Real-Time SCADA Attack Detection by means of Formal Methods

Fabio Martinelli[*], Francesco Mercaldo[*][†], Antonella Santone[†]
[*]Istituto di Informatica e Telematica, Consiglio Nazionale delle Ricerche, 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

***Abstract*—SCADA control systems use programmable logic controller to interface with critical machines. SCADA systems are used in critical infrastructures, for instance, to control smart grid, oil pipelines, water distribution and chemical manufacturing plants: an attacker taking control of a SCADA system could cause various damages, both to the infrastructure but also to people (for instance, adding chemical substances into a water distribution systems). In this paper we propose a method to detect attacks targeting SCADA systems. We exploit model checking, in detail we model logs 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 water 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

The Supervisory Control and Data Acquisition (SCADA) is a control system architecture considering computers and networked data communications for high-level process supervisory management, using peripheral devices such as programmable logic controller (PLC) [1].

There are concerns about SCADA systems being vulnerable to cyber attacks [2]. In last years, for instance, water distribution systems rely on computers, sensors and actuators for both monitoring and operational purposes. This combination of physical processes and embedded systems improves the level of service of water distribution networks but exposes them to the potential threats of cyber attacks. During the past decade, several water supply and distribution systems have been attacked, with the consequent creation of cyber security agencies and international partnerships to defend water networks [3].

For instance, in 2016 a group infiltrated in a water utility control system and changed the levels of chemicals being used to treat tap water[1]. This system, which was connected to the internet, managed PLCs that regulated valves and ducts that controlled the flow of water and chemicals used to treat it through the system.

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 [4] 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). Kim et al. [5] propose the application of anomaly detection techniques to detect failures in wind turbines. They adopt a machine learning based approach, considering as feature the sampling frequency of SCADA data. Another method applying anomaly detection is the one proposed by Dayu and colleagues [6]. 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 attempts, is proposed in [7]. 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 an SCADA based critical infrastructure system. Furthermore, there is lack of methods able to analyse characteristics specifically related to the SCADA systems.

Guided by these motivations, in this paper we propose a method aimed to detect attacks targeting SCADA water 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 system behaviour. Thus, exploiting model checking technique, 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.

[1]https://www.theregister.co.uk/2016/03/24/water_utility_hacked/

II. Background

Preliminary concepts about the formal methods technique considered in this paper are provided in this section i.e., the model checking timed automata. To apply model checking technique we resort to: (i) Formal Model, (ii) Temporal Logic and, (iii) 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 [8]. 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 the basic definition about the syntax and semantics for the basic 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 [8].

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 $I : L \rightarrow B(C)$ 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 $\langle S, s_0, \rightarrow \rangle$, where $S \subseteq L \times \mathbb{R}^C$ is the set of states, $s_0 = (l_0, u_0)$ is the initial state, and $\rightarrow \subseteq S \times (\mathbb{R}_{\geq 0} \cup A) \times S$ 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 into 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) [8], 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 it 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 $\phi \rightarrow \psi \equiv (\neg\ \phi \vee \psi)$, the eventually operator $F_I \phi \equiv \text{tt}\ U_I \phi$ 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 [9] is considered.

In the model checking technique 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 UPPAL[2], an integrated tool environment for modeling, validation and verification of real-time systems modeled as timed automata networks.

III. The Method

The goal of the proposed method is to detect attacks on SCADA water distribution systems. Water consumption in distribution systems is usually fairly regular throughout the

