A Framework for Automatic Labeling of Log Datasets from Model-driven Testbeds for HIDS Evaluation

Max Landauer, Maximilian Frank, Florian Skopik, Wolfgang Hotwagner, Markus Wurzenberger
firstname.lastname@ait.ac.at
Austrian Institute of Technology
Vienna, Austria

Andreas Rauber
rauber@ifs.tuwien.ac.at
Vienna University of Technology
Vienna, Austria

ABSTRACT

Intrusion detection systems are essential for network security. To verify their detection capabilities and facilitate comparison, benchmark log datasets are used to measure evaluation metrics such as accuracy and false alarm rates. Thereby, it is necessary that these datasets come with a correct ground truth that differentiates normal and attacker behavior. While it is relatively straightforward to generate labels for network-based datasets by selecting events according to IP addresses of attacker hosts, system logs do not necessarily involve such identifiers and are possibly only recognizable as malicious by their combined occurrences. Even more problems emerge when log data is collected in model-driven testbeds, i.e., automatically generated networks that simulate differently parameterized attack scenarios in diverse infrastructures. In these testbeds, parameters such as IP addresses are subject to change and thus cannot simply be used for matching. We thus propose a framework that integrates template-based labeling rules for model-driven testbeds. In this paper we describe the syntax for rule templates with different query types specifically designed to match sequential or interrelated system log events. An evaluation of our open-source implementation shows that only 27 rules are necessary to assign 15 labels to 8 system log files containing attack manifestations.

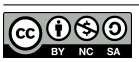
CCS CONCEPTS

• **Information systems** → **Information retrieval query processing**; • **Security and privacy** → **Intrusion detection systems**; *Systems security*.

KEYWORDS

log data analysis, intrusion detection, log testbed, detection rules

ACM Reference Format:
Max Landauer, Maximilian Frank, Florian Skopik, Wolfgang Hotwagner, Markus Wurzenberger and Andreas Rauber. 2022. A Framework for Automatic Labeling of Log Datasets from Model-driven Testbeds for HIDS Evaluation. In *Proceedings of the 2022 ACM Workshop on Secure and Trustworthy Cyber-Physical Systems (SaT-CPS '22), April 27, 2022, Baltimore, MD, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3510547.3517924



SaT-CPS '22, April 27, 2022, Baltimore, MD, USA.

ACM ISBN 978-1-4503-9229-7/22/04.
https://doi.org/10.1145/3510547.3517924

1 INTRODUCTION

Cyber attacks pose threats to computer systems at any scale. To counteract the continuously increasing frequencies and sophistication of such attacks, security experts are perpetually improving Intrusion Detection Systems (IDS), which autonomously monitor system behavior for suspicious activities and report alerts in case that manual intervention is required. Thereby, it is possible to differentiate host-based IDSs (HIDS) that analyze system logs and network-based IDSs (NIDS) that monitor network packets, as well as signature-based IDSs that detect predefined attack indicators and anomaly-based IDSs that rely on self-learning [14].

Independent of their type, it is common to compute detection accuracies and false alarm rates of IDSs on benchmark datasets as part of their evaluation. This is essential to verify the capabilities of IDSs, quantitatively compare performances of different approaches, and develop new detection techniques. A crucial aspect is thereby the availability of ground truths for benchmark datasets that allow to differentiate benign from attacker behavior and thus classify detection results [10]. Only when this ground truth is created in a transparent and documented process, evaluation results are reliable and representative. While it is relatively easy to check whether reported alerts are within start and stop times of launched attacks, this strategy is insufficient to appropriately label long-term attacks and does not address the issue that benign logs occur simultaneous to attacks. Therefore, labels should be assigned to single log instances rather than chunks of logs from certain time intervals.

In general, the process of labeling log data makes use of similar techniques as signature-based IDSs: Log events are scanned for particular keywords, e.g., IP addresses of attacker hosts, and labels are assigned to matching logs to categorize them as malicious [2]. This is intuitively reasonable for simple scenarios, for example, where all activities originating from a dedicated attacker machine are known to be malicious. In testbeds where analysts have full control and information about the simulated attackers, labeling is trivial for such cases [21]. However, more complex and realistic testbeds involve attack manifestations that are interleaved with traces of normal system behavior, which makes it difficult to discern these two classes. In particular, labeling by simple IP-based matching is impossible when normal and malicious activities are executed simultaneously on the same host, which necessarily occurs when attackers manage to compromise and misuse actively used systems. Even worse, system logs that are necessary to evaluate HIDSs rarely contain network information and do not even need to involve any expressive descriptors for keyword matching; in fact, manifestations of attacks are possibly identical to normal events and only their combined occurrences in a specific execution context allow to

determine their root cause. In addition, system logs are usually unstructured and generated in heterogeneous formats so that labeling rules cannot simply be applied on all log files [27]. Unfortunately, common ways of data labeling based on keyword matching are thus unable to adequately label logs for HIDS evaluation.

In recent scientific works, log datasets were generated with model-driven testbeds [7, 15]. Such testbeds model the network infrastructure, simulated user behavior, and attack scenarios as abstract templates that leave out specific parameters, e.g., IP addresses or user names. Only when testbeds are instantiated from the models, these variables are dynamically selected and thus generate unique testbed configurations. Model-driven testbeds have several benefits, including low manual effort for deployment, high reusability, and the possibility to generate many testbeds with variations that represent large varieties of IT environments and thus enable robust evaluations [19]. However, these properties also imply that labeling rules based on hard-coded values cannot simply be reused across testbeds, since these values are purposefully subject to change. Therefore, it is necessary to repeatedly adapt labeling rules, creating a bottleneck for model-based testbed generation.

With the Kyoushi framework we aim to resolve aforementioned problems by integrating labeling rule design into a model-driven testbed generation process. For this, our framework utilizes abstract labeling rule templates that are completed with automatically extracted testbed parameters. In addition, we propose four rule types that allow to label system log datasets for HIDS evaluation. We summarize our contributions as follows:

- A framework for model-driven labeling of system log data[1].
- A publicly available labeled dataset for HIDSs evaluation[2].
- An evaluation of the generated dataset labels.

The remainder of this paper is structured as follows. Section 2 reviews labeling strategies of existing datasets. Section 3 explains relevant concepts of the Kyoushi framework. Section 4 describes requirements and technical details of our labeling rule templates. We present an illustrative scenario and evaluation thereof in Sect. 5 and discuss the results in Sect. 6. Section 7 concludes the paper.

2 RELATED WORK

In the last years, many scientific works have proposed techniques for log- or network-based intrusion detection [14]. In contrast, only few benchmark datasets have been published for evaluating such detectors, and the lack of representative data is an often stated problem in scientific literature [5, 12, 15, 19, 21]. The issue of labeling datasets is often neglected, even though a reliable ground truth is essential to calculate evaluation metrics [10, 22]. We therefore review existing datasets and focus on their labeling strategies.

CIDDS [21] is a labeled netflow dataset for NIDS evaluation generated by user simulations on a virtualized testbed. As stated by the authors, labeling network traffic in testbeds is simple, because sources, destinations, and timestamps of attack-related flows are known. However, since their testbed is accessible over the Internet, external and possibly malicious activities could occur at any time. The authors therefore label flows that cannot be classified otherwise as unknown or suspicious depending on the targeted ports.

