FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Security-Schätzung von Open Source Webanwendungen

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering and Internet Computing

eingereicht von

Tamás Hernádi BSc
Matrikelnummer 0500374

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Privatdoz. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn. Edgar Weippl

Wien, 19. April 2015

Tamás Hernádi	Edgar Weippl

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Estimating Security in Open Source Web Applications

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering and Internet Computing

by

Tamás Hernádi BSc
Registration Number 0500374

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Privatdoz. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn. Edgar Weippl

Vienna, 19th April, 2015

Tamás Hernádi

Edgar Weippl

Erklärung zur Verfassung der Arbeit

Tamás Hernádi BSc
Alserbachstraße 6/12A 1090 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 19. April 2015

Tamás Hernádi

Danksagung

Mein Dank gebührt meiner Familie und meinen Freunden, die die Entstehung dieser Arbeit durch viel Verständnis und gutes Zuhören unterstützt haben. Ein besonderes Dankeschön an alle Tester, Korrekturleser und selbstverständlich meinem Betreuer Privatdoz. Dr.techn. Edgar Weippl für seine wertvolle Hilfe in Form von Hinweisen und Feedback.

Acknowledgements

My thanks go to all my family and friends who supported the creation of this work, for their patience and for listening to the evolving stages of the underlying concepts. Special thanks to all testers, proofreaders, and of course, my advisor Privatdoz. Dr. Techn. Edgar Weippl, who provided extremely helpful guidance and feedback.

Kurzfassung

Viele kleine und mittelständische Unternehmen außerhalb des EDV-Sektors beschäftigen oft nur eine Person, die für "die EDV" zuständig ist. Vorgesetzte teilen ihr daher vielfältige Aufgaben zu, wie beispielsweise die Erstellung einer Firmenwebseite. Da Kostenersparnis hinter solchen Aufträgen steht, erhält sie kein Budget für Ausgaben. Folglich sind kostenlose Open Source Lösungen unter den wenigen verbleibenden Alternativen.

Sicherheitsprüfung von Software ist ein komplexes Themengebiet. Will sich eine EDV-bedienstete Person ohne Erfahrung mit Sicherheitsprüfungen aufgrund von Sicherheitseigenschaften zwischen zwei Open Source Webanwendungen entscheiden, benötigt sie dafür geeignete Werkzeuge. Leider gibt es keine kostenlos verfügbaren Werkzeuge, die alle Anforderungen einer solchen Person erfüllen. Artikel über die Sicherheit von Webanwendungen, die im Internet veröffentlicht werden, unterliegen oftmals voreingenommenen Ansichten der Autoren oder sind für Personen ohne Erfahrung unverständlich verfasst. Folglich stellt eine Sicherheitsprüfung von Webanwendungen eine Herausforderung dar.

Diese Arbeit stellt eine Methode vor, welche es EDV-Bediensteten ermöglicht, von mehreren Alternativen die als am sichersten einzuschätzende Webanwendung zu wählen. Die im Vorfeld ausgewählten Produkte entsprechen den Firmenrichtlinien und die Person beherrscht die Grundlagen der Webentwicklung in der relevanten Programmiersprache und Technologie. Die Methode gliedert die vorgeschlagene Untersuchung der Sicherheit in nachvollziehbare Teilaufgaben und nutzt die zur Verfügung stehenden, kontextuellen Informationen der prüfenden Person. Zudem priorisiert sie die Aufgaben nach dem Kosten/Nutzen-Prinzip, um effektivste Zeitnutzung zu gewährleisten.

Informatikstudenten, die nicht auf Web-Sicherheit spezialisiert sind, haben eine funktionsfähige Implementierung dieser Methode getestet. Sie haben in etwa den selben Wissensstand im Bezug auf Web-Sicherheit wie jene EDV-Bediensteten, von denen diese Arbeit ausgeht.

Die erlangten Ergebnisse deuten darauf hin, dass die Methode und ihre Implementierung für die Untersuchung und den Vergleich von Open Source Webanwendungen im Bezug auf ihre Sicherheitseigenschaften gut geeignet ist. Die akkurate Arbeitsweise der untersuchenden Person spielt dabei eine große Rolle für die Qualität der Ergebnisse. Die erweiterbare Implementierung erleichtert die kontinuierliche Verbesserung der Methode.

Abstract

Many small and medium-sized companies outside the Information Technology (IT) sector employ a person responsible for IT "in general." Supervisors commission such IT individuals with a variety of tasks, such as, creating a company website. Given that cost considerations often result in such assignments, management allocates no budget for expenses. This leaves free open source web applications as one of the rare alternatives.

Security evaluation is a complex field of expertise. There are no freely available tools that cover all the requirements of an IT individual without security experience to choose between open source web applications. Security appraisal on the Internet is exposed to author bias, or it is not understandable to an average IT professional. Consequently, a security-based decision for a web application is difficult to make.

This work proposes a method, called EstSecure, that supports IT professionals in finding such a security-based choice. It helps them determine the estimably most secure web application from a preselected set of candidates. It is assumed that the IT professional has basic web development knowledge. Then, EstSecure divides the security evaluation into manageable tasks, leverages user-provided context information, and ranks the tasks based on a cost-benefit ratio to achieve maximum time efficiency for the user.

IT students not focused on IT security tested the proposed method using its implementation. The students have approximately the same security knowledge as the assumed IT professionals. The results indicate that EstSecure and its implementation are suitable for evaluating and comparing open source web application security properties. They further imply that the rigor of the user is essential for the success of EstSecure. Extensible implementation will allow future work to further improve the method and continuously optimize its utility.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Motivation

According to the Austrian chamber of commerce [WKO14], 420 351 companies registered in Austria in December 2014 had fewer than 50 employees. This number accounts for 98.6% of all companies in Austria. Whereas every company has core competencies, many are not directly associated with IT. However, in today's society, businesses that operate fully without the use of IT are rare. It is common practice for small companies to employ a person responsible for the entirety of IT tasks.

There are many cases[1] where cost considerations make supervisors instruct IT employees to create a website for the company, rather than commission the work to an external contractor specialized in building websites. The desire to minimize expenses and the lack supervisor awareness leads to situations in which IT employees, without the specifically required resources, are expected to choose a secure solution and implement a website. Such IT individual has limited knowledge in web application security, no access to expensive security analysis tools, and a constricted time budget.

As the company Q-Success states in their web technology survey [QS15], 82.1% of all websites rely on PHP: Hypertext Preprocessor (PHP) as the server-side programming language. Controversially, there is no sufficiently efficient and freely available security evaluation tool for PHP that requires only low initial knowledge. This issue motivates the present work.

1.2 Problem Statement

This work references the group of IT employees (female or male) as IT-Persons. IT-Persons are responsible for IT in its entirety in one of the numerous Small and Medium

[1] At the time of writing, the author knew two distinct cases of this exact situation.

Enterprises (SMEs) – a responsibility they might share with other employees. Furthermore, IT-Persons receive the commission to implement a website for the company.

IT-Persons have at least basic knowledge in the field of web development; however, they do not necessarily have experience with security audits, or deep knowledge of web application security. Furthermore, IT-Persons have to decide which product to use without having a budget available, either for the software or for the tools involved in the process or creation of the website. Consequently, IT-Persons want to base the website on an open source solution without increasing company costs. Closed source products, even if free, can hamper security evaluation by nature because they prohibit source code evaluations. Furthermore, IT-Persons can presumably spend only limited time on the selection process; therefore, they are assumed to have already preselected open source candidates that match the company's requirements.

The problem this work aims to solve is the support of IT-Persons through the means of adopting an open source web application from their previously nominated software candidates, that is, the most secure software solution. The IT-Persons' expectations with regard to security are assumed to be realistic and in accordance with the limited time and resources available for the evaluation. Furthermore, IT-Persons should be able to defend their choice to management.

1.3 Definitions

The method elaborated in this work is called *EstSecure*, which is derived from the words *estimate* and *security*. A prototypical implementation is tested and verified.

IT-Persons are IT professionals for whom those assumptions stated in Section 1.4 apply. A *candidate* is one of the open source web applications that IT-Persons deem capable of fulfilling their requirements. *Criteria* describe evaluation tasks that, if passed, increase trust in the candidate.

1.4 Assumptions

This work makes the following assumptions of IT-Persons and their context. An IT-Person:

- wants do determine the most secure of their preselected candidates
- needs to justify the decision to management
- has basic web development skills
- preselected candidates that are programmed in a language of which they have basic knowledge
- has a tight time budget
- has no budget for purchasing software

- does not necessarily have security-related skills and experience
- has founded expectations on the outcome of a structured estimation with the given assumptions

1.5 Aim of the Work

This work shows that an appropriate method can aid IT-Persons. An environment for structured evaluation allows IT-Persons with the above assumed circumstances to estimate and compare the security of open source web applications. For IT-Persons, this work makes the assumptions listed in Section *1.4 Assumptions*. A solution built on an extensible method can be optimized to further improve the capability of solving real-life problems.

1.6 Methodological Approach

A broad literature research covers a variety of relevant aspects of the web security area. The conceived and proposed method supplies a structural frame for the security evaluation of web applications. In a meeting with an expert on Internet security who works at an Austrian Certificate Authority (CA) company, we evaluated the first draft of the concept. The meeting supported EstSecure with ideas for extensions and suggestions for further reading.

EstSecure integrates the most positive aspects of related approaches studied in the literature research. It became obvious that the EstSecure method requires implementation as a web application to unfold its full potential.

With the concepts finished, a paper prototype test appraises the conceived mechanism's utility. In the test, the main objective is to determine whether EstSecure works for people without comparable security knowledge, i.e., without having conducted a similar literature research. The paper prototype test validates the underlying concept's suitability. The next steps refine the previously worked out method.

Comparing available and potential platforms results in a favorite expected-to-be-best-suited solution. Additional refinement improves the method. Holding a meeting with a senior web developer in order to discuss the method and receive expert opinion on the rationality of the concept regarding implementation helps confirm the planned concepts.

In the next phase, the implementation of a basic prototype provides insight into the problem domain. As recommended by Agile method experts, the initial prototype is discarded after fulfilling its purpose. Starting over "from scratch" leads to a fully functional version of the web application, thus making the EstSecure method testable.

Recently defined evaluation criteria allow IT professionals not specialized in security to conduct a structured assessment of web application security.

Open source candidates that deliberately contain security vulnerabilities should show the developed method to be effective. The candidate's preparation includes best practices learned from the multiple publications referenced in Chapter *2 State of the Art*. IT

students not specialized in security test the EstSecure method based on its implementation to verify usefulness.

1.7 Structure of the Work

Chapter *2 State of the Art* introduces related publications relevant for this work. The chapter also provides annotations on how the referenced materials are relevant for this work. Next, Chapter *3 Methodology* details the methodology that led to the solution to the given problem.

Chapter *4 Suggested Solution* first explains the suggested estimation method derived from the related work, and the concepts and terms used in the remainder of the work. The next section *4.2 Implementation* elaborates the most relevant aspects of the method's prototype implementation. Finally, the last section *4.3 Verification* of the chapter describes the two types of user tests conducted, and the corresponding results found in the validation phase.

Chapter *5 Critical Reflection* critically reflects on the present work and illuminates the limitations of the same. After that, Chapter *6 Summary and Future Work* provides an outline of possibilities for the future to extend the proposed approach and further improve the implementation's utility. The last section *6.2 Conclusion* of the chapter concludes and summarizes this work.

CHAPTER 2

State of the Art

Software security is a broad field of knowledge, yet it is not the only area important for solving the given problem. After elaborating background information on security, the associated risk management and security evaluation sections of this chapter further introduce related work on Agile methods and usability engineering.

2.1 Background

In Open Web Application Security Project (OWASP) Application Security Desk Reference (ASDR) [Mil08], Militelli offered a compendium of security-related issues and controls. The extensive collection of vulnerabilities supports readers with a profound overview on the matter. A focal point of the work is the description and enumeration of known attacks to software systems, including web applications. Although the book dates from 2008, OWASP continuously updates their web-based content. More recent versions of vulnerability and attack descriptions are available at [OWAb] and [OWAa].

The OWASP Top 10 [WW13] enumerates the ten most critical web application security risks. Injection related issues (e.g., SQL Injection), Broken Authentication and Session Management, and Cross-Site Scripting represent the top three most critical security risks in 2013. A range of organizations contribute measured vulnerability prevalence data to the project. In addition, the project supplies consensus estimates of exploitability, detectability, and possible impact of vulnerabilities. Combining them with the prevalence data using a variant of the OWASP Risk Rating Methodology, the authors established a well-founded overview on the current relevance of vulnerability classes. This work taps into this real-world data and adapts the Top 10's used method for more precise evaluation of vulnerabilities and security properties in the present context.

2.2 Security Evaluation

In order to measure security, metrics are essential. Mellado et al. [MFMP10] studied a variety of security metrics and models that originated from a range of standards and organizations. The standards Mellado et al. categorized in their work include International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC) 15408 and ISO/IEC 27004, as well as Common Weakness Enumeration (CWE) and Common Vulnerability Scoring System (CVSS). Furthermore, Mellado et al. presented a framework for security assessment in the design phase of Software Development Life Cycle (SDLC). Such framework contains multiple elements that are sufficiently generic in nature to be applied to the process of comparing the security properties of already released software.

In their work, Vaidyanathan and Mautone [VM09] compared two web-based open source Content Management System (CMS) solutions from a security perspective. They chose a CMS in the traditional sense with the name Mambo and the forum software vBulletin. In addition, Vaidyanathan and Mautone created a list of security evaluation criteria to compare and inspect the security properties of the two CMS products. Because of the suitable representation of properties, the selected CMS can be compared clearly. The present work adapts this used scheme of comparing candidates in a table of relevant criteria. It can incorporate many of the criteria defined by Vaidyanathan and Mautone.

Similarly, Meike et al. [MSW09] made a comparison between two open source CMS solutions: they decided in favor of Joomla and Drupal. In addition to describing vulnerability types and the consequences of their exploits, Meike et al. introduced the used method. After installing CMS, they applied penetration testing tools to the systems under test to evaluate their behavior. Upon finishing this dynamic test, Meike et al. conducted a code review to verify the developer's efforts in validating the user alterable input values before they reached sensitive sinks. The table format with which the authors opted to organize the criteria and candidates proved similarly useful as that from the aforementioned work by Vaidyanathan and Mautone.

Curphey and Arawo [CA06] provided a thorough overview on security assessment tools. The authors categorized them into types, and underscored the fitting lifecycle phase and recommend skill level for the utilization of each type. In addition, Curphey and Arawo outlined their benefits and limitations. Their overview assists with making the decision of whether to apply tools in a given phase of a software's lifecycle. However, the article dates back to 2006. Many of the freely available tools are no longer current. Nevertheless, the Rough Auditing Tool for Security (RATS)[1] scanner is an interesting tool for vulnerability detection in future work. However, the tool's behavior of reporting different types of possible vulnerabilities at once conflicts with the principle of isolated vulnerability search that this present work follows. The suggested tools for dynamic evaluation state interesting incorporation options for future work.

Mao [Mao09] charted a high-level road map for security-testing web applications. He

[1] `https://code.google.com/p/rough-auditing-tool-for-security/` (accessed 2015-04-14)

advocated a process that uses static and dynamic evaluation tools. Shi et al. [SCY10] also emphasized the utility of evaluation tools in their work, where they examined five software products and highlighted the methodical limitations of each. Detecting vulnerabilities in outdated server modules and software versions is important in general; however, it is beyond the scope of the present work. The authors underlined the utility of commercial vulnerability detection software. Although this work assumes that the EstSecure users do not have a budget assigned, it still advises using commercial software, provided that it is available to the users.

With the OWASP Testing Guide, Meucci et al. [MKC08] provided a sound framework and guidance for security-testing web applications. Among others, the topics of their work cover the what, when, how, and why to test. While clearly promoting security tests throughout the entire software development process, they also counseled on the testing methods suitable for project phases after implementation. Meucci et al. characterized the strengths and weaknesses of each security testing method. Currently, the Testing Project Team at OWASP is working on an updated version of the Testing Guide available on their project page [OWA14b]. The Testing Guide allows readers to explore web applications for vulnerabilities. However, because the Testing Guide aims to be complete, it is extensive in volume: there are 374 pages in the version for 2008. The approach this current work proposes considers the user's available time and prioritizes the security checks to be applied to their software candidates. Because of the similar purposes of the Testing Guide and the method proposed in the current work, future versions of EstSecure will incorporate the accomplished vulnerability discovery methods of the Testing Guide with slight effort.

