Generation of Applicative Attacks Scenarios Against Industrial Systems

Maxime Puys, Marie-Laure Potet, Abdelaziz Khaled

Generation of Applicative Attacks Scenarios Against Industrial Systems

Maxime Puys, Marie-Laure Potet and Abdelaziz Khaled

Université Grenoble Alpes, CNRS, VERIMAG, UMR 5104 700 av. centrale, IMAG/CS-40700
38058 Grenoble Cedex 9 France Firstname.Name@univ-grenoble-alpes.fr *

Abstract. In the context of security, risk analyzes are widely recognized as essential. However, such analyzes need to be replayed frequently to take into account new vulnerabilities, new protections, etc.. As exploits can now easily be found on internet, allowing a wide range of possible intruders with various capacities, motivations and resources. In particular in the case of industrial control systems (also called SCADA) that interact with the physical world, any breach can lead to disasters for humans and the environment. Alongside of classical security properties such as secrecy or authentication, SCADA must ensure safety properties relative to the industrial process they control. In this paper, we propose an approach to assess the security of industrial systems. This approach aims to find applicative attacks taking into account various parameters such as the behavior of the process, the safety properties that must be ensured. We also model the possible positions and capacities of attackers allowing a precise control of these attackers. We instrument our approach using the well known model-checker UPPAAL, we apply it on a case study and show how variations of properties, network topologies, and attacker models can drastically change the obtained results.

1 Introduction

In the context of security, risk analyzes are widely recognized as essential. However, due to the extremely fast evolution of the state of the art of attacks, they need to be replayed frequently to take into account new vulnerabilities, new protections, etc. It is also often required for auditors to be able to replay risk analyses made by vendors in a certification process. Moreover, the increasing number of updates to apply encourages to replay both security and safety tests to ensure that new updates do not break the system. Thus, we need tools able to quantify the robustness of applications or to find attack scenarios. Furthermore, as a whole ecosystem is emerging around vulnerabilities and attacks, exploits can easily be found on internet, allowing a wide range of possible intruders from script-kiddies to governments including hacktivists, mafias, or terrorists organizations. Those attackers can present various capacities, motivations and resources and can even collude together. Such differences must be taken into account when assessing the security of a system.

In this paper, we focus on industrial systems. Generally called SCADA, they control industrial processes such as electricity production, water treatment or transportation.

* This work has been partially funded by the SACADE (ANR-16-ASTR-0023) project.

Since those processes are usually critical, any incident can potentially harm humans and the environment. One of the most advertised attack was Stuxnet in 2010 [1] where a worm managed to sabotage a nuclear facility in Iran. This attack made people realize that a computer attack can have disastrous effects in the physical world. More recent attacks against these systems have been revealed in the past few years. For instance in 2014 against a German steel mill [2] where attackers managed to take control of a blast furnace or in 2015 in Ukraine [3] causing a massive power outage in winter.

Industrial systems are specific in various ways. First they want to ensure mainly availability and integrity while traditional IT systems often focus on confidentiality and authentication. Also the lifetime of their devices can vary between 20 to 40 years and they are really difficult to be updated in case of vulnerabilities. Industrial systems communicate over particular protocols which where not designed with security in mind. For example, MODBUS and DNP3 do not provide any security at all while a more recent communication protocol named OPC-UA includes the use of cryptography and has been show secure [4, 5] (but currently rarely used in practice).

Related Work. Verifying the security of industrial systems have keep gaining in interest and various approaches were proposed since Byres et al. in 2004 [6]. In 2015, Cherdantseva et al. [7] performed a survey of 24 methods published between 2004 and 2014. They base their list on criteria such as the domain of application, the use of probabilities or not, the presence of case studies or if the method is implemented. Similar surveys have been released in 2012 by Pièvre-Cambacédès and Bouissou [8], and in 2015 by Kriaa et al. [9]. We briefly summarize some of the works listed in these surveys either for their notoriety or for their closeness to our approach. In 2004, Byres et al. [6] propose a qualitative approach relying on attack trees to evaluate the security of industrial systems. Their approach is focused on systems communicating over MODBUS and targeting the electrical domain. In 2012, Kriaa et al. [10] present a method based on fault trees combined with Markov processes to model attacks on industrial systems. They implement this approach with the KB3 [11] tool and apply it to the Stuxnet attack. In 2015, they publish S-CUBE [12], an implementation of the former approach in the Figaro language. This approach takes into account the applicative logic of the process. In 2017, Rocchetto and Tippenhauer [13] present a method based on the cryptographic protocol verification tool CL-Atse [14]. They use the ASLAN++ language to model the industrial system and its applicative logic alongside with an augmented Dolev-Yao intruder, able to physically interact with the process [15].

Contributions. In this context, we propose an approach to assess the security of industrial systems. This approach aims to find what we call *applicative attacks*. That is, considering an attacker that already exploited some security breaches to gain access to the system, we focus on finding what actions can he actually perform and what are the consequences on the industrial process. To find such attacks, we take into account various parameters such as the behavior of the process, the safety properties that must be ensured. We also model the possible positions and capacities of attackers allowing a precise and flexible control of these attackers. We implement our approach within the UPPAAL model-checker [16] to automate the discovery of attacks scenarios.