Similarly, Sharafaldin et al. [23] generate a ground truth for the CIC-IDS dataset by labeling flows where IP addresses, ports, and protocols match their attacker host. They assign labels for specific attacks by matching flow timestamps with their attack schedule. Buchanan et al. [3] apply a similar scheme and state that they are able to label 99.9% of all events. The ISCX dataset [25] is labeled by utilizing network resource information as well as information gathered from the attack profiles. The flow-based dataset CTU-13 [9] is also labeled by matching IP addresses of the involved components. The authors differentiate between normal traffic originating from known hosts, botnet traffic from or to infected machines, and all other background traffic. The authors of the UGR'16 [17] dataset point out that such a strategy is only feasible when normal behavior generation is part of the simulation. They use a combination of signature-based labeling and anomaly-based labeling. The problem with such anomaly-based strategies is that labeling accuracy depends on the detectors and is thus prone to errors.

Other than NIDS datasets where labeling based on IP addresses is simple, HIDS datasets require alternative strategies. For example, ADFA-LD [5] is a dataset for HIDS evaluation that contains system call traces, i.e., sequences of low-level system operations handled by the kernel. Since each system call is executed in context of a specific process, the authors were able to differentiate normal from malicious traces of purposefully launched attacks for labeling. While the ADFA-LD is limited to system call numbers, the LID-DS [11] enables more realistic evaluations by involving additional trace information such as process names, arguments, and return values. The authors differentiate normal and attack behavior, where all traces generated after the starting times of attacks are labeled as such. The AWSCTD [4] also collects system calls with parameters and labels individual traces.

Labeling other log types that do not involve process information is even more difficult. For example, Skopik et al. [26] collect application logs from databases and web servers. They record start and stop times of attacks to label logs, but are unable to differentiate benign from malicious log lines in this interval. In addition to such a time-based labeling technique, the AIT-LDSv1.1 [15] relies on manually crafted attack log dictionaries and similarity metrics to label individual lines with sufficient similarity. The disadvantages of this method include high manual effort and the chance of incorrectly assigned labels due to inappropriate similarity thresholds.

He et al. [12] provide a collection of several system log datasets, of which some are labeled by handcrafted rules that are known to match anomalous events. However, labeled logs mostly relate to errors or misconfigurations rather than attacks, limiting their relevance for IDS evaluation. In their study on system logs, Oliner et al. [20] explain that they consulted system administrators to label logs related to system issues through pattern matching. While labeling rules that are an integral part of our proposed framework also rely on domain knowledge about the attack, they are not specific to a particular system or attack execution by leveraging templates. As such, they have the advantage of being reusable across many instances where logs should be labeled.

Huang et al. [13] use a neural network to label a large system log dataset based on a smaller one. Of course, this still leaves the problem of manually labeling a representative amount of events in an initial dataset. As such, this method is more appropriate

[1] Accessible at https://github.com/ait-aecid/kyoushi-dataset
[2] Accessible at https://zenodo.org/record/5779411

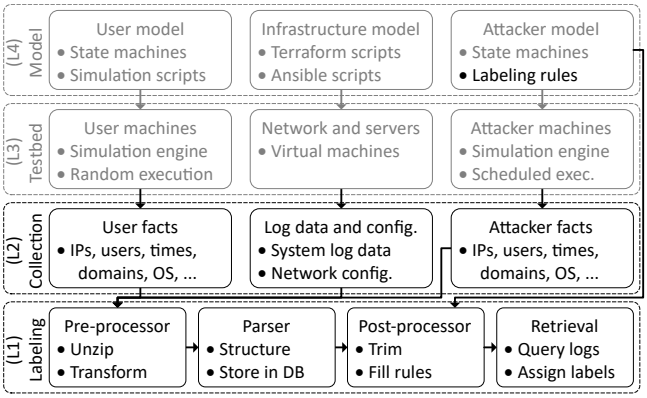


Figure 1: Labeling concept for model-driven testbeds.

for events describing erroneous system states rather than attack scenarios. The approach proposed by Makiou et al. [18] also relies on an initial labeled training dataset and uses signature-based and anomaly-based detectors that run in parallel to assign labels to a larger dataset. This assumes that public signatures as well as detector configurations are suitable to detect attack manifestations with high accuracy, which may be difficult to realize in practice. Rather than labeling a log data sample, our approach pursues the generation of generic rules that assign labels to the same attack type independent of its execution context or targeted system.

Similar to our approach, Sigma[3] describes log events with structured detection rules. The format supports regular expressions, allows to link queries with logical operators, includes time-based filters, and allows to automatically modify values, e.g., apply base64 decoding prior to matching. However, to our knowledge there is no feature that allows to specify sequences of events or nested queries, which is necessary to label complex attack manifestations and thus realized in the Kyoushi framework.

3 CONCEPT

As outlined in the previous section, testbeds are commonly used for the generation of log data suitable for evaluating HIDSs. While most of these testbeds are relatively static, the Kyoushi framework employs a model-driven methodology for testbed instantiation. Our framework is named Kyoushi after the Japanese term for *teacher*, which also refers to training datasets. In this section we first give an overview of model-driven testbed generation and then describe the integration of our proposed log data labeling procedure into such an approach. Figure 1 shows an overview of the layers involved in the concept: layers *(L4) model* and *(L3) testbed* are known from existing works [7, 15] and layers *(L2) collection* and *(L1) labeling* are the main contributions of this paper. In the following, we describe each layer in detail.

3.1 Model-driven Testbed Generation

The purpose of introducing concepts from model-driven engineering in testbed generation is to ease and automate deployment of simulations. This is especially useful when attack executions are repeated multiple times and with changes in configurations, which is essential to adequately depict the diversity of networks occurring in real-world scenarios, increase robustness of evaluation results by obtaining more distinct datasets, and enable evaluation of alert aggregation approaches [15, 19]. The key idea is to design components of the testbed as abstractions of the actual technical realizations, i.e., setup routines of components are templates that leave several pre-selected parameters as variables, including IP addresses, user names, behavior profiles, network size, etc. These templates are usually referred to as testbed-independent models (TIM) [7, 15].

The model layer (L4) deals with these templates and differentiates between models for infrastructure, user behavior, and attacks. Infrastructure models include templates for setup and provisioning scripts for all involved components. Normal user behavior is modeled through state machines, where states act as decision points that specify the order and frequencies of actions carried out when moving between states. Similarly, attacker models also involve state machines and scripted commands, but additionally contain templates for labeling rules that describe attack artifacts in log data in an abstract way, i.e., without specifying parameter values that are unknown at the time of modeling since they are only selected during instantiation of the testbed.

The testbed layer (L3) holds all testbed-specific models (TSM) generated by instantiating TIMs. For this, a transformation engine selects parameter values by predefined functions, e.g., randomly selects IP addresses from pools or user names from databases. The automatically generated TSMs are executable scripts for setting up and running specific testbed instances without manual interference.

