Timed Automata Networks for SCADA Attacks Real-Time Mitigation

Fabio Martinelli*, Francesco Mercaldo*†, Antonella Santone†, Christina Tavolato-Wötzl‡, Paul Tavolato§
*Institute for Informatics and Telematics, National Research Council of Italy, Pisa, Italy
{fabio.martinelli, francesco.mercaldo}@iit.cnr.it
†Department of Bioscience and Territory, University of Molise, Pesche (IS), Italy
{francesco.mercaldo, antonella.santone}@unimol.it
‡MeteoServe, Vienna, Austria
c.tavolato@aon.at
§Department of Computer Science and Security, Saint Pölten University of Applied Sciences, Saint Pölten, Austria
paul.tavolato@fhstp.ac.at

***Abstract*—SCADA systems are nowadays widespread in critical infrastructures, from oil pipelines to chemical manufacturing plants: an attacker taking control of a SCADA system could cause a plethora of damages, both to the infrastructure but also to people. In this paper we propose a method to detect attacks targeting SCADA systems. We consider a model checking technique: we model time-series logs obtained from SCADA systems into a network of timed automata and, through timed temporal logic, we characterize the behaviour of a SCADA system under attack. Experiments performed on a SCADA gas distribution system confirmed the effectiveness of the proposed method.**

***Index Terms*—SCADA, model checking, formal methods, timed automaton, temporal logic, critical infrastructure, security, safety.**

I. Introduction and Related Work

Public gas distribution utilities have a need to monitor the performance of their pipe networks to ensure the correct delivery of gas, and to ensure pressures and flows are maintained for safety and other compliance reasons [1]. They also need to meter gas which is sold to customers. Oil companies have very long pipelines which traverse often very remote regions [2]. Pressures, temperature and other parameters along these pipelines need to be monitored regularly.

The process of getting gas from the well head (production source) to the refinery (processing facility) and ultimately to the consumer usually requires a large infrastructure of pipeline networks [2]. The advances made in gas distribution Supervisory Control and Data Acquisition (SCADA) technology have significantly reduced the cost for remote telemetry devices and the communications infrastructure required to retrieve data from these devices [1].

SCADA represents a control system architecture considering computers and networked data communications for high-level process supervisory management, using peripheral devices such as programmable logic controller [3].

The SCADA concept was developed as a universal means of remote access to a variety of local control modules, which could be from different manufacturers allowing access through standard automation protocols. In practice, large SCADA systems have grown to become very similar to distributed control systems in function, but using multiple means of interfacing with the plant. They can control large-scale processes that can include multiple sites, and work over large distances as well as small distance. In last years, SCADA systems have been subject to cyberattacks [4], [5]. For instance, a cyberattack on a shared data network forced four of the United States natural-gas pipeline operators to temporarily shut down computer communications with their customers in 2018[1]. This attack highlighted the vulnerability of the United States energy system: the increasing dependence of pipeline infrastructure on digital systems makes them a particularly ripe target. Control valves, pressure monitors and other equipment connected to wireless networks are vital to daily functions of everything from refineries to oil wells.

Furthermore, several popular pieces of malware have enjoyed success against SCADA system. Since Stuxnet was first observed, more and more malware attacking Scada systems appeared in the wild, mostly able to gather information and control the entire system [6].

With the aim to mitigate SCADA attacks, in the last years the research community has proposed several methods to detect this kind of attacks. For instance, authors in [7] generate models to characterize the acceptable behavior of a SCADA systems. Their approach detects attacks that cause the system to behave outside of the models. Considering that the developed rules are focused on TCP packets, they can detect only variations from the expected communication pattern (if the attacker generates the same communication pattern the attack will be undetected). Another method applying anomaly detection is the one proposed by Dayu and colleagues [8]. They consider features related to the CPU and to access to storage (i.e., files read and write operation) of the several computers interconnecting the SCADA systems. Periodicity to detect traffic anomalies, which represent potential intrusion

[1]https://www.nytimes.com/2018/04/04/business/energy-environment/pipeline-cyberattack.html

attempts, is proposed in [9]. They discuss an anomaly detection approach based on the observation that SCADA traffic is highly periodic.