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

$$(l,u) \models a \iff a \in \mathcal{L}(l)$$
$$(l,u) \models a \neg \phi (l,v) \not\models \phi$$
$$(l,u) \models \phi \vee \psi \iff (l,u) \models \phi \; or \; (l,u) \models \psi$$
$$(l,v) \models E\phi \; U_I \psi \iff there \; is \; an \; infinite \; run \; \varrho \; in \; \mathcal{A} \; from \; (l,u) \; such \; that \; \varrho \; \models \; \phi \; U_I \psi$$
$$(l,u) \models A\phi \; U_I \phi \iff any \; infinite \; run \; \varrho \; in \; \mathcal{A} \; from \; (l,u) \; is \; such \; that \; \varrho \models \phi U_I \psi$$
$$\varrho \models \phi U_I \psi \iff there \; exists \; a \; position \; \pi > 0 \; along \; \varrho \; such \; that \; \varrho[\pi] \models \psi, \; for \; every \; position \; 0 < \pi' < \pi, \varrho[\pi'] \models \psi, \; and \; duration(\varrho \leq_\pi) \in I.$$

TABLE I: Timed Temporal Logic Semantics.

year with no seasonal variations. Water storage and distribution across the demand nodes are guaranteed by a series tanks, whose water levels trigger the operations of valves and pumps.

We consider the tank level measurements as features, in detail whether the SCADA water distribution systems is formed by two tanks i.e., two features describing the water measurement in the two tanks are considered.

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., *water distribution system* in Figure 1) and the technical report (i.e., *post mortem technical report* in Figure 1), containing the water level analysis during the day, a technician manually marks the specific day log as *attack*. These information i.e., the features gathered from the tank levels and the label to mark a specific trace as an *attack* are the input for the *one day log* stored in CSV files, containing the tank measurements for one day at fixed interval time (i.e., one hour).

The next step of the *Formal Model Creation* is the *Discretisation*, aimed to discretize each tank level feature. The registered tank level 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 method proposed by research community to discretize continuous values, we resort to the one proposed by authors in [10]: 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.

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*).

Starting from the feature discretisation, the next step of the *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 (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.

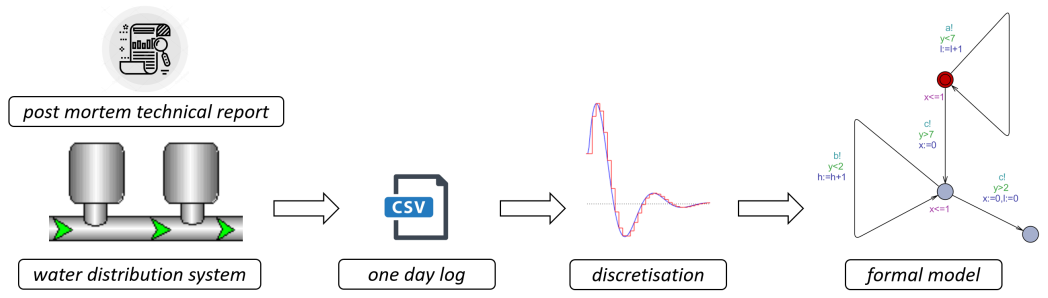

Fig. 1: Formal Model Creation.

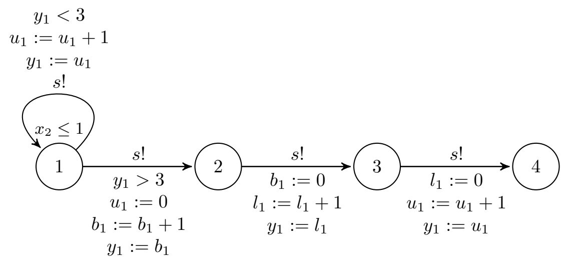

Fig. 2: The F_1 automaton.

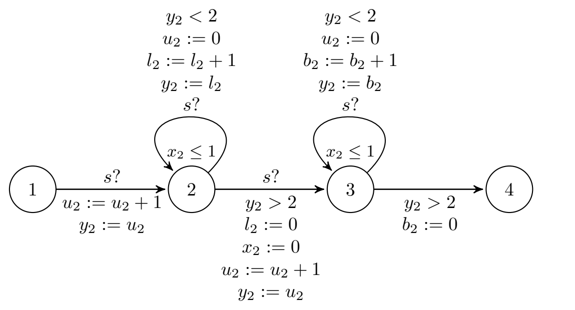

Fig. 3: The F_2 automaton.