3.2 Model-driven Log Data Labeling

Once the simulation running on the rolled out testbed is completed, the collection layer (L2) handles the extraction of relevant information from the components. Foremost, this concerns the log data to be labeled, which is collected from machines that are typically monitored by IDSs, e.g., web servers and firewalls. In addition, we collect logs relevant for labeling from all other machines in the testbed. Thereby, logs from the attacker machine that outline the timeline of attack executions are likely the most important source of information; however, also logs from hosts running simulations of normal user behavior can be used to verify attack labels or assign labels to benign activities. Moreover, we gather system information that we refer to as *facts* from all components. Facts are the main source for filling out labeling rule templates and include artifacts such as IP addresses, domain names, user names, OS versions, etc [8]. Finally, we also extract configuration files of installed services and logging frameworks. All gathered data is then transmitted to a central storage system where labeling takes place.

The labeling layer (L1) is the most crucial part for generating ground truth labels and thus the main focus of this paper. The labeling procedure consists of four steps. First, a pre-processor prepares the logs for further analysis. In particular, this includes unzipping log files that are compressed during rotation and converting logs stored as binary or other formats into raw text files. Second, a parser transforms the system log data, which is usually only available in unstructured form, into a semi-structured format so that all tokens can be referenced individually. These parsed log events are then loaded into a database that supports storing and searching such

[3] https://github.com/SigmaHQ/sigma

semi-structured data. Third, a post-processor trims the logs to fit the desired simulation time interval and prepares all rules by inserting facts extracted by the collection layer into rule templates designed as part of the attacker model. Fourth, the retrieval step iterates over all rendered labeling rules and executes queries on the database storing the parsed log data so that all matching events are assigned one or more labels corresponding to the respective rule. The following section provides more details on these labeling rule templates, in particular, an overview of different query types.

4 KYOUSHI FRAMEWORK

The previous section outlines the overall concept of the model-driven Kyoushi framework and showed the integration of our log data labeling procedure in an automatic testbed generation approach. In this section we first discuss the requirements on such a labeling procedure and then go into more detail on the design of labeling rules and realization of our approach.

4.1 Requirements

For unbiased and representative evaluation of IDSs it is essential that the used monitoring data fulfills certain requirements [22]. Accordingly, most works mentioned in Sect. 2 discuss requirements on the infrastructure where data is collected [15] as well as the log data itself [17, 21]. However, existing works rarely put any special requirements on the data labeling procedure. We therefore collected relevant insights from existing approaches and aggregated them in the following list of requirements on a log data labeling procedure.

Correctness. Labels should be assigned to log events that are consequences of attacker behavior, while all other events originating from normal user or system activity should be recognizable as benign. Log data labeled without a transparent strategy is of little value for HIDS evaluation, since any results may be disputable.

Automation. Since log data is generated in high volumes, it is usually infeasible to manually label all or even only parts of the logs. Strategies based on simple keyword matching for labeling (cf. Sect. 1) have limited expressiveness and require manual adjustments when relevant artifacts such as IP addresses are subject to change. To alleviate these issues, a log data labeling procedure should reduce such repetitive and time-consuming manual tasks to a minimum and pursue label assignment as a fully automatic procedure.

Adaptability. Model-driven testbeds enable adapting, extending, and reusing components within and across testbeds multiple times without the need to always start from scratch. Since any modifications of the TIMs, e.g., changes of attack scripts or logging infrastructure settings, possibly influence the way attacks manifest in the logs, it is necessary to adjust the labeling rules accordingly. A log data labeling framework should therefore support adaptation and extension of the existing ruleset in accordance with all TIMs.

Granularity. Labeling logs solely by start and stop times of attacks is not sufficient, since this strategy incorrectly labels logs corresponding to normal behavior that occur interleaved with attack logs as malicious. In addition, logs corresponding to long-term attacks that span over large time windows cannot appropriately be labeled. Labels should be therefore be assigned to individual events.

Moreover, instead of binary labels that only differentiate between benign and malicious events, distinct labels should be assigned for each attack activity to increase evaluation granularity, e.g., compute detection accuracy for different attack types [3]. This also allows to arrange labels in a hierarchical order so that relationships between attacks, attack steps, and commands, become apparent.

Applicability. Since log data originates from many distinct sources, the labeling procedure needs to support a wide range of log formats. In addition, attack consequences are diverse and may affect single or multiple events that occur with delays or across different log files. The framework should thus offer techniques to label such dispersed and interrelated events.

Reproducibility. Both testbeds and labeling procedures should only involve open-source tools so that simulations and log datasets are reproducible by others. This concerns technologies required to run the labeling framework, e.g., log databases, as well as labeling rule templates that should be published along with the TIMs.

We designed the Kyoushi framework with the aforementioned requirements in mind. The following section describes the design of labeling rule templates, which are a core element of our approach.

4.2 Labeling Rule Templates

As outlined in Sect. 1, common labeling strategies are usually centered around searching for specific keywords, e.g., the IP address of an attacker machine, and marking all matching log events as malicious. Unfortunately, this is not possible in model-driven testbed generation approaches, because these keywords are not available at the time of designing the TIMs. Furthermore, system log data is not always discernible by such keywords and only combined and contextual occurrences of events allow correct label assignment. To overcome these issues, we propose labeling rule templates that are designed on the same level of abstraction as TIMs and are thus independent of artifacts specific to TSMs. In addition, we propose four types of rules to enable the assignment of labels to events that could not be labeled with common labeling strategies: *query*, *sequence*, *sub query*, and *parent query* rules. In the following, we describe each rule type in detail and provide examples.

4.2.1 Query Rule. Query rules are the most basic type of labeling rule template. Their purpose is to match collected facts with specific parts of log events and assign labels to all retrieved logs. Accordingly, this type of rule is only applicable when all relevant logs are known to match the respective fact in the selected field, and no logs generated by benign behavior yield matches in that field. For example, in the simple case where all activities associated with a malicious domain should be labeled as part of an attack, it is possible to label a log file that monitors DNS connections by matching the domain name occurring in the logs with the attacker's domain name that was previously extracted as a fact [1].

Figure 2 shows an exemplary query rule for this case. The rule specifies that all logs in the *dnsmasq-inet-firewall* index matching the malicious domain referenced by variable *attacker.dnsteal.domain* in the fields *dns.answers.name* or *dns.question.name* are assigned the labels *dnsteal*. Note that variables such as *dns.answers.name* are resolved by the respective field of the parsed logs in the database where the query is executed, while *attacker.dnsteal.domain* is a templated variable as indicated by the curly braces and thus replaced by the respective fact when the rule is rendered from the template. Thereby, the domain name is a random string extracted

```
- type: elasticsearch.query
  id: dnsteal.domain.match
  labels: [dnsteal]
  index: [dnsmasq-inet-firewall]
  query:
    bool:
      should:
        - regexp:
            dns.answers.name: '.*\.{{ attacker.dnsteal.
                 domain | replace('.', '\.') }}'
        - regexp:
            dns.question.name: '.*\.{{ attacker.dnsteal.
                 domain | replace('.', '\.') }}'
```

Figure 2: Query labeling rule that matches domain names.

```
- type: elasticsearch.sequence
  id: attacker.foothold.apache.access
  labels: [attacker_http, foothold]
  index: [pcap-attacker_0, apache_access-intranet_server]
  by: url.full
  max_span: 3m
  filter:
    - range:
        "@timestamp":
          gte: "{{ ( foothold.start | as_datetime) }}"
          lte: "{{ ( foothold.stop | as_datetime) +
               timedelta(seconds=1) }}"
  sequences:
    - '[ apache where event.action == "access" and source
        .address == "{{ attacker.vpn_ipv4_address }}" ]'
    - '[ http where source.ip == "{{ servers.
        intranet_server.default_ipv4_address }}" and
        layers.http.http_http_response == true ]'
```

Figure 3: Labeling rule of sequence type with filtering.

from the TSM for attacker behavior. As visible in the rule, queries may be connected with boolean operators, e.g., the keyword *should* represents a logical OR operation. In addition, it is possible to apply functions on the terms, e.g., we use a replace function in the sample rule to escape dots and enable matching with regular expressions.

