2019 IEEE International Conference on Big Data (Big Data)

A Framework for Cyber Threat Intelligence Extraction from Raw Log Data

Max Landauer, Florian Skopik, Markus Wurzenberger, Wolfgang Hotwagner
Dept. for Digital Safety and Security
Austrian Institute of Technology
Vienna, Austria
firstname.lastname@ait.ac.at

Andreas Rauber
Inst. of Information Systems Engineering
Vienna University of Technology
Vienna, Austria
rauber@ifs.tuwien.ac.at

***Abstract*—Intrusion Detection Systems (IDS) rely on the availability and correctness of Indicators of Compromise (IoC), i.e., artifacts such as IP addresses that are known to correspond to malicious system activities. However, the simple nature and limited validity of these indicators impairs protection against cyber threats. Tactics, Techniques and Procedures (TTP) provide abstract information on attacker behavior, but are only available in human-readable format that prevents automatic detection using IDSs. In this paper we therefore propose an approach that extracts cyber threat intelligence from raw log data and combines the advantages of IoCs and TTPs by producing detectable patterns of complex system behavior. Other than existing approaches, our approach employs log data anomaly detection to disclose suspicious log events, which are used for iterative clustering, pattern recognition, and refinement. Our evaluations show that automatically extracted threat intelligence corresponding to a multi-step attack is suitable for detection of the same attack on another system.**

***Keywords*-threat intelligence; log data; anomaly detection; pattern recognition;**

I. Introduction

The widespread availability of attack tool kits make it easy for adversaries to target all kinds of digital systems. Cyber security counteracts these threats by employing Intrusion Detection Systems (IDS) that continuously monitor systems for so-called Indicators of Compromise (IoC), i.e., artifacts that signify that security has been breached. These indicators are usually distributed by Cyber Threat Intelligence (CTI) vendors that manually analyze and reverse-engineer attacks.

Unfortunately, IoCs are often criticized for not providing sufficient protection against advanced threats due to their simplicity and limited validity [1], [2], [3]. Generally valid information on attacker Tactics, Techniques and Procedures (TTP) that describes threats in a more abstract way is available, but typically only exists in human-readable format. This impedes automated monitoring by IDS [4], [2].

Thus, there is a need for actionable CTI of detectable behavior patterns consumable by IDS. However, as pointed out by Navarro et al. [5] in their recent survey, existing methods rely on manual generation of this kind of CTI, which is slow, time-consuming, requires expert knowledge, and is prone to human errors. In addition, manually defined CTI does not protect against unknown attacks.

We propose to automatically generate CTI from raw log events to address these issues. We thereby employ anomaly detection techniques for the disclosure of suspicious system activities and use methods from pattern recognition to combine anomalies and form complex scenarios. This has several advantages over existing methods: (i) It is difficult for adversaries to prevent that attacks manifest themselves in low-level log events that describe the system behavior in detail. (ii) Anomaly detection is an unsupervised approach that enables the detection of unknown attacks. (iii) Leveraging complex attack patterns enables detection of attacks that cannot be described by simple IoCs, such as event correlations, parameter values, or context data. (iv) Other than human-readable descriptions, these attack patterns allow automatic consumption and detection by IDSs.

Several challenges must be overcome by such an approach. While log data is semantically rich, analysis is difficult due to its unstructured nature. In addition, attacks carried out on separate systems may manifest themselves differently in log data, making automatic comparisons non-trivial. Another issue is that anomaly detection is prone to high false alarm rates, which impairs the confidence in the generated CTI. Finally, with large numbers of events occurring on systems, linking related anomalies is difficult.

In this paper we tackle these challenges by introducing a novel log processing pipeline. The pipeline makes use of a parser tree to identify events and dissect relevant values from a continuous stream of log data. Existing anomaly detection algorithms are then used to disclose artifacts that are possibly related to malicious activities. The system iteratively applies clustering, pattern recognition, and data enrichment to transform the anomalies into abstract attack patterns, while at the same time reducing the influence of false positives. Sharing the extracted threat intelligence allows other organizations affected by similar threats to detect attacks in an early stage.

We summarize our contributions as follows:

- A review of existing definitions and shortcomings of available cyber threat intelligence,
- a novel model for the automatic extraction of actionable cyber threat intelligence from raw log data, and
- an evaluation thereof within an illustrative scenario.

978-1-7281-0858-2/19/$31.00 © 2019 IEEE

978-1-7281-0858-2/19/$31.00 ©2019 IEEE

The remainder of this paper is structured as follows. Section II reviews and criticizes existing threat intelligence concepts. Section III outlines a model for the generation of actionable threat intelligence from raw log data. Technical implementation and algorithmic details of this model are stated in Sect. IV. The evaluation results of the proposed approach are given in Sect. V and discussed in Sect. VI. An overview of related works is given in Sect. VII. Finally, Sect. VIII concludes the paper.

II. Background

With an increasing number of cyber incidents occurring every year and organizations investing large amounts of money for protection, cyber threat intelligence has become one of the most relevant topics in business and science. However, many associated concepts and notions are used interchangeably. We thus devote this section to review existing viewpoints and clarify our understanding of the terms.

A. Cyber Threat Intelligence

The term Cyber Threat Intelligence (CTI) is used highly ambiguously throughout all kinds of literature. In particular, it is unclear at what point any available information on cyber threats is regarded intelligent rather than just data.

Chismon and Ruks [3] define CTI through the process of detecting and subsequently analyzing previously unknown threats with the aim of understanding and mitigating risks. Zhu et al. [6] state that unlike automatically collected data, generating CTI encompasses manual threat analysis and reasoning by domain experts. McMillan [7] provides a definition that involves evidence-based knowledge and context information on mechanisms, indicators, implications, and actionable advice about existing or emerging threats. Dalziel et al. [8] state that CTI must be refined, analyzed, and processed in order to be relevant, actionable, and valuable.

The consensus of these definitions is that security-related data needs to undergo a process of advanced analysis and enrichment to provide usable insights into cyber threats and be regarded as actionable CTI. The term actionable is thereby used just as ambiguously as the term threat intelligence itself. Dalziel et al. [8] denote CTI as actionable if it is specific enough to enable decision-making and response to present threats. Tounsi et al. [1] point out that outdated CTI loses its actionability, but mention that fast sharing of CTI is not sufficient to prevent targeted attacks. They also discuss the relevance of standardized CTI formats to ensure data quality and enable automated analysis. Popular CTI formats are STIX [9], IODEF[1], OpenIOC[2], and CAPEC [10].

