D4.6

Evaluation using the VESSEDIA use cases

Project number:	731453
Project acronym:	VESSEDIA
Project title:	Verification engineering of safety and security critical dynamic industrial applications
Start date of the project:	1st January, 2017
Duration:	36 months
Programme:	H2020-DS-2016-2017

Deliverable type:	Report
Deliverable reference number:	DS-01-731453 / D4.6 / 1.1
Work package contributing to the deliverable:	WP4
Due date:	December 2019 – M36
Actual submission date:	3rd March 2020

Responsible organisation:	SLAB
Editor:	Balázs Berkes
Dissemination level:	PU
Revision:	V1.1

Abstract:	This document contains the evaluation report of the VESSEDIA use-cases using the proposed security evaluation methodology for VESSEDIA project in D4.2. During the evaluation VESSEDIA tools were used besides common evaluation techniques.
Keywords:	Security, evaluation, IoT, certification, verification, tools, tooling

The project VESSEDIA has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731453.

Editor

Balázs BERKES (SLAB)

Contributors (ordered according to beneficiary numbers)

Gergely EBERHARDT (SLAB)

Balázs BERKES (SLAB)

Disclaimer

Executive Summary

The current deliverable describes the evaluation work carried out in the VESSEDIA use cases. The evaluation followed the VESSEDIA evaluation methodology, which was described in D4.2 [8] and used the Frama-C framework, the EVA plugin, WP plugin, Clang plugin, and the AFLSCA plugin from the VESSEDIA tools.

The document contains the evaluation report of the following part of the VESSEDIA use-cases:

- MQTT client and CoAP server in the new generation Contiki-OS for Inria Use Case described in D5.2 [13]
- MPL routing protocol used by the 6LoWPAN management protocol for CEA Use Case described in D5.4 [15]
- Open source TCP Proxy implementation for DA Use Case described in D5.6 [17]

During our evaluation work, we also focused on how VESSEDIA tools can be used towards the Objectives for WP4 set forth in DoA, namely to review how VESSEDIA results can be used to improve evaluation results.

We have evaluated each use case against a ***Maximum Target Security Level*** pre-established. The targets have reached the following Security Certification Levels according to the VESSEDIA evaluation methodology:

Table 1: Security Certification Levels of targets

Target	Maximum Target Level	Threats found	SCL Level (Result)
MQTT client and CoAP server (Chapter 2)	SCL 6	3	SCL 4
MPL routing protocol (Chapter 3)	SCL 4	1	SCL 4
Open source TCP Proxy (Chapter 4)	SCL 4	0	inconclusive

Our work showed that while the tools still have some shortcomings, and compatibility can be further developed, there are equally capable tools that an analyst can use as a basis of systematic search for issues. Threats uncovered were useful in one of the use cases to pinpoint real issues that would pull down the security of the developed component. We were able to provide specific recommendations for the issues so that the Use Case developer would be able to correct the security-related problems.

Due to the timing, we were not able to carry out review phase with the issues. Nevertheless, we communicated the issues with the tool developers and Use Case owners, and we will verify the resulting fixes after the submission of this report.

The evaluation of the MQTT protocol with Contiki OS in Chapter 2 resulted in an exploitable vulnerability, which is also present in the latest public release of the Contiki-NG[1]. We contacted the developers at 15 January 2020. The developers acknowledged the problem and reported that the fix will be merged as soon as possible.

[1] https://github.com/contiki-ng/contiki-ng/tree/release/v4.4

Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 VESSEDIA motivation and background

The VESSEDIA project aims to bring safety and security to the next generation of software applications and Internet connected devices. In our rapidly changing world, the Internet has been the source of many benefits for individuals and companies alike, transforming entire industries. With this new technology, capable of connecting billions of devices and people together, new threats have also appeared –VESSEDIA will help software developers to address these in order to create connected applications that are safe and secure. VESSEDIA proposes to enhance and scale up modern software analysis tools, in particular the mostly used open-source Frama-C analysis platforms, to make them useful and accessible to a wider audience of developers of connected applications. At the forefront of connected applications is the Internet of Things (or IoT for short), which has undergone explosive growth and where security risks have become all too real. VESSEDIA will focus on this domain to demonstrate the benefits our tools bring to the table when developing connected applications. VESSEDIA will tackle this challenge by 1) developing a methodology that makes it possible to adopt and use source code analysis tools as efficiently and with similar benefits as it is already possible in the case of highly-critical applications, 2) enhancing the Frama-C toolbox to enable efficient and fast implementation, 3) demonstrating the capabilities of the new toolbox on typical IoT applications, including an IoT Operating System (Contiki), 4) developing a standardisation plan for generalising the use of the toolbox, 5) contributing to the Common Criteria certification process, and 6) defining a “Verified in Europe” label for validating software products with European technologies such as Frama-C.

1.2 Structure of the document

The document can be divided into 3 major parts:

Chapter 2 contains the evaluation report of the Contiki OS use case by evaluating an MQTT client and a CoAP server example application (Inria’s use case).

Chapter 3 contains the evaluation report of the MPL routing protocol used by the 6LoWPAN use case (CEA’s use case).

Chapter 4 contains the evaluation report of the open source TCP proxy (DA’s use case).

1.3 Related deliverables

The evaluation methodology was described in D4.2 [8]. The security objectives and threat modelling were derived from the general security objectives described in D1.1 [1], from the use case specific security objectives in D1.2 [2], and from the use case final reports D5.2 [13], D5.3 [15] and D5.6 [17].

The tools used in the evaluation were described in several deliverables, FlowGuard and Frama-C in D2.1 [3], Frama-Clang plugin in D2.3 [5] and AFLSCA in D2.2 [5] and D2.4 [6].

Chapter 2 Contiki OS - Inria's use case

2.1 Use case description (Applicability Field definition)

2.1.1 Target of Evaluation

In the Security Evaluation of Inria's use case, we selected representative example applications which were included with the Contiki OS environment described in D5.2 [13] in detail.

2.1.2 Scope

In case of an operating system, the actually used parts and services highly depend on the application and the used configuration settings. Because we had limited time for this evaluation, we chose the following specific parts of the Contiki OS using simple example applications:

- CoAP protocol: a simple CoAP server application
- MQTT protocol: a simple MQTT client application

The used Contiki OS was integrated with FlowGuard as described in D5.2 [13].

2.1.3 Applicable requirements

In this section we collected requirements based on D1.2 [2], and evaluated their applicability in the current scope.

Table 2 Requirements applicable for Contiki OS (Inria Use Case)

Functional Family Name	Applicability
Cryptographic Key Management (CKM)	*No: The evaluated parts do not use cryptographic keys.*
Cryptographic Operation (COP)	*No: The evaluated parts do not use cryptographic operations.*
Access Control Policy (ACC)	*No: Evaluated protocols do not provide any access control functionality.*
Access Control Functions (ACF)	*No: Evaluated protocols do not provide any access control functionality.*
Information Flow Control Policy (IFC)	*No: No security policy was implemented in the evaluated protocols.*
Information Flow Control Functions (IFF)	*No: No security policy was implemented in the evaluated protocols.*
Import from outside TSF control (ITC)	*No: User data was imported without security attributes in the evaluated protocols.*
Residual Information Protection (RIP)	**Yes: Previous state or information from previous sessions should not be disclosed by the evaluated protocols.**
Authentication Failures (AFL)	*No: Evaluated protocols do not provide any access control functionality.*
User Attribute Definition (ATD)	*No: Evaluated protocols do not provide any access control functionality.*
Specification of Secrets (SOS)	*No: There were not secrets used during the evaluated protocols.*
User Authentication (UAU)	*No: Evaluated protocols do not provide any access control*

Functional Family Name	Applicability
	functionality.
User Identification (UID)	*No: Evaluated protocols do not provide any access control functionality.*
User-subject Binding (USB)	*No: Evaluated protocols do not provide any access control functionality.*
Management of Security Attributes (MSA)	*No: Evaluated protocols do not provide any access control functionality.*
Management of TSF Data (MTD)	*No: TSF data was not present in the evaluated protocols.*
Revocation (REV)	*No: Evaluated protocols do not provide any access control functionality.*
Specification of Management Functions (SMF)	*No: Management functions were not defined for the evaluated protocols.*
Security Management Roles (SMR)	*No: Evaluated protocols do not provide any access control functionality.*
Time Stamps (STM)	*No: Time stamps were not used in the evaluated protocols.*
Inter-TSF TSF Data Consistency (TDC)	**Yes: Data between client and server components should be consistently interpreted.**
Session Locking (SSL)	*No: User sessions were not involved in the evaluated protocols.*
Inter-TSF Trusted Channel (ITC)	*No: Trusted channel was not required for the evaluated protocols.*

2.1.4 Security objectives

In this section we collected the assets and security objectives based on D1.2 [2]. We collected the specific assets for the example applications also.

- Data assets
- Software
- Hardware
- Cryptographic assets

From the above lists, hardware and cryptographic assets were not relevant for the CoAP and MQTT examples evaluated, they were not included in the scope of this evaluation.

The following security objectives were listed for Contiki OS in D1.2 [2]:

- **Cryptographic services**
- **Identification and authentication**
- **Discretionary access control**
- **Management of security mechanisms**
- **Network information flow control**
- **Subject communication**
- **Trusted channel**

The evaluated parts provided communication protocol components, and as such, the objectives are interpreted on other layers, as described in Section 2.3 in D1.2 [2]:

> *In the case of Contiki, the notion of user does not appear at the OS level and is most of the time expressed at the application level, thus the questions of identification, authentication and discretionary access control are not considered here. This is also the case for the management of security mechanisms that relies on the identification of some kind of "administrator".*
>
> *(…)*

At the OS level, we can only partially consider the two latter objectives since the subjects are generally defined at the application layer. Thus, to fulfil the objectives, we must rely on these definitions that are use-case specific.

In this case, one can define plausible security objectives for the components in question based on the security requirements found relevant in the previous chapter.

Residual Information Protection (RIP) and Inter-TSF TSF Data Consistency (TDC) would be fulfilled by pertaining to the following objectives:

- Previous state or information from previous sessions should not be disclosed by the evaluated protocols (protecting the ***confidentiality*** of data related to previous state and sessions)
- Data between client and server components should be consistently interpreted (protecting the ***integrity*** and ***availability*** of the data assets)

2.1.5 Threat modelling

The threat modelling was carried out for the use case in D1.2 [2] Section 2.4. Once again, identified threats were only partially applicable while evaluating the COaP and MQTT communication examples. The following security threats were plausible in the current case:

- **Access user data**: A threat agent might gain access to user data stored, processed or transmitted by the TOE without being appropriately authorized according to the TOE security policy.
- **Restriction of net traffic**: A threat agent might get access to information or transmit information to other recipients via network communication channels without authorization for this communication attempt by the information flow control policy.

We tested the ToEs against these threats.

2.2 Evaluation Plan

According to the Evaluation Methodology described earlier in D4.2 [8], we define the elements for the evaluation plan in the following, and describe the test plan.

2.2.1 Test environment

The tests were performed in an Ubuntu Linux virtual machine running in Oracle VM VirtualBox. The example applications were compiled to the native architecture (x86_64).

2.2.2 Version tested

The Contiki OS version string was Contiki-NG-EACSL, which was a fork from the Contiki-NG[2] in order to perform verification activities.

2.2.3 Tools and test equipment

We used the following tools and software versions during the evaluation:

- Frama-C 19.0 (Potassium)
- Ubuntu Linux, Linux 4.15.0-54-generic x86_64 x86_64 x86_64 GNU/Linux
- AFLSCA with AFL version 2.56b and StaDy v0.5.0
- fuzzcoap, https://github.com/bsmelo/fuzzcoap

[2] https://github.com/contiki-ng/contiki-ng/releases

- Ida PRO interactive disassembler (v6.4.130306)

2.2.4 SCL Target

According to the methodology described in D4.2 [8] section 2.5.1, we set a ***Maximum Target Security Level***, which defines the efforts allocated for the evaluation activities.