As demonstrated by the overview on current literature, current research on attack detection for critical infrastructure mainly concentrates on protocol specific attacks: there is little consideration of high-level cyber attacks targeting SCADA based critical infrastructure system. Furthermore, there is lack of methods able to analyse characteristics specifically related to SCADA systems.

Guided by these motivations, in this paper we propose a method aimed to detect attacks targeting SCADA gas distribution systems exploiting high level features related to the SCADA infrastructure. We model a SCADA system as a network composed of timed automata and through timed temporal logic properties we express the systems behaviour. Thus, exploiting model checking techniques [10], we verify whether the designed properties are satisfied on the model. Considering that the properties are representing an attack on a SCADA system, if a property is verified, an attack is in progress on the analysed system.

The paper proceeds as follows: in the next section, we provide preliminary notions about timed automata and timed temporal logic; Section III describes the proposed method; Section IV discusses the performed experiment aimed to evaluate the method and, finally, in Section V conclusions and future research directions are drawn.

II. Background

Preliminary concepts about model checking with timed automata exploited in this paper are provided. To apply model checking techniques we need: (i) a Formal Model, (ii) a Temporal Logic and, (iii) a Formal Verification Environment.

Formal Model: Specification is used to give a formal description of the system which is being analysed. The formal specification considers a language with a mathematically defined syntax and semantics. In this paper we represent the system behaviour as a network of timed automaton i.e., a finite-state machine extended with clock variables [11]. A system is modelled as a network of timed automata in parallel. The model is extended with bounded discrete variables that are part of the state. A state of the system is defined by the location of all automata, the clock values and the values of the discrete variables. Automaton may fire an edge (i.e., perform a transition) separately or synchronise with another automaton with the aim to lead a new state.

We give basic definitions about the syntax and semantics for timed automata. We consider the following notation: C is a set of clocks and $B(C)$ is the set of conjunctions over simple conditions of the form $x \bowtie c$ or $x - y \bowtie c$, where x, $y \in C$, $c \in \mathbb{N}$ and $\bowtie \in \{ <, \leq, =, >, \geq \}$. A timed automaton is a finite directed graph annotated with conditions over and resets of non-negative real valued clocks [11].

Definition 1: Timed Automaton. A timed automaton is a tuple (L, l_0, C, A, E, I) where L is a set of locations with $l_0 \in L$, C is the set of clocks, A is a set of actions, co-actions and internal τ-action, $E \subseteq L \times A \times B(C) \times 2^c \times L$, is a set of edges between locations with an action, a guard and a set of clocks to be reset, and $I : L \to B(C)$ assigns invariants to locations.

Definition 2: Semantics of Timed Automaton: Let (L, l_0, C, A, E, I) be a timed automaton. The semantics is defined as a labelled transition system $\langle S, s_0, \to \rangle$, where $S \subseteq L \times \mathbb{R}^C$ is the set of states, $s_0 = (l_0, u_0)$ is the initial state, and $\to \subseteq S \times (\mathbb{R}_{\geq 0} \cup A) \times S$ is the transition relation such that:

- $(l, u) \xrightarrow{d} (l, u + d)$ if $\forall d' : 0 \leq d' \leq d \implies u + d' \in I(l)$, and
- $(l, u) \xrightarrow{a} (l', u')$ if there exists $e = (l, a, g, r, l') \in E$ $s.t.$ $u \in g, u' = [r \mapsto 0]u$, and $u' \in I(l')$,

where for $d \in \mathbb{R}_{\geq 0}$, $u + d$ maps each clock x in C to the value $u(x) + d$, and $[r \mapsto 0]u$ denotes the clock valuation which maps each clock in r to 0 and agrees with u over $C \backslash r$. Timed automata are often composed of a network of timed automata over a common set of clocks and actions, consisting of n timed automata $\mathcal{A}_i = (L_i, l_i^0, C, A, E_i, I_i)$, $1 \leq i \leq n$. A location vector is a vector $\bar{l} = (l_1^0, \ldots, l_n^0)$. The invariant functions is composed into a common function over location vectors $I(\bar{l}) = \wedge_i I_i(l_i)$.

