Security-Schwachstellen mit Fuzzing aufdecken
2. – 4. Juli 2019, München

Frank Büchner, Principal Engineer Software Quality, Hitex GmbH
Hitex

- Founded 1976 in Karlsruhe, Germany
- 50 employees
- Subsidiary in UK (20 employees)
- Part of the Infineon Group since 2003
- Software tools, middleware
- Engineering, development, production
- AURIX preferred design house (PDH)
Detect Security Vulnerabilities through Fuzzing

1. A Look at IEC 62443
2. Fuzzing
3. A Look at IEC 61508 and ISO 26262
4. Fuzzing, Robustness Testing, and Fault Injection
5. Fuzzing Example
6. Studies
7. Conclusion
Security Standard IEC 62443

Overview

3-3, System security requirements and security levels

4-1 Product development requirements
Security Standard IEC 62443

Overview

IEC TS 62443-1-1: Terminology, concepts and models
IEC TR 62443-1-2: Master glossary of terms and abbreviations
IEC TS 62443-1-3: System security compliance metrics
IEC TR 62443-1-4: IACS security life-cycle and use-cases

IEC 62443-2-1: Establishing an industrial automation and control system security program
IEC TR 62443-2-2: Implementation guidance for an IACS security management system
IEC TR 62443-2-3: Patch management in the IACS environment
IEC 62443-2-4: Security program requirements for IACS service providers

IEC TR 62443-3-1: Security technologies for industrial automation and control systems
IEC TR 62443-3-2: Security risk assessment and system design
IEC 62443-3-3: System security requirements and security levels
IEC TR 62443-3-4: Product development requirements
IEC 62443-4-2: Technical security requirements for IACS components

3-3, System security requirements and security levels
Seven foundational requirements (FR)

a) Identification and authentication control (IAC)
b) Use control (UC)
c) **System integrity (SI)**
d) Data confidentiality (DC)
e) Restricted data flow (RDF)
f) Timely response to events (TRE), and
g) Resource availability (RA)
For each FR:

- Series of system requirements (SRs)
  - For each SR:
    - Requirement
    - Rationale and supplemental guidance
    - Zero or more requirement enhancements (REs)
      - Notes, if necessary
    - Security mapping
FR 3 – System Integrity

- SR 3.1 – Communication Integrity
- SR 3.2 – Malicious code protection
- SR 3.3 – Security functionality verification
- SR 3.4 – Software and information integrity
- **SR 3.5 – Input validation**
- SR 3.6 – Deterministic output
- SR 3.7 – Error handling
- SR 3.8 – Session integrity
- SR 3.9 – Protection of audit information
SR 3.5 – Input validation

- Requirement
  - ... shall validate the syntax and content of any input ...

- Rationale and supplementary guidance
  - Generally accepted industry practices for input data validation
    - out-of-range values
    - invalid characters
    - missing data
    - buffer overflow
    - ...
    - malformed packets (as commonly created by protocol fuzzers)
Security Standard IEC 62443

Overview

4-1
Product development requirements
Security Standard IEC 62443-4-1:2018

- 4-1: Product development requirements
- 4-1: Secure development life-cycle (SDLC) requirements for industrial automation and control systems (IACS)
Eight practices

- Practice 1: Security management (SM)
- Practice 2: Specification of security requirements (SR)
- Practice 3: Secure by design (SD)
- Practice 4: Secure implementation (SI)
- Practice 5: Security verification and validation testing (SVV)
- Practice 6: Management of security-related issues
- Practice 7: Security update management
- Practice 8: Security guidelines
Security Standard IEC 62443-4-1:2018

- For each practice:
  - Purpose
  - One or more requirements
    - Requirement
    - Rationale and supplemental guidance
  - The nomenclature for requirements is:
    Name of practice + number, e.g. SVV-3
SVV-3: Vulnerability testing

Requirement:

- ... tests that focus on identifying and characterizing potential security vulnerabilities ...

