hpc_user (1001)
 - MD5: 61f59b531499a1e42257233801f04f70
 - SHA1: e7227e0de43c8d1190da9f0715266988d44e5635
 - SHA256: 53dc75e160495df2ac[...]2e11feddd7fbf6fed2bcc47a1bb
 - (Audit) User name: root
 - (Audit) Audit name: matthias
```

```
     - (Audit) Effective name: root
     - (Audit) Group name: root
     - (Audit) Process id: 2267
     - (Audit) Process name: /usr/bin/nvim
     - (Audit) Process cwd: /home/matthias/check_dir
     - (Audit) Parent process name: /usr/bin/sudo
     - (Audit) Parent process id: 2266
     - (Audit) Parent process cwd: /home/matthias/check_dir

    What changed:
    17a18
    > change with sudo
```

The previously created directives to monitor folders can be extended and restricted with user-defined rules. For this purpose rules can be created in `/var/ossec/etc/rules/local_rules.xml`. Wazuh reserves the range with IDs from 100000 to 120000 for custom rules of the user. The following rule gets the ID 101337 and a security level 0, which means that no alerts are generated for applications of this rule. The rule is triggered if a sent alert had the original ID 550 (File integrity changed). Now it is possible to filter for attributes that the FIM uses in its alerts. This new rule is applied if the parent folder is `/home/matthias/check_dir` and the user who changes a file has the primary group `hpc_admin`. Also, the special rule is used only when a file is modified. This means that no alerts are sent when a user of the group `hpc_admin` changes a file in the folder, but they are sent, when he creates or deletes a new one.

```
<rule id="101337" level="0">
  <if_sid>550</if_sid>
  <field name="parent_cwd">/home/matthias/check_dir</field>
  <field name="group_name">hpc_admin</field>
  <match>modified</match>
  <description>Silence modifications of files in check_dir
    by users with primary group hpc_admin</description>
</rule>
```

Another way to filter the amount of alerts an admin gets to see is to send custom queries to the indexer, which stores the alerts. These can be entered in the dashboard, it's also possible to use the REST API of the indexer to make the queries programmatically. The queries can be written in the human-readable Dashboard Query Language (DQL). The following rule restricts the alerts displayed so that only alerts with ID 550 (File Integrity changed) are displayed, where the login username is different from the effective username that changes the file. This happens for example when a user gets root privileges or changes a file with sudo.

```
Query = 'rule.id:550 and
  not syscheck.audit.login_user.name:syscheck.audit.effective_user.name'
```

2.2.2 Use Case 2: Malicious Access

As already described, it is particularly important in an HPC context to recognize when a normal user is given access to privileges to which he is not actually entitled (privilege

escalation). To prevent users from gaining unauthorized access, it is necessary that the system is secure. This means that known exploits of the installed software are fixed as soon as possible. For this, an update policy is very important, which ensures that known security holes are closed quickly. However, complete security against all possible security vulnerabilities can never be guaranteed, so-called zero-day exploits, which are not yet known to the public, can always cause that users could gain unauthorized access. In this case, it is important for an IDS to quickly detect when a user has gained such access in order to be able to initiate appropriate measures. If a malicious user gains access to the system, he may be able to exploit vulnerabilities of the computer and install rootkits, with the help of which he may gain further access to the system in the future and cover his tracks. For an IDS it is also an important task to reliably detect such rootkits.

To prevent a malicious user from gaining access to a system in the first place, security configuration assessment (SCA) is important. SCA ensures that all systems follow predefined rules that can reduce the attack surface. Wazuh has a SCA component that can detect misconfigurations and vulnerabilities in the nodes. For this purpose, wazuh uses different policy files that define the rules that have to be followed. These could include how the passwords policies are configured on the system or which crypto algorithms are allowed to be used. The policies are stored as `.yaml` files and can be found in `/var/ossec/ruleset/sca`. Out-of-the-box Wazuh delivers the SCA benchmark from the Center for Internet Security (CIS)[Cen], which checks many basic policies. Other policies that run advanced SCA can be installed additionally[Mar19]. To enable SCA, the following block in `/var/ossec/etc/ossec.conf` is enabled by default.

