hochschule mannheim

Conception, Implementation and Postmortem Documentation of a Modular Proxy Application for Testing Internet of Things Applications

Moritz Laurin Thomas

Master Thesis

for the acquisition of the academic degree Master of Science (M.Sc.)

Course of Studies: Computer Science

Department of Computer Science

University of Applied Sciences Mannheim

31.05.2021

Tutors

Prof. Dr. Thomas Specht, Hochschule Mannheim

Pierre-Alain Mouy, M.Sc., NVISO GmbH

Thomas, Moritz Laurin:
Conception, Implementation and Postmortem Documentation of a Modular Proxy Application for Testing Internet of Things Applications / Moritz Laurin Thomas. –
Master Thesis, Mannheim: University of Applied Sciences Mannheim, 2020. 27 pages.

Thomas, Moritz Laurin:
Konzeption, Implementierung und Post-mortem-Analyse eines modularen Proxys zum Testen von Anwendungen im Internet der Dinge / Moritz Laurin Thomas. –
Master-Thesis, Mannheim: Hochschule Mannheim, 2020. 27 Seiten.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich bin damit einverstanden, dass meine Arbeit veröffentlicht wird, d. h. dass die Arbeit elektronisch gespeichert, in andere Formate konvertiert, auf den Servern der Hochschule Mannheim öffentlich zugänglich gemacht und über das Internet verbreitet werden darf.

Mannheim, 31.05.2021

Moritz Laurin Thomas

Abstract

Conception, Implementation and Postmortem Documentation of a Modular Proxy Application for Testing Internet of Things Applications

TBD

Konzeption, Implementierung und Post-mortem-Analyse eines modularen Proxys zum Testen von Anwendungen im Internet der Dinge

TBD

Acknowledgements

TBD

Contents

1. Introduction **1**

1.1. Motivation . . . 1

1.2. Purpose and Structure of the Thesis . . . 2

2. Related Work **3**

2.1. Computer Network Analysis in General . . . 3

2.2. Homogenization of the IoT Landscape . . . 3

2.3. IoT Security Analysis . . . 3

3. Theoretical Background **5**

3.1. Computer Networks . . . 5

3.1.1. Network Layers . . . 5

3.1.2. Proxying Network Traffic . . . 5

3.1.3. Deep Packet Inspection . . . 6

3.2. (Industrial) Internet of Things . . . 6

3.2.1. Fields of Use . . . 6

3.2.2. Application Architectures . . . 6

3.2.3. Common Protocols . . . 6

3.2.4. Security Considerations . . . 7

3.3. Information Security . . . 7

3.3.1. Key Concepts . . . 7

3.3.2. Legal Background . . . 7

3.3.3. Integration in Software Development . . . 7

3.3.4. Methodology . . . 7

4. Understanding the Problem Space **9**

4.1. Prototypical Implementation . . . 9

4.1.1. Example Scenarios . . . 10

4.1.2. Requirements . . . 12

4.1.3. Design . . . 13

4.1.4. Implementation . . . 14

4.1.5. Insights Gained . . . 15

4.2. Interviewing Experts for Insights . . . 15

4.2.1. Interview Guideline . . . 16

4.2.2. Conducting Interviews . . . 16
4.3. Analysis of Existing Software . . . 17

5. Conceptual Design **19**
5.1. Requirements: Design Implications . . . 19
5.2. User Interactions: Designing the Intended Workflow . . . 19
5.3. Inferring Software Components . . . 20
5.4. Summary: An Abstract Design Concept . . . 20

6. Implementing the Modular Proxy Application **21**
6.1. Goals and Constraints . . . 21
6.2. Tool Selection . . . 21
6.2.1. Requirements to The Tools . . . 21
6.2.2. Comparison of Programming Languages, Frameworks and Libraries . . . 22
6.3. Individual Components . . . 22
6.3.1. Network Stack . . . 22
6.3.2. Finite State Machine . . . 22
6.3.3. Configuration Parsing and Building . . . 22

7. Postmortem Documentation **23**
7.1. Quantitative: Project Timeline . . . 23
7.2. Qualitative: Deliverables . . . 24

8. Summary **25**
8.1. Requirements Engineering . . . 25
8.2. Concept . . . 25
8.3. Implementation . . . 25

9. Conclusion **27**
9.1. Outlook . . . 27

List of Abbreviations **ix**

List of Tables **xi**

List of Figures **xiii**

Listings **xv**

Bibliography **xvii**

Index **xix**

A. Diagrams **xix**

Chapter 1

Introduction

This chapter will introduce the underlying motivation of this thesis. Then, it will give an overview of this thesis' purpose and structure. Lastly, this chapter will show the process that the work on this thesis went through, explaining the scientific methods applied and software engineering practices used.

