



**CHALMERS**  
UNIVERSITY OF TECHNOLOGY



UNIVERSITY OF GOTHENBURG

---

# **Semi-Automatic Software Security Model Extraction**

Semi-Automatic Extraction of Security Relevant Information  
from Source Code for Formally Based Security Models

Master's thesis in Computer science and engineering

**NEDA FARHAND**



MASTER'S THESIS 2019

# Semi-Automatic Software Security Model Extraction

Semi-Automatic Extraction of Security Relevant Information from  
Source Code for Formally Based Security Models

NEDA FARHAND



UNIVERSITY OF  
GOTHENBURG

---



**CHALMERS**  
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering  
CHALMERS UNIVERSITY OF TECHNOLOGY  
UNIVERSITY OF GOTHENBURG  
Gothenburg, Sweden 2019

Semi-Automatic Software Security Model Extraction  
Semi-Automatic Extraction of Security Relevant Information from Source Code for  
Formally Based Security Models  
NEDA FARHAND

© NEDA FARHAND, 2019.

Supervisors: Katja Tuma and Riccardo Scandariato, Department of Computer Science and Engineering

Examiner: Jan-Philipp Steghöfer, Department of Computer Science and Engineering

Master's Thesis 2019

Department of Computer Science and Engineering

Chalmers University of Technology and University of Gothenburg

SE-412 96 Gothenburg

Telephone +46 31 772 1000

Typeset in L<sup>A</sup>T<sub>E</sub>X  
Gothenburg, Sweden 2019

Semi-Automatic Software Security Model Extraction  
Semi-Automatic Extraction of Security Relevant Information from Source Code for  
Formally Based Security Models  
NEDA FARHAND  
Department of Computer Science and Engineering  
Chalmers University of Technology and University of Gothenburg

## Abstract

As society becomes increasingly integrated and dependant on software systems, software security is more relevant than ever before. In order to ensure that software applications are secure, different threat modelling techniques are employed. However, many of these rely a great deal on the availability of a security expert and require significant manual effort, often resulting in high time consumption. This thesis describes the development of a tool which automatically extracts a formally specified representation of the software architecture with extended security annotations. The extracted architectural model is known as a “SecDFD”, which is a graph-like representation of software architecture populated with security relevant information from source code, which in turn allows for automated analysis of information flow properties. The SecDFD extraction tool performs semi-automatic extraction of architectural security information from the implementation by processing textual representation of call-graphs together with the source code of the project under analysis. The tool was evaluated by black box testing, and controlled empirical experiments. Our evaluation shows that, while the tool requires further work, it holds potential for use in threat modelling activities.

Keywords: Software, Security, Automation, Extraction, eDFD, Threat Modeling.



## Acknowledgements

I would like to extend my gratitude towards my supervisors, Katja Tuma, Riccardo Scandariato, and Gul Calikli, at Chalmers University of Technology for their guidance, encouragement, and support throughout this process. I would also like to thank those who participated in my experiment not only for giving me and this project a significant portion of their day, but also for their insightful remarks and input. Last but not least I would like to thank my examiner, Jan-Philipp Steghöfer, for his time and valuable feedback.

Neda Farhand, Gothenburg, September 2019





# Contents

|  |             |
|--|-------------|
| <b>List of Figures</b>                         | <b>xiii</b> |
| <b>List of Tables</b>                          | <b>xv</b>   |
| <b>1 Introduction</b>                          | <b>1</b>    |
| 1.1 Scientific Contribution . . . . .          | 3           |
| 1.2 Research Questions . . . . .               | 3           |
| <b>2 Theory</b>                                | <b>5</b>    |
| 2.1 Background . . . . .                       | 5           |
| 2.1.1 The SecDFD . . . . .                     | 6           |
| 2.1.2 Call-graphs . . . . .                    | 6           |
| 2.2 Research methodology . . . . .             | 7           |
| 2.2.1 Design science methodology . . . . .     | 7           |
| 2.2.2 Black box testing . . . . .              | 8           |
| 2.2.3 Qualitative content analysis . . . . .   | 8           |
| 2.3 Related work . . . . .                     | 9           |
| 2.3.1 Architecture reconstruction . . . . .    | 9           |
| 2.3.1.1 Graph based solutions . . . . .        | 10          |
| 2.3.1.2 Clustering algorithms . . . . .        | 11          |
| 2.3.2 Automation of threat modelling . . . . . | 11          |
| <b>3 Methods</b>                               | <b>13</b>   |
| 3.1 Development . . . . .                      | 13          |
| 3.2 Black box tests . . . . .                  | 14          |
| 3.3 Experiment design . . . . .                | 15          |
| 3.3.1 Participants . . . . .                   | 16          |
| 3.3.1.1 Selection . . . . .                    | 16          |
| 3.3.1.2 Participant profiles . . . . .         | 17          |
| 3.3.1.3 Training . . . . .                     | 17          |
| 3.3.2 Subject of analysis . . . . .            | 17          |
| 3.3.2.1 Selection . . . . .                    | 17          |
| 3.3.2.2 Ground truth . . . . .                 | 19          |
| 3.3.3 Execution of experiment . . . . .        | 20          |
| 3.3.3.1 Manual SecDFD extraction . . . . .     | 20          |
| 3.3.3.2 Automatic SecDFD extraction . . . . .  | 20          |
| 3.3.4 Interviews . . . . .                     | 21          |

|           |   |           |
|-----------|---|-----------|
| 3.4       | Analysis . . . . .                              | 22        |
| 3.4.1     | SecDFD comparison . . . . .                     | 22        |
| 3.4.2     | Interviews . . . . .                            | 23        |
| <b>4</b>  | <b>Results</b>                                  | <b>25</b> |
| 4.1       | SecDFD extraction tool . . . . .                | 25        |
| 4.1.1     | Tool development . . . . .                      | 25        |
| 4.1.2     | Implementation choices . . . . .                | 26        |
| 4.1.3     | Tool description . . . . .                      | 27        |
| 4.1.3.1   | Required input . . . . .                        | 28        |
| 4.1.3.2   | Functionality . . . . .                         | 28        |
| 4.1.3.3   | Output format . . . . .                         | 30        |
| 4.2       | Evaluation by black box testing . . . . .       | 30        |
| 4.2.1     | Tool Compatibility . . . . .                    | 31        |
| 4.2.2     | Tool Correctness . . . . .                      | 31        |
| 4.2.3     | C1 . . . . .                                    | 33        |
| 4.2.4     | C2 . . . . .                                    | 33        |
| 4.2.5     | C8 . . . . .                                    | 35        |
| 4.2.6     | C10 . . . . .                                   | 35        |
| 4.3       | Evaluation by Experiment . . . . .              | 35        |
| 4.3.1     | Experiment results . . . . .                    | 35        |
| 4.3.1.1   | Correctness . . . . .                           | 36        |
| 4.3.1.1.1 | SC1 . . . . .                                   | 37        |
| 4.3.1.1.2 | SC2 . . . . .                                   | 38        |
| 4.3.2     | Interviews . . . . .                            | 40        |
| 4.3.2.1   | Participant A . . . . .                         | 40        |
| 4.3.2.1.1 | Automatic extraction . . . . .                  | 40        |
| 4.3.2.1.2 | Manual extraction . . . . .                     | 41        |
| 4.3.2.1.3 | Preferred method . . . . .                      | 41        |
| 4.3.2.2   | Participant B . . . . .                         | 41        |
| 4.3.2.2.1 | Automatic extraction . . . . .                  | 41        |
| 4.3.2.2.2 | Manual extraction . . . . .                     | 42        |
| 4.3.2.2.3 | Preferred method . . . . .                      | 43        |
| <b>5</b>  | <b>Conclusion</b>                               | <b>45</b> |
| 5.1       | Discussion . . . . .                            | 45        |
| 5.1.1     | Relation to the field . . . . .                 | 45        |
| 5.1.2     | Functionality and correctness . . . . .         | 45        |
| 5.1.3     | Information from source code (RQ1) . . . . .    | 47        |
| 5.1.4     | Comparison to manual extraction (RQ2) . . . . . | 47        |
| 5.2       | Threats to validity . . . . .                   | 47        |
| 5.2.1     | Internal validity . . . . .                     | 47        |
| 5.2.2     | External validity . . . . .                     | 48        |
| 5.2.3     | Reliability . . . . .                           | 49        |
| 5.3       | Conclusion . . . . .                            | 49        |
| 5.3.1     | Future work . . . . .                           | 50        |

|                     |           |
|---------------------|-----------|
| <b>Bibliography</b> | <b>53</b> |
|---------------------|-----------|



# List of Figures

|     |   |    |
|-----|---|----|
| 3.1 | An overview of the experiment procedure . . . . .   | 16 |
| 4.1 | The process of the semi-automatic SecDFD extraction procedure from<br>the user's perspective. . . . . | 26 |
| 4.2 | A domain model of the SecDFD extraction tool's core elements. . . .                                   | 29 |
| 4.3 | The number of files generated by Doxygen per project used in the<br>black box testing phase. . . . .  | 32 |



# List of Tables

|      |   |    |
|------|---|----|
| 3.1  | SC1 characteristics . . . . .   | 19 |
| 3.2  | SC2 characteristics . . . . .   | 19 |
| 4.1  | Requirements for the SecDFD extraction tool . . . . .   | 27 |
| 4.2  | Settings used in Doxygen to produce call-graphs in the format required by the SecDFD extraction tool . . . . .  | 27 |
| 4.3  | A list of the projects used for the black box testing. . . . .  | 30 |
| 4.4  | Illustration of whether or not each project produced the expected files when running Doxygen, and produced a non-empty SecDFD when running the extraction tool. . . . . | 31 |
| 4.5  | Reason for negative result (empty SecDFD) in cases where Doxygen produced call-graph files, when applying the “open box” keyword determination method. . . . .          | 32 |
| 4.6  | Adjacency matrix for the SecDFD automatically extracted from the CoAP IoT Server (C1). . . . .  | 34 |
| 4.7  | Logical appearances of elements from Ground Truth in automatically extracted SecDFD for the CoAP IoT Server (C1). . . . .   | 34 |
| 4.8  | Adjacency matrix for the SecDFD manually extracted from the CoAP IoT Server project. . . . .  | 34 |
| 4.9  | Logical appearances of elements from Ground Truth in automatically extracted SecDFD for JPetStore (C10). . . . .  | 36 |
| 4.10 | Time consumption for SecDFD-extraction using the two different extraction methods, Session A . . . . .  | 36 |
| 4.11 | Time consumption for SecDFD-extraction using the two different extraction methods, Session B . . . . .  | 37 |
| 4.12 | Logical connections for the Watering System (SC1). . . . .  | 37 |
| 4.13 | Adjacency matrix for the Ground Truth of the Watering System (SC1). . . . .   | 37 |
| 4.14 | Adjacency matrix from session A for the Watering System (SC1). . . . .  | 38 |
| 4.15 | Adjacency matrix from session B for the Watering System (SC1). . . . .  | 38 |
| 4.16 | Logical connections for the Access Control system (SC2). . . . .  | 38 |
| 4.17 | Adjacency matrix for the Ground Truth of the Access Control system (SC2). . . . .   | 39 |
| 4.18 | Adjacency matrix from session A for the Access Control system (SC2). . . . .  | 39 |
| 4.19 | Adjacency matrix from session B for the Access Control system (SC2). . . . .  | 39 |





# 1

## Introduction

As information technology continues to be increasingly integrated into every aspect of our lives – ranging from medical monitoring devices to smartphones, home assistants, and even kitchen appliances – the computer security community has become increasingly aware of the shortcomings in established security protocols [41]. In addition, the number of connected devices has seen a large increase, and the numbers are projected to grow even further with forecasts stating that there will be 22 billion networked devices in 2022, out of which 18 billion will be in the Internet of Things (IoT) domain as opposed to approximately 400 million IoT devices with a cellular connection at the end of 2016 [12].

The relatively young field of IoT has come to show that the protocols written on the foundation of personal computers are not applicable to applications running on resource constrained devices [41], meaning that a significant portion of connected devices are not secure. Such vulnerabilities are not limited to edge cases, but are apparent in everyday networked appliances such as smartphones, which are susceptible to a wide range of malware attacks [31]. With this follows the risk of not only compromised privacy, but also physical security. With the technological area increasing at such a rapid pace, implying that a substantial portion of software development will come to take place within the IoT domain, it is more important than ever that the security community keeps up.

By taking security into account during the development of such applications, aiming for software that is secure by design, companies and developers are able to prevent major maintenance costs and loss of face that would otherwise have occurred upon the discovery of a vulnerability. Yet, this remains a difficult task since many security-by-design approaches require a security expert – a resource that is scarce in many organizations [37]. Hiring a security expert for threat modelling is, while arguably a necessary and common practice, an additional cost to the project. Some problems may have been easily preventable given awareness of the how the vulnerability manifests. Thus, it would be less costly if developers and software architects themselves were to detect such security issues before reaching the code production stage, as the cost of mending vulnerabilities is known to increase over time [24]. Even so, there remains a risk of a disconnect between the architectural models and the implementation of the software system, meaning that there is an additional gap to bridge in the *planned* security, and the *implemented* security. This presents the need for post-production security analysis of existing applications, which would take even an expert some time as it requires code review. In addition, if even the most basic security issues manifest in the code, the analyst’s effort may be diverted from more complex issues.

By automating the threat modelling process, the industry could save a great amount of resources – reducing not only the number of working hours spent on manual threat analysis and modelling, but possibly also reducing the risk of potentially costly human errors. Should such an automation framework build on static analysis, a correct and versatile solution would require an extensive set of formally expressed, predefined rules that map to the desired security model. Such a ruleset may prove highly cumbersome and time-consuming to produce.

By obtaining an initial threat model without the involvement of a security expert, resources could be saved, and said expert could primarily focus on more complex tasks. On the other hand, should this threat model be manually obtained by the architects or developers, the earlier presented problem remains; such a model may not be correct and would also eventually become outdated, resulting in a waste of time and resources rather than benefit. However, should you automate the threat modelling procedure, basing such a model on resources produced during the architecture design phase – namely, Data Flow Diagrams (DFDs) – one could greatly lower the threshold of security analysis, as well as increase the chance of correctness.

This thesis starts from the bottom-up part of the problem, addressing the challenges of threat analysis from code to model. In this case, the end-model to be extracted is a Security Data-Flow Diagram, also known as a SecDFD – a data-flow diagram with extended security annotations, focusing on the flow of information assets through a software system [34]. This is particularly challenging within the complex, large-scale systems often found in IoT, which will be the main domain of application in this thesis.

This thesis concerns an exploratory research project which consisted of the development and two part evaluation of a static analysis tool, referred to as the “SecDFD extraction tool”, which extracts assets, flows, and processes by utilizing the calls within the code base of a given application to parse its source code and extract these necessary elements of a SecDFD. The outcome of running the extraction tool is a SecDFD of the application, outputted by the program as a file written in the SecDFD specification language. The evaluation was done by means of black box tests, as well as empirical experiments.

The core idea of the automatic extraction procedure developed as part of this thesis is to use the information given by a call graph in order to 1) identify the information assets by cross-referencing the parameters of the functions against a list of security relevant strings, 2) determine the responsibilities of the respective nodes by observing their manipulation of the information assets, and 3) based on the collective information, produce a SecDFD that represents the source project.

The remainder of this thesis is structured as follows: Section 1.2 presents the research questions of this thesis. Chapter 2 concerns the background and related work that lay the foundation for the conclusions, assumptions, and procedures used throughout this thesis, as well as the justification for the design choices regarding both tool- and experiment design. Chapter 3 presents the methodology followed for developing the tool, the evaluation with black box testing, and the evaluation with an empirical experiment. Chapter 4 presents the results. Chapter 5 is structured as follows: Section 5.1 discusses the results, and answers the research questions. Section 5.2 discusses the threats to validity. Finally, Section 5.3 contains a reflection and

summary of the thesis, as well as the future work.