Outline. We first describe our global approach in Section 2. Then in Section 3 we detail how we instrument it using the UPPAAL model-checker. In Section 4, we apply the resulting framework on a concrete industrial example.

2 Context

In this section, we first detail how the analysis presented in this paper is included in a larger approach. Then we propose a case study and detail the parameters we will take into account.

2.1 The A^{2}SPICS Approach

Our goal is to create a framework to detect applicative attacks against industrial systems. In this framework, industrial systems are modeled along with safety properties that they must ensure (e.g.: A furnace should not be started if its door is open). Then using formal methods such as *model-checking*, the model is analyzed in presence of intruders. In a later stage, found attacks could then be concretized into real networks packets that can be sent to a testbed representing the modeled system. Benefits are two-fold : besides being able to find applicative attacks, we can check if they are feasible and quantify their plausibility on the testbed.

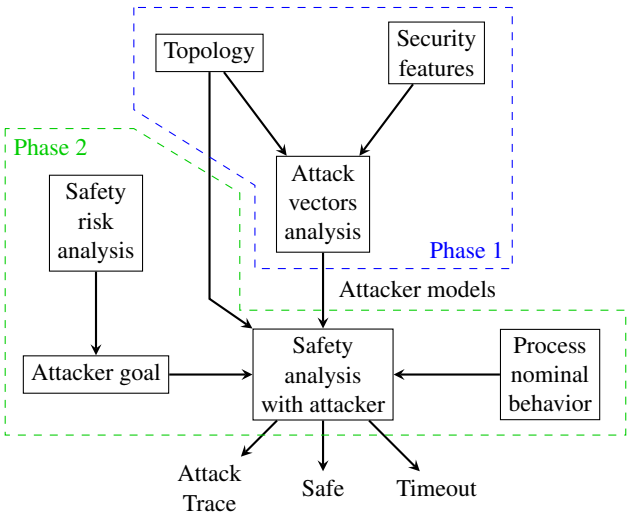


Fig. 1: The A^{2}SPICS approach

In Figure 1, we present the A^{2}SPICS approach for *Applicative Attack Scenarios Production for Industrial Control Systems*. We focus on systems that respect safety properties in absence of attackers. In this context, we consider two phases of analysis. In

the first phase (depicted in blue), we perform what we call an attack vector analysis [17]. It is a risk analysis in terms of security aiming to model attackers. It differs from well-known risk analysis methods such as EBIOS or MEHARI [18, 19] since they are focused on the assets to protect and the threats they face. Our risk analysis method relies on the topology of the system and the security features of communication protocols and produces what we call *attacker models*. Such models consists in placing possible attackers in the topology alongside as their capacities. For instance, if a protocol between two devices is considered secure, then no attacker is placed on this network channel. Similarly, if the protocol provides authentication but neither confidentiality nor freshness of messages, then we can place an attacker that can listen and replay messages. This first analysis thus allows us to place attackers in the network and choose their capacities according to their objectives and the security features of the communication protocols. In a second phase (depicted in green), we take advantage of the fact that industrial systems are usually well analyzed in terms of safety. Thus, we consider as attacker goals the negation of a subset of the properties that the system has to ensure, resulting of these safety risk analyses. Then, based on the nominal behavior of the system, we are able to conclude if the safety properties can be jeopardized by the attackers. This second phase is the one presented in this paper.

2.2 Case Study

To illustrate our approach and show its validity, we will apply it on a case study along this paper. We choose as example a bottle filling factory taken from the VirtualPlant simulator[1]. This simulator, designed by Jan Seidl, aims at providing a process simulator for experimentations. Empty bottles are carried by a conveyor belt. A sensor tells when a bottle is positioned under a nozzle which then pours liquid into the bottle. A second sensor detects when the bottle is full and then tells the nozzle to close and the conveyor belt to move until the next bottle is in place. Finally, a client can start and stop the whole process. Regardless of the communication protocol used, messages sent by the clients to the servers are read or write requests followed by read or write responses from the server of the form:

$$C \rightarrow S : READ, variableToRead$$
$$S \rightarrow C : READ, variableToRead, valueRead$$

And respectively for write requests and responses:

$$C \rightarrow S : WRITE, variableToWrite, newValue$$
$$S \rightarrow C : WRITE, variableToWrite, writeSuccessOrNot$$

Figure 2 shows a synoptic view of the bottle factory process from the VirtualPlant process simulator. Although this example is quite simple, it allows a wide variety of instantiations. First, several properties to guarantee can be expressed: (i) bottles must leave the factory full, (ii) liquid should not be spilled out of bottles, (iii) the conveyor belt should start when a bottle is full, etc. Different topologies of the network controlling the process can also be studied. We can consider the conveyor belt and

[1] https://github.com/jseidl/virtuaplant

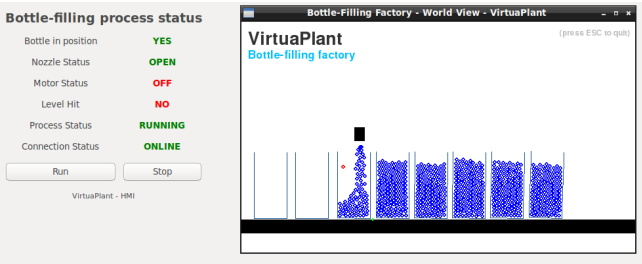


Fig. 2: VirtualPlant simulator

the nozzle as two distinct components. They could both be controlled by a single server (as shown in Figure 3) or they can each be controlled by a individual server. Moreover, the communication protocols used in the network can present different levels of security allowing more or less powerful attackers. Even the positions of attackers can be considered. It can for instance be positioned on a network channel as a Man-In-The-Middle or as a corrupted client or server (e.g.: a legitimate device infected by a virus).

