Abstract

Currently, software applications operate using highly sensitive data that only a limited number of people can access. In some cases, human lives might depend on the continuous and correct deployment of these applications. Due to the rise of cybercrime and reported vulnerabilities, organizations now consider security a top concern. Therefore, it is problematic that software developers generally have no security experience, which could be a reason for occurring vulnerabilities. A further problem that underlines the criticality of missing know-how is that the presence of a security expert within projects is not always guaranteed. A frequent practice to address knowledge gaps is to ask the developer community for help in solving concrete implementation concerns.

In addition to websites of the Stack Exchange network, such as Stack Overflow, developers can use the National Vulnerability Database (NVD) to discover vulnerabilities in concrete software artifacts or products. Using vulnerability databases to find security flaws within written source code is complex. Much information has to be considered, which can overwhelm developers. Moreover, the NVD does not directly provide code fragments showing the expression of vulnerabilities in source code. Sometimes NVD provides links to other websites containing proofs of concept or a security-patch code.

The goal of this dissertation is to support developers in applying security checks using community knowledge. Artificial intelligence approaches combined with natural language processing techniques are employed to identify security-related information from community websites such as Stack Overflow or GitHub. All security-related information is stored in a security knowledge base. This knowledge base provides code fragments that represent the community's knowledge about vulnerabilities, security-patches, and exploits. Furthermore, meta information regarding vulnerability types and identifiers of concrete security flaws are provided. Comprehensive knowledge is required to carry out security checks on software artifacts, such as data covering known vulnerabilities and their manifestation in the source

code as well as possible attack strategies. Approaches that check software libraries and source code fragments are provided for the automated use of the data.

Insecure software libraries can be detected using the NVD combined with metadata and library file hash approaches introduced in this dissertation. Vulnerable source code fragments can be identified using community knowledge represented by code fragments extracted from the largest coding community websites: Stack Overflow and GitHub. A state-of-the-art clone detection approach is modified and enriched by several heuristics to enable vulnerability detection and leverage community knowledge while maintaining good performance. Using various case studies, the approaches implemented in Eclipse plugins and a JIRA plugin are adapted to the users' needs and evaluated.

Keywords: Community knowledge, Security, Vulnerability detection, Machine learning, Natural language processing, Artificial intelligence, Code clone detection, Software libraries, Source code

Zusammenfassung

Derzeit arbeiten Softwareanwendungen mit hochsensiblen Daten, auf die nur eine begrenzte Anzahl von Personen zugreifen können. In einigen Fällen können von der kontinuierlichen und korrekten Bereitstellung dieser Anwendungen Menschenleben abhängen. Aufgrund des Anstiegs der Cyberkriminalität und der in diesem Zusammenhang gemeldeten Schwachstellen ist Sicherheit für Unternehmen mittlerweile ein wichtiges Thema. Vor diesem Hintergrund ist es als problematisch zu bewerten, dass Softwareentwickler in der Regel keine Sicherheitserfahrung haben (was seinerseits ein Grund für die oben genannten auftretenden Schwachstellen sein könnte). Da in vielen Projekten nicht immer die Anwesenheit eines Sicherheitsexperten gewährleistet ist, wird damit die Kritikalität des fehlenden Know-hows unterstrichen. Stattdessen besteht die Praxis, um Wissenslücken zu schließen, häufig darin, die Entwicklergemeinde punktuell um Hilfe bei der Lösung konkreter Implementierungsprobleme zu bitten.

Um Schwachstellen in konkreten Software-Artefakten oder Produkten zu entdecken, können Entwickler neben Webseiten des Stack-Exchange-Netzwerks, wie z. B. Stack Overflow auch die National Vulnerability Database (NVD) nutzen. Die Verwendung solcher Schwachstellendatenbanken zum Auffinden von Sicherheitslücken in geschriebenem Quellcode ist jedoch komplex. Denn es müssen viele Informationen berücksichtigt werden, was Entwickler überfordern kann. Außerdem stellt die NVD nicht direkt Codefragmente zur Verfügung, die die Ausprägung von Schwachstellen im Quellcode zeigen. Manchmal stellt die NVD nur Links zu anderen Websites bereit, die Proofs of Concept oder Sicherheits-Patch-Codes enthalten.