The use case targets defined by the use case (D5.2 [13]) aimed at more formal verification of the target. In commercially realistic setups, such a high degree of security would necessitate a high level of ***Maximum Target Security Level*** in order to provide useful results, such as SCL 9 or above, with white-box approach.

However, in the scope of VESSEDIA project, we aimed at demonstrating more techniques with more use cases, and selecting partial examples from the use cases of the partners. In the evaluation of this use case, we used target level SCL 6. This level was defined as 10 ***Expert Days of Evaluation***, within 4 weeks of ***Execution time of Evaluation***.

While SCL Level 6 was described as grey-box or black-box evaluation, we in fact received and used the full source code for the ToEs. Our test plan thus includes evaluation activities, which is normally part of the evaluation against higher Target levels only. We have also used tools included in the VESSEDIA toolbox as defined in D4.2 [8] as well, in order to provide a use case study and demonstration of the use of the technologies developed within the project.

At the end of evaluation, we calculate the SCL level reached in our final Risk Analysis in section 2.5.1.

2.2.5 Test Plan

Based on the above collected information, we devised a test plan that included manual and automated testing using the tools also described.

The following tests were carried out against the ToE in this Evaluation:

- 2.3.1 – Fuzz testing of CoAP server with AFLSCA
- 2.3.2 – Fuzz testing of CoAP server with fuzzcoap
- 2.3.3 – Source code analysis of the MQTT protocol
- 2.3.4 – MQTT analysis with Frama-C EVA plugin

In the following chapters, we describe the test process and then, the test results.

2.3 Test cases

2.3.1 Fuzz testing of CoAP server with AFLSCA

In this test case we fuzzed the `coap_parse_message` function in the `coap.c` source file. The `coap_parse_message` had the following declaration:

```
coap_status_t coap_parse_message(coap_message_t *coap_pkt, uint8_t *data,
  uint16_t data_len)
```

The `data_len` contained the size of the data buffer and the `coap_pkt` contained the parsed information after the execution of the function. Because the `data_len` was calculated correctly we fuzzed only the data buffer with proper length information.

The AFL started by the AFLSCA plugin was performed 2.72 million executions, but it was not found any crashes.

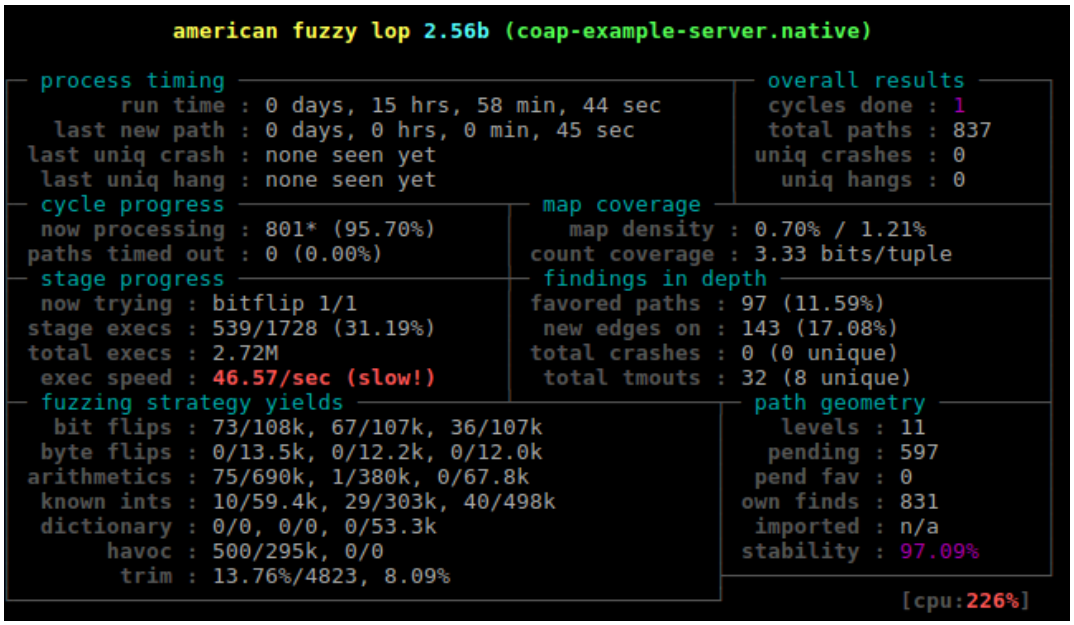

Figure 1 AFL execution on coap_parse_message function

2.3.2 Fuzz testing of CoAP server with fuzzcoap

Since AFL is a generic fuzzer, the achieved coverage was not sufficient even after 2.72 million executions. So, we tried a CoAP specific fuzzer called fuzzcoap[3], which uses the scapy to generate CoAP messages using various fuzzing techniques, such as Random, Informed Random, Mutational, Smart Mutational and Generational fuzzers. Although the fuzzcoap supported the Contiki CoAP implementation, we had to perform some modifications in the downloaded source. In the utils.py we changed the following lines:

```
TARGET_IPV6 = True
COAP_AUT_DEFAULT_DST_HOST = "fd00::302:304:506:708"
```

We performed the following commands from the fuzzcoap folder to start the fuzzing process:

```
sudo sysctl -w net.ipv6.conf.all.forwarding=1 && sudo ip tuntap add tap0 mode
  tap user ${USER} && sudo ip link set tap0 up && sudo ip tuntap add tun0 mode
  tun user ${USER} && sudo ip link set tun0 up && sudo ip address add
  2001:db8:1::a/64 dev tap0 && sudo ip address add fd00::1/64 dev tun0 && sudo
  service radvd restart
./run_monitor.sh contiki-native-erbium-plugtest
./run_fuzzer.sh -e s -t contiki-native-erbium-plugtest out-get/contiki3
```

The fuzzcoap found 2 crashes in the performed 22950 test cases and provided the following summary:

```
Using 236 as seed

AUT-specific Strings (user-defined):
[]
Extracted Paths:
['test/push', '.well-known/core', 'test/sub', 'debug/mirror', 'test/separate',
  'test/b1sepb2', 'test/chunks', 'test/hello']
Extracted Strings:
```

[3] https://github.com/bsmelo/fuzzcoap

```
['rt="Text"', 'rt="Data"', 'title="Sub-resource demo"', 'title="Hello world:
  ?len=0.."', 'title="Blockwise demo"', 'ct=40', 'title="Block1 + Separate +
  Block2 demo"', 'title="Periodic demo"', 'title="Separate demo"',
  'title="Returns your decoded message"', 'rt="Debug"', 'obs']


 Option Name                Templates/Generators              Test Cases
 string                                       16                    7600
 opaque                                       12                      48
 uint                                         18                    8226
 empty                                        14                      28
 payload                                      14                     994
 field                                         6                    6054
 ====================   ======================   ======================
 Total                                        80                   22950


Crashes for option string model StrEmpty: 0 (TCs Executed:475/475)
Crashes for option string model StrAddNonPrintable: 0 (TCs Executed:475/475)
Crashes for option string model StrOverflow: 0 (TCs Executed:475/475)
Crashes for option string model StrPredefined_ : 0 (TCs Executed:475/475)
Crashes for option string model StrPredefined_8: 0 (TCs Executed:475/475)
Crashes for option string model StrPredefined_#: 0 (TCs Executed:475/475)
Crashes for option string model StrPredefined_ðŸ˜• : 0 (TCs Executed:475/475)
Crashes for option string model StrPredefined_%: 0 (TCs Executed:475/475)
Crashes for option string model RandKTarget_StrEmpty: 0 (TCs Executed:475/475)
Crashes for option string model RandKTarget_StrAddNonPrintable: 0 (TCs
  Executed:475/475)
Crashes for option string model RandKTarget_StrOverflow: 0 (TCs
  Executed:475/475)
Crashes for option string model RandKTarget_StrPredefined_ : 0 (TCs
  Executed:475/475)
Crashes for option string model RandKTarget_StrPredefined_8: 1 (TCs
  Executed:475/475)
Crashes for option string model RandKTarget_StrPredefined_#: 0 (TCs
  Executed:475/475)
Crashes for option string model RandKTarget_StrPredefined_ðŸ˜• : 0 (TCs
  Executed:475/475)
Crashes for option string model RandKTarget_StrPredefined_%: 0 (TCs
  Executed:475/475)
Crashes for option string: 1
Total time for option string: 335.89956s

Crashes for option opaque model OpaqueEmpty: 0 (TCs Executed:4/4)
Crashes for option opaque model OpaqueOverflow: 0 (TCs Executed:4/4)
Crashes for option opaque model OpaquePredefined_ : 0 (TCs Executed:4/4)
Crashes for option opaque model OpaquePredefined_ÿ: 0 (TCs Executed:4/4)
Crashes for option opaque model OpaquePredefined_ðŸ˜• : 0 (TCs Executed:4/4)
Crashes for option opaque model OpaquePredefined_%: 0 (TCs Executed:4/4)
Crashes for option opaque model RandKTarget_OpaqueEmpty: 0 (TCs Executed:4/4)
Crashes for option opaque model RandKTarget_OpaqueOverflow: 0 (TCs Executed:4/4)
Crashes for option opaque model RandKTarget_OpaquePredefined_ : 0 (TCs
  Executed:4/4)
Crashes for option opaque model RandKTarget_OpaquePredefined_ÿ: 0 (TCs
  Executed:4/4)
Crashes for option opaque model RandKTarget_OpaquePredefined_ðŸ˜• : 0 (TCs
  Executed:4/4)
Crashes for option opaque model RandKTarget_OpaquePredefined_%: 0 (TCs
  Executed:4/4)
Crashes for option opaque: 0
Total time for option opaque: 2.74745s
```

```
Crashes for option uint model UintNull: 0 (TCs Executed:457/457)
Crashes for option uint model UintAbsoluteMinusOne: 0 (TCs Executed:457/457)
Crashes for option uint model UintAbsoluteOne: 0 (TCs Executed:457/457)
Crashes for option uint model UintAbsoluteZero: 0 (TCs Executed:457/457)
Crashes for option uint model UintAddOne: 0 (TCs Executed:457/457)
Crashes for option uint model UintSubtractOne: 0 (TCs Executed:457/457)
Crashes for option uint model UintMaxRange: 0 (TCs Executed:457/457)
Crashes for option uint model UintMinRange: 0 (TCs Executed:457/457)
Crashes for option uint model UintMaxRangePlusOne: 0 (TCs Executed:457/457)
Crashes for option uint model RandKTarget_UintNull: 0 (TCs Executed:457/457)
Crashes for option uint model RandKTarget_UintAbsoluteMinusOne: 0 (TCs
  Executed:457/457)
Crashes for option uint model RandKTarget_UintAbsoluteOne: 0 (TCs
  Executed:457/457)
Crashes for option uint model RandKTarget_UintAbsoluteZero: 0 (TCs
  Executed:457/457)
Crashes for option uint model RandKTarget_UintAddOne: 0 (TCs Executed:457/457)
Crashes for option uint model RandKTarget_UintSubtractOne: 0 (TCs
  Executed:457/457)
Crashes for option uint model RandKTarget_UintMaxRange: 0 (TCs Executed:457/457)
Crashes for option uint model RandKTarget_UintMinRange: 0 (TCs Executed:457/457)
Crashes for option uint model RandKTarget_UintMaxRangePlusOne: 0 (TCs
  Executed:457/457)
Crashes for option uint: 0
Total time for option uint: 331.75852s

Crashes for option empty model EmptyAbsoluteMinusOne: 0 (TCs Executed:2/2)
Crashes for option empty model EmptyAbsoluteOne: 0 (TCs Executed:2/2)
Crashes for option empty model EmptyAbsoluteZero: 0 (TCs Executed:2/2)
Crashes for option empty model EmptyPredefined_ÿ: 0 (TCs Executed:2/2)
Crashes for option empty model EmptyPredefined_#: 0 (TCs Executed:2/2)
Crashes for option empty model EmptyPredefined_ðŸ˜• : 0 (TCs Executed:2/2)
Crashes for option empty model EmptyPredefined_%: 0 (TCs Executed:2/2)
Crashes for option empty model RandKTarget_EmptyAbsoluteMinusOne: 0 (TCs
  Executed:2/2)
Crashes for option empty model RandKTarget_EmptyAbsoluteOne: 0 (TCs
  Executed:2/2)
Crashes for option empty model RandKTarget_EmptyAbsoluteZero: 0 (TCs
  Executed:2/2)
Crashes for option empty model RandKTarget_EmptyPredefined_ÿ: 0 (TCs
  Executed:2/2)
Crashes for option empty model RandKTarget_EmptyPredefined_#: 0 (TCs
  Executed:2/2)
Crashes for option empty model RandKTarget_EmptyPredefined_ðŸ˜• : 0 (TCs
  Executed:2/2)
Crashes for option empty model RandKTarget_EmptyPredefined_%: 0 (TCs
  Executed:2/2)
Crashes for option empty: 0
Total time for option empty: 1.54114s

Crashes for option payload model PayloadEmpty: 0 (TCs Executed:71/71)
Crashes for option payload model PayloadAddNonPrintable: 0 (TCs Executed:71/71)
Crashes for option payload model PayloadPredefined_ : 0 (TCs Executed:71/71)
Crashes for option payload model PayloadPredefined_ÿ: 0 (TCs Executed:71/71)
Crashes for option payload model PayloadPredefined_#: 0 (TCs Executed:71/71)
Crashes for option payload model PayloadPredefined_ðŸ˜• : 0 (TCs Executed:71/71)
Crashes for option payload model PayloadPredefined_%: 0 (TCs Executed:71/71)
Crashes for option payload model RandKTarget_PayloadEmpty: 0 (TCs
  Executed:71/71)
Crashes for option payload model RandKTarget_PayloadAddNonPrintable: 0 (TCs
  Executed:71/71)
```

```
Crashes for option payload model RandKTarget_PayloadPredefined_ : 0 (TCs
  Executed:71/71)
Crashes for option payload model RandKTarget_PayloadPredefined_ÿ: 0 (TCs
  Executed:71/71)
Crashes for option payload model RandKTarget_PayloadPredefined_#: 0 (TCs
  Executed:71/71)
Crashes for option payload model RandKTarget_PayloadPredefined_ðŸ˜• : 0 (TCs
  Executed:71/71)
Crashes for option payload model RandKTarget_PayloadPredefined_%: 0 (TCs
  Executed:71/71)
Crashes for option payload: 0
Total time for option payload: 56.20460s

Crashes for option field model FieldNull: 0 (TCs Executed:1009/1009)
Crashes for option field model FieldRemove: 0 (TCs Executed:1009/1009)
Crashes for option field model FieldDuplicate: 0 (TCs Executed:1009/1009)
Crashes for option field model RandKTarget_FieldNull: 0 (TCs Executed:1009/1009)
Crashes for option field model RandKTarget_FieldRemove: 1 (TCs
  Executed:1009/1009)
Crashes for option field model RandKTarget_FieldDuplicate: 0 (TCs
  Executed:1009/1009)
Crashes for option field: 1
Total time for option field: 300.90116s

Total Time: 1029.05417s
```