```
<sca>
    <enabled>yes</enabled>
    <scan_on_start>yes</scan_on_start>
    <interval>12h</interval>
    <skip_nfs>yes</skip_nfs>
</sca>
```

If an attacker has managed to gain access to a system despite SCA, it is important to quickly detect a possible privilege escalation that the attacker could use, for example, to install rootkits. Wazuh can monitor if a user uses the `sudo` command to escalate his privileges. In order to do that, wazuh can examine the logs in `/var/log/auth.log` and report an alert if a user uses `sudo`. The auth logs are included in wazuh with the following block:

```
<localfile>
    <log_format>syslog</log_format>
    <location>/var/log/auth.log</location>
</localfile>
```

In this way logs from different sources can be integrated into wazuh. To analyze the logfiles wazuh uses so-called decoders, which usually apply regular expression patterns to read the relevant information from an incoming log message. The decoded information is then passed to wazuh rules that generate the alerts. The rules can also be customized as shown in the previous section. If custom decoders are created, they must be added to `/var/ossec/etc/decoders/local_decoder.xml`. As described above, custom rules are defined in `/var/ossec/etc/rules/local_rules.xml`. In the following example, a

custom decoder and rule is created to detect users that are escalating their priviliges using the fictional `sudu` command. Only when the user `hackerman` uses the command, an alert will be sent.

```
<decoder name="sudu_example">
    <program_name>^sudu</program_name>
    <regex>User '(\w+)' used sudu to attack '(\w+)'</regex>
    <order>srcuser, system_name</order>
</decoder>

<rule id="100010" level="10">
    <program_name>sudu</program_name>
    <decoded_as>sudu_example</decoded_as>
    <user>hackerman</user>
    <description>Hackerman attacked $(system_name)!</description>
</rule>
```

To test the created decoders and rules wazuh provides the binary `/var/ossec/bin/wazuh-logtest`. This can be used to test and debug encoders and rules in a sandbox environment. `wazuh-logtest` is used in the following to test the previously defined decoder and rule. Since the logs are following the syslog protocol[Ger09], wazuh is able to extract information such as the date and name of the used program in a pre-decoding phase.

```
Starting wazuh-logtest v4.3.10
Type one log per line

Jul 26 20:42:07 MyHost sudu[12345]: User 'hackerman' used sudu to
    attack 'nuclearMissiles'

**Phase 1: Completed pre-decoding.
    full event: 'Jul 26 20:42:07 MyHost sudu[12345]: User 'hackerman'
        used sudu to attack 'nuclearMissiles'  '
    timestamp: 'Jul 26 20:42:07'
    hostname: 'MyHost'
    program_name: 'sudu'