4.2.2 Sequence Rule. Some attack artifacts in log data cannot be labeled with query rules, but require a more advanced strategy. In particular, this concerns logs that can only be identified as part of the attack by their collective occurrence, while each of the events individually is indiscernible from logs related to benign behavior. We therefore propose sequence rules to model such cases.

Figure 3 shows a sample sequence rule that labels two consecutively generated log events from two different sources, packet capture (PCAP) stored in index *pcap-attacker_0* and Apache access logs stored in index *apache_access-intranet_server*. We use the *by* parameter to obtain groups of logs with the same value in the *url.full* field and set the maximum time span in which these logs have to occur through parameter *max_span* to 3 minutes. The sequence itself is specified through a list of queries, where each query matches log event fields to facts or predefined values, e.g., *source.address* of Apache access logs matches the attacker's VPN IP extracted from the infrastructure TSM. While this sample demonstrates labeling of events occurring across different files, we point out that this rule type is also highly useful to label log sequences of arbitrary lengths that occur in the same file but are interleaved with benign logs.

The rule also involves a filter that limits the number of logs on which the query is executed based on their occurrence times. In the sample rule, this time range spans between start time *foothold.start* and stop time *foothold.stop* of the respective attack phase. Note that 1 second is added to the stop time to avoid incorrect label assignment due to rounding of log timestamps without sub-second precision. Filtering improves runtime performance since fewer comparisons are carried out and further decreases the probability of incorrect label assignment, e.g., benign events that match the query but are generated outside of the interval are not labeled. It is also possible to filter logs based on value matches or already assigned labels.

```
- type: elasticsearch.sub_query
  id: attacker.foothold.apache.access_dropped
  labels: [attacker_http, foothold]
  index: [pcap-attacker_0]
  query:
    - term:
        destination.ip: "{{ servers.intranet_server.
             default_ipv4_address }}"
  sub_query:
    index: [apache_access-intranet_server]
    query:
      - term:
          url.full: "{{ HIT.url.full }}"
      - term:
          source.address: "{{ attacker.vpn_ipv4_address
               }}"
```

Figure 4: Labeling rule of sub query type.

4.2.3 Sub Query Rule. Some logs do not contain all fields required to assign labels to them. Instead, it is necessary to link them to other events to determine whether they correspond to attacker behavior or not. Therefore, we propose sub query rules that involve a two-stage query mechanism: First, a main query is executed to retrieve a set of events. Then, each of these events is used in another query that allows matching based on the fields of the main query result in addition to the usual matching based on facts.

Consider the example in Fig. 4 which aims to label attacker requests in Apache access logs that cannot be labeled by the corresponding responses through the sequence rule from Fig. 3, e.g., because the response is not sent or lost. The main query retrieves all log events with destination IP addresses matching the IP of the intranet server from PCAP logs in index *pcap-attacker_0*. The sub query then iterates over each of these events and labels all Apache access logs from index *apache_access-intranet_server* that have the attacker's VPN IP in field *source.address* and the same *url.full* as the PCAP event that is accessible through variable *HIT*.

We point out that sub queries have the disadvantage of a long runtime since a new query needs to be executed for each result of the main query. Accordingly, it is usually necessary to include time- and attribute-based filters in the main query to keep the number of retrieved logs to a manageable size. We omit these filters in the sample for brevity and refer to our open-source implementation.

4.2.4 Parent Query Rule. Parent query rules function similar to sub query rules, i.e., they comprise a main query and execute another nested query for each of the retrieved lines. However, while sub query rules label the results of the nested query, the purpose of parent query rules is to assign labels to each retrieved log of the main query if the nested parent query yields at least n results, where $n = 1$ by default. This rule type is useful to assign labels to log events that were missed by earlier executed rules, e.g., timeout events that occur some time after the stop time of the respective attack. In this case, the main query selects all logs within an extended time interval (i.e., a sufficiently long delta is added to the stop time in

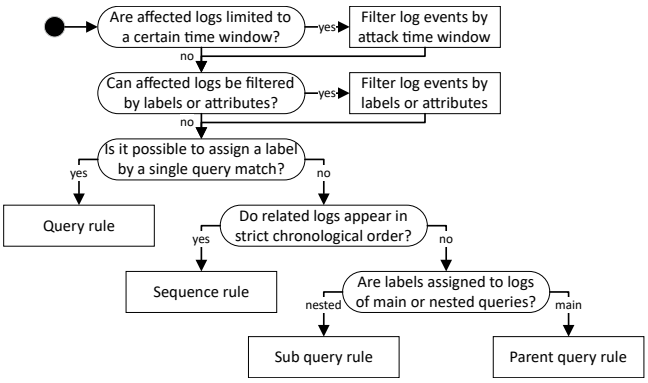


Figure 5: Procedure for labeling rule type selection.

the filter) that do not have a specific label assigned, and the parent query then iterates over all results and queries for related events with matching attributes. In case that at least one related event is retrieved in the parent query, a label is assigned to the respective log from the main query. Parent query rules have similar disadvantages with respect to the runtime as sub query rules and are also similarly structured. We therefore omit an example for brevity.

4.2.5 Rule Selection. The previous sections outlined four types of labeling rules. Thereby, each type offers specific functionalities for labeling logs in certain situations where common labeling strategies cannot be applied. To ease the selection of appropriate rule types depending on the log data at hand, we outline a procedure that maps properties of attack artifacts to the available types.

Figure 5 depicts this procedure as a flow chart. Whenever it is possible to limit the queried logs to a certain time interval, e.g., the start and stop times of attacks, we recommend to add time-based filters. Second, the presence of certain attributes or labels assigned by previously executed rules allow to further reduce the number of logs. Query rules assign labels to logs retrieved by single queries. In case that attacks reflect in chronological sequences or correlating events across files, sequence rules should be applied. Otherwise, sub query rules can be used to label logs retrieved by nested queries and parent query rules can be used to label logs retrieved by main queries that also fulfill constraints from nested queries.