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.

Once generated the network of timed automata (i.e., the *formal model*) aimed to model SCADA systems, the *Formal Model Verification* phase (Figure 4) is aimed to checking 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 is related to the overflow attack, while the second one, i.e., ψ, is related to the underflow attack (both *overflow* and *underflow* in the tank measurements can be symptomatic of a cyber attack in progress [3]).

The set of logic properties is checked against the network of timed automata obtained (*model checking* in Figure 4) using the UPPAL model checker. Whether the UPPAL formal verification environment outputs true when is 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 (*overflow* whether the evaluated formula is ϕ or *underflow* 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 in the analysed 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. The dataset contains the

[3]http://www.batadal.net

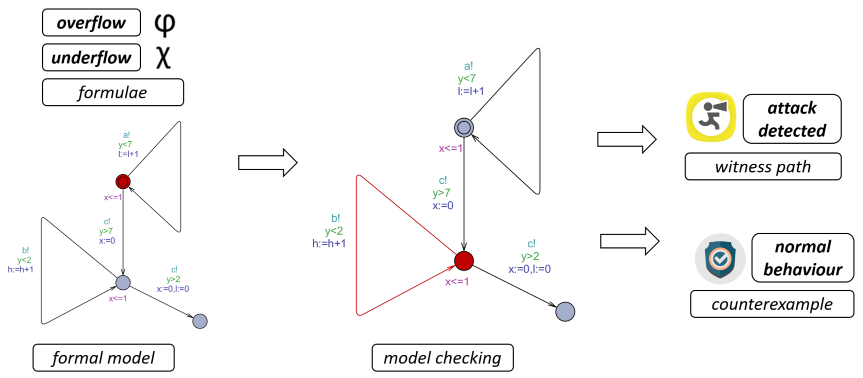

Fig. 4: Formal Model Verification.

$$E\langle\rangle\,\varphi\ ,\ where\ \ \varphi\ =\ h_1 \geq 11$$
$$E\langle\rangle\,\varphi\ ,\ where\ \ \chi\ =\ l_2 \geq 9$$
$$E\langle\rangle\,\varphi\ \wedge\ \chi$$

TABLE IV: Timed temporal logic formula for *overflow* and *underflow* SCADA attack detection.

log (stored in CSV files) of a SCADA system related to a modern water distribution system. The water consumption is fairly regular with no seasonal variations. Water storage and distribution across the demand nodes is guaranteed by two tanks, whose water levels trigger the operations of one valve and pumps distributed in several pumping stations. The dataset contains one day logs composed from the water levels of the two tanks (i.e., *lt1* and *lt2*), where remote data collection from sensors is enabled, in which there are hourly water measurements under *legitimate* operating conditions and when an *overflow* or *underflow* attacks targeting, respectively, the *lt1* and *lt2* tanks is happening. We consider logs related to 20 days of measurements: 10 day with the *overflow* attack on *lt1* and with the *underflow* attack on *lt2* (attacks have happened each day), while the remaining 10 days without attacks. For each day log a timed automata network is built.

B. The Formulae

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

Table IV shows the the *overflow* and *underflow* formulae.

In detail the φ formula, related to the *overflow* attack, is able to verify whether an *overflow* is in progress (i.e., when the *lt1* level exhibits an *up* value at least for 11 times). The *underflow* attack, described by the χ formula, is able to verify whether an *underflow* is in progress (i.e., when the *lt2* level exhibits a *low* value at least for 9 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 contextually can occur *overflow* and *underflow* attacks (i.e., $E\langle\rangle\,\varphi\ \wedge\ \chi$).

C. Experimental Results and Discussion

The formula verification results are shown in Table V.

