# Triage of IoT Attacks through Process Mining

Simone Coltellese[1], Fabrizio Maria Maggi[2], Andrea Marrella[1], Luca Massarelli[1], and Leonardo Querzoni[1]

[1] DIAG, Sapienza Universitá di Roma, Rome, Italy
coltellese.1534700@studenti.uniroma1.it,
{marrella,massarelli,querzoni}@diag.uniroma1.it
[2] University of Tartu, Estonia
f.m.maggi@ut.ee

**Abstract.** The impressive growth of the IoT we witnessed in the recent years came together with a surge in cyber attacks that target it. Factories adhering to digital transformation programs are quickly adopting the IoT paradigm and are thus increasingly exposed to a large number of cyber threats that need to be detected, analyzed and appropriately mitigated. In this scenario, a common approach that is used in large organizations is to setup an *attack triage* system where security operators, supported by appropriate tools, can cherrypick new attack patterns, which require further in-depth investigation, from a mass of known attacks, which can be managed by automatic means. In this paper, we propose an attack triage system that helps operators quickly identify attacks with unknown behaviors, and later analyze them in detail. The novelty introduced by our solution is in the usage of process mining techniques to model known attacks and identify new variants. We demonstrate the feasibility of our approach through a twofold evaluation based on three well-known IoT botnets, *BASHLITE*, *LIGHTAIDRA* and *MIRAI*, and on real current attack patterns collected through an IoT honeypot.

**Keywords:** IoT Security · Process Mining · Behavioral Attack Analysis.

## 1 Introduction

The Internet of Things (IoT) is supposed to revolutionize, in the forthcoming years, the way we interact with the physical world. Nowadays, this interaction mainly happens through smartphones and connected gadgets, but, in the soon-to-come future, people will heavily rely on automated vehicles, wearable medical devices, and other connected items to avoid potentially harmful incidents.

In this scenario, cybercriminals are starting to grasp the opportunities of a new era where an impressively large number of connected devices can be exploited for criminal activities. Even if this phenomenon is still in its infancy, we already experienced the first glimpses of a glooming future: between 2016 and 2018, a large botnet, named *MIRAI*[12], was used to launch some of the most intense distributed denial of service (DDoS) attacks ever recorded, topping at

more than 1Tbps [2]. The source code of the botnet was openly released to the public by the end of 2017, paving the way for new breeds of the same threat.

Three factors mainly justify the growing alarms surrounding the security of forthcoming IoT solutions:

- so far, most producers and system integrators have not paid enough attention to security issues in IoT devices. Most of them are designed to be sold to the masses at the lowest possible price, and consumers are still hardly willing to spend more to pay for security features against advanced functionalities;
- IoT devices are often built on dedicated HW/SW platforms; this results in a large heterogeneity of platforms to be defended, with a growing number of potentially exploitable vulnerabilities that are hardly patched by producers;
- IoT systems are growing in size and complexity, with boundaries that are sometimes difficult to define precisely, so that it becomes complex to identify their exposed attack surface.

A common approach used by large organizations to protect complex systems is to setup an internal structure (e.g., a Security Operations Center) to manage cyber incidents. Incident response processes are typically based on the acquisition of data from probes and sensors (firewalls, intrusion detection systems, AVs, etc.) that is then analyzed by security operators in order to characterize ongoing attacks. As the number of cybersecurity incidents increases, this approach needs to be supported by a filtering phase that quickly discards cases representing known attack patterns (for which remediation plans are already known and in-place) thus allowing security operators to concentrate their efforts on new attack patterns. This phase is known as *triage*[1] and its output is a prioritized list of attacks to be analyzed, where higher priority is assigned to attacks that do not resemble known patterns.

In this paper, we introduce a novel solution to support security operators during the triage of attacks that target IoT systems. Our solution leverages state-of-the-art process mining techniques [4] to recognize known attack patterns and identify new variants, providing the operators with information on the differentiating details. Process mining stands for techniques to analyze business process models and their execution traces (logs). It provides methods for reconstructing process models from logs (*process discovery*), checking the conformance of an existing or reconstructed model and logs (*conformance checking*), and enhancing process models based on the results of analysis (*process enhancement*).

Specifically our proposed approach analyzes logs of commands issued by a botnet against a IoT device during the fingerprinting phase[2], and discovers a process model representing an up-to-date picture of the recorded attacks. New observed traces are prioritized by aligning them to the model and then calculating their *fitness* score: the smaller the score, the more the log contains actions

[1] The name comes from the process used by ER-units in hospitals to quickly prioritize incoming patients depending on the severity of their health status.

[2] In this phase the botnet issues commands on the shell of a device found on the internet to identify its architecture, before deploying the appropriate attack payload. See Section 2 for further details.

that were not observed in previous attacks, and therefore the larger is the priority for the operator. The main finding of our research is that process mining techniques provide a powerful tool for the automated discovery of new IoT attacks and of their "anatomy". This allows our solution to provide detailed feedbacks of such unknown malevolent behaviors to support security operators in their identification and classification as new attack patterns. Differently from other solutions, commonly based on statistical models trained with machine learning algorithms, this system allows us to inspect the model, extract human-readable information from it, and use this information to notify the security operator about how a new attack differs from previous observations.

To demonstrate the feasibility of our approach, we evaluated our solution on attacks generated by three well-known IoT botnets, namely *MIRAI*, *LIGHTAIDRA* and *BASHLITE*. The evaluation demonstrates that our solution is able to build a general attack model that correctly represents the various behaviors characterizing these botnets, identify their attack patterns and gracefully evolve as new variants are observed. Furthermore, we propose an experiment based on current attack patterns collected through an IoT honeypot. It shows how our system correctly identifies new botnets, and how it supports security operators by providing precise information on the behavior of new attacks.

The rest of the paper is organized as follows. Section 2 provides a background on the security of IoT systems, with a specific focus on botnets. Section 3 introduces the process mining techniques used to realize our approach. Section 4 describes our system, while Section 5 discusses its evaluation. Section 6 presents the related works and, finally, Section 7 concludes the paper.