4.3 Implementation

This section summarizes our implementation decisions of aforementioned concepts. We realize testbed deployment as well as data collection with Ansible[4] roles. All processors of the labeling layer are implemented as scripts. We use open-source Logstash [24] parsers that are available for a large number of common log formats and integrate well with Elasticsearch [24], which we use as a database for log storage. The main advantage of Elasticsearch for our approach is that it is designed for carrying out complex queries on semi-structured data efficiently. We define our rule types in YAML syntax based on the Elasticsearch query language and use the Event Query Language[5] for sequence rules. Finally, we generate rules of these types as templates using the Jinja[6] templating engine.

[4]https://www.ansible.com/
[5]https://www.elastic.co/guide/en/elasticsearch/reference/current/eql.html
[6]https://jinja.palletsprojects.com/

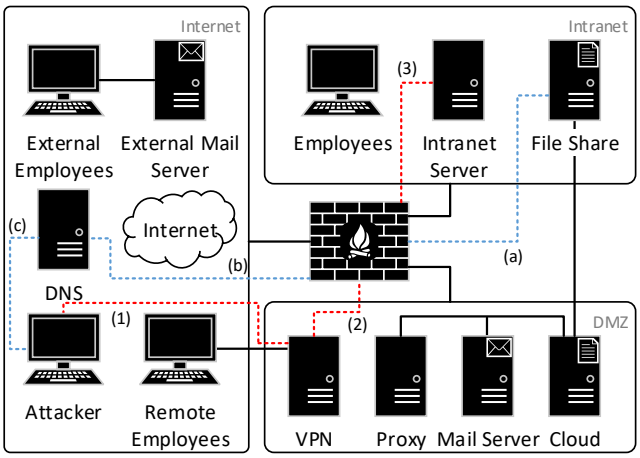


Figure 6: Network structure of the deployed testbed. Connections (1)-(3) correspond to the server takeover attack and connections (a)-(c) correspond to the data exfiltration attack.

5 EVALUATION

This section outlines the evaluation of our approach. First, we describe an illustrative scenario for the Kyoushi framework and provide an overview of the generated labels. We then present the results of an expert survey that was carried out to validate the correctness of the assigned labels.

5.1 Illustrative Scenario

We set up a testbed following the model-driven testbed generation approach outlined in Sect. 3.1. Figure 6 depicts an overview of the testbed network, which is designed to resemble the network infrastructure of a small or medium sized enterprise. The testbed simulates employees accessing file shares, sending and receiving mails, reading and editing WordPress articles, browsing the web, and executing commands over SSH. Thereby, the behavior of seven employees within the Intranet and three employees remotely connected through a VPN tunnel is based on state machines with transition probabilities derived from user profiles. In addition, we deploy an attacker machine that generates malicious activity as part of a *server takeover* attack and a *data exfiltration* attack, which are depicted in Fig. 6 as (1)-(3) and (a)-(c) respectively.

The server takeover attack assumes that the attacker obtained VPN credentials from a compromised computer and involves a multi-step attack that takes place over two of the five days of simulation. On the first day, the attacker attempts to gain more information about the network during the so-called *foothold* phase. This involves running the network scanner Nmap[7] to discover possible targets, execution of WPScan[8] and dirb[9] to find vulnerabilities of the WordPress instance running on the intranet server, uploading a webshell by exploiting a vulnerability of the wpDiscuz plugin (CVE-2020-24186), reading out the WordPress configuration to obtain the database password, and dumping the database of usernames and password hashes. For the second day, we assume that the attacker

[7]https://nmap.org/
[8]https://wpscan.com/wordpress-security-scanner
[9]https://tools.kali.org/web-applications/dirb

Table 1: Overview of the number of rule templates and number of labeled lines per file for each log data label.

Label	Rule templates				vpn	intranet					firewall	share	Total
	Query	Seq.	Sub	Parent	openvpn.log (4083)	wp-access.log (417516)	wp-error.log (1272)	error.log (13)	auth.log (777)	audit.log (2824)	dnsmasq.log (158360)	audit.log (760)	
dnsteal	2										43302	1	43303
dnsteal-received	1										36536		36536
dnsteal-dropped	1			1							6766		6766
exfiltration-service	1											1	1
foothold	1	5	2	2	308	406402	1268						407978
attacker_http		4	2	2		406402	1268						407670
dirb	1					406392	1268						407660
webshell_upload		1				3							3
webshell_cmd	1					7							7
escalate	2	7			28			1	12	19			60
reverse_shell	1							1					1
attacker_change_user		2							5	9			14
escalated_command		4							8	10			18
escalated_sudo_session		1							6				6
attacker_vpn	2	2			336								336
Total	9	13	2	3	336	406402	1268	1	12	19	43302	1	451341

was able to crack one of the retrieved passwords of a system user. The attack thus continues with the attacker deploying a reverse shell through the webshell and misusing the compromised account to gain elevated privileges. This so-called *escalate* phase ends with the attacker executing some commands, e.g., inspecting SSH keys.

The second attack scenario is centered on the exfiltration of sensitive data from the internal file share. Other than the first attack case, we assume that the exfiltration is already ongoing at the beginning of the simulation and stops after two days. The exfiltration script scans the file share and encodes all found files in base64 before sending it through the DNS server to the attacker, where the exfiltration tool DNSteal[10] receives the data and reconstructs the files. Note that we designed this test case as a challenge for anomaly-based IDSs, since it is more difficult to detect the absence of activities rather than the appearance of new events.

We selected eight log files that we considered relevant for evaluating our labeling framework, since they contain traces of the attacks and are files typically monitored by IDSs. These sources include VPN logs, access logs, authentication logs, audit logs, and more. To design our labeling rule templates, we disabled user simulations and executed the attacks in an idle network to observe their consequences. In total, 27 rule templates were developed to assign 15 distinct labels to events from all selected files. In the following, we present the resulting labels in detail.

5.2 Results

We ran the simulation as described in the previous section and collected logs and facts from all machines. We then executed the labeling procedure with the rule templates that we modeled as part of the attack TIMs. We gathered the generated labels for each log file and analyzed their frequencies and relationships. In the following, we describe the results of our analysis.

The left side of Table 1 provides an overview of the numbers and types of rule templates and their corresponding labels. For example, the template rule shown in Fig. 2 is a query rule that assigns label *dnsteal* and thus contributes to the respective counter in column *Query*. Note that it is possible that a single rule assigns multiple labels and that different rules assign the same label. For this reason, the total number of rules displayed in the last row

[10]https://github.com/m57/dnsteal

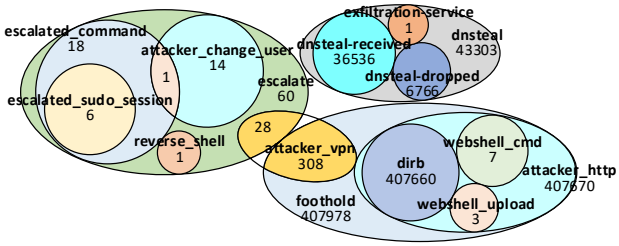


Figure 7: Euler diagram depicting label relationships.

does not represent the sum of all labels assigned per rule type, but instead the total number of rules independent from the number of labels they assign. For example, we designed two sub query rule templates and both of them assign labels *foothold* and *attacker_http* (cf. Fig. 4), thus both counters and the total show 2. This breakdown shows that our scenario mostly requires query and sequence rules and comparatively few sub query and parent query rules.

