Modeling Wazuh rules with Weighted Timed Automata

Anass Haydar, Mathias Ramparison

Modeling Wazuh rules with Weighted Timed Automata

Anass Haydar, Mathias Ramparison[0000−0001−6764−1214]

Univ. Grenoble Alpes, CNRS, Grenoble INP†, VERIMAG, UMR 5104, 38000, Grenoble, France

Abstract. In the rapidly evolving landscape of cybersecurity, Endpoint Detection and Response (EDR) systems have emerged as a critical advancement in cybersecurity, providing organizations with enhanced capabilities to detect, investigate, and mitigate sophisticated attacks. The open-source EDR platform Wazuh is specifically highlighted as an attractive option for organizations and provides comprehensive security monitoring, threat detection, and incident response capabilities without the burden of licensing costs. A critical component of the Wazuh system is its rules set, which are predefined or custom-written conditions that analyze log data and system events to identify potential security threats, anomalies, or policy violations. These rules, typically written in XML format, form the core of Wazuh's threat detection capabilities. However, in complex architectures, the set of rules can be challenging to understand and update, and different rules can overlap, preempt, or cancel each other. To address this issue, we propose to model Wazuh rules as Weighted Timed Automata, which helps to verify that rules are well-triggered by verifying the reachability of the corresponding state in the automaton using the model checker Uppaal.

1 Introduction

In the face of escalating cyber threats, Endpoint Detection and Response (EDR) systems have emerged as a critical advancement in cybersecurity, offering organizations enhanced capabilities to detect, investigate, and mitigate sophisticated attacks. These systems provide real-time monitoring and analysis of endpoint activities, leveraging advanced analytics and machine learning to identify anomalous behaviour that may indicate a security breach. The growing interest in EDR technology is driven by its ability to offer comprehensive visibility into endpoint activities, facilitate rapid incident response and provide valuable forensic data, thus enabling organizations to adopt a more proactive and robust approach to cybersecurity in an increasingly complex threat landscape. Specifically, the open-source EDR Wazuh [Waz22] attracts considerable attention in the cybersecurity field as an open-source platform that provides comprehensive security monitoring, threat detection, and incident response capabilities without the burden of licensing costs. Its versatility in supporting multiple operating systems,

†Institute of Engineering Univ. Grenoble Alpes

coupled with features such as intrusion detection, log analysis, and compliance monitoring, makes it a compelling choice for organizations seeking to bolster their security infrastructure efficiently and economically. A critical component of Wazuh is the rules set. Wazuh rules are predefined or custom-written conditions that analyze log data and system events to identify potential security threats, anomalies, or policy violations within the Wazuh security platform. These rules, typically written in XML format, form the core of Wazuh's threat detection capabilities by defining specific patterns to look for in collected data and triggering appropriate alerts or actions when these patterns are detected, thereby enabling organizations to effectively monitor and respond to security incidents. However in complex architectures the set of rules can be difficult to understand and update and different rules can overlap and preempt or even cancel each other.

Several methods have been developed to tackle this issue, especially for Intrusion Detection Systems (IDS). In [AL15] the authors parse every rules and generate a payload to test each of them in order to detect overlapping rules. In [NK20] the authors propose a semantic analysis approach to identify redundant rules.

In this paper, we propose a framework to model rules as Uppaal Weighted Timed Automata [BY03] and to ensure that rules are well triggered by verifying the reachability of the corresponding state in a network of automata. In contrast with [AL15,NK20] we do not test rules by generating payloads nor check for redundancy of rules but we verify that custom-rules are able to be triggered. On top of that, Wazuh integrates IDS such as Suricata [Sur] and processes its output: Wazuh covers a wider range of behaviours than IDSs.