1.1. Motivation

Today scientific and industrial parties work on connecting physical entities such as machines, buildings and even humans to the internet by equipping them with digital sensors and actuators, referred to as "Internet of Things (IoT)". While this progression promises many positive effects, such as simplifying tasks in our personal day-to-day life ("Smart Home" applications), monitoring our personal health ("eHealth") and increasing efficiency and safety of industrial plants ("Industrial Internet of Things (IIoT)", also referred to as "Industry 4.0"), it also yields the risk of introducing new attack-vectors to parts of our environment: "smart" devices used at home or at other sensitive places may implement weak security implementations or faulty security design, resulting in private and personal data being available to parties interested in violating the privacy of one's home (e.g. vaccum robots leaking information about the interior design of homes[10]) or conducting industrial espionage which is an acute threat [2, p. 14].
The diversity of both deployed smart devices and the internet services those devices are connected to lead to the need and use of ever-increasing complex technologies used for communication, data storage and access management, further adding to po-

tential attack-vectors of connected devices and distributed applications [5, p. 119]. This complexity and the sheer number of connected devices is actively being exploited by attackers today and the number of attacks on IoT devices is increasing [4].
There are security guidelines, best practices and innovative approaches for developing secure smart applications [5, p. 120][7, p.326-328], however testing such applications proves to be cumbersome: intercepting, dissecting, inspecting and manipulating the communication in these applications requires working on various abstraction layers. In order to evaluate the security of such applications, penetration testers often spend a considerable amount of time dissecting applications and setting up a test-environment.
The goal of this thesis is to conceptualize, implement and evaluate a modular proxy application that allows to evaluate the security of IoT applications by...

1.2. Purpose and Structure of the Thesis

This thesis is separated into eight chapters: chapter (2) will give an overview of and discuss related and previous work. After that, relevant fundamentals about computer networks, IoT applications and information security will be covered in chapter 3.
The chapters 4 to 7 describe the research and development process of the IoT proxy application in chronological order: the problem space of the application is shown and dissected in chapter 4, yielding essential insights into potential challenges and technical requirements. Building upon these, the conceptual design of the IoT proxy application is proposed in chapter 5. This included the process of collecting, documenting and analysing of software requirements, describing the application's work context and designing a software architecture that complies with the aforementioned requirements. Subsequently, chapter 6 involves a prototypical implementation of the aforementioned software concept, focusing on the goals and constraints of the implementation, the tools and frameworks used and the implementation of core components of the application. The resulting implementation and the project itself are then analysed in a postmortem documentation, pointing out the reasons why and how the project ultimately failed.
The thesis ends with a summary of all results produced and conclusions drawn from the work on this thesis.

Chapter 2

Related Work

This chapter will discuss related and previous work on topics in this thesis' context. This includes network analysis in general (and IoT in particular), homogenization and unification of various IoT related technologies and performance of security evaluations of these technologies.

2.1. Computer Network Analysis in General

TBD: Polymorph [8]

2.2. Homogenization of the IoT Landscape

TBD: IoT proxy for homogenization [6]

2.3. IoT Security Analysis

As part of their master's thesis, Bellemans conducted a study in 2020 that evaluated the security and privacy implementations of fifteen "*smart*" devices from a wide price range available on the market at the time. They performed automated analyses and requested data access from manufacturers [3]. The thesis showed that the devices made use of a variety of both standardized and proprietary transport and application protocols. It also found severe flaws in the devices' compliance to General

Data Protection Regulation (GDPR): about a third of the devices' manufacturers did not reply to GDPR requests at all, however Bellemans noted that the COVID-19 pandemic may have had an impact on their data access requests. The thesis suggests that the introduction of a quality label that guarantees appropriate implementation of security and privacy aspects could prove beneficial for customers.

In 2017, Apthorpe et al. presented a three stage strategy to examine metadata of network traffic of four smart devices [1]. By monitoring the devices' traffic, they showed that even though the communication between the devices and their corresponding internet servers were encrypted, passive observers could deduce information about users' behaviour by identification of the destination server and analysis of the rate of traffic being sent. A noteworthy aspect of their work is that they performed this analysis from an Internet Service Provider (ISP)'s point of view, exclusively examining metadata of the communication that took place. The strategy described in the paper consists of the following (greatly simplified) steps:

1. Identifying communication streams of individual devices (e.g. by examining the TCP packets' destination IPs).
2. Associating the streams with specific device models (e.g. by performing reverse-look ups of the aforementioned IPs).
3. Analysing traffic rates (presuming that traffic is generated upon taking measures).

TBD: Add simple process diagram

Apthorpe et al. conclude that their strategy works well on inferring behaviour from regular internet traffic of smart devices, however they assume that shaping traffic or making use of proxies (that effectively mask the destination IPs) could be effective counter-measures. It is safe to assume that regular smart home setups do not make use of proxies or traffic shaping though, thus being vulnerable to this kind of attack.

TBD: NVISO Labs: Théo Rigas, IOXY [9]

Chapter 3

Theoretical Background

This chapter provides an overview of the technologies and concepts referred to in subsequent chapters. Starting with section 3.1, essential concepts of computer communication in networks will be presented and examined, covering the concept of network layers, intercepting of communication between two parties and analysis of transferred data. Building upon these fundamentals, section 3.2 introduces the fields of use of IoT applications, common architectures used today to implement them and popular protocols they make use of. Lastly, it will discuss security considerations important to IoT applications. After that, section 3.3 will provide insights into relevant concepts and the practices used and applied in information security. It covers key concepts and legal considerations, integration of information security in software development and common practices and methods involved.