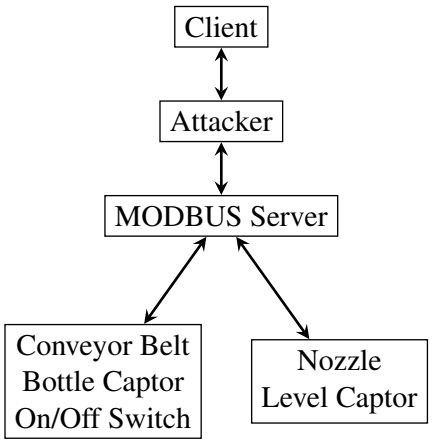


Fig. 3: Example of topology

2.3 Parameters of the model

Our model is composed of various parameters including different entities communicating together:

Process The process is the industrial application controlled by the system. It can for instance describe electricity production, liquid treatment or transportation. It is composed

of a set of variables $\mathcal{V}_\mathcal{P}$ linked together by an automaton $\mathcal{B}_\mathcal{P}$. We denote this automaton as the behavior of the process.

Clients The clients $\mathcal{C}$ are used to send commands to monitor and modify the process. They manage a set of variable $\mathcal{V}_c \subseteq \mathcal{V}_\mathcal{P}, \forall c \in \mathcal{C}$ and a behavior $\mathcal{B}_c, \forall c \in \mathcal{C}$ determining which command they send and how they react to responses sent by the servers.

Servers The servers are receiving commands sent by clients and applying them to the process. The security of the communication channel they use is determined by the protocol they implement (e.g.: MODBUS or OPC-UA). They also manage a set of variables $\mathcal{V}_s \subseteq \mathcal{V}_\mathcal{P}, \forall s \in \mathcal{S}$.

Properties The safety properties Φ to check on the system in presence of possibly active intruders are logical predicates (e.g.: CTL [20] temporal logic properties) on variables from $\mathcal{V}_P$.

Attackers The attackers $\mathcal{A}$ are possibly active intruders aiming to violate the safety properties from Φ. Their position in the network determines the clients and servers they will be able to communicate with while their capacities determine what type of action they will be able to perform (e.g.: intercept a message, encrypt a message, etc.). Depending on their capacities, attackers can also possess their own knowledge.

Topologies We denote as components all clients, servers and attackers. We also denote the network channels linking these components as network topology of the system.

3 Implementation in UPPAAL

In this section, we describe how we deploy our approach in the UPPAAL model-checker [16]. We first show how to model the system. Then we detail the attackers we consider and finally the specifications of the safety properties.

3.1 Framework Architecture

Figure 4 depicts the overall architecture of our framework. It contains three components: (i) the system's model, (ii) the attacker models, and (iii) the specification of the the safety properties. Several models are already predefined as templates in a library we provide to the user (including clients, servers, attackers, security primitives, etc.). Thus, the user is only required to provide the topology of the system using templates from the library and behaviors of clients and servers.

3.2 The system's model

In UPPAAL, we model the components interacting with attackers as a composition of timed automata. Clients can create, send requests and receive responses while the server can receive requests, send responses and execute actions according to the clients'

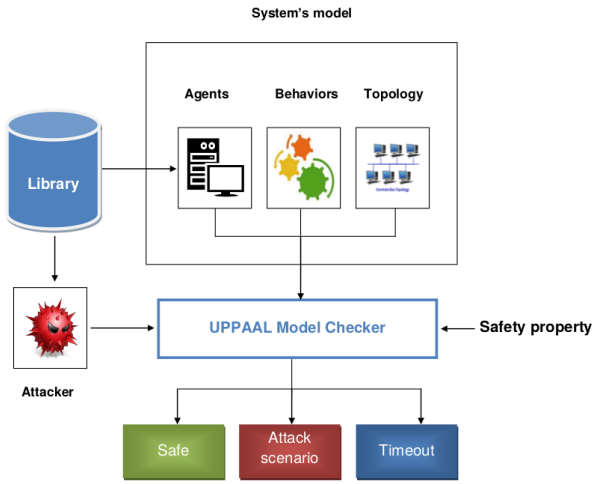


Fig. 4: Framework Architecture

requests. Attacker act as *Man-In-The-Middle* intruders and have different capacities depending on the configuration. Among those capacities, they can listen to the network, stop, forge, replay or modify some messages according to its knowledge.

In our framework, we model six automata named: *Client*, *BehaviorClient*, *Server*, *BehaviorServer*, *SecureData* and *Attacker*. They access global variables such as cryptographic keys, messages exchanged over channels[2], as well as the system variables $V_{\mathcal{P}}$. According to Section 2.2, commands are formatted using the data structure $\langle cmdType, variable, value \rangle$ where:

- $cmdType$ is a constant that expresses the purpose of the command (e.g.: read or write);
- $variable$ is a constant denoting the different variables of the system;
- $value$ is a the value of $variable$ when needed by the command (for instance the new value of the variable in a write request or the value read in a read response).