Timed automata (TAs) [AD94] represent a powerful formalism to model and verify systems where concurrency is mixed with hard timing constraints. TAs are an extension of finite-state automata with clocks, *i. e.,* real-valued variables, that can be compared to integer constants and updated to 0 along edges (called reset in the literature). TAs benefit from many decidability results such as the reachability of a discrete location, which is PSPACE-complete [AD94] (and some undecidability results too, such as language inclusion). Several optimization issues have been demonstrated to be solvable, such as minimum-cost reachability in the context of Weighted/Priced Timed Automata where integer constants can be used to model costs [BFH+01,ALP04].

Outline. Section 2 recalls necessary definitions. Section 3 introduces the formalism of Weighed Timed Automata while Section 4 presents Wazuh rules. Section 5 describes our framework. Section 6 concludes the article and outlines future research directions.

2 Preliminaries

Let $\mathbb{N}$, $\mathbb{Z}$, and $\mathbb{R}_+$ denote the set of non-negative integers, integers and non-negative reals, respectively.

We assume a set $\mathbb{X} = \{x_1, \ldots, x_H\}$ of *clocks*, *i. e.,* real-valued variables that evolve at the same rate. A *clock valuation* is a function $v : \mathbb{X} \to \mathbb{R}_+$. We write $\vec{0}$ for the clock valuation assigning 0 to all clocks. Given $d \in \mathbb{R}_+$, $v + d$ denotes the valuation s.t. $(v + d)(x) = v(x) + d$, for all $x \in \mathbb{X}$. Given $R \subseteq \mathbb{X}$, we define the *reset* of a valuation v, denoted by $[v]_R$, as follows: $[v]_R(x) = 0$ if $x \in R$, and $[v]_R(x) = v(x)$ otherwise.

We assume a set $\mathbb{W} = \{w_1, \ldots, w_M\}$ of *weights*. A *weight valuation* μ is a function $\mu : \mathbb{W} \to \mathbb{Z}$. We write $\vec{0}_{\mathbb{W}}$ for the weight valuation assigning 0 to all weights.

A linear arithmetic expression over $\mathbb{W}$ is $\sum_i a_i w_i + c$, where $w_i \in \mathbb{W}$ and $a_i, c \in \mathbb{Z}$. Let $\mathcal{LA}(\mathbb{W})$ denote the set of arithmetic expressions over $\mathbb{W}$. A weight update is a partial function $\alpha : \mathbb{W} \nrightarrow \mathcal{LA}(\mathbb{W})$. That is, we can assign a weight to an arithmetic expression of weight values and constants.

We assume $\bowtie \in \{<, \leq, =, \geq, >\}$. A *guard* g is a constraint over $\mathbb{X} \cup \mathbb{W}$ defined by a conjunction of inequalities of the form $x \bowtie d$ or $w \bowtie k$ with $x \in \mathbb{X}$, $w \in \mathbb{W}$, $d, k \in \mathbb{N}$. Given g, we write $(v, \mu) \models g$ if the expression obtained by replacing each x with $v(x)$ and each w with $\mu(w)$ in g evaluates to true.

Given a weight valuation μ and a weight update α, we need an evaluation function $eval(\alpha, \mu)$ returning a weight valuation, and defined as follows: $eval(\alpha, \mu)(w) = \mu(w)$ if $\alpha(w)$ is undefined, and $\mu(\alpha(w))$ otherwise, where $\mu(\alpha(w))$ denotes the replacement within the linear arithmetic expression $\alpha(w)$ of all occurrences of a weight w_i by its current value $\mu(w_i)$. Observe that this replacement gives a integer constant, therefore $eval(\alpha, \mu)$ is indeed a weight valuation $\mathbb{W} \to \mathbb{Z}$.

3 Weighted Timed Automata

We take advantage of the multiple modeling opportunities offered by Timed Automata and Uppaal [BY03] to establish the following formalism to model the activation of a security rule as a combination of discrete and continuous events, enabling the verification of its behavior. The resulting structure can be seen as an extension of a Weighted/Priced Timed Automaton [BFH+01,ALP04] with only integer weights on edges (the discrete "switch" weight part of [BFH+01,ALP04]) in guards and/or weight updates. Moreover, we allow multiple weights and also to set to zero and decrement weights. Our formalism can also be seen as the non parametric version of [ALRS21] where operations can be performed on multiple parametric weights, but where weights cannot be compared in guards.