3.1. Computer Networks

3.1.1. Network Layers

TBD Transmission Control Protocol (TCP)

3.1.2. Proxying Network Traffic

TBD; planned:

1. Definition; Working Principle
2. Use Cases

3. Abuse Cases

3.1.3. Deep Packet Inspection

TBD

3.2. (Industrial) Internet of Things

3.2.1. Fields of Use

3.2.2. Application Architectures

3.2.3. Common Protocols

Building up on pre-existing network infrastructure and in order to meet requirements specific to individual fields of use and use-case scenarios, the landscape of IoT attends with a great variety of *communication protocols* (further used to refer to both transport and application protocols). This section will provide a brief overview of the working principles, use cases and history of some protocols commonly used in IoT and IIoT applications today.

Hypertext Transfer Protocol (HTTP) *TBD*

WebSockets (WS) *TBD*

Message Queuing Telemetry Transport (MQTT) *TBD* Amazon Web Services (AWS) IoT

Modbus TCP *TBD*

Profibus/Profinet *TBD*

OPC Unified Architecture (OPC U/A) *TBD*

3.2.4. Security Considerations

3.3. Information Security

TBD

3.3.1. Key Concepts

3.3.2. Legal Background

Compliance

Data Protection

3.3.3. Integration in Software Development

Traditional Approaches

Modern Approaches

3.3.4. Methodology

Risk Management

Incident Response

Reverse Engineering

(Physical) Penetration Testing

Source Code Audits

Application Configuration

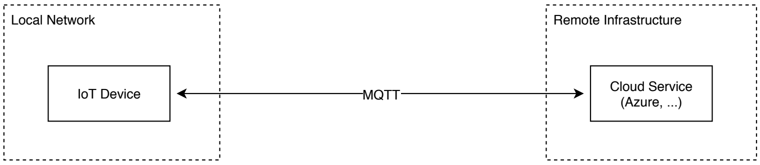
Chapter 4

Understanding the Problem Space

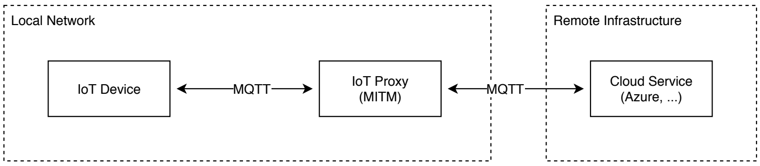
In order to provide a satisfying solution to the problem at hand, the problem itself and the environment it occurs in must be researched. This chapter aims to explore and examine the problem space, resulting in a set of artefacts (namely a domain model and a set of requirements) that aid in understanding the context and designing an appropriate solution. First, a prototypical network proxy is designed and implemented in section 4.1 to get an understanding of the problems and challenges involved in designing, implementing and using such software. Based on these experiences, interviews with experts in penetration testing are conducted and evaluated in section 4.2 to get a proper understanding of their everyday work and resulting problems. Lastly, existing software that aims to intercept communication for various scenarios and technologies is examined in section 4.3, compared to each other and their usefulness for the problem-specific scenarios is assessed.

4.1. Prototypical Implementation

The prototype was designed to be used in two realistic scenarios; one in an Industrial Control System (ICS) context and a more complex one in an IoT cloud context. The goal of this section was to implement a prototype that could be used as a proxy to intercept communication between an IoT device and its cloud service as shown in figure 4.1. It was developed incrementally so individual components could be derived from requirements, designed, implemented and evaluated in fixed sprints.



(a) Regular communication between an IoT device and a cloud service.



(b) Communication intercepted by a man-in-the-middle (MITM) proxy.

Figure 4.1.: Installing a MITM proxy to intercept network communication for penetration testing.

4.1.1. Example Scenarios

The following scenarios describe realistic configurations of IoT/IIoT devices that should be tested with the prototype:

Scenario #1: ICS Modbus TCP In this IIoT scenario, a human-machine interface (HMI) (*Siemens KTP400 Basic*) sends commands to and receives data from a programmable logic controller (PLC) (*Siemens S7-1200*) using Modbus TCP. The PLC continually counts up a value up to 100 and begins anew at zero while the HMI displays the current value and provides a button that, upon being pressed by a user, resets the current value to zero.
In this scenario, attackers could perform a variety of attacks on the system by intercepting and manipulating network traffic, for example:

- By dropping messages sent from the PLC to the HMI, the application may appear unresponsive as new data is not displayed on the HMI. In production environments, this could lead to dangerous situations as sensor readings that indicate harmful environmental conditions would not be presented to supervising personnel.
- When dropping messages sent from the HMI to the PLC, control commands can be suppressed. This attack can result in catastrophic situations when emergency shutdowns issued by supervising personnel are not registered by the affected machines.

Due to the rather simple nature of the Modbus TCP protocol, intercepting and manipulating communication is expected to be trivial.

Scenario #2: AWS IoT This IoT smart home scenario utilizes two local IoT devices that are integrated into a cloud environment such as the AWS IoT platform: a thermometer and an air conditioner (A/C) unit. Both devices connect to the cloud platform, authorize themselves at a Representational State Transfer (REST) interface via HTTP and upgrade their HTTP-connection to WS streams upon successful authorization. They eventually communicate to a remote MQTT broker by tunnelling MQTT packets over the WS stream. At this stage, the thermometer publishes temperature readings to an MQTT topic while the A/C unit subscribes to the same topic and adjusts its operation depending on the incoming temperature readings.
This distributed communication setup introduces a set of possible attacks that could be performed when attackers *impersonated* client-devices or the remote server:

- Impersonating the thermometer, attackers could send incorrect temperature data and effectively control the A/C unit. When sending low temperature readings while the environment temperature is high, the A/C unit would stop running. Conversely, when high temperature readings are sent while the environment temperature is low, the A/C unit would run, and thus further cool down the environment.
- Attackers that impersonate the remote server could drop or manipulate incoming publish packets, thus altering whether and/or what information is relayed other connected devices. For example, temperature readings that indicate a high environment temperature that would lead to the A/C unit to be powered up could be rewritten in such a way that the transmitted temperature value is considered to indicate a low environment temperature, thus preventing the A/C unit from running automatically.