The right side of Table 1 depicts the number of labeled log events in each file, where the top header row states the host at which the file was collected, and the bottom header row states the specific file and its number of lines in brackets. Similar to rules, lines can be assigned multiple labels and thus the bottom row depicts the total number of labeled events per file. We state the total number of assigned labels across all files on the right hand side of the table. The results indicate that the numbers of labeled events differ greatly depending on the attack and considered file, i.e., most labels are either related to the dirb attack that mainly affects the "intranet/wp-access.log" file or the DNSteal attack that mainly affects the "firewall/dnsmasq.log" file. The table also shows that most attacks manifest in specific files, except for the "vpn/openvpn.log" file that contains traces of both *foothold* and *escalate* phase of the first attack scenario.

This relationship is also depicted in the Euler diagram of Fig. 7, where *attacker_vpn* stretches across *foothold* and *escalate*. Moreover, this visualization makes it easy to see the hierarchical structure of labels, e.g., the dirb attack involves HTTP traffic and thus logs labeled *dirb* are a subset of logs labeled *attacker_http*.

5.3 Expert Survey

The previous section provided an overview of the labels generated by the Kyoushi framework. We evaluate our approach by validating

the correctness of these labels. For this, we set up a survey and ask security engineers and IT analysts to review the logs and labels.

Our dataset involves a total of 451, 341 labels, which makes verification by humans impossible without sampling the data. We restrict the evaluation to labels *dirb*, *webshell_upload*, *webshell_cmd*, *reverse_shell*, *attacker_change_user*, *escalated_command*, *dnsteal*, *exfiltration-service*, and *dnsteal* to avoid labels of supersets so that users are presented with the most specific label. We then randomly select events corresponding to these labels and present them to survey participants. Since the context of occurrence is helpful and sometimes essential for the correct interpretation of logs, we display the events with four immediately succeeding and preceding log lines. Note that the experts also have access to the whole dataset without labels as well as details on the launched attacks.

Figure 8 shows a sample question from the survey. The participant is informed that the *dirb* label was assigned to the marked Apache access log line and tasked to determine whether this label is correct or not. Participants may select their opinion on a seven-point scale ranging from strong disagreement to strong agreement and including "No answer" as a neutral option. The log sample in the figure depicts several requests made in a short time interval (all lines have the same timestamp) and in seemingly alphabetical order, where all requested pages start with the letters "em". A participant with sufficient knowledge about the attack could therefore conclude that these lines are likely artifacts of a dictionary scan, and thus agree with the assigned label *dirb*. We decided for such a quantitative evaluation over expert reviews of our developed rules for two reasons: First, we aim to ensure an objective evaluation with adequate efforts. Second, we base the evaluation on the generated labels rather than the rules to focus on the actual output of our approach and recognize any incorrectly labeled events.

In addition to labels for attacker behavior, we also sampled unlabeled lines and asked participants to determine whether these lines actually correspond to benign behavior. Moreover, control questions ensure that participants do not just agree to all questions, but are actually able to differentiate malicious from benign behavior. We therefore add questions with purposefully incorrect labeled logs, i.e., events that received an attack label from the Kyoushi framework but are displayed with label *normal*, and benign events that are presented with a randomly selected attack label from the same file. This setup allows us to identify and possibly exclude participants who select random answers or do not have the technical skills required to interpret the logs. However, we point out that the purpose of this survey is not to rate the ability of participants to recognize attacks in log data, but instead to determine whether manually assigned labels based on expert knowledge diverge from labels generated by our automatic procedure. Thereby, the survey format aims to discover incorrect rules rather than missing rules, since it is unlikely that logs without labels that are actually part of an attack are selected during sampling of benign events.

We hosted the survey online and asked engineers with security expertise for anonymous participation. In particular, we contacted cyber security experts, penetration testers, and capture-the-flag contestants and invited them to share the link to the survey among their peers. In the beginning of the survey, we asked participants about their roles. Then, the same questions were displayed in random order to each participant. After one week, we obtained responses from 16 participants, out of which 8 skipped more than 25% of all questions and were thus excluded. The remaining 8 participants skipped less than 2% of all questions on average, indicating their high confidence in filling out the survey. The majority of these participants (5) identify their roles as security analysts, 2 as penetration testers, and 1 as a cyber security research engineer. In the following, we analyze the answers of these participants.

Figure 9 shows the answers to questions with correct labels as a boxplot, where the labels and log files associated with the respective question are displayed on the horizontal axis and each point represents an answer. For example, the answers to the question from Fig. 8 are displayed on the left hand side of the plot with label *dirb (intranet/wp-access.log)* and show that 7 out of 8 participants agree or strongly agree with the assigned label. Note that "No answer" responses are excluded so that distributions are not distorted.

Overall, the plot shows a clear trend towards agreement for labeled attacks, with few exceptions. One outlier is the label *reverse_shell* in file *intranet/error.log*, which refers to a log event with message "Bad file descriptor". We argue that the missing timestamp and rather general error message made it difficult for participants to relate the event to the attack. Moreover, label *attacker_change_user* in file *intranet/audit.log* received mixed answers. This is likely due to the facts that audit logs are more difficult to interpret for inexperienced analysts and that in this case relevant attack indicators occur in the preceding lines, not in the marked line itself.

Interestingly, logs correctly labeled as normal have slightly lower agreement scores on average than logs corresponding to attacks. We argue that this is due to the fact that analysts have higher confidence in their answer when they recognize indicators for specific attacks that fit the proposed label, while the lack of such indicators is not sufficient to deem normal logs as such with high certainty.

Figure 10 shows the survey results with respect to incorrectly labeled lines. Note that the horizontal axis shows the actual labels assigned to the logs, not the randomly selected incorrect ones. The overall trend towards disagreement indicates that participants were able to recognize incorrect labels. Similar to the previous results from the correctly labeled lines, there appears to be less consensus among participants when verifying labels of normal behavior. One extreme example is the question labeled *normal (intranet/wp-error.log)*, where logs are incorrectly labeled as *dirb* and involve "no such file or directory" warnings that are unrelated to the attacks. Since such events likely occur during scans, it is understandable that some participants were drawn towards agreement.

6 DISCUSSION

The evaluation results indicate that the Kyoushi framework is useful for log data labeling. In the following, we discuss the fulfillment of the requirements stated in Sect. 4.1 and point out some limitations.