Definition 1. *A Weighted Timed Automaton (*WTA*) $\mathcal{A}$ is a tuple $\mathcal{A} = (\Sigma, L, l_0, F, \mathbb{X}, \mathbb{W}, I, E)$, where:*

1. *Σ is a finite set of synchronization actions,*
2. *L is a finite set of locations,*
3. *$l_0 \in L$ is the initial location,*
4. *$F \subseteq L$ is the set of accepting locations,*
5. *$\mathbb{X}$ is a finite set of clocks,*

6. $\mathbb{W}$ *is a finite set of weights,*
7. I *is the invariant, assigning to every* $l \in L$ *a guard* $I(l)$*,*
8. E *is a finite set of edges* $e = (l, g, a, R, \alpha, l')$ *where* $l, l' \in L$ *are the source and target locations,* g *is a guard,* $a \in \Sigma$*,* $R \subseteq \mathbb{X}$ *is a set of clocks to be reset, and* $\alpha : \mathbb{W} \nrightarrow \mathcal{LA}(\mathbb{W})$ *is a weight update.*

Let us now define the concrete semantics of WTA.

Definition 2 (Semantics of a WTA). *Given a* WTA $\mathcal{A} = (\Sigma, L, l_0, F, \mathbb{X}, \mathbb{W}, I, E)$*, the semantics of* $\mathcal{A}$ *is given by the timed transition system (TTS)* $(S, s_0, \to)$*, with*

- $S = \{(l, v, \mu) \in L \times \mathbb{R}_+^H \times \mathbb{Z}^M \mid v \models I(l)\}$,
- $s_0 = (l_0, \vec{0}, \vec{0}_{\mathbb{W}})$,
- $\to$ *consists of the discrete and (continuous) delay transition relations:*
  1. *discrete transitions:* $(l, v, \mu) \overset{e}{\mapsto} (l', v', \mu')$*, if* $(l, v, \mu), (l', v', \mu') \in S$*, and there exists* $e = (l, g, a, R, \alpha, l') \in E$*, such that* $(v, \mu) \models g$*,* $v' = [v]_R$*, and* $\mu' = eval(\alpha, \mu)(w)$*;*
  2. *delay transitions:* $(l, v, \mu) \overset{d}{\mapsto} (l, v + d, \mu)$*, with* $d \in \mathbb{R}_+$*, if* $\forall d' \in [0, d], (l, v + d', \mu) \in S$*.*

That is, a state is a triple made of the current location, the current clock valuation, and the current weight valuation. The clock valuations evolve naturally as in timed automata, while the current weight evolves according to the weight update function.

Moreover we write $(l, v, \mu) \overset{(e,d)}{\longrightarrow} (l', v', \mu')$ for a combination of a delay and discrete transition if $\exists v'' : (l, v, \mu) \overset{d}{\mapsto} (l, v'', \mu) \overset{e}{\mapsto} (l', v', \mu')$. Given $\mathcal{A}$ with concrete semantics $(S, s_0, \to)$, we refer to the states of S as the *concrete states* of $\mathcal{A}$. A *run* of $\mathcal{A}$ is an alternating sequence of concrete states of $\mathcal{A}$ and pairs of edges and delays starting from the initial state s_0 of the form $s_0 \overset{e_0}{\longrightarrow} s_1 \overset{e_1}{\longrightarrow} \cdots \overset{e_{m-1}}{\longrightarrow} s_m \overset{e_m}{\longrightarrow} \cdots$, such that for all $i = 0, 1, \ldots$, $e_i \in E$, and $(s_i, e_i, s_{i+1}) \in \to$. Given a state $s = (l, v, \mu)$, we say that s is reachable if s belongs to a run of $\mathcal{A}$. By extension, we say that l is reachable in $\mathcal{A}$, if there exists a state (l, v, μ) that is reachable.