This scenario makes use of three communication protocols, uses these protocols dependent on the state of authentication and even tunnels one protocol through another one. Therefore the proxy application has to implement a state-machine (as seen in 4.2) and testing communication in this scenario is expected to be more complex than the first one.

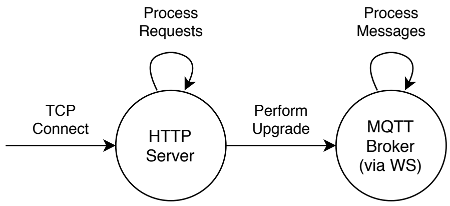


Figure 4.2.: State machine of AWS IoT communication

4.1.2. Requirements

To be able to operate in both of the aforementioned scenarios, the prototype had to implement a set of functional requirements:

- **F1 Protocols:** The software must implement parsing/crafting messages/packets of the following communication protocols: HTTP, WS, MQTT and Modbus TCP.
  Fit criterion: TBD
- **F2 Network Stacks:** The software must be able to parse protocols that are tunnelled through other protocols ("*stacked*"). It must provide an interface to the user where they can specify which communication protocols are used and whether and how they are stacked (further referred to as *network stack*).
  Fit criterion: The software processes a configuration file that lets users specify which protocols to be used and whether/how they are stacked.
- **F3 State-Machines:** The software must be able to switch network stacks dependent on configurable *states*. It must provide an interface for the user to specify when to switch to using another network stack, represented using finite-state machines (FSMs) and rule sets for transmission between states.
  Fit criterion: The software processes a configuration file that lets users specify when to switch between network stacks.
- **F4 Integration:** The software shall provide interfaces for integration of third-party software.
  Fit criterion: The software allows sending requests/responses to "Burp Suite" for manipulation.
- **F5 Scripting:** The software shall provide scripting capabilities for automated manipulation of messages/packets.
  Fit criterion: Users can define script-snippets to be executed on messages/-packets.

The following non-functional requirements were defined:

N1 **Extensibility:** To allow for future implementation of further communication protocols the software shall be implemented in a modular fashion.

N2 **Platform Compatibility:** In order to support a broad spectrum of target platforms, the software shall be implemented platform-independently.

N3 **Reusability:** The software shall be reusable so it can be used in future tests that may feature new configurations of network stacks.

N4 **Open Source:** The software shall be available as open source software so programmers and members of the IT community may contribute to improving it.

Due to this implementation serving as a prototype and being of an academic nature, no specific constraints were defined. It was to be developed strictly ignoring aspects of usability and stability as it should not be used in production environments but in laboratories exclusively.

4.1.3. Design

The prototype was designed to be fit for use in the second scenario as, regarding network communication, it was more complex than the first one. Specifically, the second scenario demanded the implementation of a network stack and a state machine to switch between states. Parsing protocols that were tunnelled through other protocols appeared to be a potentially challenging requirement. In order to tackle it, a variation of the *"pipeline"* (sometimes referred to *"pipes and filters"*) design pattern was used (as shown in 4.3). It was designed to be used as follows:

"Messages" originate from a listener, for example messages with raw byte payloads are received from a TCP socket. These messages are sent to an initial "pipe" to be processed *down*. Pipes are bi-directional routers that perform the following actions on messages:

1. Use optional "encoders" to disassemble/de-serialize messages when processing them *down* the pipeline and re-assemble/serialize them when they process messages *up* the pipeline.
2. Use optional "filters" to perform operations on messages such as replacing header values or manipulating payloads.

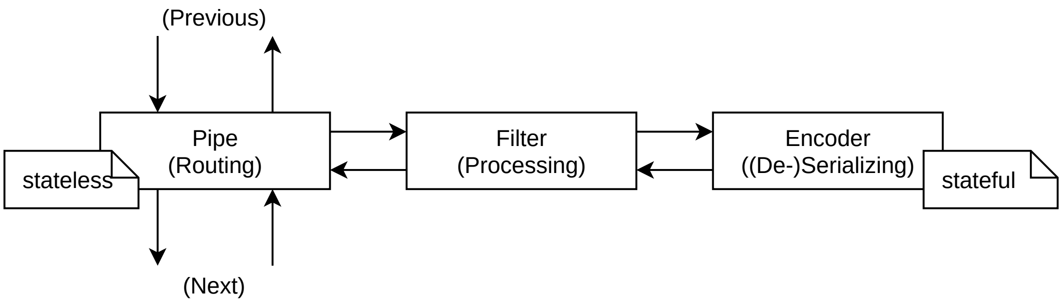


Figure 4.3.: The variation of the *"pipes and filters"*/*"pipeline"* design pattern used in the prototype.