**Phase 2: Completed decoding.
    name: 'sudu_example'
    dstuser: 'hackerman'
    system_name: 'nuclearMissiles'

**Phase 3: Completed filtering (rules).
    id: '100010'
    level: '10'
    description: 'Hackerman attacked nuclearMissiles!'
    groups: '['local', 'syslog', 'sshd']'
    firedtimes: '1'
    mail: 'False'
**Alert to be generated.
```

Wazuh has predefined decoders to recognize `sudo` commands in the `auth.log` in `/var/ossec/ruleset/decoders/0320-sudo_decoders.xml`. To detect the execution of `sudo` commands, wazuh has also already predefined a ruleset in `/var/ossec/ruleset/rules/0020-syslog_rules.xml`. For example, a more urgent alert is sent when a user executes a sudo command for the first time. As with the FIM, custom rules can also be written that can be tailored to the user's use cases to detect even more granular privlege escalation. In addition to authentications that can be tracked via `/var/log/auth.log`, it is also possible to monitor other system calls that users initiate. This could be used to monitor the use of existing rootkits that circumvent the explicit usage of sudo. To do this, the `audit` package must be installed on the system. The `/var/log/audit/audit.log` is filled by the the `audit` package and can be included in wazuh as a `localfile`, then rules can be defined as to which accesses and system calls are supposed to result in an alert.

If a user with elevated privileges or knowledge of system vulnerabilities and exploits gained access, they could install rootkits to exploit the system in the future. Rootkits are malware that usually modify parts of the operating system in such a way that other processes, files or connections can be hidden. Among other things, they can also be used to set up so-called backdoors, through which an attacker can continue to gain access to the system even after his successful attack. The rootkit-detection module of wazuh has two different approaches to detect rootkits: On the one hand it compares bottom-up the outputs of system binaries like `ps` with executed system calls to check if there are processes that are not displayed with `ps`, open ports that are hidden from `netstat`, suspicious files owned by the `root` user and files that have been hidden so that they cannot be found with `ls`[Waz]. On the other hand, wazuh utilizes a top-down database of well known rootkit signatures and traces of trojans. This database needs to be kept updated, such that new exploits can be detected as fast as possible. Though wazuh already comes with a built in database of signatures, they can be extended by the user. By using knowledge of how known rootkits, such as the rootkit 'reptile', modify the system, wazuh's bottom-up methods can detect if there are traces that might signal that an attack has occurred[Bas22].

2.2.3 Use Case 3: Brute-force attack

In a brute-force attack, an attacker tries to guess the correct data by quickly trying out various possible access credentials and thus obtain access to a system once the correct pair of credentials has been found. For example, if users use insecure passwords, brute-force attacks can be successful. Wazuh offers the possibility to detect such attacks and also to react actively to them. Thus wazuh already offers the possibilities of an intrusion prevention system. Predefined active response scripts are located in `/var/ossec/active-response/bin/`, for example `firewall-drop`. With this binary wazuh can block certain IP addresses with iptables. To do this, the script or binary must be added as a command in `/var/ossec/etc/ossec.conf`:

```
<command>
    <name>firewall-drop</name>
    <executable>firewall-drop</executable>
    <timeout_allowed>yes</timeout_allowed>
</command>
```

Under `<executable>` scripts or binary executables can be stored which will be executed when the command is activated. With `<timeout_allowed>` it is possible to

set a timeout for the command, after which the command will be reversed (this requires the command to be reversible). Custom executables can also be defined. To set up the active response to a brute-force attack, the command must also be added in `/var/ossec/etc/ossec.conf`:

```
<active-response>
    <command>firewall-drop</command>
    <location>local</location>
    <rules_id>5763</rules_id>
    <timeout>180</timeout>
</active-response>
```

This block is used to invoke the firewall-drop command when rule 5763 is triggered. `location=local` specifies that the response takes place on the respective node, `timeout=180` specifies that the blocked attacker is unblocked again after 180 seconds. Rule 5763 detects brute-force attacks with ssh, the rule is defined in `/var/ossec/ruleset/rules/0095-sshd_rules.xml` and is activated if within 120 seconds a source IP address enters a wrong password at least eight times during an ssh login attempt. To simulate a brute-force attack the software hydra[van21] can be used. Hydra was installed on a third virtual machine to act as the attacker. The command `sudo hydra -t 4 -l matthias -P passwords.txt 10.102.66.200 ssh` is used to start the attack on the node with IP 10.102.66.200 with a list of common passwords. After a short time, rule 5763 is triggered and the attacker with IP 10.102.66.202 is blocked, as can be seen in the node's iptables:

```
>> sudo iptables -L
    Chain INPUT (policy ACCEPT)
    target     prot opt source               destination
    DROP       all  --  10.102.66.202        anywhere

    Chain FORWARD (policy ACCEPT)
    target     prot opt source               destination
    DROP       all  --  10.102.66.202        anywhere

    Chain OUTPUT (policy ACCEPT)
    target     prot opt source               destination
```

It is now no longer possible for the attacker to reach the node. After the timeout of 180 seconds the attacker is automatically deleted from the iptables and the node is reachable again.

2.2.4 Use Case 4: Malicious signatures of network traffic

The best way to examine the traffic in the network is to use NIDS. Wazuh allows to integrate the output of different sources, also from a NIDS like Suricata. For this purpose Suricata has to be installed on the node at first. Then it can be configured which network interface should be monitored by Suricata. To prevent the nodes themselves from having to spend the computing power to monitor the traffic, Suricata could also run on a separate machine that monitors the whole network's traffic. Suricata matches the traffic with predefined rules, by default the emerging-threats rule set is used here[Inc23].

Custom rule sets can be added. Once Suricata is installed and set up, alerts are logged in `/var/log/suricata/eve.json`. As described before, the suricata logs can be included in wazuh as a local json file.