According to the packets.log, the crashes were caused by the following packets:

```
TC: 6089
0000  48016937BA7B0756A86AD6CDBB2E7765 H.i7.{.V.j....we
0010  6C6C2D6B6E6F776E04636F7265447274 ll-known.coreDrt
0020  3D2A8172                         =*.r

TC: 21472
0000  4802037D91EFF77FD74F258EB4746573 H..}.....O%..tes
0010  740568656C6C6F10B1E2             t.hello...
```

To test the crashes, we sent these packets again to the CoAP server, but during this execution, the crashes did not happen, so these crashes were false positives.

2.3.3 Source code analysis of the MQTT protocol

Every incoming control packet was parsed by the `tcp_input` function. This function called the various handler functions after the packet was received. First, it parsed the fixed header, which contains the header byte and the payload length.

After the payload length was calculated, the `tcp_input` function copied data from the TCP buffer to the payload buffer. The size of the copied data was correctly limited to the maximum size of the payload buffer. However, the code contained a debug snippet, which printed out the copied bytes using the following code:

```
    uint8 t i;
    DBG("MQTT - Copied bytes: \n");
    for(i = 0; i < copy_bytes; i++) {
      DBG("%02X ", conn->in_packet.payload[i]);
    }
    DBG("\n");
```

Because the `i` was defined as `uint8`, if the `copy_bytes` value is larger than 255 **the `i` will be overflow** and the `for` loop won't be finished.

Depending on the used optimizations, an empty `for` loop will be compiled into the binary even if the DBG function is switched off.

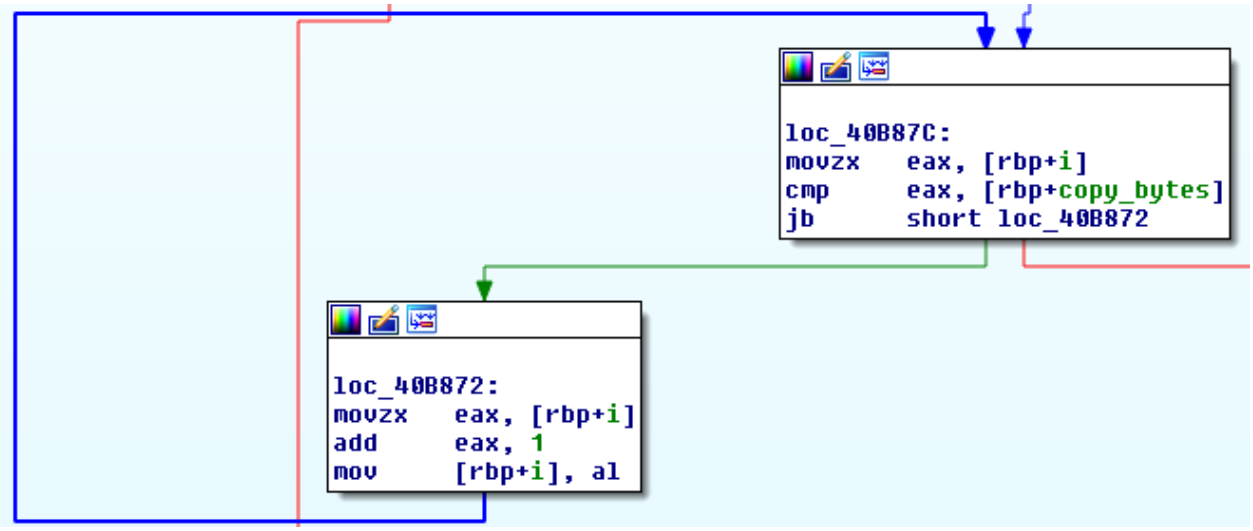

Figure 2 Compiled empty for loop

As a result, the `mqtt-client` will be stuck in an endless loop in case a large packet was sent by the server.

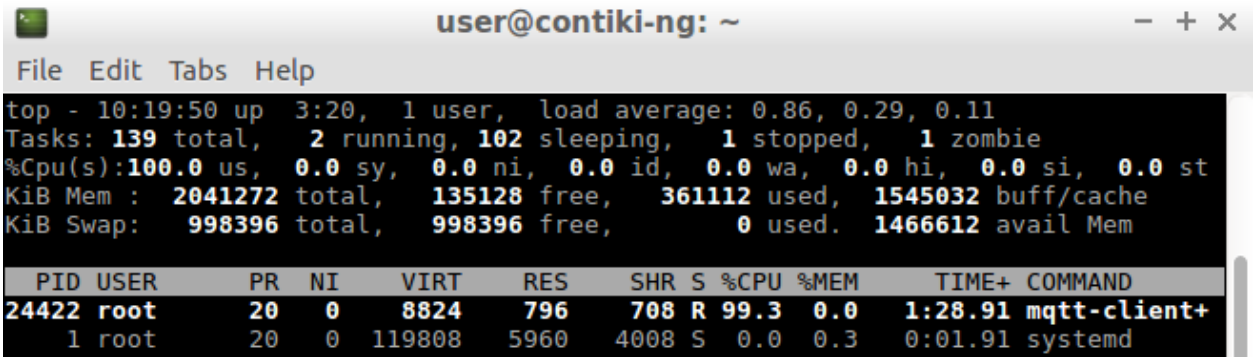

Figure 3 mqtt-client in endless loop

After the payload bytes were copied, the `tcp_input` function has been checked to see whether the payload buffer was filled up. This would happen in case of `PUBLISH MQTT` message when the size of the message payload is larger than 512 bytes. If the buffer was filled up, the `handle_publish` was called without checking the actual message type. To protect the client against calling `handle_publish` incorrectly, the size of the packet was checked before the payload was copied using the following code:

```
  if((conn->in_packet.remaining_length > MQTT_INPUT_BUFF_SIZE) &&
     (conn->in_packet.fhdr & 0xF0) != MQTT_FHDR_MSG_TYPE_PUBLISH) {
```

According to this code, the remaining length could be larger than the `MQTT_INPUT_BUFF_SIZE` (512) only if the packet type was `PUBLISH`. But, the check before calling `handle_publish` verified whether the payload buffer was filled up using the following code:

```
if(MQTT_INPUT_BUFF_SIZE - conn->in_packet.payload_pos == 0) {
```

So, if the remaining length contained exactly the `MQTT_INPUT_BUFF_SIZE`, the first check would be met, and thus the full payload buffer would be read. To verify this problem we constructed a `PUBACK` MQTT packet with 512 bytes length and 512 bytes ‘a’ in the payload and we fixed the endless loop problem described above.

After sending the modified `PUBACK` packet, the client logged the following:

```
MQTT - Read VHDR '40'
MQTT - Read Remaining Length byte
MQTT - Read Remaining Length byte
MQTT - Finished reading remaining length byte 512
MQTT - Input data len: 512
MQTT - Pos: 3
MQTT - MQTT_INPUT_BUFF_SIZE: 512
MQTT - payload_pos: 0
MQTT - byte_counter: 3
- Copied 509 payload bytes
MQTT - Payload pos before: 0
MQTT - Payload pos: 253
MQTT - Copied bytes:
61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61
   61 61 61 61 61 61 61 61 ...
```

As it is seen from the log, the client received the `PUBACK` header (VHDR=0x40), read the remaining byte, which was 512 and copied 509 bytes (remaining length was decreased with the header size) to the payload buffer. Since the payload buffer was not full at this step, it did not call the handler function. As it was showed before, the status of the payload buffer was checked by the `payload_pos` field in the packet. According to the client log, the `payload_pos` was 253 after copying 509 bytes. Since it should be 509, we checked the `payload_pos` modification in more depth and it turned out that it was declared in the `mqtt_in_packet` structure as uint8, so **the position of the payload data was overflowed** (509-256=253).

```
struct mqtt_in_packet {
  /* Used by the list interface, must be first in the struct. */
  struct mqtt_connection *next;

  /* Total bytes read so far. Compared to the remaining length to to decide when
   * we've read the payload. */
  uint32_t byte_counter;
  uint8_t packet_received;
```

Finally, the `uint8` declaration made impossible to call `handle_publish` function in case of other message types than `PUBLISH`, but it also caused incorrect functionality.

During our tests, we found another way to bypass the length verification for the size remaining. If we specified the remaining length as a small value (e.g. 4), but sent a large packet, the copied bytes and the `payload_pos` was calculated based on the received bytes and not the remaining length.

```
MQTT - Read VHDR '40'
MQTT - Read Remaining Length byte
MQTT - Finished reading remaining length byte 4
MQTT - Input data len: 512
MQTT - Pos: 2
MQTT - MQTT_INPUT_BUFF_SIZE: 512
MQTT - payload_pos: 0
MQTT - byte_counter: 2
- Copied 510 payload bytes
MQTT - Payload pos before: 0
MQTT - Payload pos: 254
MQTT - Copied bytes:
61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61
   61 61 61
```

An MQTT client has to handle the following control packet types received from the server:

- `PUBLISH`: Publish message

- `PUBREC`: Publish received (not supported)
- `PUBREL`: Publish release (not supported)
- `PUBCOMP`: Publish complete (not supported)
- `PINGRESP`: Ping response

And the following acknowledgment control packets:

- `CONNACK`: Connection acknowledgment
- `PUBACK`: Publish acknowledgment
- `SUBACK`: Subscribe acknowledgment
- `UNSUBACK`: Unsubscribe acknowledgment

Acknowledgement control packets have a very simple and fixed format, which was parsed by the `handle_connack`, the `handle_suback`, the `handle_unsuback` and the `handle_puback` functions.