## 1.1 Scientific Contribution

This thesis aims to present a novel threat modelling procedure by means of automating the process of extracting a SecDFD from source code. In particular, this thesis describes a semi-automated procedure that would significantly reduce the time consumption and need for manual involvement in SecDFD-creation from source code. Automation thus results not only in saving a significant amount of resources, but also in a greatly reduced risk of manifestation of inaccuracies as a result of human error, as well as consistency in the final model seeing as the SecDFD-creation would no longer necessarily rely on individual human interpretation. Thus, the thesis presents an implementation of a tool which extracts a SecDFD based on call-graphs obtained from the source code combined with a list of keywords, a semi-automated procedure for extracting a SecDFD, as well as a means for qualitative comparison between SecDFDs.

## 1.2 Research Questions

The aim of this study is to facilitate the threat modelling procedure by designing and developing a novel approach to automatically extract high-level security information from source code, and transforming this into a SecDFD. As of this time, the only occurrence of SecDFDs is found in research, more specifically in the paper by Tuma [34], which serves as the origin of the SecDFD and details the creation of a SecDFD from a thorough conceptual description of the software under analysis and its different components. Much like other approaches for security-by-design [35], SecDFD creation requires security expertise – manual SecDFD extraction, especially, relies heavily on expert knowledge of the information assets and their significance. Thus, the current assumed method of extraction requires expert knowledge of the respective roles and functionality of the components that build up the software, in correspondence to the different elements of the SecDFD. This raises the question of how to best mimic this kind of expert knowledge, which gives us our first research question:

*How can the security-relevant information be leveraged during automatic SecDFD extraction? (RQ1)*

In order to formulate a means for extraction, it is important to identify what exactly is to be extracted. As the current method strongly relies on the understanding and case-to-case interpretation of a human expert, there is no clear indication of any particular syntactic marker in a set of source code which will definitely and uniquely translate to a SecDFD element. Should such a marker exist and be identifiable by a machine it will determine which approach is best suited for implementing automated SecDFD extraction. This renders us the additional question:

*What information is required from the source code in order to automatically extract a SecDFD? (RQ1.1)*

To solidify the scientific contribution and contextual significance of the novel approach described in this thesis, it is important to consider the performance of this tool in relation to correctness, as well as whether its usability overall puts usage of the tool at an advantage over manual extraction. Thus, we are interested in comparing these two approaches.

*How does manual creation of a SecDFD compare to SecDFD extraction using the tool? (RQ2)*

In particular, we are interested in whether one method is of greater advantage than the other in some particular use case. Additionally, should one method of SecDFD creation appear to be overall preferable, it may come to affect the future direction of research conducted on the topic. Thus we should aim to identify the differences and areas of greater suitability of each method, as compared to the other. This poses us with the following research question:

*What are the advantages or disadvantages of manual creation, compared to using the tool? (RQ2.1)*

# 2

## Theory

### 2.1 Background

As briefly touched upon in Chapter 1, there are a multitude of ongoing challenges and research opportunities in IoT security, as presented in the research conducted by Zhang et al. [41]. While the study was conducted in 2014, many of the problems presented still hold true today. This is further emphasized by the research of Conti et al. [8] conducted in 2018. They identified, among other things, that building a secure architecture is one of the main ways of overcoming vulnerabilities presented by so called Software Defined Networks that would otherwise be passed down to any underlying IoT sensors. As the need for understanding and implementing IoT security rises, so does the need for efficient and reliable methods of evaluating the security of IoT applications.

Security will not cease to be a relevant field of research in software engineering and computer science. On the contrary, with the impending shift of quantum computing, we may soon find that our de-facto security measures, especially cryptographic algorithms, will soon be obsolete. There is ongoing research on security in the age of quantum computing, such as that by Liu et al. [22] which considers the security of edge devices in such a context. By preparing, and by securing more efficient means of securing software at a time when cracking security is becoming rapidly more effective, the security community will secure itself against the pending shift.

Implementing security requires understanding of not only the application which is to be secured, but also the threats it needs to be secured against. For this purpose, we turn to the Microsoft STRIDE threat modelling procedure which is a method for systematically identifying and evaluating security threats by analyzing a graphical representation of the software architecture [27]. The STRIDE threat modelling procedure is advocated by Microsoft for ensuring IoT security architecture [32], and is mirrored in the SecDFD – the core elements of the threat model being processes, data flows, data stores, and external entities, as well as both being based on creating a diagram of the reference architecture.

The verification of automated techniques for architecture recovery based on any kind of architectural ground truth has proved itself to be a significant hurdle in multiple cases, enough so to warrant the collection of ground truths being a significant portion of a project. In their 2013 study, Garcia et al. [15] performed a comparative analysis of six different techniques for architecture recovery, in which they collected a total of eight architectural ground truths from the open source com-

munity that had been independently verified. In the same spirit, in 2015 Lutellier et al. [23] evaluated six architecture recovery techniques, with the addition of a project significantly larger than those previously studied with the same measures – consisting of millions of lines of code – for which creating the architectural ground truth required collaboration with the project’s developers over the course of two years.

There is also the matter of investigating the correlation between human readable language and code, to reflect the high-level and individualistic approach taken during modelling processes. The semantic parser presented in the research by Quirk et al. [28] was described to show a promising foundation for further work, with the added remark that out of the taken approaches it performed best in a loosely synchronous context. Their method clearly relied on a deterministic, self described as an “if-this-then-that” approach to lessen the gap between programming and natural language. Judging by their work, discerning the significance and role of coding components by their semantics is not only possible, but promising.

### 2.1.1 The SecDFD

The SecDFD [34] can be described as a graph where each node represents either a process, external entity, or data store. Each node, in turn, is distinguished by one of four contracts in regard to label propagation; encrypt or hash contract, decrypt contract, join contract, or copy contract [34]. These different properties describe the responsibilities of that particular node towards any asset it comes into contact with. The SecDFD is designed to consider information security and the securing of flows in networked applications on different abstraction levels, making it highly versatile and applicable to the challenges that lie ahead in the sense of the oncoming growth in IoT application areas.

In this thesis, we interpret the nodes to represent classes; the directed flows between the processes represent method calls; and the parameters passed by those methods represent information assets. We consider the CIA (Confidentiality, Integrity, Availability) model for computer security. In practice, if naming conventions in regard to descriptive naming are followed, the manifestation of such an information asset in source code will be named – entirely or in part – after the information it represents. Thus, when strings such as “key” or “id” appear as parameters in a function, it is likely that they represent somewhat sensitive information. Examples of such assets would be user data such as user id, authentication keys, passwords, and so forth.

### 2.1.2 Call-graphs

For the purposes of this study, textual representation of call-graphs are chosen as an intermediate between source code and SecDFD generation. This is due partly to the straightforward relationship between call-graphs and SecDFDs in terms of each node and flow being clearly represented, which in turn makes the flows more clearly distinguishable and thus distills the entirety of the information given through the source code down into the information needed for SecDFD extraction. This is

due to the correlation between the direction of the call, and the direction of the data flow, which creates a link between the call-graph elements and the components of the SecDFD. Additionally, there exist established tools on the market that can extract call-graphs from source code, which in itself is a complex task. From there, it is only a matter of parsing the textual representation of the call-graph given the syntax specifications of the language it is defined in.

## 2.2 Research methodology

Runesson et al. [29] define experiments as the measurements of effects from manipulating variables on another variable, combined with the random assignment of treatments to subjects. Additionally, they mention the different purposes served by different research methodologies – namely *exploratory*, *descriptive*, *explanatory*, and *improving* research [29]. This thesis concerns *improving* research, specifically improving threat modelling techniques, as the goal is to automatize SecDFD extraction and thus reduce the time consumption.

Engstrom et al. [11] categorize *solution design* as belonging to the category of design science, and have in their 2019 research, by studying 38 ICSE distinguished papers, evaluated how well viewing research from a design science point of view can help illustrate phenomena within software engineering communication – framing the projects they analyzed as models of iterative loops between a *problem instance* and *solution*, between which validation occurred. Their recommendations for communication of software engineering research, to name a few, are to be clear regarding the technological settings in which the research was conducted in order to help advance the field and aid fellow researchers. They also mention placing descriptions on an apt level, tying the research into tangible problems in the domain, and visual aids as a means of carrying the findings of one’s research across.

This thesis strives to meet the recommendations stated by Engstrom et al. through relating the work to the design science life cycle, as well as providing a clear description of the methodology and technological settings in which the thesis was conducted.

### 2.2.1 Design science methodology

When it comes to design science, the main topic is understanding the activities of the research, which are defined by Wieringa [39] as the *study*, *design*, and *investigation* of a software artifact in its context. Furthermore, Wieringa defines the design process as a cycle consisting of five phases; *problem investigation*, *treatment design*, *treatment validation*, *treatment implementation*, and *implementation evaluation*. These phases are a part of a rational problem solving process which concerns the application of a treatment to a problem and an evaluation of the effects and effectiveness of said treatment.

In the context of this thesis, and by placing the parts of this thesis in relation to the definitions from Wieringa [39], the relation is as follows:

1. **Problem investigation:** defining what must be treated. In this thesis, this part constitutes both the acquirement of what Wieringa refers to as *prior knowledge* [39] by reviewing literature, but also requirements elicitation as described in Chapter 3.
2. **Treatment design:** the design of an artifact to treat the problem, which in the context of this thesis constitutes the design of the SecDFD extraction tool by identifying and specifying the scope, functionality, and dependencies.
3. **Treatment validation:** observing whether the designed treatment would treat the problem, which in this case would be more aptly phrased as “would this program be able to generate a SecDFD from source code?”. In the thesis, this step corresponds to the tuning of the tool during the development phase.
4. **Treatment implementation:** the application of the designed artifact to treat the problem, which corresponds to the final implementation of the tool being used to extract a SecDFD in full.
5. **Implementation evaluation:** determining the success of the treatment. In this thesis, this step corresponds to the black box tests and experiments, which were used to determine the correctness of the tool and answer the research questions.

Wieringa goes on to describe the “Engineering cycle” as the re-iteration of the first four steps by feeding the implementation evaluation into the problem investigation step, noting that the difference between the problem investigation and the implementation evaluation is that the purpose of the problem investigation is to prepare for the design of a treatment while the purpose of the implementation evaluation is to evaluate the success of a treatment when applied to the original problem [39].

### 2.2.2 Black box testing

Black box testing, also known as functional testing, is a kind of software testing technique which can be used once the software is fully implemented. The test cases are specified based on knowledge of the expected behaviour and functionality of the software under test, often based on the functional specification of said software. The purpose of black box testing is to test modules independently, verify system behaviour, check performance, stress the system, and check for interface errors, amongst other things [7].

In the tests carried out in this thesis, the black box testing technique is used, and the software under test is the SecDFD extraction tool. It should be noted that this is not interchangeable with the “open box” keyword determination method, which refers to the process of specifying the keywords used by the tool based on prior knowledge of the names of the assets in an available, manually extracted SecDFD for a project under analysis.

### 2.2.3 Qualitative content analysis

Wildemuth et al. describe qualitative content analysis as the procedural condensation of raw data into categories or themes based on inference and interpretation,



which is a process that relies on the researcher’s consideration and comparison of the data in order to apply *inductive reasoning* [40].

Furthermore, the literature goes on to describe the process of data preparation, specifically the preparation of interview transcripts as the most often used data format for qualitative content analysis. The transcripts can be either literal or a summary. This is followed by defining a unit of analysis, which can be a theme – a theme, in turn, is described as “the expression of a single idea”, which can be found in a single sentence or an entire document alike [40].

This definition of qualitative content analysis and inductive reasoning by the experimenter is applied throughout this thesis in regard to the correctness evaluation as part of both the treatment validation and the implementation evaluation, including both the correctness evaluation of the output from the semi-automatic SecDFD extraction procedure as well as the evaluation of the interviews as part of the experiment, both described in Chapter 3.

## 2.3 Related work

This section concerns previous research relevant to the work conducted in this thesis, addressing research regarding architecture reconstruction as well as the state of IoT security and practices in threat modelling.

### 2.3.1 Architecture reconstruction

A great deal of work has been conducted in the area of architecture reconstruction. A large part of the body of research in this domain concerns the application of machine learning and other statistical methods for extracting software architecture. As the SecDFD is first described in the work by Tuma [34], there is no precedent of applying such research to this particular model at this time.

Regarding applying a machine learning approach to the problem of architecture reconstruction, work conducted in 2012 by Sajnani [30] describes the process of using machine learning as a means of recovering software architecture without having a pre-establish set of insights into the software and its various properties. Their end goal was to produce a model that illustrated the call flow within an application. They had studied among other things, domain feature extraction and feature classification using supervised learning techniques, as per what was state of the art at the time.

Maqbool et al. [25] have, in their 2007 work, explored the possibility of using Bayesian learning methods to recover software system architecture by labelling known software systems in order to train a so called “Naïve Bayes Classifier”. They observed the percentage of correct matches by dividing systems into subsystems, reaching about a 50% observed correctness rate. As they state, investigating how their method would perform using other supervised learning methods and a greater number of classifiers could prove interesting.

The work by Garcia et al. [16] investigated whether extracting concerns could aid in producing increasingly correct outcomes from automatic architecture extraction processes. In their work they describe a novel model which utilizes the high level concepts of *components* and *connectors* to describe what would make up a concern,

and apply statistical methods to in turn extract the concerns. Their initial results indicate the usefulness of a supervised learning approach to the problem. There is also research supporting semi-supervised approaches for label-propagation as the process of labelling data sets may in itself prove cumbersome, such as the work by Wang et al. [13] which introduces a means of doing such based on traversing linear neighbourhoods.

Regarding tools for architecture extraction, Fontana et al. [4] propose an Eclipse plugin called “MARPLE”, which is capable of extracting architectural information such as design pattern detection and metrics from a project. They mention that their tool is a novel contribution, but that attempts to make comparisons of their results were inconclusive [4]. There is also the work of Granchelli et al. [17], which concerns a program for extracting micro-service based architectures, and showed promising results based on the carried out validation.

Using reflexion models, as introduced by Murphy et al. [26] in 2001, software engineers can better reason about software artifacts by leveraging the gaps between design and implementation, and evaluating their consistency. Additionally, Koschke et al. [21] have extended the software reflexion model to include non-hierarchical software architectures. The technique describes the process of comparing software models to their reference material, which in turn is essential for evaluating architectural correctness. This principle, in turn, is considered in the evaluation parts of my work.

This thesis concerns the extraction of security architecture from source code, more specifically a SecDFD, which distinguishes it from the methods for architecture reconstruction mentioned in this paragraph in the sense that the extraction of this more specialized security model poses different requirements on the methods used for extraction, such as identification of design patterns. Additionally, the works that apply machine learning to the problem are based on extracting general architecture, whereas the SecDFD is specialized, making appropriate data sets more difficult to obtain.

### 2.3.1.1 Graph based solutions

Regarding similar research, Brown et al. [6] propose, in their 2004 study, a method of applying a graph based genetic algorithm (GA) for multiobjective evolution of median molecules, utilizing graph-based mutation- and crossover operators in order to evolve their nodes and sub-graphs. A particularly interesting aspect of their research concerned graph-based crossover operators<sup>1</sup>, in which they bring up multi-point crossover and sub-graph crossover. Their initial results were promising, showing that the method is capable of producing the desired results.

Using another approach than applying statistical methods strictly to the specific problem of architecture recovery, Fouss et al. [14] researched how random walk computation of similarities between nodes in a weighted, undirected graph could be used to compute similarities between elements in a database. While this strays a bit from the original problem of this thesis, which could in a way be likened to wanting

---

<sup>1</sup>Crossover, in a GA, is the exchange of genetic material around a certain point. A crossover operator would determine what the point around which the exchange takes place is.

to treat a *directed* graph, it is worth noting that their work lead to a computable quantity, which is highly relevant in the need for numeric scoring.

In future applications, it may be of interest to further investigate the applicability of the scoring mentioned by Fouss et al. [14]. Since this thesis treats the SecDFD as a graph, and concerns the comparison between two or more such graphs, having a numeric scoring system becomes highly relevant when facing machine learning application.