```
<localfile>
  <log_format>json</log_format>
  <location>/var/log/suricata/eve.json</location>
</localfile>
```

To verify that the NIDS is working, the command `curlhttp://testmynids.org/uid/index.html` can be used to send a request to a sample endpoint that generates a suspicious response. Suricata recognizes the signature of the network response as suspicious and generates an alert log that looks like the following[1]:

```
{
    "timestamp": "2023-01-10T19:42:22.984130+0100",
    "event_type": "alert",
    "src_ip": "18.155.145.16",
    "src_port": 80,
    "dest_ip": "10.0.2.15",
    "dest_port": 43928,
    "proto": "TCP",
    "alert": {
        "signature": "GPL ATTACK_RESPONSE id check returned root",
        "severity": 2}
}
```

In `/var/ossec/ruleset/rules/0475-suricata_rules.xml` generic rules are already predefined by wazuh, so that Suricata alerts can be converted to wazuh alerts as soon as the event_type `alert` is present in the JSON of Suricata. As described in the previous sections, custom decoders and rules can now be created that can analyze the Suricata logs at a higher resolution. These may also trigger an active response for specific alerts. For example, suricata and wazuh can be used to respond automatically to denial of service attacks[Odi22].

3 Discussion

Intrusion detection offers an important building block in the security of a system. With modern tools such as wazuh, a scalable IDS solution can also be used for HPC environments, in which the manager node that inspects the logs can also be scaled horizontally. Nevertheless, IDS is only one part of an integrative security solution[Luo+19], which must be complemented by other aspects. It is important to follow strict policies that mitigate vulnerabilities of a system. In the best case, the IDS does not need to report alerts at all because the system is that secure. There are also some vulnerabilities that an IDS cannot directly help against. Social engineering and phishing are two gateways that cannot be addressed by technical means. Everyone who has access to a secure system must

[1] some information has been truncated from the json for better readability

be trained in how to protect themselves from such attacks. Systems can also be architecturally secured better so that some intrusions are no longer possible in the first place. For example, new forms of virtualization can be used such that users may no longer be technically able to gain root privileges on a system at all[Pei17].

However, intrusion detection itself still has weaknesses and areas that could be expanded. Sometimes IDS could be too slow; during an attack, seconds can decide whether an attacker can cause lasting damage by stealing important data or installing ransomware. A human being who has to respond to an intrusion can at most respond in minutes or hours, which would be too slow in such cases. Automated responses to intrusions would therefore be required, but these would also have to be weighed up in terms of when exactly they should be used, so that a system does not shut itself down for no reason in the event of a false positive. Furthermore, performance must also be taken into account with IDS. In an HPC environment in particular, users expect high performance and this should not be impaired too much by IDS. Especially with fast networks there are still unsolved problems[CM11]. It is also not possible to check everything with an IDS. It is unrealistic to scan every folder for hidden folders via system calls or to detect all hidden processes. Therefore, even with the best IDS, it is even more important to have an up-to-date knowledge base that can point the IDS in the right