Model Checking of Security Patterns Implementation: Application to SCADA

Fadi Obeid
PhD student, Lab-STICC
ENSTA Bretagne, Brest, France
Email: fadi.obeid@ensta-bretagne.org

Philippe Dhaussy
HDR, Lab-STICC
ENSTA Bretagne, Brest, France
Email: philippe.dhaussy@ensta-bretagne.fr

I. Introduction

Supervisory control and data acquisition (SCADA) systems can be found in modern industrial facilities such as water pipes, power plants, oil refineries, chemical factories and nuclear facilities. These systems use coded signals over communicating channels to monitor and control numerous devices on multiple and distant sites. Securing SCADA communications has always been a challenge for security experts, especially that the used devices usually lack the support of cryptography. The increased connectivity of SCADA systems, increases the security requirements as well as the security issues and possible threats. Recently, there has been a high increase in SCADA attacks [1].

Many studies [2], [3], [4] were conducted to address the security threats in SCADA and/or propose some theoretical solutions and guides. Other studies [5], [6] aim to enhance the security level and fortify the provided services along with the managed critical data. One of the proposed solutions [7] is to apply security patterns on SCADA architectures.

A security pattern unifies efforts to create a trustworthy solution for a specific reoccurring security problem, which would make sure best practices are being used. Security patterns are usually defined and formalized by security specialists and used later by developers who may lack the needed security knowledge. A security pattern provides detailed guideline to solve a security issue using the best found solution. When applying security countermeasures on a system, we need to make sure that they do not interfere with the correct functionality of the system and its services.

Model checking (MC) proved to be effective and trustworthy in certifying whether a system meets its requirements and ensuring its correct functionality [8]. In this paper, we use MC tools to verify the our architectures regarding its functionality, the security requirements, and potential attacks.

We test our concept using a simple architecture with one security pattern and only one type of attacks. However, this is only a foothold for the next step which is combining multiple security patterns and automatic application of these patterns into an architecture.

Figure 1 presents a general SCADA architecture and its communication to a second SCADA, a local network, and the Internet. In our work, we are interested in the SCADA network, which includes the communications between the supervisory system, the PLCs, and the RTUs.

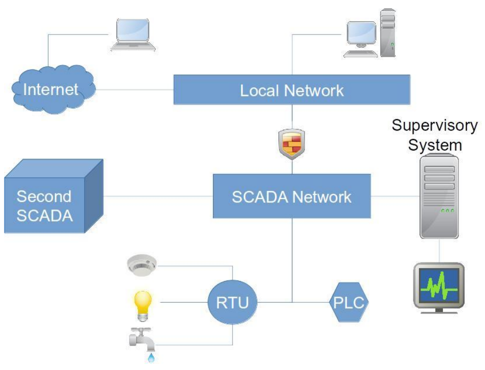

Figure 1. SCADA network presentation

Some related studies are presented in section 2. In section 3 of this paper we present the applicative part of this work by defining a use case, a security pattern, and an attack. We demonstrate and explain our results in section 4. Finally, we conclude this work in section 5 and reveal the focus of our research from this point forward.

II. Related Work

SCADA security, security patterns, and formal verification can be found individually in many researches. A discussion on the different security issues in SCADA networks can be found in [3]. A classification of SCADA cyber attacks can be found in [4], classified based on the type of the target: software, hardware, or communication stack. SCADA-specific security solutions and SCADA-specific IDS can be found in [9], [10] respectively. A survey on security patterns can be found in [11]. [12], [13] introduce many security patterns. Finally, a survey on Formal verification of security protocol implementations can be found in [14].

Other studies are closer to the work presented in this paper. [7] proposes using security patterns to design secure SCADA systems. [15] defines formal constrants of some security patterns to formally verify these patterns. MC security patterns compositions can be found in [16] where they use a case study to show how wrongly combining security patterns may result in new problems.

III. Application

Our use case consists of a nuclear power plant that satisfies a small city's consumption. The electricity consumption may increase or decrease (by δ) at any point of time. To respond to the changing consumption, the power plant may increase its working level to increase the production (by 1δ).

Apart from the central server, the plant is divided into 3 sections, each supervised by one PLC: A central core: Uses uranium to heat up pressed water. A steam generator: Uses heated pressed water to transform river water into steam. A turbine generator: Uses steam to generate electricity.

After each change in the consumption, the city sends a message about its current consumption needs to the central server. The central server recomputes the needed working level, and if it needs to be modified, it sends modification orders to the PLCs. Each PLC would then forward the command to the associated actuator(s). Afterwards, the PLCs would send status requests to all associated switches and sensors. Once a PLC receives all status updates, it measures the state of the section and sends it to the server.

When an order is executed, it may change some inputs and outputs. To simplify our model, these changes are managed by a process, and only happen when all current orders are terminated.

Figure 2 demonstrates the different sections of the plant along with their inputs and outputs.