Communication between processes can occur synchronously through hand-shake synchronization, which utilizes input and output actions. To represent hand-shake synchronization, it is assumed that the set of synchronization actions Σ contains symbols for input actions (denoted as a?) and output actions (denoted as a!). As in [BY03], we will manipulate a network of WTA where they interact using a synchronization channel, *i. e.,* pairs of labels (a!, a?). One automaton can broadcast a!, and the other remains in a state of anticipation a?. Either a single automaton can move on its own or several automata can move simultaneously.

Uppaal facilitates the modeling of atomic action sequences, such as atomic broadcast, through the concept of committed locations. In a committed location, no delay is permitted. In a network of WTA, if a process occupies a committed

location, only action transitions originating from that committed location are allowed to occur. Consequently, processes in committed locations can only be interleaved with other processes residing in committed locations. Each WTA in a network may contain a subset $L_c \subseteq L$ of locations, designated as committed locations. As a syntactical constraint, only predicates over weights are permitted to appear in a guard on an outgoing edge from a committed location; no clock constraints are allowed. Regular locations of $L \setminus L_c$ are represented as location and committed locations of L_c as committed location.

4 Rules in Wazuh

Wazuh is an open-source security platform that provides comprehensive security monitoring, threat detection, and incident response capabilities. It combines a powerful Security Information and Event Management engine (SIEM) with advanced capabilities for endpoint detection and response, file integrity monitoring, and vulnerability detection. Wazuh offers a centralized management console– called *Wazuh manager*–, real-time alerts, and comprehensive reporting to help organizations effectively secure their IT infrastructure and protect against cyber threats. To detect and report behaviours of any endpoint connected to the Wazuh manager, *rules* have to be defined. In the usual workflow:

- Wazuh's agent software (endpoint) collects security-related events from the monitored systems and sends them to the Wazuh manager;
- the Wazuh manager then processes these events and matches them against the defined rules;
- when an event matches the conditions specified in a rule, the corresponding action(s) are triggered, such as generating an alert.

4.1 Existing and custom rules

In Wazuh, rules play a crucial role in the security monitoring and detection process. Wazuh comes with a set of pre-defined rules that cover a wide range of security scenarios, such as file integrity monitoring, user activity monitoring, and detection of known threats. In addition to existing rules that cover basic security events, administrators can create custom rules to address specific security requirements or to detect organization-specific threats. Rules can be configured to have different levels of severity, allowing for prioritization and effective incident response.

Wazuh rules are defined in XML files located by default in the `/var/ossec/etc/rules` directory [Waza]. Each rule consists of various elements, such as the rule's `id` (used to identify the rule that is triggered in the Wazuh manager), `description` (log to display in the Wazuh manager), `level` (incident level, that infers in some cases a priority between rules, at the discretion of the administrator), and `match` conditions [Wazb]. Match conditions within rules are a crucial component that define the criteria for triggering alerts or other actions: they are the logical expressions that are used to evaluate the incoming

security events. These conditions are defined within the `<match>` tag in the rule configuration. The match conditions can include various elements, such as event attributes, file paths, user information, regex, or other contextual data. We do not focus on match conditions any further; we only detect (or match) a triggered rule using the the tags `<if_sid>` and `<if_matched_sid>`: the rule id that we are trying to match. Rules matching rules is called *rules encapsulation*. Additionally we focus on the following components:

- `frequency`: the number of times a rule is matched before triggering;
- `timeframe`: the duration in seconds where we consider the frequency.

In Listing 1.1, rule 100101 is a simple rule that is triggered if 60122 is triggered, and shown in the Wazuh manager with the log in the `<description>` tag. Rule 60122 is a default rule that indicates logon failures *e. g.,* a failed ssh authentication on Windows systems (see [Wazc] for the default set of rules).

