MONID

A Temporal Logic Based Framework for Intrusion Detection

Zanolin Lorenzo[1]

[1]DMIF
University of Udine

September 2023

UNIVERSITÀ
DEGLI STUDI
DI UDINE
hic sunt futura

Table of Contents

Introduction

Intrusion Detection

Intrusion detection means maintaining constant surveillance on a system in order to detect any misuse of these weak areas as soon as feasible so that they can be repaired.

There are three approaches:

- *signature-based*: aims to identify patterns and match them with known signs of intrusions;
- *anomaly-based*: can identify new attacks when it detects behavior that differs significantly from previously learned normal behavior;
- *hybrid*: combines the best of both worlds by looking at patterns and one-off events.

We will present MONID which is a *signature-based* intrusion detector.

What is MONID? I

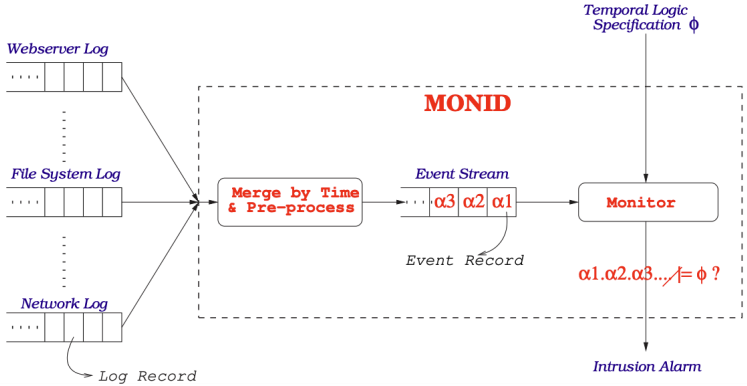
MONID is a prototype which can detect intrusions on a system and operates in both online and offline modes.

In order:

1. we will use the logic **EAGLE** to define intrusion patterns using temporal logic formula φ; in this case the monitored formula will be $\psi = \Box(\neg\varphi)$.
2. MONID will create a stream of events $\sigma = \alpha_1, \alpha_2, \ldots$ obtained from a merge of the logs by ascending time order;
3. a monitor will processes each event α_i as it happens and updates the monitored formula ψ to store a relevant summary;
4. an intrusion alarm is triggered if, for any reason, $\alpha_1, \alpha_2 \ldots \not\models \psi$.

What is MONID? II

The architecture is the following.



Assumptions

Two assumptions must be made:

1. There is a finite sequence of events called $\sigma = \alpha_1, \ldots, \alpha_n$ that is a merge of the system registered logs organized by ascending time. The structure of an event record α_i is the following:

   $$\texttt{LoginLogoutEvent}\{userId : \underline{\text{string}},\ action : \underline{\text{int}},\ time : \underline{\text{double}}\}$$

   An example of event could be:
   $\{userId : \text{"}\texttt{Lori}\text{"},\ action : \texttt{login},\ time : 20\}$
2. For each attack, there is a formula ψ which specifies the absence of it.

Now let us start on the basics of EAGLE.

EAGLE, a Temporal Monitoring Logic

Basics I

EAGLE offers a succinct but powerful set of primitives, supporting recursive parameterized equations with a minimal/maximal fix-point semantics together with three temporal operators: next-time ($\bigcirc$), previous-time ($\odot$), and concatenation ($\cdot$).

As a result, rules in EAGLE give us the power to create specific temporal operators as well as to bind and modify data. This property turns out to be crucial for succinctly expressing executions of attack-safe systems. EAGLE

operates with *finite trace* semantics, meaning it checks formula satisfaction only at the end of a trace. However, in intrusion detection where event sequences can be infinite, the goal is to trigger an alarm as soon as a property is violated, thus MONID continuously checks the formula's satisfaction status after each event.

Basics II

Let us start with an example.

Example 1

We want to express the property "*Whenever there is a login by any user x, then eventually the user x logs out*". In EAGLE we can do it with the following rules:

$$\underline{\textit{min}}\ \texttt{EvLogout}(\underline{\textit{string}}\ k) = (action = \texttt{logout} \wedge userId = k) \vee \bigcirc \texttt{EvLogout}(k)$$
$$\underline{\textit{mon}}\ M_2 = ((action = \texttt{login}) \rightarrow \texttt{EvLogout}(userId))$$

Once the rules are created, the monitor will evaluate and update the monitored formula M_2. A possibile trace $\sigma = \alpha_1, \alpha_2$, where:

$\alpha_1 = \{userId : \text{"}\texttt{Lori}\text{"},\ action : \texttt{login},\ time : 17.0\}$
$\alpha_2 = \{userId : \text{"}\texttt{Lori}\text{"},\ action : \texttt{logout},\ time : 150.0\}$

satisfies M_2.

