AECID: A Self-learning Anomaly Detection Approach Based on Light-weight Log Parser Models

Markus Wurzenberger[1], Florian Skopik[1], Giuseppe Settanni[1] and Roman Fiedler[1]
[1]*AIT Austrian Institute of Technology, Center for Digital Safety and Security, Vienna, Austria*
firstname.lastname@ait.ac.at

Keywords: Anomaly Detection; Intrusion Detection System, Machine Learning; Log Analysis

Abstract: In recent years, new forms of cyber attacks with an unprecedented sophistication level have emerged. Additionally, systems have grown to a size and complexity so that their mode of operation is barely understandable any more, especially for chronically understaffed security teams. The combination of ever increasing exploitation of zero day vulnerabilities, malware auto-generated from tool kits with varying signatures, and the still problematic lack of user awareness is alarming. As a consequence signature-based intrusion detection systems, which look for signatures of known malware or malicious behavior studied in labs, do not seem fit for future challenges. New, flexibly adaptable forms of intrusion detection systems (IDS), which require just minimal maintenance and human intervention, and rather learn themselves what is considered normal in an infrastructure, are a promising means to tackle today's serious security situation. This paper introduces ÆCID, a new anomaly-based IDS approach, that incorporates many features motivated by recent research results, including the automatic classification of events in a network, their correlation, evaluation, and interpretation up to a dynamically-configurable alerting system. Eventually, we foresee ÆCID to be a smart sensor for established SIEM solutions. Parts of ÆCID are open source and already included in Debian Linux and Ubuntu. This paper provides vital information on its basic design, deployment scenarios and application cases to support the research community as well as early adopters of the software package.

1 Introduction

In 1980, James Anderson was one of the first researchers who indicated the need of intrusion detection systems (IDS), contradicting the common assumption of software developers and administrators, that their computer systems were running in 'friendly', i.e. secure, environments (James P. Anderson, 1980). Recent reports (Verizon, 2016) (Symantec, 2016) show that while companies started to invest large amounts of resources into cyber security already years ago, many basic security issues unfortunately remain. On the one side, the increasing interconnection of previously isolated infrastructures, and the creation of systems of systems as well as the emergence of the Internet of Things (IoT) (Atzori et al., 2010) tremendously increases the attack surface. On the other side of the scale, the awareness in non-security company is still rather low – most successful attacks today still use phishing, skimming and baiting (Verizon, 2016) and exploit human negligence to obtain unauthorized access to a company's ICT network.

A wide variety of security solutions have been proposed in recent years to cope with this challenging situation. While some of them were able to relax the situation at least partially, research on intrusion detection systems seems – due to rapidly changing technologies and system design paradigms – to be a never-ending story. Signature-based approaches (Mitchell and Chen, 2014), i.e., black-listing methods, are still the de-facto standard applied today for some good reasons: they are essentially easy to configure, can be centrally managed, i.e., do not need much customization for specific networks, yield a robust and reliable detection for known attacks and provide low false positive rates. While these are all great benefits for the application in today's enterprise environments, there are, nevertheless, solid arguments to watch out for more sophisticated anomaly-based detection mechanisms (Liao et al., 2013), which should be applied additionally to black-listing approaches for the reasons explained as follows.

- The exploitation of new zero-day vulnerabilities are hardly detectable by blacklisting-approaches. Simply, there are no signatures to describe the indicators of an unknown exploit.

- Attackers can easily circumvent the detection of Malware, once indicators are widely distributed. Simply re-compiling a Malware with small modifications will change hash sums, names, IP addresses of command and control servers and the like – in the worst case, rendering all these data which is used to describe indicators useless.
- Eventually, many sophisticated attacks use social engineering as an initial intrusion vector. Here no technical vulnerabilities are exploited, hence, no indicators on a blacklist can appropriately describe malicious behavior.

Especially the latter requires smart anomaly detection approaches to reliably discover deviations from a desired system's behavior as a consequence of an unusual utilization through an illegitimate user. This is the usual case when an adversary manages to steal user credentials and is using these actually legitimate credentials to illegitimately access a system. However, an attacker will eventually utilize the system differently from the legitimate user, to reach his target, for instance running scans, searching shared directories and trying to extend his influence to surrounding systems at either unusual speed, at unusual times, taking unusual routes in the network, issuing actions with unusual frequency, causing unusual data transfers at unusual bandwidth. This causes a series of events within an infrastructure which are picked up by anomaly-based approaches and used to trigger off alerts.

Certainly, even black-listing approaches can cover some of these cases. For instance, log-in attempts outside business hours is a standard case, which every well-configured IDS can detect. The point here is, that with just using black-listing the security personnel must think of a wide variety of potential attack cases ahead and how they manifest in the network, i.e., which kind of events they would trigger. This is not only a cumbersome never-ending task, but also extremely error-prone. Eventually, there will always be ways to exploit a system of which the designers and operators did not think in advance. In contrast to that, the application of white-listing approaches seems intriguing here: one needs to describe the 'normal and desired system behavior' (this means to 'white list' what is known good) and everything that differs from this description is potentially classified as hostile.

The effort is comparatively lower, which demonstrates impressively the advantage of an anomaly-based approach. However, while this seems quite attractive, the advantages come with a price. Often, high false positive rates, complex behavior models, potentially error-prone learning phases and the like are just some of the drawbacks to consider. Deploying, configuring and effectively operating an anomaly detection system is a highly non-trivial task.

Substantial effort has been invested to apply anomaly detection on network traffic in real-time (Liao et al., 2013). With these methods, irregularities of data flows can be effectively spotted by investigating network packets and their meta data, such as flows between previously decoupled systems or changes in the interaction patterns and behavior. However, many argue that in the future a reliable detection of anomalies will only be feasible on the host. The wide adoption of numerous tunneling technologies, end-to-end encryption, virtualization/containerization, and software defined networks makes it hard – if not impossible – to track the real system behavior by inspecting network traffic only. More sophisticated, but also more complex is the detection of anomalies based on actual system events. These system events are often captured in system log files and are just waiting to be harnessed by more modern anomaly detection techniques.