Below we itemize the timed automata distinctive elements:

- *Guard*: a particular expression aimed to verify a boolean expression evaluated on edges;
- *Invariant*: a condition that indicates the time that can be spent on a node;
- *Channel*: considered for synchronizing the progress of two or more automata.

Temporal Logic: to define properties we need a precise notation. We consider the Timed Temporal Logic (TCTL) [11], which extends the classical untimed branching time logic CTL with time constraints on modalities.

The TCTL syntax is defined by the following grammar:

$$\phi \quad ::= \quad a \mid \neg\phi \mid \phi \vee \phi \mid E\phi \; U_I \phi \mid A\phi \; U_I \phi$$

where $a \in AP$ (we denote with AP a set of atomic propositions), and I is an interval of $\mathbb{R}_+$ with integral bounds.

There are two possible semantics for TCTL, one which is said *continuous*, and the other one which is more discrete and is said *pointwise*. We consider the second one i.e., the *pointwise* semantic:

where $\varrho[\pi]$ is the state of ϱ at position π, and duration $\varrho \leq_\pi$ is the prefix of ϱ ending at position π, and duration($\varrho \leq_\pi$) is the sum of all delays along ϱ up to position π.

In the pointwise semantics, a position in a run:

$$\varrho = s_0 \xrightarrow{\tau_1, e_1} s_1 \xrightarrow{\tau_2, e_2} s_2 \ldots s_{n-1} \xrightarrow{\tau_n, e_n} s_n$$

is an integer i and the corresponding state s_i. In this semantics, formulas are checked only right after a discrete action has been done. Sometimes, the pointwise semantics is given in terms of actions and timed words, but that does not change anything.

As usually in CTL, TCTL: $\texttt{tt} \equiv a \vee \neg\, a$ standing for true, $\texttt{ff} \equiv \neg\, \texttt{tt}$ standing for false, the implication $\phi \to \psi \equiv (\neg\, \phi$

$$(l,u) \models a \iff a \in \mathcal{L}(l)$$

$$(l,u) \models a \neg\phi (l,v) \not\models \phi$$

$$(l,u) \models \phi \vee \psi \iff (l,u) \models \phi \; or \; (l,u) \models \psi$$

$$(l,v) \models E\phi \; U_I \psi \iff there\ is\ an\ infinite\ run\ \varrho\ in\ \mathcal{A}\ from\ (l,u)\ such\ that\ \varrho \models \phi \; U_I \psi$$

$$(l,u) \models A\phi \; U_I \phi \iff any\ infinite\ run\ \varrho\ in\ \mathcal{A}\ from\ (l,u)\ is\ such\ that\ \varrho \models \phi U_I \psi$$

$$\varrho \models \phi U_I \psi \iff there\ exists\ a\ position\ \pi > 0\ along\ \varrho\ such\ that\ \varrho[\pi] \models \psi,\ for\ every\ position\ 0 < \pi' < \pi, \varrho[\pi'] \models \psi,\ and\ duration(\varrho \leq_\pi) \in I.$$

TABLE I: Timed Temporal Logic Semantics.

$\vee \psi$), the eventually operator $F_I\phi \equiv$ tt $U_I\phi$ and the globally operator $G_I\phi \equiv \neg\ (F_I\ \neg\ \phi)$.

Formal Verification Environment: once defined the model and the temporal logic properties, we need something enabling us to check whether the timed automata based model satisfies the defined properties. To this aim we consider formal verification, a system process exploiting mathematical reasoning to verify if a system (i.e., the model) satisfies some requirement (i.e., the timed temporal properties).

Several verification techniques were proposed in last years, in this paper model checking [12] is considered.