Since only the `PUBLISH` and `PINGRESP` were supported, except from the acknowledgment packets, we focused the handling of these control packet types.

The `PINGRESP` was parsed by the `handle_pingresp` function, which was only logged that the `PINGRESP` was received.

The `PUBLISH` was parsed by the `handle_publish` function and only called the `PUBLISH` event and reset the incoming packet.

2.3.4 MQTT analysis with Frama-C EVA plugin

Since the `tcp_input` was the main function of the MQTT message parsing, we analysed this function using Frama-C EVA plugin. Without annotating the function, the Frama-C generated 63 yellow alarms on the 253 lines long function. From the 63 alarms 41 were memory access problems, which were caused that there was not any assumption about the incoming buffers. For example the following memory access can be invalid if the received connection pointer (ptr function parameter) was not initialized correctly by the caller:

```
static int tcp_input(struct tcp_socket *s_0, void *ptr,
                     uint8_t const *input_data_ptr, int input_data_len)
{
  int __retres;
  uint8_t byte;
  struct mqtt_connection *conn_0 = (struct mqtt_connection *)ptr;
  uint32_t pos = (unsigned int)0;
  uint32_t copy_bytes = (unsigned int)0;
  if (input_data_len == 0) {
    __retres = 0;
    goto return_label;
  }
  /*@ assert Eva: mem_access: \valid_read(&conn_0->in_packet.packet_received);
  */
  if (conn_0->in_packet.packet_received) {
    reset_packet(& conn_0->in_packet);
  }
```

Figure 4 Memory access alarm

Because we assumed that the caller initialized the pointers and structures correctly we focused on the remaining 22 alarms.

The following two signed overflow alarms were generated because the `remaining_length_bytes` was declared as `uint8` (unsigned char) and may be overflowed after reading more than 255 remaining bytes.

```
/*@ assert
    Eva: signed_overflow:
      -2147483648 ≤
      (int)conn_0->in_packet.remaining_length_bytes + 1;
*/
/*@ assert
    Eva: signed_overflow:
      (int)conn_0->in_packet.remaining_length_bytes + 1 ≤
      2147483647;
*/
conn_0->in_packet.remaining_length_bytes = (unsigned char)(
(int)conn_0->in_packet.remaining_length_bytes + 1);
```

Figure 5 Invalid signed overflow alarm

However, this overflow cannot happen because the next line verified that the `byte_counter` of the packet is larger than 5, which means that 4 remaining byte values have been read. So the signed overflow alerts were false positives.

The remaining length calculation generated 4 signed overflow alarms, because the `remaining_length` was declared as unsigned short and it can be overflow easily. The MQTT standard [19] allows remaining length value up to 268435455 bytes, which will not fit into the `remaining_length` variable. So, the alarms were valid, but because of other verifications (see above) it could cause only functional issues as the MQTT client will not be able to handle large PUBLISH messages correctly.

```
/*@ assert
    Eva: signed_overflow:
      -2147483648 ≤
      (int)((int)byte & 127) *
      (int)conn_0->in_packet.remaining_multiplier;
*/
/*@ assert
    Eva: signed_overflow:
      (int)((int)byte & 127) *
      (int)conn_0->in_packet.remaining_multiplier ≤ 2147483647;
*/
/*@ assert
    Eva: signed_overflow:
      -2147483648 ≤
      (int)conn_0->in_packet.remaining_length +
      (int)((int)((int)byte & 127) *
            (int)conn_0->in_packet.remaining_multiplier);
*/
/*@ assert
    Eva: signed_overflow:
      (int)conn_0->in_packet.remaining_length +
      (int)((int)((int)byte & 127) *
            (int)conn_0->in_packet.remaining_multiplier)
      ≤ 2147483647;
*/
conn_0->in_packet.remaining_length = (unsigned short)((int)conn_0->in_packet.remaining_length +
                                                      ((int)byte & 127) * (int)conn_0->in_packet.remaining_multiplier);
```

Figure 6 Remaining length calculation alarms

The `remaining_multiplier` had the same problem as the `remaining_length`. It was declared as unsigned char, which overflowed after the second remaining byte. The `remaining_multiplier` was used to calculate the `remaining_length` value, so it could cause functional problem only.

```
/*@ assert
    Eva: mem_access: \valid(&conn_0->in_packet.remaining_multiplier);
*/
/*@ assert
    Eva: signed_overflow:
      -2147483648 ≤
      (int)conn_0->in_packet.remaining_multiplier * 128;
*/
/*@ assert
    Eva: signed_overflow:
      (int)conn_0->in_packet.remaining_multiplier * 128 ≤
      2147483647;
*/
conn_0->in_packet.remaining_multiplier = (unsigned char)((int)conn_0->in_packet.remaining_multiplier * 128);
```

Figure 7 Remaining multiplier calculation alarms

The following alarm was caused by a possible integer overflow in the verification of last PUBLISH packet. Although the alarm was correct, it was not possible, because the `remaining_length`

cannot be larger than 268,435,455 according to the specification and 65,535 according to the `remaining_length` declaration.

```
/*@ assert
    Eva: signed_overflow:
      -2147483648 ≤ 1 + (int)conn_0->in_packet.remaining_length;
*/
/*@ assert
    Eva: signed_overflow:
      1 + (int)conn_0->in_packet.remaining_length ≤ 2147483647;
*/
if (conn_0->in_packet.byte_counter >= (unsigned int)(1 + (int)conn_0->in_packet.remaining_length)) {
```

Figure 8 Verification of last PUBLISH packet

Exactly the same problem was found at the start of the payload copy loop, where the packet end was checked.

```
/*@ assert
    Eva: signed_overflow:
      -2147483648 ≤ 1 + (int)conn_0->in_packet.remaining_length;
*/
/*@ assert
    Eva: signed_overflow:
      1 + (int)conn_0->in_packet.remaining_length ≤ 2147483647;
*/
if (! (conn_0->in_packet.byte_counter < (unsigned int)(1 + (int)conn_0->in_packet.remaining_length))) {
  break;
```

Figure 9 Verification of packet end

The calculation of the payload bytes count was also alerted because of possible integer overflow. However, it was not possible, because the `payload_pos` cannot be larger than 512. It was guaranteed by additional checks and the fact that the `payload_pos` was declared as `uint8` (unsigned char).

```
/*@ assert
    Eva: signed_overflow:
      -2147483648 ≤ 512 - (int)conn_0->in_packet.payload_pos;
*/
/*@ assert
    Eva: signed_overflow:
      512 - (int)conn_0->in_packet.payload_pos ≤ 2147483647;
*/
if ((unsigned int)input_data_len - pos < (unsigned int)(512 - (int)conn_0->in_packet.payload_pos)) {
  copy_bytes = (unsigned int)input_data_len - pos;
}
else {
  /*@ assert
      Eva: mem_access: \valid_read(&conn_0->in_packet.payload_pos);
  */
  /*@ assert
      Eva: signed_overflow:
        -2147483648 ≤ 512 - (int)conn_0->in_packet.payload_pos;
  */
  /*@ assert
      Eva: signed_overflow:
        512 - (int)conn_0->in_packet.payload_pos ≤ 2147483647;
  */
  copy_bytes = (unsigned int)(512 - (int)conn_0->in_packet.payload_pos);
}
```

Figure 10 Calculation of payload bytes count

Since the DBG was not defined, the `for` loop, which printed out the copied payload, was an empty loop. Although the casting was correctly identified during the preprocessing step, the EVA plugin did not raise any alert for the integer overflow.

```
i = (unsigned char)0;
while ((unsigned int)i < copy_bytes) {
  i = (unsigned char)((int)i + 1);
}
```

Figure 11 Debug loop to print out payload bytes

The `payload_pos` was also marked as possible integer overflow in case of the verification of whether the payload buffer was filled up fully. Similarly to the previous cases, the alert was not valid, because the `payload_pos` cannot be larger than 512.

```
/*@ assert
    Eva: signed_overflow:
      -2147483648 ≤ 512 - (int)conn_0->in_packet.payload_pos;
*/
/*@ assert
    Eva: signed_overflow:
      512 - (int)conn_0->in_packet.payload_pos ≤ 2147483647;
*/
if (512 - (int)conn_0->in_packet.payload_pos == 0) {
```

Figure 12 Full buffer verification

The `payload_left` variable was filled up by the variable header parser of the `PUBLISH` message. Since it received the value of the `remaining_length` at initialization, this branch could be executed only if the `remaining_length` was larger than 512 bytes. In this case the `payload_left` would not be overflowed.

```
/*@ assert
    Eva: signed_overflow:
      -2147483648 ≤ (int)conn_0->in_publish_msg.payload_left - 512;
*/
/*@ assert
    Eva: signed_overflow:
      (int)conn_0->in_publish_msg.payload_left - 512 ≤ 2147483647;
*/
conn_0->in_publish_msg.payload_left = (unsigned short)((int)conn_0->in_publish_msg.payload_left - 512);
handle_publish(conn_0);
conn_0->in_publish_msg.payload_chunk = conn_0->in_packet.payload;
conn_0->in_packet.payload_pos = (unsigned char)0;
```

Figure 13 Payload_left calculation

The following alert was also a false positive, because the `remaining_length` cannot be overflow after casting to integer, since its value cannot be larger than 512.

```
/*@ assert
    Eva: signed_overflow:
      -2147483648 ≤ 1 + (int)conn_0->in_packet.remaining_length;
*/
/*@ assert
    Eva: signed_overflow:
      1 + (int)conn_0->in_packet.remaining_length ≤ 2147483647;
*/
if (conn_0->in_packet.byte_counter < (unsigned int)(1 + (int)conn_0->in_packet.remaining_length)) {
  __retres = 0;
```

Figure 14 Remaining data verification

In the following table, we collected the MQTT related findings in the `tcp_input` function:

Table 3 MQTT related findings by Frama-C EVA plugin

Description of finding/alarm	Line number	Found by Frama-C	Valid	Security problem
Integer overflow in the `remaining_length_bytes` calculation.	963	Yes	***No***	No
Integer overflow in the `remaining_length_bytes` calculation.	972	Yes	Yes	No
Integer overflow in the `remaining_multiplier` calculation.	974	Yes	Yes	No
Integer overflow in the `remaining_length` calculation.	993	Yes	Yes	No
Integer overflow in the `remaining_length` calculation.	1006	Yes	Yes	No

Description of finding/alarm	Line number	Found by Frama-C	Valid	Security problem
Integer overflow in the `copy_bytes` calculation.	1015	Yes	***No***	No
Integer overflow in the `payload_pos` calculation.	1022	***No***	Yes	No
Integer overflow in the copied bytes print out loop.	1027	***No***	Yes	***Yes***
Integer overflow in the buffer verification.	1033	Yes	***No***	No
Integer overflow in the `payload_left` calculation.	1036	Yes	***No***	No
Integer overflow in the remaining data verification.	1045	Yes	***No***	No

During the evaluation of the client-side MQTT protocol implementation, we found one integer overflow vulnerability, which caused Denial of Service against the client process. We also found 5 other integer overflow vulnerabilities, which caused functional problems.

2.4 Findings and recommendations

2.4.1 Possible DoS attack in case of large MQTT messages

In case of a large MQTT message payload, if the payload size is larger than 255 bytes, the index variable in debug print out function overflowed and the print out was never stopped, which caused a Denial-of-Service type attack against the MQTT client application. An attacker might use this vulnerability to block user data.