In this paper, we have a close look on the applicability of anomaly detection approaches in IDS which specifically exploit semantically rich log data. In particular the contribution of this paper is threefold:

- we discuss the design principles of a modern scalable anomaly detection system to be applied in large-scale distributed systems;
- we outline the ÆCID approach, which is an actual implementation based on the aforementioned design principles;
- we demonstrate the successful application of the ÆCID approach specifically in non-enterprise IT environments, such as Industry 4.0 and cyber-physical systems.

The remainder of the paper is organized as follows. Section 2 outlines important background and related work. Section 3 discusses important concepts for modern anomaly detection approaches applied in IDS systems, specifically, the distribution of the parsing and un-supervised classification process over numerous nodes and continuous synchronization among them. In Section 4 the implementation of the most important features of the ÆCID system is described, followed by its mode of operation in Section 5. Then, Section 6 highlights typical application cases of ÆCID and demonstrates where such a system is superior over a standard signature-based solution. Finally, Section 7 concludes the paper.

2 Background and Related work

Like other security tools, IDS aim to achieve a higher level of security in ICT networks. Their primary goal is to timely and rapidly detect invaders, so that it is possible to react quickly and reduce the caused damage. In opposite to Intrusion Prevention Systems (IPS) that are able to actively defend a network against an attack, IDS are passive security mechanisms, which allow to detect attacks without enabling any countermeasure automatically (Whitman and Mattord, 2012), but informing administrators of the discovered intrusion through notification procedures.

(Sabahi and Movaghar, 2008; Liao et al., 2013; García-Teodoro et al., 2009) survey various methods applicable for intrusion detection. IDS can roughly be categorized as host-based IDS (HIDS), network-based IDS (NIDS) and hybrid or cross-layer IDS (Vacca, 2013). Similarly to simple security solutions such as anti-viruses, HIDS are installed on every system (host) of the network that needs to be monitored. While HIDS deliver specific high-level information about an attack and allow comprehensive monitoring of a single host, they can be potentially disabled by, for example, a Denial of Service (DoS) attack, because if a system is once compromised also the HIDS is. NIDS monitor and analyze the network traffic crossing the network internal to an organization. For monitoring a network with an NIDS, one single sensor-node can be sufficient and the functionality of this sensor is not effected if one system of the network is compromised. A major drawback of NIDS is that if the network is overloaded, a complete monitoring cannot be guaranteed, because some network packets may be dropped to avoid congestion. Cross-layer and hybrid IDS merge different methods. Hybrid IDS usually provide a management framework that combines HIDS and NIDS to reduce their drawbacks and benefit from their advantages. Cross-layer IDS aim to maximize the available information while minimizing the false alarm rate. To achieve this, various data sources, such as log data and network traffic data, are used for intrusion detection.

There exist generally three detection methods applied in IDS: signature-based detection (SD), stateful protocol analysis (SPA), and anomaly-based detection (AD) (Liao et al., 2013). SD and SPA can only detect previously known attack patterns using signatures and rules that describe malicious events and thus are also called black-listing approaches (Whitman and Mattord, 2012) (Scarfone and Mell, 2007). AD approaches are more flexible and able to detect novel and previously unknown attacks. They establish a baseline of normal system behavior and therefore are also called white-listing approaches (García-Teodoro et al., 2009).

The rapidly changing cyber threat landscape demand for flexible and self-adaptive IDS approaches. One solution are self-learning AD based approaches that automatically learn and continuously adapt the normal system behavior; this serves as ground truth to detect anomalies that expose attacks and especially invaders. Generally, there are three ways to realize self-learning AD (Goldstein and Uchida, 2016): *supervised*, *semi-supervised*, and *unsupervised*. Unsupervised methods do not require any labeled data and are able to learn distinguishing normal from malicious system behavior during the training phase. Semi-supervised methods are applied when the training set only contains anomaly-free data; they are also known as ‘one-class’ classification. Supervised methods require a fully labeled training set containing both normal and malicious data. These three methods do not necessitate active human intervention during the learning process. While unsupervised self-learning is entirely independent from human influence, for the other two methods the user has to ensure that the training data is anomaly free and correctly labeled. Consequently, the previously described methods can be categorized as unsupported self-learning approaches.

3 Motivation and Challenges

This section discusses the motivation behind the introduction of the novel anomaly detection approach presented in this paper, and summarizes the challenges this approach aims to address. This work builds on our prior research on the topic, published in ((Friedberg et al., 2015)), and enhances the approach therein presented by extending its features and improving its flexibility.

Existing anomaly detection approaches that analyze computer log data are mostly designed for forensic analysis. However, these techniques can complement on-line Intrusion Detection System (IDS) and timely detect sophisticated attacks, such as advanced persistent threats (APT), and eventually mitigate their impact (Friedberg et al., 2015). Furthermore, contrarily to available state-of-the-art IDS, which mostly process network traffic data, a paradigm shift towards textual log data should be considered. In fact, the current trend shows an increasing adoption of end-to-end encryption in network communications, to secure critical information in transit. Hence, only clear text meta information is available for the investigation, if only network traffic is inspected. Thus, alternative data sources need to be leveraged to effectively perform anomaly detection. To the best of our knowledge, no online IDS is available, which processes log data to perform anomaly detection.

Anomaly-based IDS normally implement white-listing mechanisms; they are also called *behavior based approaches* because they compare computer network's or system's activities against a model that characterizes the normal behavior. This reference is considered to be "anomaly-free" and is also known as *ground-truth*. Today's rapidly changing cyber threat landscape accounts for flexible and self-adaptive IDS approaches. Thus, to build a system behavior model semi-supervised self-learning methods can be applied, whose main strength is the capability to detect unknown attacks without requiring attack signatures.

Leveraging self-learning and white-listing methods it is possible to implement anomaly detection systems that are independent from semantical interpretation of the processed log data, and from the syntax of the data. Since the system behavior model can be built automatically, the only requirement is that the syntax of the log lines does not change over time, as this would be considered as a deviation from the normal system behavior and therefore marked as anomaly. This is feasible because log data, contrarily to free text, follows a predefined structure. Normally, a log message comprises static chunks, i.e., constant strings that occur often in the log sequence (e.g., prepositions, commands, protocol names), and variable chunks occurring less frequently (e.g., IP addresses, host names, TCP port numbers). Moreover, the order in which different parts occur in a log line is deterministic and does not vary.