Van der Stock et al. [VdSCC+08] presented further excellent work published as the OWASP Code Review Guide. They emphasized that many serious security problems cannot be detected by means other than reviewing source code. Certain tools can perform static security analysis on source code; however, they cannot understand the context. Therefore, a human must always verify the tool's findings. The Code Review Guide lists an ample compilation of vulnerability types that the reviewers should search, and explains the background of a security source code review. However, as with the Testing Guide, the volume of the work is high, which hampers the efficient work of people on a constricted time budget. Again, future versions of this presented work will assimilate the checks elaborated in the Code Review Guide.

Howard [How06] promoted a process for performing security reviews of source code. In his work, he informed on the signs that reveal those parts of the source code that should be evaluated first, such as old code or code that manages sensitive data. Future work will apply this prioritization process, but the current version already implements his recommendation for a good source code editor for quick browsing, performing look ups, and navigating the code structure.

Thomas et al. [TXX11] manufactured realistic test subjects for benchmarking security scanners. In their work, they deliberately hid well selected vulnerabilities in an e-commerce software. After mutant creation, the security scanners checked the fault injected versions of the software system. This present work applies the approach to generate vulnerable

versions of those web applications on which EstSecure's implementation and content is tested. In addition, Thomas et al. inspired to prefer real-world applications over trivially-sized code examples for isolated vulnerability search. They argued that a method's scalability can only be shown using larger systems under test.

In 2009, Walden et al. [WDWW09] publicized their work in which they questioned the significance of traditional software metrics, such as code size, cyclomatic complexity, code churn, and nesting complexity in relation to vulnerability density. The authors used a commercial security source code scanner to analyze an array of open source web applications in the time frame between summer 2006 and summer 2008. Warden et al. found a significant, but small, correlation between the change in Static Analysis Vulnerability Density (SAVD) and the individual aforementioned metrics. Based on the assumption that those projects that advertise their security effort tend to improve in security over time, they proposed a novel metric. The Security Resource Indicator (SRI) metric asks four key questions with regard to the security resources of the systems under test. The software candidate websites can answer these questions. With the data collected by the authors, they found a strong correlation between SRI and the change in SAVD over the given time frame. The present work depends on SRI for an approximate estimate for users with succinct available time.

2.3 Usability

Bevan [Bev05] investigated available studies that witness the cost benefits of usability engineering. Among numerous other arguments, he cited studies that report reduced task time and increased productivity. Given that the users of this present work are assumed to have limited time available, saving time on their end is crucial for EstSecure's success.

In her work, Kappel [Kap05] organized a range of usability testing topics to produce a sound overview. She contrasted the different usability test settings, materials, and the test methods. In particular, the advantages and disadvantages sections highlighted valuable considerations for usability test planning.

In 1994, Retting [Ret94] argued for a low-fidelity prototyping method that used paper prototypes. Among the advantages are the low cost that allows designers to test many ideas with end users. Furthermore, he observed the behavior of test persons to provide more open feedback compared with using higher definition prototypes that obviously require more time to develop.

Vijayan and Raju [VR11] argued for a method that utilizes disposable paper prototypes to acquire domain knowledge, establish system understanding, and elicit requirements for a system in planning. In the present work, a paper prototype helps understand the challenges associated with the general approach that uses a web-based system for security evaluation management.

Nielsen [Nie01] contributed the usability engineering method with the name heuristic evaluation. Heuristic evaluation involves a small number of experienced usability engineers who evaluate the user interface of an application against a recognized set of usability

principles. The method depends on the observation that different people tend to find different usability flaws.

In their German work, Harms et al. [HSS02] presented a list of usability evaluation criteria for websites, which they called Web Usability Index (WUI). According to the authors, their method is not suitable for the in-depth usability evaluation of websites. However, Nielsen stated in the previously referenced publication that a usability checklist can be immensely useful when conducting a heuristic evaluation. Many of the WUI criteria are considered during the creation of the present work. A full heuristic evaluation that searches usability flaws is among the future plans for the web application prototype that implements the EstSecure method.

2.4 Risk Management

The vast amount of existing vulnerability types, possible attack scenarios, and lack of time that IT-Persons are assumed to have make prioritizing security activities essential. In the article [OWA14a], which is part of the OWASP Testing Guide, the authors described a simple, yet efficient, method for assessing risks. Their method relies on the well-known formula $Risk = Likelihood * Impact$. Moreover, dissecting the otherwise difficult-to-appraise factor *Likelihood* to threat agent and vulnerability factors make the model more tangible and comprehensive. The present work derives a method from the OWASP Risk Rating Methodology using the data available from the OWASP Top 10 project.

Mellado et al. [MFMP10] compared a number of security metrics, and delivered an overview of covered security aspects and considerations. The CVSS metric offers a standardized punctuation of vulnerabilities, and it is an open system that allows vendors and security researchers to generate comparable values from publicized vulnerabilities. The MITRE Common Vulnerabilities and Exposures (CVE) database website [MIT99] grants access to CVE and CVSS entries. However, because of filtering options and visualization, accessing the data via the cvedetails.com website [Ö10] seems more convenient and efficient.

Security researchers often publish the vulnerabilities they discover. Making available the information on the vulnerability of a software system used by a given company implies additional risks for the business. Potential attackers are more likely to exploit a vulnerability in a deployed system if they know it exists. Allodi and Massacci [AM13] investigated the risk associated with vulnerabilities found in software deployed by a given company. They found indicators that suggest the market share of a software can be a valuable addition to the CVSS score. This insight for an approximate estimate based on CVSS and the market share information available online improves the proposed method.

2.5 Agile Methods

In the second edition of their book, Beck and Andres characterized and improved the Extreme Programming (XP) software development methodology [BA04]. A characteristic of XP is to bring best practices of software development to extreme levels. In their work,

Wolf and Bleek [BW10] provided an overview of several Agile software development methods. User stories mark desired features from an end user's perspective, and are a tool for approximately calculating effort and structuring the work. Wolf and Bleek further emphasized the need for adapting a process to the current software project. In 2008, Agarwal and Umphress [AU08] argued for a method to adapt the XP method to single-person teams. They conceived a procedure for organizing relevant planning and programming-related tasks. Pair programming, i.e., continuously writing code with two programmers at a single workstation, is a core principle of XP. However, Agarwal and Umphress agreed with Wolf et al. [WRL05] in that XP can also work without pair programming, but the inherent, immediate code review benefit is lost. Wolf et al. advertised the need for an exploration phase in which the developers dive into the domain topics and produce a prototype to evaluate concepts. The authors advised discarding the prototype after this phase. Although it is tempting to continue improving the prototype to a finished application, a prototype in XP is not intended for such purpose. Starting over from scratch one can heed the insights gained from the prototype. This way, it is possible to circumvent the prototype's design or architectural shortcomings. The authors further introduced the concept of acceptance tests. Ideally, end users or customers write such tests that encapsulate functional requirements. It is possible to use acceptance tests as a form of specification for the application. Developers can automate acceptance tests to reduce the effort of running them. Whereas unit tests check internal components, acceptance tests verify if the demanded features are correctly provided to the user.

In his work, Martin [Mar11] shared advice and a set of best practices based on his extensive experience as a developer. Among others, he advocated automated acceptance tests. Such tests influence the availability of features toward the user positively if they serve as regression tests throughout implementation.

Janzen [JS05] explained Test Driven Development (TDD) as a development approach that requires the developer to define test cases before writing productive code. He underlined that TDD is more about analysis and design than it is about testing. Such method makes the implementer continuously think about the code structure in advance while answering the prevalent question of how to test the code to be written. Applied at full extent, this method makes the developer write productive code only if there is a failing test case, and to stop writing code as soon as the test case succeeds. According to Beck [Bec01], changes related to the Graphical User Interface (GUI) do not necessarily need to be tested using test cases. This present work supports this exception because unreasonably high effort would be required to write a test case before making any small change to the web application's output.

Meszaros [Mes07] examined the refactoring of test cases in his work. He contributed a collection of patterns and best practices for refactoring unit test cases. Having the list of patterns available during refactoring and writing test cases shows benefits throughout the implementation phase of the current work.

2.6 Software Patterns

Evans and Fowler [EF00] presented a pattern for implementing specifications. The object-oriented implementation allows building a tree structure that represents the formula. Object references map Boolean operators between terms. For evaluation, one passes a context object to the root node of the formula tree, which starts a recursive evaluation in the child-objects. This way, using dependency injection via the passed object, evaluating the formula becomes a simple task. An additional advantage of the approach is that a simple implementation can immediately signal those terms of the formula that are not satisfied – a property exploited in this current work.

CHAPTER 3

Methodology

3.1 Literature Research

As assumed in the introduction, IT-Persons have very limited time, limited security knowledge, no experience conducting security audits, and no budget for purchasing new software of any kind. Understanding the problem domain is essential for supporting IT-Persons with their decision processes. The OWASP documentation projects constitute an appropriate starting point for gaining domain knowledge. As Section *2.2 Security Evaluation* discusses, this includes the OWASP Testing Guide [MKC08], OWASP Code Review Guide [VdSCC+08], OWASP Top 10 [WW13], and OWASP ASDR [Mil08]. Furthermore, the OWASP documentation projects studied comprise the OWASP Guide for building secure web applications [vdSWCS05], OWASP Comprehensive, Lightweight Application Security Process (CLASP) [CWFW06], OWASP Software Assurance Maturity Model (SAMM) [C+08], and OWASP Backend Security Project [Pel08]. The publications studied in addition to those discussed in Section *2.2 Security Evaluation* are well written and enlighten the problem area. However, this work cannot reuse their content in its context.

In addition to the OWASP materials, the literature research further covers a range of publications also listed in Section *2.2 Security Evaluation*. In particular, the studies that compare open source web applications from a security perspective are extremely valuable [VM09], [MSW09]. Their table-based representation of security criteria and the software candidates strongly influence the method presented in this work.

A spreadsheet for saving citations and notes along with a numerical appraisal of the read work's relevance captured important status information throughout the work process. In addition, Portable Document Format (PDF)-reading software that is capable of adding highlights and comments to the PDF proved its utility. The two mentioned methods saved time and maintained the overview on the studied materials.

3.2 Developing a Method

The aim of this work is to estimate the most secure solution from a given set of open source web application systems. This inevitably involves comparing multiple software candidates from a security perspective. Ultimately, a method is required that is suitable for comparing the security aspects of software candidates, which is further elaborated in Chapter *4 Suggested Solution*.

A meeting with Christian, an employee at a security-related Austrian CA company, evaluated the first set of ideas from a basic concept version. After the expert with practical experience in security issues at customer sites verified the concept, the next conceptual phase starts.

The initial plan would have provided IT-Persons with a written guide in a printable, textual format. Such approach would have contained different chapters, listings, and tables for classes of time budgets available for IT-Persons. However, later conception deemed the document-based approach to be at risk of ending with too much information to read and browse. The solution needs to be applicable quickly and efficiently. A document-based approach would not offer inherent advantages over the highly useful and already existing work, for example, from the OWASP foundation. Moreover, the approach would require IT-Persons to capture the current state of evaluating the distinct software candidates. For this, using an Excel worksheet or another manual method would demand discipline, while causing additional error-prone efforts.

Updating a printable document tends to be troublesome as well because of the (downloadable) nature of PDF documents. Publishers cannot update downloaded copies on users' computers. In a quickly changing and evolving area, such as Internet security, this would cause additional problems.

3.3 Planning a Web Application

Because of the considerations made in Section *3.2 Developing a Method*, the plan foresees the integration of the EstSecure method into a web application. Chapter *4 Suggested Solution* describes this proposed method. A usable prototypical implementation then shows that the EstSecure method works.

Using an existing platform is superior to developing an entire application from scratch. A suitable platform among the considered possibilities extends the concept. Refining the concept and further exploring the already existing platforms lead to a more precise vision of the desired solution.

Stefan, an experienced senior web developer, kindly agreed to a meeting. The meeting's participants discussed the growing list of ideas for the concept and the vision for implementation. The meeting's focal point was to ensure the basic planned solution is feasible to implement.

A paper prototype test evaluated the concept of this work. In the test, a prepared model ("mock-up") of the user interface design constitutes the base framework. A sheet of paper contains lines drawn manually that outline the user interface background and

layout. Adhesive repositionable notes (similar to Post-it® notes) simulate ("mock") dynamic elements of the web application, including dialogs, texts, tables, and more. This allows simulating the dynamic nature of a web application by relocating adhesive notes on a layout-background. Waved lines replace most of the text, excluding headings, in the paper prototype. Using low-fidelity prototypes and communicating the early stage of the design draft encourages testers to share their feedback and ideas. The gracious help of Viktoria, an IT-professional who works in a large logistics-related company, allowed the paper prototype test to occur. Narrating the text represented by the waved lines provided Viktoria with insight into the application. Each of her questions indicated a potential lack of information in the web application. Each resulted in a journal entry and influenced later phases of the project.

The paper prototype test helped approve the design draft, and eliminated several usability problems in this early design phase.

3.4 Developing a Web Application

To test EstSecure in a realistic setting, implementing it in a prototype is necessary. Section *4.2.2 Providing Required Backgrounds* further explains the rationale behind choosing MediaWiki[1] as the platform for implementing the method. This choice implies an implementation in PHP using MySQL as database backend.

Components of Agile methods improve the software development process of this work. Using TDD provides inherent benefits: testing extensively and embracing the change [BA04] that later insights induce.

The vast amount of unit tests[2] found the confidence to change aspects of the web application as required during the programming phase. Because adding new features introduces errors to the system, such errors immediately become obvious through failed unit tests. This type of regression testing proves invaluable. The project uses PHPUnit[3] as the testing framework. However, there are tasks difficult to implement using PHPUnit tests. Testing whether a mock object's method has *not* been called with an argument that matches a regular expression is troublesome with PHPUnit. Because the Mockery framework[4] offers mock objects with this type of test available, it complements the PHPUnit test cases. At the cost of a slight decrease in performance, Mockery compensates immensely by reducing the code writing effort for the same results.

Automated acceptance tests ([BW10], [WRL05],[Mar11]) continuously verify the correctness of user-visible features. The Selenide framework[5] eases implementation of this type of tests. Selenide is based on the Selenium framework[6]. On its own, Selenium is suitable for realizing planned acceptance tests. However, Selenide simplifies the definition

[1] `http://www.mediawiki.org/wiki/MediaWiki` (accessed 2015-02-14)

[2] A total of 637 unit tests with approximately 6,000 Source Lines Of Code (SLOC) for approximately 7,200 SLOC production codes at the time testers evaluated the solution

[3] `http://phpunit.de` (accessed 2015-02-14)

[4] `https://github.com/padraic/mockery` (accessed 2015-02-14)

[5] `http://selenide.org/` (accessed 2015-02-14)

[6] `http://www.seleniumhq.org/` (accessed 2015-02-14)

of test cases by hiding those details that a programmer does not need to consider, such as, writing code to open and close the browser that executes the automated tests. Selenide allows the use of Cascading Style Sheets (CSS) and XPath selectors to query Hyper Text Markup Language (HTML) element properties and content. Elements selected this way can be checked against expectations. Compared to Selenium, Selenide works with more concise code for the same purpose.

The combination of PHPUnit/Mockery unit tests for the application's internals and the Selenium/Selenide acceptance tests for the user-visible behavior of the system is extraordinarily useful. Despite the additional effort of maintaining the test cases current, the gain in trust in the application correctness is well compensated.

A project diary, as suggested by Wolf et al. [WRL05], monitors important events and insights during the practical phase. Noting the difficulties and errors along with the solutions saves valuable time with, for example, reoccurring errors caused by the work environment or the relied on frameworks.

The search function of the spreadsheet software that searches for current error messages enormously simplifies troubleshooting a broad range of errors. The open source Agile project management tool Agilefant[7] organizes user stories and iterations of the implementation phase. It records each iteration along with its user stories; this includes maintenance tasks such as planned code refactorings. The software helps retain overview on the project, and allows controlling of the progress. Used Selenide and unit tests outnumber the productive parts of the web application in lines of code.

At the start, a prototypical implementation explores the problem domain, thus heeding the advice of Wolf et al. [WRL05]. The rudimentary implementation is sufficient to gain insight into the impending problems. One of the issues encountered is the Object-Relational Mapping (ORM) Application Programming Interface (API) MediaWiki made available for accessing the database. As the prototype grows, it quickly becomes unmaintainable.

After the prototype has served its purpose, implementation of the prototype for demonstration starts. Such prototype uses an external ORM framework that proved to be a good choice, as elaborated in Chapter *4 Suggested Solution*.

After implementation of the web application is complete, it requires content. The next step of the process develops sufficient content to test the practical value of the underlying method.

3.5 Testing the Method

Adrian, a Linux system administrator, was once commissioned to implement a company website for his direct supervisor. He is the archetype of an IT-Person in the meaning of this work. In a meeting with him, we discussed the previously prepared draft of tests. The tests aim to prove the practical value of the proposed method for IT-Persons in the assumed context. Heeding Adrian's valuable input on the topic allowed us to improve the two previously derived tests for appropriate test persons.