With model checking the properties are formulated in temporal logic: each property is evaluated against the system. The model checker accepts as input a model and a property, it returns "true" whether the system satisfies the formula and "false" otherwise. The performed check is an exhaustive state space search that is guaranteed to terminate since the model is finite. In this paper we consider as model checker UPPAAL[2], an integrated tool environment for modeling, validation and verification of real-time systems modeled as timed automata networks.

III. The Method

The aim of the proposed method is to detect attacks on SCADA gas distribution networks. In detail we consider two features, the *pressure* measurement and the *pump* (i.e., how much fluid it is pumping through the pipe).

The proposed approach consists in two main phases: the *Formal Model Creation* (Figure 1) and the *Formal Model Verification* (Figure 4).

The *Formal Model Creation* phase output is the formal model representing the SCADA system.

From the SCADA system under analysis (i.e., *gas distribution system* in Figure 1) and the technical report (i.e., *technical report* in Figure 1), containing the *pressure* and the *pump* feature values, a technician manually marks the specific log as *attack*. These information i.e., the features gathered from the SCADA gas distribution system and the label to mark a specific trace as an *attack* are the input for the *control information* stored in textual files, containing the the *pressure* and the *pump* measurements at a fixed time interval (for instance, every millisecond).

[2]http://www.uppaal.org/

Time	F_1	F_2
t_1	*u*	*u*
t_2	*u*	*l*
t_3	*u*	*l*
t_4	*b*	*u*
t_5	*l*	*b*
t_6	*u*	*b*

TABLE II: Example of feature fragmentation with three intervals (u = *Up*, b = *Basal* and l = *Low*).

The next step of *Formal Model Creation* is the *Discretisation*, aimed to discretize each tank level feature. The registered *pressure* and *pump* continuous values are divided into three intervals, i.e., we map the numeric feature values into one of the following classes: *Up*, *Basal*, and *Low*. There are several methods proposed by the research community to discretize continuous values, we resort to the one proposed by authors in [13]: basically this method divides the features in three intervals: *Low*, *Basal* and *Up*. We consider the equal-width partitioning dividing the values of a given attribute into 3 intervals of equal size. The width of the interval is computed using the following formula: $W = (Max - Min)/3$, where Max and Min are respectively the maximum and the minimum values achieved by the feature. The equal-width partitioning has been applied to any feature under analysis. Furthermore, each discretised feature previously obtained, is converted into a timed automaton (i.e., *formal model* in Figure 1). To better understand the adopted discretisation method, Table II shows an example of discretised feature fragment.

The first column (i.e., *Time*) indicates the interval time (in the example $1 \leq t \leq 6$), while the F_1 and F_2 columns are respectively related to the two considered features. For instance, at the t_3 interval time, the F_1 exhibits the u value, while the F_2 exhibits the l one.

Starting from the feature discretisation, the next step of *Formal Model Creation* is the *formal model*. In this step a network of timed automata is generated (i.e., the *formal model* in Figure 1): in detail for each discretised feature an automaton is built. Resuming the feature fragmentation example provided in Table II, whether the same value is repeated between consecutive temporal instances the automaton will contain a loop: the automaton generated from the F_1 feature will contain a loop for the t_1, t_2 and t_3 time intervals (the repeated value is *u*), while the automaton generated from the F_2 features will contain two loops, the first one for the t_2 and t_3 time intervals

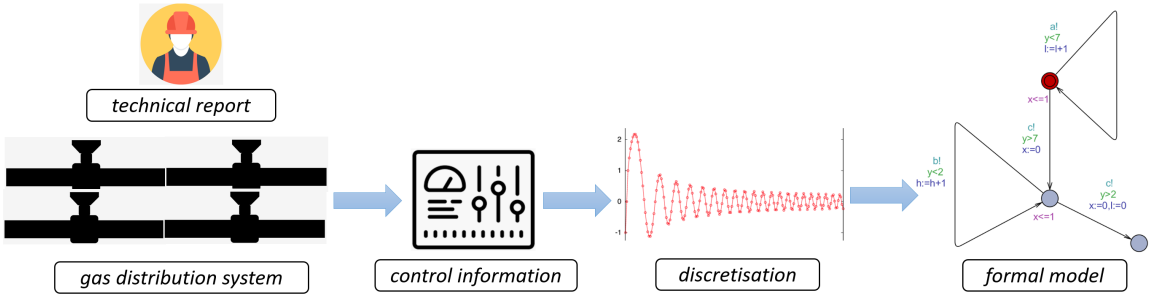

Fig. 1: Formal Model Creation.