Das Ziel dieser Dissertation ist es, Entwickler bei der Anwendung von Sicherheitsprüfungen mit Hilfe von Community-Wissen zu unterstützen. Dabei werden Ansätze der künstlichen Intelligenz mit Techniken der natürlichen Sprachverarbeitung kombiniert und dazu eingesetzt, um sicherheitsrelevante Informationen von Community-Websites wie Stack Over-

flow oder GitHub zu identifizieren. Die so gesammelten sicherheitsrelevanten Informationen werden in einer Sicherheits-Wissensbasis gespeichert. Diese Wissensbasis enthält Codefragmente, die das Wissen der Community über Schwachstellen, Sicherheits-Patches und Exploits repräsentieren. Darüber hinaus werden in der Wissensbasis Metainformationen zu Schwachstellentypen und Bezeichnern konkreter Sicherheitslücken bereitgestellt. Um Sicherheitsüberprüfungen von Software-Artefakten durchführen zu können, ist umfangreiches Wissen erforderlich, wie z. B. Wissen über Daten zu bekannten Schwachstellen und deren Manifestation im Quellcode sowie über mögliche Angriffsstrategien. Für die automatisierte Nutzung der Daten werden Ansätze bereitgestellt, anhand derer Softwarebibliotheken und Quellcodefragmente überprüft werden können.

Unsichere Softwarebibliotheken können mit Hilfe der NVD in Kombination mit den in dieser Dissertation vorgestellten Ansätzen für Metadaten und Bibliotheksdateihashs erkannt werden. Anfällige Quellcodefragmente können mit Hilfe von Community-Wissen identifiziert werden, das durch Codefragmente repräsentiert wird, die von den größten Websites der Coding-Community extrahiert wurden: Stack Overflow und GitHub. Ein hochmoderner Ansatz zur Erkennung von Klonen wird modifiziert und mit mehreren Heuristiken angereichert, um die Erkennung von Schwachstellen zu ermöglichen und das Wissen der Community zu nutzen, während gleichzeitig eine gute Performance beibehalten wird. Anhand verschiedener Fallstudien werden die in Eclipse-Plugins und einem JIRA-Plugin implementierten Ansätze an die Bedürfnisse der Anwender angepasst und evaluiert.

Chapter 1

Introduction

Headlines frequently report security incidents. Due to the rise of cybercrime and incidents of reported vulnerabilities [124], security is a top concern for organizations organizations [45, 153, 20]. Security is also an essential quality aspect in recent software projects. In the past, there were several examples of insecure software that lead to leaks of sensitive data. The consequences of security breaches for software companies can be severe due to reputational damage to corporate identity and fines [82, 159]. Furthermore, the impacts of security incidents could harm private users and enterprises. Undetected vulnerabilities during implementation can lead to increasing maintenance costs and financial penalties that can exceed the development budget [38]. Moreover, human lives can depend on software that ensures the continuous and faultless provision of service, for example, the software that manages the cooling of radioactive centrifuges within a nuclear reactor [36].

Software project security relies on the expertise of project members. One reason for security issues could be the absence of security expertise in developers [60]. For instance, a study by Acar et al. [2] reviewing 307 professionals and students revealed that developers usually have no security expertise. They discovered that there is no difference between students and professionals considering their security know-how. Furthermore, the availability of security experts within a project is not always guaranteed. Moreover, if an expert is available, it is usually not for the entire development phase.

Ideally, security experts should review the complete source code of a project for vulnerabilities. Considerable knowledge is required to search for vulnerabilities in self-written source code [73]. This knowledge includes information about attack strategies and patterns of previously occurred vulnerabilities. Additionally, knowledge of the top libraries and insecure hash algorithms used to exploit vulnerabilities can help prevent their usage, which

leads to a more secure source code. Furthermore, guidelines or examples of code for the secure implementation of concrete concerns could help as well.

This dissertation investigates how developers can be supported within the knowledge-based task of security checks on source code artifacts. For this purpose, freely accessible historical community knowledge of vulnerabilities should be used. Section 1.1 describes community knowledge and potential knowledge sources. The problems and concrete challenges (Ch's) considered in this thesis are summarized in Section 1.2. The underlying research methods and questions are presented in Sec. 1.3, and the contributions of this dissertation is placed in Section 1.4. Section 1.5 shows the structure of this thesis.