Another benefit of self-learning and white-listing approaches utilized for anomaly detection is their intrinsic flexibility. They can be applied to process logs produced by legacy systems and by appliance with small market shares (like those largely employed in cyber physical systems (CPS)). Such systems often lack of vendor support and suffer from poor documentation. Usually neither security solution providers supply signatures for monitoring these systems, nor vendors make patches available to keep the systems up-to-date. Furthermore, modern networks, such as the Internet of Things (IoT), comprise many devices that are often not powerful enough to allocate the large amounts of resources required to run anomaly detection tools. Thus, light-weight anomaly detection mechanisms that can operate consuming a minimal amount of memory and CPU are necessary. One way to meet this requirement is to foresee a decentralized architecture. Different light-weight processing instances can be distributed across the infrastructure, while a central control instance, running on a powerful machine, executes resource intensive operations (such as machine-learning functions) and allows an administrator to control the distributed light-weighted instances.

3.1 Detectable anomalies

In order for an anomaly detection method to be considered effective, different types of anomalies have to be recognizable by using it, with a certain level of confidence. In the following we list the main categories of anomalies a modern anomaly detection tool should be capable to reveal.

The simplest type of anomaly is represented by *anomalous single events*. On the one hand, these can be so-called *outliers* representing rarely occurring events, which appear so seldom that are not part of the normal system behavior model. On the other hand, these anomalies can be violations of prohibited parameter values; for example, a client access executed through a user agent normally not utilized in a network, therefore not white-listed. In case of blacklisting approaches, user agents that are not allowed need to be added (one-by-one) to the blacklist, and hence imply a high risk of incompleteness.

Anomalous event parameters are point anomalies such as IP addresses, port numbers or software versions that are not white-listed and therefore are not part of the normal system behavior. This type of anomalies includes, for example, events that occur outside of business hours, or are triggered by accounts of employees who are on vacation.

Anomalous single event frequencies are events usually considered normal, which occur with an anomalous frequency. For example, in case of data theft, an anomalously high number of database accesses from a single client would be recorded in the log data, triggering an anomaly.

Anomalous event sequences are anomalies revealable by observing the dependency between related events. Such dependency can be formalized by defining correlation rules. A correlation rule describes a series of events that have to occur in an ordered sequence, within a given time-window, to be considered non anomalous. To detect more complex anomalous processes, which may involve different systems on a network, multiple log lines need to be examined. After a particular log line type (recording a conditioning event) is observed, another specific log line (recording the expected implied event) has to occur within a predefined time slot. Otherwise, an anomaly is raised. Additionally, such correlation rules should be definable so that once a given (conditioning) event occurred, the algorithm checks if in a predefined time window previous to such event, another specific (implied) event has occurred. Correlation rules allow, for example, the detection of access chain violations, e.g., in course of an SQL-injection.

4 The ÆCID Approach

This section introduces ÆCID, a novel anomaly detection approach we propose to address the challenges outlined in the previous section. We illustrate here the system architecture and we describe the two main components ÆCID consists of: the *AMiner* and *ÆCID Central*.

Figure 1 depicts the system architecture of ÆCID. ÆCID is designed to allow the deployment in highly distributed environments; in fact, due to its lightweight implementation, an AMiner instance can be installed, as vantage point, on any relevant node of the network; ÆCID Central is the component responsible of controlling and coordinating all the deployed AMiner instances.

The AMiner operates similarly to an HIDS sensor. It has to be installed on every host and network node that needs be monitored, or – if possible – on a centralized logging storage which collects the log data generated by the monitored nodes. Each AMiner instance interprets the log messages acquired from the node it is deployed on, following a specific model, called *parser model*, generated ad-hoc to represent the different events being logged on that particular node. A tailored rule set identifies the events that are considered legitimate on that system; an AMiner instance checks every parsed log line against this rule set and reports any mismatch. Furthermore, each AMiner instance comprises a report generator that produces a detailed record of parsed and unparsed lines, alerts and triggered alarms. The reports can be sent either via the AECID Central Interface to the AECID Central, or through additional interfaces (e.g., via e-mail) to system administrators. The parser model in combination with the rule set, characterize the normal system behavior, i.e., describe the type, structure, and content of log lines representing events allowed to occur on the monitored system. Every log message violating this behavioral model represents an anomaly.

While the AMiner performs lightweight operations such as parsing log messages and comparing them against a set of existing rules, ÆCID Central provides more advanced features, and therefore requires more computational resources than a single AMiner instance. One of the main functions executed by ÆCID Central is to learn the normal system behavior of every monitored system, and consequently configure the AMiner instance, running on that system, to detect any logged abnormal activity. To do this, ÆCID Central analyzes the logs received from each AMiner instance, generates a tailored parser model (*Parser Model Generator* function) and a specific set of rules (*Rule Generator* function), and sends them to the AMiner instance, which adopts them to examine future log messages. ÆCID Central provides the different AMiner instances with self-learned parser models and rule sets, and adapts them, when the network infrastructure and/or the user behavior change. Hence, ÆCID Central needs to control and configure all the deployed AMiner instances; these operations are performed through the *AMiner Interface*. Moreover, a *Control Interface* allows a system administrator to communicate with ÆCID Central, adjust its settings, and setup the deployed AMiner instances. Additionally ÆCID Central leverages a *Correlation Engine* that allows to analyze and associate events observed by different AMiner instances, with the purpose of white-listing events generated by complex processes involving diverse network nodes. ÆCID Central also provides a Web-based GUI, to explore and review statistics as well as information on triggered alerts and alarms. If the log data collected within a network infrastructure includes records of communication events between network devices (e.g.,

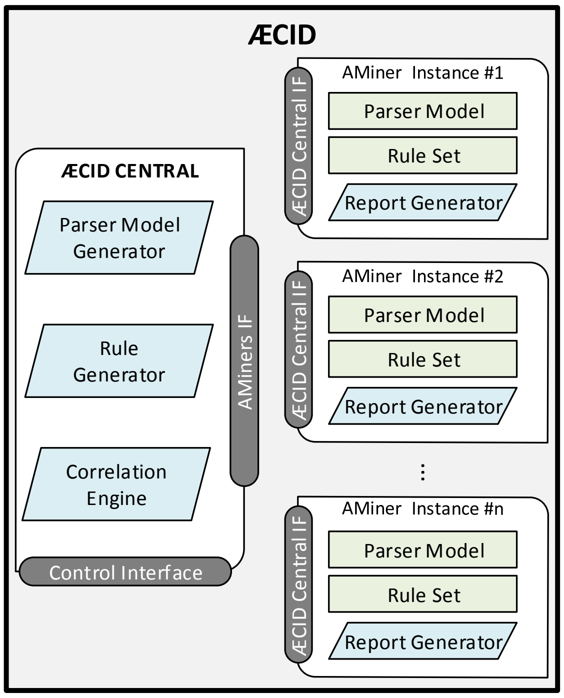

Figure 1: ÆCID architecture.