(the repeated value is l) and the second one for the t_5 and t_6 time intervals (the repeated value is b). The exit condition from the loops is guaranteed by a guard, while the entering one is guaranteed from an invariant. In detail, two different clocks are considered for each automaton: the first one (i.e., x) to respect the loop entering condition, while the second one (i.e., y) to respect the exit one.

Furthermore, each automaton locally stores the count of u, b and l values. The variables related to the F_x automaton are marked with a subscript x: considering two features, $x \in \{1, 2\}$. Only the channel (i.e., s), allowing synchronisation between automata, is not stored locally: in fact it must guarantee the continuous and progressive automata advancement. One sender automata (i.e., $s!$) can synchronise with an arbitrary number of receivers automaton (i.e., $s?$). In practice, considering that each line of the discretized features in Table II corresponds to the value of the features in the same time interval, the synchronization allows to switch between a time interval to the next obliging the automata to go ahead with the next transition and to update the values of the features with the values related to the next time interval. This mechanism avoids inconsistencies between the values of the features and the time intervals.

Figures 2 and 3 show the automaton respectively obtained from the F_1 and F_2 discretisation.

For each loop we note the presence of a guard, furthermore the two automata are synchronized by using the s channel.

The enabled transitions for the for the F_1 and F_2 automata are shown in Table III.

Trans.	F_1 *automaton*					F_2 *automaton*				
	node	u_1	b_1	l_1	y_1	*node*	u_2	b_2	l_2	y_2
1	1	1	0	0	< 3	1	1	0	0	–
2	1	2	0	0	< 3	2	0	0	1	< 2
3	1	3	0	0	< 3	2	0	0	2	< 2
4	2	0	1	0	> 3	3	1	0	0	> 2
5	3	0	0	1	–	3	0	1	0	< 2
6	4	1	0	0	–	3	0	2	0	< 2
7	4	1	0	0	–	4	0	2	0	> 2

TABLE III: Enabled transitions for F_1 and F_2 automata.

As shown in Table III, the F_1 automaton is iterating in the loop (node 1 in Figure 2) for three time intervals (i.e., $y_1 < 3$), while the F_2 automaton after the increment of the u_1 variable (node 1 in Figure 3) is iterating for two time intervals (i.e., $y_2 < 2$). Subsequently, the F_1 automaton does not exhibit any loops, while the F_2 automaton is iterating for two time intervals in the loop in the node 3 in Figure 3 and then it continues with the last node.

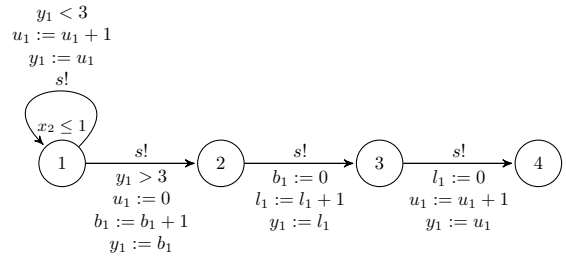

Fig. 2: The F_1 automaton.

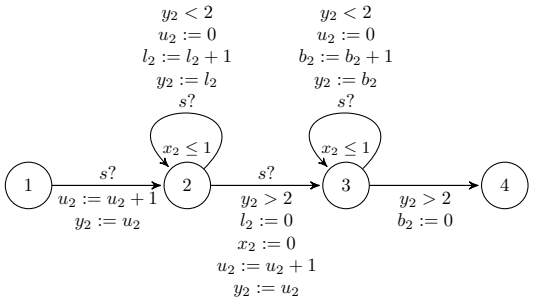

Fig. 3: The F_2 automaton.