We conclude that actionability means that no additional analyses are necessary to utilize available CTI; however, the specific requirements on actionable CTI depend on the desired use-case, such as detection, analysis, or containment.

[1]http://xml.coverpages.org/iodef

[2]https://www.fireeye.com/blog/threat-research/2013/10/openioc-basics

Chismon and Ruks [3] separate CTI into four subtypes: (i) technical, i.e., low-level Indicators of Compromise (IoC) with limited validity, (ii) tactical, i.e., low-level information on Tactics, Techniques and Procedures (TTP) with longer validity, (iii) operational, i.e., high-level details on imminent threats, and (iv) strategic threat intelligence, i.e., high-level reports on organizational risks. Due to their relevance to this paper, we discuss IoCs and TTPs in the following sections.

B. Indicators of Compromise

IoCs are typically described as artifacts which presences provide concrete evidence that system security was breached with high confidence. The well-known STIX [9] format defines indicators as "patterns that allow detection of suspicious or malicious cyber activity". Patterns thereby include IP addresses, email addresses, domain names, and computed indicators such as hash values. They are highly specific and thus IDSs usually report only few false alarms.

The actionability of IoCs is debatable and depends on the use-case at hand. On the one side, Tounsi et al. [1] state that IoCs are immediately actionable, because they can be automatically fed into Intrusion Detection Systems (IDS) once they become available. On the other hand, detection based on predefined IoCs is more reactive than proactive, i.e., detecting an IoC usually means that the system is already compromised. Moreover, Rhoades [2] argues that IoCs are too simple to identify complex malicious system activities. Tounsi et al. [1] even state that a key failing of CTI is that it is relatively simple for adversaries to ensure that attacks generate no artifacts that match pre-existing IoCs.

Another important aspect pointed out by Chismon and Ruks [3] is that IoCs from different CTI feeds yield small overlaps. Their explanation is twofold: First, it is easy to vary attack parameters such as IPs. Second, existing CTI is not of any intelligent value. Either way, these observations discredit the actionability of IoCs. Finally, one more problem with IoCs is that they are usually consumed by IDS without their context of occurrence, i.e., an IoC is either observed in the data or not [2]. To alleviate these issues, a more abstract way of describing threats is required. In the following section, TTPs are reviewed as a method to provide such information.

C. Tactics, Techniques and Procedures

While IoCs are detective in nature, Tactics, Techniques and Procedures (TTP) provide abstract and descriptive characterizations of threats, typically in human-readable form [9]. The main purpose of these descriptions is to detail the modus operandi, i.e., actions that attackers carry out on affected systems, and how these actions are carried out, e.g., exploits of certain vulnerabilities.

Among the advantages of TTPs over IoCs is that they are valid for longer time spans and that their abstract descriptions increase the difficulty of evasion by attackers. The reason for both effects is that it is relatively difficult for

adversaries to discover completely new ways of executing attacks in comparison to the low efforts of changing artifacts such as IP addresses [1]. This is also represented in the so-called "Pyramid of Pain" [11] that places TTPs as the most valuable type of CTI on top and IoCs at the bottom.

The main problem is that it usually takes extensive manual work and domain knowledge to generate TTPs on an adequate level of abstraction. Furthermore, the currently wide-spread human-readable descriptions of TTPs impede their usage for automatic detection [4], [2].

For example, consider the entry "Embedding Scripts within Scripts" (CAPEC-19) in the CAPEC database[3]. The attack is detailed on the "Standard" abstraction level and contains an extensive description of the typical attack execution flow. However, based on the available texts, it is not possible to manually or automatically extract indicators that support attack detection for particular systems.

Enriching existing manually defined TTPs with measurable indicators that support automatic detection of attacks or attack steps could improve this situation. Our research efforts are therefore directed towards bringing IoCs and TTPs closer together by combining and mapping IoCs to TTPs to yield more intelligent indicators.

III. CTI Generation

We address the outlined shortcomings of CTI concepts regarding their applicability by proposing a method for automatic or semi-automatic extraction of CTI from raw log data. In this section, we provide the problem statement, characterize log data as an appropriate data source, and introduce a model for CTI generation by iterative enhancement.

A. Problem Statement

Generating CTI is a time- and labor-consuming task that requires manually gathering, reviewing, consolidating, and integrating data from various sources into an abstract, understandable and shareable format. It is state-of-the-art that the resulting CTI involves simple IoCs that enable automatic detection and more complex TTPs for human consumption. Unfortunately, the actionability of both IoCs and TTPs suffers from the following shortcomings:

- **IoCs suffer from limited validity and reliability.** The patterns are too simple and easy to evade for attackers. Thus, IoCs primarily provide temporary protection.
- **TTPs fail to provide detectable patterns.** Their descriptions are too abstract and lack measurable indicators for automatic detection. Therefore, TTPs mostly support attack analysis in hindsight rather than providing proactive protection mechanisms.

Currently, there is no proper way to combine the benefits of IoCs and TTPs, i.e., provide permanently valid TTPs that offer measurable and therefore detectable indicators. It is non-trivial to represent TTPs using complex patterns of indicators, because attacks often occur as variations and are executed differently depending on technical environments.

These issues are tackled by our approach, which automatically generates detectable TTPs. Thereby, the selection of data sources is crucial. In the following section, we discuss the characteristics of log data to show its usefulness regarding the extraction of indicators.

B. Data Sources

IoCs in CTI feeds usually contain single tokens such as IP addresses or file names that are suitable for threat detection in diverse data sources. Other than most existing tools that focus on network traffic, we encourage to carry out analyses on system log data for a number of reasons: First, log data is semantically more expressive, because it contains human-readable phrases and parameters that were consciously placed to describe the system state. Second, log data not only includes what is communicated with other machines, but provides insights into the processes running on the machine. Third, encrypted network traffic makes inspection of actual contents difficult, while system log data is usually accessible in raw format.

In addition to event information such as accessed file paths and executed commands, logs also frequently contain information on the execution of these events, for example, IDs of associated users. Moreover, context information can be derived from the logs. Due to the fact that many processes are timed or periodically recurring, event time is arguably the most relevant context data available. Other information includes the source of the log event, log sensor data such as ID, location, or manufacturer, as well as system data [12].

Unfortunately, automated log analysis is non-trivial. In particular, the syntax of human-readable natural language present in system logs is difficult to parse, especially when the log format is not specified or known. In addition, log formats are subject to change over time and thus parsers need to be adapted regularly. Despite these challenges, our focus in this work lies on system log data. Our proposed concepts however can be applied with any type of log data.