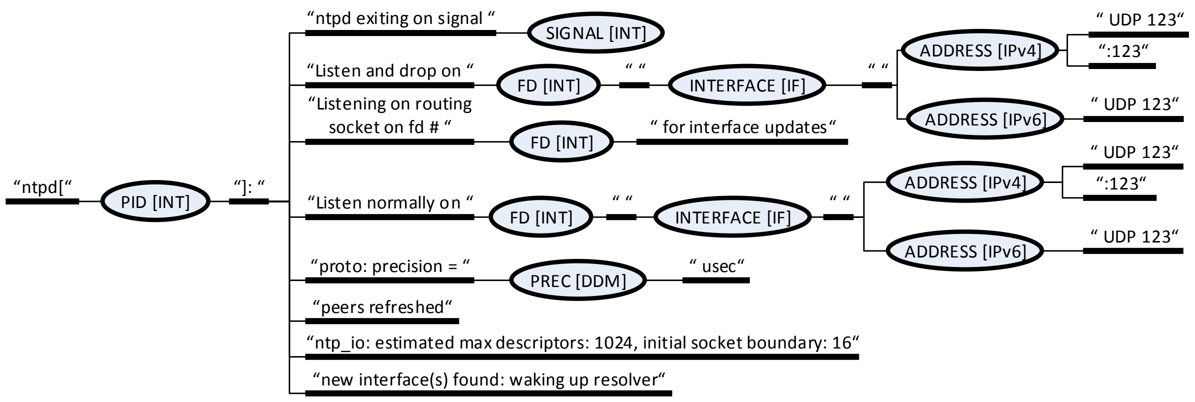

Figure 2: The graph describes the parser model for ntpd (Network Time Protocol) service logs. String under quotes are fixed elements or elements of a list. Oval entities symbolize model elements that allow variable values (an example of a parsed log-line is provided in Figure 3).

packets' headers observed via *tcpdump*), ÆCID can operate as NIDS, and therefore be utilized as hybrid IDS.

4.1 AMiner

The AMiner is the peripheral component of the ÆCID system, which executes lightweight operations on log data collected from the system it is deployed on. The current AMiner implementation – available open source[1] – is compatible with Python 2.6-2.7 and has been tested on CentoOS 6.7 and Ubuntu Xenial 16.04. In the lightest available configuration, AMiner requires only 32 MB of memory to perform simple log line filtering. If stricter memory requirements are in place, the AMiner can run on a separate (more powerful) machine, and listen to the log stream coming from the monitored device. This makes it possible to process logs produced by devices with low computational power, legacy systems, and devices whose hardware does not meet the AMiner's installation requirements.

The following sections illustrate in detail how the AMiner parses the captured log messages and how it identifies whether the logged event is to be considered legit or it indicates an anomaly.

4.1.1 Parser Model

The AMiner interprets every incoming log message according to a specific parser model, which characterizes the events observed on the device or network component it monitors. The parser model represents a path entropy model that efficiently describes the white-listed, i.e. permitted, log lines. It describes the log model of the monitored system as a graph, specifically an ordered tree (see Figure 2). The goal of using the parser model is to filter out as much redundant information as possible before a detailed analysis of the log line is performed. The parser model allows to efficiently extract all the information contained in a log line, while retaining only a minimum amount of data. Thus, every branch of the graph includes fixed segments, which represent constant strings always occurring in the same position of the log line, and variable segments, which represent strings that differ from line to line.

Log messages produced by different services have generally different syntaxes; for this reason, a parser model can only describe the set of possible log messages produced by one single service. This implies that an AMiner instance will adopt a distinct parser model for each different service running on the system that it monitors.

There exist several components, named `ModelElements`, a parser can consist of. The most frequently used are[2]:

- `AnyByteDataModelElement`: Match anything till the end of a log-atom.
- `DateTimeModelElement`: Simple datetime parsing using python datetime module.
- `DecimalIntegerValueModelElement`: Parsing integer values.
- `FirstMatchModelElement`: Branch the model taking the first branch matching the remaining log-atom data.
- `FixedDataModelElement`: Match a fixed (constant) string.

[1]https://git.launchpad.net/logdata-anomaly-miner/

[2]A more exhaustive list of model elements can be found in the AMiner documentation at: https://git.launchpad.net/logdata-anomaly-miner/tree/source/root/usr/share/doc/aminer/ParsingModel.txt

Listing 1: Example log data of an ntpd service.

```
0: Jun 14 16:17:12 ghive-ldap ntpd[16721]: Listen and drop on 0 v4wildcard 0.0.0.0 UDP 123
1: Jun 14 16:17:12 ghive-ldap ntpd[16721]: Listen and drop on 1 v6wildcard :: UDP 123
2: Jun 14 16:17:12 ghive-ldap ntpd[16721]: Listen normally on 2 lo 127.0.0.1 UDP 123
3: Jun 14 16:17:12 ghive-ldap ntpd[16721]: Listen normally on 3 eth0 134.74.77.21 UDP 123
4: Jun 14 16:17:12 ghive-ldap ntpd[16721]: Listen normally on 4 eth1 10.10.0.57 UDP 123
5: Jun 14 16:17:12 ghive-ldap ntpd[16721]: Listen normally on 5 eth1 fe80::5652:ff:fe5a:f89f UDP 123
6: Jun 14 16:17:12 ghive-ldap ntpd[16721]: Listen normally on 6 eth0 fe80::5652:ff:fe01:1aee UDP 123
7: Jun 14 16:17:12 ghive-ldap ntpd[16721]: Listen normally on 7 lo ::1 UDP 123
8: Jun 14 16:17:12 ghive-ldap ntpd[16721]: peers refreshed
9: Jun 14 16:17:12 ghive-ldap ntpd[16721]: Listening on routing socket on fd #24 for interface updates
```