### 2.3.1.2 Clustering algorithms

The SecDFD extraction involved the clustering of processes in areas of the diagram where it is considered applicable [34]. When it comes to machine comprehension, Tzerpos et al. [36] present a number of properties that will in turn ensure helpful output in the case of attempting to cluster source files. These include effective naming and a pattern oriented approach, for which a number of possible subsystem patterns for naming purposes are suggested. Additionally, research by Corazza et al. [9] showed that applying weights to clustered areas when applying an Expectation-Maximization algorithm contributed to better results.

A further remark on the work by Tzerpos et al. [36] is that the results produced by applying their ACDC clustering algorithm showed fairly stable and correct results when applied to skeleton construction problems. Their algorithm is pattern-driven, and their identified stages of skeleton construction are applicable to a wide range of graph based problems.

Another such algorithm is LIMBO, as presented in 2004 by Andritsos et al. [2], which is a scalable, hierarchical clustering algorithm. While LIMBO was proven to show increases in efficiency, there were some impacts on quality.

While these methods are of interest, this thesis recognizes that the extraction of a SecDFD from source code will, as the size of the code base increases, become increasingly complex if clustering was to take place only after all nodes have been obtained. While clustering algorithms are useful and applicable for further refinement of the process, the initial work utilizes the clustering effect that takes place by conditional adding of nodes, as discussed in Section 4.1.2.

### 2.3.2 Automation of threat modelling

There is research stating the benefits of automated architecture extraction and analysis procedures, such as that by Velasco-Elizondo et al. [38] indicating that automatic information extraction leads to less time-consumption, as well as better recall. When looking specifically into the subject of threat modelling, Tuma et al. [35] introduce a means of increase the efficiency of the threat modelling process by using an extended DFD notation (eDFDs), focusing on end-to-end asset flows, and have thus shown that the effects of abstraction before threat analysis greatly counteracts the threat explosion otherwise common when taking the traditional approach to the STRIDE security model on a larger DFD. Another study that incorporates DFDs into the threat modelling process is that of Jasser et al. [20], which proposes an approach for bringing security into the workflow in an architecture-centric develop-

ment process. Additionally, Jasser et al. [20] touch upon the fact that very few attempts have thus far been made on automating STRIDE.

In a related take, in 2010 Abi-Antoun et al. [1] presented SECORIA – a semi-automated approach to threat modelling, which mapped to the STRIDE threat modelling procedure, and proved to be highly correlated to expert behaviour in the sense of threat modelling. It is also worth noting that there are other approaches to security evaluation, such as that presented by Antonini et al. in 2010 [3] which focuses on service-oriented architectures, dividing the security evaluation process into three phases; extraction, identification, and analysis.

Extended data-flow diagrams, or eDFDs, have been applied to ongoing research in the field of architecture-centric security analysis. Furthermore, research has been conducted by Tuma et al. [34] in their work on SecDFDs. Additionally, Bergher et al. [5] have studied the automatic extraction of threats from eDFDs using graph query rules that each map to an entry in a threat catalogue based on established threat-modelling resources. Furthermore, Bergher et al. mention that static analyzers are widely used by industry in the context of threat modeling. There is most likely good reason for this, as is further solidified by Shabtai et al. [31] who achieved promising results by combining static code analysis with machine learning algorithms for classification of Android applications.

This work concerns the semi-automatic extraction of SecDFDs, which can be analyzed using the SecDFD analysis plugin as described in the work by Tuma et al. [34]. By partially automating the SecDFD extraction process, it is likely that an outcome will be a model which can be more easily interpreted by a machine, as well as less resource-consuming means for producing large quantities of SecDFDs, which in turn opens up for machine learning application to the SecDFD analysis process.

# 3

## Methods

The work was conducted by design, implementation, and evaluation of the program designed to statically extract a SecDFD from source code also referred to as the “SecDFD extraction tool”. Viewing these phases from the point of view of the design cycle for design science research, as described in Sections 2.2.1 and 2.2, the methodology describes a single iteration of the whole cycle over the course of the thesis where the first four phases – i.e. the engineering cycle – are internally iterated as part of the development described in Section 3.1.

The implementation evaluation of the thesis was conducted in two parts: an evaluation of a number of SecDFD pairs, extracted by both manual- and semi-automatic extraction, as well as an experiment, where both perceived user experience and produced results using these two different methods. This section will provide further detail on the development of the program as well as the methodology of the tests and experiments, followed by a description of the analysis of the results.

### 3.1 Development

The development mainly consisted of four phases, corresponding to the first four phases of the design science cycle as defined in Section 2.2, which were carried out sequentially in a single iteration. Below is a summary of each phase, including an indication of which section of the thesis it is addressed in:

1. **Problem investigation:** understanding the goal of the tool by literature review, eliciting requirements from SecDFD experts – which, in this case, consisted of discussing with my supervisors to determine what the requirements should be – and studying formally specified SecDFDs. The results of this phase are described in Section 4.1.1.
2. **Treatment design:** designing the tool by programming a draft and specifying the program flow, including the interaction with third party programs. The results of this phase are found in Section 4.1.2.
3. **Treatment validation:** validating the procedure effectiveness by following the program flow and running the tool on a set of source code for which there was already a manually extracted SecDFD available, and observing the gap between the automatically extracted SecDFD. The sets of source code that were paired with a manually extracted SecDFD were attained by the author manually extracting SecDFDs from openly sourced Java projects in

the *Internet of Things* category on GitHub<sup>1</sup>. The gap, in turn, consisted of the differences between the manually extracted SecDFD, and the one produced by the extraction tool. The manually extracted SecDFD was considered the template for a "correct" SecDFD, and thus the ground truth. The gap was identified by following the SecDFD comparison process as further described in Section 3.4.1. This phase followed the black box methodology as described in Section 3.2.

4. **Treatment implementation:** final implementation of the tool by adjusting it in order to bridge the gap identified in the previous step, and specification of the end-to-end process of semi-automatically extracting a SecDFD. The gap was bridged by making adjustments to the tool implementation in order to bring the SecDFD produced by the tool closer to the ground truth, i.e. reduce the differences between the semi-automatically extracted SecDFD and the manually extracted SecDFD to the point where they were considered equally correct representations of the software under analysis. The results of this phase, and thus the sequential execution of all steps above, are found in section 4.1.3.

## 3.2 Black box tests

The black box tests were a part of the *implementation evaluation* in the design science context. A complementary set of tests were carried out on every available set of SecDFDs manually extracted by an acting expert *other than the author* from source code that was openly available. It was important not to use projects that had already been used or were intended for other parts of the project, such as the open source java projects that were reserved for use in the experiment phase, or any project used for fine-tuning of the extraction tool (see section 4.1.2). This was a necessary action due to the fact that the ground truths in these cases, which had been extracted by the author, thus followed the same principles and process flow that the program was tuned to. Thus, using projects that had manually extracted SecDFDs extracted by someone else, as well as not re-using projects that had already been analyzed, allows for making more generally based statements in regard to the correctness of the tool due to a greater spread of data.

Due to the fact that SecDFDs manually extracted by the author could not be used during this phase, the projects were chosen on an availability basis – that is, only if there existed a manually extracted SecDFD, that the author had not extracted themselves, for a certain project could that project be used for black box testing. Due to the novel nature of the SecDFD there exist few projects of that kind, meaning that there is little room to impose requirements without further reducing the data quantity. Thus, all projects made available by the supervisors was used in this phase. These projects varied in programming language, size, and compliance with naming conventions.

Most importantly, the black box testing phase utilized the “open box” keyword determination method, which is based on knowing the correct asset names before-

---

<sup>1</sup><https://github.com/topics/internet-of-things?l=java>

hand due to having expertly extracted SecDFDs available that act as the ground truth. The “open box” keyword determination method is followed by putting the names of the assets from the ground truth into the file `keywords.txt`, which is where the list of keywords is stored, as the acting keywords – splitting up the longer asset names to sub-parts where these still constituted grammatical words. These then acted as the set of selected keywords over a *single* iteration, in order to test the apparent association between asset names and variable naming. The output retrieved from this process was then subjected to the analysis described in section 3.4.1, paired with the manually extracted SecDFD constituting its ground truth.

The keywords are used by the tool to find assets, and thus a crucial aspect of the tool functionality. Each keyword, defined as a string, is searched for in the source code, and if the string appears as a substring in a variable or function in the source code, an asset is identified and extracted to the generated SecDFD, providing that there exists a data-flow.

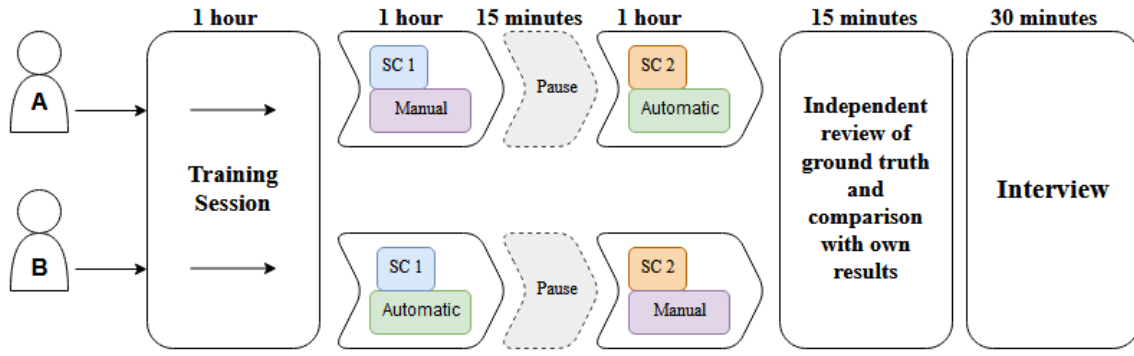
The black box tests were carried out by generating the required call-graph files using Doxygen, and subsequently running the SecDFD extraction tool on each of the projects providing that the necessary files had been generated by Doxygen. Each project was only subjected to a single iteration of the tool, where *iteration* refers to adjustment and re-specification of keywords. This means that once the keywords were defined, the tool was executed once, and the produced results were considered final even though re-specification of keywords may or may not have led to more correct output. This choice was made in order to ensure that the projects were tested under as equal conditions as possible, and eliminating human interaction with the tool during the extraction process to the greatest possible extent given the current implementation of the tool.

### 3.3 Experiment design

Like the black box test, the experiments were a part of the *implementation evaluation* in the design science context. This section details the design of the experiment, dividing each sub-task of the experiment into subsections. A graphical overview of the experiment can be found in Figure 3.1, where the label “Manual” means that the participant was tasked with manual SecDFD extraction at the indicated stage, and the label “Automatic” means that they were tasked with using the tool and thus the semi-automatic SecDFD extraction procedure. The respective labels “SC 1” and “SC 2” indicate which of the projects they were tasked with performing the indicated extraction method on.

Each session was recorded in audio for purposes of documentation and post-experiment analysis. Additionally, the participants signed contracts assuring their anonymity, as well as their agreement to save the collected data for a specified amount of time, and to not to discuss the events of the experiment for that duration as to counteract the risk of threats to internal validity by diffusion caused by interaction between participants.

The participants were encouraged to think aloud throughout the entirety of the experiment. This was done in order to counteract the risk of the author making faulty assumptions about their actions and motivations throughout the experiment



**Figure 3.1:** An overview of the experiment procedure

which would in turn run the risk of affecting the validity of the conclusions.

### 3.3.1 Participants

The experiment involved two participants, hereon forth individually referred to as *Participant A* and *Participant B*. As the aim of the experiment was to attain qualitative results indicating the benefit of using the tool for automated SecDFD-extraction, the main objective was not to have as large a quantity as participants as possible, but rather to ensure that the participants had the necessary prerequisites to carry out the tasks, and that the conditions given to each participant would equally emulate a realistic scenario.

#### 3.3.1.1 Selection

The participants were selected based on the following criteria:

1. *The participant has completed at least one year of M.Sc. studies in Software Engineering (or equivalent)*
2. *The participant has at least three years of experience with Java*
3. *The participant is able to understand and communicate well in English*

*Requirement 1* ensures that the participants not only have knowledge of software engineering corresponding to, at least, a completed bachelor's degree. This is necessary in order to ensure that they have the fundamental understanding required to utilize different software models, but also the knowledge required to model software from source code, which is a substantial task in this experiment.

*Requirement 2*, together with the previous requirement, should ensure that the participant has the necessary knowledge and fundamental understanding of the Java programming language and its functionality to be able to systematically browse and understand the source code of the project they will be tasked with analyzing.

In order to ensure that the training material is thoroughly understood, and that the interviews are able to accurately and as precisely as possible reflect the experiences of the participants, it is necessary for the participant to have a good understanding of the different nuances of the English language. Thus, we conclude with defining *requirement 3* for participant selection.



### 3.3.1.2 Participant profiles

The selected participants both fulfilled the criteria listed in Section 3. Participant A had completed two years of M.Sc. studies, and Participant B was near completion of two years of M.Sc. studies, both at a Computer Science and Engineering division at Chalmers University of Technology. Note that the English language skill entry level requirement for admissions to their respective M.Sc. programs ensures that they fulfill the language requirement for participating in the experiment [33].

Additionally, both had experience working as developers. Participant A had experience with web development. Both participants were well-versed Windows users. Neither of the participants had any prior experience whatsoever with eDFDs, SecDFDs, or Doxygen.

### 3.3.1.3 Training

Identical training sessions on SecDFD properties and extraction were conducted with both participants, which included a practical, hands-on demonstration of SecDFD extraction for a small-scale project both manually and using the tool. In this step the participants performed sub-tasks independently, and were then shown an example solution after which a small discussion was held. Throughout this process, the author would answer any questions. This was done not only in order to ensure that both participants had the necessary knowledge for SecDFD extraction and utilization of the tool, but also that they were on as equal a footing as possible in regard to prior knowledge of the problem. Thus, there was no difference in the content of the training they were given as a base, but they could still come to individually ask questions during the training phase that could lead to them having different extended knowledge bases during the experiment. The details of the content covered during the training session are found in the project repository<sup>2</sup>. The contents of the training session was also provided to the participants as a physical booklet, in order for the participants to have material to consult should they wish to review particular elements of the training during the experiment.

## 3.3.2 Subject of analysis

Two sets of source code, hereon referred to as *SC 1* and *SC 2*, were selected to serve as projects for analysis as part of the experiment.

### 3.3.2.1 Selection

Having two sets of source code, along with the two participants, was done in order to ensure that there would be no instances of confirmation bias during the second SecDFD extraction as an outcome of the learning effect from already having analyzed the project once.

A manual SecDFD-extraction was carried out on both projects to serve as *ground truth* for purposes of analysis. These SecDFDs were not revealed to the participants until they had carried out both the manual- and automatic SecDFD

---

<sup>2</sup><https://github.com/ExtractionTool/ExtractionTool>

extraction they were tasked with, as to not introduce any bias or otherwise compromise the results of the experiment. The ground truths were revealed before the interviews, in order for the participants to be able to assess the correctness of their own SecDFDs.

Additionally, a set of requirements were formulated as criteria for selecting the projects for the experiment phase. By fulfilling these requirements, the two selected sets of source code were considered to be of equal complexity for the intended purpose. *SC 1* and *SC 2* were both selected from a pool of open source projects in the Internet of Things category on GitHub, under the following criteria:

1. *The source code is written in Java*
2. *The project does not consist of more than 1000 SLOC*
3. *The project contains at least one instance of network communication*
4. *The two selected projects should not differ in more than  $\pm 200$  SLOC*
5. *Coding conventions in regard to descriptive naming are followed throughout the project*

*Requirement 1* was given by the fact that the target language for the SecDFD extraction tool is Java. Thus, it was necessary for the selected projects to be written in Java in order for them to be candidates for *both* manual and automatic extraction.

As the participants will not have prior experience with SecDFD extraction, it is important for the project to be of reasonable complexity. This presents us with the need for *requirement 2*. While this size limitation will rule out a great number of commercial applications, it is still possible to find open source projects that do not exceed the size limitation while containing the properties described in the next requirement.