1.1 Security Community Knowledge

Developers can research security flaws within publicly available vulnerability databases such as the National Vulnerability Database (NVD) [129] to obtain information about almost all already identified vulnerabilities. In practice, additional effort is required to ascertain that the necessary information is non-trivial. Unfortunately, the associated manual process to find vulnerabilities in projects is time-consuming and error-prone: Developers need to use a search engine to find relevant entries based on the names of the libraries used. Then, developers must manually scan the source code to uncover problematic uses of the affected libraries. Even worse, the support with available examples of vulnerabilities, patches, and exploit codes in the databases is scarce. Vulnerability entries sometimes just contain a link to a report or to a repository that provides additional information on proofs of concept, patches, and exploits.

Furthermore, security knowledge is also included in user contributions to coding community websites such as Stack Exchange [69] and GitHub [112]. Part of the Stack Exchange network is the well-known website Stack Overflow [9], which developers use to find implementations that solve specific problems. Stack Overflow contrasts with NVD, which is a similar workflow for coding communities to find security flaws in source code. NVD makes a distinction regarding the availability of source code contributed within user posts. The NVD partially provides links to other websites that provide source code related to listed vulnerabilities. On Stack Overflow, predefined areas can be used to post source code fragments.

Within a search, developers can be overwhelmed by the vast number of results on these websites. The resulting information can be both natural language texts and code fragments, and the information contained in posts is not restricted to security issues. If security-related content is found, it exhibits granularity, which makes it challenging to obtain the necessary security information for developers without security experience. For example, content could be about a concrete software library, a native Java method, or a program that has security issues. Furthermore, if security-related content is found, it is not easy to distinguish whether the information is about code that is securely patched, vulnerable, or describes the exploit [169]. Without any security knowledge, developers can only trust into contributions of the community. If developers are looking for security information about a concrete concern, they have to interpret the statements of numerous posts.

Developers can use several sources of security community knowledge that affect the implementation of secure software. Figure 1-1 shows some of these knowledge sources with various information affecting selected software development phases. The center of the visualization contains the specific kind of information that influences the software development phases. All knowledge sources are visualized in a surrounding cloud containing examples of concrete sources providing this specific information. Thereby, security patterns and bug reports are expressed by security experts' user contributions or experienced developers using the remaining knowledge sources.

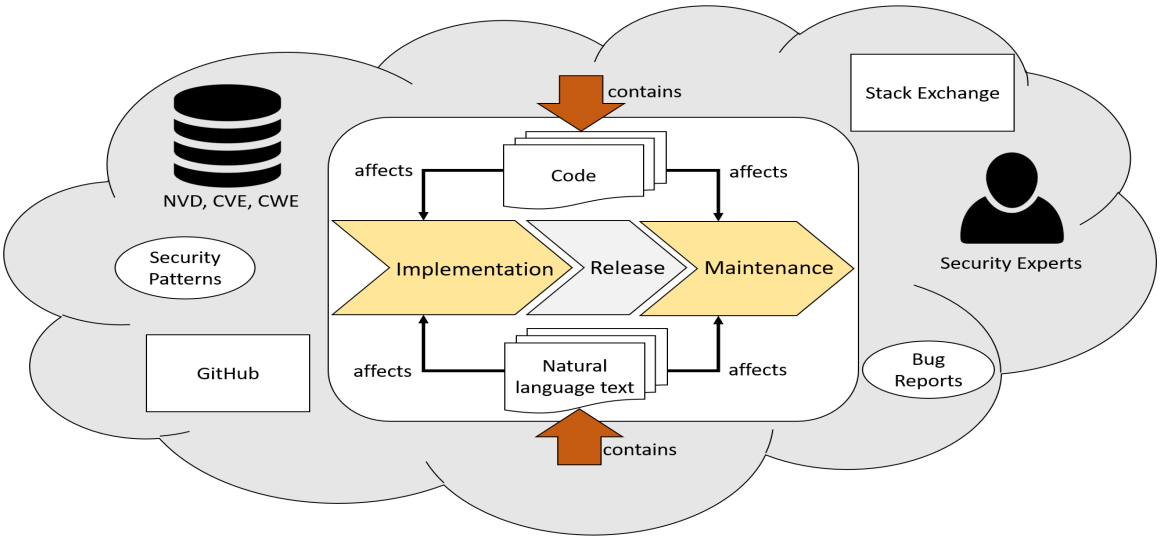

Figure 1-1: Effects of Security Knowledge on Software Implementation.