[7] `http://agilefant.com/` (accessed 2015-02-14)

The objective of the first test is to show the method's utility for IT-Persons with almost no time available for a security-based comparison of open source candidates. In this test, a scenario briefs the user by indicating that they have two hours to choose one of two candidates, Typo3 and WordPress, based on security aspects. The chosen candidate will be the CMS that serves as basis for the company website. The test directs the user to the web application that implements EstSecure without further information. PIWIK[8], a web analytics software, logs the user's actions. Upon completing the test, all testers submit their reports along with the answers to a short questionnaire.

In the second test, the method guides inexperienced users through the process of estimating security on two open source candidates. Originally, the testers should have tested code samples for vulnerabilities. However, as Thomas et al. indicated in their work [TXX11], trivially sized code samples bias the outcomes in such tests. Consequently, the tests choose two real-life CMS solutions with different traits. WolfCMS[9] is a simple CMS that uses a database backend to store and retrieve pages created by an administrator. Monstra[10] is a flat file open source CMS that relies on no database to run because it stores all data in an Extensible Markup Language (XML) file-based storage engine. The first concept was intended to use old versions of software solutions that comprised real vulnerabilities. However, the work of Thomas et al. [TXX11] suggested a better approach. They adopted fault injection to deliberately hide security vulnerabilities in software systems. The procedure benchmarks Source Code Analyzers (SCAs), but the same strategy can evaluate a manual approach of vulnerability testing. Fault injection allows to precisely control the type of vulnerabilities that the testers can find. I addition, the test does not depend on real-world software to contain vulnerabilities. Again, a scenario briefs the test persons, and requests them to compare the security properties of the two deliberately insecure software candidates. The briefing scenario instructs testers not to disqualify a candidate in which they find a vulnerability as the method would otherwise recommend. Again, PIWIK documents the testers using the EstSecure implementation, and the testers once more submit answers to another questionnaire. In addition, the testers are asked to capture their comments directly in the EstSecure application to immediately document their decisions.

After preparing the tests, a small website announces the search for testers. A short description that lists the requirements and an outline of the benefits advertise the search to potential testers. The promised benefits are gained experience and a small incentive. The announcement consists of German and English versions.

Publishing the Unique Resource Locator (URL) along with a short description to the Informatik Forum[11] and Facebook should acquire testers. The social environment shares the advert in order to increase its visibility on social media. Sadly, recruiting testers proved difficult. Section *4.3 Verification* informs on the test results.

[8] `http://piwik.org` (accessed 2015-02-14)
[9] `https://www.wolfcms.org` (accessed 2015-02-14)
[10] `http://www.monstra.org` (accessed 2015-02-14)
[11] `http://www.informatik-forum.at` (accessed 2015-02-14)

CHAPTER 4

Suggested Solution

4.1 Estimation Method

This work uses the term *candidate* for an open source software solution selected by IT-Persons based on the specifications implied by the individual's context. They deem the software capable of fulfilling their company's requirements. The term *project* refers to the process of IT-Persons selecting the (presumably) most secure candidate from multiple choices. This work aims to support IT-Persons with their projects, while assuming such individuals have limited time, no budget for software of any type, basic web development experience, and do not necessarily have security audit expertise. Section *1.4 Assumptions* itemizes the complete list of assumptions on IT-Persons.

A clear and understandable structure for the procedure guides an individual through a previously unknown area. Similarly to related work in Section *2.2 Security Evaluation* ([VM09], [MSW09]), defining criteria facilitates this. Each criterion encapsulates instructions to examine a given candidate for a type of Vulnerability, or to increase trust in a candidate's security in other ways.

4.1.1 Sub-Criteria

Vulnerabilities in Internet security can be manifold. Creating a single criterion, for example, to find all possible Structured Query Language (SQL) Injection Vulnerabilities, might result in long and complex descriptions. Vulnerability types tend to have a broad range of relevant aspects and peculiarities. Long criterion descriptions might inevitably contradict the goal of time efficiency for IT-Persons.

Instead, *sub-criteria* differentiate between distinct manifestations of a given vulnerability type. Starting with the easily accomplished items, IT-Persons should first search for obvious security flaws. In the example of SQL Injection, a criterion instructs to search for function/method calls prone to SQL Injection that receive user-changeable variables as argument. For this, the criterion provides a shell command that searches

for those patterns that match the vulnerable method calls. The shell command and the criterion's short description allow IT-Persons to quickly determine whether there such obvious flaws exist in the candidate's source code. It is important for IT-Persons to immediately determine whether there are no vulnerabilities of the currently investigated type present, i.e., the shell script should unambiguously display the empty result set. Gradually, the sub-criteria become more generic and time-consuming as they yield more general outcomes. This iterative approach allows EstSecure to quickly disqualify those candidates that contain easy-to-find vulnerabilities.

4.1.2 Properties

EstSecure requests IT-Persons for contextual information. The questions asked are divided into three classes. *User properties* ask for details regarding the IT-Person, such as whether they have conducted a security audit previously. *Project properties* inquire on the circumstances of the project. This class asks questions similar to the following:

- How much time does the IT-Person have for the security decision?
- Does the deployed solution need to be accessible from the Internet?
- Does the IT-Person have access to a commercial security SCA for this project?

The third class of properties is called *candidate properties* and request basic data on individual candidates. Examples for this class of properties include technical questions, such as whether the candidate depends on a certain programming language, or whether a specific database backend is used.

Contextual data captured this way is essential for a range of components used in the EstSecure method. The remainder of this chapter explains them.

4.1.3 Criteria Ranking

IT-Persons have limited time, thus making time efficiency a key success criterion. For this reason, EstSecure needs to suggest an efficient order for the evaluation tasks.

Simplified, an approach based on cost-benefit ratio ranks criteria. The term *benefit* refers to an abstract value of trust that a candidate earns if applying a criterion to it shows no indicator for the presence of vulnerabilities. This is relative to other candidates being evaluated by the same criterion. *Effort* refers to the estimated number of minutes required to check a candidate based on a criterion, thus representing the criterion's cost in the cost-benefit ratio.

4.1.3.1 Static and Dynamic Modifiers

Benefit and effort of a criterion may depend on properties in two ways. A predefined value can change the benefit or effort value if a Boolean formula (Section *4.1.4 Conditions*) based on user context properties evaluates to true. This type of *static modifier* can consider a necessary set-up task, for example. When the task is done, a property captures

the information, and the dependent formula evaluates to true. Then, the modifier reduces the criterion's effort in minutes; this is defined as a static value at the modifier's creation. Section *4.1.4 Conditions* discusses the concept of Boolean formulas in this work's context.

On the other hand, *dynamic modifiers* change the benefit or effort by a value calculated using a contextual property-based formula. Dynamic modifiers also use *conditions*. This approach can, for example, modify the effort estimated for code review to require based on the SLOC of the candidate source code.

4.1.3.2 Benefit-Cost Ratio

Equation 4.1 calculates Benefit-Cost Ratio (BCR) for a criterion.

$$BCR(r, C, T) = \frac{\sum_{i=1}^{c_n} Benefit(r, c_i, T)}{\sum_{i=1}^{c_n} Effort(r, c_i, T)} \tag{4.1}$$

The equation first calculates the sum of the benefit and effort values using the criterion r and the set C containing all candidates $c_1 \dots c_n$ of an IT-Person's current project. In addition, this calculation integrates the collection of contextual properties denoted as T. Subsequently, a division of the two values results in the average benefit/effort ratio of the criterion and all candidates. To order the criteria by BCR, a single BCR value is required for each criterion. Considering the average is important for cases where considerably different values occur for the distinct candidates. This can occur if, for example, one wants to conduct a manual source code review, and one candidate contains an order of magnitude more source code than the others.

4.1.3.3 Benefit

The benefit of a single candidate c for a criterion r with respect to the user context properties T uses Equation 4.2 that denotes the sub-criteria of r as $r'_1 \dots r'_n$.

$$Benefit(r, c, T) = BBase_r + \sum_{j=1}^{m} BMod_j(r, c, T) + BFormula(r, c, T) + \sum_{i=1}^{r'_n} Benefit(r', c, T) \tag{4.2}$$

Section *4.1.2 Properties* introduces the contextual properties T . $BBase_r$ represents the base value for the benefit of criterion r. All the m static and dynamic benefit modifiers for the (criterion,candidate)-tuple use the set of contextual properties T to calculate the sum $\sum_{j=1}^{m} BMod_j(r, c, T)$ that aggregates the benefit modifiers to the given expression. $BFormula(r, c, T)$ refers to the outcome of the *benefit formula* for the current (criterion,candidate) pair and context. Section *4.1.10 Benefit Formula* elaborates the details of the *benefit formula*. The final version is denoted in Equation 4.12. The expression $\sum_{i=1}^{r'_n} Benefit(r', c, T)$ states the sum of sub-criterion benefits.

4.1.3.4 Effort

Similarly, Equation 4.3 uses the same notation as Equation 4.2 and calculates the effort of a criterion:

$$Effort(r, c, T) = EBase_r + \sum_{j=1}^{m} EMod_j(r, c, T) + \sum_{i=1}^{r'_n} Effort(r', c, T) \tag{4.3}$$

Compared to Equation 4.2, $EstFormula(r, c, T)$ is missing. The *benefit formula* only appraises the gain in trust implied by the absence of a vulnerability pattern. Consequently, the benefit formula only calculates benefit values, as Section *4.1.10 Benefit Formula* explains. Other means are required to estimate the effort of a criterion, such as an educated guess from a security expert that creates criteria.

4.1.3.5 Sub-Criteria Ranking

The ranking (Section *4.1.3 Criteria Ranking*) of a criterion includes its sub-criteria. This leads to fewer time consuming content switches for IT-Persons because they do not have to read the criteria descriptions of many vulnerability types in an alternating fashion.

For example, SQL Injection is the first criterion, and it combines efficient and less efficient sub-criteria. On average, the sub-criteria testing for SQL Injection are more efficient than the sub-criteria searching for path traversal. Nevertheless, EstSecure suggests evaluating even the less efficient SQL Injection sub-criteria before the most efficient of the path traversal sub-criteria. This reduces the number of context switches for IT-Persons.

EstSecure currently considers SQL Injection as more important (it has a higher benefit value) than checking for path traversal vulnerabilities. This is because the *benefit formula* respects the OWASP Top 10 project's data in which SQL Injection is the more severe vulnerability class. Section *4.1.10 Benefit Formula* describes the specifics of the formula.

4.1.4 Conditions

Many aspects of the EstSecure method depend on the concept of *conditions* that use Boolean formulas. In EstSecure, these may consist of:

- the logical operators AND, OR, and NOT
- user contextual properties
- comparison operators
- static numeric values.

Based on the conditions, EstSecure can apply static and dynamic modifiers to effort or benefit values. In other cases, the conditions show (or respectively, hide) criteria not appropriate for the current project's candidates.

4.1.5 Candidate Scores

EstSecure monitors project progress using the *ratings*. IT-Persons may grant any (criterion,candidate)-tuple a rating from 0 to 10. Pairs not yet rated have an empty rating. If a rating is present, Equation 4.4 calculates the score candidate c receives from the evaluation of criterion r.

$$Score(r, c, T) = Benefit(r, c, T) * \frac{Rating(r, c)}{10} + \sum_{i=1}^{r'_n} Score(r'_i, c, T) \tag{4.4}$$

The *rating* scales the maximum benefit of the current (criterion,candidate) pair. The sum of optional sub-criterion scores is encompassed in the overall score.

Furthermore, EstSecure calculates the sum of scores for every candidate over all criteria, as Equation 4.4 defines.

$$ScoreSum(c, T) = \sum_{i=1}^{r_n} Score(r_i, c, T) \tag{4.5}$$

Criteria and sub-criteria are respected only if they are applicable (*4.1.7 Criterion Applicability*). Each criterion has hints for reasonable rating. Considering an example criterion, which should check password storage in the database, the criterion recommends these ratings:

- 0 if the candidate stores passwords in plain text
- 1 if the candidate stores passwords using the Message-Digest Algorithm 5 (MD5) hashing algorithm
- 2 if the candidate stores passwords using a suitable hash algorithm, but without using salt
- 10 if the candidate stores passwords using a suitable hash algorithm and per-user salt.

For this example, EstSecure specifies a *passing score* (Section *4.1.6 Candidate Disqualification*) in such a way that those candidates that store passwords as plain text or MD5-hashed passwords are discarded. A candidate that uses a suitable hashing algorithm, but no salt, results in a low score. However, EstSecure still considers the candidate to possibly be the most secure, unlike a candidate with plain text storage for passwords.

The suitable rating for a (criterion,candidate)-tuple can be ambiguous. For instance, let us consider a criterion that examines input validation of arguments for a shell command. The criterion advertises the rating 0 if blacklist-validation without previous input canonicalization is used, and 1 if blacklist-validation is done for a web application that first canonicalizes the input. If an IT-Person finds an instance of both mentioned flaws, the (criterion,candidate) pair should obtain the lower rating assigned, which makes the method's estimate as conservative as possible.

4.1.6 Candidate Disqualification

As already mentioned, IT-Persons are assumed to not be experts in security-auditing software. EstSecure implies that even with adequate support, such individuals might not be as successful finding security flaws as an expert in the field. Therefore, if such an IT-Person actually finds a vulnerability using their limited time and knowledge, the candidate probably has more vulnerabilities to be discovered. A candidate that contains vulnerabilities that can be detected by a novice reviewer in short time needs to be discarded as an insecure solution. Consequently, EstSecure advises IT-Persons to do so if they find a vulnerability.

In addition, a criterion may define a *passing score*. The passing score is a number that indicates the minimum score that a candidate must attain from an already rated criterion. If a criterion provides a passing score and a rating is available for the criterion with a candidate, the resulting score must be higher than the passing score. Otherwise, the candidate has to be discarded. This way, one can mark a class of ratings that disqualify candidates.

4.1.7 Criterion Applicability

Determining whether a criterion should be applied to the candidates can save time for IT-Persons. In EstSecure, four reasons exist that can prevent a criterion from evaluating candidates. First, if the benefit value is lower than the *hiding threshold*, the criterion is not displayed to the IT-Person at all. This can occur, for example, if the criterion benefit is reduced by a dynamic condition based on context data properties. Second, criteria with a BCR lower than the *disabling threshold* are considered inefficient. As criteria are sorted by their BCR (Section *4.1.3 Criteria Ranking*), they are placed at the bottom of the criteria table and marked as inefficient. The two mentioned thresholds are globally configurable parameters for the EstSecure method.

Whereas the benefit and effort of a criterion control the two aforementioned cases, the next two depend directly on user context properties. A criterion can have a condition attached to it that may disable it. This can induce a mixed set of candidates that satisfy (or respectively, fail) the condition. Candidates failing the condition do not need to be evaluated, and are automatically considered as succeeding the criterion with the highest possible rating. This concept ensures, for example, that IT-Persons do not check a single candidate without a relational database backend for SQL Injection. In the fourth case, no candidate satisfies the condition attached to a criterion. In this case, the criterion does not show at all. For example, IT-Persons do not need to worry about SQL Injection-related criteria if *none* of their candidates use a relational database backend.

4.1.8 Choosing a Candidate

The EstSecure method considers three possible endings for a decision project. First, all, but one, candidates are discarded. In this case, IT-Persons should select the only remaining candidate. Second, IT-Persons rate all candidates for all applicable criteria.

Consequently, IT-Persons should choose the candidate with the highest score. Third, if IT-Persons have exhausted their time budget and cannot continue the candidate evaluation, they should adopt the candidate with the current highest score.

4.1.9 Decision Documentation

Whereas EstSecure offers guidance for evaluating candidates using predefined criteria, it cannot foresee all candidate peculiarities. An advantage this method has over solely relying on automated analysis is the IT-Persons' ability to evaluate using contextual information not available at the time of criterion definition.

Because IT-Persons are bound by instructions, they might need to defend their choice to management. As prevention in this regard, IT-Persons should take notes for each criterion-candidate rating. Ideally, IT-Persons capture the relevant events: discovered locations in the candidate source code that influence the choice and context otherwise not available. In addition, any assumptions or thoughts are valuable for understanding the thought processes behind the ratings.

Taking notes is especially important if IT-Persons deviate from the suggested course of action, for example, they might rely on context unavailable to EstSecure, as stated before. For example, IT-Persons could opt to omit a criterion that considers SQL Injection if their candidate uses a relational database, but they might decide to use the product without a database and not use the features that require it.

4.1.10 Benefit Formula

Criteria benefit values are essential to EstSecure because they order criteria by time efficiency. Assigning a criterion benefit value based on experience is possible, and in many cases, also feasible. However, EstSecure aims to be extensible. Immediately after two criterion writers independently define criterion benefit values based on their individual experience, the criterion order could become disorganized. To prevent this, EstSecure includes a formula to approximate benefits in addition to an optional immediate estimate.