Recommendation

Change the declaration of the `i` variable at line `mqtt.c:1114` from

```
uint8_t i;
```

to

```
uint32_t i;
```

2.4.2 Multiple integer overflows cause functional problems

During the analysis we found integer overflow in the `remaining_length_bytes` calculation, in the `remaining_multiplier` calculation, in the `remaining_length` calculation and in the `payload_pos` calculation, which caused incorrect handling of large MQTT payloads.

Recommendation

Check variable declarations in the MQTT related structures and use proper variable types, which can handle the maximum values correctly.

2.4.3 Publish handler may be called in case of other commands also

In case the payload buffer is full, the registered publish handler is called with the partial payload data. Since the verification was based on the size of the payload, we found multiple ways to bypass this check, which would cause incorrect calls of the publish handler. Because of the multiple integer overflows in the code, vulnerable cases actually never happen.

Recommendation

Check the command type, before the publish handler would be called at line `mqtt.c:1127`.

2.5 Risk analysis

In this section we enumerate the findings that we introduced in 2.4, and analyse their risk by examining the severity and likelihood of their occurrence. The **severity** level corresponds to the items mentioned below:

- *Low*: Vulnerabilities that cannot be exploited or can only result in unexpected (functional) errors. Minor data leakage, user misleads or transient denial-of-service type attack.
- *Medium*: Leakage of confidential information or unwarranted access to system resources. Denial-of-service that affects multiple users.
- *High*: Subversion of system components or code execution. Permanent denial-of-service type attacks.

We categorized the **likelihood** with the following levels:

- *Very low (VL)*: Infeasible attack scenarios or very rare events, which require using zero-day vulnerabilities or weaknesses of trusted components.
- *Low (L)*: Rare events. The attacker needs detailed knowledge about the system, or needs special equipment. Some of these events may only be performed with the help of an insider.
- *Medium (M)*: The event may happen. The attacker only needs normal knowledge about the system and the attack can be performed with normally available equipment.
- *High (H)*: The event occurs quite often. The attacker only needs minor knowledge about the system and does not need any additional equipment. The event can occur due to wrong or careless usage.

Finally, we calculated the risk of each threat using the standard likelihood × severity risk calculation using the table below.

Table 4 Risk level calculation

Likelihood / Severity	Very Low	Low	Medium	High
Low	Very Low	Low	Medium	High
Medium	Low	Medium	High	Very High
High	Medium	High	Very High	Catastrophic

The **risk** value of each threat can take the following levels:

- *Very Low* (VL): The threat has a very minor effect on the security of the asset.
- *Low* (L): The threat has a minor effect on the security of the asset.
- *Medium* (M): The threat has a noticeable effect on the security of the asset.
- *High* (H): The threat significantly endangers the asset.
- *Very high* (VH): The threat significantly endangers the asset or the system as a whole
- *Catastrophic* (C): The threat presents a critical risk to the system as a whole; if not mitigated, its effects could put the entire business process at risk.

In the table below we represented the severity, likelihood and risk values of each threat associated with our findings. We highlighted threats with Very High or Catastrophic risk.

Table 5 Risk analysis for Contiki OS (Inria Use Case)

Threat	S	L	R
2.4.1 – Possible DoS attack in case of large MQTT messages	**M**	**H**	**VH**
2.4.2 – Multiple integer overflows cause functional problems	L	M	M
2.4.3 – Publish handler may be called in case of other commands also	L	-	-

2.5.1 SCL result

Based on the assigned risk values for the threats identified, we can calculate the Security Certification level assigned as a result for this evaluation. We based our calculation on D4.2 [8] section 2.5.2.

$$R_A = \sum_{v=1}^{N} M(R_v) = M(very\ high) + M(medium) = 1 + 0.25 = 1.25$$

$$D_A = E_A / max(1, 0.5 + \mathrm{R}_A) = 10/\max(1,1.75) = 5.714$$

$$SCL_A \leq 2 * ln(1.67 * D_A) = 4.512$$

Based on these calculations, we assign the **SCL value of 4** (the floor of the above SCL_A value) to the target in the scope of this security evaluation.

Chapter 3 CEA's use case

3.1 Use case description (Applicability Field definition)

3.1.1 Target of Evaluation

In the Security Evaluation of CEA's use case, we selected representative example application which was included with the Contiki OS environment described in D5.4 [15] in detail. The Contiki OS version used was 3.1[4]. For confidentiality reasons, the source code of the firmware management on the 6LoWPAN node could not be provided to this Security Evaluation.

From the whole CEA use-case, the MPL routing was chosen for the evaluation. The MPL routing was defined in RFC 7731[5]. The routing algorithm was implemented in `core/net/ipv6/multicast/roll-tm.c` file and for the evaluation, CEA provided an annotated version of this implementation called `roll-tm_wp.c`.

3.1.2 Scope

During the evaluation, we analysed the MPL routing protocol without other parts of the use case. To perform requests using MPL routing we used the ipv6/multicast example included with the Contiki OS distribution, by sending simple data over the network.

The Contiki OS version in use by this use case was slightly different from the one used in the first Use case in Chapter 2. We did not include analysis of Contiki OS environment core services in the scope of this check.

3.1.3 Applicable requirements

In this section we collected requirements based on D1.2 [2], and evaluated their applicability in the current scope.

Table 6 Applicable requirements for CEA Use Case

Functional component	Critical functionality	Security requirement	Applicability
Application server	Transmission of firmware packets to the gateway	Authentication, data integrity and confidentiality	**Yes: MPL service employs link-layer security according to IEEE802.15.4 [21]**
	CRC value generation (over the firmware file) and transmission to the gateway	Authentication and data integrity	**Yes: MPL service employs link-layer security according to IEEE802.15.4 [21]**
Gateway	Firmware packet forwarding to LLN node	Authentication, data integrity and confidentiality	**Yes: MPL service employs link-layer security according to IEEE802.15.4 [21]**
LLN node	Notification of end of firmware update. The notification is sent to the application server via the gateway	Authentication and data integrity	**Yes: MPL service employs link-layer security according to IEEE802.15.4 [21]**

[4] https://github.com/contiki-os/contiki/tree/release-3-1
[5] https://tools.ietf.org/html/rfc7731

Functional component	Critical functionality	Security requirement	Applicability
	Read/write/delete operations on the flash	User access control	*No: Application layer functionality was not provided*
	Cryptographic key storage and management	Hardware security module	*No: Application layer functionality was not provided*

3.1.4 Security objectives

In this section we collected the assets and security objectives based on D1.2 [2]. We collected the specific assets for the example applications as well.

The following security objectives were listed for 6LowPAN Management platform:

- **Data origin authentication and integrity:** to prevent source impersonation and data forging during both firmware transfer and associated exchanges of control messages between concerned the GW and LLN nodes.
- **Data confidentiality:** in order to prevent eavesdropping and reverse engineering on the firmware image, data confidentiality needs to be enforced though data encryption.
- **Service availability:** firmware update is a critical phase that needs to be protected against DoS attacks to ensure complete firmware transmission and successful loading of the new firmware.
- **Data access control:** only authorized LLN nodes will get (and have access to) the firmware data.
- **Network access control:** only authorized LLN nodes can connect to the routing infrastructure.

3.1.5 Threat modelling

Threat modelling was carried out for the Use Case in D1.2 [2] Section 3.3.

The STRIDE threat modelling table is repeated here, with the threats applicable to our ToE and scope marked as X, and other identified threats marked as N/A.

(S: Spoofing of user identity. T: Tampering. R: Repudiation. I: Information disclosure. D: Denial of service. E: Elevation of privilege) [22].

Table 7 - STRIDE threat modelling for the 6LowPAN Management Platform

Threat description	S	T	R	I	D	E
Reverse engineering of the firmware binary image into assembly or a higher level engineering language to analyze its functionality and contents.				N/A		N/A
Product cloning where a firmware image from the product manufacturer is loaded onto a device that is not authorized.				N/A		N/A
Alteration of the firmware distributed by the product manufacturer.		X				X
Loading an unauthorized firmware image onto the device, which may correspond to an older firmware version from the product manufacturer with known bugs or code created by an unauthorized party, or firmware not intended for the specific device (firmware downgrading).					N/A	N/A
Transmitting fake image to the LLN nodes.	X		X			X
Transmitting fake notification of end of firmware transmission	X		X			X
Transmitting fake reboot command to LLN nodes	X		X			X
Unauthorized initialization/write operations on the flash memory					N/A	N/A

The analysis identified the following misuse cases:

- Transmission error: this could occurs for instance when some bits are flipped during transmission.
- Transmission failure: a typical example of such an event is when losing device power or losing connection with host during the firmware update process. This would lead to transmission failure.
- Information loss: because of the unreliable nature of LLN networks, parts of the data may be lost during firmware update process.

We based our analysis on assessing the possibility of the threats marked above, in relation to the use case example of MPL routing example.

3.2 Evaluation Plan

According to the Evaluation Methodology described earlier in D4.2 [8], we define the elements for the evaluation plan in the following, and describe the test plan.

3.2.1 Test environment

The tests were performed in an Ubuntu Linux virtual machine running in Oracle VM VirtualBox. The example applications were compiled to the native architecture (x86_64).

3.2.2 Version tested

The Contiki OS version was 3.1[6] with the annotated roll-tm.c file.

3.2.3 Tools and test equipment

We used the following tools and software versions during the evaluation:

- Frama-C 19.0 (Potassium)
- Ubuntu Linux, Linux 4.15.0-64-generic x86_64 x86_64 x86_64 GNU/Linux

3.2.4 SCL Target

According to the methodology described in D4.2 [8] section 2.5.1, we set a ***Maximum Target Security Level***, which defines the efforts allocated for the evaluation activities.

The use case targets defined by the use case (D5.4 [15]) aimed at more formal verification of the target. In commercially realistic setups, such a high degree of security would necessitate a high level of ***Maximum Target Security Level*** in order to provide useful results, such as SCL 9 or above, with white-box approach.

However, in the scope of VESSEDIA project, we aimed at demonstrating more techniques with more use cases, and selecting partial examples from the use cases of the partners. In the evaluation of this use case, we used target level SCL 4. This level was defined as 5 ***Expert Days of Evaluation***, within 3 weeks of ***Execution time of Evaluation***.

While SCL Level 4 was described as black-box evaluation, we in fact received and used the full source code for the ToEs. Our test plan thus includes evaluation activities, which are normally part of the evaluation against higher Target levels only. We have also used tools included in the VESSEDIA toolbox as defined in D4.2 [8] as well, in order to provide a use case study and demonstration of the use of the technologies developed within the project.

[6] https://github.com/contiki-os/contiki/tree/release-3-1

At the end of evaluation, we calculate the SCL level reached in our final Risk Analysis in section 3.5.1.

3.2.5 Test Plan

Based on the above collected information, we devised a test plan that included manual and automated testing using the tools also described.

The following tests were carried out against the ToE in this Evaluation:

- 3.3.1 – Evaluation with WP plugin
- 3.3.2 – Evaluation with EVA plugin

In the following chapters, we describe the test process and then, the test results.

3.3 Test cases

The TOE was the implementation of a network protocol and its design was evaluated already in Chapter 12 in RFC 7731, so we focused on the implementation problems.

3.3.1 Evaluation with WP plugin

The main source component of the routing protocol implementation (`roll-tm.c`) was partially annotated by CEA, so we were able to use the WP plugin during the evaluation. After executing the WP analysis with the received `fr_wp_mpl` script, the kernel generated 12 alarms with the following message:

```
Neither code nor specification for function clock_time, generating default
   assigns from the prototype
```

The WP plugin generated 77 further alarms in the following categories:

Table 8 Frama-C alarms by WP plugin for CEA Use Case

Alarm category	No. of alarms	Description
Missing RTE guards	1	
Cast with incompatible pointers types	62	Most of these warnings were raised by a cast to void* from a struct or a cast to struct from a buffer. Struct to void* casts were used in `memset` and `memcpy` functions, where the length was correctly set. Buffer to struct casts were used to parse the IP header. The size of the buffer was pre-allocated and it was filled by the IP and IPv6 protocol stack.
Missing assigns clause (assigns 'everything' instead)	6	Over-approximation by the WP plugin, because of a missing assign.