These knowledge sources partially contain natural language texts and code fragments

consisting of security-relevant data. In a way, the users' contributions to community websites reflect their security knowledge. Some of the posted code fragments will be identified as vulnerable or secure by the community. The decisions about the security relevance of code express the state of security know-how. In summary, this dissertation considers the following aspects:

1) Extraction of Security knowledge from various sources for identifying vulnerabilities in source code artifacts.

2) Automation of security checks during software development.

1.2 Challenges

In practice, security knowledge is insufficiently integrated into the software development process [76, 84], which leads to vulnerabilities in software projects. Preventing vulnerabilities during implementation is cheaper than removing them during later maintenance phases [97]. US-Cert has announced that a system is only as secure as its weakest component, which implies that a single weakness within a written code fragment or a used library can make a whole system insecure [164, 73]. Also, the Open Web Application Security Project (OWASP) [32], identified the use of third-party software libraries with known vulnerabilities as one of the ten most significant risks for web projects [133]. This means that if a developer implements a code fragment with a vulnerable method in it or uses an insecure third-party software library, the entire project becomes susceptible to hackers. A single line of source code can invoke a vulnerability. Therefore, the apparent granularity for detecting vulnerabilities is on the file-level while considering the containing source code. Knowledge of security-related topics on the web pages of the coding community is expressed in two types of information: source code and natural language texts.

For interleaving security knowledge expressed by distinct types of information, it is necessary to store data in a reusable manner. Information provided by different knowledge sources makes it essential to structure it within a unique scheme. This security knowledge can be used to identify vulnerabilities during the implementation of software. As Figure 1-1 shows, the security community knowledge from various sources should be used to make developers aware of potential security risks in source code artifacts. Thus, this dissertation

aims to develop an approach for extracting community knowledge to store provided data in a unique structure and enrich it with further information. Furthermore, a semi-automated procedure is offered to systematically support security checks within the software implementation phase.

For the identification of vulnerabilities, the knowledge of already-occurred security flaws can be beneficial. With this knowledge, it is possible to avoid making the same mistakes so that known vulnerabilities will be mitigated. The effort to obtain, enrich, and apply this security knowledge has to be minimized so that the time consumption is not excessive. Simply providing community security knowledge to developers will result in them being overwhelmed by the vast amount of information. Therefore, support for using this information during software development is required.

Existing challenges are listed and divided into the two previously named aspects:

Using Security Knowledge

- **Challenge 1.1 - Identify sources of reusable security knowledge:** There exist distinct sources to obtain security knowledge. In this thesis they will be sighted and evaluated for their suitability to deliver reusable security knowledge.

- **Challenge 1.2 - Identify security knowledge:** Not all of the coding community websites like the NVD contain only security-related content. Usually, they contain information about coding from which only a small portion considers security. This makes the identification of security-related content in the context of this dissertation necessary because the plan is to use security community knowledge to identify vulnerabilities.

- **Challenge 1.3 - Distinguish security information into three classes - vulnerabilities, patches, or exploits:** For security experts, it is probably easier to determine whether security-related content belongs to any of these classes. Developers with no security experience more tend to confuse the classes. They get overwhelmed by multiple statements referred to source code fragments, which makes it difficult to interpret whether code contributes to a vulnerability, patch or exploit.

- **Challenge 1.4 - Store security knowledge in a reusable manner:** Security community knowledge is distributed over multiple websites and accessible in different

formats. Some data representing security knowledge are natural language texts and some are source code fragments. These data are yet not stored in a consistent and reusable way; thus it is difficult to leverage it for semi-automated security checks. Therefore, the data preparation and transformation in a uniquely shared scheme is a part of that challenge.

Automatization of security checks

- **Challenge 2.1 - Reuse security knowledge on reported vulnerabilities:** Based on the available security information, developers should be supported during the implementation of source code with security checks. Software developers should not be prevented from carrying out their work. Thus, a goal is to provide security checks with less user interaction as possible. Suitable procedures have to be found for integrating them into the software development process.