4.2 Rules preemption

Several factors affect rules and define the preemption of a rule over another one *i. e.,* whether a rule will have priority and prevent other rules from triggering. In this section, we list the properties that we found to be followed by Wazuh considering rules encapsulation, the frequency (number of occurrences) and timeframe parameters and the order in which the rules are written in the configuration file w.r.t. their incident level.

In the following, we will consider the following set of rules.

Listing 1.1: Example set of rules.

```
<rule id="100101" level="5">
  <if_sid>60122</if_sid>
  <description>Failed tentative to log on</description>
</rule>
<rule id="100102" level="5">
  <if_sid>100101</if_sid>
  <description>Failed tentative to log on</description>
</rule>
<rule id="100103" frequency="3" timeframe="100" level="
    11" >
    <if_matched_sid>100101</if_matched_sid>
    <description>frequency test: 3 in 100 sec</
       description>
</rule>
<rule id="100107" frequency="3" timeframe="100" level="
    11" >
    <if_matched_sid>100102</if_matched_sid>
    <description>frequency test: 3 in 100 sec</
       description>
```

```
</rule>
<rule id="100108" frequency="3" timeframe="100" level="
    11" >
     <if_matched_sid>100102</if_matched_sid>
     <description>frequency test: 3 in 100 sec</
        description>
</rule>
<rule id="100109" frequency="4" timeframe="100" level="
    11" >
     <if_matched_sid>100102</if_matched_sid>
     <description>frequency test: 4 in 100 sec</
        description>
</rule>
<rule id="100110" frequency="2" timeframe="2" level="11
    " >
     <if_matched_sid>100102</if_matched_sid>
     <description>frequency test: 2 in 2 sec</
        description>
</rule>
```

We will now present the constraints that determine priority and preemption between rules.

Encapsulation

If a rule a encapsulates a rule b, only a can be triggered. If rule c counts the number of occurrences of b and rule d counts the number of occurrences of a, only d can be triggered.

In the above example Listing 1.1, rule 100102 encapsulates 100101 then only 100102 can be triggered. Rule 100103 counts the number of occurrences of 100101, while rule 100107 counts the number of occurrences of 100102: therefore only 100107 can be triggered.

Frequency

If two rules a and b use the same `timeframe` to count the occurrences of the same rule c, rule a has frequency f_a and rule b has frequency f_b with $f_a < f_b$ then only a can be triggered.

In Listing 1.1, rule 100107 counts if rule 100102 is triggered *three* times in 100s, while rule 100109 counts if rule 100102 is triggered *four* times in 100s: therefore only 100107 can be triggered.

Order

If two rules a and b that count occurrences of a rule c with the same frequency f can be triggered (*i. e.,* conditions to trigger both rules are satisfied) and a is written before b in the configuration file then only a can be triggered, unless rule b has an incident level i_b greater than the incident level i_a of a.