3. Forward messages to the next pipe in its pipeline when processing messages down or to the previous pipe when processing messages back up.

There are extensions to basic pipes such as:

- "EndPipes" are appended to the end of a pipeline and reverse the message processing direction so messages that were processed down are sent back up the pipeline to be processed up.
- "ProcessingPipes" mandate encoders and filters to be used. These pipes are used to indicate that messages are not not only routed but also processed and encoded or decoded.
- "IntegrationPipes" allow integration of other software into the pipeline. For example, penetration testing software such as Burp Suite could be integrated.

TBD:

- *Designed during sprints so only pipes are designed, state-machine only rough concept (States, Transitions, Rules)!*
- *Show diagrams of messages?*
- *Explain Figure A.1 and A.2*

4.1.4. Implementation

TBD:

- *technology: typescript*
- *sprints: two sprints, started with communication (http + ws + mqtt)*

- *failed: high-level API, callback-hell (debugging/tracing), missing typescript typings, very tight coupling*

4.1.5. Insights Gained

The following insights were gained through the prototypical implementation. Some resulted in questions relevant for the expert interviews that were to be held:

- Due to the maximum transmission unit (MTU), large messages are broken into chunks that are transferred sequentially. This requires the proxy to work on streams of incoming data and reassemble messages from said chunks.
- Support of multiple clients is non-trivial as communication between clients and servers is not necessarily connection-oriented (e.g. HTTP).
  Q: Do penetration testers need to test multiple devices at the same time?
- Increasing the size of the payload of a messages can result in the payload being split upon multiple messages (e.g. WS).
  Q: Do penetration testers require exact control over the implementation of protocols?
- Manipulating messages, on the fly via scripting or by hand using third-party integrations (e.g. to *Burp Suite*), can introduce latency to the communication.
  Q: Are there strict timing requirements during penetration tests?
- Many libraries offer high-level functions to the programmer while avoiding exposure of low-level functionalities like crafting or parsing messages.

4.2. Interviewing Experts for Insights

Interviews may be an efficient way to get an expert's opinion on something they are proficient in. Thus, expert interviews were conducted to let security researchers give insight into their everyday work and the challenges they face when working with IoT and IIoT applications. The information and insights gathered in these interviews were then used to model a persona, various work scenarios and use-cases that as a whole aim to represent their work.

4.2.1. Interview Guideline

An interview guideline (shown in *TBD*) was created to keep focus on key points during interviews so that interviewees would not stray too far from the relevant points. The guideline also served as a checklist so the interviewer could make sure that all questions and points that should be covered initially, were in fact covered by the end of the interviews. It was composed of three sections:

1. Experiences with IoT The answers to these questions would give insights into what kind of applications the security researchers had worked on in the past. Answers to question *1.1.* were of particular interest as they might represent what technologies were being examined by security researchers and may be popular in today's applications.

2. Processes in Everyday Life This section aimed to cover questions about the processes and tasks security researchers perform during penetration tests of IoT applications in their everyday life. Ideally, answers to those questions would show the approaches taken and challenges faced during their work, uncovering potential needs and underlying motivation.

3. The Future of IoT This section had security researchers assess what the future of IoT may be like from their point of view. This required the interviewees to make a critical assessment of the status quo.

4.2.2. Conducting Interviews

Interviews were conducted with six *NVISO* employees that all had worked on security assignments on IoT or IIoT applications in the past. There is considerable variety in

- the experience they had in working on security assignments in general: all interviewees had a strong background in cyber security that reached back multiple years except one who was a working student at *NVISO Labs*.
- and the experience they had in working on IoT/IIoT applications: two interviewees worked on assessing IoT/IIoT applications only occasionally, one

was part of a car manufacturer's automotive security team in the past and three were part of *NVISO Labs* and worked with smart devices on a regular basis.

The duration of the interviews varied from 45 minutes to two hours depending on the amount and level of detail of information provided by the interviewees and the number of times that the interviewer had to ask further questions.

TBD:

- *Summary of the interviews*
- *conclusions drawn*
- *personas and user stories -> new requirements!*

4.3. Analysis of Existing Software

Wireshark 3,690,000 lines of code[1] *TBD*

MITMf *TBD*

Ettercap *TBD*

bettercap *TBD*

mitmproxy *TBD*

mProxy *TBD*

IOXY *TBD*

TBD; planned: paragraph about each program including a general description, uses, capabilities and usefulness

[1]This number was returned by the *cloc* utility run on commit *3a8111e1c2adcdc0603993c6ed5d20a40f162125* from Aug. 4th 2020 of Wireshark's Github mirror.

Name	*Latest Release*	*Implemented in*	*Supported Protocols*	*R*	*W*	*D*
Wireshark	2020-07-01	C	Various	F	N	N
MITMf	2015-08-28	Python	Various	?	F	F
Ettercap	2019-07-01	C	Various	F	F	F
bettercap	2020-03-13	Go	Various	F	F	F
mitmproxy	2020-03-13	Python	HTTP/S, WS	P	P	P
mProxy	Pre-Releases only	Go	MQTT	?	F	-
IOXY	Source only	Go	MQTT	F	F	F

Table 4.1.: Comparison of existing software where R, W and D describe read, write and deletion capabilities, respectively. F, N and P indicate full, no or partial functionality, respectively.