directions. In intrusion detection there are also other aspects and possibilities that have not been considered in this work. There are special requirements, if one considers wireless data transmissions[Lia+13], the attack vector does not have to lie in a HPC system, but can lie also already in the WIFI network of the user, when he is connected to an unsecured public network. To increase the performance of an IDS it could be useful to use extended berkeley packet filters (ebpf), which an IDS could integrate directly into the operating system and thus run the operations more efficiently[WC22]. Wazuh works primarily with log file analyses. Here, the individual log-lines were only analyzed individually, for a better IDS, however, more sophisticated methods could be applied, where logs from different sources could be integrated with each other[Luo+19]. In addition, it would be possible to apply machine learning methods that could utilize additional input data to support intrusion detection[Gao+19; Ahm+21].

3.1 Conclusion

In this paper, first an overview of intrusion detection is provided and a description of how high-performance clusters are designed is given. Then it was analyzed which aspects of intrusion detection in HPC are particularly important. After an overview of different IDS solutions, the software wazuh, which enables comprehensive and efficient intrusion detection, was described in more detail. With the help of four different use cases, which are based on the requirements of IDS in HPC, the core functionalities of wazuh were practically demonstrated. Finally, limitations of IDS were pointed out and approaches were given, which aspects can be further investigated in the future. Overall, this paper highlights the significance of IDS in HPC clusters and the potential of Wazuh as a valuable tool for safeguarding sensitive data in these settings.

References

[Ada11] Dewayne Adams. *Six Security Risks in High Performance Computing (HPC)*. Nov. 2011. (Visited on 03/07/2023).

[Ahm+21] Zeeshan Ahmad et al. "Network Intrusion Detection System: A Systematic Study of Machine Learning and Deep Learning Approaches". In: *Transactions on Emerging Telecommunications Technologies* 32.1 (Jan. 2021). ISSN: 2161-3915, 2161-3915. DOI: 10.1002/ett.4150. (Visited on 03/04/2023).

[Bac00] Rebecca Gurley Bace. *Intrusion Detection*. Sams Publishing, 2000.

[Bas22] Chris Bassey. *Using Wazuh Rootcheck to Detect Reptile Rootkit | Wazuh | The Open Source Security Platform*. https://wazuh.com/blog/using-wazuh-rootcheck-to-detect-reptile-rootkit/. Aug. 2022. (Visited on 03/26/2023).

[Bec+95] Donald J Becker et al. "BEOWULF: A Parallel Workstation for Scientific Computation". In: *Proceedings, International Conference on Parallel Processing*. Vol. 95. 1995, pp. 11–14.

[Bul+18] Ramesh Bulusu et al. "Addressing Security Aspects for HPC Infrastructure". In: *2018 International Conference on Information and Computer Technologies (ICICT)*. DeKalb, IL: IEEE, Mar. 2018, pp. 27–30. ISBN: 978-1-5386-5382-1 978-1-5386-5384-5. DOI: 10.1109/INFOCT.2018.8356835. (Visited on 12/20/2022).

[Cen] Center for Internet Security. *Benchmarks Overview - CIS®*. https://www.cisecurity.org/cis-benchmarks-overview/. (Visited on 03/24/2023).

[Cisa] Cisco. *Snort - Network Intrusion Detection & Prevention System*. https://www.snort.org/. (Visited on 03/09/2023).

[Cisb] Cisco. *What Are the Differences in the Rule Sets?* https:// www.snort.org/faq/what-are-the-differences-in-the-rule-sets. (Visited on 03/09/2023).

[CM11] Scott Campbell and Jim Mellander. "Experiences with Intrusion Detection in High Performance Computing". In: (2011).