C. Anomaly Detection

Traditional IDS use IoCs for blacklisting, i.e., each detected IoC indicates a security breach. Anomaly detection on the other hand is a whitelisting approach, i.e., potential threats are indicated by deviations from a self-learned model of normal behavior. A key design principle of our approach is that indicators for blacklisting are effectively mined through anomaly detection. For example, consider a sophisticated anomaly detection algorithm that recognizes the appearance of an abnormal IP, hash, url, process, or file as a consequence of malicious activities. It is possible to use the artifact as an indicator that enables detection of the same adversarial activity in the future or on another system.

[3]https://capec.mitre.org/

Anomaly detection offers a number of advantages. First, the discovery of previously unknown attacks is possible without any need to possess prior information on the threat actor, the system at hand, or the attack itself. Second, timely responses to threats are enabled, because anomaly detection is a mostly automatic process that supports online monitoring. Third, the effort of manually gathering indicators decreases. Rather than analyzing the logs by hand, cyber security experts are presented with anomalies almost immediately after they occur, making forensic investigations easier and faster. Anomaly detection is further able to identify subtle divergences and side-effects accompanied by attacks that humans may easily oversee, but are nonetheless valid candidates for indicators, such as missing events.

Unfortunately, anomaly detection also faces drawbacks. Foremost, false positive rates are typically higher in comparison to blacklisting approaches. The reason for this is that system behavior is often affected by fluctuations that are not caused by attacks, but rather by random events, erratic user behavior, arbitrarily scheduled tasks, or changes in the system landscape. For example, an unknown IP occurring in the logs more likely belongs to a benign new machine added to the network, rather than an adversary. There is usually no way around manually verifying the anomalies and adjusting the normal system behavior model if necessary. In addition, anomaly detection systems suffer from poisoning, i.e., detection capabilities are impaired when models are trained on systems already affected by attacks, and attackers may thus attempt to inject artifacts that neutralize detectors.

Clearly, the risk of deriving inaccurate IoCs from false positives seems unpromising and single anomalies cannot be considered TTPs. In the following, we therefore propose a model to transform anomalies into superior patterns, while at the same time reduce the influence of false positives.

D. Concept for CTI Generation

Our model for log-based CTI extraction employs anomaly detection as an initial step and involves a sequence of activities that progressively add context information to the data and enhance it with respect to the abstraction level. Figure 1 displays the overall procedure, including a schematic example. Note that the use of anomaly detection is a major difference to related approaches based on the CRIM architecture [13], which obtain alerts from rule-based detectors. Our approach is thus in accordance with the recommendations by Navarro et al. [5], who see a need to develop aggregation and detection algorithms based on raw events rather than well-formatted IDS alerts. In the following, we discuss the steps required to transform anomalies into CTI.

Anomaly Detection. The detection component continuously reports anomalies. Every anomaly is primarily defined by the feature that triggered the detection mechanism. However, the anomalous feature alone is not necessarily an appropriate indicator useful for CTI, because the occurrence

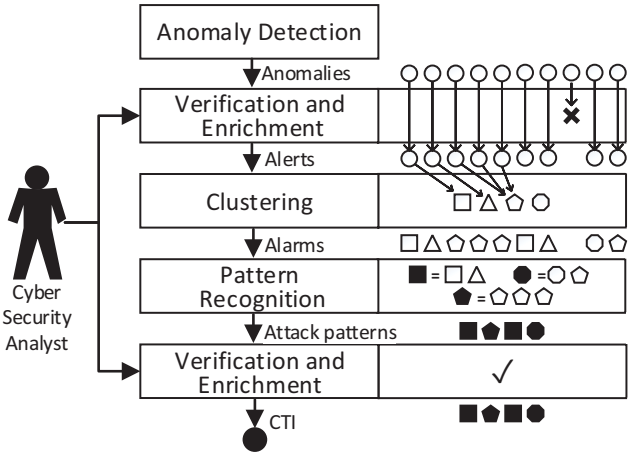

Figure 1. Procedure to transform anomalies to alerts, alarms, attack patterns, and CTI.

of the same feature in another context could be normal. For example, 100 login attempts per minute may be normal on a web server, but anomalous on a workstation. It is therefore necessary to enrich each anomaly with context data.

Verification and Enrichment. Relevant context data includes the event time, related events such as events occurring within small time frames, anomalous as well as normal values and their respective positions in the event as well as identifiers for the detector, log file, and system. Each anomaly is transformed into an alert with attributes storing contextual data as well as event and execution information. Alerts should be sorted out by domain experts if they are determined to be false positives.

Clustering. In busy systems, alerts occur frequently, which can be overwhelming for analysts. Alert clustering, i.e., allocating individual alerts to more abstract alarm classes, provides a remedy to this problem. In order to measure the similarity of alerts, we propose to compare their contextual, execution, and event attributes, where more identical values indicate a higher similarity, and vice versa. In order to ensure that alarms are representative for their allocated alerts, we append all alert attribute values to the alarm, or insert wildcards if the values are too diverse.

Pattern Recognition. Some alarms occur frequently and are part of different actions, making an allocation of alarms to actions difficult. However, alarms often occur in groups, for example, single actions carried out on computer systems usually trigger sequences of events executed within short time frames. Batches of alarms are in general better indicators for specific actions than single alarms. We therefore generate clusters of alarms that frequently occur together. Thereby, we measure the similarity of alarm sequences by computing the lengths of their common subsequences, where longer sequences indicate a higher similarity, and vice versa. We merge differing alarms that separate otherwise identical subsequences by joining their attributes to generate patterns

that represent all their allocated alarm sequences. We refer to the clustered sequences of alarms as action patterns.

Similar to alarms, actions are often executed in specific sequences in order to achieve certain tasks. We pursue an identical clustering strategy as for the generation of action patterns and refer to the resulting classes of action sequences as attack patterns. Variations of these attack patterns are considered during merging, where actions that provide equivalent capabilities are learned as alternatives [14]. Such a repeating behavior with variations is common in real-world scenarios, since adversaries frequently carry out attacks multiple times and reuse their techniques [15].

Verification and Enrichment. Besides modifications of attack steps, variations of action or attack patterns are also linked to technical conditions, such as particular services being available on the attacked system. Providing such data greatly improves the efficacy of our approach by reducing false positives during detection on other systems [13].