- Testing shall include
  - Abuse case or malformed or unexpected input testing
  - Examples include fuzz testing and network traffic load testing and capacity testing
Detect Security Vulnerabilities through Fuzzing

1. A Look at IEC 62443
2. Fuzzing
3. A Look at IEC 61508 and ISO 26262
4. Fuzzing, Robustness Testing, and Fault Injection
5. Fuzzing Example
6. Studies
7. Conclusion
Fuzzing

Definition (informal)

Throw random test data at your test object and see what happens
Fuzzing

Definition (from IEC 62443)

Process of creating **malformed** or **unexpected** data or call sequences to be consumed by the entity under test to verify that they are handled appropriately.
The Fuzzing Cycle

1. Analyse format of input
2. Create abnormal input data
3. Provide data to test object
4. Misbehavior?
   - no
   - yes
5. Log data
Those involved in fuzzing
You can fuzz almost anything

- Nodes in communication networks
  - CAN nodes
  - Web Server (HTTP)
  - SCADA (Modbus, TCP/IP, ...)
  - ...

- Software reading files
  - PDF reader
  - ...

- APIs
Fuzzing

- Fuzz data creation methods
  1. random
  2. mutation / genetic algorithm
  3. serial / exhaustive
  4. promising
Fuzzing

1. Also called “negative testing” because of its destructive nature
2. No code needed
3. Black-box testing
   - White-box is a plus
4. No specification needed
5. Not functional testing
6. Needs feedback from the SUT
7. Can detect 0-day-vulnerabilities
8. Detects only “obvious misbehavior”

Fuzzing is robustness testing
1. A Look at IEC 62443
2. Fuzzing
3. A Look at IEC 61508 and ISO 26262
4. Fuzzing, Robustness Testing, and Fault Injection
5. Fuzzing Example
6. Studies
7. Conclusion
security

robustness

fuzzing

injection

are not mentioned in 61508-4:2010 Definitions and abbreviations
Security

- security,
- security risk,
- security risk analysis,
- security threat,
- security threat analysis,
- security measures,
- ...

are mentioned 61508-1:2010 General requirements

but “for guidance see ... IEC 62443”
Security

- In annex D.2.4 of 61508-3:2010 Software requirements

The following shall be included in the safety manual...

m) Details of any security measures that may have been implemented ...
Robustness

- In a note to section 7.2.2.3 of 61508-3:2010 Software requirements
  - An adequate specification of functional behavior may include requirements for ... robustness ...

- In some notes of 61508-7:2010 Techniques and measures
  - E.g. as (one) aim of black-box testing
... is no big help
Main changes to the edition from 2011:

- ...
- references to cybersecurity
- ...

In preparation:

ISO-SAE AWI 21434 Road Vehicles – Cybersecurity Engineering
5.4.2.3 Communication channels between functional safety, cybersecurity, and other ...

Guidance in Annex E

→ Cybersecurity needs to be considered in the phases of development, but no detailed instructions
Part 1, Vocabulary

3.129 robust design

Design that can function correctly in the presence of invalid inputs or stressful environmental conditions

NOTE Robustness can be understood as follows

- For software, robustness is the ability to respond to abnormal inputs and conditions;

Fuzzing is robustness testing
Part 6, Software

- Robustness
  - 6.4.1, Example 2
    “Robustness against erroneous inputs”
  - 8.4.5 f) robustness
    “Methods to prevent implausible values, ...”
  - 9.4.2, Example
    “Evidence of the effective implementation of the error detection and error handling mechanisms specified to achieve robustness ... against erroneous inputs.”
  - 10.4.2 d) Example
    “Reliability due to ... robustness against erroneous inputs ...”

Fuzzing is robustness testing
Part 1, Vocabulary

3.54 fault

abnormal condition that can cause an element ... to fail
Part 1, Vocabulary

3.57 fault injection

method to evaluate the effect of a fault within an element by inserting faults, ..., in order to observe the reaction by observation points

New in 26262:2018, not present in 26262:2011
Part 6, Software

- Fault injection test
  - Table 7 h)

  “In the context of **software unit testing** fault injection test means to modify the tested software unit (e.g. introduce faults in the software) for the purpose of [verification].