## 2 Background on IoT (In-)Security

Even if the success of IoT is today a reality, the security of IoT devices remains a big challenge. In 2014, the OWASP Foundation [33] published a list of the top ten most dangerous vulnerabilities in IoT. At the first place, they listed insecure web interfaces that often permit to an attacker to login into a device using weak credentials or to capture plain-text password. Other sources of danger come from the insecurity of the network services exposed by devices and the lack of transport encryption. As evidenced by Hossain et al. [23], these security issues are sometimes due to HW (e.g., limited memory, constrained energy consumption, etc.) and SW (e.g., poor testing, unavailable security updates, etc.) constraints that characterize IoT devices.

Given the roles played by IoT devices, oftentimes the risks involved by the presence of such vulnerabilities cannot be easily mitigated. Indeed, these devices are often deployed in places where the most common form of connection is an internet link, which exposes their attack surfaces to remote threats. As a consequence, user's privacy is at high risk and operational safety cannot be guaranteed. This scenario becomes even more worrying if we consider that an attacker can leverage the vulnerabilities of each device to enroll it into a malicious network of connected devices, namely a *botnet*, which respond to her commands.

### 2.1 IoT Botnets

IoT botnets are networks of infected devices (*bots*) which perform malicious actions issued as commands from a command and control (C2) server controlled by an attacker. Bertino el al. [15] discussed the several potential usages of a botnet: distributed denial of services (DDoS) attacks, crypto-mining, password cracking, email spam and key logging. One of the most famous IoT botnets was *MIRAI*, which infected several hundreds of thousands devices [12] all over the world and was successfully used to launch strong DDoS attacks against several large companies. Other botnets, like *BASHLITE* [31] or *LIGHTAIDRA* [11] may present some technical differences, but most of them act similarly. The operations of an IoT botnet can be grouped into the following four phases:

**Target Selection.** Bots continuously scan the IPv4 address space searching for new vulnerable devices. They look for online devices that expose SSH/Telnet consoles or web interfaces, and try get access by either brute-forcing the login using a dictionary of credentials or exploiting known vulnerabilities. When a new vulnerable device is found, the bot informs the C2 server.

**Device fingerprinting and infection.** Once a new vulnerable device is found, the botnet tries to infect it. In order to load the correct infection code on the target device, the botnet first needs to discover its architecture (e.g., x86 vs ARM). To do this, the target has to be fingerprinted by issuing a sequence of different commands on its shell. After a matching fingerprint is found, the botnet uses shell commands to download and execute the infection code.

**Detection evasion and persistence.** The infection code uses detection evasion mechanisms to avoid being detected. *MIRAI*, for example, deletes the downloaded binary code and changes the bot process name using an alphanumeric string. In this particular case, the botnet software does not persist if the device is rebooted. Recent techniques try to avoid the blacklisting of C2 server IPs using domain fluxing [15]. Other botnets do not directly connect bots to the C2 server, but rather use proxies or *peer2peer* overlay network architectures to evade detection.

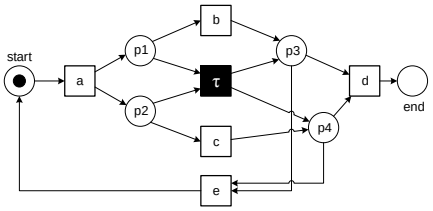
**Activation.** When the malicious code is running on the infected device, the new bot can be activated by the C2 server. Once activated, it starts performing the malicious actions requested by the attacker, like, for example, opening connections towards a targeted server in order to overload it.

## 3 Petri Nets and Process Mining

In this section, we present the process mining techniques that are the starting point of our attack triage approach. Preliminaries on Petri nets, which act as main artifacts to represent process models, and event logs are introduced as well.

### 3.1 Petri Nets and Event Logs

Many notations have been introduced to represent process models, such as BPMN, EPC or UML Activity Diagrams [22], and some of those are charac-




(a) Example of a Petri net

$$\gamma_1 = \begin{array}{|c|c|c|c|c|} a & d & b & c & \gg \\ a & \gg & b & c & d \end{array}$$

$$\gamma_2 = \begin{array}{|c|c|c|c|c|c|c|c|} a & \gg & d & \gg & \gg & b & c & \gg \\ a & \tau & \gg & e & a & b & c & d \end{array}$$

$$\gamma_3 = \begin{array}{|c|c|c|c|c|c|c|c|c|} a & \gg & \gg & \gg & \gg & \gg & d & b & c \\ a & \tau & e & a & c & b & d & \gg & \gg \end{array}$$

(b) Examples of alignments

Fig. 1: Examples of a Petri net and of trace alignments

terized by an ambiguous semantics. Since we need a simple language with clear semantics to explain our approach, we opted for Petri nets, which have proven to be adequate for representing process models [3]. This is especially true when the focus is only on the control-flow perspective, which is the case in this paper.

A *Petri net* $N = (P, T, F)$ is a directed graph with a set $P$ of nodes called *places* and a set $T$ of *transitions*. Places are represented by circles and transitions by rectangles. The nodes are connected via directed arcs $F \subseteq (P \times T) \cup (T \times P)$. Connections between two nodes of the same type are not allowed. Fig. 1a illustrates an example of a Petri net. Given a transition $t \in T$, ${}^\bullet t$ is used to indicate the set of *input places* of $t$, which are the places $p$ with a directed arc from $p$ to $t$ (i.e., such that $(p, t) \in F$). Similarly, $t^\bullet$ indicates the set of *output places*, namely the places $p$ with a direct arc from $t$ to $p$. At any time, a place can contain zero or more *tokens*, drawn as black dots. The state of a Petri net, a.k.a. *marking* $m$, is determined by the number of tokens in places, i.e., $m : P \to \mathbb{N}$.