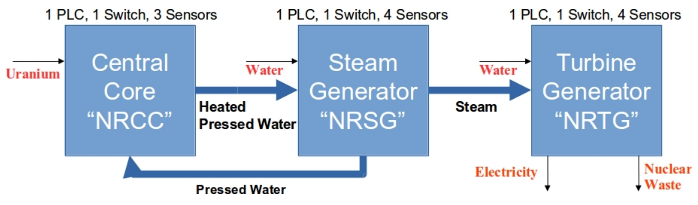

Figure 2. Nuclear Power Plant

A. Authorization Pattern

The Authorization pattern [13], [17] is used in various systems. It solves the problem of supervising resources accessed by subjects by defining a relation of access rights between subjects and protected objects (Figure 3).

Figure 3 represents the general class diagram of the authorization pattern. In the means of our application, an object can be the working level of the plant, or a switch, or even the value shown by a sensor.

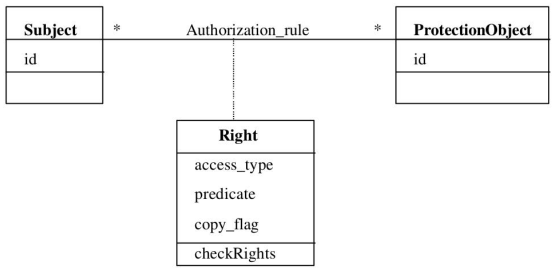

Figure 3. Authorization Pattern Class Diagram

B. Attack

Our simple attack consists of sending bad commands, for example the attacker would send a message to a PLC ordering it to change its working level even if there is no need to do so. Since in the initial architecture, no authorizations are needed to change a value, the orders sent by an attacker are going to be accepted by the receiver as authorized commands.

IV. Tests and Results

Our research consists of applying security patterns to SCADA systems. We need to formally prove, not only the correctness of such a combination, but also the actual need for applying security patterns in current SCADA systems.

A. Utilities

To be able to verify our model we use 3 essential elements: We use the Intermediate Format for the Embedded Distributed Component Architectures (FIACRE) [18] to define our architecture and the authorization pattern and the attack. We use the Context Description Language (CDL) [19] to formalize the security properties. One of the main advantages of using CDL is the reduction in state explosion [20]. We use the Observer-Based Prover (OBP) explorer (www.obpcdl.org) to explore our model and observe our properties. Figure 4 demonstrates the connections between the system requirements, the properties, the specifications and the explorer.

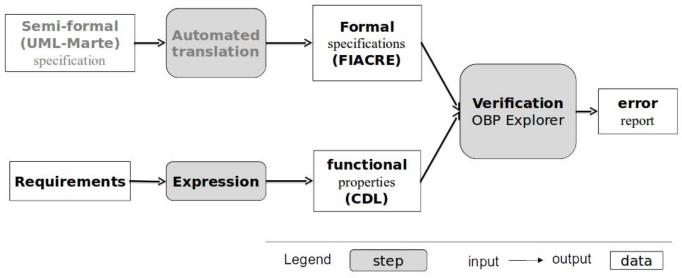
Semi-formal (UML-Marte) specification
Automated translation
Formal specifications (FIACRE)
Verification OBP Explorer
error report
Requirements
Expression
functional properties (CDL)
Legend
step
input
output
data

Figure 4. Observer-Based Prover

B. Properties Definition

The most basic requirement of our system is for the plan to provide enough power for the city without producing unnecessary power. This means that production should always be higher or equal to consumption (*pty1*) but never too much higher (*pty2*). Using CDL, we can specify a property (*pty*) based on an event (*eve*) which can be a simple predicate (*pred*) change. The state of the property changes depending on events so we can observe if at some point the property is rejected (something bad happened) and in some cases accepted (something good happened).

Using CDL, we write our first property (*pty1*) as follows:

- predicate $pred1$ is: $consumption > production$
- event $eve1$ is: $pred1\ becomes\ true$
- property $pty1$ is: $start - -//eve1/- > reject$

This means that $pred1$ is true if the $consumption$ is higher then the $production$. $eve1$ is observed at the point where

$pred1$ passes from $false$ to $true$. If at some point, $eve1$ is observed, then pty becomes rejected.

We write the second property (*pty2*) in the same fashion but we consider $pred$ as follows: $production - 3\delta >= consumption$

In our case, if an attacker successfully sends a message to a device, and the production still corresponds to the consumption, the attack would go unnoticed. This would be a successful attack, but it does not damage our system since it does not present a risk to the system requirements.

C. Different Modes

We verify our system under different circumstances. First, we check our system in a normal mode (NM), to prove that it represents the system correctly. In the attacked mode (NMUA), we added an attacker to prove that the system lacks of security. We added the authorization pattern to the normal mode to create a secure mode (SM). We finally tested the secure mode with the presence of an attacker (SMUA). The results are shown in table I.