In the case of immediate estimate, criterion creators determine the benefit of a criterion as a numeric value based solely on their experience without using the formula. This approach is useful for criteria that do not directly consider a vulnerability class. For example, a criterion could calculate source code quality metrics to estimate the security of candidates or use a commercial SCA that considers a multitude of vulnerability classes simultaneously.

Criterion writers can opt to use either method or both combined.

4.1.10.1 Base Formula

The formula-based approach quantifies the benefit based on a particular (sub)class of vulnerabilities. EstSecure adapts the OWASP Risk Rating Methodology formula [OWA14a] to suit its needs. The rating methodology is based on the well-known risk formula:

$$Risk = Likelihood * Impact \tag{4.6}$$

In the OWASP method, questions segregate the risk calculation. Each question has predefined answers that correspond to a numeric value from 1 (least severe) to 9 (most severe). Likelihood consists of four *threat agent* factors and four *vulnerability* factors. Four *technical impact* factors and four *business impact* factors influence impact in the formula above.

Likelihood is the average of threat agent and vulnerability factors. If answers to business impact questions are available, impact in the above formula is equal to the average of their values. Otherwise, the technical impact questions contribute to the average. This reflects the preference of business-related needs and risks over merely technical considerations.

4.1.10.2 OWASP Top 10 Variant

The variant of the formula that ranks the vulnerabilities in the OWASP Top 10 [WW13] maps each question-based factor from 1 to 9 to one of the three values of high (1), medium (2) or low (3). Using these values, they calculate the average of the likelihood factors (respectively, technical impact and business impact factors). The resulting averages are decimal numbers between 1.0 and 3.0, inclusively.

The basic Risk Rating Methodology is intended for programmers who estimate the severity of their vulnerabilities. However, the Top 10 ranks global vulnerability classes. The basic method applied by programmers to their code might have access to threat agent and business impact data. The variant of the Top 10 project has no access to that data because it surveys the most critical vulnerability classes. Obviously, a global generic method cannot respect the peculiarities of every possible problem instance.

To mitigate this, the Top 10 project ranks its vulnerabilities without factors for threat agents or business impact, and leaves it open to readers to adapt the table using their particular circumstances for those fields.

Furthermore, the Top 10 project method uses different factors that influence the likelihood factor. Because they rank global vulnerability classes, they cannot access the contextual information of concrete vulnerabilities. Consequently, the Top 10 method removes the likelihood factors that the vulnerability does not influence directly: awareness, intrusion detection, skill level, motivation, opportunity, and group size.

The Top 10 project adds the likelihood factor *prevalence* to the original Risk Rating Methodology. They derive prevalence data for vulnerability classes from statistics supplied by security research companies. This renders the prevalence factor a valuable data source otherwise barely available. Aside from the rest of the factors, one prevalence entry in the OWASP Top 10 2013 has the value 0. The next section explains why that is a class more severe than *high* (1).

The Top 10 project recommends using the more severe value of the two alternatives *technical impact* and *business impact*. The value should then be multiplied by the calculated likelihood average. Equation 4.7 shows the formula.

$$Risk = Likelihood * Min(TechnicalImpact, BusinessImpact) \tag{4.7}$$

The formula results in a value for the calculated risk between 1.0 (highest risk) and 9.0 (lowest risk). The main overview table in the Top 10 orders vulnerabilities according to this formula.

4.1.10.3 Quantifying Risks

EstSecure requires each criterion to have a benefit value directly proportional to the risk associated with the vulnerability for which the criterion is testing. Consequently, criteria testing for the vulnerabilities with the lowest numerical values (i.e., the highest risk) need to have the highest benefit values. If a criterion can increase the confidence that vulnerabilities for a certain class are absent, the benefit value needs to be in relation to the associated risk.

To achieve this ranking, EstSecure requires a function that derives distinct benefit values from the numeric risk values. The OWASP Top 10 variant of the formula derives a value for the technical impact (1 to 3) from the associated questions $q_1 \dots q_n$ using Equation 4.8. Each answer can have a value from 1 to 9.

$$TechnicalImpact = \lfloor \frac{9 - \frac{\sum_{i=1}^{q_n} q_i}{n}}{3} + 1 \rfloor \tag{4.8}$$

The depicted formula is inappropriate for this present work. For example, the OWASP Risk Rating Methodology asks four questions of the *vulnerability* factors. One such question is: "Ease of exploit: How easy is it for this group of threat agents to actually exploit the vulnerability? Theoretical (1), difficult (3), easy (7), automated tools available(9)". Assuming one answers all four questions with a response that corresponds to the value 7, the average is obviously 7. If the question above is answered with *automated tools available (9)* instead of *easy (7)*, the average for the vulnerability factors increases from 7 to 7.5 . Using the above formula, the technical impact is still 7 because the OWASP variant always reduces the number using the floor function.

The goal of the Risk Rating Methodology is to order approximate classes of severity. However, EstSecure wants to use all available information. An arithmetical simplification such as the floor function is counterproductive. Consequently, EstSecure adapts Equation 4.8 to a version that does not use the floor function, noted as Equation 4.9

$$TechnicalImpact = \frac{9 - \frac{\sum_{i=1}^{t_n} t_i}{n}}{3} + 1 \tag{4.9}$$

This formula yields values between 1.0 and $3.\overline{6}$ for technical impact and analogously the other parts of the risk formula (business impact and likelihood). In conjunction with the risk rating formula (Equation 4.7), the resulting overall risk for a criterion ranges from 1.0 to $13.\overline{4}$.

4.1.10.4 Linearization

EstSecure aims to derive a benefit value from the calculated risk value. However, the above detailed formula has an unfavorable property for this purpose. The values that indicate the highest risk are those closest to 1.0, whereas $3.\overline{6}$ represents the lowest risk level. Using a scaling function such as Equation 4.10, one receives a benefit of zero if the risk value is $13.\overline{4}$, which corresponds to the lowest possible risk. The highest possible risk represented by the value 1.0 equals to a benefit of 10,000.

$$Benefit(Risk) = 10000 - \frac{10000}{13.\overline{4} - 1} * (Risk - 1) \tag{4.10}$$

However, instead of a linear distribution, the present distribution is exponential because of the multiplication used. The resulting distribution is more dense at high values. Having high-risk criteria benefit values close to each other is unfavorable. IT-Persons evaluate criteria starting from the highest benefits. Even a minor benefit or effort modifier could change the criteria order.

To address this issue, EstSecure linearizes the risk before scaling, which leads to Equation 4.11.

$$Benefit(Risk) = 10000 - \frac{10000}{\sqrt{13.\overline{4}} - 1} * (\sqrt{Risk} - 1) \tag{4.11}$$

This measure ensures equal difference of benefit values of two criteria if their risk value difference is the same.

4.1.10.5 Minimal Benefit

The above Equation 4.11 yields a benefit of zero for a criterion that considers a vulnerability of lowest possible risk. The *hiding threshold* hides any criterion with such a low benefit value. However, IT-Persons can also consider criteria with the lowest risk class (and lowest benefit value) if they have time to do so. Hiding the criteria would make such consideration impossible. To mitigate this, EstSecure introduces a minimal benefit for criteria with the least severe possible risk value ($13.\overline{4}$), and states the final benefit formula as Equation 4.12.

$$Benefit(Risk) = MaxBenefit - \frac{MaxBenefit - MinBenefit}{\sqrt{13.\overline{4}} - 1} * (\sqrt{Risk} - 1) \tag{4.12}$$

This version of the formula assigns a benefit value to criteria for which the formula-based approach is suitable.

Section *4.1.5 Candidate Scores* considers the concept of *ratings*. Using the rating than an IT-Person assigned to a (criterion,candidate)-tuple, the EstSecure method determines the criterion-candidate score for comparing the security estimates of candidates.

4.1.11 Criteria and Benefits

Section *4.1.10 Benefit Formula* elaborates the formula used to calculate the criteria benefit values. Currently, however, the derived formula only considers the static aspects of the vulnerability. It still needs to include the IT-Persons' context that can influence the likelihood and impact of the risk formula.

EstSecure relies on a combination of the likelihood factors from the OWASP Risk Rating Methodology and OWASP Top 10. From the OWASP Top 10, it takes the factors *exploitability*, *detectability*, and *prevalence*. Whereas *exploitability* corresponds to *ease of exploit*, and *detectability* corresponds to *ease of discovery* from the Risk Rating Methodology, *prevalence* is unique to the Top 10 project. As already mentioned, *prevalence* is distilled from real-world statistical data collected by companies that conduct security research.

Technical Impact	**Business Impact**	**Likelihood**	**Threat Agent**
Loss of confidentiality	Financial damage	Awareness	Skill level
Loss of integrity	Reputation damage	Intrusion detection	Motivation
Loss of availability	Non-compliance	Exploitability	Opportunity
Loss of accountability	Privacy violation	Prevalence	Group size
		Detectability	

Table 4.1: Risk factors in EstSecure

The criteria definition procedure checks the OWASP Top 10 to determine whether the publication contains information on the current vulnerability class. If such case, EstSecure finds the likelihood factors based on the Top 10 entries: exploitability, prevalence, and detectability. The next step is to determine what the values 1, 2, or 3 (high, medium, and low) correspond to on a scale from 1 (low) to 9 (high) used by the OWASP Risk Rating Methodology questions.

Table 4.1 summarizes the factors on which the EstSecure method relies.

The type of sub-criterion in creation influences this quantification. In addition, EstSecure considers the open source property of candidates. This increases the detectability of vulnerabilities compared with closed source candidates where potential attackers do not have the application's source code available.

In the Top 10 project, a vulnerability class with exploitability *average* means, simplified, that the *exploitability* question was rated between 4 and 6 (as Equation 4.8 shows).

While adding criteria, the values assigned to the factors in Table 4.1 require one of the values on a scale from 1 to 9. Inverting a surjective function (such as Equation 4.8) carries a degree of uncertainty, and consequently, allows for interpretation.

For example, EstSecure is searching for an obvious SQL Injection vulnerability. For this, it searches the code using a pattern that would match

mysql_query("SELECT * FROM users WHERE id = " . $_GET['id']);

because a user-changeable PHP superglobal variable is accessed within the argument of a MySQL function. Changing such a variable is trivial for the user. According

to EstSecure, one would set the detectability factor to 6 (or even higher) because a potential attacker can access the source code of the application, and could trivially find a vulnerability such as this. This equals to the maximum value Equation 4.8 can categorize in the *average* rating SQL Injection received from the Top 10 project.

In addition to the likelihood factors, the Top 10 project publishes a technical impact value for each listed vulnerability class. EstSecure maps the technical impact severity class to one or more of four technical impact factors. EstSecure considers the vulnerability class and the type of technical consequences that an exploit can have. This influences the technical impact factors that are assigned values.

For example, it is uncertain whether exploiting an SQL Injection vulnerability results in *loss of accountability* – this depends on the individual circumstances of the vulnerability's occurrence. However, an exploit of such a vulnerability would most likely affect the candidate's confidentiality. Consequently, *loss of confidentiality* is one of the factors that must contribute to the risk value. In an unclear case, the criterion only includes those factors that can be estimated. EstSecure ignores missing elements when it calculates the factor average. Otherwise, missing information would bias the outcome based on present values.

In addition, the current criterion can consider the vulnerability manifestation for which it is searching. The structure of the search pattern may disclose the severity of a vulnerability. For example, if a criterion is searching queries such as

mysql_query($_POST['query']),

where an SQL query string is received from a visitor's browser, a complete database takeover is possible: no restrictions on the query's structure would apply. Such circumstances can be considered by assigning an even higher value to the relevant factor.

4.1.12 Benefit Formula and User Context Data

The OWASP Top 10 project recommends to consider the reader's situation. The reader is responsible for estimating the business impact and threat agent factors. EstSecure can access user context data using the properties provided by IT-Persons. Using a *condition* (Section *4.1.4 Conditions*), a formula modifier can override some, or all, of the risk formula factors. For example, if IT-Persons deploy a system accessible via the Internet, the potential group of attackers is larger than with deployment in an Intranet environment. The benefit formula can respect this circumstance with a condition and formula modifier that increase the likelihood factor *size* [of attackers] if satisfied.

4.1.13 Using Available Resources

A key goal of the present work is to achieve the best possible time efficiency for IT-Persons. This target is best approached using every available support they can obtain. EstSecure uses criteria for this purpose; if IT-Persons have a security SCA available, they are encouraged to run it on the candidates. If IT-Persons have a *commercial* SCA available, they should exploit the product to their advantage. Depending on the tool and security requirements of IT-Persons, running the commercial SCA alone might be sufficient.

Comparing the SAVD metric of the candidates [WD12] may be acceptable for some IT-Persons. As an alternative to the SAVD, IT-Persons could also compare the severity of the most critical reported issues. Having in-depth information from a commercial SCA run available could allow IT-Persons to entirely omit manual evaluation. This type of situation suggests to entirely omit the use of EstSecure and its implementation in order to save time. EstSecure encourages this for the sake of the goal of time saving.

If no commercial SCA is available, but the candidate is programmed in the procedural development style in PHP, the freely available RIPS scanner [Dah11] is a viable option. It cannot scan object-oriented code; however, it can scan procedural code automatically. IT-Persons should run the scanner in such a case, and investigate the findings for faster evaluation. As elaborated in Section *4.1.7 Criterion Applicability*, a criterion may also evaluate only some of the candidates, while omitting others.

4.1.14 Criterion Content

Criteria and sub-criteria instruct IT-Persons to check their candidates for certain vulnerabilities. The way they instruct them to do so has only few limitations. Every (sub)criterion has to be understandable by the IT-Person to whom it appears. In addition, EstSecure has the option of displaying hints if said IT-Person answered questions that indicate an inability of understanding (sub)criteria without further hints. Obviously, EstSecure allows creating properties that inquire about the security-related knowledge, and apply conditions to hide criteria too complex for a user. Tables summarize rating hints and the circumstances for which they are intended. Having advice available increases user evaluation efficiency.

4.1.15 Short Estimation

EstSecure aims for time efficiency. Nevertheless, some IT-Persons might not even have a few hours to evaluate and compare candidate security. If IT-Persons aim to find a secure solution to build a website, choosing WordPress and regularly updating it yields sufficient security for many non-critical deployment scenarios. However, this work proposes a slightly more general approach that can help make a choice for applications different from regular websites. To achieve this, EstSecure includes a process for a crude estimate on software security if a user has virtually no time for an evaluation.

If IT-Persons declare a time budget of less than five hours, the method advertises using a short estimation alternative instead.

The SRI metric that Walden et al. [WDWW09] explained in their work fits seamlessly with EstSecure's concept of a quick and basic security estimate of a candidate. Examining candidate product websites for four key security resources leads to a simple metric. Walden et al. found that it significantly correlated to the change of SAVD open source project experience over time.

Another source of information for IT-Persons is the National Vulnerability Database (NVD) via the website cvedetails.com [Ö10]. Whereas IT-Persons can easily access the count of published vulnerabilities of a candidate in the previous year (for example), the

CVE entries alone are not sufficiently convincing. Unknown open source projects might not have any vulnerability publicized in the database, which does not mean they are free from vulnerabilities. Security researchers prefer evaluating some projects to others.

To alleviate this bias, EstSecure seizes the observation of Allodi and Massacci [AM13] that market share data states a valueable addition for CVE based estimates on security, and consequently integrates market share data into the estimate. For this, it asks IT-Persons to find the market share that the candidate has in their corresponding software type. The method recommends a Google search for the candidate name and the term "market share". This method finds websites that collect statistical information on web application market shares, such as w3techs.com [w3t15].

The method assumes an increase of security researcher attention proportional to the market share of open source web applications. Based on this assumption, it asks IT-Persons to divide the number of public CVEss of the previous year (for example) by the percent of market share. We assume that this simple metric is correlated with the SAVD of the candidate. Access to a commercial SCA could help verify this assumption.

For simplicity, the remainder of the work uses the name Vulnerabilities per Percent of Market share (VPM) for this metric and notes it as Equation 4.13

$$VPM = \frac{\#CVEs}{\%MarketShare} \tag{4.13}$$

4.1.16 Criteria Extensibility

EstSecure aims to be extensible and easy to adapt for the requirements that emerge from different fields of security evaluation. A criterion only states an abstract benefit value that represents the trust in candidates passing it. Combined with the directly estimable effort, specifying the criterion is completed. This loose definition makes criteria extraordinarily versatile. Criteria can, for example, check the code for bad programming practices, calculate code metrics, and even incorporate content from the OWASP Testing Guide or other sources.

4.2 Implementation

Section *4.1 Estimation Method* details the proposed method for estimating the security of open source web applications.

Test users cannot evaluate the method directly because time efficiency is a key aspect of it. Reading long descriptions contradicts this goal. A prototypical implementation closes the gap between testers and the method. This section briefly outlines the relevant aspects.