To send a message, the *Client* automaton first asks the *BehaviorClient* automaton to obtain the applicative content he will send. Then, in the case of a client with using a secure communication protocol, the message will by signed and/or encrypted using the *SecureData* automaton. Concerning the *Server* automaton, it waits for a message sent by the *Client* automaton. When received, if the server implements a secure protocol, it decrypts the message and/or checks the message signature. Then, depending on the type of message (read/write), it either writes the new value of the variable addressed or reads its current value. Either way, the server creates and sends a response to the client according to the security of the request.

[2] In UPPAAL, messages are not exchanged directly on channels. Instead signals are sent telling processes to access messages as global variables.

The *SecureData* automaton is used to manage security operations (encryption, decryption, signature, verification, etc.) according to cryptographic keys known by each component (including attackers).

3.3 Attackers

We consider four attackers with different capacities, each modeled as an automaton. Attacker A_1 (shown in Figure 5a), based on the Dolev-Yao model [21], can listen to the network, stop, forge, replay or modify messages according to its knowledge. Such attacker is often considered as extremely powerful [22] making him really suited to prove absence of attacks but less realistic when considered within a vulnerability analysis. In Figure 5a, the execution of the state diagram of the attacker begins in state A_1 where the attacker can choose the action it can execute where:

- $Intercept$ allows the attacker to intercept a message msg sent by a client or a server on some channel $chan$;
- $Send$ allows the attacker to send a message msg to a client or a server on some channel $chan$;
- $Copy$ allows the attacker to memorize a message msg into its knowledge $\mathcal{K}_{A_1}$;
- $Keys$ allows the attacker to retrieve cryptographic keys from its knowledge $\mathcal{K}_{A_1}$;
- $Secure$ allows the attacker to perform cryptographic operations according to its knowledge on the keys;
- $Forge$ allows the attacker to create a new message msg from its knowledge $\mathcal{K}_{A_1}$;
- $Modify$ allows the intruder to modify an intercepted message msg according to its knowledge $\mathcal{K}_{A_1}$.
- $Replay$ allows the attacker to replay a message msg from its knowledge $\mathcal{K}_{A_1}$;

Capacities $Modify$ and $Replay$ could be seen as special cases of $Forge$ in the sense that modifying a message is the action of forging a message at the time where a legitimate message is intercepted rather than sending a message at any time. Similarly, replaying a message can occur at any time but restricts the set of possible messages to the one previously memorized. Attacker A_2 (shown in Figure 5b) is a subset of A_1 which can only to modify messages or parts of messages. To be more realistic, it can for example be limited to only modify the variable and value fields in order to not transform a read message into a write and vice versa. Such attacker would represent an attacker that want to avoid coarse attacks to be discrete. Attacker A_3 (shown in Figure 5c) is a subset of A_1 which can only to forge new messages according to its knowledge. Thus it can be used to model a blind attacker that is not able to wiretap communications. Finally attacker A_4 (shown in Figure 5d) is a subset of A_1 which can only to replay messages after memorizing them in its knowledge.

3.4 Safety properties

To specify the properties, UPPAAL uses a simplified version of CTL that is expressed by the following syntax.

$$\Phi ::= A\Box\Phi | E \diamond \Phi | E\Box\Phi | A \diamond \Phi | \Phi \rightarrow \Phi | \neg\Phi$$