Chapter 5

Conceptual Design

This chapter will detail the process of conceptualizing the design of the modular proxy application based on the results of the preceding chapter. First, the requirements are analysed for their potential design implications in section 5.1. Afterwards the user interactions and domain entities identified in chapter 4 are examined and broken down into communication flows between actors and systems in section 5.2 and individual software components that complete the design are discussed in section 5.3. Lastly, an overview of the complete design concept is given in section 5.4, discussing potential advantages and constraints.

Note: sections 5.1 and 5.2 should probably be merged as they overlap a lot

5.1. Requirements: Design Implications

TBD

- *Stream-based: treat communication as streams. message-based systems are simpler and supported by design*
- *Server-client: proxy is server, client can interface to control + monitor, communication via REST + WS*

5.2. User Interactions: Designing the Intended Workflow

TBD

- *Passive: Logging*
- *Passive: Scripting*
- *Passive: Fuzzing*
- *(Inter-)Active: REST+WS or Burp Suite integration*

5.3. Inferring Software Components

TBD

- *State-machine: active network stack/pipeline dependent on state of the connection*
- *NetStacks: series of pipelines, bound to states*
- *Pipes: basic pipes, loose routing, injectable, specialized, generic processors + specialized encoders*
- *Factory: parse state-machine and netstack configuration and instantiate + configure instances*

5.4. Summary: An Abstract Design Concept

TBD (maybe obsolete as this is covered in preceding sections)

- *Component view?*

Chapter 6

Implementing the Modular Proxy Application

This chapter covers an exemplaric implementation of the concept that was worked out in chapter 5, starting with formally describing the goals and constraints of this implementation in section 6.1. Afterwards, an overview and comparison of available and suitable tools for the task is performed in section 7.2. The chapter concludes with details about the implementation of individual components in section 6.3, describing how specific challenges were overcome and what design patterns were used.

6.1. Goals and Constraints

TBD

- *Focus on complex scenario #2: HTTP, WS, MQTT*
- *No interactive mode*
- *Fully implement factories, state-machines and netstacks*

6.2. Tool Selection

6.2.1. Requirements to The Tools

TBD

- *Needs to run cross-platform*
- *Low-level access to APIs*
- *Rich set of low-level libraries for protocol implementations*
- *Accessible and easily extendable*

6.2.2. Comparison of Programming Languages, Frameworks and Libraries

TBD: Discuss how the candidates match the above-mentioned criteria and point out why python was chosen

- *Native C: Win32 API / Linux ABI*
- *.NET Visual C# & NuGet*
- *JavaScript/TypeScript & npm*
- *Python & pip*

6.3. Individual Components

6.3.1. Network Stack

Gateways

Pipes, Encoders and Processors

Scripting

Pipelines

6.3.2. Finite State Machine

States

Transitions

Nested FSMs

6.3.3. Configuration Parsing and Building

Factories, Builders and Templates

Chapter 7

Postmortem Documentation

This chapter attempts to identify and spell out the causes of the project failure. The project timeline will allow a quantitative overview of the project progression and show what parts of the project slowed down progress. Then, an overview of the qualitative aspects of the deliverables will discuss the maturity of the implementation and which parts reached a satisfactory level.

7.1. Quantitative: Project Timeline

TBD: Overview of the project timeline: here we will see that implementation went out of control and took too much time. Feature reduction came too late and too few. Also, discuss additional problems that took up much time:

- *Race-conditions: newly arrived messages could change states in state-machines, decoupling network stacks and causing currently processed messages that were processed up to raise null pointer exceptions.*
- *Poor documentation: some open-source libraries (e.g. python's WS library) were poorly documented, resulting in days worth of diving through source code.*
- *Time-consuming debugging: testing scenario # 2 meant loading a large configuration file, resulting in dynamic and long pipelines and interweaving of multiple state-machines and pipelines. Tracing messages became very time-consuming and confusing.*

7.2. Qualitative: Deliverables

TBD: Which fit-criteria were met? What is the implementation currently capable of? Which requirements were not full-filled?

Chapter 8

Summary

This chapter provides a summary of the concept shown in chapter 5 and the implementation thereof in chapter 6.

8.1. Requirements Engineering

8.2. Concept

8.3. Implementation

TBD: Discuss technical debt!

Chapter 9

Conclusion

TBD

9.1. Outlook

TBD: Discuss specific steps that can be taken to fully implement the concept.

List of Abbreviations

A/C air conditioner
AWS Amazon Web Services
FSM finite-state machine
GDPR General Data Protection Regulation
HMI human-machine interface
HTTP Hypertext Transfer Protocol
ICS Industrial Control System
IIoT Industrial Internet of Things
IoT Internet of Things
ISP Internet Service Provider
MITM man-in-the-middle
MQTT Message Queuing Telemetry Transport
MTU maximum transmission unit
PLC programmable logic controller
REST Representational State Transfer
TCP Transmission Control Protocol
OPC U/A OPC Unified Architecture
WS WebSockets

List of Tables

4.1. Comparison of existing software 18

List of Figures

4.1. Installing a MITM proxy to intercept network communication for penetration testing. 10
4.2. State machine of AWS IoT communication 12
4.3. The variation of the *"pipes and filters"*/*"pipeline"* design pattern used in the prototype. 14

A.1. AWS IoT Scenario - State 1: HTTP Server xx
A.2. AWS IoT Scenario - State 2: MQTT via WS xxi

Listings

Bibliography

[1] Noah Apthorpe, Dillon Reisman, and Nick Feamster. *A Smart Home is No Castle: Privacy Vulnerabilities of Encrypted IoT Traffic*. 2017. arXiv: 1705.06805 [cs.CR].