- **Challenge 2.2 - Security checks for different source code artifacts :** Source code could be placed within raw source files and also be part of software components like libraries. Not for all programming languages, libraries and source files can be treated equally. There are languages which not always obtain raw source code of libraries. Therefore, security checks for software libraries and source files have to be processed differently.

To support developers in solving these challenges, approaches have been developed to fulfill the two derived tasks: extraction and the use of security knowledge. These approaches have been implemented in tools for enabling their use.

1.3 Research Questions and Methodology

Exploratory research is a well-known research design [88, 31]. The center is the construction of an artifact, such as a product or system. Typically, this artifact is a prototype to solve a domain-specific problem, such as the tools created for this dissertation. The approaches, or formed tools, are evaluated with suitable metrics to measure their performance.

To emphasize the research perspective of this thesis, the identification and use of community knowledge in different forms is focused upon to support security checks on source code artifacts. At first, related work was considered to identify a domain-specific problem,

which is reflected in the challenges. Concerning the challenges of this thesis, the following research questions (RQ's) arise:

- **Research Question 1: Can security knowledge be semi-automatically extracted from coding communities?**
- **Research Question 1.1: Can security information be semi-automatically differentiated into vulnerabilities, patches, and exploits?**
- **Research Question 2: How can the extracted community knowledge be used for heuristic security checks on source code artifacts to identify vulnerabilities?**
- **Research Question 2.1: How valuable is the use of community knowledge about security in detecting vulnerabilities?**
- **Research Question 2.2: How useful are the developed approaches for detecting vulnerabilities?**

The origin of each research question resulted from one to multiple challenges. Table 1.1 shows the constraints of the research questions to the challenges.

Table 1.1: Research Questions and Challenges Relationship.

Challenges / RQs	C1.1	C1.2	C1.3	C1.4	C2.1	C2.2
RQ1	✔	✔		✔		
RQ1.1			✔			
RQ2				✔	✔	✔
RQ2.1				✔	✔	✔
RQ2.2				✔	✔	✔

The validity of the community knowledge of IT-security was determined via a sample of Stack Overflow contributions. For identifying the performance of developed approaches, different case studies used common metrics for the information retrieval. To measure the success of developed tools for real software projects, a qualitative analysis was applied within case studies during the development of two software projects in a cooperating software house. To obtain insights across the use of created tools, an experiment was applied in which the manual task of detecting vulnerabilities through the use of the NVD and handed source code

verification is compared to the software. Furthermore, the resulting security knowledge base containing security-related code fragments is validated within a further case study. Figure 1-2 visualizes the described constructive research process.

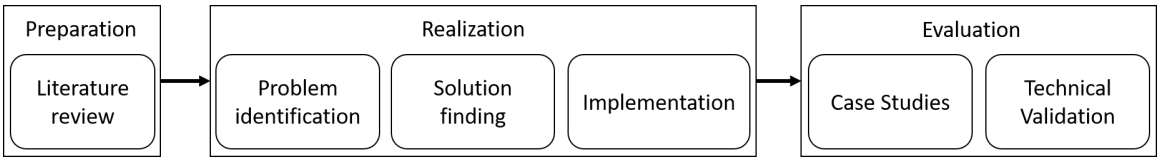

Figure 1-2: Constructive Research Process.

1.4 Contributions

To address mentioned challenges and solve the research questions identified to be relevant for this dissertation, the following contributions resulted from this.

- An automated approach is employed to detect vulnerabilities within source code artifacts based on community knowledge about security. Therefore, eclipse plugins and a JIRA plugin were created.
- A tool-based approach to obtain, enrich, and maintain security information and knowledge from coding communities such as Stack Overflow and GitHub. The results are stored in a repository containing security-related code examples for previously reported vulnerabilities, security-patches, and their exploits.

1.5 Structure

The dissertation is organized into three parts: the introduction, main part, and the evaluation. The introduction contains the research motivation and describes some background information to understand the thesis.

The main part contains its own related work section for each chapter in addition to the prototypical implementation. This is due to the broad scope of approaches. The main section describes the concepts of every sub-step necessary to realize security checks using community knowledge. The solutions of RQ1 and RQ1.1 are shown within the concepts described in Chapter 4.2. Chapter 5 and 6 show the solution for RQ2. The technical

validation (Ch. 8 shows the performance of approaches using community knowledge and responds thereby to RQ2.1.