Alarm category	No. of alarms	Description
Logic cast to sized integer (int) from (…) not implemented yet	5	The used Frama-C version (v19) did not support all of the annotations.
Allocation, initialization and danglingness not yet implemented	3	The used Frama-C version (v19) did not support all of the annotations.

At the end of the WP analysis 55 goals out of the 154 goals were proved.

3.3.2 Evaluation with EVA plugin

Since the MPL routing parts of the Contiki OS was called at specific events, the EVA analysis could not be executed from the main of the example code. Instead of this, we executed EVA analysis to some function in the MPL routing code, which were responsible for handling the main functionality.

Table 9 Findings by Frama-C EVA plugin for CEA Use Case

Alarm message	Source:line number / function	Description
division by zero. assert (unsigned long)((unsigned long)((unsigned long)(i_min << (int)d) - 1) - min) ≠ 0;	roll-tm_wp.c:497 / random_interval	According to the EVA plugin the calculation of the random interval may result in a division by zero. By analysing the alerted function, we concluded that the division by zero can happen if the `i_min` parameter is small enough (0,1 or 2). However, the `i_min` was set to 32 or 64 only, which guaranteed that division by zero cannot be possible in practice. This analysis is extended in D5.4 [15] section 3.1.3.2.1.
pointer comparison. assert \pointer_comparable((void *)iterswptr, (void *)((struct sliding_window *)windows));	roll-tm_wp.c:1310 / icmp_input roll-tm_wp.c:1317 / icmp_input roll-tm_wp.c:1327 / icmp_input	In several places pointers were compared to check the start or the end of a buffer. In every checked place, these pointers were compared correctly.

Alarm message	Source:line number / function	Description
out of bounds read. assert \valid_read(&locslhptr->flags);	roll-tm_wp.c:1332 / icmp_input	The `locslhptr` pointer was initialized with `UIP_ICMP_PAYLOAD`, which was a pointer to the `ui_buf`. The `ui_buf` was a pre-allocated buffer. So, even if it was not filled correctly for some reason, out of bounds read was not possible.
out of bounds read. assert \valid_read(&locslhptr->seq_len);	roll-tm_wp.c:1357 / icmp_input	Similarly to the previous assert, the out of bounds read was not possible.
out of bounds read. assert \valid_read(seq_ptr);	roll-tm_wp.c:1379 / icmp_input	The `seq_ptr` was iterated over the sequence values received in the ICMP message. The message was stored in the pre-allocated `ui_buf`. The buffer size was 2042. Since the sequence length was defined as uint8, its maximum value is 255, so, the `icmp_input` function will read 510 bytes maximum from the `ui_buf`. So, out of bounds read will not happen, but because it was not verified whether the ICMP message contained so much sequence values than it was claimed, the function may read invalid data left other function in the buffer.

During our analysis we verified the `in`, the `out`, the `init`, the `icmp_input`, the `icmp_output` and the `handle_timer` functions.

3.4 Findings and recommendations

3.4.1 Sequence number was not checked in the ICMP message

We found that the sequence number was not properly checked before the sequence values would be iterated in the ICMP message. Because of the used buffer size, this could not cause out of bounds read or any security problem.

Recommendation

Perform sequence number verification.

3.5 Risk analysis

In this section we enumerate the findings that we introduced in 2.4, and analyse their risk by examining the severity and likelihood of their occurrence. The **severity** level corresponds to the items mentioned below:

- *Low*: Vulnerabilities that cannot be exploited or can only result in unexpected (functional) errors. Minor data leakage, user misleads or transient denial-of-service type attack.
- *Medium*: Leakage of confidential information or unwarranted access to system resources. Denial-of-service that affects multiple users.
- *High*: Subversion of system components or code execution. Permanent denial-of-service type attacks.

We categorized the **likelihood** with the following levels:

- *Very low (VL)*: Infeasible attack scenarios or very rare events, which require using zero-day vulnerabilities or weaknesses of trusted components.
- *Low (L)*: Rare events. The attacker needs detailed knowledge about the system, or needs special equipment. Some of these events may only be performed with the help of an insider.
- *Medium (M)*: The event may happen. The attacker only needs normal knowledge about the system and the attack can be performed with normally available equipment.
- *High (H)*: The event occurs quite often. The attacker only needs minor knowledge about the system and does not need any additional equipment. The event can occur due to wrong or careless usage.

Finally, we calculated the risk of each threat using the standard likelihood × severity risk calculation using the table below.

Table 10 Risk level calculation

Likelihood / Severity	Very Low	Low	Medium	High
Low	Very Low	Low	Medium	High
Medium	Low	Medium	High	Very High
High	Medium	High	Very High	Catastrophic

The **risk** value of each threat can take the following levels:

- *Very Low* (VL): The threat has a very minor effect on the security of the asset.
- *Low* (L): The threat has a minor effect on the security of the asset.
- *Medium* (M): The threat has a noticeable effect on the security of the asset.
- *High* (H): The threat significantly endangers the asset.
- *Very high* (VH): The threat significantly endangers the asset or the system as a whole
- *Catastrophic* (C): The threat presents a critical risk to the system as a whole; if not mitigated, its effects could put the entire business process at risk.

In the table below we represented the severity, likelihood and risk values of each threat associated with our findings. We highlighted threats with Very High or Catastrophic risk.

Table 11 Risk analysis for CEA Use Case

Threat	S	L	R
3.4.1 – Sequence number was not checked in the ICMP message	L	-	-

3.5.1 SCL result

Based on the assigned risk values for the threats identified, we can calculate the Security Certification level assigned as a result for this evaluation. We based our calculation on D4.2 [8] section 2.5.2.

$$R_A = \sum_{v=1}^{N} M(R_v) = 0 \ (as\ no\ threats\ were\ found\ with\ assignable\ risk)$$

$$D_A = E_A / max(1, 0.5 + \mathrm{R}_A) = 5/\max(1,0.5) = 5.000$$

$$SCL_A \leq 2 * ln(1.67 * D_A) = 4.245$$

Based on these calculations, we assign the **SCL value of 4** (the floor of the above SCL_A value) to the target in the scope of this security evaluation.

Here, evaluation effort was more limited than in Inria Use Case, where the result was also SCL 4 – see section 2.5.1. In that use case, more vulnerabilities found limited the level for the SCL Target 6 down to SCL 4. In this case, the Target level was reached, since no actual threats with assignable risk value were found. This also means that the confidence of this SCL level is slightly lower, than in the first use case, due to more effort spent in that use case.

Chapter 4 DA's use case

4.1 Use case description (Applicability Field definition)

4.1.1 Target of Evaluation

In the Security Evaluation of DA's use case, we received a representative example application to be tested instead of the original confidential source, upon which the original Use Case described in D5.6 was based. For confidentiality reasons, the original source code could not be provided to this Security Evaluation. DA provided the following representative component, which was used as a stand-in instead of DA's original source.

The target we received was a pre-processed and headerless source version of tcp_proxy open-source component. It consisted of parts from the sources described below.

Tcp_proxy_preprocessed.cpp was a 6.5MB, 195031-line source file. It was based on the following C++ TCP Proxy Server: http://www.partow.net/programming/tcpproxy/index.html.

The main source itself was a 331-line CPP file, as can be found in:

https://github.com/ArashPartow/proxy/blob/29d09e6bcef563b2e03a4100346c055e9a4128f6/tcpproxy_server.cpp

Committed by ArashPartow on Jan 1, 2017 (http://www.partow.net) .

It included the following components:

```
#include <cstdlib>
#include <cstddef>
#include <iostream>
#include <string>

#include <boost/shared_ptr.hpp>
#include <boost/enable_shared_from_this.hpp>
#include <boost/bind.hpp>
#include <boost/asio.hpp>
#include <boost/thread/mutex.hpp>
```

The preprocessing took place under Linux 4.15.0-45-generic x86_64 (Ubuntu 18.04.2 LTS) version[7].

The libraries used in the preprocessing step were:

- gcc version 7.4.0 (Ubuntu 7.4.0-1ubuntu1~18.04.1)
  - GCC v7.4 Dec 6, 2018[8]
- libboost-all-dev/bionic 1.65.1.0ubuntu1 amd64
  - boost v1.65.1 September 7th, 2017 17:31 GMT[9]

The standard libraries preprocessed into the ToE were part of GCC, compliant to ISO/IEC 14882:2017[10], and they were similar to this version:

cstdlib - https://gcc.gnu.org/onlinedocs/libstdc++/latest-doxygen/a00242_source.html

[7] http://old-releases.ubuntu.com/releases/18.04.2/
[8] https://gcc.gnu.org/gcc-7/
[9] https://www.boost.org/users/history/version_1_65_1.html
[10] https://www.iso.org/standard/68564.html

cstddef - https://gcc.gnu.org/onlinedocs/libstdc++/latest-doxygen/a00233_source.html

iostream - https://gcc.gnu.org/onlinedocs/libstdc++/latest-doxygen/a00086_source.html

string - https://gcc.gnu.org/onlinedocs/libstdc++/latest-doxygen/a00152_source.html

The boost library preprocessed into the ToE was version 1.65.1.

The preprocessing was done using *Make* with a makefile similar to the following, and using the preprocessed .ii file retained due to the *-save-temps* option:

```
#-----------------------------------------------------------------------
COMPILER         = -c++
OPTIMIZATION_OPT = -O3
#OPTIONS          = -pedantic -ansi -Wall -Werror $(OPTIMIZATION_OPT) -o
OPTIONS            = -c -save-temps -o
PTHREAD          =
#-lpthread
LINKER_OPT       =
#-L/usr/lib -lstdc++ $(PTHREAD) -lboost_thread -lboost_system

BUILD_LIST+=tcpproxy_server

all: $(BUILD_LIST)

tcpproxy_server: tcpproxy_server.cpp
    $(COMPILER) $(OPTIONS) tcpproxy_server tcpproxy_server.cpp $(LINKER_OPT)

strip_bin :
    strip -s tcpproxy

clean:
    rm -f core *.o *.bak *~ *stackdump *#
#-----------------------------------------------------------------------

$ > make clean ; make
```

4.1.2 Scope

In case of this example, a large amount of compile-time libraries were included in the source, but we focused our efforts on the operational security of the main source (*tcpproxy_server.cpp*).

The exact definition of scope would also be influenced by the choice of an SCL level according to our Security Certification Levels scheme described in D4.2. In the case of formally verified code a higher level such as SCL 8 or above would be practical, together with analysis of the environment. Due to the limited timeframe within this project (and considering evaluation of multiple targets described in this report), we aimed at a lower level, and thus excluded evaluation of common runtime libraries, but researched known problems regarding them.

4.1.3 Applicable requirements

In D1.2, DA detailed the requirements to be applied against their use case. However, those were described as not feasible to be tested in the scope of the VESSEDIA project, and general requirements from D1.1 are prioritized.

However, the generic requirements collected for IoT in D1.1 are not applicable to a TCP proxy, which operates on the network layer, above or below the features described in Section 4.3.2 of D1.1. Keeping this discrepancy in mind, we proceed with collecting the relevant objectives and threats, and devised test cases applicable to the functionality offered by the device class under test.

4.1.4 Security objectives

In this section we collected the assets and security objectives based on D1.2 [2].

In the scope of VESSEDIA, the following security objectives were found relevant for DA's Aircraft Maintenance System, AMS.

Table 12 - Security objectives for the AMS, in the scope of VESSEDIA

Objective	Description	Applicability in evaluation
privacy or confidentiality	Only DL users are authorized to send/receive data.	N/A
data integrity	Unauthorized users cannot alter data.	Yes
functional logic integrity	Data sent over the DL cannot alter the behaviour of the drone in an unexpected way (this objective acts as a safety goal for security purpose).	Yes

We found these requirements partially relevant in the case of the tcp proxy tested as well: this layer of functionality needs to provide functional logic integrity, and should not introduce data integrity errors, but confidentiality, integrity, and privacy of the sent or received data would be checked on higher layers of the protocol stack.