Mode	States	transitions	Time (sec)	Violated Properties
NM	319	460	0.511	-
NMUA	3316	7809	1.365	2
SM	319	460	0.514	-
SMUA	1515	3952	0.934	-

Table I
MODELS EXPLORATIONS RESULTS

In NM None of the security properties were violated, meaning the system worked correctly as it is supposed to without problems. In $NMUA$, the difference in the number of states and transitions, can already be understood as a problem. The increased number of states and transitions means that the system did some unexpected transitions, and entered some unexpected states Both of our security properties were violated. In SM none of the security properties were violated. We can notice that the number of states and transitions is the same as the ones in the normal mode. This is due to the fact that the used security patterns did not change the normal functionality of the system. Finally, in $SMUA$, none of the security properties were violated which means that the countermeasure was effective. The increased number of states and transitions here is due to the messages sent by the attacker and the refusals of the system devices. The system is still working correctly and all additional states and transitions are deviations that do not affect the security of the system. Example: From state A, receive a message from the attacker, refuse the message, go back to state A.

V. CONCLUSION

We were able to use MC to verify the correct implementation of a security pattern on a SCADA architecture.

Based on this work, We started constructing a small library of security patterns, a library of potential attacks, and a meta model for SCADA architectures. Any architecture based on our meta model can use the security patterns from our library to generate a secure architecture. The secure architecture is then verified automatically using model checking tools. The resistance of the secure architecture is also verified by proving its correctness in the case of attacks from the attacks library.

In this work we consider that messages are encrypted/signed, however, when it comes to SCADA communication, cryptography is a big issue. Therefore, we are also working on a lite cryptography protocol that considers the requirements and limitations of SCADA.

REFERENCES

[1] Eric Byres and Justin Lowe. The myths and facts behind cyber security risks for industrial control systems. In *Proceedings of the VDE Kongress*, volume 116, pages 213–218, 2004.

[2] Ronald L Krutz. *Securing SCADA systems*. John Wiley & Sons, 2005.

[3] Vinay M Igure, Sean A Laughter, and Ronald D Williams. Security issues in scada networks. *Computers & Security*, 25(7):498–506, 2006.

[4] Bonnie Zhu, Anthony Joseph, and Shankar Sastry. A taxonomy of cyber attacks on scada systems. In *Internet of things (iThings/CPSCom), 2011 international conference on and 4th international conference on cyber, physical and social computing*, pages 380–388. IEEE, 2011.

[5] Yongge Wang. sscada: securing scada infrastructure communications. *International Journal of Communication Networks and Distributed Systems*, 6(1):59–78, 2010.

[6] Igor Nai Fovino, Alessio Coletta, Andrea Carcano, and Marcelo Masera. Critical state-based filtering system for securing scada network protocols. *Industrial Electronics, IEEE Transactions on*, 59(10):3943–3950, 2012.

[7] Eduardo B Fernandez and Maria M Larrondo-Petrie. Designing secure scada systems using security patterns. In *System Sciences (HICSS), 2010 43rd Hawaii International Conference on*, pages 1–8. IEEE, 2010.

[8] Edmund M Clarke, Orna Grumberg, and Doron Peled. *Model checking*. MIT press, 1999.

[9] Igor Nai Fovino, Alessio Coletta, and Marcelo Masera. Taxonomy of security solutions for the scada sector. *Project ESCORTS Deliverable*, 2, 2010.

[10] Bonnie Zhu and Shankar Sastry. Scada-specific intrusion detection/prevention systems: a survey and taxonomy. In *Proceedings of the 1st Workshop on Secure Control Systems (SCS)*, 2010.

[11] Nobukazu Yoshioka, Hironori Washizaki, and Katsuhisa Maruyama. A survey on security patterns. *Progress in informatics*, 5(5):35–47, 2008.

[12] Joseph Yoder and Jeffrey Barcalow. Architectural patterns for enabling application security. *Urbana*, 51:61801, 1998.

[13] Eduardo B Fernandez and Rouyi Pan. A pattern language for security models. In *proceedings of PLOP*, volume 1, 2001.

[14] Matteo Avalle, Alfredo Pironti, and Riccardo Sisto. Formal verification of security protocol implementations: a survey. *Formal Aspects of Computing*, 26(1):99–123, 2014.

[15] Ronald Wassermann and Betty HC Cheng. Security patterns. In *Michigan State University, PLoP Conf.* Citeseer, 2003.

[16] Jing Dong, Tu Peng, and Yajing Zhao. Model checking security pattern compositions. In *Quality Software, 2007. QSIC'07. Seventh International Conference on*, pages 80–89. IEEE, 2007.

[17] Markus Schumacher, Eduardo Fernandez-Buglioni, Duane Hybertson, Frank Buschmann, and Peter Sommerlad. *Security Patterns: Integrating security and systems engineering*. John Wiley & Sons, 2013.

[18] Bernard Berthomieu, Jean-Paul Bodeveix, Patrick Farail, Mamoun Filali, Hubert Garavel, Pierre Gaufillet, Frederic Lang, and François Vernadat. Fiacre: an intermediate language for model verification in the topcased environment. In *ERTS 2008*, 2008.

[19] Philippe Dhaussy, Frédéric Boniol, Jean-Charles Roger, and Luka Leroux. Improving model checking with context modelling. *Advances in Software Engineering*, 2012:9, 2012.

[20] Philippe Dhaussy, Jean-Charles Roger, and Frederic Boniol. Reducing state explosion with context modeling for model-checking. In *High-Assurance Systems Engineering (HASE), 2011 IEEE 13th International Symposium on*, pages 130–137. IEEE, 2011.