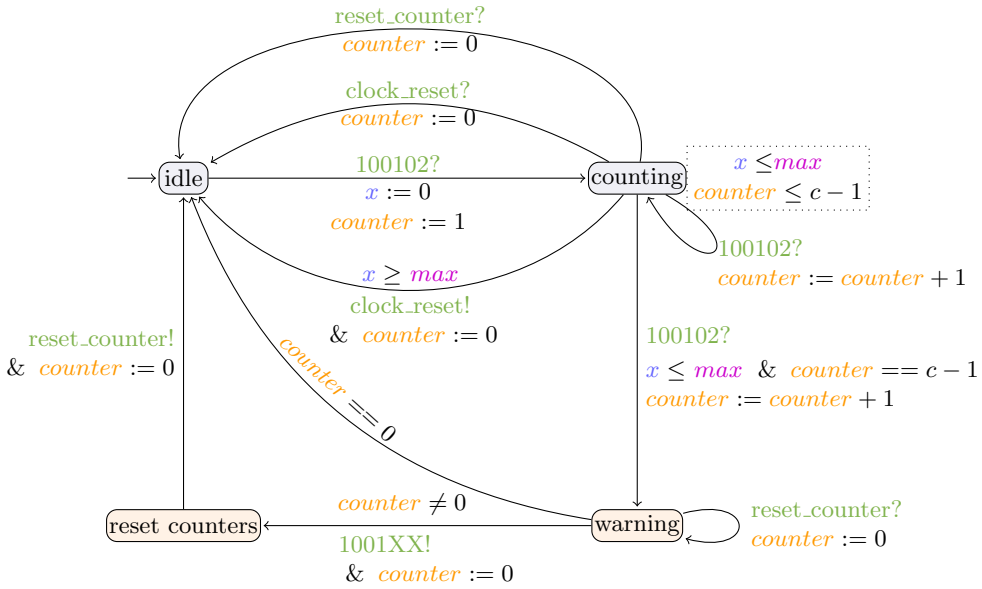


Fig. 1: WTA to model a rule with timeframe and frequency

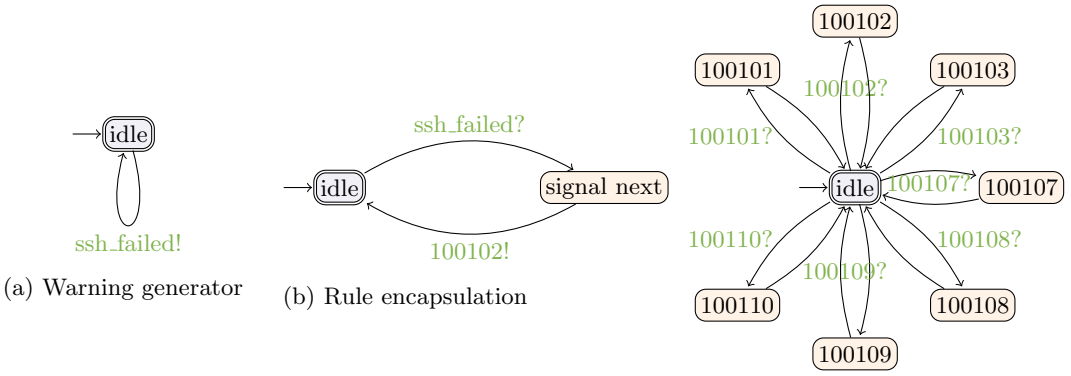
In Listing 1.1, rules 100107 and 100108 count if rule 100102 is triggered *three* times in 100ms, but rule 100107 is written before 100108: therefore only 100107 can be triggered as both 100107 and 100108 have `level="11"`. If 100108 had had a lower incident level (*e. g.,* `level="9"`) only 100108 could have been triggered.

5 Modeling rules with WTA

Multiple constraints and conditions, described in Section 4.2, impact the triggering or occurrence of a rule. Crafting and maintaining a cohesive set of rules in a complex environment with the possibility of multiple administrators can be an extremely challenging endeavor with unpredictable outcomes. In this section we propose a framework to model a set of Wazuh rules using WTA and the Uppaal model checker [BY03] that aims at preserving a robust set of rules in such a complex environment.

5.1 Clocks, channels and counters

By default we will use different counters (which are the weights of our WTA), and clocks for each automaton modeling a rule. If several rules count the occurrences of the same rules, the automata will share a `reset_counter` synchronization channel to reset all counters at once. If these rules use the same frequency, they will share a clock and a `clock_reset` synchronization channel.



(a) Warning generator (b) Rule encapsulation

(c) Rules selector, based on the declaration of priorities

Fig. 2: Rules contraints modeled as WTA

5.2 Declaration of rules priority

We declare in Uppaal the priority between rules, modeled as synchronization channels. In the left most position, we model encapsulation rules: rule 100102 encapsulates (therefore has priority over) 100101. In the right most position we model counter resets which are equivalently at the highest priority to take its transition before lower priority channels (we discuss this further in Section 5.6). In between we model rules that use the frequency and timeframe parameters written, if they use the same frequency, from right to left by incident level, and by order in the configuration file if incident levels are equal.