In any run of a Petri net, the number of tokens in places (i.e., the marking) may change. A transition $t$ is *enabled* at a marking $m$ iff each input place contains at least one token, i.e., $\forall\ p \in {}^\bullet t,\ M(p) > 0$. A transition $t$ can *fire* at a marking $m$ iff it is enabled. As result of firing a transition $t$, one token is "consumed" from each input place and one is "produced" in each output place. This is denoted as $m \xrightarrow{t} m'$. In the remainder, given a sequence of transition firing $\sigma = \langle t_1, \ldots, t_n \rangle \in T^*$, $m_0 \xrightarrow{\sigma} m_n$ is used to indicate $m_0 \xrightarrow{t_1} m_1 \xrightarrow{t_2} \ldots \xrightarrow{t_n} m_n$.

An *event log* $L$ is a multi-set of *traces* $\sigma_L \in T^*$. A trace is a sequence of transition firings and describes the execution of a *process instance* in terms of the executed *activities*.[3] Transition firings in an event log are known as *events*. Event logs may store additional information about events such as the *timestamp* when the activity was executed. Some transitions do not represent process activities but are necessary to correctly represent a process model through Petri nets. These transitions are *invisible transitions* (the black-colored transition $\tau$ in the Petri net of Fig. 1a is invisible) and are not recorded as log events.

[3] We use multisets because the same trace can appear multiple times in an event log.

### 3.2 Process Discovery and Trace Alignment in Process Mining

Event logs are the starting point for any process mining technique. Typically, three types of process mining techniques can be distinguished [4]: (a) process discovery (learning a model from example traces in an event log), (b) conformance checking (comparing the observed behavior in the event log with a given modeled behavior), and (c) model enhancement (extending models based on additional information in the event logs, e.g., to highlight bottlenecks).

Process discovery techniques [13] automatically construct a representation of complex processes based on example executions in an event log, without using any a-priori information. In particular, we focus on online process discovery from event streams [29, 16, 17] as a way to deal with big amounts of data. Events are processed on-the-fly, as they occur, and only information about the most relevant ones is stored in a limited budget of memory. The discovered process models are represented using Petri nets that change over time, as new events are processed.

Then, we perform conformance checking by constructing an *alignment* of an event log and a process model [4, 9] to pinpoint where exactly deviations occur. To this aim, events in the log need to be matched with transitions in the model, and vice versa. In addition, to identify the alignment, we need to relate "moves" in the log to "moves" in the model. To represent moves in the log and moves in the model, we will use the symbol $\gg$ to indicate "no moves", i.e., moves in the log that cannot be mimicked by the model and vice versa. .

**Definition 1 (Alignment Moves).** *Let $N = (P, T, F)$ be a Petri net and $L$ be an event log with events in $E$. A legal* alignment move *for $N$ and $L$ is represented by a pair $(s_L, s_M) \in (E \cup \{\gg\} \times T \cup \{\gg\}) \setminus \{(\gg, \gg)\}$ such that:*

- *$(s_L, s_M)$ is a* move in the log *if $s_L \neq \gg$ and $s_M = \gg$,*
- *$(s_L, s_M)$ is a* move in the model *if $s_L = \gg$ and $s_M \in T$,*
- *$(s_L, s_M)$ is a* synchronous move *if $s_L = s_M$.*

An alignment is a sequence of alignment moves:

**Definition 2 (Alignment).** *Let $N = (P, T, F)$ be a Petri net with initial marking and final marking denoted with $m_i$ and $m_f$ Let $L$ be an event log. Let $\Gamma_N$ be the universe of all alignment moves for $N$ and $L$. Let $\sigma_L \in L$ be a log trace. Sequence $\gamma \in \Gamma_N^*$ is an* alignment *of $N$ and $\sigma_L$ if, ignoring all occurrences of $\gg$, the projection on the first element yields $\sigma_L$ and the projection on the second one yields a sequence $\sigma'' \in T^*$ such that $m_i \xrightarrow{\sigma''} m_f$.*

A move in the log for a transition $t$ indicates that $t$ occurred when not allowed; a move in model for a visible transition $t$ indicates that $t$ did not occur, when, conversely, expected. Many alignments are possible for the same trace. For example, Fig. 1b shows three possible alignments for a trace $\sigma_1 = \langle a, d, b, c \rangle$. Note how moves are represented vertically. For example, as shown in Fig. 1b, the first move of $\gamma_1$ is $(a, a)$, i.e., a synchronous move of $a$, while the second and the fifth move of $\gamma_1$ are a move in the log and in the model, respectively.

We aim at finding an alignment of $\sigma_L$ and $N$ with minimal deviation cost. In order to define the severity of a deviation, we first introduce a cost function on legal moves and, then, generalize it to alignments. The alignments with the lowest cost are called *optimal alignments*.

**Definition 3 (Cost Function).** *Let $N = (P, T, F)$ be a Petri net and $\sigma_L$ a log trace, respectively. Assuming $\Gamma_N$ as the set of all legal alignment moves, a* cost function *$\kappa$ assigns a non-negative cost to each legal move: $\Gamma_N \to \mathbb{N}_0^+$. The* cost of an alignment *$\gamma \in \Gamma_N$ between $\sigma_L$ and $N$ is computed as the sum of the cost of all constituent moves: $\mathcal{K}(\gamma) = \sum_{(s_L, s_M) \in \gamma} \kappa(s_L, s_M)$.*

$\gamma$ is an optimal alignment if, for any alignment $\gamma'$ of $N$ and $\sigma_L$, $\mathcal{K}(\gamma) \leq \mathcal{K}(\gamma')$. Consider the following cost function for the example in Fig. 1:

$$\kappa\big((s_L, s_M)\big) = \begin{cases} 1 \text{ if } s_M = \gg, \\ 1 \text{ if } s_L = \gg \text{ and } s_M \neq \tau, \\ 0 \text{ otherwise.} \end{cases} \tag{1}$$

With reference to the alignments in Fig. 1b, alignment $\gamma_1$ has cost 2, since it has 1 move in the model and 1 move in the log. Conversely, $\gamma_2$ has cost 4 and $\gamma_3$ has cost 6 (moves for invisible transitions $\tau$ have cost 0). Since no alignment exists with cost lower than 2, $\gamma_1$ is an optimal alignment. To quantify the amount of deviations between a trace and a model, we use the notion of *fitness* presented in [9] that takes into account the cost of the deviations. The outcome of the fitness is a score that may vary between 0 (very poor fitness) to 1 (perfect fitness between the trace and the model).