  Such modifications include injections of **arbitrary** faults (e.g. by corrupting values of variables, by introducing code mutations, or by corrupting values of CPU registers).”

Fault injection test is robustness testing

Is fuzzing fault injection testing?
Detect Security Vulnerabilities through Fuzzing

1. A Look at IEC 62443
2. Fuzzing
3. A Look at IEC 61508 and ISO 26262
4. Fuzzing, Robustness Testing, and Fault Injection
5. Fuzzing Example
6. Studies
7. Conclusion
Fuzzing is (one kind of) robustness testing
- “... against erroneous inputs.”

Fault injection test is robustness testing
- “... arbitrary …”, “corrupting ...”

Is fuzzing (one kind of) fault injection test?
When is the injected data provided?

- Synchronous: Prior to start of test
- Asynchronous: During the test ("corrupting")
Fuzzing, Robustness Testing, Fault Injection

Do we have a specified expected result?

- Fault injection testing
  - Specified input + specified expected result
    - Requirement-based testing
  - No specified input, no specified expected result
    - Robustness testing (e.g. fuzzing)
Categorization Fault Injection Test

Fault Injection vs. Robustness Testing
1. A Look at IEC 62443
2. Fuzzing
3. A Look at IEC 61508 and ISO 26262
4. Fuzzing, Robustness Testing, and Fault Injection
5. Fuzzing Example
6. Studies
7. Conclusion
Fuzzing Example

- Commercial fuzzer: beSTORM
- The system-under-test can be HW or SW
- Approx. 180 protocols, file formats
  - Internet
  - Bluetooth
  - CAN
  - EDSA (Embedded Device Security Assurance)
  - USB
  - SCADA (Supervisory Control And Data Acquisition)
  - ...
Fuzzing the Simple Web Server

- Simple Web Server (SWS)

Document vulnerability → HTTP → SWS → Vulnerability caught? → Report vulnerability → beSTORM Monitor
Fuzzing the Simple Web Server

Start SWS

Hello World! (Request200) / (.)
Fuzzing the Simple Web Server

- Start beSTORM Monitor and attach it to the SWS

![Image of beSTORM Monitor interface]

1. Choose an item to monitor:
   - Processes
   - Services
   - TCP

2. Enter the host IP the monitor should report to:
   - Host: localhost

3. Watch monitor for errors. You can view the log file for extended information or view previous monitoring session logs.
   - Monitoring
   - Persistent Monitoring
   - Watchdog

Monitor Output:

+ VIEW LOG + BROWSE LOGS
Fuzzing the Simple Web Server

Start beSTORM Fuzzer
Fuzzing the Simple Web Server

Setup project: Protocol, host, port

The fuzzer acts as HTTP client
Fuzzing the Simple Web Server

- Select External Monitor

[Image of beSTORM New Project Wizard with checkboxes for ARP Echo, ICMP Echo, UDP Echo, and TCP Echo, and fields for Monitored IP address, Port, External Monitor IP address, Incoming Command Port, Incoming Exception Port, and Outgoing Command Port.]
Fuzzing the Simple Web Server

- Ready to fuzz

![Fuzzing the Simple Web Server](image_url)
After 48 seconds, the first vulnerability is found
Details of the first vulnerability
You can export the attack vector which revealed the vulnerability to a Perl script.
Executing the script will send the same data to the SUT

```perl
#!/usr/bin/perl

# Automatically generated by beSTORM (tm) version 8.4.8 ( 7305 )
# Copyright Beyond Security (c) 2003-2016

# Attack vector:
# M0:P0:B0.BT0:B0.BT0:B0.BT0:SE0:B0.BT0:B0.BT0:B4.BT2049:B0.BT0:B
# Module:
# HTTP (Simple Web Client)

use strict;
use warnings;

use Getopt::Std;
use IO::Socket::INET;

my @commands = (
    {Command => 'Send',
     Data => "GET / QUFBQUFBQUFBQ"
    });
```