$$\text{chan priority } \underbrace{100101 < 100102}_{\text{encapsulation rules}} < \overbrace{100110 < 100109 < 100108 < 100107 < 100103}^{\text{rules using frequency and timeframe}} < \underbrace{reset\_counter}_{\texttt{reset\_counter}(s)} ;$$

Rule 100102 is encapsulated in 100108 but also in 100107 with the same frequency. The order in the configuration file gives priority to 100107, as they have equal incident levels (`level="11"`). The order of 100109 and 100110 does not influence the behaviour as they have different frequencies or timeframes. Optionally in the Wazuh configuration file we can ignore a rule in the encapsulation chain using the field `noalert="1"`; we simply remove the rule from the declaration chain.

5.3 Generator

This automaton in Fig. 2a, with a single transition, operates as a log generator under the name ssh_failed (rule 60122) in our example. It models a failed logon by an active user (*e. g.,* wrong password).

5.4 Rule encapsulation

This automaton in Fig. 2b models the priority according to encapsulation, as it receives the most deeply encapsulated rule (in Listing 1.1 it is 60122 ssh_failed) and returns the most exposed rule (100102 in Listing 1.1).

5.5 Rule selector

The automaton in Fig. 2c models the fact that if we reach a location 1001XX then the corresponding rule $1001XX$ is triggered in Wazuh. Otherwise if not reachable, it cannot be triggered in Wazuh. Using Uppaal, we analyse the reachability of all locations in this automata.

5.6 Rule with timeframe and frequency

In the following, we discuss the behaviour of the main automaton in Fig. 1 that models the rules with frequency and timeframe. max is the timeframe in seconds of the rule (*e. g.,* if `timeframe="100"`, max $= 100$), and c is the number of occurrences to count before the rule is triggered (*e. g.,* if `frequency="3"`, $c = 3$)

We start in the idle location. As soon as we receive 100102! from the rule encapsulation automaton (see Fig. 2b), the synchronization 100102? makes us move to the counting location; the *counter* is set to 1 as it is the first match and the clock x is reset to 0 to initiate the countdown to max.

From the counting location, we can return to idle and reset *counter* to 0 if the clock exceeds the threshold max of the timeframe. As several rules can count the occurrences (with possibly different frequencies) of the same rule in the same timeframe (see Section 4.2), we broadcast the clock_reset! message so the other automata that share the same channel clock_reset? also reset their *counter* to 0 and return to the idle location. Note that if two rules count the occurrences of the same rule with two different frequencies and timeframes, they share a `reset_counter` synchronization channel but not a `clock_reset` synchronization channel. When we exceed the smallest time threshold then its *counter* is reset and clock_reset! is broadcasted, but the other rule is not affected: the `clock_reset` synchronization channel is not shared so *counter* is not reset and it occurrences counting continues (see rules 100107 and 100110 in Listing 1.1).

While in the counting location we take the loop each time we receive 100102! from the rule encapsulation automaton and increase the *counter*.

If we receive our last 100102! without exceeding our timeframe max, *i. e.,* when *counter* $== c - 1$ and we receive 100102! (therefore we received exactly c messages), we move to the warning location (and set *counter* to c for coherence).

In the case where two rules that possibly have the same timeframe but especially two different frequencies, the one with the smallest frequency, which will be triggered, goes from counting to warning then to the reset counters location (*counter* > 0) and will broadcast reset_counter!. The other automaton

that has a higher frequency is still in the counting location (and the rule will not be triggered), receives reset_counter! and returns to idle.

In the other case, where two rules count the occurrences of the same rule with the same frequency, they arrive simultaneously in the warning location.

The automaton with the highest priority channel, as defined in the declarations (see Section 5.2), broadcasts its warning 1001XX! (used by the selector automaton in Fig. 2c to detect the triggered rule) and goes to reset counters.

Since the channel reset_counter has the highest priority (see Section 5.2, it immediately broadcasts reset_counter! before the any other automata take a transition transition and goes to idle location to be able to receive 100102! again and reset its *counter* to 0.