## 4 An Approach for Attack Triage

In this section, we introduce our solution and its approach for attack triage in IoT systems that leverages process mining to help prioritizing the analysis of new attack patterns. The output of our system is a prioritized list where unknown attacks have a higher priority with respect to known attacks. Moreover, a classification score for each attack is derived, which allows us to associate it to a specific botnet campaing. We first introduce an overview of the proposed approach and then move to describe the system.

### 4.1 Overview

The basic idea at the core of our approach is that operations performed by an attacker while infecting an IoT device can be logged, and this log can be analyzed to mine the infection processes. Therefore, the starting point of our approach is a log that contains *attack traces* each referring to a different IoT attack[4]

[4] Note that distinguishing attack interactions from bening interactions, namely *detecting attacks*, is a different problem that is out of the scope of this paper. We assume that data fed as input only contains traces of attacks.

Since the same botnet often exhibits a repetitive behavior during the device fingerprinting phase, our intuition is that process mining techniques can shed light on that behavior and support security operators to automatically distinguish between known and unknown attacks. Our approach models commands issued by the attacker on the target device while fingerprinting it as events of a process, it groups them into a trace representing the evolution of the attack and stores the trace in a log. Then, online discovery techniques of process models are leveraged to provide security operators with a process model representing an up-to-date picture of the attacks recorded in the log. Note that the log becomes indefinitely large over time as new attacks are recorded. This is the reason why online discovery techniques allow us to keep the analysis of the log computationally feasible by "forgetting" obsolete attacks. New observed traces are prioritized by aligning them to the discovered model and then calculating their fitness score.

Fig. 2 depicts the main steps in our approach. A target system keeps track of incoming connections from the internet and logs commands received on its interfaces (Raw log). The log is then filtered to get rid of spurious information and to prepare it for the subsequent model discovery phase. An online model discovery algorithm is then applied to the filtered logs in order to extract an up-to-date general *attack model*. When a new incoming *attack trace* is collected, we check its conformance with the attack model using trace alignment and compute the fitness of the new attack trace with the model. According to the fitness, we can assign a priority to the attack and report it to the security operators. When we detect that a given trace does not belong to a new attack, the trace is fed into the classification sub-system that is described in Section 4.2. Finally, the trace is used to feed the discovery algorithm to update the attack model.

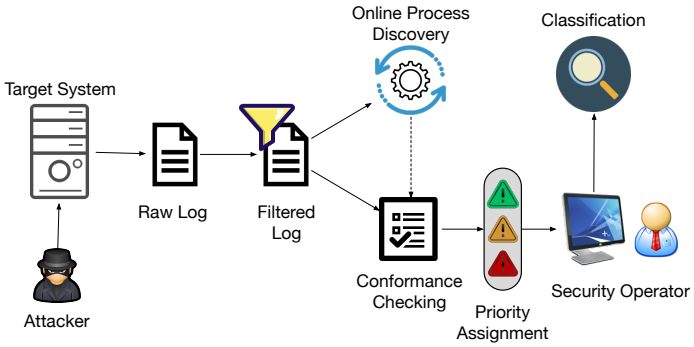



Fig. 2: Schematic overview of the proposed approach

### 4.2 Attack Classification

As already discussed in the previous section, given an attack trace, our approach is able to assign it a priority based on how much the trace differs from known

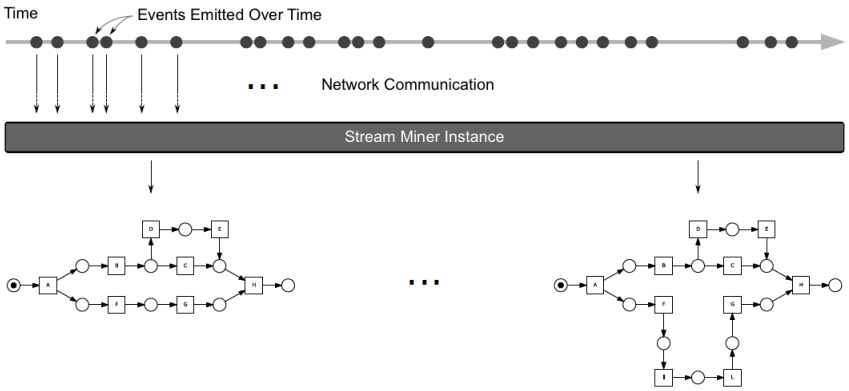



Fig. 3: Online process discovery [16]

behaviors. This information can be further enhanced by classifying "similar" attack traces in the same class of (already seen) attacks. Attack classification is a well known practice in IT security that enables important threat intelligence activities. In particular, it allows security operators to identify common characteristics of a class of attacks, information that is fundamental when studying the provenance of attacks or their attribution (i.e., who is performing the attack).

Even the classification task has been realized through process mining. Given a set of traces belonging to different classes of attack, it is possible to represent each class with a different process model. Then, trace alignment can be employed to understand which class (i.e., model) fits better with a recent captured trace. When a trace is particularly fitting with the model related to a class of attacks, its behavior can be later injected in the model of the identified class in order to keep it updated. Conversely, if the trace shows a low fitness value after performing the alignment task with any of the available (attack) models, then the trace is considered as belonging to a new, yet unknown class of attacks.

The above classification procedure requires an initial effort by the security operators to build the initial models reflecting the different classes of attacks; this is particularly true when just few attack traces have been collected and when a trace seems to belong to an unknown class. It is worth noting that the precision of the classification task strongly depends on the fitness threshold values used to state if a trace belongs to a class of attacks or not.