The resulting CTI alleviates the problems outlined in the beginning of this section. First, our approach requires minimal human intervention. Second, evasion is difficult, because attacks are described on a system level. In addition, our approach learns which steps, events, or artifacts are constant in each attack manifestation. Finally, even complex attacks are represented as patterns of detectable indicators.

IV. System Description

We implemented a proof-of-concept for the approach described in the previous section. In the following, we provide an overview and discuss applied mechanisms.

A. Overview

We designed a pipeline that automatically processes system log data and outputs attack patterns. Figure 2 displays an overview that shows that the system consists of a logging infrastructure and four analytical modules for parsing, anomalous system behavior detection, aggregation of anomalies to alarms, and alarm pattern recognition. The figure also depicts the data sent from one module to another with the numbers (1)-(6). Note that these snippets are shortened to fit the purpose of this visualization and that the actual data objects contain considerably more information.

The sample logs (1) are Network Time Protocol (NTP) events. We assume that it is possible to obtain anomaly-free log data for training, before detection and generation of attack patterns is carried out on the live system. In the following, we discuss each analytical module in detail.

B. Procedure

As outlined in Sect. III-B, system log data (1) typically contains unstructured text that requires preprocessing. We pursue two main goals: First, classify each individual log line as a log event, i.e., an abstract class that describes its underlying trigger. For example, the first two sample lines relate to the same event ("Listen and drop on"). Second, extract values such as IP addresses in a structured way.

Parsing. Log data must be parsed to achieve these goals. Existing approaches frequently use regular expressions to parse logs, however, our approach employs a parser tree [16]. Nodes of a parser tree are tokens that are either constant strings or variables of certain data types. The tree thus represents the grammar of a log file in a compact format and allows fast processing of lines.

The parser tree (2) for the NTP logs involves control elements such as sequences and branches as well as token elements that represent strings or variables. It involves a sequence of a fixed string element for the ntpd service name, an integer element for the process ID, and a branch followed by multiple elements. There exist references ("ref") for all elements and values ("value") that define constant strings.

As seen in the sample parsed logs (3), the parser tree facilitates a structured way of accessing tokens through paths, i.e., sequences of traversed nodes. For example, the path "/ntpdSeq/ntpd" indicates that log line $l1$ passed over the sequence ("ntpdSeq") and fixed string element ("ntpd") in the parser tree. The sample also shows that concrete values are accessible through these paths, e.g., path "/ntpdSeq/pid" points to process ID 16721. The event class of a log line is determined by the set of all its paths, i.e., log lines traversing the same paths correspond to the same log event.

Anomaly Detection. Parsed log data is suitable for anomaly detection. We consider the following types of anomalies relevant for our approach: (i) event-based outliers, i.e., log lines that do not fit the parser tree and thus represent new or unknown events, (ii) anomalous event sequences or correlations, i.e., log events appearing in new orders, (iii) timing-based anomalies, i.e., unusual inter-arrival time between log events, (iv) frequency-based anomalies, i.e., log events appearing too often or too rare within a certain time window, (v) value-based outliers, i.e., parameter values such as IP addresses that have not been observed before, (vi) value-combination-based outliers, i.e., new combinations of parameter values, and (vii) statistical anomalies, i.e., changes of continuous or discrete distributions of parameter values.

Each detection of unusual system behavior produces an anomaly e. The sample anomalies (4) show that different attributes are used depending on the involved detectors. For example, $e1$ provides a parser tree path and the associated IP address, while $e2$ is unable to provide such details due to the fact that event-based outliers are unparsable by definition. Note that due to space limitations, we omit presenting all attributes appended in the enrichment phase, but consider these anomalies as alerts in the following, i.e., $a1 = e1, a2 = e2, ...$ Note that we use subscripts to denote attributes, e.g., the anomalous value of alert $a1$ is $a1_v = e1_v = 10.0.0.1$.

Alarm generation. We intend to generate a set of alarms $\mathcal{B} = \{b1, b2, ...\}$ based on the disclosed sequence of alerts $a1, a2, ...$, where each alert is allocated to exactly one alarm

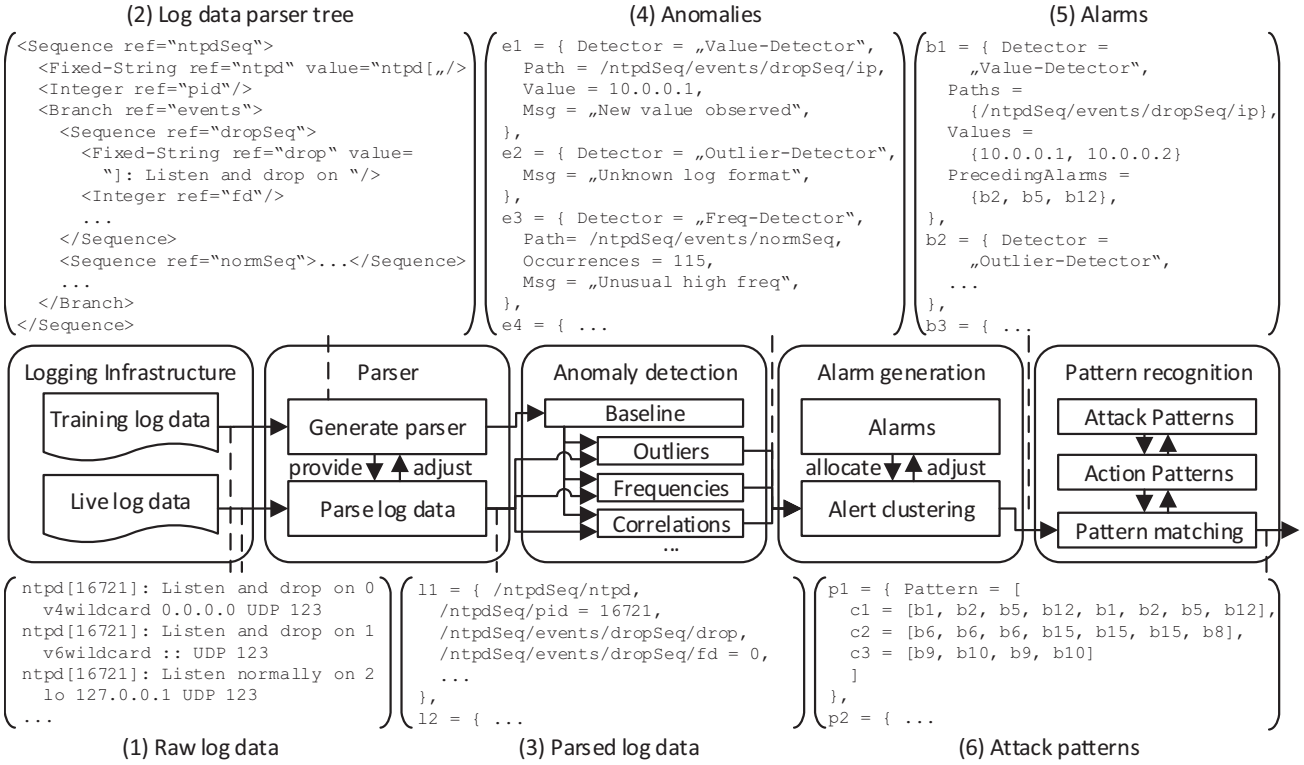

Figure 2. CTI extraction pipeline that processes raw log data in four analytical modules. (1)-(6) represents sample data flow.