The thesis is finalized with the evaluation, technical validation, case studies, and conclusions that address the research questions and fine-grained contributions to challenges. RQ2.1 and RQ2.2 are thereby answered during the case studies. Figure 1-3 provides an overview of the complete thesis.

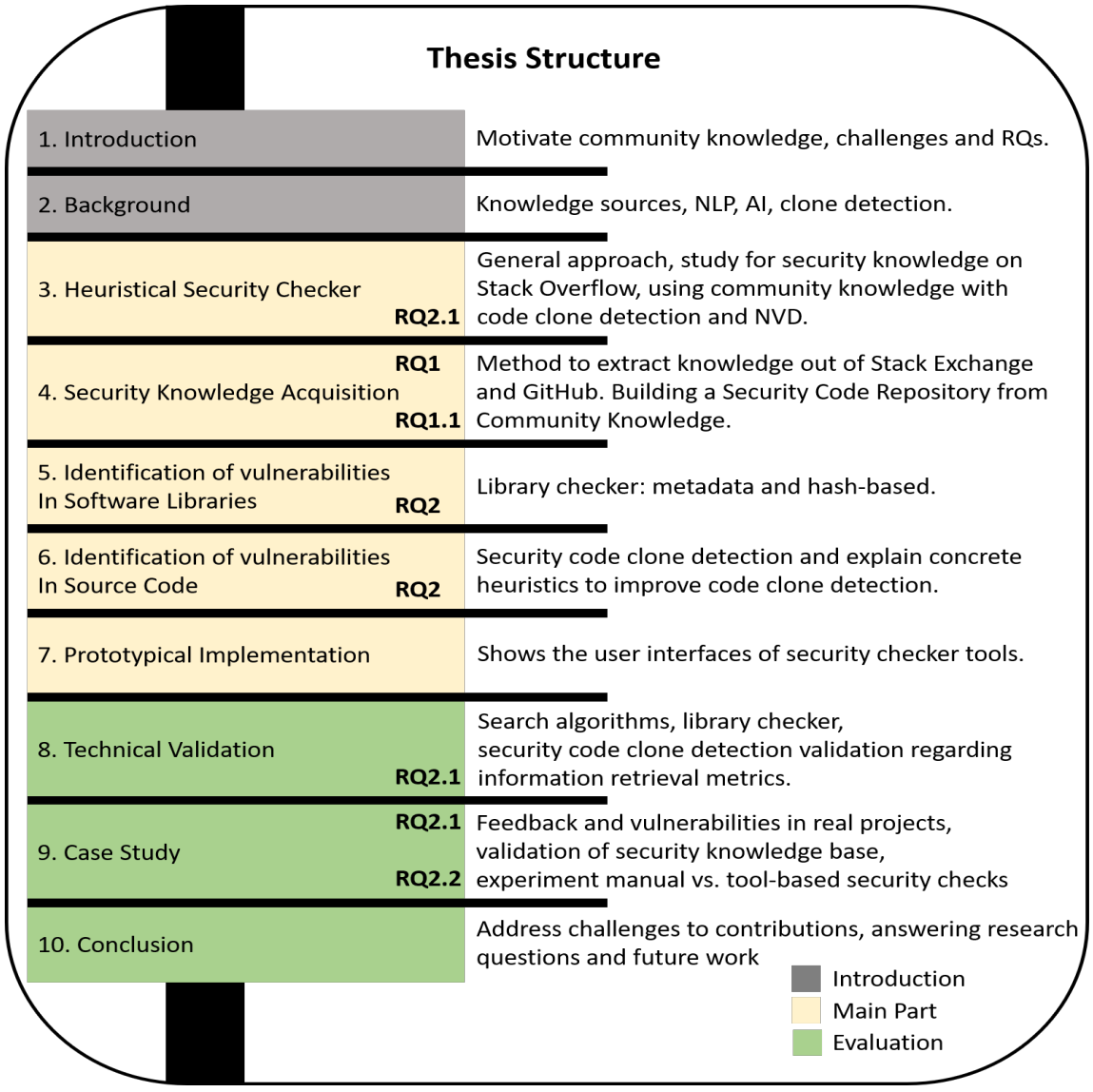

Figure 1-3: Thesis Structure.