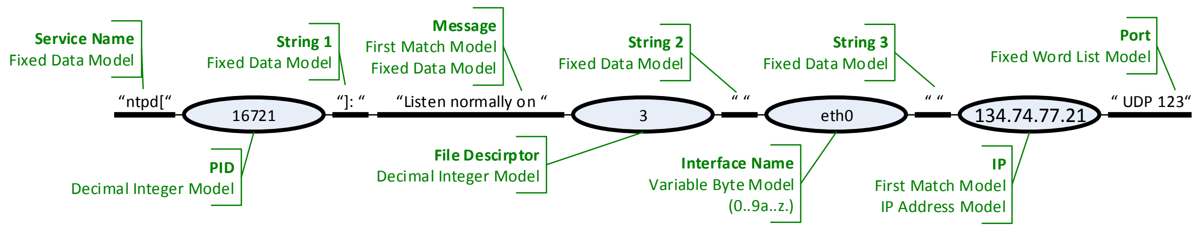

Figure 3: Example of log line parsing (cf. List 1 line number 3 and Figure 2).

- `IpAddressDataModelElement`: Match an IPv4 address.
- `SequenceModelElement`: Match all the sub-elements exactly in the given order.
- `FixedWordlistDataModelElement`: Match one of the fixed strings from a list.
- `VariableByteDataModelElement`: Match variable length data encoded within a given alphabet.

When the AMiner parses a log line, it usually applies a preamble model to parse the beginning of the line; for example, in case of a syslog message, the preamble corresponds to the timestamp and the host name. Afterwards, a `FirstMatchModelElement` decides which service the log line reports about, and applies the corresponding parsing model. For a better understanding of the parser, Figure 2 shows the graph of the parser model for the log lines of an ntpd service (see List 1). Figure 3 illustrates how the line number 3 in List 1 is parsed. The figure shows how the different elements of the log line are mapped to the nodes of the graph that represents the parser of the ntpd service. For example, at each fork of the graph a `FirstMatchModelElement` is applied to find the correct branch, before the next part of the line can be parsed. The oval entities mark variable segments of the log line; these have to follow specific rules, e.g., a PID parsed by the `DecimalIntegerValueModelElement` must consist only of digits between 0 and 9. Non-oval entities mark fixed segments of the parsed log line that must always occur for a log line to match the same branch of the parser model.

4.1.2 Detecting Anomalies

There are two ways for the AMiner to reveal anomalies within the log messages: i) by observing deviations of the log lines from the parser model, ii) by identifying log lines which do not follow certain predefined rules.

Thanks to the acquired knowledge on the normal system behavior, formalized through the corresponding parser model, the most advantageous way to reveal anomalies is to detect significant deviations of the logged events from the normal system behavior model. Usually, an information system operates only in a contained number of system states. Those states, which occur while the system runs normally, i.e. when the system is not in maintenance or recovery mode, define the model for the normal system behavior. Given that log data records occurring system events, a log message that does not match any available path in the parser model graph is to be considered anomalous, because it represents an unexpected system event. Thus, the AMiner considers a log line non-anomalous only if it matches one entire path of the parser model graph. Hence, every deviation from the graph's paths indicates an anomalous event. The anomaly can be caused either by a technical failure, by maintenance activities, or by an unused system function that has been activated by an attack. The AMiner white-lists all paths described by the graph defined by the parser model, and raises an alert every time a log line cannot be fully parsed.

Another way to detect anomalies is by defining white-listing rules, which are formulated in order to

Listing 2: Output of the parser shown in Figure 2 applied to log line 3 from List 1

```
Jun 14 16:17:12 ghive-ldap ntpd[16721]: Listen
normally on 3 eth0 134.74.77.21 UDP 123
/model/syslog/time: 'Jun 14 16:17:12'
/model/syslog/host: 'ghive-ldap'
/model/services/ntpd/sname: 'ntpd['
/model/services/ntpd/pid: '16721'
/model/services/ntpd/s1: ']: '
/model/services/ntpd/msg/text: 'Listen normally
on '
/model/services/ntpd/msg/descriptor: '3'
/model/services/ntpd/msg/s2: ' '
/model/services/ntpd/msg/if: 'eth0'
/model/services/ntpd/msg/s3: ' '
/model/services/ntpd/msg/ipv4/ip: '134.74.77.21'
/model/services/ntpd/msg/ipv4/port/: ' UDP 123'
```

allow only specific system events. The AMiner extracts all paths occurring in a log line and the associated parsed values as shown in List 2. White-listing rules can be defined by simply allowing only specific values for a certain log line element, or specific combination of different values. For example, in `/model/services/ntpd/msg/ipv4/ip`, it is possible to white-list only a certain list of IP addresses which are allowed to occur. If a non-white-listed IP address is detected, an alert or an alarm can be raised, and consequently an e-mail message can automatically be sent to system administrators to notify them of the anomaly. The same concept can be applied for a combination of values; for example to allow that specific user names only appear together with certain IP or MAC addresses. A rule can also be configured as a black-listing rule, to filter out known unwanted events, whose impact is considered dangerous. Finally, it is possible to formulate rules in a way to permit a range or a list of values.

Signature based IDS normally analyze log lines individually, however, malicious network behavior often manifests in an sequence of multiple log lines. Only by correlating such a sequence of events the anomaly can become apparent. For this reason, the AMiner can be configured to detect anomalies based on statistics. For example, a specific event which normally occurs 5 to 7 times per hour, will trigger an alert if it suddenly occurs 10 times in one hour. In this case, the AMiner will detect changes in the distribution of path occurrences. For example, assuming that the occurrence of a path follows a normal distribution over a predefined time interval, a fluctuation of the mean and of the standard deviation will indicate an anomaly. This demonstrates that the AMiner is able to detect not only anomalous single events, but also anomalous event frequencies.