and each alarm appropriately represents all allocated alerts. This is performed through incremental clustering and merging. We use a similarity metric based on weighted attribute matching, so that alerts and alarms with sufficiently many common attributes are considered similar. In particular, we compute the similarity between alert a and alarm $b \in \mathcal{B}$ as

$$sim_1(a,b) = \begin{cases} \sum\limits_{x \in \{p,v,ep,ev,o\}} \frac{|a_x \cap b_x|}{|a_x|} w_x & \text{if } a_d = b_d \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

where a_d is the detector that raised alert a, set a_p contains the paths and set a_v the corresponding anomalous values of the alert, set a_{ep} contains all paths and set a_{ev} all corresponding values of the affected event, set a_o contains related alarms, and w_x is the weight of attribute x. Note that $|a_x|$ is the size of set a_x and that weights should be selected so that $\sum_x w_x = 1$ holds. We consider alarms related to an alert if they occur within a time window δ_r before the alert.

The incremental clustering procedure that allocates each alert a to one of the existing alarm classes $b \in \mathcal{B}$ if the similarity exceeds a certain threshold, i.e., $sim_1(a,b) \geq \theta_1$, and adds a as a new alarm otherwise, i.e., $\mathcal{B} = \mathcal{B} \cup \{a\}$, enables fast processing of incoming alerts in a single pass. This also applies for merging, where alarms are updated with additional attribute values from their allocated alerts, i.e., $b_x = b_x \cup \{a_x\}, \forall x \in \{p, v, ep, ev, o\}$. Since alerts and alarms from different detectors are never similar according to Eq. 1, it is not possible that attributes are missing. The sample alarms (5) show that resulting alarms cover multiple alerts, e.g., alarm $b1$ contains two anomalous IP addresses.

Pattern recognition. The output of the alert clustering step is a continuous stream of alarms suitable for the detection of frequent alarm patterns. As outlined in Sect. III-D, patterns usually occur either on a relatively small time scale, i.e., alarms that belong to a single action are executed within milliseconds, or a much larger scale, i.e., sequences of actions that are executed seconds apart. We therefore carry out the frequent pattern detection on two levels: short-term action patterns and long-term attack patterns.

On the short-term level, we measure the inter-arrival time of alarms in order to locate subsequences that represent actions. This is achieved by cutting off sequences when the time difference to the next alarm exceeds a threshold δ_s. A value of $\delta_s = 0.1$ seconds has proven useful in empirical studies (cf. Sect. V). We then use incremental clustering to generate sequences of alarms that represent actions on the system. The Longest Common Subsequence (LCS) [17] is a suitable similarity metric for this purpose, because it enables the computation of a similarity between sequences as well as the disclosure of common patterns. Since LCSs may consist of interrupted patterns in the sequences, the results are robust against misclassifications and outliers.

The clustering procedure is as follows: Each complete alarm sequence s, i.e., a sequence of N alarms $s =$

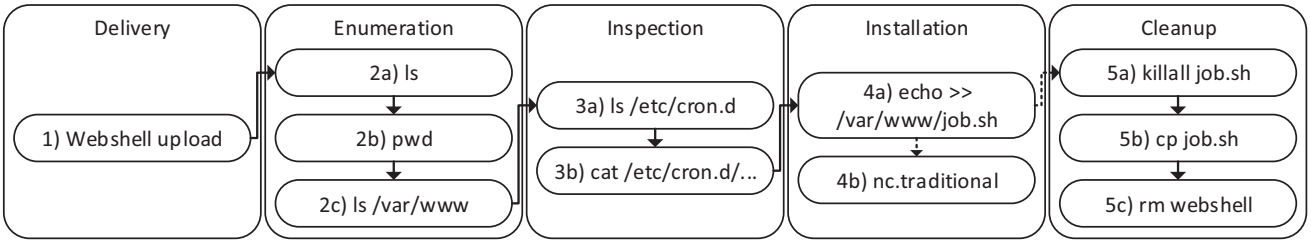

Figure 3. Multi-step attack executed on an Nginx web server. All steps are triggered by the attacker, except 4b which is triggered by the system itself.

$b1, b2, ..., bN$ that has no other alarms occurring within δ_s is allocated to the most similar action pattern $c \in \mathcal{C}$ if their similarity exceeds a certain threshold, i.e., $sim_2(s, c) \geq \theta_2$, and is added as a new action pattern otherwise, i.e., $\mathcal{C} = \mathcal{C} \cup \{s\}$. Thereby, the similarity is computed by the relative LCS, i.e.,

$$sim_2(s, c) = \frac{LCS(s, c)}{max(|s|, |c|)} \quad (2)$$

Each allocation of an alarm sequence s updates the action pattern c to appropriately represent all allocated alarm sequences. In particular, we merge different alarms in c and s that interrupt the LCS. For example, $c = b1, b2, b3$ and $s = b1, b4, b3$ differ in the second position and merging will thus update c to $c = b1, b5, b3$, where $b5_x = \{b2_x \cup b4_x\}, \forall x \in \{p, v, ep, ev, o\}$, i.e., all attributes are merged. The action pattern c remains unchanged if merging is not possible due to mismatching detectors or misaligned patterns.

We repeat this procedure with sequences of action patterns. Another threshold δ_l is used to disclose delimited action sequences of length M, e.g., $d = c1, c2, ..., cM$, and LCS is used to cluster these sequences into attack patterns, where each attack pattern $p \in \mathcal{P}$. A value of $\delta_l = 20$ seconds has proven useful in empirical studies (cf. Sect. V). We use the same similarity measure as before, i.e., d is allocated to p if $sim_2(d, p) \geq \theta_2$, and is added to the set of action patterns otherwise, i.e., $\mathcal{P} = \mathcal{P} \cup \{d\}$. The figure shows that each attack pattern (6) is a sequence of action patterns that in turn consist of alarm sequences. All patterns consist of alarms that are detectable, because their attributes are comparable with logs and raised anomalies; thus, the patterns themselves are detectable. In the following section, we provide an example that demonstrates this functionality.