### 4.3 Online Process Discovery

One of the main aims of process mining is *automated process discovery*, i.e., learning process models from example traces recorded in some event log. Many different process discovery algorithms have been proposed in the past [13]. Basically, all such algorithms have been defined for batch processing, i.e., a complete event log containing all executed activities is supposed to be available for their execution. However, when dealing with data coming from IoT attacks, we have

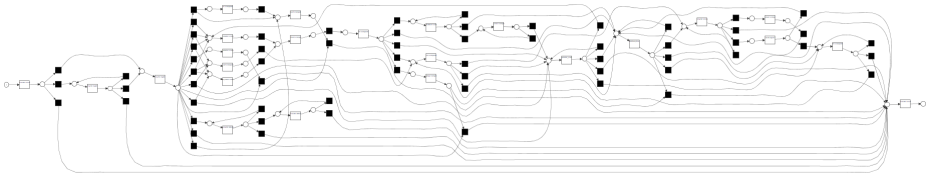


Fig. 4: Petri net representing the attack model

to deal with large amounts of events so that it becomes impossible to store all of them. Moreover, even if it would be possible to store all event data, it is often impossible to process them due to the exponential nature of most discovery algorithms. Finally, the process evolves over time when new attacks are detected.

For these reasons our solution makes use of the *online discovery algorithm* presented in [16], which is able to mine process models based on streaming event data. The general representation of the online discovery problem is shown in Fig. 3: one or more sources emit events (represented as solid dots). Events are observed by the miner that keeps the representation of the discovered model up-to-date. Algorithms that are supposed to interact with event streams must respect some requirements, such as: a) it is impossible to store the complete stream; b) backtracking over an event stream is not feasible, so algorithms are required to make only one pass over data; c) it is important to adapt the model to cope with unusual data values. The algorithm used in [16] is based on the Heuristics Miner [34], one of the most effective algorithms for practical applications of process mining [13]. Fig. 4 shows the Petri net discovered from the event stream provided by our honeypot at a given point in time.

### 4.4 Conformance Checking

One of ours goals is to provide security operators with an effective tool that supports them in the analysis of incoming traces representing malicious attacks. Since our approach is based on process mining, one of its strengths lies in the possibility of employing trace alignment to extend the range of the available security analysis features. In particular, with trace alignment, it is possible to "build" a relevant feedback for the security operators that are in charge of monitoring incoming attacks. This feedback includes the identification of unseen attack traces and insights into their structure. The latter enable the security operator to have a prioritized list of malicious traces ranked according to their distance from known behaviors and pinpoint where these traces differ from the up-to-date general attack model.

After an initialization phase of 1 day, the model discovered by the online discovery algorithm from the event stream provided by our honeypot was the one shown in Fig. 4. In Fig. 5, we show a trace containing a new attack and its discrepancies with the original attack model. In particular, the new attack requires 3 moves in the log to be aligned with the process model (the 3 activities

indicated in the figure should be skipped according to the original model) and the new trace has a fitness of 0.97 with the model. As shown in Section 4.3, the new behavior is taken into consideration by the online discovery algorithm that changes the process model into the one shown in Fig. 6.

It is important to observe that the cost of each legal alignment move depends on the specific application domain. Hence, the cost function $\kappa$ needs to be defined specifically for each setting and cannot be automated. For instance, in our context, inserting an *ls* command should not be punished too hard, since this command is not too relevant in the context of an IoT attack, while inserting a *ping* command should lead to a lower fitness of the trace. While approaches could be researched to support security operators in the definition of the cost function, this is beyond the scope of this paper and left for future work.

## 5 Evaluation

We developed a prototype implementing our proposed approach using the ProM[5] framework. To evaluate our approach, we first collected data from three botnets that we ran against an instance of a *Cowrie* honeypot. Then, we tested the ability of our solution to *recognize* unknown attacks and *classify* attack traces according to their behaviour. Finally, we performed a further experiment through a honeypot instance collecting real attacks over the web.

### 5.1 Experimental Setup

In order to collect data to test our approach, we setup an honeypot to mimic the behavior of a target system. A honeypot is a software device that simulates the

[5] ProM (`http://www.promtools.org/`) is an is a open-source framework for implementing process mining tools and algorithms.


Fig. 5: Alignment for a non-compliant trace

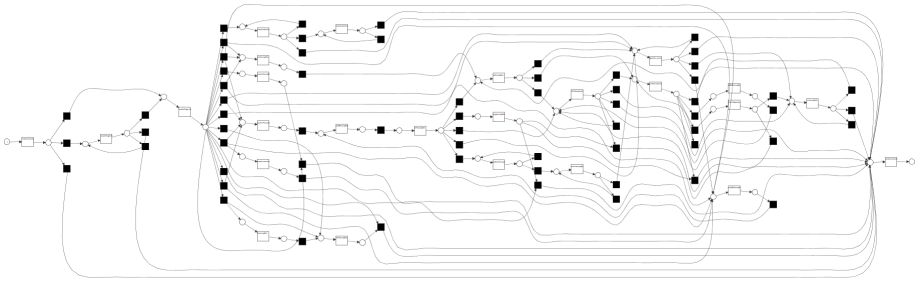


Fig. 6: Petri net modified with the addition of a non-compliant trace

behavior of a real system to fool attackers in infecting it. Using a honeypot, it is possible to collect data about attacks without running the risk of compromising real systems. In addition, a honeypot includes security features that block the attacker as soon as s/he tries to execute dangerous commands (e.g., execute infection code). In particular, we setup an instance of *Cowrie* [1], a medium interaction SSH and Telnet honeypot. This software can be configured to mimic different linux-based environments.[6] We configured the honeypot to accept login attempts with user "root" and any password.

**Data Filtering.** When a new connection is received, Cowrie logs all the commands prompted by the attacker into a JSON file, also registering if the command is executed successfully or not. Since the logs produced by Cowrie are highly verbose, we filtered out all the events that were not related to an interaction between the attacker and the honeypot. For example, after each login, the honeypot logs the internal event *cowrie.client.size*; we discarded this event, since it does not provide interesting information about the attack behavior.