With L^x we indicate the log of the water tanks for a single day, where $L \in \{attack, normal\}$, x identifies the day under analysis. The dataset comprises 20 day log, respectively ten exhibiting a normal behaviour, while the remaining ten afflicted by *overflow* and *underflow* attacks. The column (φ) identifies the models resulting *true* (✓ in Table V) or *false* (✗ in Table V) to the *overflow* formula, while the column (χ) identifies the models resulting *true* or *false* to the *underflow* formula.

Performances / **Model**	φ	χ
$attack^1$	✓	✓
$attack^2$	✓	✓
$attack^3$	✓	✓
$attack^4$	✓	✓
$attack^5$	✓	✓
$attack^6$	✓	✓
$attack^7$	✓	✓
$attack^8$	✓	✓
$attack^9$	✓	✓
$attack^{10}$	✓	✓
$legitimate^1$	✗	✗
$legitimate^2$	✗	✗
$legitimate^3$	✗	✗
$legitimate^4$	✗	✗
$legitimate^5$	✗	✗
$legitimate^6$	✗	✗
$legitimate^7$	✗	✗
$legitimate^8$	✗	✗
$legitimate^9$	✗	✗
$legitimate^{10}$	✗	✗

TABLE V: Formula Verification.

From the model checker output shown in Table V we can state the φ and the χ are able to detect all the models afflicted by the *overflow* and *underflow* attacks. Furthermore, all the models without attacks are marked as *false* by φ and χ formulae.

Moreover, 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, whether we are evaluating the φ formula, this value represents the number of one day logs whose related *overflow attack* model is correctly labelled as *true* by the formal verification environment) and *fp* indicates the number of false positives (for instance, whether we are 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 *overflow attack* model is wrongly labelled as *false* by the formal verification environment).

The F-Measure is a measure of a 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 all the evaluated models:
$Accuracy = \frac{tp+tn}{tp+fn+fp+tn}$
where *tn* indicates the number of true negatives (for instance, whether we are evaluating the φ formula, this value represents the number of one day logs whose related *legitimate* model is correctly labelled as *false* by the formal verification environment).

Precision	Recall	F-Measure	Accuracy	Attack
1	1	1	1	*overflow*
1	1	1	1	*underflow*

TABLE VI: Performance results.

The evaluation results shown in Table VI are really 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

Considering the environments monitored by SCADA systems (from oil pipelines to water distributions systems), an attack targeting this critical infrastructure can be easily reflected on human health. With the aim to mitigate these attacks, in this paper we propose a model checking based method to detect *overflow* and *underflow* attacks on SCADA system. The obtained performances are encouraging: 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 the 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 [11], [12] and security [13], [14].

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] S. A. Boyer, *SCADA: supervisory control and data acquisition*. International Society of Automation, 2009.
[2] 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.
[3] R. Taormina, S. Galelli, N. O. Tippenhauer, E. Salomons, A. Ostfeld, D. G. Eliades, M. Aghashahi, R. Sundararajan, M. Pourahmadi, M. K. Banks, *et al.*, "Battle of the attack detection algorithms: Disclosing cyber attacks on water distribution networks," *Journal of Water Resources Planning and Management*, vol. 144, no. 8, p. 04018048, 2018.
[4] 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.
[5] E. Gonzalez, B. Stephen, D. Infield, and J. J. Melero, "Using high-frequency scada data for wind turbine performance monitoring: A sensitivity study," *Renewable energy*, vol. 131, pp. 841–853, 2019.
[6] 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.
[7] 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.
[8] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical computer science*, vol. 126, no. 2, pp. 183–235, 1994.
[9] 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.
[10] J. Dougherty, R. Kohavi, and M. Sahami, "Supervised and unsupervised discretization of continuous features," in *Machine Learning Proceedings 1995*, pp. 194–202, Elsevier, 1995.
[11] 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.
[12] 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.
[13] 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.
[14] 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.