Once generated the network of timed automata (i.e., the *formal model*) aimed to model the SCADA systems, the *Formal Model Verification* phase (Figure 4) is applied to check whether the the modeled SCADA system is under attack.

The *Formal Model Verification* receives as input a network of timed automata (*formal model* in Figure 4) built in the previous phase and a set of timed temporal logic properties. In detail two timed temporal logic properties are considered (Table *formulae* in Figure 4) related to two different possible SCADA attacks. The first one, identified by the ϕ formula represent a *malicious injection* (*MI*) i.e., malicious code that, once injected and executed, it is able to overload the distribution system [14]), while the second one, i.e., ψ, is the *denial of service* attack (*DoS*) i.e., a service interruption [1]).

The set of logic properties is checked against the network of

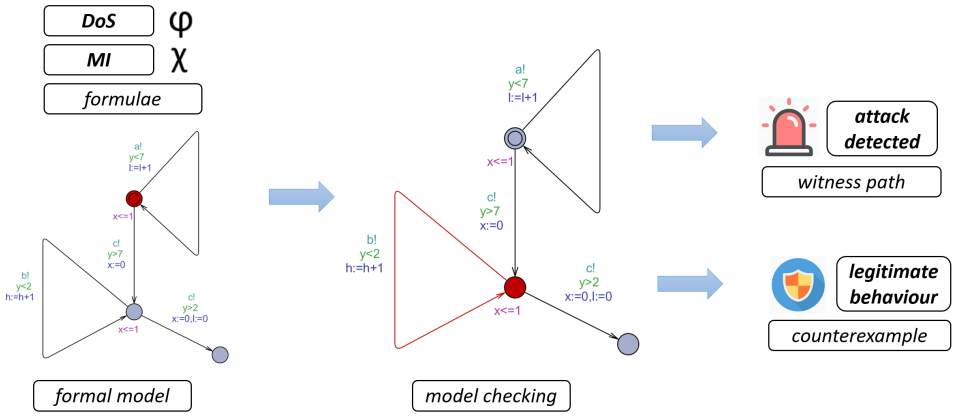

Fig. 4: Formal Model Verification.

timed automata obtained (*model checking* in Figure 4) using the UPPAAL model checker. Whether the UPPAAL formal verification environment outputs true when verifying a timed temporal logic property on a network of timed automata, it means that the proposed method labelled the formal model as belonging to the attack specified by the analysed formula (*DoS* whether the evaluated formula is ϕ or *MI* whether the evaluated formula is χ). Otherwise, the formal verification environment outputs false, meaning that the model under analysis is not belonging to the attacks described by the formula.

IV. The Evaluation

In this section we respectively describe the dataset considered to evaluate the proposed method, the timed temporal logic formulae aimed to identify attacks targeting SCADA systems and the experiment we performed to demonstrate the effectiveness of the proposed formal approach.

A. The Dataset

To evaluate the proposed method, a freely available dataset for research purpose[3] is considered[14]. The dataset contains the log (stored in an ARFF file) of a SCADA system related to a gas pipeline distribution system. The datasets contain examples of both normal activity and several attacks. The dataset contains times series logs composed from *pump* and *pressure* measurements, where remote data collection from sensors is enabled, in which there are measurements under *legitimate* operating conditions and when a *DoS* or *MI* attack is in progress. The dataset contains 269.228 *pump* and *pressure* measurements (one for each millisecond). We consider a time-window equal to 100 milliseconds, in this way 2.692 models are obtained (each model with 100 transitions): 1346 models are marked with the *legitimate* label, while 1346 are marked as attacks (in detail 673 are marked as *DoS* attacks, while the remaining 673 are *MI* attacks).