Syntax I

Each specification S is made up of an observer part O and a declaration part D.

$$
\begin{aligned}
S &::= D\,O \\
D &::= R^* \\
O &::= M^* \\
R &::= \{\underline{\text{max}} \mid \underline{\text{min}}\}\ N(T_1\ x_1, \ldots, T_n\ x_n) = F \\
M &::= \underline{\text{mon}}\ N = F \\
T &::= \underline{\text{Form}} \mid \text{primitive type} \\
F &::= \text{expression} \mid \underline{\text{true}} \mid \underline{\text{false}} \mid \neg F \mid F_1 \wedge F_2 \mid F_1 \vee F_2 \mid \\
&\quad F_1 \rightarrow F_2 \mid \bigcirc F \mid \odot F \mid F_1 \cdot F_2 \mid N(F_1, \ldots, F_n) \mid x_i
\end{aligned}
$$

Let us focus on some details.

Syntax II

Definitions can start with different keywords:

- mon: specifies the EAGLE formulas to be monitored and cannot have a recursive definition; as already told, these kind of rules will evolve as new events appear.
- max: defines *safety properties* (nothing bad ever happens) and have a maximal interpretation.
- min: defines *liveness properties* (something good eventually happens) and have a minimal interpretation.

Note.
The difference between maximal and minimal interpretation becomes important only when we are evaluating the at the boundaries of a trace; max rules evaluates at true at initial and final istants, while min rules evaluates at false.

Semantics I

The semantics of the logic is defined in terms of the *satisfaction* relation $\models$ which defines whether a finite execution trace σ satisfies the specification $\varphi = D\,O$.

Given a trace σ and a specification $D\,O$, satisfaction is defined as follows:

$$\sigma \models D\,O \text{ iff } \forall(\underline{\text{mon}}\ N = F) \in O.\ \sigma, 1 \models_D\ F$$

In other words, a trace satisfies a specification if it fulfills each monitored formula (mon) when monitoring from position 1, which is the initial state.

In the following slide we will present the definition of the satisfaction relation $\models_D \subseteq (Trace \times \mathbb{N}) \times \underline{\text{Form}}$ for a set of rule definitions D.

Semantics II

$$
\begin{array}{lll}
\sigma, i \models_D \ expression & \text{iff} & 1 \leq i \leq |\sigma| \text{ and } evaluate(expression, \sigma(i)) = true \\
\sigma, i \models_D \ \underline{\text{true}} & & \\
\sigma, i \not\models_D \ \underline{\text{false}} & & \\
\sigma, i \models_D \neg F & \text{iff} & \sigma, i \not\models_D F \\
\sigma, i \models_D F_1 \wedge F_2 & \text{iff} & \sigma, i \models_D F_1 \text{ and } \sigma, i \models_D F_2 \\
\sigma, i \models_D F_1 \vee F_2 & \text{iff} & \sigma, i \models_D F_1 \text{ or } \sigma, i \models_D F_2 \\
\sigma, i \models_D F_1 \rightarrow F_2 & \text{iff} & \sigma, i \models_D F_1 \text{ implies } \sigma, i \models_D F_2 \\
\sigma, i \models_D \bigcirc F & \text{iff} & i \leq |\sigma| \text{ and } \sigma, i+1 \models_D F \\
\sigma, i \models_D \odot F & \text{iff} & 1 \leq i \text{ and } \sigma, i-1 \models_D F \\
\sigma, i \models_D F_1 \cdot F_2 & \text{iff} & \exists j \text{ s.t. } i \leq j \leq |\sigma|+1 \text{ and } \sigma^{[1,j-1]}, i \models_D F_1 \text{ and } \sigma^{[j,|\sigma|]}, 1 \models_D F_2 \\
\sigma, i \models_D N(\overline{F}, \overline{P}) & \text{iff} & \begin{cases} \text{if } 1 \leq i \leq |\sigma| \text{ then:} \\ \quad \sigma, i \models_D F[\overline{f} \mapsto \overline{F}, \overline{p} \mapsto \ evaluate(\overline{P}, \sigma(i))] \\ \quad \text{where } (N(\underline{\text{Form}}\ \overline{f}, \text{T}\ p) = F) \in D \\ \text{otherwise, if } i = 0 \text{ or } i = |\sigma|+1 \text{ then:} \\ \quad \text{rule } N \text{ is defined as } \underline{\text{max}} \text{ in } D \end{cases}
\end{array}
$$

Evaluation algorithm I

As we can see, a function called $evaluate$ emerges; there are additional functions as well. Let us list them.