4.1.5 Threat modelling

Extensive threat modelling has been carried out for DA's use case components in D1.2 [2], Section 4.2.

In evaluating only an example of the functionality, we were able to identify only a small subset of relevant threats in the scope of the current evaluation.

Table 13 - Relevant STRIDE Threats and objectives for the AMS

Threats (STRIDE)	Related objectives	Applicability in evaluation
Spoofing identity of user or server	The TOE shall enforce *Confidentiality of identity of user or server (to prevent theft of identity)* *Authenticity of identity of user or server (to prevent usage of stolen identity or Man-in-The-Middle attack)*	N/A
Tampering with data	The TOE shall enforce *Integrity of data*	*Yes*
Repudiation of the action	The TOE shall enforce *Accountability of action*	N/A
Information disclosure	The TOE shall enforce *Confidentiality of information*	N/A
Denial of service	The TOE shall enforce *Availability of service*	*Yes*
Elevation of privilege	The TOE shall enforce *Authentication and Authorization*	N/A

4.2 Evaluation Plan

According to the Evaluation Methodology described earlier in D4.2, we here define the elements for the evaluation plan as per D1.2 [2].

4.2.1 Test environment

The tests were performed in an Ubuntu Linux virtual machine running in Oracle VM VirtualBox. The example applications were compiled to the native architecture (x86_64).

4.2.2 Version tested

The version of the main target was:

https://github.com/ArashPartow/proxy/blob/29d09e6bcef563b2e03a4100346c055e9a4128f6/tcpproxy_server.cpp

Committed by ArashPartow on Jan 1, 2017 (http://www.partow.net) .

The libraries used in the preprocessing step were:

- gcc version 7.4.0 (Ubuntu 7.4.0-1ubuntu1~18.04.1)
  - GCC v7.4 Dec 6, 2018[11]
- libboost-all-dev/bionic 1.65.1.0ubuntu1 amd64
  - boost v1.65.1 September 7th, 2017 17:31 GMT[12]

The standard libraries preprocessed into the ToE were part of GCC, compliant to ISO/IEC 14882:2017[13], and they were similar to this version:

cstdlib - https://gcc.gnu.org/onlinedocs/libstdc++/latest-doxygen/a00242_source.html

cstddef - https://gcc.gnu.org/onlinedocs/libstdc++/latest-doxygen/a00233_source.html

iostream - https://gcc.gnu.org/onlinedocs/libstdc++/latest-doxygen/a00086_source.html

string - https://gcc.gnu.org/onlinedocs/libstdc++/latest-doxygen/a00152_source.html

The boost library preprocessed into the ToE was version 1.65.1.

4.2.3 Tools and test equipment

We used the following tools and software versions during the evaluation:

- Frama-C 19.1 (Chlorine)[14]
- StaDy for Frama-C Chlorine[15]
- Linux 4.15.0-45-generic x86_64 (Ubuntu 18.04.2 LTS) version[16]
- Frama-clang 0.0.7[17]

4.2.4 SCL Target

According to the methodology described in D4.2 [8] section 2.5.1, we set a ***Maximum Target Security Level***, which defines the efforts allocated for the evaluation activities.

The use case targets defined by the use case (D5.6 [17]) aimed at more formal verification of the original, confidential target. In commercially realistic setups, such a high degree of security would necessitate a high level of ***Maximum Target Security Level*** in order to provide useful results, such as SCL 9 or above, with white-box approach.

However, in the scope of VESSEDIA project, we aimed at demonstrating more techniques with more use cases, and selecting partial examples from the use cases of the partners. In the evaluation of this use case, we used target level SCL 4. This level was defined as 5 ***Expert Days of Evaluation***, within 3 weeks of ***Execution time of Evaluation***.

[11] https://gcc.gnu.org/gcc-7/
[12] https://www.boost.org/users/history/version_1_65_1.html
[13] https://www.iso.org/standard/68564.html
[14] https://github.com/Frama-C/Frama-C-snapshot/commit/4e9997291935652e7688328922bb1ffa0ae0dfba
[15] https://github.com/gpetiot/Frama-C-StaDy/tree/chlorine
[16] http://old-releases.ubuntu.com/releases/18.04.2/
[17] https://frama-c.com/frama-clang.html

While SCL Level 4 was described as black-box evaluation, we in fact received and used the full source code for the ToE, which was not identical to the ToE in the Use Case for confidentiality reasons (see D5.6 [17]). Our test plan thus includes evaluation activities, which are normally part of the evaluation against higher Target levels only. We have also used tools included in the VESSEDIA toolbox as defined in D4.2 [8] as well, in order to provide a use case study and demonstration of the use of the technologies developed within the project.

At the end of evaluation, we calculate the SCL level reached in our final Risk Analysis in section 4.5.1.

4.2.5 Test Plan

Based on the above collected information, we devised a test plan that included manual and automated testing using the tools also described.

The following tests were carried out against the ToE in this Evaluation:

- 4.3.1 Known vulnerabilities research
- 4.3.2 Evaluation with EVA plugin
- 4.3.3 Manual source code analysis

In the following chapters, we describe the test process and then the test results.

4.3 Test cases

4.3.1 Known vulnerabilities research

Since a large part of the functionality was based on the standard libraries of GCC / C++17 and boost, we also evaluated the potentially known vulnerabilities for the libraries in use.

We used the CVE vulnerability data from NVD database via the CVE Details[18] page, where search for affected versions is possible. Based on this search, we identified the following vulnerabilities related to the components.

Table 14 Vulnerabilities related to boost C++ library

#	CVE ID	CWE ID	Vulnerability Type(s)	Publish Date	Update Date	Score
1	CVE-2013-0252	20	Bypass	2013-03-12	2013-12-05	5.0
boost::locale::utf::utf_traits in the Boost.Locale library in Boost 1.48 through 1.52 does not properly detect certain invalid UTF-8 sequences, which might allow remote attackers to bypass input validation protection mechanisms via crafted trailing bytes.						
2	CVE-2008-0172	20	DoS	2008-01-17	2018-10-15	5.0
The get_repeat_type function in basic_regex_creator.hpp in the Boost regex library (aka Boost.Regex) in Boost 1.33 and 1.34 allows context-dependent attackers to cause a denial of service (NULL dereference and crash) via an invalid regular expression.						
3	CVE-2008-0171	20	DoS	2008-01-17	2018-10-15	5.0
regex/v4/perl_matcher_non_recursive.hpp in the Boost regex library (aka Boost.Regex) in Boost 1.33 and 1.34 allows context-dependent attackers to cause a denial of service (failed assertion and crash) via an invalid regular expression.						

[18] https://www.cvedetails.com

None of the above vulnerabilities were relevant to the boost version v1.65.1, neither to the functionalities being used from boost by the proxy.

We briefly checked the boost version history at https://www.boost.org/users/history/, and identified no issues applicable to the used functionality in the release notes of newer versions (up to version 1.72.0, which was the latest version before the closing of this evaluation).

Vulnerabilities related to C++ cstdlib library:

CVE id CVE-2019-16277[19] related to cstdlib were related to PicoC 2.1, and not related to our ToE.

C++ *cstddef* library:

No vulnerability found

C++ *iostream* library:

No vulnerability found

C++ *string* library

No vulnerability found

We briefly checked the GNU GCC 7 Release Series information at https://gcc.gnu.org/gcc-7/. Namely, there was one newer GCC version 7.5 issued, and known changes were listed at https://gcc.gnu.org/gcc-7/changes.html and under https://gcc.gnu.org/bugzilla/buglist.cgi?bug_status=RESOLVED&resolution=FIXED&target_milestone=7.5, where several corrected compiler issues were listed. We did not find issues that should be considered relev

Conclusion:

Our vulnerability research did not find any known or existing vulnerability related to the components used by the ToE in the scope of the current evaluation.

4.3.2 Evaluation with EVA plugin

We installed the components required by Frama-C in order to carry out fuzz testing of the C++ TCP Proxy server. The test case differed from the earlier execution in 2.3.1 since the target used modern C++ - it was preprocessed with C++17. The Frama-Clang plugin (v0.0.7, see also 4.2.3) available was prepared to handle C++11. Thus, we were not able to carry out or analysis.

We have experimented with re-running *make* described in section 4.1.1, using C++11 libraries, but due to numerous dependency issues this test could not be concluded.

4.3.3 Manual source code analysis

Due to the analysis tools failing with the preprocessed source, we did manual source code analysis on the TCP Proxy source (attached in Appendix A).

The source code was well-structured C++ code using modern constructs of C++17 and also the boost library, which abstracted away a large portion of underlying functionality, like with the boost::asio::io_service used. In general, the data transfer services implemented relied on boost::asio, using it for buffering and socket implementation. Multithreading and mutexes were also in use from boost::mutex.

After the analysis of the code body we did not find any vulnerabilities or threats.

[19] https://www.cvedetails.com/cve/CVE-2019-16277/

4.4 Findings and recommendations

The current evaluation round for the TCP Proxy had no findings identified, while one of the test cases remained inconclusive.

4.5 Risk analysis

No risk has been identified in two of the three test cases. Due to one test finished as inconclusive, we did not consider the risk analysis complete.

4.5.1 SCL result

Based on the testing issues, some of the tests remained inconclusive till the end of the evaluation. Based on other test results, it can be concluded that if no further threats are identified, the SCL level would be SCL 4 at the end of evaluation – see the calculation at 3.5.1, where the results were similar.

Chapter 5 Summary and Conclusion

The current deliverable described the work carried out to evaluate the selected Use Case results from WP5 according to the methodology described in D4.2 [8]. The targets were selected to be able to demonstrate three examples of the evaluation process:

The document contains the evaluation report of the following part of the VESSEDIA use-cases:

- MQTT client and CoAP server in the new generation Contiki-OS for Inria Use Case described in D5.2 [13]
- MPL routing protocol used by the 6LoWPAN management protocol for CEA Use Case described in D5.4 [15]
- Open source TCP Proxy implementation for DA Use Case described in D5.6 [17]

During our evaluation work, we also focused on how VESSEDIA tools can be used towards the Objectives for WP4 set forth in DoA, namely to review how VESSEDIA results can be used to improve evaluation results.

Our work showed that while the tools still have some shortcomings, and compatibility can be further developed, there are equally capable tools that an analyst can use as a basis of systematic search for issues. Issues uncovered were useful in one of the use cases to pinpoint real issues that would pull down the security of the developed component. We were able to provide specific recommendations for the issues so that the Use Case developer would be able to correct the security-related problems.

As we described in D4.2 [8], the usual next step would be the review phase, which provides time for the Developer to review the result and fix the issues. At the end of the review phase the Evaluator verifies the corrected threats and creates a Review Report, which contains the residual risk and any new threats discovered in the review phase. Due to the timing of the report, we were not able to carry out this phase yet, but we communicated the issues nevertheless, and we will verify the resulting fixes after the submission of this report.

The evaluation of the MQTT protocol with Contiki OS in Chapter 2 resulted in an exploitable vulnerability, which is also present in the latest public release of the Contiki-NG[20]. We contacted the developers at 15 January 2020. The developers acknowledged and fixed the problem at 25 February 2020 and promised to merge these fixes to the main branch as soon as possible.