*Requirement 3* is built on utilization of a communication protocol making the project a realistic candidate for SecDFD extraction. This means that the selected application should have implemented functionality for communicating with either a server, or some other networked device. Should this element not be present, the testing of automatic extraction of implemented communication channels will be insufficient.

*Requirement 4* was defined to ensure that the complexity of the chosen projects does not drastically differ. Should the difference in SLOC be too great between the two projects, the difference in complexity between them may affect the results. While 200 SLOC is an arbitrary, small number seen to SLOC, the principle of maintaining a marginal difference in complexity is at the core. This can also be achieved by ensuring that the number of files from project to project does not drastically differ. Since there is no general guideline as to what percentage or what number of SLOC constitute a noticeable difference in complexity, it is left up to the researcher to use their sensibility to thoroughly evaluate if two projects differ drastically, and it is an area where said researcher should tread with great consideration.

*Requirement 5* ensures legibility of the source code, as well as compatibility with the tool. Legibility is a requirement in order to ensure that the participants can easily understand the functionality and context of the different processes, variables, and methods. Should the naming be non-descriptive, additional time-consumption

|                          |                               |
|--------------------------|-------------------------------|
| <b>Description</b>       | A smart plant watering system |
| <b>SLOC</b>              | 631                           |
| <b>Nbr. of processes</b> | 3                             |

**Table 3.1:** SC1 characteristics

|                          |                               |
|--------------------------|-------------------------------|
| <b>Description</b>       | A smart access control system |
| <b>SLOC</b>              | 481                           |
| <b>Nbr. of processes</b> | 5                             |

**Table 3.2:** SC2 characteristics

may become a factor during the manual extraction phase. The tool utilizes a list of keywords in order to automatically identify information assets by finding appearances of the keywords in the source code. This functionality cannot be utilized if variable- and parameter names are non-descriptive, which means that the participant will have to use the manual asset entry functionality (see chapter 4.1.2), which – while still serving the purpose of automatically extracting other elements – severely reduces the point of automatic extraction seeing as the participant will be required to manually extract these assets either way.

The properties of the projects selected for the experiment, including a description of the project, in respect to the criteria mentioned in subsection 3.3.2.1 are defined in tables 3.1 and 3.2 for *SC 1* and *SC 2* respectively. The projects selected as *SC1* and *SC2* were both retrieved from the Intel IoT Devkit’s *How To Code*-repository on GitHub<sup>3</sup>, specifically the Java versions.

The SLOC were retrieved using the IntelliJ IDEA<sup>4</sup> plugin *statistic*<sup>5</sup>.

### 3.3.2.2 Ground truth

In order to determine the correctness of this procedure, a *ground truth* is needed. In this case, a java project coupled with a SecDFD extracted by the author who will act in the place of a domain expert, will constitute as such. For the results to be valid, this project must be omitted from any training or adjustment of the procedure, and only be included in the final testing phase. By executing the novel procedure in its entirety on this project and comparing the final output in regard to the amount of correctly identified SecDFD elements as compared to that produced by the domain expert, we are able to qualitatively assess the performance of the tool.

the ground truths were validated by discussion between the author and supervisors in order to ensure that the ground truths were correct representations of their respective Java projects. The supervisors approving the SecDFDs that had been manually extracted from SC 1 and SC 2 by the author was considered validation of the ground truth.

<sup>3</sup><https://github.com/intel-iot-devkit/how-to-code-samples>

<sup>4</sup><https://www.jetbrains.com/idea/>

<sup>5</sup><https://plugins.jetbrains.com/plugin/4509-statistic>

#### 3.3.3 Execution of experiment

The experiment sessions alternated the different extraction methods, automatic and manual SecDFD extraction, between being the participants' first and second task respectively. This was done to investigate whether the order of the tasks contributed to learning effects, thus affecting the participants' performance during their second task. Additionally, the Java projects under analysis were alternated in order to counteract the risk for confirmation bias due to the possible learning effect from already having analyzed the project once. The experiment procedure for each session is described in Figure 3.1.

##### 3.3.3.1 Manual SecDFD extraction

For the participants to be able to compare usage of the tool to the current means of extraction, i.e. manual extraction, it was necessary to include an element of manual SecDFD extraction in the experiment. Thus, the participants were then tasked with the manual extraction of a SecDFD from a provided set of source code. During this time, the author remained a silent observer, as to not compromise the results of the experiment.

*Participant A* was tasked with manually extracting a SecDFD from *SC 1* as their first task. *Participant B* was tasked with manually extracting a SecDFD from *SC 2* as their second task. The participants were given an upper limit of one hour to complete the task, which was chosen to reflect the size and complexity of both *SC 1* and *SC 2*, and was approximately double the time it had taken to extract the ground truths. The time it had taken to extract the ground truths was doubled in order to account for the fact that the participants were novice to SecDFD extraction. Both participants were timed.

##### 3.3.3.2 Automatic SecDFD extraction

The participants were tasked with using the tool to extract a SecDFD from the set of source code they had *not* performed a manual extraction on. This was done in order to counteract the risk of confirmation bias. Thus, *participant A* was tasked with using the tool to extract a SecDFD from *SC 2*, and *participant B* was tasked with using the tool to extract a SecDFD from *SC 1*. Much like earlier, the author remained a silent observer, the participants were both given an upper time limit of two hours, and were both timed. The one-hour time limit was set in this instance due to the fact that the time-consumption for using the tool should not exceed the time required for manual extraction. Should the time needed to use the tool exceed that of manual extraction, the benefit of using the tool would most likely not be deemed sufficient compared to manual extraction, given that they present equal correctness and effort.

The participants were tasked with carrying out the entire semi-automatic SecDFD extraction process, which consisted of:

1. Building an understanding of the Java project under analysis by reading the documentation and reviewing the source code to the extent they wished

2. Running Doxygen and thus generating the necessary call-graph files
3. Defining the keywords
4. Running the SecDFD extraction tool
5. Re-iterating the process from step 3, i.e. re-defining the keywords and re-executing the tool, if they wished
6. Manually editing the SecDFD they had generated to the extent they wished

The participants were both provided a default list of keywords consisting of: *id*, *key*, *location*, *gps*, *password*, *pwd*, *login*, *uid*, *user*, *device*, *latitude*, *longitude*, *api*, *finance*, *stock*, *listener*. The default list of keywords consisted of arbitrary keywords that had been defined and used by the author during the development and tuning of the tool, and it was explained that the keywords were generic and should be changed or extended. Additionally, the participants were made aware of how the granularity of the keywords affected the end results – i.e., that defining a long and specific string as a keyword would be less likely to reduce in a match and thus may result in an empty SecDFD, and that a short, generic keyword such as a single letter would be likely to result in the tool finding more of what it considers matches, but that these may be false positives.

### 3.3.4 Interviews

Upon completing the task, semi-structured interviews were conducted with each participant. The semi-structured format was selected in order to ensure that the information relevant to the research questions (see chapter 1.2) was obtained, while striving for open-ended interaction in order to maximize the qualitative information gained. The interviews were centered around the following questions:

1. *To what degree do you consider the output from the tool to be correct?*
  - (a) *Did you identify the correct elements?*
  - (b) *Did you identify the correct flows?*
  - (c) *Did you identify the correct assets?*
  - (d) *For each element, flow, and asset, did you identify the correct properties?*
  - (e) *Why do you consider your output correct/incorrect?*
  - (f) *Do you have any general remarks on the correctness?*
2. *How would you compare manual extraction to using the tool?*
  - (a) *Did you experience any challenges with manual extraction, and if so, what were they?*
    - i. *Were the challenges regarding...*
      - A. *Usability and understanding?*
      - B. *Other?*
    - ii. *Did the challenges affect...*
      - A. *Time consumption?*
      - B. *Correctness?*
      - C. *Other?*
  - (b) *Did you experience any challenges with using the extraction tool, and if so, what were they?*

- i. Were the challenges regarding...*
    - A. Usability and understanding?*
    - B. Other?*
  - ii. Did the challenges affect...*
    - A. Time consumption?*
    - B. Correctness?*
    - C. Other?*
- (c) Do you have any general remarks on comparison?*

The questions were selected in consideration of eliciting a conclusive and unbiased response in relation to the research questions as listed in chapter 1.2. The participants were asked about their own performance for both the manual extraction and when using the tool, given the fact that both cases enabled them to make manual alterations, and relied on their own individual choices and interpretations. Thus, interviewing them about their own correctness even when discussing the tool enables reflection and provides information regarding the entire semi-automatic extraction process, rather than limiting it to the step in which the tool is executed.

## 3.4 Analysis

A qualitative analysis was conducted on each of the SecDFDs produced as part of the experiment, against their respective ground truths which were, as previously mentioned, extracted by the author before the course of the experiment. Additionally, the responses obtained during the interviews were taken into consideration, as well as the time-consumption for each of the participants to complete each task.

### 3.4.1 SecDFD comparison

The analysis of SecDFDs to measure the correctness of the automatically generated instance was done as such that only *assets*, *flows* and *processes* were subject to evaluation. This was due to the fact that the tool only supports the extraction of these two categories of SecDFD elements.

For an asset to be considered correctly identified, the asset in the automatically extracted SecDFD had to be 1) logically equivalent to an asset in the ground truth, meaning that it serves the same purpose and contains the same information, and 2) originates from the same source, or a logically equivalent process.

Two processes were considered correctly identified if 1) they are logically equivalent, i.e. they serve the same purpose and represent the same running code in the source project, and 2) are attached to the same flows in matching directions. The second criteria also serves the purpose of evaluating the correctness of the identified flows. The flows are evaluated by creating *adjacency matrices* for both the ground truth and automatically extracted SecDFD, with their respective processes on the edges of the corresponding matrix. If a flow goes from one process to another, it is denoted by a '1', whereas an absence of a flow is denoted by '0'. If the automatically extracted SecDFD has the same flows as the ground truth, and the processes con-

nected to the flows are logically equivalent to the ground truth, the processes tied to the flows are considered completely correct.

Throughout this process, I considered the appearances of:

- **True Positives:** *elements that appear in both the ground truth and the automatically extracted SecDFD*
- **False Positives:** *elements that appear in the automatically extracted SecDFD, but not the ground truth*
- **False Negatives:** *elements that do not appear in the automatically extracted SecDFD, but do appear in the ground truth*

A single element does not necessarily map to another single element from one SecDFD to another. Rather, it is possible for two or more elements in the automatically extracted SecDFD to map to a single element in the ground truth and vice versa. Therefore, the rates mentioned above were investigated based on what was covered in the ground truth – i.e., if two elements mapped to a single element in the ground truth, those two elements were considered as a single correctly identified element.

### 3.4.2 Interviews

The interviews that were held with each participant after each experiment session were analyzed by reviewing the recorded material and creating a transcript of the interviews, which was followed by an approach similar to the inductive reasoning and theme identification described in section 2.2.3. This was performed by qualitatively evaluating the material, including notes and recordings, from the interviews and identifying the quotes that stood in relation to the research questions by being a direct response to the corresponding interview questions as found in Section 3.3.4. This was done in order to summarize the key notes in the report in such a way that the participants remain anonymous. The key findings were then summarized into a general description of the participants' respective feedback, which can be found in section 4.3.2.





# 4

## Results

In this chapter I will describe the results of the development of the extraction tool, the black box tests, and the experiments, as described in section 3. The results of the evaluation of the experiments are divided into a correctness analysis of the SecDFDs produced by participants A and B, and a presentation of their responses to the two methods: manual- and automatic SecDFD extraction. For the black box testing, I will present how many of the projects available for this phase were compatible with the automatic SecDFD extraction method, a presentation of the number of calls for each of the projects, and a correctness analysis of the automatically extracted SecDFDs, as per the methods described in section 3.4.1.

### 4.1 SecDFD extraction tool

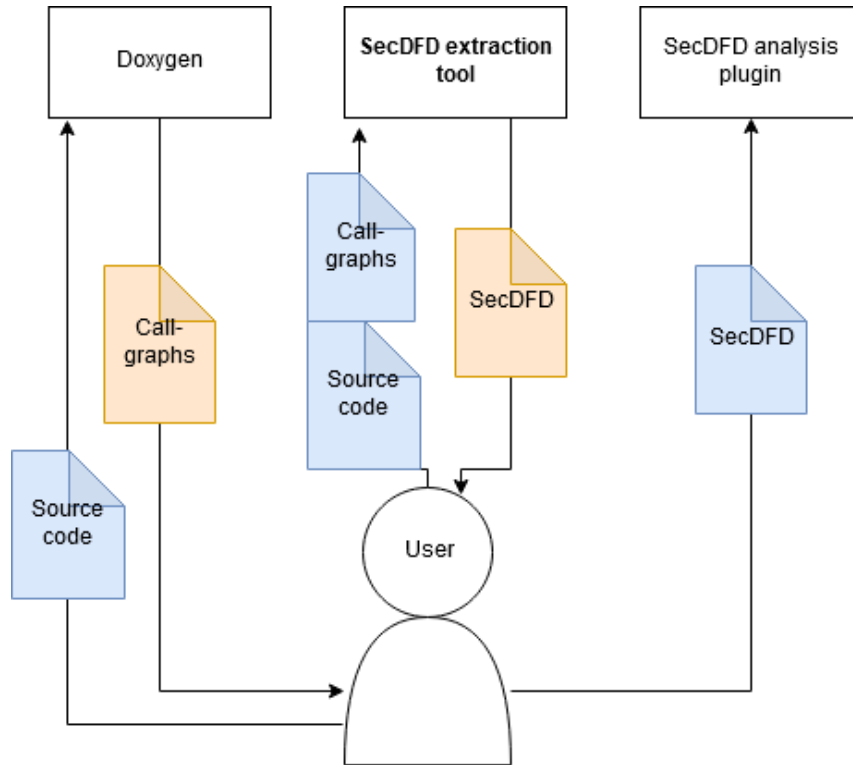
This section details the implementation choices and internal functionality of the SecDFD extraction tool, including the requirements for the tool and the third party dependencies. The SecDFD extraction tool's source code can be found in the ExtractionTool project repository<sup>1</sup>.

#### 4.1.1 Tool development

The tool relies on access to 1) the source code under analysis, 2) a call-graph extracted from the source code under analysis represented in the DOT graph description language, which is an abstract language part of the open source graph visualization software, GraphViz [19][18], and 3) a domain specific library containing keywords commonly related to information assets. The extraction of such files was achieved by using Doxygen which extracts documentation from source code, being the de facto standard tool for that purpose for annotated C++ sources but providing support for other programming languages, among those Java [10]. The design of the intended program flow from source code to fully extracted can be found in Figure 4.1. This illustrates the flow of extracting a SecDFD from information given by the call-graph and source code, which is then given to the SecDFD analysis plugin (analysis tool) for security analysis. The tool described in this thesis is allocated in the module titled “SecDFD extraction tool” in the figure, which performs the SecDFD extraction. In the given context, the SecDFD extraction tool provides the mid-tier functionality of extracting the SecDFD with the aforementioned assets provided.

---

<sup>1</sup><https://github.com/ExtractionTool/ExtractionTool>



**Figure 4.1:** The process of the semi-automatic SecDFD extraction procedure from the user's perspective.

The requirements for the SecDFD extraction tool can be found in Table 4.1. These were elicited mainly by discussion between the author and the supervisors, determining what elements were part of the desired outcome as well as what would be necessary for the internal functionality of the tool to best mimic the process of manual expert SecDFD extraction.

R1 was specified to ensure that the minimum necessary components to make up an extended data flow would be present. R1.1 was specified per recommendation that a call graph would contain the information about the flow of data that is necessary in order to, in combination with confirming that the data on the flow holds security relevance, extract a SecDFD. For further detail, see section 4.1.2. R2 was stated as a requirement due to the limitations and scope of the project. R3 was formulated to ensure that the tool produced output in a format that could be used for SecDFD security analysis. R4 was formulated as one of the main trials of the project, which was to discern if security relevant data can be identified by naming and string matching.