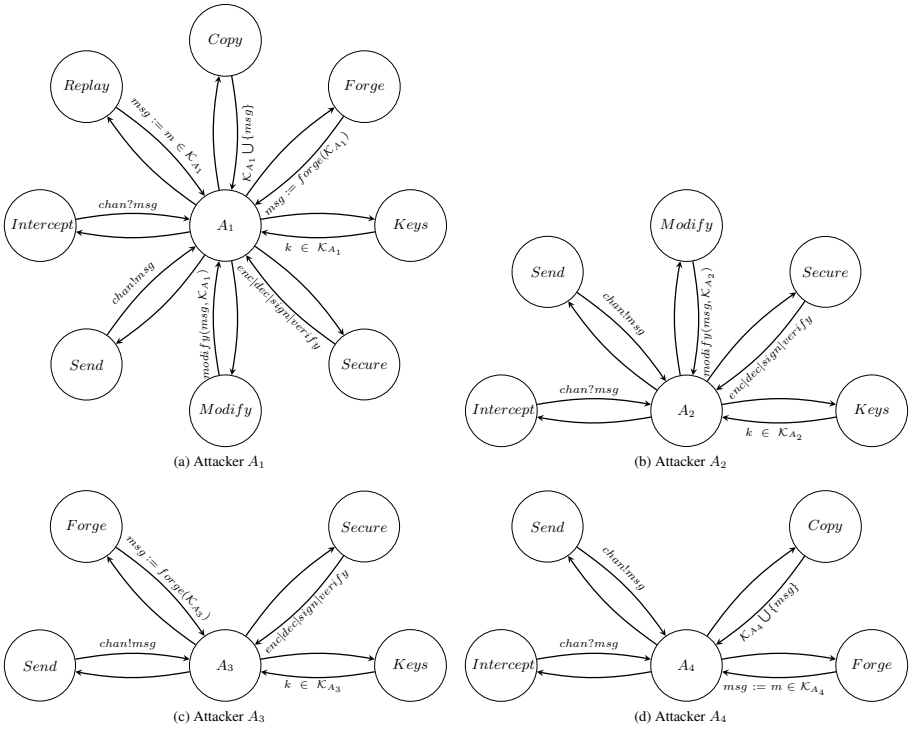


Fig. 5: Attackers considered

$A\Box\Phi$ means that Φ should be true on all paths in all reachable states. $A \diamond \Phi$ means that Φ should be eventually true on all paths. $E\Box\Phi$ means that there exists a path where Φ is true in all reachable states. $E \diamond \Phi$ means that there exists a path where Φ is eventually true. Symbols $\rightarrow$ and $\neg$ denote the implication and the negation propositional logic operators, respectively. To model safety properties we will only rely on $A\Box\Phi$.

4 Case Study

In this section, we illustrate our approach with the example described in Section 2.2. We show how we implemented it in the UPPAAL model-checker and we discuss the results we obtained by composing various attackers and topologies.

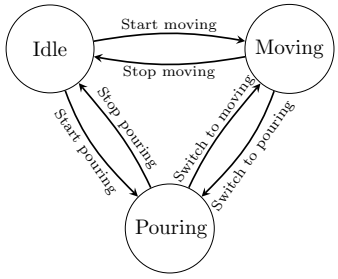
4.1 Behaviors

As described in Section 2.2, our case study is a bottle filling factory. Empty bottles are carried by a conveyor belt. A sensor tells when a bottle is positioned under a nozzle which then pours liquid into the bottle. A second sensor detects when the bottle is full and then tells the nozzle to close and the conveyor belt to move until the next bottle is

in place. A client can start and stop the whole process. In this example, the process is composed of five boolean variables:

$$\mathcal{V}_{\mathcal{P}} = \{motor, nozzle, levelHit, bottleInPlace, processRun\}$$

They respectively denote the conveyor belt ($motor$), the nozzle ($nozzle$), the liquid level sensor ($levelHit$), the conveyor belt sensor ($bottleInPlace$) and the process on/off switch ($processRun$). Figure 6a shows an automaton describing the behavior of the process while Table 6b details the transitions of the automaton. Three states are considered: $Idle$ means that the process is stopped, $Moving$ that the conveyor belt is moving to position the next bottle and $Pouring$ that the nozzle is filling a bottle. Each transition is labeled with two predicates: the guard and the output. The client will only start and stop the whole process when it wants[3]. Thus the variables that can be accessed by the client are $\mathcal{V}_c = \{processRun\}$.



(a) Process' behavior automaton

Current state	Next state	Guard	Actions
Idle	Moving	$processRun = true \wedge bottleInPlace = false$	$motor := true$
Idle	Pouring	$processRun = true \wedge bottleInPlace = true$	$nozzle := true$
Moving	Pouring	$bottleInPlace = true$	$motor := false \wedge nozzle := true$
Pouring	Moving	$levelHit = true$	$motor := true \wedge nozzle := false$
Moving	Idle	$processRun = false$	$motor := false \wedge nozzle := false$
Pouring	Idle	$processRun = false$	$motor := false \wedge nozzle := false$

(b) Details of the transitions

Fig. 6: Behaviors considered

The safety properties we want the process to guarantee would be a subset of properties considered as critical, resulting from a risk analysis in safety. For this case study, we exhibit the following properties, expressed as CTL formulas.

Φ_1: The nozzle opens only when a bottle is in position (i.e.: at all time and on all possible execution traces, $nozzle$ is never true if $bottleInPlace$ is false).
$A\Box \neg(nozzle = true \ \wedge \ bottleInPlace = false)$

Φ_2: The motor starts only when a bottle is full (i.e.: at all time and on all possible execution traces, $motor$ is never true if $levelHit$ is false).
$A\Box \neg(motor = true \ \wedge \ levelHit = false)$

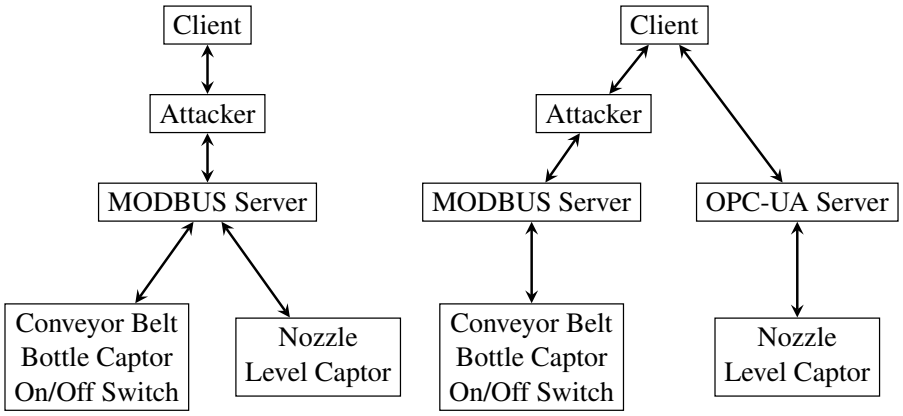
Φ_3: The nozzle opens only when the motor stops (i.e.: at all time and on all possible execution traces, $nozzle$ is never true if $motor$ is true).
$A\Box \neg(nozzle = true \ \wedge \ motor = true)$

[3] This models the actual behavior of the client in VirtualPlant and is not a limitation of our approach.