[Den87] D.E. Denning. "An Intrusion-Detection Model". In: *IEEE Transactions on Software Engineering* SE-13.2 (Feb. 1987), pp. 222–232. ISSN: 0098-5589. DOI: 10.1109/TSE.1987.232894. (Visited on 03/04/2023).

[EUD] EUDCV. *Analyzing a Compromised HPC Cluster*. https://www.educv.de/blog/post-2021-02-17-analyzing-a-compromised-hpc-cluster//. (Visited on 03/04/2023).

[For] LLC Fortra. *Tripwire | Security and Integrity Management Solutions*. https://www.tripwire.com/. (Visited on 03/09/2023).

[Fou] Open Information Security Foundation. *Home - Suricata*. https://suricata.io/. (Visited on 03/09/2023).

[Fuc05] Andreas Fuchsberger. "Intrusion Detection Systems and Intrusion Prevention Systems". In: *Information Security Technical Report* 10.3 (Jan. 2005), pp. 134–139. ISSN: 13634127. DOI: 10.1016/j.istr.2005.08.001. (Visited on 03/04/2023).

[Gao+19] Xianwei Gao et al. “An Adaptive Ensemble Machine Learning Model for Intrusion Detection”. In: *IEEE Access* 7 (2019), pp. 82512–82521. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2923640. (Visited on 03/04/2023).

[Ger09] R. Gerhards. *The Syslog Protocol*. RFC 5424. RFC Editor / RFC Editor, Mar. 2009.

[Gew] David Gewirtz. *Massively Distributed Computing Using Computing Fabrics :: DominoPower*. (Visited on 03/06/2023).

[GI12] Pablo García-Risueño and Pablo E. Ibáñez. “A REVIEW OF HIGH PERFORMANCE COMPUTING FOUNDATIONS FOR SCIENTISTS”. In: *International Journal of Modern Physics C* 23.07 (July 2012), p. 1230001. ISSN: 0129-1831, 1793-6586. DOI: 10.1142/S0129183112300011. (Visited on 03/04/2023).

[Hou+20] Tao Hou et al. “Autonomous Security Mechanisms for High-Performance Computing Systems: Review and Analysis”. In: *Adaptive Autonomous Secure Cyber Systems*. Ed. by Sushil Jajodia et al. Cham: Springer International Publishing, 2020, pp. 109–129. ISBN: 978-3-030-33431-4 978-3-030-33432-1. DOI: 10.1007/978-3-030-33432-1_6. (Visited on 03/07/2023).

[IBM] IBM. *What Is HPC? Introduction to High-Performance Computing | IBM*. https://www.ibm.com/topics/hpc. (Visited on 03/04/2023).

[Inc] Wazuh Inc. *The Open Source Security Platform | Wazuh*. https://wazuh.com/. (Visited on 03/09/2023).

[Inc23] Proofpoint Inc. *Proofpoint Emerging Threats Rules*. https://rules. emergingthreats.net/open/. 2023. (Visited on 03/27/2023).

[Laf20] Lev Lafayette. *Monitoring HPC Systems Against Compromised SSH | Lev Lafayette*. https://www.levlafayette.com/node/690. Oct. 2020. (Visited on 03/07/2023).

[Lau+00] F. Lau et al. “Distributed Denial of Service Attacks”. In: *SMC 2000 Conference Proceedings. 2000 IEEE International Conference on Systems, Man and Cybernetics. 'Cybernetics Evolving to Systems, Humans, Organizations, and Their Complex Interactions' (Cat. No.00CH37166)*. Vol. 3. Nashville, TN, USA: IEEE, 2000, pp. 2275–2280. ISBN: 978-0-7803-6583-4. DOI: 10.1109/ICSMC.2000.886455. (Visited on 03/04/2023).

[Lia+13] Hung-Jen Liao et al. “Intrusion Detection System: A Comprehensive Review”. In: *Journal of Network and Computer Applications* 36.1 (Jan. 2013), pp. 16–24. ISSN: 10848045. DOI: 10.1016/j.jnca.2012.09.004. (Visited on 03/04/2023).

[LS21] Marc-Etienne M Léveillé and Ignacio Sanmillan. “A WILD KOBALOS APPEARS”. In: (Jan. 2021).

[LUM] LUMI. *LUMI Supercomputer*. (Visited on 03/06/2023).

[Luo+19] Zhengping Luo et al. “Security of HPC Systems: From a Log-analyzing Perspective”. In: *ICST Transactions on Security and Safety* 6.21 (Aug. 2019), p. 163134. ISSN: 2032-9393. DOI: 10.4108/eai.19-8-2019.163134. (Visited on 03/28/2023).

[Mar19] Chema Martínez. *Security Configuration Assessment (SCA) | Wazuh | The Open Source Security Platform*. https://wazuh.com/blog/security-configuration-assessment/. Nov. 2019. (Visited on 03/24/2023).