V. Illustrative Scenario and Evaluation

Our approach is evaluated on real data. In this section, we outline the attack scenario and state the results.

A. Concept and Evaluation Environment

We demonstrate in an illustrative scenario that our approach is capable of extracting actionable CTI from raw log data. We show this by automatically extracting detectable TTPs that correspond to a multi-step attack carried out on a Nginx web server. We then use these TTPs to detect the same attack on an Apache web server. We selected these two technologies, because the execution of the same attack has different effects on the systems. In particular, web root directories, users, and available libraries are different, which are attributes that manifest themselves in the logs. Moreover, we configured the web servers so that Nginx is only accessible via https and Apache via http.

We set up the two virtual web servers to host the MANTIS Bug Tracker[4]. Normal behavior is generated by virtual users that perform tasks such as viewing and reporting bugs. After 30 minutes of learning the normal behavior, we switch our system to detection mode. After 30 more minutes, we execute the attack three times with an interval of 20 minutes.

The entire time we collect system call (syscall) logs using the audit daemon (auditd[5]) from the Linux Auditing System. We select syscalls for several reasons: First, they provide a low level view on system events. Second, they are available for almost all systems and same tasks executed on different machines generate similar syscalls. Finally, syscalls consist of key-value pairs and it is thus easy to define a parser tree.

B. Attack

The multi-step attack carried out on the Nginx web server is depicted in Fig. 3. The attack on Apache uses the Apache web root directory. We set up a vulnerability on the web servers that enables file uploads to arbitrary directories due to insufficient input validation (CAPEC-126 in the CAPEC database[6]). The attacker exploits this vulnerability to upload a web shell (1, CAPEC-650) that provides remote access. The attacker first inspects (2a-2c) the system to find a cron job suitable for exploitation (3a-3b) and then injects a netcat script (4a, CAPEC-19) in an existing cron job. This job is executed by root and is thus able to open a netcat connection (4b) with elevated privileges (CAPEC-233). The system executes the script within the next 2 minutes. We finish the intrusion by resetting the system to the normal state, i.e., killing the cron job (5a), overriding the injected script (5b), and removing the uploaded webshell (5c).

C. Results

During the first 30 minutes, 40,000 syscalls that correspond to normal system behavior were generated. For each

[4]https://www.mantisbt.org/
[5]https://linux.die.net/man/8/auditd
[6]https://capec.mitre.org/

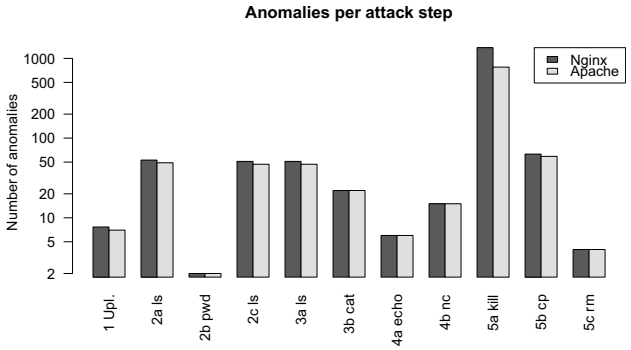

Figure 4. Anomalous event counts for each of the attack steps. A similar amount of anomalies is generated on the Nginx and Apache web server.

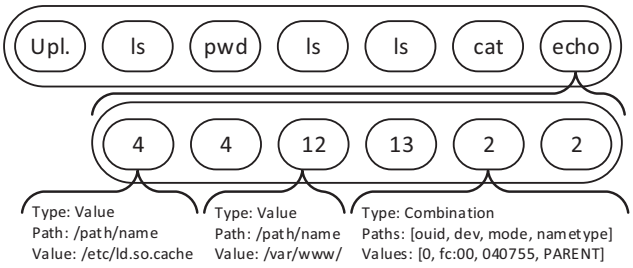

Figure 5. Top: Attack pattern. Center: Action pattern of one attack step. Bottom: Sample alarms, shortened for brevity.

execution of the attack, the anomaly detection component raised around 1600 anomalies on Nginx and 1000 anomalies on Apache. Figure 4 shows the number of anomalies associated with each attack step. The plot shows that the number of anomalies varies greatly depending on the particular action executed on the system. However, the plot also shows that the command "ls" executed in steps 2a, 2c, and 3a yields similar amounts of anomalies. Closer inspection showed that most the types, sequences and parameter values are similar across these steps. We therefore conclude that it is possible to recognize particular actions by the syscall sequences they generate. Another important aspect depicted in the plot is that Nginx and Apache roughly produce the same amount of anomalies, indicating that the action patterns learned on Nginx are sufficiently similar for detection on Apache.

Our proposed approach identified 27 different alarms, which corresponds to an alert reduction rate of 98.3% with respect to a single execution of the attack on Nginx. After splitting the stream of alarms in chunks, each attack resulted in 11 alarm sequences, corresponding to the 11 actions in the multi-step attack. During clustering, the sequences corresponding to the "ls" command were merged, resulting in 9 action patterns. Finally, these short-term action patterns are combined into long-term attack patterns. Due to time delays, a single execution of the attack yields three separate attack patterns: steps 1-4a, step 4b, and steps 5a-5c. Fig. 5 shows steps 1-4a and the composition of the action pattern of the "echo" command, which consists of 6 alarms. The

Table I
SIMILARITY MATRIX BETWEEN ATTACK STEPS FROM APACHE WEB SERVER (ROWS) AND LEARNED ACTIONS FROM NGINX WEB SERVER (COLUMNS). THE HIGHEST SIMILARITY IS MARKED BOLD.