[20] https://github.com/contiki-ng/contiki-ng/tree/release/v4.4

Chapter 6 List of Abbreviations

Abbreviation	Translation
ACSL	ANSI/ISO C Specification Language
CC	Common Criteria
CFI	Code Flow Integrity
CoAP	Constrained Application Protocol
CVE	Common Vulnerabilities and Exposures
DFI	Data Flow Integrity (See [2])
EAL	Evaluation Assurance Level
EVA plugin	Evolved Value Analysis plugin (see [2])
FAM	Formal Analysis Models (See [2])
gcc	GNU Compiler Collection
ICB	Internet Connected Box
IoT	Internet of Things
MDR plugin	MarkDown Report plugin (see [2])
MQTT	MQ Telemetry Transport protocol
NVD	National Vulnerability Database; U.S. government repository of standards-based vulnerability management data
RPP plugin	Automatic Proof of Relational Properties by Self-composition plugin (See [2])
SCL	Security Certification level (see [2])
ToE	Target of Evaluation
VC	Verification Condition
WP	Work Package
WP plugin	Weakest Preconditon plugin (see [2])

Table 15: List of Abbreviations

Chapter 7 Bibliography

[1] VESSEDIA DS-01-731453 / D1.1 / V1.0 report: Security requirements for connected medium security-critical applications
[2] VESSEDIA DS-01-731453 / D1.2 / V1.0 report: Requirements descriptions for the WP5 use-cases
[3] VESSEDIA DS-01-731453 / D2.1 / 1.0 report: Basic Analyzers - Intermediate Release
[4] VESSEDIA DS-01-731453 / D2.2 / 1.0 report: Collaboration of analyses intermediate release V1
[5] VESSEDIA DS-01-731453 / D2.3 / 1.0 report: Basic Analyzers - Final Release
[6] VESSEDIA DS-01-731453 / D2.4 / V1.0 report: Collaboration of analyses intermediate release V2
[7] VESSEDIA DS-01-731453 / D4.1 report: Metrics for VESSEDIA tools in quality assurance
[8] VESSEDIA DS-01-731453 / D4.2 / 3.0 report: VESSEDIA approach for security evaluation
[9] VESSEDIA DS-01-731453 / D4.3 report (unreleased): Benchmark for evaluating VESSEDIA tools
[10] VESSEDIA DS-01-731453 / D4.4 report (unreleased): VESSEDIA in Common Criteria evaluations
[11] VESSEDIA DS-01-731453 / D4.5 report (unreleased): Quality tests & limits of VESSEDIA tools regarding security vulnerabilities detection
[12] VESSEDIA DS-01-731453 / D5.1 report: Inria's use case intermediate report
[13] VESSEDIA DS-01-731453 / D5.2 report: Inria's use case final report
[14] VESSEDIA DS-01-731453 / D5.3 report: CEA's use case intermediate report
[15] VESSEDIA DS-01-731453 / D5.4 report: CEA's use case final report
[16] VESSEDIA DS-01-731453 / D5.5 report: DA's use case intermediate report
[17] VESSEDIA DS-01-731453 / D5.6 report: DA's use case final report
[18] Jeges E., Berkes B., Eberhardt G. and Kiss B. (2014). MEFORMA Security Evaluation Methodology - A Case Study. In Proceedings of the 4th International Conference on Pervasive and Embedded Computing and Communication Systems - Volume 1: MeSeCCS, (PECCS 2014) ISBN 978-989-758-000-0, pages 267-274. DOI: 10.5220/0004919902670274
[19] http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf
[20] The Constrained Application Protocol (CoAP) https://tools.ietf.org/html/rfc7252
[21] IEEE 802.15.4, IEEE Standard for Local and metropolitan area networks--Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs), DOI 10.1109/ieeestd.2011.6012487, <http://ieeexplore.ieee.org/servlet/opac?punumber=6012485>.
[22] A. Shostack, Threat Modeling: Designing for Security, Wiley, 2014.

Appendix A. C++ TCP Proxy server source

We used the following source in manual source code analysis under 4.3.3.

```
//
// tcpproxy_server.cpp
// ~~~~~~~~~~~~~~~~~~~
//
// Copyright (c) 2007 Arash Partow (http://www.partow.net)
// URL: http://www.partow.net/programming/tcpproxy/index.html
//
// Distributed under the Boost Software License, Version 1.0.
//
//
// Description
// ~~~~~~~~~~~
// The  objective of  the TCP  proxy server  is to  act  as  an
// intermediary  in order  to 'forward'  TCP based  connections
// from external clients onto a singular remote server.
//
// The communication flow in  the direction from the  client to
// the proxy to the server is called the upstream flow, and the
// communication flow in the  direction from the server  to the
// proxy  to  the  client  is  called  the  downstream flow.
// Furthermore  the up  anddown  stream  connections  are
// consolidated into a single concept known as a bridge.
//
// In the event  either the downstream  or upstream end  points
// disconnect, the proxy server will proceed to disconnect  the
// other  end  point  and  eventually  destroy  the  associated
// bridge.
//
// The following is a flow and structural diagram depicting the
// various elements  (proxy, server  and client)  and how  they
// connect and interact with each other.

//
//                          ---> upstream --->        +----------------+
//                                            +----->------>            |
//                     +-----------+          |         | Remote Server |
//             +--------->       [x]--->----+  +---<---[x]             |
//             |       | TCP Proxy |         |      +----------------+
// +-----------+      |  +--<--[x] Server <-----<------+
// |        [x]--->--+  |   +-----------+
// |  Client |        |
// |         <-----<----+
// +-----------+
//          <--- downstream <---
//
//


#include <cstdlib>
#include <cstddef>
#include <iostream>
#include <string>

#include <boost/shared_ptr.hpp>
```

```
#include <boost/enable_shared_from_this.hpp>
#include <boost/bind.hpp>
#include <boost/asio.hpp>
#include <boost/thread/mutex.hpp>


namespace tcp_proxy
{
  namespace ip = boost::asio::ip;

  class bridge : public boost::enable_shared_from_this<bridge>
  {
  public:

    typedef ip::tcp::socket socket_type;
    typedef boost::shared_ptr<bridge> ptr_type;

    bridge(boost::asio::io_service& ios)
    : downstream_socket_(ios),
      upstream_socket_  (ios)
    {}

    socket_type& downstream_socket()
    {
      // Client socket
      return downstream_socket_;
    }

    socket_type& upstream_socket()
    {
      // Remote server socket
      return upstream_socket_;
    }

    void start(const std::string& upstream_host, unsigned short upstream_port)
    {
      // Attempt connection to remote server (upstream side)
      upstream_socket_.async_connect(
          ip::tcp::endpoint(
              boost::asio::ip::address::from_string(upstream_host),
              upstream_port),
          boost::bind(&bridge::handle_upstream_connect,
               shared_from_this(),
               boost::asio::placeholders::error));
    }

    void handle_upstream_connect(const boost::system::error_code& error)
    {
      if (!error)
      {
        // Setup async read from remote server (upstream)
        upstream_socket_.async_read_some(
            boost::asio::buffer(upstream_data_,max_data_length),
            boost::bind(&bridge::handle_upstream_read,
                shared_from_this(),
                boost::asio::placeholders::error,
                boost::asio::placeholders::bytes_transferred));

        // Setup async read from client (downstream)
        downstream_socket_.async_read_some(
            boost::asio::buffer(downstream_data_,max_data_length),
            boost::bind(&bridge::handle_downstream_read,
```

```
                shared_from_this(),
                boost::asio::placeholders::error,
                boost::asio::placeholders::bytes_transferred));
      }
      else
        close();
    }

  private:

    /*
      Section A: Remote Server --> Proxy --> Client
      Process data recieved from remote sever then send to client.
    */

    // Read from remote server complete, now send data to client
    void handle_upstream_read(const boost::system::error_code& error,
                          const size_t& bytes_transferred)
    {
      if (!error)
      {
        async_write(downstream_socket_,
            boost::asio::buffer(upstream_data_,bytes_transferred),
            boost::bind(&bridge::handle_downstream_write,
                shared_from_this(),
                boost::asio::placeholders::error));
      }
      else
        close();
    }

    // Write to client complete, Async read from remote server
    void handle_downstream_write(const boost::system::error_code& error)
    {
      if (!error)
      {
        upstream_socket_.async_read_some(
            boost::asio::buffer(upstream_data_,max_data_length),
            boost::bind(&bridge::handle_upstream_read,
                shared_from_this(),
                boost::asio::placeholders::error,
                boost::asio::placeholders::bytes_transferred));
      }
      else
        close();
    }
    // *** End Of Section A ***


    /*
      Section B: Client --> Proxy --> Remove Server
      Process data recieved from client then write to remove server.
    */

    // Read from client complete, now send data to remote server
    void handle_downstream_read(const boost::system::error_code& error,
                            const size_t& bytes_transferred)
    {
      if (!error)
      {
        async_write(upstream_socket_,
            boost::asio::buffer(downstream_data_,bytes_transferred),
```

```
            boost::bind(&bridge::handle_upstream_write,
                 shared_from_this(),
                 boost::asio::placeholders::error));
      }
      else
        close();
    }

    // Write to remote server complete, Async read from client
    void handle_upstream_write(const boost::system::error_code& error)
    {
      if (!error)
      {
        downstream_socket_.async_read_some(
            boost::asio::buffer(downstream_data_,max_data_length),
            boost::bind(&bridge::handle_downstream_read,
                shared_from_this(),
                boost::asio::placeholders::error,
                boost::asio::placeholders::bytes_transferred));
      }
      else
        close();
    }
    // *** End Of Section B ***

    void close()
    {
      boost::mutex::scoped_lock lock(mutex_);

      if (downstream_socket_.is_open())
      {
        downstream_socket_.close();
      }

      if (upstream_socket_.is_open())
      {
        upstream_socket_.close();
      }
    }

    socket_type downstream_socket_;
    socket_type upstream_socket_;

    enum { max_data_length = 8192 }; //8KB
    unsigned char downstream_data_[max_data_length];
    unsigned char upstream_data_  [max_data_length];

    boost::mutex mutex_;

  public:

    class acceptor
    {
    public:

      acceptor(boost::asio::io_service& io_service,
               const std::string& local_host, unsigned short local_port,
               const std::string& upstream_host, unsigned short upstream_port)
      : io_service_(io_service),
        localhost_address(boost::asio::ip::address_v4::from_string(local_host)),
        acceptor_(io_service_,ip::tcp::endpoint(localhost_address,local_port)),
        upstream_port_(upstream_port),
```

```
        upstream_host_(upstream_host)
      {}

      bool accept_connections()
      {
        try
        {
          session_ = boost::shared_ptr<bridge>(new bridge(io_service_));

          acceptor_.async_accept(session_->downstream_socket(),
              boost::bind(&acceptor::handle_accept,
                  this,
                  boost::asio::placeholders::error));
        }
        catch(std::exception& e)
        {
          std::cerr << "acceptor exception: " << e.what() << std::endl;
          return false;
        }

        return true;
      }

    private:

      void handle_accept(const boost::system::error_code& error)
      {
        if (!error)
        {
          session_->start(upstream_host_,upstream_port_);

          if (!accept_connections())
          {
            std::cerr << "Failure during call to accept." << std::endl;
          }
        }
        else
        {
          std::cerr << "Error: " << error.message() << std::endl;
        }
      }

      boost::asio::io_service& io_service_;
      ip::address_v4 localhost_address;
      ip::tcp::acceptor acceptor_;
      ptr_type session_;
      unsigned short upstream_port_;
      std::string upstream_host_;
    };

  };
}

int main(int argc, char* argv[])
{
  if (argc != 5)
  {
    std::cerr << "usage: tcpproxy_server <local host ip> <local port> <forward
  host ip> <forward port>" << std::endl;
    return 1;
  }
```

```
   const unsigned short local_port = static_cast<unsigned
   short>(::atoi(argv[2]));
   const unsigned short forward_port = static_cast<unsigned
   short>(::atoi(argv[4]));
   const std::string local_host    = argv[1];
   const std::string forward_host   = argv[3];

   boost::asio::io_service ios;

   try
   {
    tcp_proxy::bridge::acceptor acceptor(ios,
                                local_host, local_port,
                                forward_host, forward_port);

    acceptor.accept_connections();

    ios.run();
   }
   catch(std::exception& e)
   {
    std::cerr << "Error: " << e.what() << std::endl;
    return 1;
   }

   return 0;
}

/*
 * [Note] On posix systems the tcp proxy server build command is as follows:
 * c++ -pedantic -ansi -Wall -Werror -O3 -o tcpproxy_server tcpproxy_server.cpp
  -L/usr/lib -lstdc++ -lpthread -lboost_thread -lboost_system
 */
```