[MHL94] B. Mukherjee, L.T. Heberlein, and K.N. Levitt. "Network Intrusion Detection". In: *IEEE Network* 8.3 (May 1994), pp. 26–41. ISSN: 0890-8044. DOI: 10.1109/65.283931. (Visited on 03/04/2023).

[MS11] A Middleton and PDLR Solutions. "Hpcc Systems: Introduction to Hpcc (High-Performance Computing Cluster)". In: *White paper, LexisNexis Risk Solutions* (2011).

[MSD14] Brojo Kishore Mishra, Minakshi Sahu, and Satya Naryan Das. "Intrusion Detection Systems for High Performance Computing Environment". In: *2014 International Conference on High Performance Computing and Applications (ICHPCA)*. Bhubaneswar, India: IEEE, Dec. 2014, pp. 1–6. ISBN: 978-1-4799-5958-7 978-1-4799-5957-0. DOI: 10.1109/ICHPCA.2014.7045369. (Visited on 12/22/2022).

[MV13] Alex Malin and Graham Van Heule. "Continuous Monitoring and Cyber Security for High Performance Computing". In: *Proceedings of the First Workshop on Changing Landscapes in HPC Security*. New York New York USA: ACM, June 2013, pp. 9–14. ISBN: 978-1-4503-1984-3. DOI: 10.1145/2465808.2465810. (Visited on 02/28/2023).

[Odi22] Ifeanyi Onyia Odike. *Responding to Network Attacks with Suricata and Wazuh XDR | Wazuh | The Open Source Security Platform*. https://wazuh.com/blog/responding-to-network-attacks-with-suricata-and-wazuh-xdr/. Nov. 2022. (Visited on 03/27/2023).

[onl17] heise online. *BitBang: Vor 30 Jahren kam der "NASA-Hack" ans Licht.* https://www.heise.de/newsticker/meldung/BitBang-Vor-30-Jahren-kam-der-NASA-Hack-ans-Licht-3832471.html. Sept. 2017. (Visited on 03/06/2023).

[Ou+10] Yang-jia Ou et al. "The Design and Implementation of Host-Based Intrusion Detection System". In: *2010 Third International Symposium on Intelligent Information Technology and Security Informatics*. Jian, China: IEEE, Apr. 2010, pp. 595–598. ISBN: 978-1-4244-6730-3 978-1-4244-6743-3. DOI: 10.1109/IITSI.2010.127. (Visited on 03/04/2023).

[Pei17] Sean Peisert. "Security in High-Performance Computing Environments". In: *Communications of the ACM* 60.9 (Aug. 2017), pp. 72–80. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/3096742. (Visited on 03/07/2023).

[Pro] The Zeek Project. *The Zeek Network Security Monitor*. https://zeek.org/. (Visited on 03/09/2023).

[Rob04] Paul Roberts. *Update: Hackers Breach Supercomputer Centers*. https://www.computerworld.com/article/2564261/update–hackers-breach-supercomputer-centers.html. Apr. 2004. (Visited on 03/06/2023).

[SAB18] Thomas Sterling, Matthew Anderson, and Maciej Brodowicz. *High Performance Computing: Modern Systems and Practices*. Cambridge, MA: Morgan Kaufmann, an imprint of Elsevier, 2018. ISBN: 978-0-12-420158-3.

[Sie21] Frank Siemons. *Open Source IDS: Snort or Suricata? [Updated 2021]*. https://resources.infosecinstitute.com/topic/open-source-ids-snort-suricata/. Jan. 2021. (Visited on 03/09/2023).

[Str] Erich Strohmaier. *Home - / TOP500*. https://www.top500.org/. (Visited on 03/06/2023).

[Tea] OSSEC Project Team. *OSSEC - World's Most Widely Used Host Intrusion Detection System - HIDS*. https://www.ossec.net/. (Visited on 03/09/2023).

[van21] Marc van Hauser Heuse. *Hydra*. Mar. 2021.

[Waz] Wazuh. *Rootkits Behavior Detection - Malware Detection · Wazuh Documentation*. https:// documentation.wazuh.com/current/user-manual/ capabilities/malware-detection/rootkits-behavior-detection.html. (Visited on 03/26/2023).

[WC22] Shie-Yuan Wang and Jen-Chieh Chang. "Design and Implementation of an Intrusion Detection System by Using Extended BPF in the Linux Kernel". In: *Journal of Network and Computer Applications* 198 (Feb. 2022), p. 103283. ISSN: 10848045. DOI: 10.1016/j.jnca.2021.103283. (Visited on 03/27/2023).