Chapter 2

Background

The background of the topic is sketched in this chapter to inform the thesis. In particular, vulnerabilities, patches, and exploits are described based on the vulnerability lifecycle. The terms "data," "information," and "knowledge" are defined, as there has been disagreement about their meaning. Furthermore, knowledge sources containing security-related data are introduced. General techniques to access or identify this knowledge, such as artificial intelligence, natural language processing, and code clone detection, are outlined. Furthermore, metrics from the information retrieval to measure the performance of the classification techniques are introduced. The final section shows types of threats to validity from the software engineering research.

2.1 Vulnerabilities, Patches, and Exploits

Software products will likely encounter security issues during their life span. A general software security life cycle (SSLC) can be described as follows [74]: A new attack strategy is developed, or a previously unknown vulnerability is discovered. An attacker maliciously exploits the functionality of a software system using the new information and techniques. The security incident is detected and it probably punishes in some manner the customers using the attacked software product. Vendors of the developed software publish security patches to close these vulnerabilities. The SSLC is visualized in Fig. 2-1.

Within the vulnerability lifecycle mentioned above, there are three types of source code that are generated. These types are vulnerabilities, patches, and exploits, defined as follows.

- **Vulnerability.** Within the common vulnerabilities and exposures (CVE) database, a "vulnerability" is defined as a "weakness in the computational logic (e.g., code) found in

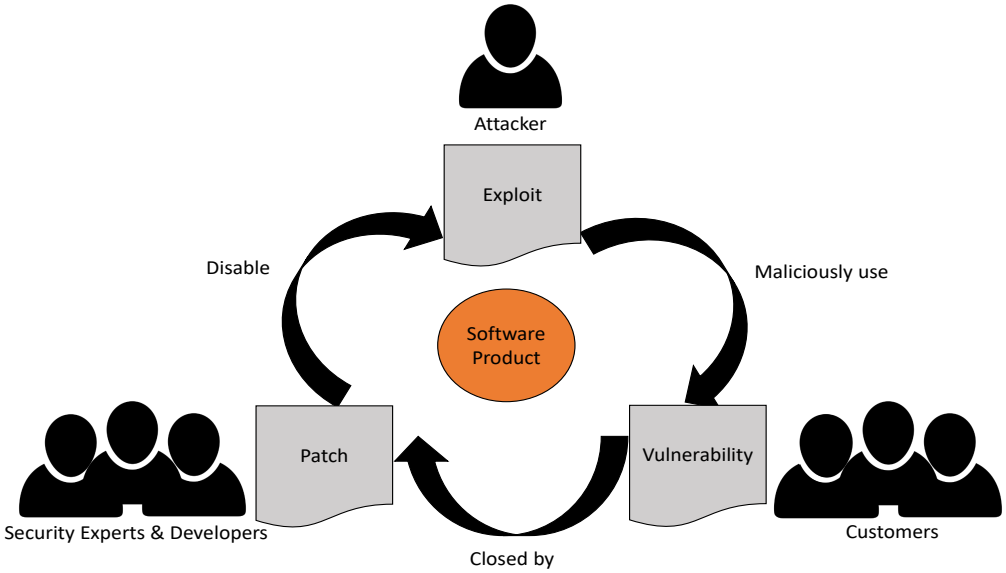

Figure 2-1: Generalized Software Security Life Cycle.

software and some hardware components (e.g., firmware) that, when exploited, results in a negative impact to confidentiality, integrity, OR availability..." [125].

- **Patch.** Removes a potential vulnerability so that it no longer can be exploited.
- **Exploit.** Maliciously uses a vulnerability to invoke a negative impact on the confidentiality, integrity, or availability of a software system.

2.2 Data, Information, and Knowledge

In the literature, there are multiple definitions of the terms "data," "information," "knowledge," and "relationships" [154]. Therefore, there is disagreement in understanding the terms. Some authors, such as Nonaka and Takeuchi, only distinguish between *information* and *knowledge* [121]. They do not define data as a separate term.

These inconsistencies underscore the importance of defining the terms. In the context of this dissertation, the author agrees with the definitions of Spek and Spijkervet [165]:

- **Data.** Not yet interpreted symbols.
- **Information.** Data assigned with a meaning.
- **Knowledge.** The ability to assign meaning.