The commands prompted by an attacker into the shell represent the most important sources of information of a botnet. Hence, in some preliminary tests, we tried to model all the shell commands as activities. However, in this way, our logs recorded more than five thousands different commands (considering also their arguments). Since most process mining techniques have been tested with a smaller maximum number of activities, it has been required to reduce the number of possible activities by filtering out some of them. Therefore, we decided to ignore the arguments of each command. For example, if the attacker prompts the command *$ ls /usr/bin*, we modeled it as *$ ls*. After this first filter, we removed all front and back spaces in order to normalize all commands. From the resulting set of shell commands, we retained only the most frequent ones:

– */bin/busybox*, *rm*, *enable*, *shell*, *sh*, *system*, *cd*, *dvrHelper*, *cat*, *echo*, *tftp*, *uname*, *killall*, *ping*, *linuxshell*, *exit*, *ls,wget*, *chmod*, */bin/busybox wget*, */bin/busybox tftp*.

[6] Many commercial IoT devices are based on linux-like operating systems.

Other commands were modeled with the activity *generic command*. Finally, we represented all shell commands starting with the symbol > with activity *redirect*. An example of attack trace obtained after the filtering step is:

$$\langle cowrie.session.connect, cowrie.login.success, enable, sh, /bin/busybox, rm, \\ cd, /bin/busybox, cowrie.session.file\_download, cowrie.session.closed \rangle.$$

Note that, together with shell commands, internal commands of the honeypot like *cowrie.session.connect* and *cowrie.login.success* can occur.

**Data Collection** We collected data from the honeypot in two different setups: Controlled Environment (CE) and Not Controlled Environment (NCE).

**CE**: We put the honeypot into an isolated LAN network to enable it receiving "controlled" attacks from three different botnets whose source code has been leaked and publicly released: *BASHLITE*, *LIGHTAIDRA* and *MIRAI*. We triggered the botnets to generate new attacks; consequently, we exactly knew which specific botnet produced an attack trace. For the CE configuration, we collected 1000 different attack traces for each botnet.

**NCE**: The honeypot had an associated public address with ports 22 and 23 open (i.e., reachable by anyone over the internet). This has allowed us to test the feasibility of our approach when deployed in a real world environment. In this configuration, we collected a total of 122 complete attack traces in four days.

### 5.2 Detection of Unknown Attacks

We initially performed an experiment to test if our system was able to distinguish between a new attack and a known one. We used the data collected from the honeypot in the CE configuration. Results are shown in Fig. 7a.

Firstly, we mined a general attack model using the first attack trace obtained with the *BASHLITE* botnet. After that, we computed the fitness of the model with the remaining 999 attack traces produced by the same botnet. Since this botnet exhibited the same behavior for any attack trace, the fitness score for each trace was 1, i.e., the general attack model was able to recognize all known attacks produced by *BASHLITE*.

Then, we computed the fitness of the previously discovered attack model with the first attack trace obtained with *LIGHTAIDRA*. In this case, we measured a fitness of 0.24, meaning that a new attack was discovered. This because the attack model has been initially trained just over *BASHLITE* attack traces. Since the first attack trace of *LIGHTAIDRA* was unknown for the general attack model (i.e., fitness score lower than 1), we updated it to reflect the new recorded behaviour. When we computed the fitness with the second trace obtained with *LIGHTAIDRA*, we measured a higher accuracy, i.e., a fitness around 0.88, and we again updated the model (even if fitness scores greater than 0.8 refer to attack traces that are only partially unknown, as they represent variants of known attacks, cf. Section 5.4). We repeated this procedure for the first 20

attack traces obtained with *LIGHTAIDRA*, being the measured fitness lower than 1. After that, for the remaining attack traces, the measured fitness was 1.

Finally, as expected, when we computed the fitness with the first trace obtained with *MIRAI*, we registered a new drop in the fitness: 0.34. Repeating the same update procedure as before, we started to measure high fitness scores (around 0.97) after the first 20 traces produced by *MIRAI*.

The above results show that with our approach we can distinguish between new attacks and known ones, just looking at the value of the fitness.

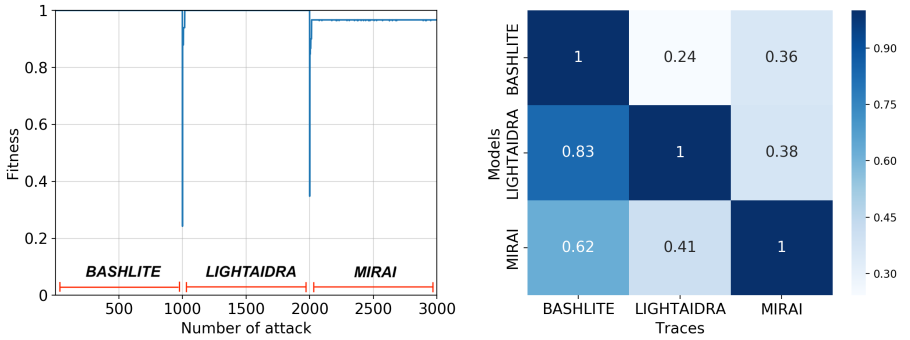
### 5.3 Attack Classification

The aim of this second experiment was to verify the ability of our system to classify the kind of attack underlying a recorded log trace. Leveraging again the CE configuration, for each botnet, we split any log in two sublogs including 500 traces each. Given a botnet, we used the 500 traces in the first sublog as training set, in order to discover the attack model underlying the specific botnet. We performed this same procedure for the other two botnets, obtaining at the end three attack models, one per botnet. Then, we used the 500 traces in the second sublogs (one per botnet) as test set, computing the fitness of any trace with respect to the attack models of the various botnets.

The results of this experiment are summarized in the confusion matrix of Fig. 7b, which includes the mean of the fitness score of traces taken by the second sublogs with the three attack models. The analysis of the matrix makes clear that our approach can classify attacks belonging to specific botnets very accurately. Moreover, we can notice that the fitness between the traces of *BASHLITE* and the *LIGHTAIDRA* model is high. This is because *LIGHTAIDRA* is an evolution of *BASHLITE*, and shares some similar behaviours in its attacks.