### 4.1.2 Implementation choices

This section details the implementation choices of the SecDFD-extraction tool, as well as the contextual use of the tool. The SecDFD extraction tool was written in Python<sup>2</sup>, version 3.7.

---

<sup>2</sup><https://www.python.org/>

|             |  |
|-------------|--|
| <b>R1</b>   | The tool shall identify the assets, flows, and processes of an application based on the data flows within the application. |
| <b>R1.1</b> | The tool will use call graphs in complement with the program code to identify the data flows.                              |
| <b>R2</b>   | The tool shall be executable on projects written in Java.  |
| <b>R3</b>   | The tool shall produce a SecDFD in the SecDFD specification language as output.  |
| <b>R4</b>   | The tool shall use a set of keywords used for identifying potential assets.  |

**Table 4.1:** Requirements for the SecDFD extraction tool

| Tab   | Wizard                  |   |           |                         | Expert              |
|-------|-------------------------|---|-----------|-------------------------|---------------------|
| Topic | Project                 | Mode                                    | Output    | Diagrams                | Dot                 |
|       | Scan recursively = True | Extraction mode: all entities = True    | [Default] | Use Dot = True          | DOT_CLEANUP = False |
|       |                         | Programming language: Java or C# = True |           | Call graphs = True      |                     |
|       |                         |   |           | Called-by graphs = True |                     |

**Table 4.2:** Settings used in Doxygen to produce call-graphs in the format required by the SecDFD extraction tool

The tool was designed to rely on the call-graph files produced by Doxygen, which uses the DOT graph specification language from Graphviz. These were both installed on the device used throughout the testing. The device in question ran the Windows 10 operating system. The Doxygen version used was 1.8.15. The Graphviz version used was 2.38.

The output required by the tool to function was produced by Doxygen with the settings specified in Table 4.2 in the Doxygen GUI for Windows, Doxywizard, given that the user had provided the correct paths to the Doxygen running directory and source code directory for the application under analysis. The table divides the settings according to the tabs they are located in, as well as which topic the setting is located under, as defined by the Doxygen Doxywizard GUI. An additional measure was to specify a path to the output directory, since the path to the Doxygen output was a required input for the tool.

The automatic SecDFD extraction process, as referred to in this manner throughout the thesis, is used to describe the process of using the SecDFD extraction tool to extract a SecDFD in its entirety. This includes generating the necessary third party files, i.e. the call-graph files generated by Doxygen, specifying keywords, running the SecDFD extraction tool for however many iterations one wishes, and manually editing the output file.

### 4.1.3 Tool description

The SecDFD extraction tool consists of the main script, which executes at runtime, and utilizes the custom models that have been implemented to act as models for the assets and elements of a SecDFD – acting as translators between the call-graph

files, the project source code, and the end-model.

### 4.1.3.1 Required input

At the beginning of the run, the tool prompts for the following input from the user:

- *The path to the directory where the source code of the application the user wishes to extract a SecDFD from*
- *The path to the directory where the call graph files for the above mentioned application are kept*
- *The desired destination path for the file the tool will output*
- *The desired name for the SecDFD that the tool will generate*

### 4.1.3.2 Functionality

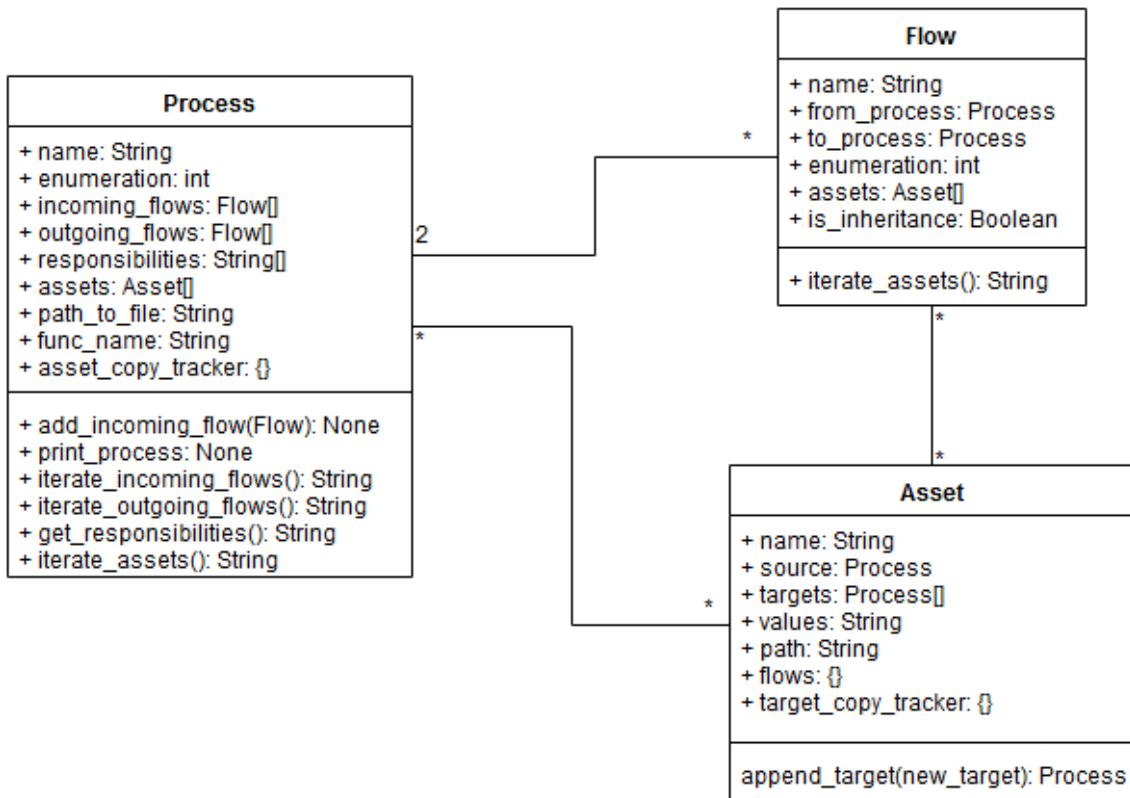
The SecDFD extraction tool tracks all communications as identified by the Doxygen call graphs by aggregating all the call graph files into one parseable string, searching all files that match the path given in the call graph node labels. It attempts to match the entire list of keywords against the method- and parameter names in the source code files, giving a positive match if any of the keywords are found to be substrings of the method- or parameter names. The SecDFD extraction tool uses a regular expression to find the declaration of a method, and identify the parameters of said method. This results in the program adding a new asset, with the asset's source process being the SecDFD process that contains the file in which the asset was found, to the list of assets. The values of the asset before manual editing will always be "High Confidentiality", which has been hard-coded into the asset generation seeing as an asset value is required by the SecDFD specification language, but automatically identifying asset values is not yet supported by the tool. The hard-coded value is an accepted value by the specification language. The asset is given a compound name in order to reflect its character and place of origin:

$$assetname = capitalize(functionname) + capitalize(parametername)$$

Similar to the asset naming procedure, processes and flows are named with an aggregation of their location in the code.

Additionally, a positive match for an asset is the foundation for identifying a flow. A flow is only added to the list of flows if it transports an identified asset, and a process is only added to the list of processes if it is attached to a security relevant flow. The tool builds a SecDFD by only adding relevant assets and elements, rather than adding everything and narrowing the SecDFD down as a separate process.

The tool is designed not to list duplicate processes, flows, or assets. It does so by maintaining "copy trackers" for each relevant instance, which basically create rudimentary hashes of the paths, class names, function names, and in the case of assets parameter names, of each of the already identified components. Before a new component is added, it is first hashed and checked against the copy tracker. Should it represent an identical component, any newly found information is bundled in with the already registered version of the component. This, together with the conditional



**Figure 4.2:** A domain model of the SecDFD extraction tool's core elements.

adding of elements to the final SecDFD, creates a natural process-bundling effect by only breaking out processes when their assets and responsibilities are unique relative the interconnected processes.

Each component – processes, flows, and assets – is modelled as an object and maintains instances of relations and information accounting their origin and characteristics, which are used for pairing them, identifying their origin in the code, and gathers the required SecDFD characteristics into one instance from the different information sources parsed by the tool. Figure 4.2 illustrates these core components and their relationships.

Since Doxygen provides unique nodes in the call graphs with unique enumerations, a matrix where processes are placed on the address of their index is used to maintain the list of identified processes. This matrix may be considered a debased variation of a hashmap.

The tool uses this procedure to attempt extraction of the following information:

- *Assets:*
  - *Source*
- *Elements:*
  - *Processes:*
    - \* *Assets*
    - \* *Responsibilities:*
      - *Store*

- *Forward*
- \* *Incoming flows*
- \* *Outgoing flows:*
  - *Unique enumeration*
  - *Assets*
  - *Targets*

### 4.1.3.3 Output format

The tool outputs a .mydsl file in the specified destination directory, containing all positively identified SecDFD components according to the procedure mentioned in Section 4.1.3.2 given in the syntax of SecDFD specification language.

## 4.2 Evaluation by black box testing

The black box testing was carried out according to the specification given in Section 3.2. The projects that were available during this phase are specified in Table 4.3.

| Index | Project                      |
|-------|------------------------------|
| C1    | COAP IoT Server <sup>a</sup> |
| C2    | DroidBench <sup>b</sup>      |
| C3    | FriendMap <sup>c</sup>       |
| C4    | Hospital <sup>d</sup>        |
| C5    | JPMail <sup>e</sup>          |
| C6    | WebRTC <sup>f</sup>          |
| C7    | ATM Simulator <sup>g</sup>   |
| C8    | CoCoMe <sup>h</sup>          |
| C9    | iTrust <sup>i</sup>          |
| C10   | JPetStore <sup>j</sup>       |
| C11   | SecureStorage <sup>k</sup>   |

**Table 4.3:** A list of the projects used for the black box testing.

<sup>a</sup><https://github.com/Mozilla9/coap-iot-server>

<sup>b</sup><https://github.com/secure-software-engineering/DroidBench>

<sup>c</sup><https://github.com/apl-cornell/fabric/tree/master/examples/friendmap>

<sup>d</sup><https://github.com/apl-cornell/fabric/tree/master/examples/hospital>

<sup>e</sup><http://siis.cse.psu.edu/jpmail/>

<sup>f</sup><https://github.com/webrtc/apprtc>

<sup>g</sup><http://www.math-cs.gordon.edu/local/courses/cs211/ATMExample/>

<sup>h</sup><https://github.com/cocome-community-case-study>

<sup>i</sup><https://sourceforge.net/projects/itrust/>

<sup>j</sup><http://www.mybatis.org/jpetstore-6/>

<sup>k</sup><https://www.eclipse.org/equinox/security/>

Code: <http://www.java2s.com/Code/Jar/o/Downloadorgeclipseequinoxsecurityjar.htm>

| Project | Positive, Doxygen | Positive, Extraction Tool |
|---------|-------------------|---------------------------|
| C1      | Yes               | Yes                       |
| C2      | Yes               | Yes                       |
| C3      | No                | -                         |
| C4      | No                | -                         |
| C5      | Yes               | No                        |
| C6      | Yes               | No                        |
| C7      | Yes               | No                        |
| C8      | Yes               | Yes                       |
| C9      | Yes               | No                        |
| C10     | Yes               | Yes                       |
| C11     | No                | -                         |

**Table 4.4:** Illustration of whether or not each project produced the expected files when running Doxygen, and produced a non-empty SecDFD when running the extraction tool.

### 4.2.1 Tool Compatibility

Table 4.4 illustrates whether Doxygen produced call-graphs for each of the projects, and whether the extraction tool produced a non-empty SecDFD. In Table 4.5, the projects for which call-graph files were correctly generated but where the extraction tool failed to generate a non-empty SecDFD are paired with the direct explanation for the failure. This correlates to the fact that the projects used for this phase were selected on an availability basis, following the explanation in Section 3.2, and thus did not necessarily meet all of the compatibility requirements of the tool, such as being written in Java.

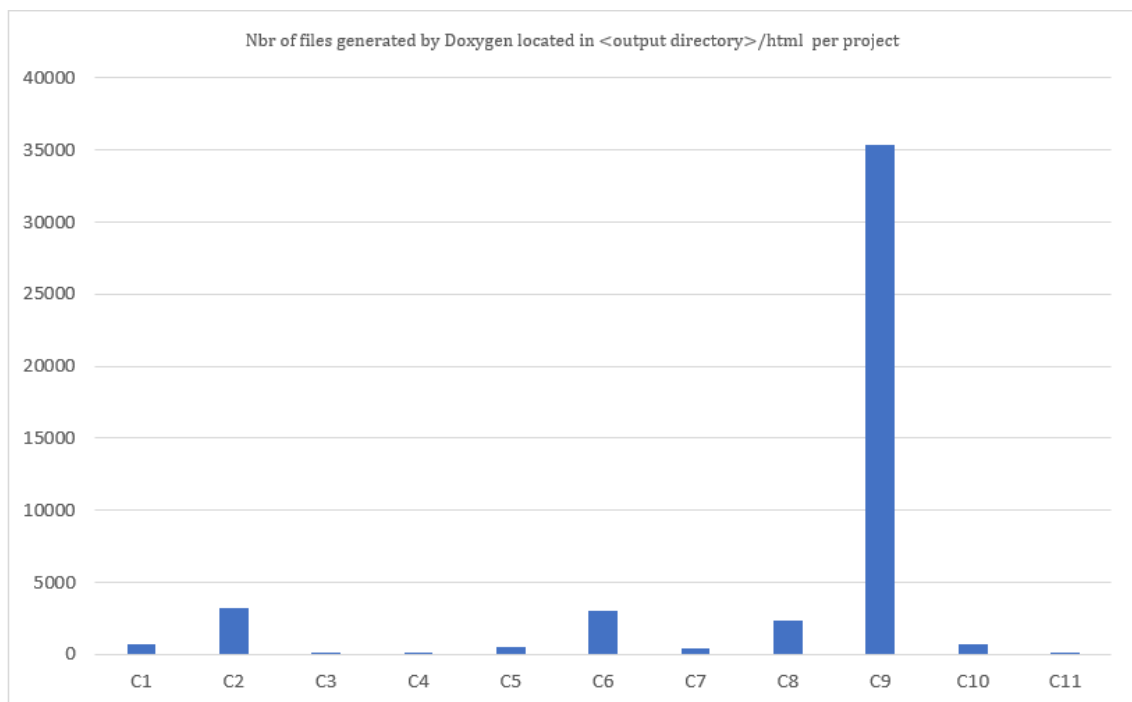
As made visible by Table 4.4, the total number of projects for which no non-empty SecDFD was produced was 7 out of the 11 available projects, resulting in a failure rate of roughly 63.6%. However, 3 of these failures were due to Doxygen failing to generate call-graphs from the projects. These projects are omitted from Table 4.5 due to the cause of failure being on a third party rather than the extraction tool. Subtracting the failures that occurred due to Doxygen not being able to process the projects in the expected way, 4 out of the 8 remaining projects failed due to the extraction tool not producing a non-empty SecDFD with the given keywords. The keywords were based on the names of the assets in the respective ground truths, and were determined according to the process defined in 3.2. This means that the pure failure rate is at 50%.

### 4.2.2 Tool Correctness

This section describes the correctness evaluation of the projects for which a non-empty SecDFD could be extracted by applying the automatic SecDFD extraction process. For the extracted assets in the comparative tables of manually extracted SecDFDs and their automatically extracted versions, a star annotation next to the automatically extracted asset in the table represents an asset that is logically equivalent to its counterpart in the ground truth as per content and purpose, but does

| Proj. | Explanation  |
|-------|--|
| C5    | Keywords did not match. Confirmed by re-executing with general keywords, which produced a non-empty SecDFD.  |
| C6    | Suspected semantic mismatch as the project is not written in Java.   |
| C7    | Suspected package-structure mismatch. The package structure expected by the extraction tool, based on the Doxygen output, did not match the file structure of the running directory since the original directory structure was not available for download.   |
| C9    | Error on host machine when executing tool on project C9. It is suspected that this is correlated to the fact that the files generated by Doxygen for project C9, that in turn needed to be parsed by the tool, were too many for the tool to compute within reasonable complexity, as illustrated in Figure 4.3. |

**Table 4.5:** Reason for negative result (empty SecDFD) in cases where Doxygen produced call-graph files, when applying the “open box” keyword determination method.