[2] Michael Bartsch et al. *Spionage, Sabotage und Datendiebstahl - Wirtschaftsschutz in der Industrie 2018*. Oct. 2018. URL: https://www.bitkom.org/sites/default/files/file/import/181008-Bitkom-Studie-Wirtschaftsschutz-2018-NEU.pdf.

[3] Jonah Bellemans. "The state of the market: A comparative study of IoT device security implementations". MA thesis. KU Leuven, 2020.

[4] Dan Demeter, Marco Preuss, and Yaroslav Shmelev. *IoT: a malware story*. Oct. 2019. URL: https://securelist.com/iot-a-malware-story/94451/ (visited on 02/17/2020).

[5] Jens Jäger et al. "Advanced Complexity Management Strategic Recommendations of Handling the "Industrie 4.0" Complexity for Small and Medium Enterprises". In: *Procedia CIRP*. Factories of the Future in the digital environment - Proceedings of the 49th CIRP Conference on Manufacturing Systems 57 (Jan. 2016), pp. 116–121. DOI: 10.1016/j.procir.2016.11.021.

[6] Wenquan Jin and DoHyeun Kim. "Development of Virtual Resource Based IoT Proxy for Bridging Heterogeneous Web Services in IoT Networks". In: *Sensors* 18 (May 2018), p. 1721. DOI: 10.3390/s18061721.

[7] Christian Lesjak et al. "Security in industrial IoT – quo vadis?" In: *e & i Elektrotechnik und Informationstechnik* 133.7 (Nov. 2016), pp. 324–329. DOI: 10.1007/s00502-016-0428-4. URL: https://doi.org/10.1007/s00502-016-0428-4.

[8] Santiago Hernández Ramos. *Polymorph: A Real-Time Network Packet Manipulation Framework*. Apr. 2018. URL: https://github.com/shramos/polymorph (visited on 02/17/2020).

[9] Théo Rigas. *IOXY - MQTT intercepting proxy*. July 2020. URL: https://blog.nviso.eu/2020/07/06 /introducing-ioxy-an-open-source-mqtt-intercepting-proxy/ (visited on 07/10/2020).

[10] Tilman Wittenhorst. *Ferngesteuert ins Smart Home: Saugroboter verrät Grundriss der Wohnung*. June 2019. URL: https://www.heise.de/security/meldung/Ferngesteuert-ins-Smart-Home-Saugroboter-verraet-Grundriss-der-Wohnung-4436657.html (visited on 02/17/2020).