	Upl.	ls	pwd	cat	echo	nc	kill	cp	rm
1	**0.86**	0.04	0.0	0.0	0.29	0.13	0.0	0.03	0.0
2a	0.02	**0.92**	0.04	0.08	0.04	0.02	0.01	0.17	0.08
2b	0.0	0.04	**1.0**	0.09	0.33	0.0	0.0	0.03	0.5
2c	0.0	**0.92**	0.04	0.09	0.04	0.0	0.01	0.17	0.09
3a	0.0	**0.92**	0.04	0.09	0.04	0.0	0.01	0.17	0.09
3b	0.0	0.08	0.09	**1.0**	0.09	0.0	0.0	0.06	0.18
4a	0.29	0.08	0.33	0.09	**0.83**	0.13	0.0	0.06	0.33
4b	0.13	0.04	0.0	0.0	0.13	**1.0**	0.0	0.03	0.0
5a	0.0	0.02	0.0	0.01	0.01	0.0	**0.56**	0.02	0.01
5b	0.03	0.22	0.03	0.07	0.08	0.03	0.01	**0.92**	0.07
5c	0.0	0.08	0.5	0.18	0.33	0.0	0.0	0.06	**1.0**

numbers depict the ID of the alarm, same IDs mean that the alerts were allocated to the same alarm during clustering. In the bottom of the figure, sample alarm descriptions are provided that include type, affected paths and anomalous values. Note that we omit further attributes for brevity.

As part of our illustrative scenario, we assume that the attack patterns have been manually validated by security analysts and are shared with another organization that runs Apache web servers. In the context of our approach, sharing CTI means that the parser tree as well as configurations of anomaly detectors are provided so that the detection of the same anomalies on another system is possible. Furthermore, we share the alarms, action patterns and attack patterns to enable matching. The other organization then sets up the detectors and uses the parser tree to detect anomalies as usual (cf. Fig. 2). During all clustering steps, the algorithm matches the alerts with the shared alarms and further matches the sequences of alarms with the shared action and attack patterns. In case that a match yields a similarity above a predefined threshold, the algorithm allocates the objects to the respective classes. We used the parameters $w_p = w_v = 0.3, w_{ep} = w_{ev} = 0.1, w_o = 0.2, \theta_1 = 0.51, \theta_2 = 0.65$.

In our evaluation, only 16 out of the 1000 anomalies from the Apache web server were not allocated to any of the alarm classes from the Nginx web server. These and other misclassifications are caused by dissimilarities of values such as directory names different on both servers. Despite their negative influence, we demonstrate that it is reasonable to match alarm sequences generated on Apache with action patterns mined from Nginx by computing their pairwise similarities. Table I shows the similarities of all combinations of executed attack steps (rows, cf. Fig. 3) and known action patterns (columns). Each attack step is assigned the class of the best matching action pattern, i.e., the action pattern achieving the highest similarity (bold). The table shows that each step of the multi-step attack is allocated to the correct action pattern, e.g., step 4a executes an "echo" command (cf. Fig. 3) and is allocated to the "echo" action pattern with a similarity of 0.83. Note that 2a, 2c, and 3a are all assigned to the action pattern "ls".

The highest similarities should be considered in relation to the similarities of the other classes for each particular attack step, e.g., step 5a only achieves a maximum similarity of 0.56, but is still allocated with a high confidence, due to the fact that the second highest similarity is only 0.02.

Finally, we derive the long-term attack patterns mined on the Apache web server. Due to the fact that all attack steps were correctly assigned to action patterns and their timing and ordering were not changed, the resulting attack patterns on Apache match perfectly to the ones from Nginx.

VI. Discussion

The results of our illustrative scenario show that despite system-dependent variations, our approach is able to extract actionable CTI that supports the detection of attacks on other systems. In our demonstration, we did not explore variations on the steps of the attack chain, because their effects are trivial. For example, skipping the optional step 2a when executing the attack on the Apache web server does not change the success of the attack, but results in an attack pattern that lacks one action in comparison to the TTPs generated on the Nginx web server, therefore reducing their similarity. However, due to the fact that our approach generates attack patterns by computing the LCS of repeatedly occurring attacks, it is sufficient to observe such a variation of an attack once to adjust the pattern accordingly. We argue that sufficient observations of variations will eventually reduce the generated TTPs to their most essential steps that are necessary for the attack to be successful.

The generated attack patterns are actionable due to the fact that they are automatically detectable, and correspond to complex patterns that are difficult to change, e.g., syscall sequences generated by particular actions. The main advantage over other approaches, such as malware profilers, is that attacks are reduced to their most characteristic features, so that detection is largely independent of variations.

It must be noted that sharing our CTI is not as simple as sharing traditional IoCs, such as IP addresses. The reason for this is that the parser tree, information on deployed anomaly detectors and their configurations, alarm clusters, as well as action and attack patterns must be shared to enable the most effective detection. Because log data often contains sensitive information, organizations may be reluctant when it comes to sharing alarms that possibly contain contextual and event execution information [15]. In addition, making highly specific attack patterns publicly available may facilitate reproduction of attacks by adversaries [18], who reuse or modify the intrusion procedure and target unprotected systems. We therefore suggest to screen any manually or automatically produced CTI that leaves an organization.

Many organizations operate great numbers of similar configured machines used for the same tasks. Since failures and attacks are expected to manifest themselves in a similar way on these machines, applying our approach for mining CTI and detecting threats could effectively improve resilience. In addition, security analysts could carry out attacks or otherwise dangerous behavior to generate detectable patterns suitable to be rolled out on many systems.

VII. Related Work

Alert aggregation and the detection of multi-step attacks are well-researched problems in cyber security. Thereby, most approaches aggregate, correlate, and connect alerts by the similarity of certain features. For example, the approach proposed by Julisch [19] generalizes alerts received from IDSs by aggregating their attributes using hierarchies. Valdes and Skinner [20] use features such as IP and port lists, user and sensor IDs, and time to compute the similarity for probabilistic alert correlation. Qiao et al. [17] also take IP, time, and the type of alerts into account for similarity computation and then use LCS to extract the attack patterns. Pei et al. [21] generate a graph based on the presence of certain attributes and then perform graph community extraction to derive attack patterns. Almost all such approaches rely on the assumptions that attack steps are detectable by traditional IDSs, i.e., that predefined signatures for detection exist, and that alerts are available in well-structured formats that facilitate feature-based similarity computation. In contrast to these methods, the approach proposed in this paper focuses on unknown anomalies detected in raw log data and incorporates every attribute available in the log lines as well as detector information for similarity computation.

Mapping security events to existing TTPs is able to enrich human-readable attack descriptions with detectable events. Scarabeo et al. [22] use methods from text mining to map IDS alerts to attack descriptions provided by CAPEC [10]. Navarro et al. [12] derive context and patterns from log events to generate a complex attack model suitable for matching with TTPs from threat databases.