4.2.1 Used Frameworks

The prototype is a MediaWiki extension that relies on the Symfony Framework[1] for business logic, and the Doctrine Project[2] for database abstraction. The implementation uses TWIG[3] as its templating engine.

4.2.2 Providing Required Backgrounds

The security knowledge of IT-Persons is uncertain. This work aims to support any level of security knowledge. Consequently, it can neither lecture security backgrounds starting from basic levels, nor can it assume IT-Persons to already have all the enquired knowledge.

As a compromise, it leverages the advantages of a knowledge management platform such as those that MediaWiki can offer. For example, the proposed method uses the Semantic Glossary MediaWiki extension[4]. Relevant articles teach about background topics. The Semantic Glossary highlights the predefined terms that appear on any page of the web application. For example, if a reader hovers the term *salt* with the cursor, a tooltip appears with a one-sentence description of the term for IT-Persons with sufficient knowledge. In addition, the prototype displays a link to the associated article. The linked article indicates relevant aspects of the topic and references other resources for more related materials, such as OWASP.

4.2.3 Discarded Prototype

Honoring the advice of Wolf et al. [WRL05] and other related work, a simple prototype explored the problem domain. It was a MediaWiki extension that used direct queries to a MySQL database. During its use, the demand for a sound web development framework and ORM tool arouse. Use of the first prototype was discontinued after this exploration phase. Nevertheless, it eliminated several implementation-related difficulties of the method, and allowed clean implementation using the mentioned frameworks.

4.2.4 EstSecure Website

This current work demonstrates the EstSecure method using an implementation accessible at www.estsecure.net. After creating an account, IT-Persons can see new entries in the main menu located on the left side.

Hints, the first menu entry, leads to an article that summarizes relevant information with regard to the web application. After showing the short hints section, the system asks for user and project properties in the next two menu entries. The fourth menu entry, Candidates, allows IT-Persons to record their candidates by entering name, description, and answers to property-related questions in the system.

[1] `http://www.symfony.com` (accessed 2015-02-26)
[2] `http://www.doctrine-project.org` (accessed 2015-02-26)
[3] `http://twig.sensiolabs.org` (accessed 2015-02-26)
[4] `http://www.mediawiki.org/wiki/Extension:Semantic_Glossary` (accessed 2015-02-14)

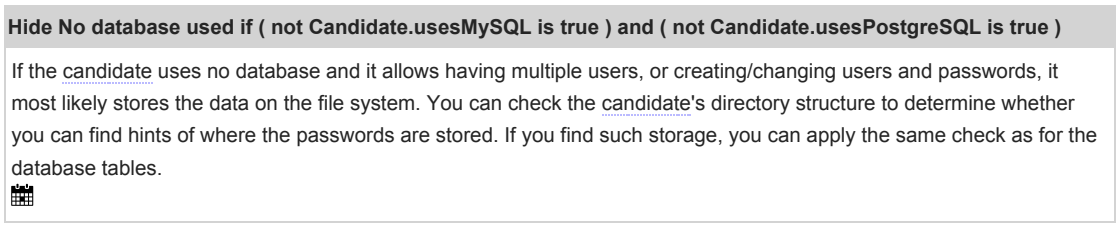

Figure 4.1: Optional description detailing its condition in administrator-mode

4.2.4.1 Scoreboard

After IT-Persons submit their user context data, the last menu entry, Scoreboard, presents a table of their candidates and suggested criteria ordered by BCR. At the bottom of the table, the resulting score highlights the sum of points the candidate earned by undergoing the criteria evaluation.

Each criterion-candidate cell in the table contains the sum of the score based on user-provided ratings, and the scores the candidate received for passing optional sub-criteria (as Section *4.1.5 Candidate Scores* informs).

To help IT-Persons navigate, each score opens a tooltip when hovering it with the cursor. These tooltips show the state of the rating process by using color codes. Red scores indicate that a candidate did not pass the criterion or one of its sub-criteria. They encourage IT-Persons to discard the candidate (as Section *4.1.6 Candidate Disqualification* explains). Orange scores indicate pending ratings; in other words, IT-Persons have not evaluated the candidate for the criterion or one of its sub-criteria, yet. Blue scores mark the (criterion,candidate)-tuples for which the tool already has a score. Because the tool can contain criteria that do not apply to all candidates (Section *4.1.7 Criterion Applicability*), omitted candidates are calculated automatically using the highest possible score, also displayed in blue. The tooltips of such scores hint to this circumstance.

4.2.4.2 Edit Rating

Clicking a score takes IT-Persons to the Edit Rating page. The criterion description follows under the candidate and criterion names.

Criterion descriptions can use *optional description blocks*. The user context influences a condition (Section *4.1.4 Conditions*) that determines whether a block of text is visible to them. This allows EstSecure to offer information relevant to some users while not cluttering the screens of others. Figure 4.1 depicts the version visible to administrators. It displays an otherwise hidden box along with the condition that needs to be satisfied for the description to show.

Below the description, a drop-down menu allows IT-Persons to select a rating from the options *undefined* and 0-10. A text area captures the IT-Persons' optional comments on relevant findings.

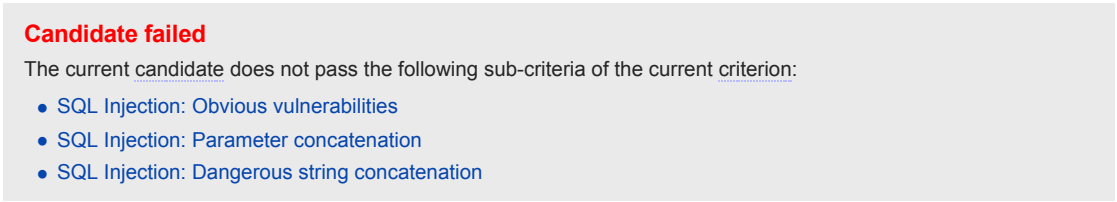

Figure 4.2: Candidate failing sub-criteria results in a warning message

There is a content box located at the bottom of the rating page that is expanded upon clicking the header. Such a box explains the calculation of the benefit, effort, and BCR values, each with and without sub-criteria. A table lists the benefit and effort modifiers with the corresponding condition, optionally enabling them. The colors green and red highlight those properties in the condition formula that are satisfied, or respectively, unsatisfied.

A further expandable element inside the calculation details block reveals internals of the benefit formula for the current (criterion,candidate) pair. If the criterion uses a benefit formula instance, it also itemizes the benefit formula modifiers depending on the conditions. Otherwise, the box contains the empty formula frame. Tooltips guide interested IT-Persons to descriptions of the used formulas.

The implementation can respect a *passing score* for criteria (as Section *4.1.6 Candidate Disqualification* elaborates), which disqualifies those candidates that receive a score below a predefined value. If a candidate fails a criterion or one of its sub-criteria, a content element with red and bold header warns the user that the candidate already failed a passing score. In the case of failing one or more *sub-criteria*, the box additionally summarizes the criteria that the candidate failed, as imaged on Figure 4.2.

4.2.4.3 Administration Backend

User accounts with administrative privileges in the underlying MediaWiki installation have access to an administrative area via the main menu of the EstSecure implementation. Administrators can edit the *criteria* following the menu entry with the same name.

Behind the link with the name *property types*, administrators can view, edit, and create project, user, and candidate properties. Properties have a readable ID that references their values in the current user context. Each property has a value type that determines whether users are expected to enter integer or Boolean values for the property. Optional *value constraints* restrict acceptable property values. They use Symfony Validation Constraints[5] available for many purposes, and may warn about errors with a customized message if a value violates a constraint.

Specifications brings administrators to an overview on the *condition* objects (Section *4.1.4 Conditions* describes them). The implementation refers to EstSecure conditions as specifications, and uses the specification pattern [EF00] to realize them. Without the

[5] `http://symfony.com/doc/current/reference/constraints.html` (accessed 2015-02-26)

pattern, implementing the evaluation of dynamically definable Boolean formulas that access contextual data would be a complex task.

The page behind the menu entry *Criteria Modifiers* lists objects of the named type. On the settings page of a modifier, administrators may define a formula that consists of a restricted set of characters. Such a formula can consist of PHP arithmetic, numbers, and references to user context property values. If the modifier employs user context properties, and/or PHP compatible arithmetic, it is called a dynamic modifier. In contrast, formulas that contain only numbers are static modifiers. Selecting a specification and criterion allows administrators to set adaptive effort and benefit values that are applied if the condition evaluates to true.

Optional Criterion Descriptions lets administrators provide a describing text that appears on the rating pages of the associated criterion if the attached specification evaluates to true for the current candidate.

The menu entry *Config* allows administrators to override the default values for the *hiding threshold*, *disabling threshold* (Section *4.1.7 Criterion Applicability*), and *MaxBenefit* and *MinBenefit* values used in the benefit formula (Section *4.1.10.5 Minimal Benefit*).

4.2.5 Extensibility

As Section *4.2.4.3 Administration Backend* indicates, EstSecure's implementation makes an administration interface available for efficient content extension.

The literature research discovered that the OWASP Testing Guide and OWASP Code Review Guide offer suitable content for integration as criteria and articles. The benefit formula is derived from the OWASP Risk Rating Methodology and the OWASP Top 10 project that itemize many vulnerability risk factor values. This similarity makes the integration effort direct.

4.2.6 Evaluation Environment

Howard [How06] demanded a suitable source code editor capable of quickly navigating the code for efficient reviews. Heeding his advice, the website supplies a VirtualBox[6] image that runs a Gentoo Linux[7] installation with K Desktop Environment (KDE)[8] and Eclipse[9]. If IT-Persons import a candidate as an Eclipse PHP project, its source code becomes mostly navigable. Limitations of the used PHP Development Tools (PDT)[10] apply, for example, using the weakly dynamic scripting language PHP, the environment cannot always determine the type of a variable reliably.

For many code review-related criteria, users need to find relevant (i.e., dangerous) pieces of code. For example, when looking for SQL Injection vulnerabilities, users want

[6] `https://www.virtualbox.org/` (accessed 2015-04-14)
[7] `https://www.gentoo.org/` (accessed 2015-04-14)
[8] `https://www.kde.org/` (accessed 2015-04-14)
[9] `https://eclipse.org/` (accessed 2015-04-14)
[10] `https://eclipse.org/pdt/` (accessed 2015-04-14)

to find exploitable function and method calls. In this case, this could mean calls that accept SQL queries with a PHP superglobal inside their argument. A tool such as grep can support this search. To avoid false negatives, one needs to be extraordinarily careful with the regular expressions used. Matching only the call and nothing else, i.e., excluding comments, whitespaces, etc., is challenging.

4.2.6.1 Abstract Syntax Tree Search

Fortunately, there is a tool to mitigate this problem. The sgrep utility from Facebook[11] is pre-installed in the VirtualBox image. Sgrep is a useful program capable of finding method calls on the Abstract Syntax Tree (AST) of the source code in many languages, one of which is PHP.

Inside the virtual machine, the evaluation environment relies on a shell script called *estgrep* that uses sgrep. The shell script finds all function/method calls and presents them along with corresponding file paths and line numbers. A series of grep and sed commands filter the list to only show calls with configurable characteristics. Criteria can specify method/function names (for example), and a regular expression that needs to be matched by each found call. In addition, an optional regular expression highlights parts of the results, and an additional argument can fine-tune sgrep behavior.

Every reported line in the terminal starts with an eclipse:// URL that includes file path and line number. If IT-Persons using the evaluation environment click one of the file paths, the shell opens Eclipse at the desired code location. The left CTRL-key needs to be held while clicking the file path. This feature allows criteria, or respectively, their shell commands, to quickly take IT-Persons to relevant places in the code. The Linux installation defines a custom protocol service to realize this.

Limitations of sgrep led to the definition of an additional shell command called *estfindincludes*. The script works similarly to estgrep and prints all variants of PHP *include* statements. As Figure 4.3 illustrates, the command marks variable names in the argument. Each argument in the output depicts a path to a PHP file to be included. Variables in such inclusions can quickly become dangerous and need to be checked.

4.3 Verification

Two user tests verify the usefulness of the EstSecure method. The first test uses a scenario in which a given IT-Person chooses the more secure CMS from two fixed candidates under severe time pressure.

The second test applies fault injection to candidates, similar to the method Thomas et al. [TXX11] presented. The testers know that the candidates deliberately contain vulnerabilities, and that they should find them.

All testers were volunteering IT-students not focused on security, and were allowed to discontinue the test at any time.

[11] `https://github.com/facebook/pfff/wiki/Sgrep` (accessed 2015-02-14)

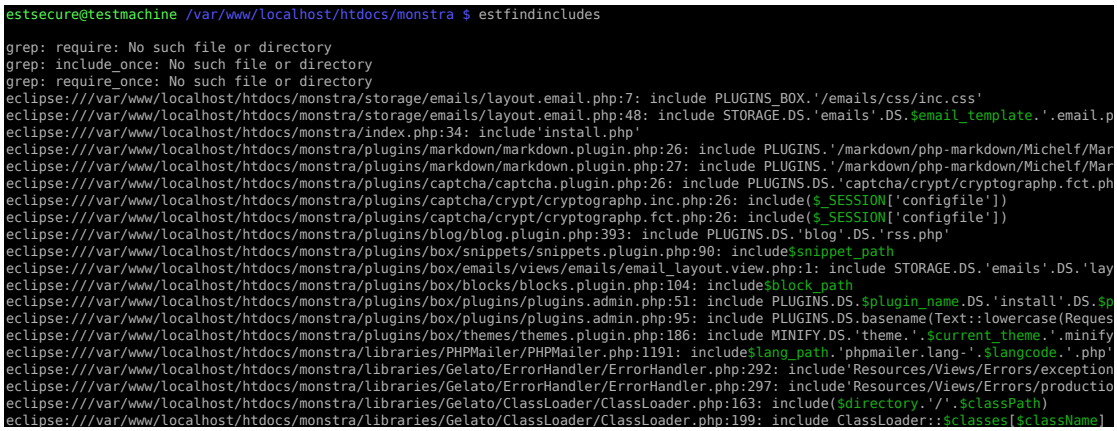

Figure 4.3: Sample output using estfindincludes

4.3.1 Short-time Test

Section *4.1.15 Short Estimation* describes a process for short and basic security estimates for candidates. A test with seven testers evaluates the practicality of the method.

4.3.1.1 Expected Results

First, the testers determined the SRI metric for the candidates. At the time of writing, Table 4.2 summarizes the expected examination results.

	WordPress	**Typo3**
Security URL	present	present
Security e-mail address	present	present
Vulnerability list	present	present
Secure development documentation	present	present

Table 4.2: Expected findings for the SRI metric

According to the summary, both WordPress and Typo3 have an SRI of four. WordPress maintains security configuration-related documentation page[12]. They provide the e-mail address security@wordpress.org for reporting discovered vulnerabilities. The website lists patches and fixed vulnerabilities [13], and an extensive guide that offers coding standards[14]. A Wiki page[15] contains Typo3's security configuration information. Their

[12] `http://codex.wordpress.org/Hardening_WordPress` (accessed 2015-02-18)
[13] `https://wordpress.org/news/category/security/` (accessed 2015-02-18)
[14] `https://make.wordpress.org/core/handbook/coding-standards/php/` (accessed 2015-02-18)
[15] `http://wiki.typo3.org/Security` (accessed 2015-02-18)

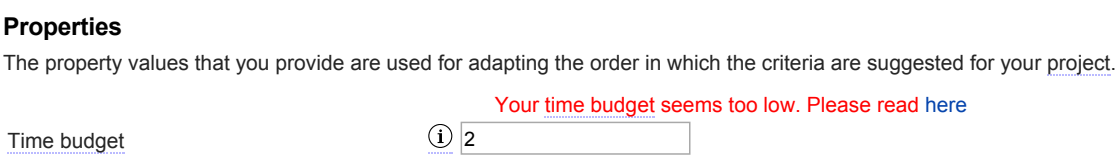

Figure 4.4: Old version of the low time budget warning

security e-mail address is security@typo3.org. Their website also presents a vulnerability list[16] and coding standards[17].

The authors of the SRI metric [WDWW09] only described each of the resources with a short sentence. Consequently, the metric is subject to the reader's interpretation, and other URLs may also be acceptable for the metric's questions.

For the market share-based metric VPM, an evaluation of market shares found a market share of 60.7% for WordPress and 1.6% for Typo3. The w3techs.com website[18] compiled this statistic. An alternative source for market share data[19] reports 66% for WordPress and 2% for Typo3.

WordPress had 29 vulnerabilities published in 2014[20], whereas Typo3 had 11 in the same time frame[21].

Calculating the vulnerabilities per percent market share leads to VPM 0.48 for WordPress, and VPM 6.88 for Typo3 if using the numbers from w3techs.com. In the case of data from opensourcecms.com, the VPM values are 0.439 and 5.5, respectively. Based on these simple estimates, testers are expected to prefer WordPress to Typo3.