Label correctness obviously depends on manually designed rules. However, we argue that our intuitive rule format makes it easy to comprehend the consequences of rules and the fact that only few rules are required to label even a complex scenario reduces the chances of errors. The requirement on automation is fulfilled as labeling rules only have to be defined once to be applied multiple times. This enables application within model-driven testbeds and thus facilitates generation of diverse log datasets on a large

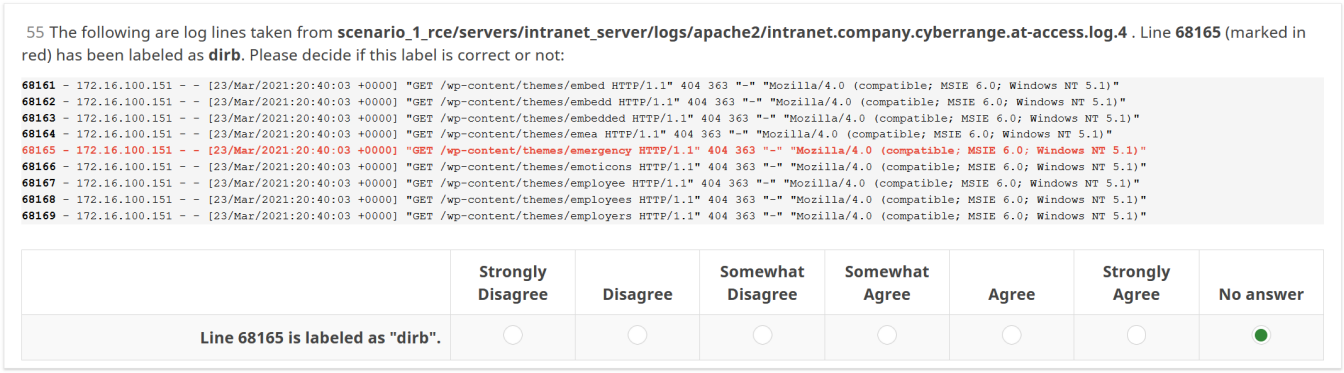


Figure 8: Survey question asking the participant to decide if the marked line is correctly labeled as part of the dirb attack step.

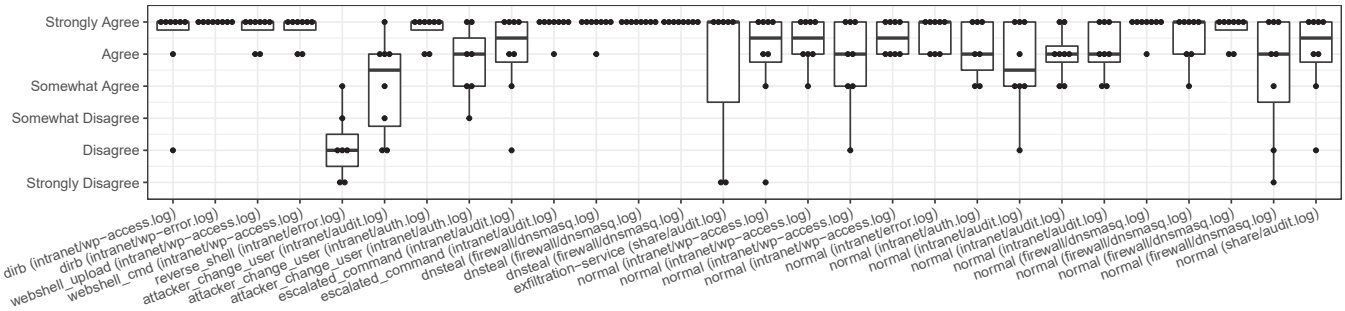


Figure 9: Boxplots of survey answers to correctly labeled lines show that participants mostly agreed with assigned labels.

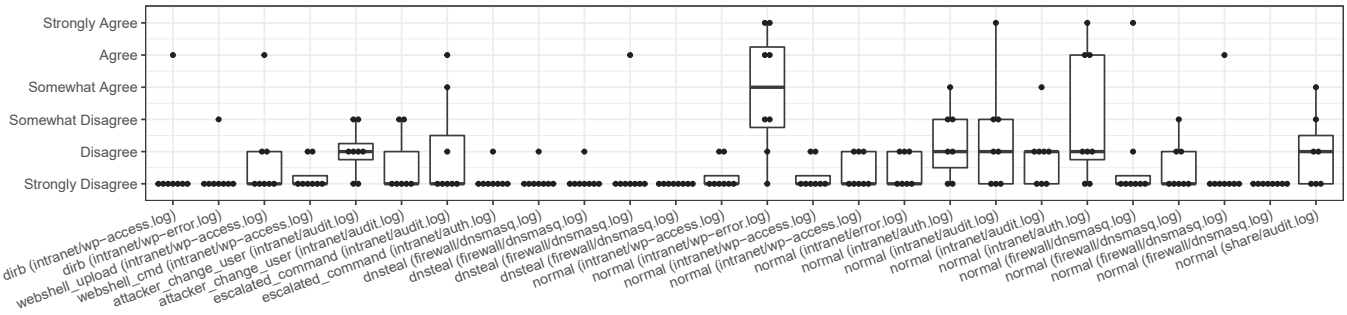


Figure 10: Boxplots of survey answers to incorrectly labeled lines show that participants recognized consciously placed errors.

scale with low manual effort [19]. Our labeling rules also fulfill the requirement on adaptability as they are dynamically filled out with testbed facts. Regarding the requirement on granularity, our methodology enables label assignment to individual events. Each rule may assign arbitrary many labels and each log event may obtain multiple distinct labels. As a consequence, our labels have hierarchical relationships (cf. Sect. 5.2) that enable evaluating HIDSs separately for different attack steps. Our labeling framework is easy to apply on many common log formats since it relies on Logstash and our rules are capable of matching diverse attack artifacts. Finally, our framework is open-source and thus all results are reproducible.

Our labeling approach is primarily designed for model-driven testbeds, but also suitable for more general use cases. In particular, labeling rules as may be applied to pre-existing log datasets without labels or logs generated in traditional testbeds as long as log parsers are available and the facts used by the labeling rules are manually filled out. We argue that it is usually possible to gather the required facts with tolerable manual effort either from expert knowledge about the testbed at hand, documentation accompanying the log dataset, or by forensically analyzing the logs.

Moreover, we presented our approach in an offline setting, i.e., labeling is carried out after all logs are available. However, our framework is also capable of assigning labels online, i.e., simultaneous to log data generation. This could be interesting for live demonstrations and experiments rather than dataset generation.

While this work mainly focuses on labeling attack artifacts, our concepts could also be applied for labeling all kinds of activities. For example, labels could be assigned to benign activities carried out by different users. This could be beneficial to evaluate user profile mining algorithms that analyze log data [16].

As pointed out in Sect. 5.1, our recommended approach for designing the labeling rules is to launch the attacks in an idle network

(i.e., without running user simulations) and observe their manifestations in log data. The main issue with this strategy is that testbeds are generally non-deterministic and thus random or otherwise unexpected events that may occur in some testbed instances are not correctly labeled in case that they were not observed before. There is no simple way to fully resolve this problem, however, we point out that our framework makes it easy to manage, adjust, and repeatedly execute labeling rules in hindsight after all logs are gathered. In case that any missing or incorrect labels are recognized when analyzing the resulting data, it is therefore simple to change or update the set of rules and regenerate all labels.

Our framework does not address the problem of labeling missing events, i.e., events that fail to occur as consequences of attacks. We argue that evaluation based on attack times is sufficient in such cases and do not expect such artifacts in our scenario. Furthermore, we considered to implement label weights, i.e., numeric values that specify the strength of association between events and labels. This could solve the problem of sometimes fuzzy boundaries between benign and malicious behavior, e.g., labels for events generated by the attacker but not directly related to exploits could receive a low weight. We leave this task for future work.