Other approaches automatically generate CTI from sources other than log files. Husari et al. [4] use machine learning to extract threat actions from human-readable CTI reports. They then map the sequences of actions to known attack patterns and output them in STIX [9] format. Zhu and Dumitras [6] propose an algorithm that automatically analyzes online security articles and generates detectable patterns that consist of combinations of IoCs. Among the drawbacks of these methods is that they rely on the availability of manually written threat reports, which do not necessarily provide a comprehensive view on the affected systems. In addition, creating these reports is time-consuming and thus threat information is not immediately available after incidents. We argue that log data captures the system in more detail and enables real-time CTI generation and detection.

VIII. Conclusion

In this paper, we introduced an approach for the automatic extraction of actionable threat intelligence from raw

log data. Other than existing methods that process well-formatted output of IDSs, our approach employs anomaly detection to enable the disclosure of unknown attacks. We pursue iterative clustering and enrichment of anomalies with optional human verification with the purpose of transforming low-level log events to complex attack patterns.

With this approach, we provide a solution to shortcomings of state-of-the-art threat intelligence. In particular, existing IoCs do not offer sufficient protection due to their limited validity and simple nature that enables evasion. TTPs solve this problem by providing abstract attack descriptions, but are only available in human-readable format that prevents automatic detection. Our research efforts aim at combining the advantages of IoCs and TTPs by mining detectable patterns describing complex behavior. We showed in an illustrative scenario that our approach is capable of extracting an attack pattern corresponding to a multi-step attack that is suitable to be used for detection on another system.

Our next steps are to test our approach with different attack scenarios and to improve the robustness of our clustering and pattern recognition algorithms. For this, we plan to set up and evaluate our approach in a complex environment, where extraction and detection is complicated by outliers and simultaneously occurring attacks.

ACKNOWLEDGMENT

This work was partly funded by the FFG projects INDICAETING (868306) and synERGY (855457), and the EU H2020 project GUARD (833456).

REFERENCES

[1] W. Tounsi and H. Rais, "A survey on technical threat intelligence in the age of sophisticated cyber attacks," *Computers & Security*, vol. 72, pp. 212 – 233, 2018.

[2] D. Rhoades, "Machine actionable indicators of compromise," in *Proceedings of the 48th International Carnahan Conference on Security Technology*. IEEE, 2014, pp. 1–5.

[3] D. Chismon and M. Ruks, "Threat intelligence: Collecting, analysing, evaluating," Tech. Rep., 2015, MWR InfoSecurity.

[4] G. Husari, E. Al-Shaer, M. Ahmed, B. Chu, and X. Niu, "Ttpdrill: Automatic and accurate extraction of threat actions from unstructured text of cti sources," in *Proceedings of the 33rd Annual Computer Security Applications Conference*. ACM, 2017, pp. 103–115.

[5] J. Navarro, A. Deruyver, and P. Parrend, "A systematic survey on multi-step attack detection," *Computers & Security*, vol. 76, pp. 214 – 249, 2018.

[6] Z. Zhu and T. Dumitras, "Chainsmith: Automatically learning the semantics of malicious campaigns by mining threat intelligence reports," in *Proceedings of the European Symposium on Security and Privacy*. IEEE, 2018, pp. 458–472.

[7] R. McMillan, "Definition: Threat intelligence," *Gartner.com*, 2013, accessed: 2019-05-13. [Online]. Available: https://www.gartner.com/en/documents/2487216

[8] H. Dalziel, E. Olson, and J. Carnall, "How to define and build an effective cyber threat intelligence capability." Syngress Publishing, 2014.

[9] S. Barnum, "Standardizing cyber threat intelligence information with the structured threat information expression (stix)," Tech. Rep., 2012, Mitre Corporation.

[10] S. Barnum, "Common attack pattern enumeration and classification (capec) schema description," Tech. Rep., 2006, Cigital.

[11] D. Bianco, "The pyramid of pain: Intel-driven detection & response to increase your adversary's cost of operations," Tech. Rep., 2014, RVASec.

[12] J. Navarro *et al.*, "Huma: A multi-layer framework for threat analysis in a heterogeneous log environment," in *Proceedings of the 10th International Symposium on Foundations and Practice of Security*. Springer, 2017, pp. 144–159.

[13] F. Cuppens and A. Miege, "Alert correlation in a cooperative intrusion detection framework," in *Proceedings of the 23rd IEEE Symposium on Security and Privacy*. IEEE, 2002, pp. 202–215.

[14] V. Mavroeidis and S. Bromander, "Cyber threat intelligence model: An evaluation of taxonomies, sharing standards, and ontologies within cyber threat intelligence," in *Proceedings of the European Intelligence and Security Informatics Conference*. IEEE, 2017, pp. 91–98.

[15] S. E. Dog *et al.*, "Strategic cyber threat intelligence sharing: A case study of ids logs," in *Proceedings of the 25th International Conference on Computer Communication and Networks*. IEEE, 2016, pp. 1–6.

[16] M. Wurzenberger, M. Landauer, F. Skopik, and W. Kastner, "Aecid-pg: A tree-based log parser generator to enable log analysis," *Proceedings of the 4th International Workshop on Analytics for Network and Service Management*, 2019.

[17] L.-B. Qiao, B.-F. Zhang, Z.-Q. Lai, and J.-S. Su, "Mining of attack models in ids alerts from network backbone by a two-stage clustering method," in *Proceedings of the 26th International Parallel and Distributed Processing Symposium*. IEEE, 2012, pp. 1263–1269.

[18] S. Barnum and A. Sethi, "Attack patterns as a knowledge resource for building secure software," in *Proceedings of the OMG Software Assurance Workshop*, 2007, Cigital.

[19] K. Julisch, "Clustering intrusion detection alarms to support root cause analysis," *ACM transactions on information and system security*, vol. 6, no. 4, pp. 443–471, 2003.

[20] A. Valdes and K. Skinner, "Probabilistic alert correlation," in *Proceedings of the 4th International Workshop on Recent Advances in Intrusion Detection*. Springer, 2001, pp. 54–68.

[21] K. Pei, Z. Gu, B. Saltaformaggio, S. Ma, F. Wang, Z. Zhang, L. Si, X. Zhang, and D. Xu, "Hercule: Attack story reconstruction via community discovery on correlated log graph," in *Proceedings of the 32nd Annual Conference on Computer Security Applications*. ACM, 2016, pp. 583–595.

[22] N. Scarabeo, B. C. Fung, and R. H. Khokhar, "Mining known attack patterns from security-related events," *PeerJ Computer Science*, vol. 1, p. 21, 2015.