4.3.1.2 Tester Results

4.3.1.2.1 Tester S1 This report uses identifiers instead of tester names to retain the tester's privacy. S1, the first tester on the short test, found the short estimation instructions not as clear as assumed. If a given IT-Person enters a time budget lower than five hours in the current implementation, they receive the error message depicted in Figure 4.4.

However, in the case of S1, this message proved insufficient. An interview suggests that S1 interpreted the error message as a dissent between the test scenario and the implementation. S1 continued the normal path of evaluation adding candidates and beginning to check the criteria.

[16] http://typo3.org/teams/security/security-bulletins/ (accessed 2015-02-18)
[17] http://docs.typo3.org/typo3cms/SecurityGuide/ (accessed 2015-02-18)
[18] http://w3techs.com/technologies/history_overview/content_management (accessed 2015-02-18)
[19] http://www.opensourcecms.com/general/cms-marketshare.php (accessed 2015-02-19)
[20] http://www.cvedetails.com/product/4096/Wordpress-Wordpress.html?vendor_id=2337 (accessed 2015-02-18)
[21] http://www.cvedetails.com/product/6858/Typo3-Typo3.html?vendor_id=3887 (accessed 2015-02-18)

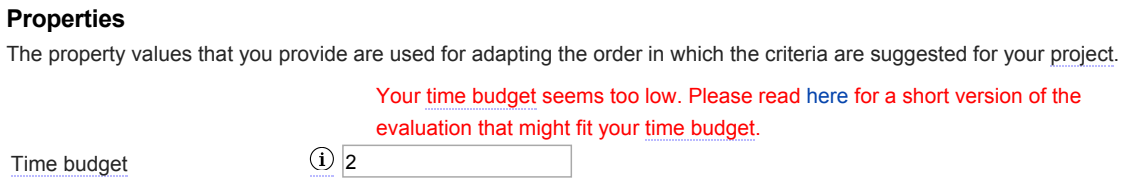

Figure 4.5: Improved and verbose version of the low time budget warning

To prevent S1 from wasting time, the test supervisor intervened and emphasized the importance of the aforementioned error message. The interview further reveals that S1, in fact, cursorily read the page linked in the error message, but did not consider it worth reading. To stress the importance of the error message for future users, it was changed to the variant visible on Figure 4.5. The page explicitly indicates that the linked page contains a shortcut. When asked to heed the page that describes the shortcut variant, S1 calculated the VPM metric. S1 noted a market share of 66% for WordPress and "approximately 6%" for Typo3. For the vulnerabilities, S1 noted 24 for WordPress and 14 for Typo3.

A talk after the test identified a problem: S1's browser blocks JavaScript by default, and the market share website S1 used (opensourcecms.com) requires JavaScript to generate a pie chart of market shares. Without the diagram, only usage count numbers were present on the page. The appearance of the website without JavaScript did not indicate missing content. Consequently, S1 believed the market share to be approximately 6% based on the displayed usage count numbers. Although S1 reported having been on the correct page of the website cvedetails.com, the researched vulnerability numbers did not match previous expectations of the test setting. However, these difficulties did not influence the correctness of the results found by the tester. S1's VPM values favor WordPress, which S1 opted to declare the more secure candidate.

When asked, S1 stated not feeling comfortable reading long passages of text in general, and that this circumstance might have influenced the findings. S1 did not calculate the SRI metrics for the website. The description page S1 cursorily reviewed contained the instructions for both short estimation methods.

4.3.1.2.2 Tester S2 S2 interpreted the property *time budget* as asking for an estimate of how much time S2 *might need* for the work. Because of this assumption, S2 did not enter the two hours derived from the test briefing. Consequently, S2 did not receive the error message that points to the shortcut description.

To alleviate the issue, the order of fields in the property form was changed. The hint icon is now directly next to the property name. Hovering the property name now also displays a tooltip that contains a novel description of the property's purpose. In addition, the error message now explicitly states that the shortcut estimate does not require the user to enter information on their candidates (Figure 4.6).

S2 did not finish the test and chose neither of the candidates.

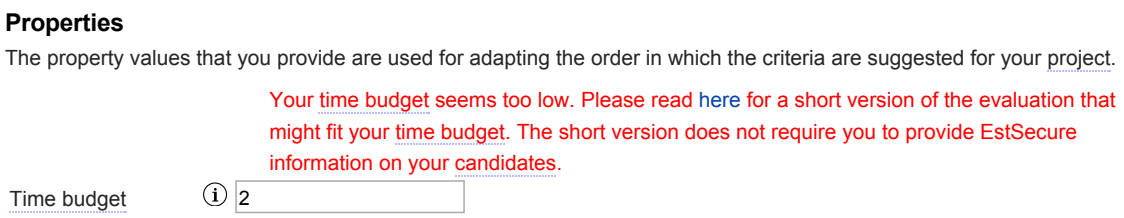

Figure 4.6: Low time budget warning now explicitly excludes candidate creation

4.3.1.2.3 Tester S3 Another tester, referred to as S3, had issues similar to S1. By analyzing the information PIWIK logged for S3's test, it was revealed that S3 provided the required low time budget, and read the error message. Furthermore, S3 spent 2.5 minutes on the shortcut solution page, which is sufficient to at least cursorily read the page. After that, instead of calculating the required metrics, S3 proceeded to provide information on the candidates. S3 further estimated the criterion rankings based on Google search results for the candidates, which was not intended by the test setting. In the questionnaire, S3 stated Typo3 to be more secure because of a tutorial on how to prevent path traversal vulnerabilities using the system. The VPM and SRI metrics were not part of S3's submitted results.

An interview with S3 made obvious the reason for not calculating the metrics, and conducting improvised evaluation. Similar to S1, S3 was confused when the system presented an error message, even when performing all tasks as requested by the test scenario. The error message is the reason S3 deviated from the suggested process. In addition, it was not clear to S3 what the metrics section, which S3 had read cursorily, required.

After S3's test, further refactoring the relevant content pages directed users to a novel Short Estimation page that described the procedure succinctly instead of the previously longer version. In addition to the error message that redirected users to the short estimation, a hint on the home page and the Hints page provided a link to the procedure.

4.3.1.2.4 Tester S4 S4 found an SRI of 3 for WordPress and 4 for Typo3. The VPM values were 0.43 for WordPress and 5.5 for Typo3. S4 used the website opensourcecms.com for the market share information and calculated the correct VPM values for the given data.

However, according to the PIWIK log, S4 accessed a variety of pages, including Scoreboard, after providing data on the candidates. This means that S4 first did not understand the hint to use the short estimation. In an interview, S4 said to first have randomly clicked the website pages, one of which was the Short Estimation page, but did not realize the process description as such.

Based on this statement, the hints that point to the Short Estimation page were further emphasized. In addition, the minimal description now displays a concise table

Short Estimation Process

Step 1)	Calculate the VPM metric for each of your software candidates (the **lower** it is, the better)
Step 2)	Calculate the SRI metric for each of your software candidates (the **higher** it is, the better)
Step 3)	Select and use the candidate with the best metrics

Figure 4.7: Prominent itemization of steps in the Short Estimation process

that itemizes the three steps of the Short Estimation process prominently. Figure 4.7 depicts it.

Furthermore, S4 was not certain whether a high or low VPM indicated higher security. Consequently, hints were added to the table that describes the process.

When calculating the SRI metric for WordPress, S4 found the mechanism for reporting security incidents on the website. However, S4 considered the link "hidden too deeply" in the page structure, and did not consider that favorable for WordPress. This led S4 to assign an SRI of 3 to WordPress, instead of the expected value of 4.

S4 declared WordPress to be the more secure candidate based on the calculated metrics.

4.3.1.2.5 Tester S5 S5 found and understood the Short Estimation process. According to PIWIK tracking, S5 directly accessed the Short Estimation page from the home page. Furthermore, S5 further viewed the Introduction page for 30 seconds. Subsequently, S5 loaded the VPM and SRI pages, and left the EstSecure website after 68 minutes.

S5 calculated the WordPress market share to be 63% based on the averages of the two data sources. For Typo3, S5 used the average of 1.8%. Searching the vulnerability count of the year 2014, S5 found 29 and 11 for WordPress and Typo3, respectively. These numbers match the test's expectations. With these values, S5 correctly calculated VPMs of 0.46 and 6.1 for WordPress and Typo3, respectively, after which S5 correctly concluded that the WordPress VPM indicates higher security than that for Typo3.

Calculating the SRI metric, S5 found both WordPress and Typo3 to earn a full score. However, S5 chose different URLs for the candidates. For the question of a secure configuration for WordPress, S5 found an article on secure WordPress setup[22] to meet the criterion. S5 responded to the request of a list or database of specific vulnerabilities with the WordPress Hardening article[23], which is a questionable choice because it provides an article on configuring WordPress to provide higher security, instead of a database of security issues. The WordPress theme review article [24] was S5's response to the question of whether secure development practices were available. Because the article provides links to guidelines and best practices, this is an acceptable answer.

[22] `https://codex.wordpress.org/User:Hakre/Secure_Wordpress_Setup` (accessed 2015-04-13)

[23] `https://codex.wordpress.org/Hardening_WordPress` (accessed 2015-04-13)

[24] `https://codex.wordpress.org/Theme_Review` (accessed 2015-04-13)

S5 found the Typo3 security cookbook[25] suitable for the secure configuration question. However, the security guide that S5 found is marked as outdated by the Typo3 security team. S5 found the page of the Typo3 quality assurance team[26] to provide the secure coding guidelines.

S5 utilized the leeway allowed by the SRI metric. Based on the metrics, S5 estimated WordPress to be more secure than Typo3.

4.3.1.2.6 Tester S6 S6 completed the test without difficulties, and reported VPMs of 6.1 and 0.46 for Typo3 and WordPress, respectively.

When investigating SRI, S6 found the Typo3 Security Guide[27] to match both the secure setup and coding practices. Furthermore, S6 reported the email address security@typo3.org as the security contact, and the Typo3 Security Bulletin[28] for the list of application-specific vulnerabilities.

Contrary to the test's expectations, S6 reported the application-specific vulnerabilities to be found at the WordPress CVEs page[29] that directs the reader to the original CVE website. S6 answered the secure coding best practices question with the plugin best practices page[30].

Consequently, S6 rated both WordPress and Typo3 with an SRI of 4. Based on the VPM value that favors WordPress, S6 chose this as the more secure candidate.

4.3.1.2.7 Tester S7 S7 found a VPM of 0.46 for WordPress, based on a market share value of 63%. For Typo3, S7 reported a VPM of 3, which resulted from a market share of 2% and six vulnerabilities. The interview made obvious that S7 used the Security Vulnerabilities list[31] instead of the Vulnerability Statistics page[32] suggested by a screenshot in the metric description. S7 assumed that the Security Vulnerabilities list would be sorted by publishing date, and counted the top entries for 2014 as six.

Deviating from the test's expectations, S7 reported the plugin vulnerabilities page[33] as providing a database of application-specific vulnerabilities. For the secure development practices, S7 reported no resources as being available on the candidate website. For the security of the Typo3 setups, S7 deemed no resources on the website as fully satisfying the requirements.

S7 rated WordPress and Typo3 with an SRI of 3 each. Based on the two calculated metrics, EstSecure suggested a preference for WordPress to Typo3.

---

[25] https://typo3.org/fileadmin/security-team/typo3_security_cookbook_v-0.5.pdf (accessed 2015-04-13)

[26] https://buzz.typo3.org/teams/qualityassurance/ (accessed 2015-04-13)

[27] http://docs.typo3.org/typo3cms/SecurityGuide/ (accessed 2015-04-14)

[28] https://typo3.org/teams/security/security-bulletins/ (accessed 2015-04-14)

[29] https://codex.wordpress.org/CVEs (accessed 2015-04-14)

[30] https://developer.wordpress.org/plugins/the-basics/best-practices/ (accessed 2015-04-14)

[31] http://www.cvedetails.com/vulnerability-list/vendor_id-3887/Typo3.html (accessed 2015-04-14)

[32] http://www.cvedetails.com/vendor/3887/Typo3.html (accessed 2015-04-14)

[33] https://wordpress.org/plugins/plugin-vulnerabilities/ (accessed 2015-04-14)

However, S7 found the VPM metric to not fit the requirements, and stated, “As a system administrator, I would be interested in the sum of vulnerabilities, and not in the market share. A higher count of vulnerabilities would imply more frequent patching.” Furthermore, S7 found Typo3 to subjectively meet the SRI requirements better than WordPress. Consequently, based on personal interpretation, S7 preferred Typo3 over WordPress, although the calculated metrics clearly favor WordPress.

4.3.1.3 Discussion of Results

EstSecure did not guide the first three testers adequately to calculate both metrics. S1 merely calculated the VPMs metric for the candidates and chose WordPress as the more secure candidate. S2 did not complete the test. S3 conducted an improvised evaluation and chose Typo3 without calculating the required metrics. These first three testers had difficulties extracting the required steps of the suggested evaluation procedure from the description.

Error messages are not appropriate for the regular flow of navigation; users seem to be overly distracted if presented to them. To circumvent this issue, an additional content element at a prominent place hints to the shortcut if the user is short of time before entering any values.

The page that describes the shortcut for the estimation also seems to not be sufficiently clear. For this, an additional heading *Limited Time – Quick Estimate* at the page that describes the shortcut states in a short paragraph, “If you have limited time available: calculating the VPM and SRI for your candidates is the *fastest offered option* for an estimate” and links to the metric descriptions. Behind the prominent link, a table that itemizes the three required steps in bold font guides users through the short estimation process (Figure 4.7).

The incremental improvements explained in this and the previous section illustrate EstSecure’s process for establishing a useful guide for estimation. Testers S4 to S7 could easily calculate the metrics and realized the method to suggest WordPress as the more secure candidate. The testers required approximately 1 hour each to form a decision. S7 opted to disregard the previously calculated VPM metric, and selected Typo3 as the preferred secure candidate.

4.3.2 Evaluation Test

The implementation has criteria and sub-criteria ready to test EstSecure in cases where IT-Persons have sufficient time. For the evaluation, vulnerabilities were hidden in the source code of two candidates, the open source CMS solutions WolfCMS, and Monstra.

4.3.2.1 Prepared Criteria

This section outlines the criteria covered by the test.

SQL Injection is a criterion that groups related sub-criteria. It merely hints at the sub-criteria to contain the checks.

4.3.2.1.1 SQL Injection: Obvious Vulnerabilities First, IT-Persons should check the candidates using the sub-criterion *SQL Injection: Obvious vulnerabilities*. A shell command uses the *estgrep* script (as Section *4.2.6 Evaluation Environment* details) to find every relevant occurrence of function and method calls. If the candidate depends on PHP and MySQL, the output itemizes calls to functions and methods potentially vulnerable to SQL Injection. The shell script relies on a collection of function and method names from the PHP documentation. Each of the three APIs for MySQL access contributed methods or functions. Ext/mysql is the oldest alternative supported since PHP 2.0. Starting from version 5.0, PHP provides ext/mysqli, whereas versions 5.1 and later also offer PHP Data Objects (PDO) with the extension PDO_MySQL. Table 4.3 lists the functions and methods in these APIs that accept SQL queries as arguments.

mysql_query	mysqli_query	query	prepare
mysqli_prepare	mysqli_real_query	real_query	multi_query
mysqli_multi_query	send_query	mysqli_send_query	exec

Table 4.3: Names of PHP functions and methods that accept strings with SQL statements as arguments

Filters ensure that only calls that have a PHP superglobal ($_GET, $_POST, $_COOKIE, $_REQUEST, $_FILE or $_SERVER) variable among their arguments are reported. In the case of $_SERVER, the script only shows those findings that access one of the indices in Table 4.4. Interpreting the PHP documentation, EstSecure assumes they are the most dangerous input arguments that originate directly from visitor browsers. The shell script highlights all occurrences of the superglobal variable names, so that users can quickly review the results.

HTTP_ACCEPT	HTTP_ACCEPT_CHARSET	HTTP_USER_AGENT
HTTP_REFERER	HTTP_ACCEPT_ENCODING	HTTP_REQUEST_URI

Table 4.4: Primarily relevant indices of the $_SERVER superglobal

The sub-criterion suggests these ratings:

- 0 if a user finds a vulnerability
- 1 if a blacklist approach validates the argument
- 2 if a whitelist approach that the IT-Person does not consider trustworthy validates the argument
- 7 if a stringent whitelist validates the input
- 10 if the script did not list any code locations

If the script does not yield results, the source code does not contain a call to MySQL function/method with a superglobal among its arguments.

Ratings below three disqualify the candidate.