**Figure 4.3:** The number of files generated by Doxygen per project used in the black box testing phase.



not originate from the logically equivalent source in the ground truth.

The tool correctness is evaluated using the SecDFD comparison method as defined in Section 3.4.1, where logically equivalent processes and assets are identified, and adjacency matrices are used to illustrate the flows between processes. As described in Section 3.4.1, the processes are located on the edges of the matrix, and a ‘1’ denotes the existence of a flow in the given direction between the processes.

Out of the projects subjected to the black box testing, which produced non-empty SecDFDs after a single iteration of keywords based on their ground truths, the automatically extracted SecDFD for C1 proved the most correct. The tool identified 50% of the logically equivalent assets, where 2/3 were confirmed to have originated from the same process. Additionally, 2/3 processes from the ground truth were represented by their logically equivalent counterpart in the automatically extracted SecDFD. Apart from C1, the automatically extracted SecDFD for C10 was considered more correct than both C2 and C8, out of which the automatic extraction for C8 performed better. Regarding C10, the automatically extracted SecDFD identified 30% of the logically equivalent assets, and 40% of the logically equivalent processes.

### 4.2.3 C1

The manually extracted SecDFD of the CoAP IoT Server contained 6 assets and 3 processes, whereas the automatically extracted SecDFD contained 8 assets and 8 processes. Of these, 3 automatically extracted assets corresponded to 3 assets in the ground truth, where 2 of the automatically extracted assets originated from the logically equivalent process in the ground truth. The number of logically equivalent processes was 2 to 2. The logically equivalent SecDFD components for C1 are listed in Table 4.7. In the table you can see that over half of the processes from the ground truth were identified, as well as half of the assets.

By observing the adjacency matrices of the ground truth and automatically extracted SecDFD, observable in Tables 4.6 and 4.8 respectively, we find that there are no identical flows when only observing process based communication.

It is noted that the processes representing Data Access objects (DAOs) correspond to the Data Stores of the ground truth.

### 4.2.4 C2

The DroidBench project analysis was inconclusive, as the general names and characteristics of the assets and processes between the 4 activities making up the manually extracted SecDFD were not semantically represented in the source code, which created a disconnect between the ground truth, the source code, and the automatically extracted SecDFD. Such an occurrence is most likely due to liberties and individual interpretations taken during the manual SecDFD extraction. Additionally, the character of the manually extracted SecDFDs keywords such as *a*, *b*, and *String* resulted in a high number of false positives in the automatically extracted SecDFD. The automatically extracted SecDFD contained 20 assets (as opposed to 6 unique assets by name across the 4 activities making up the manually extracted SecDFD),

|                       | SessionHolder | DeviceDataMapper | LoginResource | FirmwareDataMapper | DeviceConfigDAO | ParserHelper | ConfigurationResource | DeviceDataDAO |
|-----------------------|---------------|------------------|---------------|--------------------|-----------------|--------------|-----------------------|---------------|
| SessionHolder         | 0             | 0                | 0             | 1                  | 0               | 0            | 0                     | 0             |
| DeviceDataMapper      | 0             | 0                | 0             | 0                  | 0               | 0            | 0                     | 0             |
| LoginResource         | 0             | 0                | 0             | 1                  | 0               | 0            | 0                     | 0             |
| FirmwareDataMapper    | 0             | 0                | 0             | 0                  | 0               | 0            | 0                     | 0             |
| DeviceConfigDAO       | 0             | 0                | 0             | 0                  | 0               | 0            | 0                     | 0             |
| ParserHelper          | 0             | 0                | 0             | 0                  | 1               | 0            | 0                     | 0             |
| ConfigurationResource | 0             | 0                | 0             | 1                  | 0               | 0            | 0                     | 0             |
| DeviceDataDAO         | 0             | 1                | 0             | 1                  | 0               | 0            | 0                     | 0             |

**Table 4.6:** Adjacency matrix for the SecDFD automatically extracted from the CoAP IoT Server (C1).

| Manual             | Automatic          |
|--------------------|--------------------|
| Assets             |                    |
| logdata            | -                  |
| devicedata         | updateDevicedata*  |
| newconfiguration   | insertDeviceConfig |
| coapsessiontoken   | getSessionToken    |
| deviceowner        | -                  |
| deviceserialnumber | -                  |
| Processes          |                    |
| ClientApp          | ParserHelper       |
| CoAPServerApp      | -                  |
| Authenticate       | SessionHolder      |

**Table 4.7:** Logical appearances of elements from Ground Truth in automatically extracted SecDFD for the CoAP IoT Server (C1).

|               | ClientApp | CoAPServerApp | Authenticate |
|---------------|-----------|---------------|--------------|
| ClientApp     | 0         | 1             | 1            |
| CoAPServerApp | 1         | 0             | 0            |
| Authenticate  | 1         | 0             | 0            |

**Table 4.8:** Adjacency matrix for the SecDFD manually extracted from the CoAP IoT Server project.

and 15 processes.

#### 4.2.5 C8

The results from the CoCoMe project analysis showed that the automatic extraction identified a subset of the SecDFD components from the ground truth, but not necessarily in the same component category, meaning that a component listed as a Data Store or Exerternal Entity in the ground truth was identified as a Process by the extraction tool. The keywords did not correspond to any logically equivalent instance in the paths and files as determined by the call-graphs generated by Doxygen, leading to a disjoint set of assets and processes between the manually- and automatically extracted SecDFDs. It is however worth noting that the identified processes, IStoreQuery and RMI, in this case correspond to a Data Store and External Entity respectively. The manually extracted SecDFD contains 7 assets and 7 processes, whereas the automatically extracted SecDFD contains 5 assets and 2 processes. No instance could be directly mapped.

#### 4.2.6 C10

There is no inter-process communication in the manually extracted SecDFD of JPetStore, as all communications are directly to or from other element types (External Entities and Data Stores), meaning that it is not a candidate for comparison by adjacency matrix.

The manually extracted SecDFD of JPetStore contained 10 assets and 5 processes. The automatically extracted SecDFD of JPetStore contained 8 assets and 8 processes, where a total of 4 assets from the automatically extracted SecDFD mapped to 3 of the assets in the manually extracted SecDFD, and a total of 3 processes mapped from the automatically extracted SecDFD to 2 processes in the manually extracted SecDFD. The logical equivalences mentioned are given by Table 4.9. The rate of affirmed logical matches is notable. However, the processes and thus flows could not be positively affirmed as correct due to the lack of pure inter-process communication in the ground truth.

### 4.3 Evaluation by Experiment

This section details the results from the experiments. The experiment participants were both consistent in the degree of correctness of the SecDFDs they themselves produced whether it was using the manual- or semi-automatic SecDFD extraction method, while the degree of correctness varied between the participants. Both participants experienced less time consumption when using the tool, and were in the end partial towards the semi-automatic extraction process.

#### 4.3.1 Experiment results

Descriptions of the two applications used as subjects for analysis, labelled SC1 and SC2, are given in Table 3.1 and 3.2 respectively.

|                 | Manual | Automatic                |
|-----------------|--------|--------------------------|
| Assets          |        |                          |
| product         |        | -                        |
| productID       |        | -                        |
| itemID          |        | -                        |
| item            |        | setItemItem, addItemItem |
| account_info    |        | initOrderAccount*        |
| username        |        | -                        |
| password        |        | -                        |
| keyword         |        | -                        |
| order           |        | getOrderOrder            |
| cart            |        | -                        |
| Processes       |        |                          |
| Authenticate    |        | -                        |
| Search_product  |        | -                        |
| Purchase_order  |        | OrderService, Order      |
| AddRemove_Items |        | Cart                     |
| Manage_account  |        | -                        |

**Table 4.9:** Logical appearances of elements from Ground Truth in automatically extracted SecDFD for JPetStore (C10).

| Method of extraction                         | Project | Time            |
|--|---------|-----------------|
| Manual extraction                            | SC1     | 57 min. 52 sec. |
| Extraction using tool                        | SC2     | 12 min. 28 sec. |
| Extraction using tool (incl. manual editing) | SC2     | 29 min. 39 sec. |

**Table 4.10:** Time consumption for SecDFD-extraction using the two different extraction methods, Session A

Both participants were able to complete both the manual and automatic SecDFD extraction procedure, to the point where they considered themselves to have completed the task. At the point of perceived completion, the timer was stopped. In the case of automatic extraction, a partial time was noted when the usage of the tool was considered completed after  $n$  iterations of re-specifying keywords and subsequently re-executing the tool, and when the manual editing commenced. The times for Participant A can be found in table 4.10, and the times for Participant B can be found in table 4.11. Note that the time for automatic extraction for Participant B was affected by a sporadic event, which is further elaborated by the footnote given in the table.

During the automatic extraction procedure, Doxygen generated **206** files for SC1, and **157** files for SC2.

#### 4.3.1.1 Correctness

This Section describes the correctness evaluation of the manual- and automatic extraction of SecDFDs by the experiment participants for both SC1 and SC2. Par-

| Method of extraction                         | Project | Time                 |
|--|---------|----------------------|
| Manual extraction                            | SC2     | 59 min. 15 sec.      |
| Extraction using tool                        | SC1     | 48 min. <sup>a</sup> |
| Extraction using tool (incl. manual editing) | SC1     | 52 min. 41 sec.      |

<sup>a</sup>This time was affected by unpredictable issues with Doxygen, which resulted in call-graph files not being generated. This was discovered at the 32 minute mark, after which the author intervened and ran Doxygen on the same project, with the same settings as participant B, in a different system location. It is possible that this, in turn, came to affect the correctness of the SecDFD produced by participant B due to them not performing optimally due to the frustration.

**Table 4.11:** Time consumption for SecDFD-extraction using the two different extraction methods, Session B

| Ground Truth       | Session A                    | Session B (Extraction Tool)                                    |
|--------------------|------------------------------|--|
| Assets             |                              |  |
| PortNbr            | AzureServerUrl               | -  |
| AllLoggedValues    | TwilioMessage*, AzureMessage | SendMessageWithTwilioString,<br>SendMessageWithTwilio-<br>Body |
| CurrentMoisture    | TwilioMessage*, AzureMessage | SendMessageWithTwilioString,<br>SendMessageWithTwilio-<br>Body |
| Processes          |                              |  |
| Server             | TwilioApi                    | Utils  |
| MoistureController | -                            | -  |
| WateringSystem     | WateringSystem               | -  |

**Table 4.12:** Logical connections for the Watering System (SC1).

Participant A had a high degree of correctness using both manual- and semi-automatic extraction. Participant B produced a SecDFD with a greater degree of correctness using semi-automatic extraction as compared to manual extraction. The degree of correctness between the sessions and the projects used in the experiment indicate that the correctness changed between the participants, rather than between the projects, meaning that the correctness depended on the participant rather than which project they were analyzing.

**4.3.1.1.1 SC1** The logical equivalences for SC1 in relation to the ground truth, from both sessions, are given by Table 4.12.

|                    | Server | MoistureController | WateringSystem |
|--------------------|--------|--------------------|----------------|
| Server             | 0      | 0                  | 0              |
| MoistureController | 1      | 0                  | 0              |
| WateringSystem     | 1      | 0                  | 0              |

**Table 4.13:** Adjacency matrix for the Ground Truth of the Watering System (SC1).

#### 4. Results

|                | TwilioApi | HTTPConnection | WateringSystem |
|----------------|-----------|----------------|----------------|
| TwilioApi      | 0         | 0              | 1              |
| HTTPConnection | 0         | 0              | 0              |
| WateringSystem | 1         | 1              | 0              |

**Table 4.14:** Adjacency matrix from session A for the Watering System (SC1).

|            | Utils | FlowSensor |
|------------|-------|------------|
| Utils      | 0     | 1          |
| FlowSensor | 1     | 0          |

**Table 4.15:** Adjacency matrix from session B for the Watering System (SC1).

Participant A produced 9 assets and 3 processes for SC1 by manual extraction, where a total of 3 assets mapped to 3 assets in the ground truth – however, two of these were aggregated. 2 processes were logically equivalent to 2 processes in the ground truth, and while it can be seen in the adjacency matrices for the ground truth (Table 4.13) and the one from the SecDFD manually extracted by participant A (Table 4.14) that they both contain communication between the logically equivalent representations of the Server and the Watering System, the communications are in reverse direction. Therefore, it cannot be considered a complete match.

Participant B produced 8 assets and 2 processes for SC1 by automatic extraction, where a total of 2 assets mapped to 2 assets in the ground truth – however, the two assets from the automatically extracted SecDFD were aggregates, and the aggregation mapped to both assets in the ground truth. 1 process mapped to 1 process in the ground truth – however, based on the adjacency matrix for the automatically extracted SecDFD from session B as seen in Table 4.15, we cannot state that the communications match as the extracted SecDFD only includes a single logically equivalent process in relation to the ground truth.

**4.3.1.1.2 SC2** The logical equivalences for SC2 in relation the ground truth, from both sessions, are given by Table 4.16.

Participant A produced a total of 3 assets and 5 process by automatic SecDFD

| Ground Truth      | Session A (Extraction Tool)            | Session B              |
|-------------------|--|------------------------|
| Assets            |  |                        |
| MotionActivity    | -                                      | -                      |
| PortNbr           | SetupServerPort*                       | -                      |
| DisplayMessage    | NotifyAzureMessage, WriteMessageString | Message                |
| Processes         |  |                        |
| Server            | Utils, ServerSetup                     | -                      |
| AccessControl     | AccessControl                          | -                      |
| MotionSensor      | -                                      | MotionActivityHandling |
| LcdDisplayMessage | Azure, LcdSensor                       | ScreenAcitvty          |

**Table 4.16:** Logical connections for the Access Control system (SC2).

|                   | Server | AccessControl | MotionSensor | LcdDisplayMessage |
|-------------------|--------|---------------|--------------|-------------------|
| Server            | 0      | 0             | 0            | 0                 |
| AccessControl     | 1      | 0             | 0            | 0                 |
| MotionSensor      | 0      | 1             | 0            | 1                 |
| LcdDisplayMessage | 0      | 0             | 0            | 0                 |

**Table 4.17:** Adjacency matrix for the Ground Truth of the Access Control system (SC2).

|               | AccessControl | Utils | ServerSetup | Azure | LcdSensor |
|---------------|---------------|-------|-------------|-------|-----------|
| AccessControl | 0             | 0     | 0           | 0     | 0         |
| Utils         | 0             | 0     | 1           | 0     | 0         |
| ServerSetup   | 1             | 0     | 0           | 0     | 0         |
| Azure         | 0             | 0     | 0           | 0     | 0         |
| LcdSensor     | 0             | 1     | 0           | 0     | 0         |

**Table 4.18:** Adjacency matrix from session A for the Access Control system (SC2).

extraction. From these, a total of 3 assets map to 2 assets in the ground truth, and a total of 5 processes map to 3 processes in the ground truth. By observing the adjacency matrix for the SecDFD of SC2 produced by Participant A via the automatic SecDFD extraction process as seen in Table 4.18, and comparing it to that of the ground truth as seen in Table 4.17, we can see that it contains the communication between the logically equivalent processes for the LCD Display, Motion Sensor, and Access Control when considering the logical bundling – however, these are all in the reverse directions, and can thus not be considered correct.

Participant B produced a total of 1 asset and 2 processes via the manual SecDFD extraction process for SC2. Of these, a total of 1 asset mapped to 1 asset in the ground truth, and 2 processes mapped to 2 processes in the ground truth respectively. By observing the adjacency matrix for Participant B’s manually extracted SecDFD of SC2 as seen in Table 4.19, we can see that it contains the communication between the Motion Sensor and LCD Display as in the ground truth – however, it is in the reverse direction, and can thus not be considered correct. It is not certain whether the reversed flows are a product of the tool, or the participants’ interaction with the tool and the produced SecDFD. Further study is needed to determine the cause of the flow reversal.

|                        | MotionActivityHandling | ScreenActivity |
|------------------------|------------------------|----------------|
| MotionActivityHandling | 0                      | 0              |
| ScreenActivity         | 1                      | 0              |

**Table 4.19:** Adjacency matrix from session B for the Access Control system (SC2).

### 4.3.2 Interviews

This section compiles the results from the interviews held after the completion of each experiment session with the respective participant. Note that the opinions given are from the participants interview statements, with the reflections on the correctness of each approach being based on their *perceived* correctness of their results. For the correctness evaluation by SecDFD analysis, see section 4.3.1.1.