Once an anomaly is detected, the AMiner can be configured to handle alerts and alarms. Reports can be sent via e-mail, periodically at a specified frequency, or every time a certain number of alerts is reached, or immediately as soon as an alarm is triggered to timely inform an administrator.

4.2 ÆCID Central

ÆCID Central is the intelligent component of ÆCID. It includes the parser generator, the rule generator, and a correlation engine, which allows to correlate events observed by different AMiner instances. The *AMiner Interface* enables the exchange of data between ÆCID Central and the different AMiner instances deployed in the network, while a *Control Interface* allows administrators to configure the whole system as well as to visualize analysis results.

4.2.1 Control Interface

The control interface allows system administrators to communicate with ÆCID Central and, through the AMiner Interface, to orchestrate the different AMiner instances deployed in the network. This means that the administrator can change the settings of every AMiner instance, including the parser model they adopt, the set of rules that they check, as well as the data sources that they parse (i.e., their input data). Furthermore, ÆCID Central is equipped with a graphical interface visualizing analytical statistics (including for example the number of occurrences of a certain type of log line), along with information on detected anomalies and triggered alerts and alarms.

4.2.2 Parser Model Generator

The parser model generation is one of the self-learning functions provided by ÆCID. Via the control interface the system administrator can activate the parser model generator separately for each AMiner instance controlled by ÆCID Central. This is done when a new AMiner instance is deployed, new hardware or software components are added to the network, or a high number of false alerts occur, caused by deviations from the current parser model.

When the parser model generator is activated the respective AMiner instance reports every unparsed line to ÆCID Central. The lines are then analyzed to derive a new parser model, or to adapt an existing one. First, the parser model generator collects a certain number of lines; afterwards, it applies either a word-based clustering, similar to SLCT (Vaarandi, 2003), or a character-based clustering, similar to (Wurzenberger et al., 2017), to group the log lines and detect

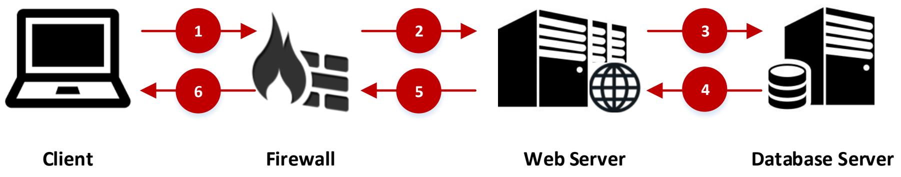

Figure 4: The figure shows the log-in process to an online shop: (1) the client tries to log into an online shop on a web server, (2) a connection through the firewall occurs, (3) the Web server checks credentials through a database query, (4) the database query returns some result, (5) a response through the firewall: access acceptance or denial, (6) client receives the response.

static and variable parts of the log lines. Based on the obtained alignments that describe the derived log line clusters, and considering the information about the log line segments occurring as variable parts, the graph describing the log line model can be built. Once the parser reaches a certain stability level, i.e. when the process is repeated for a new set of obtained log lines, and the parser does not need to be adapted anymore, ÆCID Central terminates the parser model generation process. Finally, the generated parser is forwarded to the respective AMiner instance, and the AMiner starts filtering out unknown log lines and generating alerts.

The parser model generator is also invoked to adapt an existing log line model. New paths can be added to the graph describing a parser model, lists such as `FixedWordlistDataModelElements` can be adapted, or alphabets such as `VariableByteDataModelElements` can be updated.

4.2.3 Rule Generator

Another self-learning function provided by ÆCID is the rule generation. Similarly to the parser model generator, the rule generator can be activated for a specific AMiner instance via the control interface. Once enabled, the selected AMiner instance will forward the parsed lines to the rule generator running on ÆCID Central. Based on the parsed values the rule generator creates candidates for rules. It defines lists or intervals of values that are allowed to occur in a specific path, or it defines rules enforcing that values of different paths are only allowed to occur in specific combinations, e.g., specific usernames are only allowed to occur in combination with specific IP or MAC addresses.

Once the rule generator defines a rule candidate, the candidate has to be verified. One way to accomplish this is to run a binomial test as shown in (Friedberg et al., 2015), to evaluate if a tested rule candidate is stable or not. Once a rule candidate is verified and therefore considered stable, the rule generator pushes it to the AMiner, which includes it into its rule set.

4.2.4 Correlation Engine

The correlation engine implemented in ÆCID Central allows to detect network-wide anomalies by correlating events observed by different AMiner instances. This makes it possible to detect deviations within complex processes that involve different services, and therefore produce log messages on components monitored by different AMiner instances. Consider the example depicted in Fig. 4; it shows the normal access chain of the log-in procedure to a Web-shop. If a user logs into the Web-shop, certain log lines will be produced in a specific order by the firewall, the Web-server, the database server and the Web-server again, including specific values for the paths of the parser. ÆCID is able to identify such event sequences, and to generate corresponding models. This allows ÆCID to automatically derive relevant correlation rules and verify through them if the system behavior is aligned with the generated model. All AMiner instances monitoring the services involved in the process will, in fact, forward to ÆCID Central the log events for which the correlation rule is being evaluated (even if they are not individually considered anomalous). ÆCID Central will then analyze the sequence of events collected from the different AMiner instances and verify their alignment to the model. If the illustrative access chain mentioned above is violated, because the database is accessed without a previous access to the Web-server (this could be due to an SQL-Injection), ÆCID Central will identify an inconsistency and therefore trigger an alarm. This is an example that demonstrate how ÆCID does not only detect anomalies that manifest in deviations of the normal system behavior of a single device, but also complex anomalies that can only be detected when analyzing events occurring in diverse nodes of the network. It is important to notice that the correlation rules can also be applied to a single AMiner instance to analyze complex processes running on one single node.