4.3.2.1.2 SQL Injection: Parameter Concatenation *SQL Injection: Parameter concatenation* is a sub-criterion that searches the cause for SQL Injection vulnerabilities: the concatenation of strings as SQL statements. Again, a shell command based on the *estgrep* script searches for all calls to the methods and functions, which Table 4.3 summarizes. This time, the script only informs about those calls that involve one of the following:

- a dollar sign $ within quote signs
- the PHP concatenation operator period . followed by a dollar sign $
- a dollar sign $ followed by a variable name, a period, and quotation marks
- a period . followed by a function name and an opening parenthesis (

The command marks the variable names within the calls. If a variable name is placed between curly braces, such as {$variable}, which is common in PHP for including variables in double quoted strings, the curly braces are also highlighted.

With the exception of one derivation, this sub-criterion advises the same ratings as the last sub-criterion described in paragraph *4.3.2.1.1 SQL Injection: Obvious Vulnerabilities*.

This sub-criterion recommends the rating 8 if a whitelist approach sufficiently validates the relevant variables. It only designates the full rating of 10 points if no SQL statement is concatenated using strings and variables. Prepared statements represent a means for building SQL statements without string concatenation.

A rating lower than 3 disqualifies the candidate.

4.3.2.1.3 SQL Injection: Dangerous String Concatenation Using the shell command for the sub-criterion *SQL Injection: Dangerous String Concatenation*, the implementation finds all calls to the SQL Injection related functions and methods (as Table 4.3 denotes). No filter hides any of the results. The script highlights variable names in the arguments that match calls. In contrast to the previous two sub-criteria, IT-Persons should now check the origins of variables used as arguments. By tracking the variable source, such individuals should find pieces of code where any two strings are concatenated to SQL statements, instead of using the safer prepared statement alternatives.

For the rating, this sub-criterion designates the values:

- zero if an IT-Person finds a vulnerability (e.g., they can trace an argument variable to a PHP superglobal without sufficient validation)
- six if the candidate creates an SQL statement that concatenates strings and user-provided values
- ten if the candidate creates no SQL statement that uses concatenation

4.3.2.1.4 Password Storage *Password Storage* is a criterion that encapsulates the sub-criterion of this type. *Password Storage: Check Storage* is the title of a sub-criterion that checks the password storage of the candidates. It guides IT-Persons to the storage space in the database, or respectively, the file system, where the candidate stores the passwords for user accounts. Based on the appearance of the stored passwords, IT-Persons should rate the candidate:

- 0 if the candidate stores passwords in plain text
- 1 if two accounts with identical passwords receive the same hash value (indicating that the candidate does not apply per-user salt)
- 1 if the candidate hashes passwords with the MD5 algorithm
- 10 if none of the above is true for the storage

A rating below 3 disqualifies the candidate.

4.3.2.1.5 Path Traversal The criterion *Path Traversal* instructs IT-Persons to search for vulnerabilities with the same name. Similar to earlier sub-criteria, this check relies on the PHP documentation for a list of functions that accept a string with a file path as an argument. IT-Persons should execute the estgrep script with an array of function names, and the PHP superglobals used in the previous sub-criteria. To achieve this, the criterion displays the exact command for copying and pasting. Upon completion, the script reports any function calls that derive a file path from a superglobal for an argument. For cases where the command yields no results, the criterion also offers a more general variant of the command. The more generic command does not assume a superglobal in one of the call arguments. In addition, a hint to the shell script named *estfindincludes* allows users to inspect occurring PHP include function variants, while highlighting variable names in their arguments.

In the description, the criterion guides IT-Persons to check every call for variables that can be manipulated.

For the rating, the criterion suggests 0 if a vulnerability is located, and 10 if IT-Persons finds nothing suspicious. Otherwise, such individuals should make their estimate, which will rank the candidates relatively to each other if one is more convincing than the other.

A rating of 0 disqualifies the candidate.

4.3.2.2 Fault Injection

Similarly to an approach that Thomas et al. [TXX11] explained in their work, EstSecure's test preparation controls the vulnerabilities that testers can find by manually hiding them in the candidates.

Unlike Thomas et al., the test for EstSecure did not generate multiple mutants with one distinct vulnerability in each. Instead, it prepared two candidates with the desired vulnerabilities.

4.3.2.2.1 Path Traversal (V1) In the file plugins/captcha/crypt/cryptographp.inc.php of candidate Monstra CMS, the code on line 24 changed from

```
if (is_file($_GET['cfg']) and dirname($_GET['cfg']) == '.' ) {
  $_SESSION['configfile']=$_GET['cfg'];
}
```

to the more vulnerable version without the check for the current directory:

```
if (is_file($_GET['cfg'])) {
  $_SESSION['configfile']=$_GET['cfg'];
}
```

This injected vulnerability allows visitors to specify an arbitrary file from the web server, which is interpreted as PHP code. The remainder of this chapter refers to it using the ID V1.

4.3.2.2.2 Obvious Vulnerability (V2) In WolfCMS, a vulnerability resides in the file wolf/plugins/comment/views/index.php. The original code that starts at line 35

```
$sql = "SELECT COUNT(*) FROM ".TABLE_PREFIX."comment
        WHERE is_approved = 1";
$stmt = $__CMS_CONN__->query($sql);
```

was changed to the vulnerable version:

```
if(isset($_GET['category'])){
  $stmt = $__CMS_CONN__->query("SELECT COUNT(*)
      FROM ".TABLE_PREFIX."comment
      WHERE is_approved = 1
      AND category = ".$_GET['category']);
}
else{
  $stmt = $__CMS_CONN__->query("SELECT COUNT(*)
      FROM ".TABLE_PREFIX."comment
      WHERE is_approved = 1");
}
```

The sub-criterion *SQL Injection: Obvious vulnerabilities* should find this obvious SQL Injection vulnerability with the ID V2.

4.3.2.2.3 Dangerous String Concatenation (V3) Finally, the last vulnerability is hidden in the WolfCMS file wolf/plugins/archive/archive.php on line 124. The original version changed from:

```
$sql = "SELECT DISTINCT(DATE_FORMAT(created_on, '%Y/%m'))
        FROM $tablename WHERE parent_id = :parent_id
        AND status_id != :status ORDER BY created_on DESC";
Record::logQuery($sql);
$stmt = $pdo->prepare($sql);
$stmt->execute(array(':parent_id' => $this->page->id,
                     ':status' => Page::STATUS_HIDDEN ));
```

to the vulnerable version:

```
if(isset($_GET['status'])){
$status = preg_replace("/[';]/",'',$_GET['status']);
}
else{
$status = Page::STATUS_HIDDEN;
}
$stmt = $pdo->prepare(
"SELECT DISTINCT(DATE_FORMAT(created_on, '%Y/%m'))
 FROM $tablename
 WHERE parent_id = :parent_id
 AND status_id != $status
 ORDER BY created_on DESC");
$stmt->execute(array(':parent_id' => $this->page->id));
```

where a user-changeable argument merely sanitized by a weak blacklist approach states a vulnerability. The ID for this vulnerability is V3.

4.3.2.3 Expected Results

The test for sub-criterion *SQL Injection: Obvious Vulnerabilities* expects testers to find the hidden vulnerability V2. They should report a rating of 0 for WolfCMS. For the candidate Monstra, which does not use a database, they should omit this sub-criterion.

Checking the sub-criterion *SQL Injection: Parameter Concatenation*, one should discover vulnerability V3 that also results in a rating of 0 for WolfCMS. Again, testers should omit the sub-criterion for Monstra because it does not depend on a database.

SQL Injection: Dangerous String Concatenation should find vulnerabilities V2 or V3. In addition, in WolfCMS, the file wolf/install/sql_data.php contains an SQL query on line 137 that concatenates variables and a string, and consequently also falls into

the domain of the sub-criterion. Because there are vulnerabilities to be found, testers should rate WolfCMS with 0. Although the concatenation in the sql_data.php is no vulnerability, the other vulnerabilities are real threats. Again, testers should omit this sub-criterion for Monstra.

The sub-criterion *Password Storage: Check Storage* should reveal Monstra using salt-less MD5 hashes for saving passwords. This should earn the candidate a rating of 1. WolfCMS hashes its passwords appropriately using per-user salt, and is to receive a rating of 10.

For the sub-criterion *Path Traversal*, testers should not find a vulnerability for WolfCMS, which implies a rating of ten. Because the vulnerability V1 is hidden in Monstra, the candidate should obtain a rating of 0.

Table 4.5 summarizes the expected ratings for the criteria and candidates.

	WolfCMS	Monstra
C1 SQL Injection: Obvious Vulnerabilities	0	skip (10)
C2 SQL Injection: Parameter Concatenation	0	skip (10)
C3 SQL Injection: Dangerous String Concatenation	0	skip (10)
C4 Password Storage	10	1
C5 Path Traversal	10	0

Table 4.5: Expected criteria ratings

4.3.2.4 Tester Results

To measure the deviation between expectations and reality, four students tested the implementation this work presents.

4.3.2.4.1 Test Person L1 Table 4.6 shows the results of the first tester who executed the long test. This section refers to this tester as L1.

	WolfCMS	Monstra
C1 SQL Injection: Obvious Vulnerabilities	0	skipped
C2 SQL Injection: Parameter Concatenation	0	skipped
C3 SQL Injection: Dangerous String Concatenation	0	skipped
C4 Password Storage	10	1
C5 Path Traversal	5	5

Table 4.6: Criteria ratings assigned by tester L1

Although L1's rating for sub-criterion C2 is 0, as expected, the tester's comment indicates a wrong line of code that is assumed to be an accident. L1 did not find the vulnerability V1 in Monstra. Consequently, L1 opted to rate the candidates based on a subjective estimate. For C5 both candidates received a rating of 5.

Unfortunately, the tester did not comment on the ratings, with the exception of the SQL Injection-related criteria. In addition, L1 deleted the VirtualBox image shortly after finishing the test, and was not able to recall the reason for choosing the given ratings. L1's other comments remain unknown.

In further test runs, the testers were explicitly briefed on how important it is for them to comment on their findings. In addition, extending the text improves the criterion descriptions.

4.3.2.4.2 Test Person L2 The second tester on the long test, L2, submitted the ratings in Table 4.7.

	WolfCMS	Monstra
C1 SQL Injection: Obvious Vulnerabilities	0	skipped
C2 SQL Injection: Parameter Concatenation	0	skipped
C3 SQL Injection: Dangerous String Concatenation	0	skipped
C4 Password Storage	10	1
C5 Path Traversal	9	6

Table 4.7: Criteria ratings assigned by tester L2

L2 found V1 with C2, which was not intended for the test, but is a valid vulnerability that matches the description of the criterion. Furthermore, L2 reported WolfCMS for C5 with a possible path traversal vulnerability in wolf/install/do-install.php on line 152 with the code

```
require_once 'schema_'.$_POST['config']['db_driver'].'.php';
```

Nevertheless, L2 appraised the candidate with a high rating because the installation script would not be re-run in a normal context. For Monstra, L2 discovered the correct vulnerability with C5, but estimated the candidate worth a rating of 6. Whereas V3 is a deliberately hidden vulnerability, L2 decided to rate the candidate depending on his personal estimate.

4.3.2.4.3 Test Person L3 The third tester who conducted the long test variant received the ID L3. Table 4.8 summarizes the findings.

	WolfCMS	Monstra
C1 SQL Injection: Obvious Vulnerabilities	0	10
C2 SQL Injection: Parameter Concatenation	0	10
C3 SQL Injection: Dangerous String Concatenation	0	10
C4 Password Storage	10	1
C5 Path Traversal	0	0

Table 4.8: Criteria ratings assigned by tester L3

L3 detected each vulnerability expected by the test. In addition, L3 opted to not trust the system's suggestions that indicated to omit SQL Injection- related criteria on database-less Monstra, and ran the commands; L3 rated Monstra with 10 manually for the criteria in question. For WolfCMS on C5, L3 noted the rating 0 with the comment, "Path traversing is not even required. It's possible to program PHP directly in the page object, and therefore execute arbitrary code (if you have the right to create blog posts [probably it's enough to be an Editor])." The comment refers to the way WolfCMS manages posting to an internal blog that allows PHP code to be included in the post text.

4.3.2.4.4 Test Person L4 The last tester with the ID L4 designated ratings as stated in Table 4.9. Similar to L3, L4 estimated Monstra with 10 on the SQL Injection criteria, which was redundant, but correct. L4's rating for WolfCMS on C3 is incorrect because L4 did not realize the vulnerability that the shell script output had shown.

C4 on Monstra would deserve a rating of 1 because Monstra merely uses MD5 to hash stored passwords. Instead of checking the password storage directly, L4 reviewed the source code and accepted the code

```
if (trim($user['password']) ==
   Security::encryptPassword(Request::post('password'))) {
```

as sufficient proof for a rating of 10, the rating 2 that L4 submitted instead of 10 for C5 on WolfCMS relies on a false positive finding; the file L4 reported as the reason for the rating 2 does not contain a vulnerability. It consists only of regular code for file uploads, which is no vulnerability in general. For C5 on Monstra, the tester opted for the rating 10 that indicates L4 did not find the path traversal vulnerability V1 hidden in that candidate. Finding the vulnerability required L4 to run all commands from the criterion description, which L4 indicated to have done. Among others, the command *estfindincludes* displayed a dynamic include statement:

```
(...)/monstra/plugins/captcha/crypt/cryptographp.inc.php:26:
include($_SESSION['configfile'])
```

The result leads to a location in the source code that is closely above the line of code with vulnerability V1.

4.3.2.4.5 Summary Table 4.10 itemizes the test's outcomes and determines the number of user rating assignments that are identical to the expectations.

Whereas L1 provided the correct rating on one of the SQL Injection sub-criteria, L3 and L4 decided to value the SQL Injection sub-criteria manually. L4 selected an incorrect rating for C3. With the exception of the previous, the testers appraised the rest of the SQL Injection related (criterion,candidate)-tuples according to the expectations.

	WolfCMS	Monstra
C1 SQL Injection: Obvious Vulnerabilities	0	10
C2 SQL Injection: Parameter Concatenation	0	10
C3 SQL Injection: Dangerous String Concatenation	6	10
C4 Password Storage	10	10
C5 Path Traversal	2	10

Table 4.9: Criteria ratings assigned by tester L4

	WolfCMS	Monstra
C1 SQL Injection: Obvious Vulnerabilities	4/4	2/4
C2 SQL Injection: Parameter Concatenation	4/4	2/4
C3 SQL Injection: Dangerous String Concatenation	3/4	2/4
C4 Password Storage	3/4	3/4
C5 Path Traversal	0/4	1/4

Table 4.10: Strictly counted number of testers reporting exactly the ratings as expected

All testers evaluated sub-criterion C4 Password storage correctly, with the exception of L4, who derived from the recommended path for the check.

Criterion C5 *Path Traversal* caused controversies. L1 did not enter comments, and did not find vulnerability V1 in Monstra. L2 located the vulnerability and rated spontaneously for both candidates. L3 gave WolfCMS the rating 0 because, in his opinion, bad design allows editors to execute arbitrary PHP code in blog posts. L4 did not find the hidden vulnerability V3.

Table 4.11 compiles a more liberal view on the data generated by the testers. It respects the freedom of interpretation and counts ratings that are arguably correct.

	WolfCMS	Monstra
C1 SQL Injection: Obvious Vulnerabilities	4/4	4/4
C2 SQL Injection: Parameter Concatenation	4/4	4/4
C3 SQL Injection: Dangerous String Concatenation	3/4	4/4
C4 Password Storage	4/4	3/4
C5 Path Traversal	2/4	2/4

Table 4.11: Number of users finding an arguably correct solution

4.3.2.5 Discussion of Results

Analyzing the results indicates that testers easily followed the given instructions and achieved the desired goals. All testers reported vulnerabilities immediately visible in the tool output listing.

Future test scenarios will not involve criterion scripts that print ambiguous source locations. The current sub-criterion C1 yielded code locations that are a subset of criterion C2. This led L2 to file the same vulnerability for C2 and C1, which is not incorrect, but was not intended for the test. L2 did not detect the weak blacklist input validation flaw hidden for sub-criterion C2.

For the criteria C4 and C5, L2 and L3 discovered a matching vulnerability.

The test implies that EstSecure and its implementation is indeed useful, and that further improvements and extensions to the system will render it a valuable tool for real-world applications. However, the findings also show that testers produce different outcomes using the same descriptions and suggested work path. This leads to the conjecture that user precision is as essential for the results and utility of EstSecure, as it is for any source code review.

The most valuable resources that IT-Persons have available are their knowledge, experience, and precision in the work process. Given that every testing student has the same education on IT security, their accuracy seems to dictate the differences measured in the success rate.

4.3.3 Tester Feedback

Without exception, all testers indicated that they understood the relevance of the presented EstSecure method and implementation.

