Model Checking of Security Patterns Implementation: Application to SCADA

Fadi OBEID, Philippe DHAUSSY

fadi.obeid@ensta-bretagne.org
philippe.dhaussy@ensta-bretagne.fr

Univ. Européenne de Bretagne
Lab-STICC / MOCS
UMR CNRS 6285
ENSTA-Bretagne, Brest
Objectives

- Apply security patterns on SCADA.
- Model checking of security patterns implementations.
- Complexity evaluation of security patterns modeling.
- A first pitch to finding a law of automatically applying security patterns.
Plan

- Objectives
- Security Patterns, SCADA
- Development Environment
- Experimentation and Results
- Patterns Patching
- Conclusion
Security Patterns

- A security pattern is a reusable architectural solution:
  - Proved to be efficient.
  - Resolves highly occurring security problem(s).

- Not all developers are security experts:
  - Security experts create and publish security patterns.
  - Patterns are explained in common standards (e.g., UML diagrams).
  - Developers can reuse the published security patterns.

- Security concerns exist in every development phase:
  - Requirement phase: Which assets need protection?
  - Design phase: Should satisfy security requirements.
  - Implementation phase: Implementing patterns chosen in the design.

[Yoder and Barcalow 1998, Fernandez and Pan 2001]
Security Patterns: Authorization pattern

In literature:

Subject \(\xrightarrow{\text{Has authorization on}}\) Object

Author: rights*

In our case:

Subject \(\xrightarrow{\text{Has authorization}}\) Object

On all objects of author: rights*
SCADA

Supervisory Control and Data Acquisition (SCADA):
- Remote controlling and data transfer.
- Most industrial facilities exp. power plants, oil refineries, etc.

Difference from normal networks?
- Special purpose embedded computing devices.
- Nonstop for years.
- Very sparse and geographically extensive, exp. Pipelines.
- Hard physical conditions, exp. chemical factories.

Issues:
- Open standards protocols, COTS hardware and software.
- Vulnerable protocols, lack of cryptography.

Difficulties:
- Hard to upgrade: very sparse, and should always be available.
- Cryptography would oppose to real-time computation.

[Zhu, Joseph, and Sastry 2011]
Development Environment

www.obpcdl.org

User Models (SysML, UML, AADL, etc.)

FIACRE Model

Exploration And Property Verification

Context Properties

Formal properties (predicates, invariants).
Context specifications: Interactions with environment.

Data Interpretation For Diagnostics

non-deterministic transition systems
Experimentation: System representation

The system is a SCADA example of a nuclear power plant.

The plant is divided into 3 sections:

- **Central core**: Uranium + Pressed water $\rightarrow$ Heated pressed water
- **Steam generator**: Heated pressed water + Water $\rightarrow$ Pressed steam
- **Turbine generator**: Pressed steam $\rightarrow$ electricity
Experimentation: Model

Each section is controlled and monitored by:

- One PLC: Controls a section.
- Switch(es): Controls levels exp. quantity of pumped water.
- Sensors: monitors values changes exp. quantity of steam.
Experimentation: System Architecture

City

Sensor

SensorsBandwidth

Sensor

PLC

PlcsBandwidth

PLC

Switch

Switch

Server

cityBandwidth

plcsBandwidth

switchesBandwidth

Model Checking of Security Patterns Implementation: Application to SCADA
Experimentation: Power Needs

Power needs/production vary between 180kwh/s and 240kwh/s

Production is considered high if the plant can decrease its level and still produce enough power.
Experimentation: Sequence Overview

- **Nuclear Reactor Server**
  - **t0**: Increase/Decrease level
  - **t6**: Send status

- **Each PLC**
  - **t1**: Send new level
  - **t3**: Get level info

- **PLC's connected Switches**
  - **t2**: Change level
  - **t4**: Get resource quantity info

- **PLC's connected Sensors**
  - **t4**: Send quantity info

- **t4** is mentioned multiple times because the order of the multiple t4s is nondeterministic.
Experimentation: Attacker

- The attacker: A simple unauthorized process that tries to manipulate others.
- The attacker can not send (nor edit) messages using others identifications:
  - Signature is supposed perfect.
- The attacker can not delete/redirect messages:
  - As if we are using messages broadcasting.
Experimentation: Attacker

OR

SendFakePowerNeeds

SendFakeLevelOrder

OR

SendFakeLevelOrder
Experimentation: Authorization Pattern

Diagram:
- City
- Sensor
- Sensor
- Switch
- PLC
- Server

Connections:
- sensorsBandwidth
- switchesBandwidth
- plcsBandwidth

Pattern:
- Authorization Pattern
Experimentation: Authorization Pattern
The plant (with no outside intervention) always has sufficient and not too much power production.

A successful attack would cause:

- Insufficient power production (bad).
- Too much power production (bad).
- No change to power production (no problem).

Properties examples (CDL):

- **NotEnoughPower** (*pty1*):
  - predicate *pred1* is: consumption > production
  - event *ev* is: *pred1* becomes true
  - property *pty1* is: start -- / / ev1 / -> reject  (Requirement 1)

- **TooMuchPower** (*pty2*):
  - predicate *pred2* is: production - consumption >= 30.
  - event *ev2* is: *pred2* becomes true
  - property *pty2* is: start -- / / ev2 / -> reject  (Requirement 2)
Results

Tests were carried on 4 different configurations:

- Normal model (NM): simply the planet working correctly.
- Normal model with attack (NMWA): Added the attacker to NM.
- Secured model (SM): Added the authorization pattern to NM.
- Secured model with attack (SMWA): Added the attacker to SM.

<table>
<thead>
<tr>
<th>Config.</th>
<th>nb. states</th>
<th>nb. transitions</th>
<th>Time needed (seconds)</th>
<th>Rejected properties (out of 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NM</td>
<td>75255</td>
<td>159084</td>
<td>8.676</td>
<td>0</td>
</tr>
<tr>
<td>NMWA</td>
<td>715286</td>
<td>2151963</td>
<td>75.877</td>
<td>2</td>
</tr>
<tr>
<td>SM</td>
<td>75255</td>
<td>159084</td>
<td>9.203</td>
<td>0</td>
</tr>
<tr>
<td>SMWA</td>
<td>301728</td>
<td>864677</td>
<td>34.31</td>
<td>0</td>
</tr>
</tbody>
</table>
Patterns Patching

Model Checking of Security Patterns Implementation: Application to SCADA
Patterns Patching

Automatic pattern patching:

- Using hashtags

  - ReadMessage(to, from, m).
  - AuthPatt(to, from, m).
  - DoSomething(m).

  Transformation

  - ReadMessage(to, from, m).
  - If not AuthPatt(to, from, m): Alert!. Else:
    - DoSomething(m).

- Using keywords

  - ReadMessage(to1, from1, m1).
  - DoSomething(m1).
  - Send(to2, from2, m2)#PassPatt

  Transformation

  - ReadMessage(to1, from1, m1).
  - If not AuthPatt(to1, from1, m1): Alert!. Else:
    - DoSomething(m1).
    - Send(to2, from2, m2)

Utility: Forgetting not to secure something is better than forgetting to secure it!
Conclusion

Security measurements needed in SCADA.

Interesting to study security patterns efficiency in securing SCADA.

Model checking can verify security patterns implementation.