#### 4.3.2.1 Participant A

Participant A performed manual SecDFD extraction on SC1 first, and automatic SecDFD extraction on SC2 subsequently.

**4.3.2.1.1 Automatic extraction** Participant A considers the correctness of the automatic SecDFD extraction process to be of medium correctness according to their own words, pointing out that they achieved about the same level of correctness when using the automatic extraction process as they did when performing manual extraction, noting that while the results from the manual- and semi-automatic extraction were of seemingly equal correctness, the automatic extraction process, including the manual editing, required about half the time. They remarked that they believed the degree of incorrectness of their automatically extracted SecDFD should be accredited to their lack of expertise in the subject rather than the performance of the tool.

They also noted that their frame of reference for considering their solutions correct was that they considered the components similar to the ground truth they were shown to be correct, and those that differed too greatly to be incorrect. They said that the components they considered to be incorrect in their solutions were the ones that were omitted, and pointed out that they had forgotten to include or consider them.

They mentioned that, given the chance to repeat the process, they would add one or two more keywords to the set of keywords in order to achieve a more correct solution.

They also noted that they felt that the experienced time pressure affected the end result, and that it would most likely have been easier to be two or more people working together on the task.

The participant felt that using the tool lessened their understanding of the code base – they remarked that when performing the manual extraction, they had to read through and comprehend the code in search of the different assets, which meant they had to understand the role of said assets and other components of the program. However, when using the tool, they felt that they received a list of assets that could be useful, and searched in the code for the assets they had received, rather than having to find assets by comprehending the program.

They felt that their perceived challenges of using the tool mainly affected the correctness of their end result, due to the fact that they trusted the tool too much when it came to producing the correct output. Time consumption was also added to some degree, as they performed quite a lot of manual editing.



Additionally, they believe that it would have become an element of frustration to have to manually enter the paths to the required directories over the course of many iterations, had they performed more iterations with the tool.

**4.3.2.1.2 Manual extraction** Participant A experienced that the manual extraction involved many steps, and that there were many things to keep track of. They remarked that they would sometimes perform steps in the wrong order, and have to start over. They also found it difficult to have to locate the important assets manually. Thus, they felt that understanding the manual SecDFD extraction process was one issue, and actually locating the important assets and elements was another challenge.

They remarked that their experienced challenges affected both time consumption and the correctness of their results. They mentioned that it was their lack of understanding for the process that led to them having to redo steps, and that this in turn affected the time consumption.

They found the manual SecDFD extraction process frustrating in the sense that they at time could not locate the correct assets and elements.

Regarding the SecDFD specification language syntax, they felt that it was similar to other markup- and object oriented languages that they had previously used, with some key differences, which in turn led to frustration since they would often make small mistakes when specifying the SecDFD in the specification language.

**4.3.2.1.3 Preferred method** When asked which of the methods they would use if tasked to extract a SecDFD again, they say that they would choose to use the extraction tool, mainly for the assistance it contributed with by providing the correct format and syntax in the automated output.

#### **4.3.2.2 Participant B**

Participant B performed automatic SecDFD extraction on SC1 first, and manual SecDFD extraction on SC2 subsequently.

**4.3.2.2.1 Automatic extraction** Participant B found the output produced by the tool to be relatively correct – not entirely correct, they remarked, since it did not contain all elements of the ground truth. They also expressed that they felt that they could not say with complete certainty that it was correct, since they did not know the code-base well, but rather that it appeared correct from what they could tell without delving deeper into the code.

They felt that the assets found by the extraction tool were too many, but that they would have been able to find the correct ones over more time and iterations. They also expressed that they found the properties of the assets to be correct after manual editing, and that they were relatively easy to find given the assets themselves.

When asked why they found their solution correct or incorrect, they replied that they found it correct but with too much overhead, and that one thing in the ground truth was represented by maybe three things in the automatically extracted SecDFD. They also found the variable names unreadable.

Participant B experienced difficulties with defining keywords, and entering keywords that are too specific.<sup>3</sup>

The participant felt that one advantage with the tool was that they could find potential assets they would not otherwise have thought to look for. They also found it beneficial to be able to enter single letters as keywords and get a high number of matches, in order to then narrow their search.

They expressed that they found it annoying to have to repeatedly input the required paths manually to the tool over the course of their iterations, and that it was more annoying than time consuming.

The participant stated that they believe it would be good to have performed manual SecDFD extraction at least once or twice before using the extraction tool, likening using the tool before having performed manual extraction to using greater functions on a calculator before knowing what the function actually does – that is, using a method before knowing the fundamentals of how it works.

**4.3.2.2.2 Manual extraction** The participant expressed that they found it difficult to compare manual extraction of a SecDFD to automatic extraction, since they felt much more prepared when they were tasked with manual extraction, having gained experience from performing the automatic extraction beforehand. They note that they experienced that they gained a sense of the concept and SecDFDs from the automatic extraction process, which made it easier for them during the manual extraction process.

They remark that they believe that if someone has had practical experience with manual SecDFD extraction beforehand, it becomes easier for them to use the extraction tool.

The participant felt that they were in much greater control during the manual extraction, but that it did take much more time. They also add that they would most likely not have been able to finish a single asset, had they been tasked with performing the manual SecDFD extraction first, rather than having done the automatic SecDFD extraction before being tasked with doing it manually.

They experienced that the manual SecDFD extraction process was time consuming, and add that they would have liked to have a template.<sup>4</sup>

Regarding how their end result was affected by the challenges they experienced, they felt that the correctness was affected due to the fact that they were not sure if they had found everything by manually looking through the code, and were thus missing required SecDFD elements.

---

<sup>3</sup>Author's note: Too specific keywords run the risk of causing the tool to return an empty SecDFD, as it will not find exact matches. However, an empty SecDFD will also be returned if no `*cgraph.dot` files have been generated by Doxygen. As Participant B experienced trouble with Doxygen in the shape of unexpected behaviour when using the correct settings, the tool would output empty SecDFDs over the course of their iterations. However, this was treated as keyword related issues before the missing files were detected after the author intervened, after which the participant generated the correct files.

<sup>4</sup>By template, they meant for the formal specification of SecDFD in the SecDFD specification language.

**4.3.2.2.3 Preferred method** When asked which approach they would use if tasked with extracting a SecDFD in the future, the participant replied that they would definitely use the extraction tool now that they had a greater sense of what is going on.



# 5

## Conclusion

In this chapter I will discuss the results of the thesis, bringing up reflections as well as relating the results to the research questions and the work in the field. I will also present the threats to validity, as well as a final conclusion, including my reflections and recommendations for future work and improvements.

### 5.1 Discussion

In this section I will discuss the results of the thesis, both in regard to the final product of the development that is the SecDFD extraction tool, as well as the results of the conducted tests and experiments, in relation to the research questions posed in chapter 1.2. The research questions are answered in Sections 5.1.3 and 5.1.4, where Section 5.1.3 answers RQ1 and RQ1.1, and 5.1.4 answers RQ2 and RQ2.1.

#### 5.1.1 Relation to the field

The SecDFD is, as previously mentioned, relatively novel and thus has little immediate research precedent. This has also led to small data sizes and limited resources throughout this thesis. While this particular work does not directly translate to the previous works surrounding automatic architecture reconstruction and clustering algorithms discussed in Chapter 2, many of the other works utilized machine learning in some respect. This is something that requires significantly larger sample sizes than those available throughout this thesis.

A contribution of this thesis in relation to the field is the reduced time and effort of SecDFD extraction simply by a foundation to work from, as remarked by the participants during the experiment phase of the thesis. By introducing the tool and the semi-automatic extraction process, research surrounding automation of threat modelling and threat analysis may become simpler by reducing the effort of producing data by hand. This is of course mainly relevant for the research on the topic that utilizes the SecDFD as the main security model.

#### 5.1.2 Functionality and correctness

As presented in chapter 4, the degree of correctness of both the automatically extracted SecDFDs from the user experiments and the black box testing were low or inconclusive. However, seen to true positives, the rate of correctly identified SecDFD

components was consistent, albeit lower than desired. Even so, the sample sizes are too small to make statistical inferences from the results. Therefore, iterative use of the tool seems more suitable, where the user across the iterations modifies or extends the list of keywords and manually adds SecDFD elements if needed. Even so, it is also apparent from the positive response as well as the overall ability to consistently provide correct elements – and that the correctness of the elements and number of identified assets varies depending on the specified keywords – that the procedure holds great potential.

A possible reason for the observed “reverse direction” of flows in the SecDFDs produced via the automatic extraction process, as remarked in section 4.3.1.1, is confirmation bias within the participant extracting the SecDFD automatically. This builds on the theory that the participant, when seeing the flow appear in automatically generated SecDFD and optionally reviewing the code, trusted the SecDFD extraction tool more than their own instinct or observation. However, I cannot make a firm statement regarding the explanation for this, as the sample size of both participants and automatically extracted SecDFDs displaying the same degree of correctness in flows is too small for statistical inference, and the participant made no statements regarding this as it was discovered in post analysis. It is also worth noting that the participant did not react to this in their independent review of the ground truth. Their reviews were completely independent – while they were encouraged to think aloud, there was no interaction from the author during the time the participants were reviewing the ground truths.

The overall rate at which the tool produced a non-empty SecDFD was roughly 33% – however, as was illustrated in Table 4.4, a large portion of these were due to Doxygen failing to produce the correct call-graph files, which makes it a third party dependency failure. When subtracting these, the tool produced non-empty SecDFDs at a 50% frequency. Other empty SecDFDs were, however, the product of a disconnect between the names of the assets in the model and their name as represented in the code, which supports the notion that the variable naming and keyword determination affects the quality of the extracted SecDFD.

Of the instances where tool did not manage to produce a non-empty SecDFD, the majority can be accredited to keywords being specified in a way which did not give matches, which was expected and confirmed. One error was due to a Java Runtime Error. When further investigated, a notable spike was found in the number of files produced by Doxygen for project C9, as seen in Figure 4.3. Since over 35 000 files were generated in the case of C9, as opposed to below 5 000 for the remaining 10 projects, it is very likely that this was the cause. The SecDFD extraction tool operates by parsing all relevant files when creating the aggregated call graph – in the case of C9, the number of such files vastly exceeded that of previous test runs, leading to the discovery of a vulnerability in the program which in turn caused the tool and operating system to shut down. This could also be due to hardware issues, but can likely be solved by optimizing the code. It may also be useful to perform more tests to determine the maximum number of files the tool can parse.

### 5.1.3 Information from source code (RQ1)

The SecDFD extraction tool parses the source code files in order to review the function declaration and parameter names. In addition, it needs to be provided a call-graph of the project under analysis. The analysis and discussion of the method is made based on this being the information provided to the tool.

As mentioned in section 4.1.3.2, given the path to the files apparent in the nodes of the call-graph as specified by `*cgraph.dot` files, the SecDFD extraction tool used a regular expression to find the method declaration and its given parameters. The rationale for matching against the parameters of a function was that, if naming conventions in regard to descriptive naming were regularly followed throughout the source code, any function that processed security relevant information across multiple processes may have it as a parameter. The reason for also matching against the function name was to cover for the cases where a function processes security relevant information but either has generically named parameters, or takes no parameters but rather makes, for instance, direct inquiries to databases or devices. In this case, the extraction tool largely relies on the naming of the function – specifically, that it has been descriptively named according to what it does, and to what kind of data it does it.

### 5.1.4 Comparison to manual extraction (RQ2)

The experiment participants were greatly positive to the use of the tool, as displayed in section 4.3.2. Both participants used both methods once, and were positive to the performance and shorter time consumption of the SecDFD extraction tool. Regarding manual extraction, they felt that it was a very complex task to master, especially in a short time. However, they also felt that they gained a more in-depth understanding of the code they were working with when tasked with manual extraction, rather than when performing automatic extraction.

It is worth noting that both participants remarked on the learning effects of whichever task they were given first, with Participant B hypothesizing that they would have been able to perform a more accurate automatic extraction had they performed a manual extraction first. This is further suggested by the fact that Participant A, who performed manual extraction before automatic, showed more correct results overall. However, more user experiments need to be carried out in order to make any firm conclusions regarding this effect.

## 5.2 Threats to validity

In this section I will discuss the threats to internal and external validity of the thesis, as well as the reliability.

### 5.2.1 Internal validity

There is a running risk for confounding variables in several aspects of this thesis. The participants, while to a carefully specified set of requirements during participant

selection, may perform and respond differently due to variables not considered, such as health, personal background, comfort, and technical experiences in other domains.

Regarding the black box testing phase, the SecDFDs acting as ground truth may have been subjected to learning effects throughout the course of manual extraction, causing their correctness to vary from file to file. Additionally, these may also have been affected by a number of variables that are not known to the author, who was not present at the time these particular SecDFDs were manually extracted. Therefore it is not safe to assume that there were no environmental factors such as comfort, disturbance, and so on, affecting the performance and concentration of the person performing the manual SecDFD extraction. As the projects were chosen on an availability basis, there was little room for the author to mitigate threats to validity that were introduced during, and by, the manual extraction of these particular SecDFDs.

During the user experiments, there may be a risk for experimenter bias. While the author was operating with minimal interaction, as specified in the methodology, there are always risks for unconscious communication via for instance body language, which may risk affecting the choices and actions of the experiment participant.

During the development phase of the project, throughout the iterations, there may arise uncertainties regarding temporal precedence of changes, which in turn risks affecting the validity.

It is necessary to prevent information spreading between participants regarding the content and structure of the experiment, since this may lead to diffusion. Putting a contract in place that prevents participants from discussing the contents of the experiment would be a countermeasure of this particular threat, albeit not waterproof as the actions of the participants cannot be regulated beyond the signing of the contract.

Conclusion validity may be affected due to the limitations of strictly observing participant behaviour throughout the experiments. Encouraging participants to think aloud throughout the experiment, on record, is a measure towards reducing the number of assumptions made regarding their choices, which in turn lessens the risk of drawing false conclusions regarding their impressions and choices.

### 5.2.2 External validity

The small data problem which permeates the project means that the low number of sample SecDFDs used throughout the testing and experimentation of the project affects the statistical conclusion validity of the thesis. Similarly, time and resource constraints affect the number of experiment iterations, which affects the statistical conclusion validity. Additionally, the same constraints result in convenience sampling of both SecDFDs for the various testing phases, and the selection of participants, which in turn affects the statistical conclusion validity once more.

The SecDFDs that are treated as the ground truth throughout the experiment may not be entirely correct, nor entirely consistent in their level of accuracy, due to the subjective nature of manual extraction. Thus, the quality of the ground truths used depend on a multitude of factors with the expert who was produced them, such as environment, learning effects, subconscious reasoning, and confirmation bias.



### 5.2.3 Reliability

Measures were taken throughout the thesis in order to ensure reliability to the extent possible. However, the experiment phase is of course affected by the participants to some extent – seeing as while they fulfil the criteria defined to normalize their prior experience and ensure that they had the necessary prerequisites, they were in fact different individuals with different experiences defining their approach to the task, and are thus not interchangeable with people who share similar backgrounds and experiences. When replicating the experiment, there is thus no guarantee that the same results will be produced even when using the same experiment flow, same training material, same projects, and same tool. Apart from the individuality of the participants, including the questions they asked during the training, the experiments can be replicated based on the descriptions given in the documentation in previous chapters.