4.2 Network Topologies

We consider two network topologies T_1 and T_2. In topology T_1, a single server s_{MODBUS} using the MODBUS protocol controls both the conveyor belt and the nozzle. A single client c communicates with s_{MODBUS}. The MODBUS protocol is among the most used in industrial communications and does not provide any security at all. This topology is presented in Figure 7a with:

- Set of servers $\mathcal{S} = \{s_{MODBUS}\}$ with:
  - Variables $\mathcal{V}_{s_{MODBUS}} = \mathcal{V}_{\mathcal{P}}$
- Set of clients $\mathcal{C} = \{c\}$ with:
  - Variables $\mathcal{V}_c = \{processRun\}$



(a) Topology 1 (b) Topology 2

Fig. 7: Topologies considered

In topology T_2, the conveyor belt and the nozzle are each be controlled by a individual server. The first server s_{MODBUS} communicates using MODBUS and controls the conveyor belt, the position sensor, and the on/off switch. The second server s_{OPC-UA} communicates using OPC-UA and controls the nozzle and the level sensor. OPC-UA provides three security modes: *None*, *Sign* and *SignEncrypt*. Security mode *None* does not provide any security. According to Puys et al. [4], security mode *Sign* adds cryptographic signatures and provides authentication, integrity and freshness of communications and mode *SignEncrypt* also adds encryption providing confidentiality. We suppose that security mode *SignEncrypt* is used in our second topology, thus the attacker is not able to interfere with the channel between the client c and the OPC-UA server s_{OPC-UA}. This topology is presented in Figure 7b with:

- Set of servers $\mathcal{S} = \{s_{MODBUS}, s_{OPC-UA}\}$
  - Variables $\mathcal{V}_{s_{MODBUS}} = \{processRun, motor, bottleInPlace\}$
  - Variables $\mathcal{V}_{s_{OPC-UA}} = \{nozzle, levelHit\}$

– Set of clients $\mathcal{C} = \{c\}$
  - Variables $\mathcal{V}_c = \{processRun\}$

4.3 Attackers

To demonstrate the modularity of our framework, we test both topologies against the four attackers proposed in Section 3.3. We recall the capacities of each attacker in Table 1 where ✓means that the attacker has the capacity.

Attacker	Modify	Forge	Replay
A_1	✓	✓	✓
A_2	✓	✗	✗
A_3	✗	✓	✗
A_4	✗	✗	✓

Table 1: Summary of capacities for each attacker

4.4 Results obtained using UPPAAL

After experimenting different settings in UPPAAL, we chose to apply *Breadth first search* algorithm and to represent the states as *DBM (Difference Bounded Matrices)*. The results are summarized in Table 2 where ✓ means an attack has been found and ✗ means that the property is safe as well ◆ means that UPPAAL could not conclude. This happened because the tool was requesting more memory than available. Our experiments were run on a Intel(R) Core(TM) i5-4590 CPU @ 3.30GHz with 16GB of RAM. Times of analysis can be found and discussed in Section 5.1.

		A_1	A_2	A_3	A_4
T_1	Φ_1	✓	✓	✓	✗
	Φ_2	✓	✓	✓	✗
	Φ_3	✓	✓	✓	✗
T_2	Φ_1	◆	◆	✗	✗
	Φ_2	✓	✓	✓	✗
	Φ_3	✓	✓	✓	✗

Table 2: Results obtained

In theory, none of the four attackers can violate property Φ_1 in topology T_2. The reason is that the OPC-UA server controls the $nozzle$ variable, preventing any attack on this variable. Even with the MODBUS server controlling the $bottleInPlace$ variable, if $bottleInPlace$ is forced to $false$by an attacker while $nozzle$ is $true$, then $nozzle$ will automatically switch to $false$due to the process behavior (and vice versa). Thus, the only way to break Φ_1 that is to force opening the $nozzle$ which is not possible in topology 2 (as we can see with attackers A_3 and A_4). Similarly, attacker A_4 cannot violate any property, since the messages transmitted between the client and the server are only relative to start or stop the process.

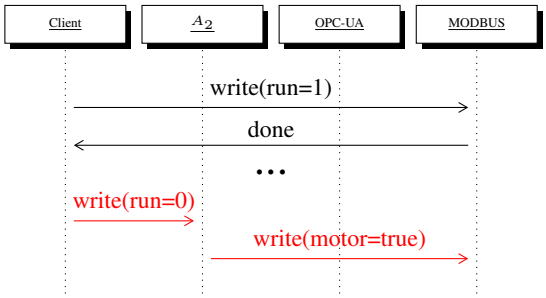


Fig. 8: Attack scenario with A_2 against Φ_2 in topology T_2

Figure 8 shows the attack scenario found by UPPAAL with attacker A_2 against Φ_2 in topology T_2. The client sends a message to the MODBUS server to start the process, the motor starts and the bottles advance on the conveyor belt. After some time, the client sends a message to stop the process. The attacker intercepts the message and modifies both the variable targeted by the write request and the new value to force the motor to start. This experimentation shows that we do not need the whole power of Dolev-Yao to find attacks. It also helps to find which are the capacities needed bye an attacker to perform attacks. Thus, it allows tailored proofs of robustness resulting of a risk analysis.