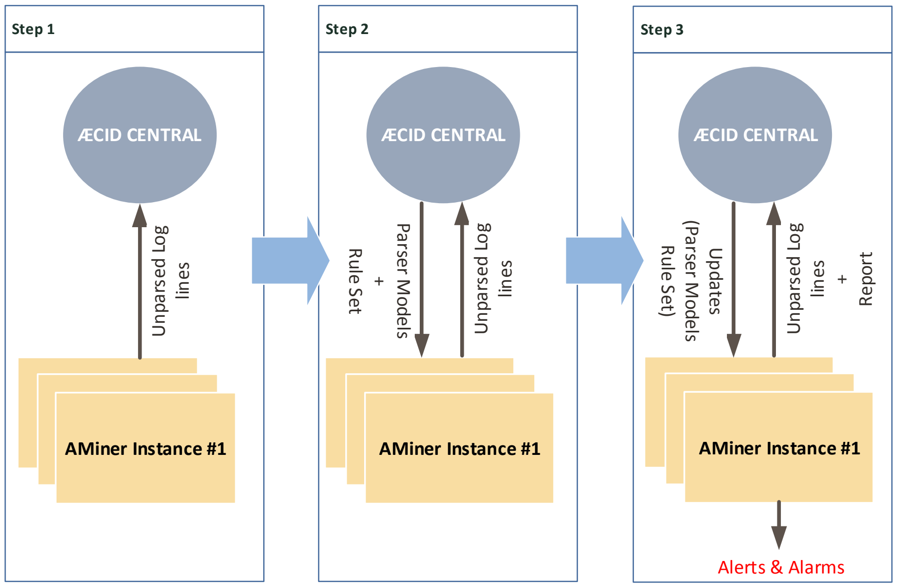

Figure 5: ÆCID Initialization Process.

5 System Deployment and Operation

This section illustrates the different topologies in which ÆCID can be deployed within a network, and presents the phases of its operational work-flow.

The simplest way to deploy ÆCID is by employing only one AMiner instance installed on a single node. This setup allows to exclusively monitor events produced and logged by that single component. Alternatively, if a log collection mechanism is in place on the node, which acquires log messages from other remote nodes in the network (à la Syslog server), the AMiner can work in a centralized fashion, and analyze events occurring on distributed systems. The drawback of this configuration is that the parser models, as well as the set of rules utilized by the AMiner for the detection of anomalies, are statically defined and need to be manually configured. The lack of an intelligent component (ÆCID central) implies, in fact, that the administrators have to: i) define the parser model describing the structure of the events being logged by the monitored node, and ii) have a thorough understanding of every event (occurring on every monitored node) that is to be consider legitimate, and instantiate a corresponding set of withelisting rules. This solution is applicable in case of small-scale systems, whose computational power is not sufficient to run ÆCID central, which perform highly recurrent operations, and are therefore simple to characterize manually.

If the infrastructure to be monitored comprises a large number of distributed nodes, with little resources and small computational power at their disposal, ÆCID can be deployed following a star topology. In this setup, an AMiner instance is installed on every distributed node, while a more powerful node hosts ÆCID Central. Every AMiner is connected to ÆCID Central and exchanges with it information regarding parsed lines and discovered anomalies.

Figure 5 illustrates the three stages required to initialize the ÆCID system when deployed in this topology. When the AMiner instances are installed for the first time on the nodes, they are not able to parse any of the log lines generated on the node, because no parser model is defined yet; thus, they forward every unparsed line to ÆCID Central, which learns the structure of the log messages received from each node and automatically builds a dedicated parser model for each service generating logs on each node. Along with the parser models, ÆCID Central builds the behavioral model of the services running on every connected node, and generates a corresponding set of white-listing rules per node, which describe the model, i.e., the normal behavior.

In the second step, ÆCID Central forwards the derived parser models and rule-sets to the respective AMiner instances; now the AMiner instances can follow the received parser models, analyze the incoming log messages and identify any suspicious event by checking the adherence to the obtained rules.

The third step represents the fully operational system; the AMiner instances continue sending any unparsed log line to ÆCID Central for further inspection, and receive from ÆCID Central updates on the enabled parser models and set of rules. If correlation rules are defined, ÆCID Central (through its correlation engine) analyzes the relevant log messages interacting with the involved AMiner instances, as described in the previous section.

Whenever an anomaly is revealed, the AMiner instance generates a notification report and, if necessary, sends it by email to a pre-configured list of recipients. Alerts and alarms are also reported to ÆCID Central, where they can be aggregated and visualized.

Finally, in case sufficient resources and computational power are available on a single node, ÆCID can be deployed in its full-fledged setup on a standalone machine. In this scenario, ÆCID Central and the AMiner instance operate on the same node. The advantage of this topology, compared to the first one, lies in the fact that the administrators do not need to manually determine the parser models nor to define the set of rules, because ÆCID Central automatically generates them following the three-step approach described above and depicted in Figure 5.

6 Application Scenarios

The multi-layer light-weight detection approach presented in this paper introduces a number of benefits that make its adoption attractive for a series of application scenarios. This section explores some of the most promising use cases, in which the employment of ÆCID, stand-alone or in conjunction with other security solutions, would be highly advantageous.

As extensively illustrated in the previous sections, ÆCID can be adopted to analyze events logged by systems running on different layers of the OSI model. If applied on network traffic information, ÆCID can identify and keep track of the communications established between different systems in the network, and perform a *network interaction graph analysis*. Observing which network nodes interact with each-other at which frequency, would allow ÆCID to build a "communication-behavior" model, and to promptly identify any divergence from such a model, which could indicate internal or external malicious attempts to access network systems.

Similarly, ÆCID could be employed to analyze events recorded on the application layer, particularly when users authenticate on the numerous services deployed in an enterprise network. *Authentication interaction graph analysis* can be performed using ÆCID, to monitor which users authenticate on which services with which frequency. The "authentication-behavior" model established by ÆCID in this scenario would allow to reveal any unusual authentication attempt, pinpointing potential intrusions, illegitimate access to critical resources, or erroneous authentication.

Moreover, ÆCID could serve as additional security layer, besides signature-based and other blacklisting solutions, such as firewalls. In this setup ÆCID would improve the overall detection capability by allowing the identification of previously unknown threats, and the verification of suspicious triggered alerts. The false positive rate (FPR) as well as the number of false negatives (FN) would decrease effectively and a higher level of security would be achieved. The alarms triggered by ÆCID could then be fed into a Security Information and Event Management (SIEM) system, which would correlate them with the events generated by other sensors.