7 CONCLUSION

This paper presents the Kyoushi framework, which integrates labeling rules into model-driven testbed generation approaches. This enables to generate arbitrary numbers of labeled datasets for HIDS evaluation in diverse environments. The main idea is to design rules as templates that are reusable across testbeds and thus eliminate the need for manual adjustments. We propose four types of rules (query, sequence, sub query, and parent query) to describe matching criteria for system logs that occur in groups and are interrelated with other events across log files. We apply our framework in an illustrative scenario and show that only few rules and manageable manual effort are necessary to label several diverse attacks that leave artifacts in different files. The expert survey carried out as part of the evaluation indicates the correctness of our labels. For future work, we plan to gather and compare datasets from several automatically generated testbeds with variations. In addition, we consider to use gamification or capture-the-flag contests rather than expert surveys to attract a larger audience for label verification.

ACKNOWLEDGMENTS

Parts of this work were carried out in course of a Master's thesis at the Vienna University of Technology [6]. This work was partly funded by the FFG projects INDICAETING (868306) and DECEPT (873980), and the EU H2020 project GUARD (833456).

REFERENCES

[1] Jawad Ahmed, Hassan Habibi Gharakheili, Qasim Raza, Craig Russell, and Vijay Sivaraman. 2019. Monitoring enterprise DNS queries for detecting data exfiltration from internal hosts. *IEEE Transactions on Network and Service Management* 17, 1 (2019), 265–279.

[2] Hamad Almohannadi, Irfan Awan, Jassim Al Hamar, Andrea Cullen, Jules Pagan Disso, and Lorna Armitage. 2018. Cyber threat intelligence from honeypot data using elasticsearch. In *International Conference on Advanced Information Networking and Applications*. IEEE, 900–906.

[3] Molly Buchanan, Jeffrey W Collyer, Jack W Davidson, Saikat Dey, Mark Gardner, Jason D Hiser, Jeffry Lang, Alastair Nottingham, and Alina Oprea. 2021. On Generating and Labeling Network Traffic with Realistic, Self-Propagating Malware. *arXiv:2104.10034* (2021).

[4] Dainius Čeponis and Nikolaj Goranin. 2018. Towards a robust method of dataset generation of malicious activity for anomaly-based HIDS training and presentation of AWSCTD dataset. *Baltic Journal of Modern Computing* 6, 3 (2018), 217–234.

[5] Gideon Creech and Jiankun Hu. 2013. Generation of a new IDS test dataset: Time to retire the KDD collection. In *Wireless Communications and Networking Conference*. IEEE, 4487–4492.

[6] Maximilian Frank. 2021. Quality improvement of labels for model-driven benchmark data generation for intrusion detection systems. Master's Thesis.

[7] Fermin Galan, David Fernandez, Jorge E López De Vergara, and Ramon Casellas. 2010. Using a model-driven architecture for technology-independent scenario configuration in networking testbeds. *IEEE Communications Magazine* 48, 12 (2010), 132–141.

[8] Peng Gao, Fei Shao, Xiaoyuan Liu, Xusheng Xiao, Zheng Qin, Fengyuan Xu, Prateek Mittal, Sanjeev R Kulkarni, and Dawn Song. 2020. Enabling Efficient Cyber Threat Hunting With Cyber Threat Intelligence. *arXiv:2010.13637* (2020).

[9] Sebastian Garcia, Martin Grill, Jan Stiborek, and Alejandro Zunino. 2014. An empirical comparison of botnet detection methods. *Computers & Security* 45 (2014), 100–123.

[10] Amirhossein Gharib, Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. 2016. An evaluation framework for intrusion detection dataset. In *International Conference on Information Science and Security*. IEEE, 1–6.

[11] Martin Grimmer, Martin Max Röhling, D Kreusel, and Simon Ganz. 2019. A modern and sophisticated host based intrusion detection data set. *IT-Sicherheit als Voraussetzung für eine erfolgreiche Digitalisierung* (2019), 135–145.

[12] Shilin He, Jieming Zhu, Pinjia He, and Michael R Lyu. 2020. Loghub: a large collection of system log datasets towards automated log analytics. *arXiv:2008.06448* (2020).

[13] Shaohan Huang, Yi Liu, Carol Fung, Rong He, Yining Zhao, Hailong Yang, and Zhongzhi Luan. 2020. Transfer Log-based Anomaly Detection with Pseudo Labels. In *International Conference on Network and Service Management*. IEEE, 1–5.

[14] Ansam Khraisat, Iqbal Gondal, Peter Vamplew, and Joarder Kamruzzaman. 2019. Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity* 2, 1 (2019), 1–22.

[15] Max Landauer, Florian Skopik, Markus Wurzenberger, Wolfgang Hotwagner, and Andreas Rauber. 2021. Have it Your Way: Generating Customized Log Datasets With a Model-Driven Simulation Testbed. *IEEE Transactions on Reliability* 70, 1 (2021), 402–415.

[16] Haibin Liu and Vlado Kešelj. 2007. Combined mining of Web server logs and web contents for classifying user navigation patterns and predicting users' future requests. *Data & Knowledge Engineering* 61, 2 (2007), 304–330.

[17] Gabriel Maciá-Fernández, José Camacho, Roberto Magán-Carrión, Pedro García-Teodoro, and Roberto Therón. 2018. UGR '16: A new dataset for the evaluation of cyclostationarity-based network IDSs. *Computers & Security* 73 (2018), 411–424.

[18] Abdelhamid Makiou and Ahmed Serhrouchni. 2015. Efficient training data extraction framework for intrusion detection systems. In *International Conference on the Network of the Future*. IEEE, 1–4.

[19] Petteri Nevavuori and Tero Kokkonen. 2019. Requirements for training and evaluation dataset of network and host intrusion detection system. In *World Conference on Information Systems and Technologies*. Springer, 534–546.

[20] Adam Oliner and Jon Stearley. 2007. What supercomputers say: A study of five system logs. In *International Conference on Dependable Systems and Networks*. IEEE, 575–584.

[21] Markus Ring, Sarah Wunderlich, Dominik Grüdl, Dieter Landes, and Andreas Hotho. 2017. Flow-based benchmark data sets for intrusion detection. In *16th European Conference on Cyber Warfare and Security*. 361–369.

[22] Markus Ring, Sarah Wunderlich, Deniz Scheuring, Dieter Landes, and Andreas Hotho. 2019. A survey of network-based intrusion detection data sets. *Computers & Security* 86 (2019), 147–167.

[23] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. 2018. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *International Conference on Information Systems Security and Privacy*. 108–116.

[24] Moza Shibani and Anupriya E. 2019. Automated Threat Hunting Using ELK Stack - A Case Study. *Indian Journal of Computer Science and Engineering* 10 (2019), 118–127.

[25] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A Ghorbani. 2012. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers & Security* 31, 3 (2012), 357–374.

[26] Florian Skopik, Giuseppe Settanni, Roman Fiedler, and Ivo Friedberg. 2014. Semi-synthetic data set generation for security software evaluation. In *International Conference on Privacy, Security and Trust*. IEEE, 156–163.

[27] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R Lyu. 2019. Tools and benchmarks for automated log parsing. In *International Conference on Software Engineering*. IEEE, 121–130.