Appendix A

Diagrams

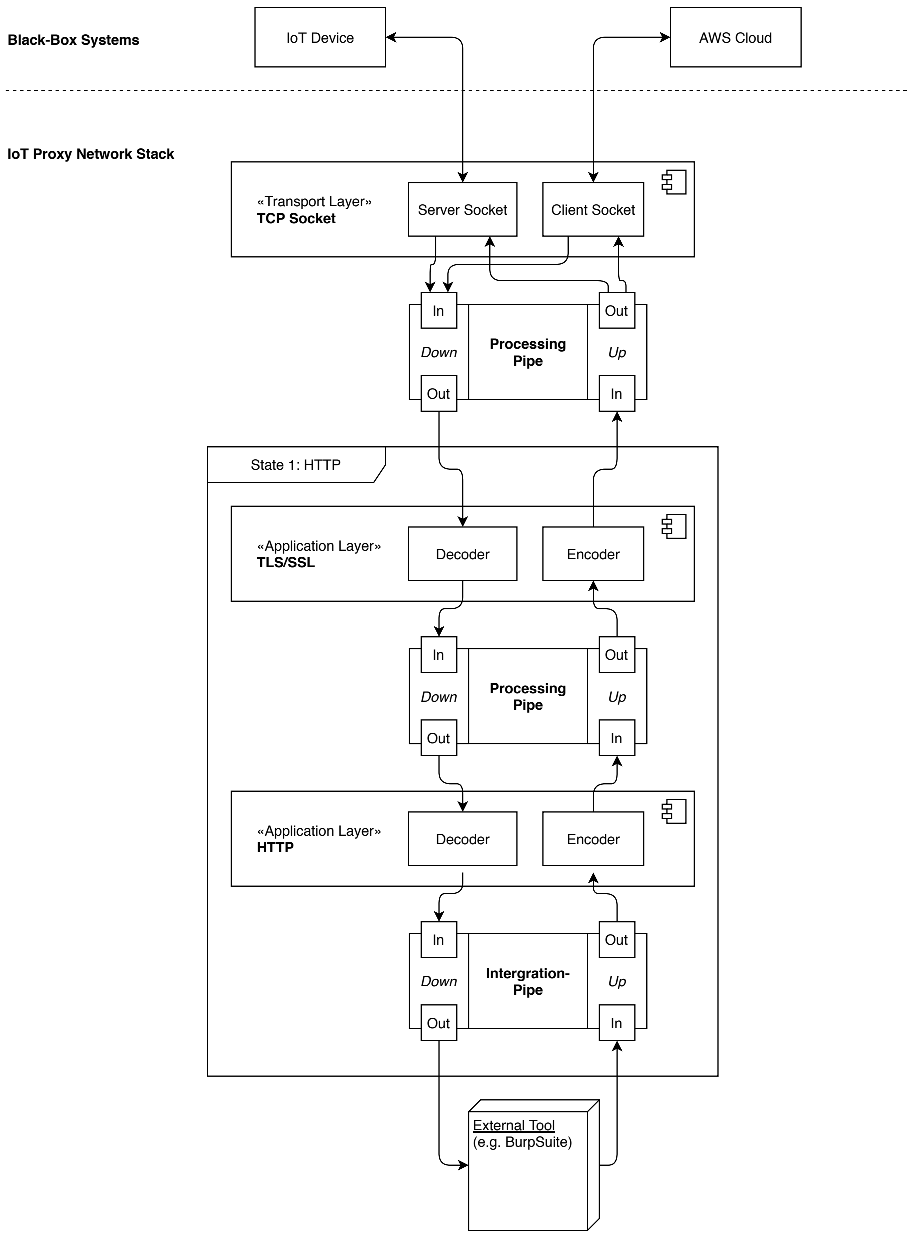


Figure A.1.: AWS IoT Scenario - State 1: HTTP Server

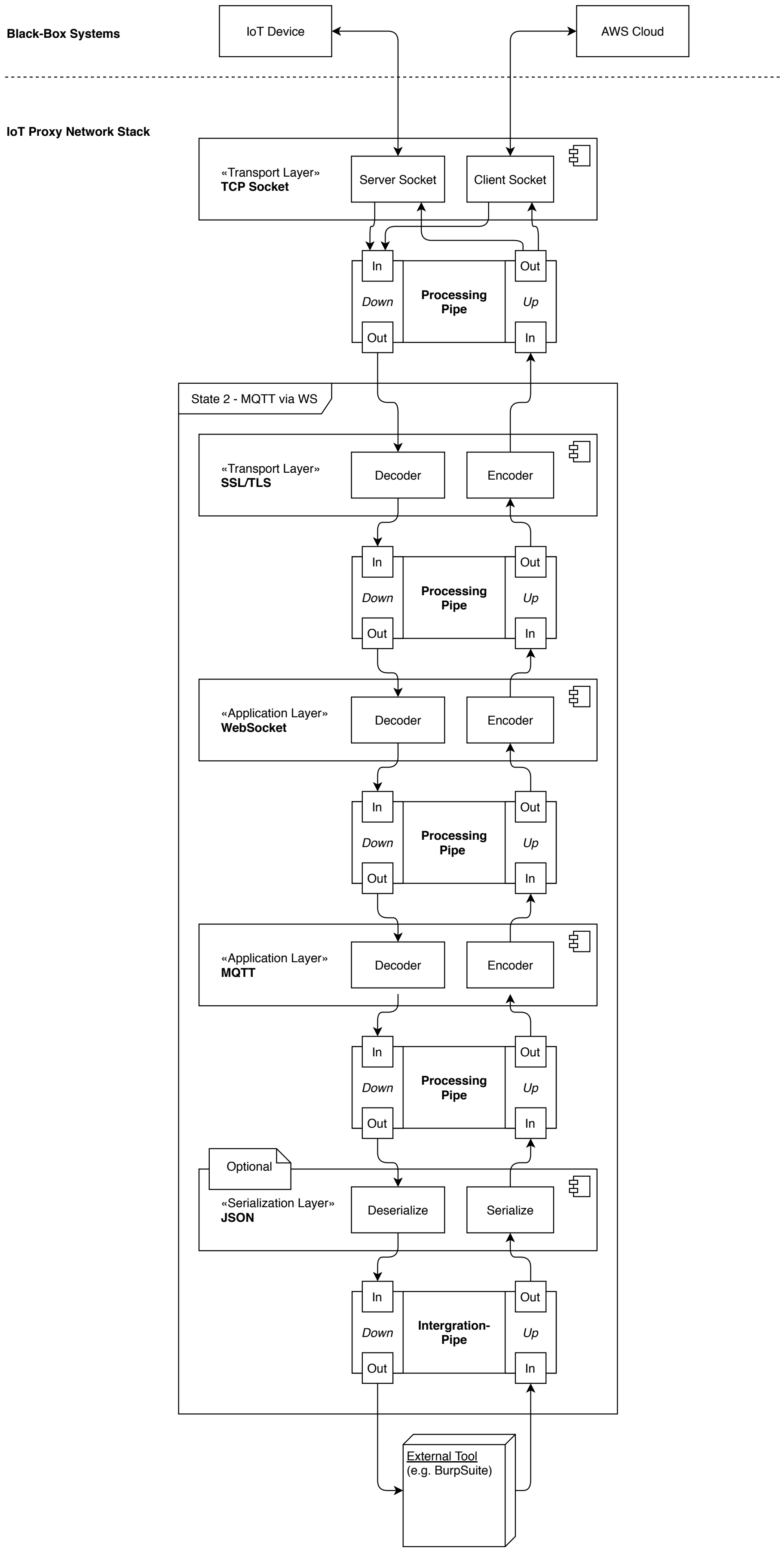


Figure A.2.: AWS IoT Scenario - State 2: MQTT via WS