5 Discussions

In this section, we discuss the times taken for each analysis. We then compare our approach to related works presented in Section 1 and address some limitations and hypotheses we made.

5.1 Discussion of analysis timings

According to Tables 2 and 3, attacker A_2 obtains the same results as A_1 (Dolev-Yao) in shorter time. Attacker A_3 takes a bit longer but is able to conclude on property Φ_1 in topology 2 while attackers A_1 and A_2 cannot due to the system being out of memory.

		A_1	A_2	A_3	A_4
T_1	Φ_1	0.43 s	0.07 s	1.05 s	0.84 s
	Φ_2	0.52 s	0.10 s	0.69 s	0.35 s
	Φ_3	0.47 s	0.04 s	0.37 s	0.42 s
T_2	Φ_1	Out of memory		601 s	31.55 s
	Φ_2	0.66 s	0.23 s	2.17 s	35.20 s
	Φ_3	0.78 s	0.21 s	2.35 s	34.85 s

Table 3: Verification times

These results show that really powerful intruders such as Dolev-Yao are often too complex and only parts of them are sufficient to find attacks. Such intruders are however preferred when trying to prove the absence of attacks. On the other hand, attacker A_4 obtains larger times which can be surprising since it is the simplest of our attackers. A likely explanation is that since all of his results are absence of attacks, UPPAAL must explore every possible state which can take way longer that finding a counter example.

5.2 Comparing to State-of-the-Art

Our approach differs from most of the works presented in [7–9] that look more like risk analysis methods such as EBIOS [18, 19] for security or FMEA [23] for safety. It is typically the case of Byres et al. [6] who quantifies criteria such as likelihood or severity on a scale of four values. Moreover, 18 out of the 23 approaches listed in Cherdantseva et al. [7] are quantitative (i.e.: probabilistic) and thus require an initial distribution of probabilities to work. Nevertheless, a lot of these approaches give very few details on the source of these probabilities and their trustworthiness. It is also hard to evaluate the impact of variations of these probabilities. These approaches have however the advantage to quantify the likelihood and severity of resulting attacks. In [12], Kriaa et al. define four criteria to classify approaches combining security and safety:

1. analyzing formal models;
2. being both qualitative and quantitative;
3. being automated;
4. being adaptable to different assumptions.

Kriaa et al. also list some related works and conclude that none validate the automation criterion. In our case, the A^{2}SPICS approach respects criteria 1, 3 and 4 (relying on a formal and automated verification tool, UPPAAL and allowing to simply change attacker's positions and capacities as well as behaviors). To the best of our knowledge, the closest related work to the A^{2}SPICS approach from Rocchetto and Tippenhauer [13] which also seem validates criteria 1, 3 and 4. Our approach shows nevertheless key differences with it, particularly in terms of considered attackers. Using cryptographic protocol verification tools such as CL-Atse allows to not require to model the attacker which is hardcoded in the tool making the Dolev-Yao attacker difficult to restrict. In their work, Rocchetto and Tippenhauer strengthen it by adding equational theories (allowing to handle physical interactions with the process [15]). We aim to focus on attackers resulting of a risk analysis which are often less powerful than Dolev-Yao. Moreover, to the best of our knowledge, Rocchetto and Tippenhauer do not take into account the network topology of the system, although it seems possible in ASLAN++. It means in their case that all agents (or multiple groups of agents) communicate over one unique channel accessible to the attacker, which is again not very realistic.

5.3 Discussion of Limitations and Hypotheses

Similarly to [13], we consider that time is discretized (i.e.: expressed as steps of execution). The state of the process is also discretized (e.g.: the bottle is either empty or

full). Moreover, due to the complexity of attackers A_1, A_2, and A_3, we have to bound the number of actions they can perform in an attack. This limit of the number of action being configurable. This is a classical limitation of model-checking approaches that will not terminate if the model can loop infinitely. Moreover, an under-approximation of the approach can lead to some attacks not being found and robustness not being established. In the results showed in Table 2, we pointed that property Φ_1 was never violated. This is due to the fact that two states of the system can be considered: (i) the real state (i.e.: if a bottle is physically present or not), and (ii) the logical state (i.e.: if the variable $bottleInPlace$ is set to $true$). It appears that when a captor is modified by the intruder, then a decorrelation is introduced between these two state (in logical state, a bottle could be present while it is not the case in reality). However, properties are checked by UPPAAL on the logical state meaning possibly missing attacks (in particular for property Φ_1). This is a classical limitation due to the fact that we model the system without taking into account the physical environment.