- $evaluate$: Form $\times$ State. This function is used to evolve at every event the monitored formulas.
- $update$: Form $\times$ State $\rightarrow$ Form. This function is used by $evaluate$ to pre-evaluate a formula if it is guarded by the previous operator $\odot$
- $value$: Form $\rightarrow \{\text{true}, \text{false}\}$. This function returns true if and only if $\sigma, |\sigma| + 1 \models F$ (or $\sigma, 0 \models F$), and returns false otherwise.

Note.
Formally, a formula $F' = evaluate(F, \alpha_i)$ is created when a formula F is evaluated at an event $\alpha_i = \sigma(i)$ and has the condition that $\sigma, i \models F$ if and only if $\sigma, i + 1 \models F'$. With F being the evolved formula, we compute the boolean function $value(F)$ at the conclusion of the trace.

Evaluation algorithm II

Better details are in the paper; let us introduce an example.

Example 2

Given the following rules

$$\underline{\textit{max}}\ \texttt{Always}(\underline{\textit{Form}}\ F) = F \wedge \bigcirc\texttt{Always}(F)$$
$$\underline{\textit{min}}\ \texttt{EvTimedLogout}(\underline{\textit{string}}\ k, \underline{\textit{double}}\ t, \underline{\textit{double}}\ \delta) = (\textit{time} - t \leq \delta) \wedge ((action = \texttt{logout} \wedge userId = k) \vee \bigcirc\texttt{EvTimedLogout}(k, t, \delta))$$
$$\underline{\textit{mon}}\ M_3 = \texttt{Always}((action = \texttt{login}) \rightarrow \texttt{EvTimedLogout}(userId, time, 100))$$

and a trace $\sigma = \alpha_1, \alpha_2$*, where* $\alpha_1 = \{userId : \text{"Lori"},\ action : \texttt{login},\ time : 17.0\}$ *and* $\alpha_2 = \{userId : \text{"Lori"},\ action : \texttt{logout},\ time : 150.0\}$.
M_3 *gets modified, at first step, in*

$$evaluate(M_3, \alpha_1) = \texttt{EvTimedLogout}(\text{"Lori"}, 17.0, 100) \wedge \texttt{Always}((action = \texttt{login}) \rightarrow \texttt{EvTimedLogout}(userId, time, 100)).$$

Conclusions

Conclusions

References I

[1] Howard Barringer et al. “EAGLE can do Efficient LTL Monitoring”. In: *Fossacs Ortacas*. 2003.

[2] Howard Barringer et al. “Program monitoring with LTL in EAGLE”. In: *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings*. IEEE. 2004, p. 264.

[3] Howard Barringer et al. “Rule-based runtime verification”. In: *Verification, Model Checking, and Abstract Interpretation: 5th International Conference, VMCAI 2004 Venice, Italy, January 11-13, 2004 Proceedings 5*. Springer. 2004, pp. 44–57.

[4] Wei Gao and Thomas H Morris. “On cyber attacks and signature based intrusion detection for modbus based industrial control systems”. In: *Journal of Digital Forensics, Security and Law* 9.1 (2014), p. 3.

References II

[5] Akash Garg and Prachi Maheshwari. “A hybrid intrusion detection system: A review”. In: *2016 10th International Conference on Intelligent Systems and Control (ISCO)*. IEEE. 2016, pp. 1–5.

[6] VVRPV Jyothsna, Rama Prasad, and K Munivara Prasad. “A review of anomaly based intrusion detection systems”. In: *International Journal of Computer Applications* 28.7 (2011), pp. 26–35.

[7] Urupoj Kanlayasiri, Surasak Sanguanpong, and Wipa Jaratmanachot. “A rule-based approach for port scanning detection”. In: *Proceedings of the 23rd electrical engineering conference, Chiang Mai Thailand*. Citeseer. 2000, pp. 485–488.

References III

[8] John McHugh. “Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory”. In: *ACM Transactions on Information and System Security (TISSEC)* 3.4 (2000), pp. 262–294.

[9] Prasad Naldurg, Koushik Sen, and Prasanna Thati. “A temporal logic based framework for intrusion detection”. In: *Formal Techniques for Networked and Distributed Systems–FORTE 2004: 24th IFIP WG 6.1 International Conference, Madrid Spain, September 27-30, 2004. Proceedings 24*. Springer. 2004, pp. 359–376.

[10] Krerk Piromsopa and Richard J Enbody. “Buffer-overflow protection: the theory”. In: *2006 IEEE International Conference on Electro/Information Technology*. IEEE. 2006, pp. 454–458.

References IV

[11] Gholam Reza Zargar and Peyman Kabiri. “Identification of effective network features to detect Smurf attacks”. In: *2009 IEEE Student Conference on Research and Development (SCOReD)*. IEEE. 2009, pp. 49–52.

Thanks for the attention