[3]https://sites.google.com/a/uah.edu/tommy-morris-uah/ics-data-sets

$E\langle\rangle\,\varphi$, *where* φ	=	$l_1 \geq 7 \wedge l_2 \geq 12$
$E\langle\rangle\,\varphi$, *where* χ	=	$h_1 \geq 9 \wedge h_2 \geq 14$
$E\langle\rangle\,\varphi \wedge \chi$		

TABLE IV: Timed temporal logic formula for the *DoS* and the *MI* SCADA attack detection.

B. The Formulae

In this section we present the formulae characterizing attacks on SCADA systems.

Table IV shows the the *DoS* and *MI* formulae.

In detail the φ formula, related to the *DoS* attack, is able to verify whether a *denial of service* is in progress (i.e., when the *pump* feature exhibits a *low* value at least for 7 times and the *pressure* feature exhibits a *low* value at least for 12 times). The *malicious injection* attack, described by the χ formula, is able to verify whether a *malicious code injection* is in progress (i.e., when the *pump* level exhibits a *up* value at least for 9 times and the *pressure* feature exhibits an *up* for at least 14 times). We want to verify whether the φ and χ formulae, possibly, can be satisfied by a reachable state (the so-called *reachability* property). We express that some state satisfying φ should be reachable using the path formula $E\langle\rangle\,\varphi$. Similar considerations can be done for the χ formula. We also provide the formula expressing that can occur at the same time *DoS* and *malicious injection* attacks (i.e., $E\langle\rangle\,\varphi \wedge \chi$). The formulae were generated with domain experts help by looking the data.

C. Experimental Results and Discussion

Four different metrics were used to evaluate the performance of the proposed approach: Precision, Recall, F-Measure and Accuracy.

The precision has been computed as the proportion of the observations that truly belong to investigated logs among all those which were assigned to the specific attack. It is the ratio of the number of relevant records retrieved to the total number of irrelevant and relevant records retrieved:

$Precision = \frac{tp}{tp+fp}$

where *tp* indicates the number of true positives (for instance, evaluating the φ formula, this value represents the number of models whose related *DoS* model is correctly labelled as *true* by the formal verification environment) and *fp* indicates the number of false positives (for instance, evaluating the φ formula, this value represents the number of models whose related *legitimate* model is wrongly labelled as *true* by the formal verification environment).

The recall has been computed as the proportion of attacks that were assigned to a given class, among all the attacks that truly belong to the class. It is the ratio of the number of relevant records retrieved to the total number of relevant records:

$Recall = \frac{tp}{tp+fn}$

where *tp* indicates the number of true positives and *fn* indicates the number of false negatives (for instance, whether we are evaluating the φ formula, this value represents the number of one day log whose related *DoS* model is wrongly labelled as *false* by the formal verification environment).

The F-Measure is a measure of test's accuracy. This score can be interpreted as a weighted average of the precision and recall:

$F\text{-}Measure = 2 * \frac{Precision*Recall}{Precision+Recall}$

The Accuracy is the fraction of the model correctly identified and it is computed as the sum of true positives and negatives divided by all the evaluated models:

$Accuracy = \frac{tp+tn}{tp+fn+fp+tn}$

where *tn* indicates the number of true negatives (for instance, evaluating the φ formula, this value represents the number *legitimate* models correctly labelled as *false* by the formal verification environment).

Precision	Recall	F-Measure	Accuracy	Attack
1	**1**	**1**	**1**	*DoS*
1	**1**	**1**	**1**	*MI*

TABLE V: Performance results.

The evaluation results shown in Table V are promising: a precision and recall equal to 1 are obtained, symptomatic that the model checker correctly outputs *true* when analysing all the *attack* models and correctly outputs *false* when analysing all the *legitimate* models.

V. Conclusion and Future Work

In this paper we propose a model checking based method to detect *denial of service* and *malicious injection* attacks on a SCADA system. The obtained performances are encouraging, in fact we reach a precision and a recall equal to 1, confirming the ability of the proposed method to rightly discern between the considered attacks and normal behaviour. As future work, we will investigate whether it is possible to automatically learn attack properties from SCADA system logs, as already demonstrated in biology [15], [16] and security [17], [18], [19].