6 Conclusion

We provided a modular approach to assess the security of industrial control systems. This approach aims to find applicative attacks taking into account different parameters such as the behavior of the process, the properties that an attacker can aim to jeopardize, as well as the possible positions and capacities of attackers. We show how this approach can be implemented using the UPPAAL model-checker. We apply it on an example and show how variation of properties, network topologies, and attackers can change the obtained results. We also discuss key difference with approaches relying on protocol verification tools. Even when considering all possible variations of our example, it remains very simple. Still, the timing results we obtained encourage us to address the question of scalability. In the future, we would be interested into studying how to address the limitation pointed in Section 5.3. It would be useful to apply our approach to the case study proposed by Rocchetto and Tippenhauer to obtain a concrete comparison of the two approaches. We are also interested into modeling possible collusions between intruders so they can share knowledge and synchronize during attacks. Finally, we aim to generalize the implementation and build an open-source tool to automatically generate UPPAAL models and interpret the results.

References

1. Ralph Langner. Stuxnet: Dissecting a cyberwarfare weapon. *Security & Privacy, IEEE*, 9(3):49–51, 2011.
2. Robert M Lee, Michael J Assante, and Tim Conway. German steel mill cyber attack. *Industrial Control Systems*, 30, 2014.
3. Robert M Lee, Michael J Assante, and Tim Conway. Analysis of the cyber attack on the ukrainian power grid. *SANS Industrial Control Systems*, 2016.
4. Maxime Puys, Marie-Laure Potet, and Pascal Lafourcade. Formal analysis of security properties on the OPC-UA SCADA protocol. In *Computer Safety, Reliability, and Security - 35th International Conference, SAFECOMP 2016, Trondheim, Norway, September 21-23, 2016, Proceedings*, pages 67–75, 2016.

5. Jannik Dreier, Maxime Puys, Marie-Laure Potet, Pascal Lafourcade, and Jean-Louis Roch. Formally Verifying Flow Integrity Properties in Industrial Systems. In *SECRYPT 2017 - 14th International Conference on Security and Cryptography*, page 12, Madrid, Spain, July 2017.
6. Eric J Byres, Matthew Franz, and Darrin Miller. The use of attack trees in assessing vulnerabilities in scada systems. In *Proceedings of the international infrastructure survivability workshop*, 2004.
7. Yulia Cherdantseva, Pete Burnap, Andrew Blyth, Peter Eden, Kevin Jones, Hugh Soulsby, and Kristan Stoddart. A review of cyber security risk assessment methods for {SCADA} systems. *Computers & Security*, 56:1 – 27, 2015.
8. Ludovic Piètre-Cambacédès and Marc Bouissou. Cross-fertilization between safety and security engineering. *Reliability Engineering & System Safety*, 110:110–126, 2013.
9. Siwar Kriaa, Ludovic Pietre-Cambacedes, Marc Bouissou, and Yoran Halgand. A survey of approaches combining safety and security for industrial control systems. *Reliability Engineering & System Safety*, 139:156–178, 2015.
10. Siwar Kriaa, Marc Bouissou, and Ludovic Piètre-Cambacédès. Modeling the stuxnet attack with bdmp: Towards more formal risk assessments. In *Risk and Security of Internet and Systems (CRiSIS), 2012 7th International Conference on*, pages 1–8. IEEE, 2012.
11. Ludovic Piètre-Cambacédès, Yann Deflesselle, and Marc Bouissou. Security modeling with bdmp: from theory to implementation. In *Network and Information Systems Security (SAR-SSI), 2011 Conference on*, pages 1–8. IEEE, 2011.
12. S Kriaa, M Bouissou, and Y Laarouchi. A model based approach for SCADA safety and security joint modelling: S-Cube. In *IET System Safety and Cyber Security*. IET Digital Library, 2015.
13. Marco Rocchetto and Nils Ole Tippenhauer. Towards formal security analysis of industrial control systems. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 114–126. ACM, 2017.
14. Mathieu Turuani. The CL-Atse Protocol Analyser. In Frank Pfenning, editor, *17th International Conference on Term Rewriting and Applications - RTA 2006 Lecture Notes in Computer Science*, volume 4098 of *LNCS*, pages 277–286. Springer, August 2006.
15. Marco Rocchetto and Nils Ole Tippenhauer. Cpdy: Extending the dolev-yao attacker with physical-layer interactions. In *International Conference on Formal Engineering Methods*, pages 175–192. Springer, 2016.
16. Gerd Behrmann, Re David, and Kim G. Larsen. A tutorial on UPPAAL. pages 200–236. Springer, 2004.
17. Maxime Puys, Marie-Laure Potet, and Jean-Louis Roch. Génération systématique de scénarios d'attaques contre des systèmes industriels. In *Approches Formelles dans l'Assistance au Développement de Logiciels, AFADL 2016, Besançon, France*, 2016.
18. ANSSI. Expression des besoins et identification des objectifs de sécurité. Agence nationale de la sécurité des systèmes d'information, 2010.
19. CLUSIF. Méthode harmonisée d'analyse des risques, 2010.
20. Edmund Clarke and E Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. *Logics of programs*, pages 52–71, 1982.
21. D. Dolev and Andrew C. Yao. On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29(2):198–208, March 1981.
22. Iliano Cervesato. The dolev-yao intruder is the most powerful attacker. In *16th Annual Symposium on Logic in Computer Science—LICS*, volume 1, 2001.
23. IEC-60812. Analysis techniques for system reliability - Procedure for failure mode and effects analysis (FMEA). International Electrotechnical Commission, 1985.