Very useful for debugging!
You can create a report (HTML)

**beSTORM Report**

**Project Settings**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>beSTORM Version:</td>
<td>8.4.8 (build 7305)</td>
</tr>
<tr>
<td>Project Name:</td>
<td>SWS</td>
</tr>
<tr>
<td>Thread Count:</td>
<td>1</td>
</tr>
<tr>
<td>Remote Monitor IP Address / Hostname:</td>
<td>localhost</td>
</tr>
</tbody>
</table>

**Module Settings**

<table>
<thead>
<tr>
<th>Module name:</th>
<th>HTTP (Simple Web Client)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fuzzing conditioned elements:</td>
<td>Yes</td>
</tr>
<tr>
<td>Generator type:</td>
<td>Text</td>
</tr>
<tr>
<td>Increment order:</td>
<td>Normal</td>
</tr>
</tbody>
</table>

**Hostname (textual) Attack Types**

<table>
<thead>
<tr>
<th>Name</th>
<th>Replicated Buffer</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Broadcast Address</td>
<td>255.255.255.255</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Localhost Address</td>
<td>127.0.0.1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Multicast Address</td>
<td>239.255.255.253</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Nocast Address</td>
<td>0.0.0.0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

The following information was reported:

Exception reported on localhost by monitor located at localhost (15:59:39.417 06/27/2019)

During the testing of: HTTP Type
Test type: Repeated Base64 (A)

Information:
Page fault on write access to 0x42465675

Stack trace:
ERROR: SymGetSymFromAddr64, GetLastError: 126 (Address: 00000000006B4AE369)
Continued fuzzing reveals more vulnerabilities
Detect Security Vulnerabilities through Fuzzing

1. A Look at IEC 62443
2. Fuzzing
3. A Look at IEC 61508 and ISO 26262
4. Fuzzing, Robustness Testing, and Fault Injection
5. Fuzzing Example
6. Studies
7. Conclusion
Automated Testing

With Commercial Fuzzing Tools

A Study of Commercially Available Fuzzers: Identification of Undisclosed Vulnerabilities with the Aid of Commercial Fuzzing Tools

Prof. Dr. Hartmut Pohl and Daniel Baier, B.Sc.
Department of Computer Sciences,
Bonn-Rhein-Sieg University of Applied Sciences
An Evaluation of Free Fuzzing Tools

University of Oulu
Department of Information Processing
Science
Master’s Thesis
Mikko Vimpari
08.05.2015

Detect Security Vulnerabilities through Fuzzing

1. A Look at IEC 62443
2. Fuzzing
3. A Look at IEC 61508 and ISO 26262
4. Fuzzing, Robustness Testing, and Fault Injection
5. Fuzzing Example
6. Studies
7. Conclusion
Discussion

1. No test cases to specify
2. Usually fully automated
3. Independent of the functionality of the SUT
4. It can be reused easily for similar SUTs
5. Can detect 0-day-vulnerabilities
6. Non-functional, just robustness testing
7. Error detection rate depends on quality of SUT
8. Some effort to set it up (feedback)
9. Only one aspect of security (and safety) testing
10. Fuzzer and its data generation needs to be understood to be able to rate the result “finished without detected vulnerabilities”
Conclusion

- No safety without security!

- Complements other testing techniques, i.e. one aspect of security (and safety) testing

- If affordable – why skip it?
Thank you for your attention!
Any questions?
Now or at the Hitex table at the exhibition!
Frank Büchner  
Dipl.-Inform.  
Principal Engineer Software Quality  

Hitex GmbH  
Greschbachstr. 12  
Karlsruhe  
Germany  

Tel.: +49 / 721 / 9628 – 125  
frank.buechner(at)hitex.de