Acknowledgments

This work has been partially supported by H2020 EU-funded projects SPARTA contract 830892 and C3ISP and EIT-Digital Project HII and PRIN "Governing Adaptive and Unplanned Systems of Systems" and the EU project CyberSure 734815.

References

[1] J. F. Lea Jr and L. Rowlan, *Gas well deliquification*. Gulf Professional Publishing, 2019.

[2] J. L. Kennedy, *Oil and gas pipeline fundamentals*. Pennwell books, 1993.

[3] S. A. Boyer, *SCADA: supervisory control and data acquisition*. International Society of Automation, 2009.

[4] V. M. Igure, S. A. Laughter, and R. D. Williams, "Security issues in scada networks," *computers & security*, vol. 25, no. 7, pp. 498–506, 2006.

[5] A. Belqruch and A. Maach, "Scada security using ssh honeypot," in *Proceedings of the 2nd International Conference on Networking, Information Systems & Security*, p. 2, ACM, 2019.

[6] R. Langner, "Stuxnet: Dissecting a cyberwarfare weapon," *IEEE Security & Privacy*, vol. 9, no. 3, pp. 49–51, 2011.

[7] S. Cheung, B. Dutertre, M. Fong, U. Lindqvist, K. Skinner, and A. Valdes, "Using model-based intrusion detection for scada networks," in *Proceedings of the SCADA security scientific symposium*, vol. 46, pp. 1–12, Citeseer, 2007.

[8] D. Yang, A. Usynin, and J. W. Hines, "Anomaly-based intrusion detection for scada systems," in *5th intl. topical meeting on nuclear plant instrumentation, control and human machine interface technologies (npic&hmit 05)*, pp. 12–16, 2006.

[9] R. R. R. Barbosa, R. Sadre, and A. Pras, "Towards periodicity based anomaly detection in scada networks," in *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012)*, pp. 1–4, IEEE, 2012.

[10] A. Santone and G. Vaglini, "Abstract reduction in directed model checking ccs processes," *Acta Informatica*, vol. 49, no. 5, pp. 313–341, 2012.

[11] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical computer science*, vol. 126, no. 2, pp. 183–235, 1994.

[12] N. D. Francesco, G. Lettieri, A. Santone, and G. Vaglini, "Heuristic search for equivalence checking," *Software and System Modeling*, vol. 15, no. 2, pp. 513–530, 2016.

[13] J. Dougherty, R. Kohavi, and M. Sahami, "Supervised and unsupervised discretization of continuous features," in *Machine Learning Proceedings 1995*, pp. 194–202, Elsevier, 1995.

[14] T. H. Morris and W. Gao, "Industrial control system cyber attacks," in *Proceedings of the 1st International Symposium on ICS & SCADA Cyber Security Research*, pp. 22–29, 2013.

[15] M. Ceccarelli, L. Cerulo, and A. Santone, "De novo reconstruction of gene regulatory networks from time series data, an approach based on formal methods," *Methods*, vol. 69, no. 3, pp. 298–305, 2014.

[16] G. Ruvo, V. Nardone, A. Santone, M. Ceccarelli, and L. Cerulo, "Infer gene regulatory networks from time series data with probabilistic model checking," pp. 26–32, 2015.

[17] F. Martinelli, F. Mercaldo, V. Nardone, and A. Santone, "Car hacking identification through fuzzy logic algorithms," in *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pp. 1–7, IEEE, 2017.

[18] G. Canfora, F. Mercaldo, G. Moriano, and C. A. Visaggio, "Composition-malware: building android malware at run time," in *2015 10th International Conference on Availability, Reliability and Security*, pp. 318–326, IEEE, 2015.

[19] F. Mercaldo, C. A. Visaggio, G. Canfora, and A. Cimitile, "Mobile malware detection in the real world," in *Proceedings of the 38th International Conference on Software Engineering Companion*, pp. 744–746, ACM, 2016.