### 5.4 Experiments with real attack traces

The aim of this third experiment is to test the effectiveness of the proposed approach in a real world scenario. Leveraging the NCE configuration, which allowed us to collect data associated to real attack traces, we investigated the amount of new attacks that was possible to identify using our approach. The results of this experiment are reported in Fig. 8, together with the fitness score for each trace. Traces are sorted according to their timestamp. Analysing such results, two considerations can be made: *(i)* looking at the drop in the fitness we distinguished 13 different types/classes of attacks; and *(ii)* the threshold of the fitness score for identifying new unknown attacks can be fixed to 0.8. This value, which is a result of this last empirical test, is useful to consider attacks having a fitness score greater than 0.8 but lower than 1 as variants of already known attacks, and not as totally new attacks.




(a) Fitness score for attack traces of the three botnets. The arrow represent the fitness drop due to a new type of attack.

(b) Confusion Matrix for the classification experiment. The columns represent the traces of the 3 botnets and the rows represent the respective attack models.

Fig. 7: Experiments for detection of unknown attacks and attack classification

## 6 Related Work

**IoT Botnets** — Research in the area of IoT security is recent by its nature, and only contains few important works that have been driven by the observation of attacks in the last 5 years. In particular there are several works that aimed at analyzing the evolution of the largest botnets based on IoT devices. We already cited the work by Antonakakis [12] that provides an in-depth analysis of *MIRAI*, while the evolution of *MIRAI* and *BASHLITE* was described by Marzano et al. in [31]. From a more general standpoint, Cozzi et al. [19] studied common characteristics of current linux malware, an important contribution for this research area, as a large number of IoT devices run on some form linux-based platform.

**Attack Triage** — The idea of applying the concept of triage to attacks comes from practitioners that first experimented it within security operation centers of large companies. In particular, it is often associated with the analysis of malware. Recent research contributions applied this concept to malware analysis for android-based platforms. Bitshred [25] proposes a probabilistic data structure created through feature hashing for large-scale correlation of malware samples. Bitshred was designed to efficiently identify samples of similar malware, but differently from us, its internal data structures are not designed to details how two samples differ. SigMal [27] shares some similarities with Bitshred, but uses signal processing based analysis to improve resistance to noise. More recently, Calleja et al. [18] showed that confusing statistical classification systems may be easy for malware writers. More generally, the recent research trend on adversarial machine learning [24] cast a shadow on the robustness of triage solutions based on statistical models. Recently, Shen et al. have shown that it is possible to find similarities between attacks with temporal word embeddings [32]. If compared

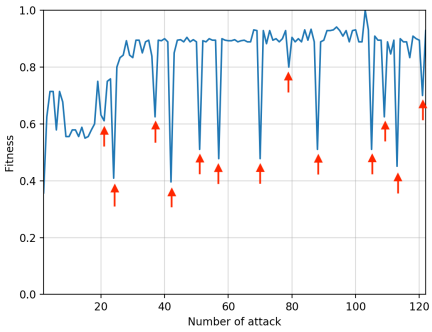



Fig. 8: Fitness score for the attacks in the NCE configuration. In red we report the values of fitness below 0.8 (from trace 20 and on) that represent a new attack.

with our approach, the main difference is that the work [32] relies on explicit alerts given by an Intrusion Detection System (IDS) and requires a very large dataset of alerts to produce reliable results.

**Process Mining and Security** — In the research literature, there are some studies that advocate the use of process mining to analyse logs for the detection of security violations in business processes [6, 5, 7, 26, 8]. It is worth noting that such studies focus on investigating security aspects regarding the *execution* of business processes. Conversely, our work aims at applying process mining techniques directly to malicious traces extracted from event logs containing attacks. We found only two existing studies that go in our same direction [10, 14].

In [10], the authors combine process mining techniques and visual features to help a network administrator analyzing the alerts (i.e., the amount of malicious events included in a trace) generated by IDSs. If this amount is greater than a predefined threshold, the alert is captured. Process discovery techniques have been also employed for studying and classifying malware and malicious code in [14]. In this work, by extracting a process model from the system logs of infected devices and comparing it against the normal execution of non-infected and similar devices, the authors create a classification of malware families.

Differently from the above studies, which employ offline process discovery algorithms coupled with ad-hoc attack detection techniques, our solution uses online discovery in combination with trace alignment. This ensures a more precise identification of malicious traces and gives insights about their potential impact.

## 7 Concluding Remarks

In this paper, we have presented a novel solution for attack triage support dedicated to IoT systems that, leveraging process mining techniques, can help security operators to quickly identify new attacks with unknown behaviors, to later analyze them in detail. We implemented and tested our solution with traces generated by running publicly available botnet code (*MIRAI*, *BASHLITE* and

*LIGHTAIDRA*) in a controlled environment, experimentally demonstrating the validity of the proposed approach. Furthermore, we validated the effectiveness of this system on a real-world use case considering traces of attacks collected with our IoT honeypot.

It is worth noting that current algorithms for online process discovery and trace alignment performs and scales very well with input models consisting of hundreds of transitions and logs including thousands of execution traces [28, 20, 13, 21], making them very suitable for the automated triage of IoT attacks.

Future work will be devoted to validate our approach against larger collections of attack traces characterized by larger botnet variability. Furthermore, we will investigate ways to automatically tune at run-time the cost function used for trace alignment and the alarm threshold applied to the fitness score.

Finally, we plan to investigate how our approach can be used to monitor systems and lock attacks as they unfold. To this aim, *predictive process monitoring* methods [30] can be employed to provide the user with predictions about the future of an ongoing trace. The forward-looking nature of predictive monitoring enables applications where evolving traces can be analyzed before completion, to predict how they evolve, and possibly identify them as malicious (and thus block them), before the infection vector is installed on the target system.

**Acknowledgments.** This work has been partially supported by the Estonian Research Council Grant IUT20-55, the Italian "Dipartimento di Eccellenza" grant for DIAG at Sapienza University of Rome, the Sapienza grants IT-SHIRT, ROCKET and METRICS, the PANACEA project under the grant agreement 826293, and a student grant from Vitrociset S.p.A.