The other automata with lower priority in warning take the loop transition and reset their *counter* to 0 and immediately jump to the idle location (as *counter* == 0) without broadcasting a warning message.

5.7 Reachability analysis

Using Uppaal, we now can check whether the rules can be triggered by checking whether the state corresponding to the rule in the automata of Fig. 2c is reachable. Our analysis shows that only rules 100102, 100107 and 100110 can be triggered* in Wazuh for sequences of failed ssh logon by an active user.

6 Conclusion

In this work we proposed a framework to model Wazuh rules as WTA and verify that the rules can be triggered using Uppaal and reachability analysis to ensure the robustness of the set of rules.

Currently, from a set of Wazuh rules, knowing only the deepest encapsulated and the most exposed rules in an encapsulation chain, and the order in which rules are written in the configuration file to define the priority declarations, we are able to model the set of rules as WTA in Uppaal and automatically check whether a rule can be triggered. More than that it only takes a few lines in Uppaal to create and add a new rule to the model.

Future works. We aim to enhance our framework by enabling the automatic generation of Wazuh rules from the Uppaal model. This approach will allow us to create a robust set of rules that can be triggered when they are reachable within the Uppaal model. Additionally, we plan to apply machine learning techniques to suggest new rules based on analyzing potentially malicious log sets.

More broadly, we recognize that the current model is quite rigid and specifically tailored to Wazuh detection rules. We are interested in increasing its flexibility to adapt to similar solutions, making it more versatile and applicable to a wider range of security systems.

*The complete set of rules and the Uppaal model can be found at https://gricad-gitlab.univ-grenoble-alpes.fr/MRprojects/wazuhrulesuppaal/

Finally, a promising direction is the addition of timed and constraints parameters as in [ALRS21] where timed constraints and weights can have unknown values. This would empower our framework in order to model incomplete Wazuh rules.

References

AD94. Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, April 1994.

AL15. Zeeshan Afzal and Stefan Lindskog. Automated testing of IDS rules. In *IEEE International Conference on Software Testing, Verification and Validation Workshops*, pages 1–2. IEEE Computer Society, April 2015.

ALP04. Rajeev Alur, Salvatore La Torre, and George J. Pappas. Optimal paths in weighted timed automata. *Theoretical Computer Science*, 318(3):297–322, 2004.

ALRS21. Étienne André, Didier Lime, Mathias Ramparison, and Mariëlle Stoelinga. Parametric analyses of attack-fault trees. *Fundam. Informaticae*, 182(1):69–94, 2021.

BFH+01. Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim Guldstrand Larsen, Paul Pettersson, Judi Romijn, and Frits W. Vaandrager. Minimum-cost reachability for priced timed automata. In *HSCC*, volume 2034 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 2001.

BY03. Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In *Lectures on Concurrency and Petri Nets, Advances in Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 87–124. Springer, 2003.

NK20. Piyawat Noiprasong and Assadarat Khurat. An IDS rule redundancy verification. In *17th International Joint Conference on Computer Science and Software Engineering, JCSSE 2020, Bangkok, Thailand, November 4-6, 2020*, pages 110–115. IEEE, 2020.

Sur. Suricata. Suricata intrusion detection system. https://suricata.io/. Accessed: 2024-07-10.

Waza. Wazuh. Wazuh documentation: custom rules. https://documentation.wazuh.com/current/user-manual/ruleset/rules/custom.html. Accessed: 2024-06-30.

Wazb. Wazuh. Wazuh documentation: rules syntax. https://documentation.wazuh.com/current/user-manual/ruleset/ruleset-xml-syntax/rules.html. Accessed: 2024-06-30.

Wazc. Wazuh. Wazuh github rules. https://github.com/wazuh/wazuh-ruleset/blob/master/rules/0580-win-security_rules.xml. Accessed: 2024-06-30.

Waz22. Wazuh. Wazuh for GDPR white paper. https://wazuh.com/resources/Wazuh_GDPR_White_Paper.pdf, 2022. Accessed: 2024-06-30.