The black box tests should, when replicated with the same projects and methods, produce the same results in thanks to the deterministic keyword determination and single iteration approach. However, seeing as the evaluation of correctness was qualitative and thus relied on individual interpretation, the details of the conclusions may come to vary depending on the software experience and SecDFD expertise of the one conducting the tests and evaluations.

Regarding the implementation choices and development of the tool based on the requirements, there is of course a chance for variation depending on the developer and their software experience, as this task – being a vague, largely unspecified development task – relies greatly on individual skill and interpretation.

## 5.3 Conclusion

As the use of information technology surges, software systems become increasingly integrated in all aspects of society. These systems are processing a wide range of sensitive data on a daily basis, and trends indicate that the use of such systems is going to drastically increase within the next few years. In order to ensure that developed applications are secure, many organizations employ threat modelling, which requires the involvement of a security expert – however, security experts remain a scarce resource in many companies. Additionally, the threat modelling process can be highly time consuming. By automating parts of the threat modelling process, organizations could thus save a great amount of time and resources.

In this thesis I have developed and evaluated a procedure for semi-automatically extracting SecDFDs from source code by leveraging call-graphs and a set of keywords which reflect the information assets of the software under analysis. The evaluation was performed by independent tests, and user experiments. The results were then qualitatively evaluated.

The information required from the source code is, in combination with a formally specified call graph of the software under analysis, the function- and parameter names, which act as indicators for information assets (**RQ 1**). These are extracted by searching the source files using regular expressions, to match against the list of keywords (**RQ 1.1**). Using the tool can provide a similar degree of correctness

as manual extraction, but with less time consumption (**RQ 2**). However, manual SecDFD extraction is indicated to allow for better code comprehension and a greater degree of control over the process (**RQ 2.1**).

While the SecDFD extraction tool requires further work for completion and correctness solidification, the tool and the automatic SecDFD extraction process holds great potential. While being a straight-forward, deterministic process, the SecDFD extraction tool has proved a determined first step in the right direction judging by positive user experiences and stable performance, providing the user flexibility while saving time. It is perhaps to be regarded as a complement to manual extraction at this time, acting as an augmentation of human intelligence. Furthermore, this thesis opens up for several new areas of investigation, offering a foundation to build upon such as the application of machine learning to the SecDFD extraction process.

### 5.3.1 Future work

Section 4.1.3.2 details what components of a SecDFD the tool delivers. While these need further tuning, there are also SecDFD components that need to be added, such as External Entities and Data Stores, as well as Asset targets and a wider range of Process responsibilities. The method of adding these will require additional investigation, but I believe that identifying most of these components *correctly* requires more than simply regular expressions and call-graphs.

As is, the tool does not rely on documentation, apart from the call-graphs generated by Doxygen. I believe that it may be worth investigating whether a natural language processor can be used to process documentation in cases where it exists, in order to discern which method plays a role in handling security relevant information and thus identify assets by those means. Ideally, this matching would occur in combination with complementary techniques.

The SecDFD and the work surrounding it is still very much novel. This comes with an issue common machine learning problems such as image recognition are less affected by – a great lack of labelled datasets. In order to be an applicable problem for machine learning, a much greater number of SecDFDs coupled with their respective sources is necessary. Even throughout this project, the low number of pre-existing SecDFDs resulted in having to resort to use projects that were not purely written in Java for the black box testing phase, which was necessary in order to investigate the stability of the tool.

Ideally, such a testing phase would be conducted with more projects in order to be able to make any kind of reliable statistical inference. I believe that an added improvement to future tests in this sentiment would be a numeric scoring system for correctness, which in itself needs careful consideration and development.

I would thus like to propose a few possible future topics for research in the area, other than improvement of the tool, which are welcome to build upon the SecDFD extraction tool. These are:

1. Creating labelled datasets for SecDFDs, which in turn requires determining the necessary labels

2. Developing a SecDFD correctness scoring system
3. Conducting large scale user experiments on automatically extracting SecDFDs
4. Investigating best means for keyword determination
5. Conducting large scale black box tests, or by another appropriate keyword determination method
6. Applying machine learning to automatic SecDFD extraction

I suggest that these works are carried out in a sequential order, in order to maximize the outcome of each study by utilizing the full extent of the previous one.



# Bibliography

- [1] Marwan Abi-Antoun and Jeffrey M. Barnes. “Analyzing security architectures”. In: *Proceedings of the IEEE/ACM international conference on Automated software engineering - ASE '10*. New York, New York, USA: ACM Press, 2010, p. 3. ISBN: 9781450301169. DOI: 10.1145/1858996.1859001. URL: <http://portal.acm.org/citation.cfm?doid=1858996.1859001>.
- [2] Periklis Andritsos et al. “LIMBO: Scalable Clustering of Categorical Data”. In: Springer, Berlin, Heidelberg, 2004, pp. 123–146. DOI: 10.1007/978-3-540-24741-8\_9. URL: [http://link.springer.com/10.1007/978-3-540-24741-8%7B%5C\\_%7D9](http://link.springer.com/10.1007/978-3-540-24741-8%7B%5C_%7D9).
- [3] Pablo Antonino et al. “Indicator-based architecture-level security evaluation in a service-oriented environment”. In: *Proceedings of the Fourth European Conference on Software Architecture Companion Volume - ECSA '10*. New York, New York, USA: ACM Press, 2010, p. 221. ISBN: 9781450301794. DOI: 10.1145/1842752.1842795. URL: <http://portal.acm.org/citation.cfm?doid=1842752.1842795>.
- [4] Francesca Arcelli Fontana and Marco Zanoni. “A tool for design pattern detection and software architecture reconstruction”. In: *Information Sciences* 181.7 (Apr. 2011), pp. 1306–1324. ISSN: 0020-0255. DOI: 10.1016/J.INS.2010.12.002. URL: <https://www.sciencedirect.com/science/article/pii/S0020025510005955>.
- [5] Bernhard J. Berger, Karsten Sohr, and Rainer Koschke. “Automatically extracting threats from extended data flow diagrams”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2016. ISBN: 9783319308050. DOI: 10.1007/978-3-319-30806-7\_4.
- [6] Nathan Brown et al. *A graph-based genetic algorithm and its application to the multiobjective evolution of median molecules*. 2004. DOI: 10.1021/ci034290p.
- [7] Naresh Chauhan. *5.1 Need of White-Box Testing*. 2010. URL: <https://app.knovel.com/hotlink/khtml/id:kt00U5AM11/software-testing-principles/need-white-box-testing>.
- [8] Mauro Conti et al. “Internet of Things Security and Forensics: Challenges and Opportunities”. In: (July 2018). DOI: 10.1016/j.future.2017.07.060. arXiv: 1807.10438. URL: <http://arxiv.org/abs/1807.10438%20http://dx.doi.org/10.1016/j.future.2017.07.060>.

- [9] Anna Corazza et al. “Weighing lexical information for software clustering in the context of architecture recovery”. In: *Empirical Software Engineering* 21.1 (Feb. 2016), pp. 72–103. ISSN: 1382-3256. DOI: 10.1007/s10664-014-9347-3. URL: <http://link.springer.com/10.1007/s10664-014-9347-3>.
- [10] Doxygen. *Doxygen: Main Page*. URL: <http://www.doxygen.nl/> (visited on 07/30/2019).
- [11] Emelie Engström et al. “A review of software engineering research from a design science perspective”. In: (Apr. 2019). arXiv: 1904.12742. URL: <http://arxiv.org/abs/1904.12742>.
- [12] Ericsson. *Internet of Things forecast – Ericsson Mobility Report - Ericsson*. URL: <https://www.ericsson.com/en/mobility-report/internet-of-things-forecast> (visited on 07/29/2019).
- [13] Fei Wang and Changshui Zhang. “Label Propagation through Linear Neighborhoods”. In: *IEEE Transactions on Knowledge and Data Engineering* 20.1 (Jan. 2008), pp. 55–67. ISSN: 1041-4347. DOI: 10.1109/TKDE.2007.190672. URL: <http://ieeexplore.ieee.org/document/4358958/>.
- [14] Francois Fouss et al. “Random-Walk Computation of Similarities between Nodes of a Graph with Application to Collaborative Recommendation”. In: *IEEE Transactions on Knowledge and Data Engineering* 19.3 (Mar. 2007), pp. 355–369. ISSN: 1041-4347. DOI: 10.1109/TKDE.2007.46. URL: <http://ieeexplore.ieee.org/document/4072747/>.
- [15] Joshua Garcia, Igor Ivkovic, and Nenad Medvidovic. “A comparative analysis of software architecture recovery techniques”. In: *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, Nov. 2013, pp. 486–496. ISBN: 978-1-4799-0215-6. DOI: 10.1109/ASE.2013.6693106. URL: <http://ieeexplore.ieee.org/document/6693106/>.
- [16] Joshua Garcia et al. “Enhancing architectural recovery using concerns”. In: *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*. IEEE, Nov. 2011, pp. 552–555. ISBN: 978-1-4577-1639-3. DOI: 10.1109/ASE.2011.6100123. URL: <http://ieeexplore.ieee.org/document/6100123/>.
- [17] Giona Granchelli et al. “Towards Recovering the Software Architecture of Microservice-Based Systems”. In: *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. IEEE, Apr. 2017, pp. 46–53. ISBN: 978-1-5090-4793-2. DOI: 10.1109/ICSAW.2017.48. URL: <http://ieeexplore.ieee.org/document/7958455/>.
- [18] GraphViz. *Graphviz - Graph Visualization Software*. URL: <https://www.graphviz.org/> (visited on 07/30/2019).
- [19] GraphViz. *The DOT Language*. URL: <https://www.graphviz.org/doc/info/lang.html> (visited on 07/30/2019).
- [20] Stefanie Jasser et al. “Back to the Drawing Board - Bringing security constraints in an architecture-centric software development process”. In: *ICISSP*. 2018. ISBN: 9789897582820.

- 
- [21] R. Koschke and D. Simon. “Hierarchical reflexion models”. In: *10th Working Conference on Reverse Engineering, 2003. WCRE 2003. Proceedings.* IEEE, pp. 36–45. ISBN: 0-7695-2027-8. DOI: 10.1109/WCRE.2003.1287235. URL: <http://ieeexplore.ieee.org/document/1287235/>.
- [22] Zhe Liu, Kim Kwang Raymond Choo, and Johann Grossschadl. “Securing Edge Devices in the Post-Quantum Internet of Things Using Lattice-Based Cryptography”. In: *IEEE Communications Magazine* (2018). ISSN: 01636804. DOI: 10.1109/MCOM.2018.1700330.
- [23] Thibaud Lutellier et al. “Comparing Software Architecture Recovery Techniques Using Accurate Dependencies”. In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering.* IEEE, May 2015, pp. 69–78. ISBN: 978-1-4799-1934-5. DOI: 10.1109/ICSE.2015.136. URL: <http://ieeexplore.ieee.org/document/7202951/>.
- [24] Kyriakos Malamas and Danial Hosseini. “Design Flaws as Security Threats”. MA thesis. 2017. URL: <https://odr.chalmers.se/handle/20.500.12380/250250>.
- [25] O. Maqbool and H.A. Babri. “Bayesian Learning for Software Architecture Recovery”. In: *2007 International Conference on Electrical Engineering.* IEEE, Apr. 2007, pp. 1–6. ISBN: 1-4244-0892-X. DOI: 10.1109/ICEE.2007.4287309. URL: <http://ieeexplore.ieee.org/document/4287309/>.
- [26] G.C. Murphy, D. Notkin, and K.J. Sullivan. “Software reflexion models: bridging the gap between design and implementation”. In: *IEEE Transactions on Software Engineering* 27.4 (Apr. 2001), pp. 364–380. ISSN: 00985589. DOI: 10.1109/32.917525. URL: <http://ieeexplore.ieee.org/document/917525/>.
- [27] Larry Osterman. *Threat Modeling Again, STRIDE*. 2007. URL: <https://blogs.msdn.microsoft.com/larryosterman/2007/09/04/threat-modeling-again-stride/> (visited on 04/03/2019).
- [28] Chris Quirk, Raymond Mooney, and Michel Galley. “Language to Code: Learning Semantic Parsers for If-This-Then-That Recipes”. In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 2015. ISBN: 9781941643723. DOI: 10.3115/v1/P15-1085.
- [29] Per Runeson and Martin Höst. “Guidelines for conducting and reporting case study research in software engineering”. In: *Empirical Software Engineering* 14.2 (Apr. 2009), pp. 131–164. ISSN: 1382-3256. DOI: 10.1007/s10664-008-9102-8. URL: <http://link.springer.com/10.1007/s10664-008-9102-8>.
- [30] Hitesh Sajnani. “Automatic software architecture recovery: A machine learning approach”. In: *2012 20th IEEE International Conference on Program Comprehension (ICPC)*. IEEE, June 2012, pp. 265–268. ISBN: 978-1-4673-1216-5. DOI: 10.1109/ICPC.2012.6240501. URL: <http://ieeexplore.ieee.org/document/6240501/>.

- [31] Asaf Shabtai, Yuval Fledel, and Yuval Elovici. “Automated Static Code Analysis for Classifying Android Applications Using Machine Learning”. In: *2010 International Conference on Computational Intelligence and Security* (2010). ISSN: 00320935. DOI: 10.1109/CIS.2010.77.
- [32] Robin Shahan and Bryan Lamos. *IoT Security Architecture / Microsoft Docs*. 2018. URL: <https://docs.microsoft.com/en-us/azure/iot-fundamentals/iot-security-architecture> (visited on 04/03/2019).
- [33] Chalmers University of Technology. *English Language Requirement*. URL: [https://www.chalmers.se/en/education/application-admission/entry\\_requirements/Pages/documentation-of-english-language-proficiency.aspx](https://www.chalmers.se/en/education/application-admission/entry_requirements/Pages/documentation-of-english-language-proficiency.aspx) (visited on 07/30/2019).
- [34] Katja Tuma, Riccardo Scandariato, and Musard Balliu. “Flaws in Flows: Unveiling Design Flaws via Information Flow Analysis”. In: *2019 IEEE International Conference on Software Architecture (ICSA)*. IEEE, Mar. 2019, pp. 191–200. ISBN: 978-1-7281-0528-4. DOI: 10.1109/ICSA.2019.00028. URL: <https://ieeexplore.ieee.org/document/8703905/>.
- [35] Katja Tuma et al. “Towards security threats that matter”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2018. ISBN: 9783319728162. DOI: 10.1007/978-3-319-72817-9\_4.
- [36] V. Tzerpos and R.C. Holt. “ACCD: an algorithm for comprehension-driven clustering”. In: *Proceedings Seventh Working Conference on Reverse Engineering*. IEEE Comput. Soc, pp. 258–267. ISBN: 0-7695-0881-2. DOI: 10.1109/WCRE.2000.891477. URL: <http://ieeexplore.ieee.org/document/891477/>.
- [37] Alexander Van Den Berghe et al. “A model for provably secure software design”. In: *Proceedings - 2017 IEEE/ACM 5th International FME Workshop on Formal Methods in Software Engineering, FormaliSE 2017*. 2017. ISBN: 9781538604229. DOI: 10.1109/FormaliSE.2017.6.
- [38] Perla Velasco-Elizondo et al. “Knowledge representation and information extraction for analysing architectural patterns”. In: *Science of Computer Programming* 121 (June 2016), pp. 176–189. ISSN: 0167-6423. DOI: 10.1016/J.SCICO.2015.12.007. URL: <https://www.sciencedirect.com/science/article/pii/S0167642316000101>.
- [39] Roel J. Wieringa. *Design science methodology for information systems and software engineering*. Springer, 2014, p. 332. ISBN: 3662438399.
- [40] Yan Zhang and Barbara M. Wildemuth. “Applications of social science research methods to questions in information and library science”. In: *Qualitative Analysis of Content*. 2005, pp. 308–319. ISBN: 9781440839054.
- [41] Zhi Kai Zhang et al. “IoT security: Ongoing challenges and research opportunities”. In: *Proceedings - IEEE 7th International Conference on Service-Oriented Computing and Applications, SOCA 2014*. 2014. ISBN: 9781479968336. DOI: 10.1109/SOCA.2014.58.