A further promising application area is CPS. CPS of the future will be the backbone of *Industry 4.0*, and will operate following a self-adaptation paradigm, which foresees that the components of a system are capable of configuring, protecting and healing themselves when certain internal and/or external conditions demand to (Musil et al., 2017). The process of self-adaptation follows four principal phases: *monitor*, *analysis*, *plan*, *execute*. Critical events, observed in the systems (in the monitor phase) and opportunely examined (in the analysis phase), would trigger specific changes in the system configuration (through the execute phase), which would follow suitable adaptation policies (evaluated in the plan phase). In the context of ICT security this approach could allow CPS being targeted by security threats to timely identify the indicators of an attack and swiftly react to contain its effects, reducing its impact. In this scenario, employing ÆCID in the monitoring and analysis phase, would be an asset. Thanks to their light-weight nature, AMiner instances could be installed on numerous low-power components deployed across the CPS, which would not be able to run any traditional IDS. By recording system events they would allow to have an accurate and comprehensive overview of the security situation of the entire CPS in real-time. The anomalies identified by ÆCID Central would then be evaluated and, in line with predefined security policies, trigger configuration changes in the monitored CPS, that would contain the effect of the detected threat.

Finally, the system behavior model built by ÆCID Central, makes it possible for ÆCID to identify critical security events previously unseen, which may potentially indicate the occurrence of a *zero-day* attack. Integrating ÆCID with a threat intelligence (TI) management system would allow security operation centers to greatly improve their incident handling capability. Indicators of compromise (IoC), obtained by inspecting the anomalies revealed by ÆCID, could in fact be combined and correlated with the intelligence gathered from multiple data sources by TI management solutions (such as the tool proposed in (Settanni et al., 2017)). This correlation is fundamental to interpret the IoCs, confirm the occurrence of an attack, and identify possible mitigation strategies. Additionally, this integrated framework would allow to dynamically reconfigure any deployed monitoring system in order to center their focus towards those critical assets vulnerable to the discovered threat. Eventually, this solution would also support the definition of attack signatures, which would be used to update any black-listing security solutions deployed in the infrastructure, and guarantee a higher level of protection.

7 Conclusion and Outlook

New light-weight forms of IDS, capable of automatically comprehending and dynamically adapt to the behavior of large-scale distributed systems, are a promising means to tackle today's advanced security threats. In this paper, we discussed the design principles a modern anomaly detection system should follow to be effective. Furthermore, we introduced the ÆCID approach, a light-weight detection mechanism based on self-learning log parser models. We presented ÆCID's system architecture, and we outlined the advantages this approach provides in terms of detection effectiveness, deployment flexibility, and applicability. Several use case scenarios demonstrate how ÆCID would greatly improve the security posture of modern organization, allowing them to achieve a higher degree of protection against advanced cyber threats.

Parts of ÆCID are open source and are already included in Debian Linux and Ubuntu; future work includes the implementation of the system in operational controlled environment, its validation in different complex application scenarios, and the evaluation of its performance on large-scale distributed systems.

ACKNOWLEDGEMENTS

This work was partly funded by the FFG project synERGY (855457), European research project SemI40 (ECSEL joint undertaking nr. 692466) and carried out in course of a PhD thesis at the Vienna University of Technology funded by the FFG project BAESE (852301).

REFERENCES

Atzori, L., Iera, A., and Morabito, G. (2010). The internet of things: A survey. *Computer networks*, 54(15).

Friedberg, I., Skopik, F., Settanni, G., and Fiedler, R. (2015). Combating advanced persistent threats: From network event correlation to incident detection. *Computers & Security*, 48:35–57.

García-Teodoro, P., Díaz-Verdejo, J., Maciá-Fernández, G., and Vázquez, E. (2009). Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security*, 28(1-2):18–28.

Goldstein, M. and Uchida, S. (2016). A Comparative Evaluation of Unsupervised Anomaly Detection Algorithms for Multivariate Data. *PloS one*, 11(4):e0152173.

James P. Anderson (1980). Computer security threat monitoring and surveillance. Technical Report 17, James P. Anderson Company, Fort Washington, Pennsylvania.

Liao, H.-J., Richard Lin, C.-H., Lin, Y.-C., and Tung, K.-Y. (2013). Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1):16–24.

Mitchell, R. and Chen, I.-R. (2014). A survey of intrusion detection techniques for cyber-physical systems. *ACM Computing Surveys (CSUR)*, 46(4):55.

Musil, A., Musil, J., Weyns, D., Bures, T., Muccini, H., and Sharaf, M. (2017). Patterns for self-adaptation in cyber-physical systems. In *Multi-Disciplinary Engineering for Cyber-Physical Production Systems*, pages 331–368. Springer.

Sabahi, F. and Movaghar, A. (2008). Intrusion Detection: A Survey. pages 23–26. IEEE.

Scarfone, K. and Mell, P. (2007). Guide to intrusion detection and prevention systems (idps). *NIST special publication*, 800(2007):94.

Settanni, G., Shovgenya, Y., Skopik, F., Graf, R., Wurzenberger, M., and Fiedler, R. (2017). Acquiring cyber threat intelligence through security information correlation. In *Cybernetics (CYBCONF), 2017 3rd IEEE International Conference on*, pages 1–7. IEEE.

Symantec (2016). ISTR Internet Security Threat Report. Technical Report 21, Symantec Corporation.

Vaarandi, R. (2003). A data clustering algorithm for mining patterns from event logs. In *IP Operations & Management, 2003.(IPOM 2003). 3rd IEEE Workshop on*, pages 119–126. IEEE.

Vacca, J. R. (2013). *Managing information security*. Elsevier.

Verizon (2016). 2016 Data Breach Investigations Report. Technical report, Verizon.

Whitman, M. E. and Mattord, H. J. (2012). *Principles of information security*. Course Technology, Cengage Learning, Stamford, Conn., 4. ed., international ed edition. OCLC: 930764051.

Wurzenberger, M., Skopik, F., Landauer, M., Greitbauer, P., Fiedler, R., and Kastner, W. (2017). Incremental clustering for semi-supervised anomaly detection applied on log data. In *Proceedings of the 12th International Conference on Availability, Reliability and Security*, page 31. ACM.