## References

1. Cowrie. `https://github.com/cowrie/cowrie`
2. The ddos that didn't break the camel's vac. `https://goo.gl/p9kUCy` (2017)
3. van der Aalst, W.M.: The application of Petri nets to workflow management. Journal of circuits, systems, and computers **8**(01), 21–66 (1998)
4. van der Aalst, W.M.: Process Mining - Data Science in Action. Springer (2016)
5. van Aalst, W.M., van Hee, K.M., van Werf, J.M., Verdonk, M.: Auditing 2.0: Using process mining to support tomorrow's auditor. Computer **43**(3) (2010)
6. van der Aalst, W.M., de Medeiros, A.K.A.: Process mining and security: Detecting anomalous process executions and checking process conformance. Electronic Notes in Theoretical Computer Science **121**, 3–21 (2005)
7. Accorsi, R., Stocker, T.: On the exploitation of process mining for security audits: the conformance checking case. In: SAC'12. pp. 1709–1716 (2012)
8. Accorsi, R., Stocker, T., Müller, G.: On the exploitation of process mining for security audits: the process discovery case. In: SAC'13. pp. 1462–1468 (2013)
9. Adriansyah, A., Sidorova, N., van Dongen, B.F.: Cost-Based Fitness in Conformance Checking. In: ACSD'11 (2011)
10. de Alvarenga, S.C., Zarpel, B., Miani, R.: Discovering attack strategies using process mining. In: AICT'15. pp. 119–125 (2015)
11. Angrishi, K.: Turning internet of things (iot) into internet of vulnerabilities (iov): Iot botnets. Tech. rep., arXiv preprint arXiv:1702.03681 (2017)

12. Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., Durumeric, Z., Halderman, J.A., Invernizzi, L., Kallitsis, M., et al.: Understanding the Mirai Botnet. In: 26th USENIX Security Symp. pp. 1093–1110 (2017)
13. Augusto, A., Conforti, R., Dumas, M., La Rosa, M., Maggi, F.M., Marrella, A., Mecella, M., Soo, A.: Automated discovery of process models from event logs: Review and benchmark. IEEE TKDE **31**(4), 686–705 (2018)
14. Bernardi, M.L., Cimitile, M., Distante, D., Martinelli, F., Mercaldo, F.: Dynamic malware detection and phylogeny analysis using process mining. Int. J. of Information Security pp. 1–28 (2018)
15. Bertino, E., Islam, N.: Botnets and Internet of Things Security. IEEE Comp. (2017)
16. Burattin, A.: Applicability of Process Mining Techniques in Business Environments. Ph.D. thesis, University of Bologna, Italy (2013)
17. Burattin, A., Cimitile, M., Maggi, F.M., Sperduti, A.: Online Discovery of Declarative Process Models from Event Streams. IEEE Trans. Serv. Comp. **8**(6) (2015)
18. Calleja, A., Martín, A., Menéndez, H.D., Tapiador, J., Clark, D.: Picking on the family: Disrupting android malware triage by forcing misclassification. Expert Syst. Appl. **95**, 113–126 (2018)
19. Cozzi, E., Graziano, M., Fratantonio, Y., Balzarotti, D.: Understanding linux malware. In: 39th IEEE Symp. on Security and Privacy (SP). pp. 161–175 (2018)
20. De Giacomo, G., Maggi, F.M., Marrella, A., Patrizi, F.: On the Disruptive Effectiveness of Automated Planning for LTL$f$-Based Trace Alignment. In: AAAI'17. pp. 3555–3561 (2017)
21. van Dongen, B.F.: Efficiently computing alignments. In: BPM'18 (2018)
22. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A., et al.: Fundamentals of Business Process Management, vol. 1. Springer (2013)
23. Hossain, M.M., Fotouhi, M., Hasan, R.: Towards an analysis of security issues, challenges, and open problems in the internet of things. In: SERVICES'15 (2015)
24. Huang, L., Joseph, A.D., Nelson, B., Rubinstein, B.I., Tygar, J.: Adversarial machine learning. In: AISEC'11. pp. 43–58 (2011)
25. Jang, J., Brumley, D., Venkataraman, S.: Bitshred: feature hashing malware for scalable triage and semantic analysis. In: CCS'11. pp. 309–320 (2011)
26. Jans, M., Alles, M., Vasarhelyi, M.: The case for process mining in auditing: Sources of value added and areas of application. Int. J. of Acc. Inf. Syst. **14**(1), 1–20 (2013)
27. Kirat, D., Nataraj, L., Vigna, G., Manjunath, B.: Sigmal: A static signal processing based malware triage. In: ACSAC'13. pp. 89–98 (2013)
28. de Leoni, M., Marrella, A.: Aligning Real Process Executions and Prescriptive Process Models through Automated Planning. Expert Syst. Appl. **82** (2017)
29. Maggi, F.M., Burattin, A., Cimitile, M., Sperduti, A.: Online process discovery to detect concept drifts in LTL-based declarative process models. In: CoopIS'13 (2013)
30. Maggi, F.M., Di Francescomarino, C., Dumas, M., Ghidini, C.: Predictive monitoring of business processes. In: CAiSE'14. pp. 457–472 (2014)
31. Marzano, A., Alexander, D., Fonseca, O., Fazzion, E., Hoepers, C., Steding-Jessen, K., Chaves, M.H., Cunha, Í., Guedes, D., Meira, W.: The Evolution of Bashlite and Mirai IoT Botnets. In: ISCC'18. pp. 813–818 (2018)
32. Shen, Y., Stringhini, G.: Attack2vec: Leveraging temporal word embeddings to understand the evolution of cyberattacks. In: 28th Usenix Security Symp. (2019)
33. The OSWAP Fundation: OWASP Internet of Things Project. `https://tinyurl.com/yc3plqr9` (2014)
34. Weijters, A.J.M.M., van der Aalst, W.M.P.: Rediscovering workflow models from event-based data using little thumb. Int. Comp.-Aided Eng. **10**(2) (2003)