4.3.3.1 Encountered Problems

L1 and L2 faced difficulties distinguishing a grouping meta-criterion from a task-bearing sub-criterion. L1 reported to have required a few minutes until realizing that the criteria could be expanded by clicking a plus symbol to the left of the criterion names, as Figure 4.8 shows. The additional hint to this icon at the bottom of the scoreboard page was not there before testing and is intended to alleviate this issue. Later, L2 had a comparable issue. He had JavaScript disabled by default and required to know that JavaScript is essential for the page, which the new note at the bottom of the page provides.

S1 could not understand the shortcut path offered by the description. As in the case of L2, S1's browser blocks JavaScript by default, thus withholding information and leading to adding a clue at the relevant place in the content.

S2 believed the time budget field should receive an estimate of the time the evaluation *would* require, and consequently postponed entering the value. Because of this, S2 did not see the error message that navigates to the shortcut method.

Later, S3 was confused by the error message and did not realize what to do from the shortcut description. Using the error message as the only way to access the short description was discontinued after his test. Instead, additional hints at the project and user properties pages were added.

Scoreboard

Current Project: Default Project (change here)

		WolfCMS	Monstra	Current BCR
+	SQL Injection	0	24443.91	809.18
-	Password Storage	3567.63	713.43	356.76
	Password Storage: Check storage	3567.13	713.43	475.62
	Path Traversal	5945.64	0	224.58
	Resulting estimate score	**9513.27**	**25157.33**	

This page uses JavaScript. Criteria which have sub criteria can be expanded clicking the + sign in the table.

Figure 4.8: The scoreboard with a closed and an expanded criterion

4.3.3.2 Feature Requests

An addition every tester of the long test wanted was integration of the shell scripts into an Eclipse plugin. L3 even proposed including the score board functionality into such an extension. L2 requested the use of a thousands separator for better readability of the scores and score sums on the scoreboard. L3 proposed the idea for a wizard navigating users through the evaluation process.

L2 and L3 suggested serving an administration interface so that interested contributors could add new criterion definitions. They did not know that the present implementation already contains this feature, but that it was invisible to them because it was out of the test scope.

As L3 advised, a hosted version of the scripts that analyse candidate source code could run on the web server (for example), and make the work environment obsolete for this purpose. This would facilitate using EstSecure if there were no resources for a virtual machine.

4.3.3.3 Suggestions on the Content

L2 would like to see a hint for users that indicates the bidirectional clipboard feature of VirtualBox, thus helping with copying & pasting shell scripts between the image and the host Operating System (OS). The previous assumption that VirtualBox allowed that feature by default proved to be incorrect.

L2 and L3 requested the long shell commands in the criterion descriptions to be replaced with short names of scripts already prepared in the virtual machine. These shell scripts should then already provide the commands along with their arguments, and users would only have to copy & paste script names. The original intention of the more complex commands presented to the users was comprehensibility.

The tested tools impressed L4 to the extent of inquiring where to acquire such tools for personal use.

CHAPTER 5

Critical Reflection

A few limitations apply to the findings of the current work. Section *5.1 Limitations* elaborates.

5.1 Limitations

5.1.1 Method

5.1.1.1 Empty Formula Factors

As described in Section *4.1.12 Benefit Formula and User Context Data*, the benefit calculation formula (Section *4.1.10 Benefit Formula*) can adapt to user context. The EstSecure method facilitates this by defining those conditions that overwrite the benefit formula risk factors if satisfied.

Section *4.1.11 Criteria and Benefits* characterizes the procedure for reversing the surjective function used in the OWASP Top 10 project. The function maps technical impact and likelihood factors from a scale of 1 to 9 (most severe) to a factor value on a scale from 1.0 (most severe) to 3.0. For this step, EstSecure requires an estimate for the ambiguous value when reverting the simplification to the larger scale. The ambiguity is described (for example) in the question "Does the Top 10's *exploitability average* correspond to an exploitability factor of 4, 5, or 6?". The technical impact component requires up to four estimated values for its four factors because it is equal to the average of four question values.

If a formula modifier based on a condition should increase the risk, but decreases it accidentally, a problem occurs.

For example, an administrator wants do create a criterion for SQL Injection. The OWASP Top 10 project uses the values *easy*, *common*, and *average* to appraise SQL Injection's likelihood factors. From this, the administrator maps the values to the more finely grained version of the formula using the values 9, 6, and 6 (for example). This

assignment leads to an average likelihood factor of 7. After that, the project adds a formula modifier that should increase the *size of attacker group* factor to 6.5 if the deployed candidate is accessible via the Internet. This should increase the likelihood value, and consequently, the risk. However, *size of attacker group* was previously empty and ignored. Now, the value that should increase the risk decreases the average likelihood factor, and consequently, reduces the risk rating value of the criterion.

A future version of this work can solve this issue by introducing relative formula modifiers that add to or subtract from a single factor, or the factor average, if a condition is satisfied. The current state of the work knowingly accepts this limitation because the administrator can easily circumvent this issue while adding criteria.

5.1.1.2 VPM Metric

For IT-Persons with diminutive time budgets, the present work prepares short estimation methods, as Section *4.1.15 Short Estimation* states. EstSecure instructs to calculate an extremely basic metric based on reported vulnerabilities per percent of the product's market share. We call this metric VPM. Whereas the reader can easily obtain the vulnerability count of a candidate online, they will face difficulties researching the market share for web application types different from CMS. Presumably, this is the case because an automated web crawler can barely distinguish reliably between a webmail solution and a project management tool (for example). At the same time, the crawler may classify any web application as CMS. Unfortunately, this renders VPM only useful for comparing CMS-type candidates. In addition, inspecting the metric for a significant correlation with the number of vulnerabilities in a candidate requires means not available at the time of writing. Access to a commercial SCA and comparing the SAVD for a range of candidates could confirm or refute the statistical significance of the metric.

5.1.2 Implementation

5.1.2.1 Matching Vulnerability Patterns

Shell commands that are part of the evaluation environment do not list every possible occurrence of each vulnerability. For example, Section *4.3.2.1 Prepared Criteria* shapes the shell command for the criterion *SQL Injection: Parameter concatenation*, which would find the function call

```
mysql_query("SELECT * FROM a WHERE id =".getSomeID());
```

but would not report

```
mysqli_prepare(getSomeQuery()." WHERE value > ?");
```

A regular expression can easily find an opening parenthesis after a function or method name. Consequently, finding a function call after a concatenation operator is reasonable. However, to match more complex versions of the second code sample, the

script would require an advanced regular expression capable of fully recognizing every series of characters that PHP allows inside parentheses. Consequently, this would include those comments that use the Universal Character Set Transforming Format (UTF)-8 character set with thousands of distinct allowed characters.

Alternatively, the script could use sgrep to recognize arbitrarily nested function calls, which is difficult to achieve even with such an advanced tool. The present solution does not consider these peculiarities for the sake of simplicity, and ignores these marginal problems.

5.1.2.2 Static Passing Score

Currently, a static number captures the criteria passing scores. Because scores can dynamically change based on modifiers or the benefit formula, in some cases, the passing score may prove difficult to define. This is strictly an implementation-related limitation that can be alleviated easily in future versions. For example, a dynamic passing score that involves user context properties could greatly alleviate this issue.

5.1.3 Limited Content

Whereas EstSecure is sufficiently universal to apply to manifold candidates, the content pages and criteria provided by this work currently are not. This work could show EstSecure's usefulness by employing a limited set of vulnerabilities that PHP-based CMS solutions could contain. Because both EstSecure and its realization strive for extensibility, many possibilities arise for managing the issue of limited content. Administrators can add novel criteria and properties for new programming languages, application types and technologies, and naturally, the vulnerabilities introduced earlier in this study.

CHAPTER 6

Summary and Future Work

6.1 Future Work

The established method that allows IT-professionals conduct security audits leaves numerous possibilities for extension.

Obviously, defining and testing additional content will increase the capabilities of EstSecure and its implementation extraordinarily. Creating new properties can help capture more relevant aspects of user context. Based on new properties, one can add conditions that modify benefit and effort values, modify benefit formulas, allow dynamic description blocks in criteria descriptions, and allow or hide criteria. New criteria can help users search for a broad range of vulnerabilities in candidates developed in a vast number of programming languages using different frameworks and toolkits. In addition, new types of criteria can instruct users (for example) to employ penetration testing tools for the dynamic evaluation of candidates. This measure supports more ways for detecting vulnerabilities automatically. Certainly, this is especially of advantage in cases of relatively unknown candidates that are yet to receive the attention of security researchers. Additional articles will help users with tooltip-based hints, linked articles, or even references and external links to fill knowledge gaps.

Testing and investigating the statistical significance of the VPM metric will allow the website to confidently highlight the process as a shortcut solution for an estimate of a candidate's security.

Section *3.2 Developing a Method* described a meeting with Christian. One of his suggestions led to an idea of adding a message board to the website that implements EstSecure. Such message board would allow users to exchange experiences and ask questions. Alternatively, a guiding article could help users ask questions on well-known public boards, such as security.stackexchange.com, while preventing them from polluting the mentioned website with questions not relevant to others.

Improvement possibilities on the evaluation environment are also manifold. All testers requested an Eclipse plugin to integrate this work's findings to the software for browsing

candidate source code. This would ease the work of IT-Persons. Some testers also suggested that providing complex commands to search for sensitive sinks in the candidate code in named shell scripts would make copying and pasting long commands obsolete.

The OWASP Orizon Project [DLP14] might be a tool interesting for integration in the presented evaluation environment. The framework for security evaluating source code is not yet complete, but it might help to find sensitive sinks with more finely adjustable call circumstances than sgrep.

A heuristic evaluation applied to the method's implementation will disclose usability issues that are yet unknown. Improved usability will help IT-Persons be more efficient with the tool. This is the main goal of this entire work: time efficiency for the users.

Translating the implementation and content to other languages will allow a broader group of users to access and use the tool. The majority of user-visible texts from the current implementation reside in easily translatable language files.

Extending the EstSecure method to account for dependencies between criteria, set up tasks and the time consumption of those will allow more precise approximations. For example, a candidate needs to be installed and configured on a web server before dynamic evaluation using (for example) penetration testing tools can examine it. This installation requires time, but once a candidate is set up, all future criteria with the same requirement do not need setup time again. Already existing mechanisms, such as user context properties and condition-based effort modifiers can adapt the criterion effort values accordingly. However, the exact way for managing dependencies (i.e., *when* to set up the candidate on a web server) requires a sound concept.

Using a SAT solver in the implementation could present an optimized order of contextual property questions. A formula could calculate how many (sub)criteria to disable based on the provision of an individual property. In general, a related approach could determine how relevant is a property for ordering the criteria on the scoreboard. This would save users time in addition to all other measures aiding this goal.

Including additional content will allow EstSecure to become part of IT security curricula. Its very nature and implementation predestines it for application to (for example) deliberately vulnerable, educational web applications, such as OWASP WebGoat.

6.2 Conclusion

Security evaluation is a broad field of expertise that involves an array of useful tools and methods. Discovering vulnerabilities well hidden deep in applications is a form of art. The present work proposed the EstSecure method that allows IT professionals to estimate, evaluate, and compare security properties of open source web applications. The proposed method does not require expensive specialized software or in-depth understanding of the heterogeneous field of security evaluation. Structuring a complex task into numerous manageable ones helped testers find almost all of the deliberately hidden vulnerabilities in real-life web applications. The offered background information educated the testers, whereas the efficient application of available context reduced the suggested tasks.

The verification tests revealed different levels of success from individuals who followed structured instructions. The feedback provided by testers will help improve EstSecure, and consequently, its implementation.

Whereas the validation phase did not yield exclusively positive results with all evaluating testers, it confirmed predominantly the positive assumptions about EstSecure, and indicated potential for even better finding rates to be achieved. The user's rigor and precision limit its efficiency. In addition, users need to be willing and able to execute the instructions compiled .

We are convinced that the EstSecure method can be of use for a broad range of IT individuals willing to learn about security and compare the security characteristics of open source web applications. In times where cloud storage systems of big companies are hacked, self-hosted applications independently secured could become increasingly prevalent. We would be proud to contribute to this future with less single points of failure for our society's data storage needs.

Acronyms

API Application Programming Interface

ASDR Application Security Desk Reference

AST Abstract Syntax Tree

BCR Benefit-Cost Ratio

CA Certificate Authority

CLASP Comprehensive, Lightweight Application Security Process

CMS Content Management System

CSS Cascading Style Sheets

CVE Common Vulnerabilities and Exposures

CVSS Common Vulnerability Scoring System

CWE Common Weakness Enumeration

GUI Graphical User Interface

HTML Hyper Text Markup Language

ISO International Organization for Standardization

IEC International Electrotechnical Commission

IT Information Technology

JSON JavaScript Object Notation

KDE K Desktop Environment

MD5 Message-Digest Algorithm 5

NVD National Vulnerability Database

LDAP Lightweight Directory Access Protocol

PDF Portable Document Format

PDO PHP Data Objects

PDT PHP Development Tools

PHP PHP: Hypertext Preprocessor

ORM Object-Relational Mapping

OS Operating System

OWASP Open Web Application Security Project

RATS Rough Auditing Tool for Security

SAMM Software Assurance Maturity Model

SAVD Static Analysis Vulnerability Density

SCA Source Code Analyzer

SDLC Software Development Life Cycle

SME Small and Medium Enterprise

SLOC Source Lines Of Code

SQL Structured Query Language

SRI Security Resource Indicator

SSL Secure Sockets Layer

TDD Test Driven Development

URL Unique Resource Locator

UTF Universal Character Set Transforming Format

VPM Vulnerabilities per Percent of Market share

WUI Web Usability Index

XML Extensible Markup Language

XP Extreme Programming

XSS Cross Site Scripting

Bibliography

[AM13] Luca Allodi and Fabio Massacci. My software has a vulnerability, should i worry? *arXiv preprint arXiv:1301.1275*, 2013.

[AU08] Ravikant Agarwal and David Umphress. Extreme programming for a single person team. In *Proceedings of the 46th Annual Southeast Regional Conference on XX*, pages 82–87. ACM, 2008.

[BA04] Kent Beck and Cynthia Andres. *Extreme Programming Explained: Embrace Change, 2nd Edition (The XP Series)*. Addison-Wesley, 2004.

[Bec01] Kent Beck. Aim, fire. *IEEE Software*, 18(5):87–89, 2001.

[Bev05] Nigel Bevan. Cost benefits evidence and case studies. *Cost-justifying usability: An update for the internet age. San Francisco: Morgan Kaufmann*, 2005.

[BW10] Wolf-Gideon Bleek and Henning Wolf. *Agile Softwareentwicklung*. Dpunkt.Verlag GmbH, 2010.

[C+08] Pravir Chandra et al. Software assurance maturity model. *A guide to building security into software development v1. 0, OWASP Project*, 2008.

[CA06] Mark Curphey and Rudolph Arawo. Web application security assessment tools. *Security & Privacy, IEEE*, 4(4):32–41, 2006.

[CWFW06] Pravir Chandra, Toni Wohleber, Jeremy Feragamo, and Jeff Williams. Owasp clasp, 2006. `https://www.owasp.org/index.php/Category:OWASP_CLASP_Project` (accessed 2015-02-10).

[Dah11] Johannes Dahse. Rips scanner, 2011. `http://rips-scanner.sourceforge.net/` (accessed 2015-02-14).

[DLP14] Greg Disney-Leugers and Paolo Perego. Owasp orizon project, 2014. `https://www.owasp.org/index.php/Category:OWASP_Orizon_Project` (accessed 2015-02-19).

[EF00] Eric Evans and Martin Fowler. Specifications, 2000. `http://martinfowler.com/apsupp/spec.pdf` (accessed 2015-02-17).

[How06] Michael Howard. A process for performing security code reviews. *IEEE Security & Privacy*, 4(4):0074–79, 2006.

[HSS02] Ilse Harms, Werner Schweibenz, and Johannes Strobel. Usability evaluation von web-angeboten mit dem web usability index. In *Proceedings der*, volume 24, pages 283–292, 2002.

[JS05] David S Janzen and Hossein Saiedian. Test-driven development: Concepts, taxonomy, and future direction. *Computer Science and Software Engineering*, page 33, 2005.

[Kap05] Karin Kappel. Usability testing. 2005. Research Group for Industrial Software (INSO), Vienna University of Technology, Austria.

[Mao09] Chengying Mao. Experiences in security testing for web-based applications. In *Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human*, pages 326–330. ACM, 2009.

[Mar11] Robert C. Martin. *Clean Coder: Verhaltensregeln fur professionelle Programmierer*. Addison Wesley Verlag, 2011.

[Mes07] Gerard Meszaros. *xUnit Test Patterns: Refactoring Test Code*. Addison-Wesley, 2007.

[MFMP10] Daniel Mellado, Eduardo Fernández-Medina, and Mario Piattini. A comparison of software design security metrics. In *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, pages 236–242. ACM, 2010.

[Mil08] Leonardo Cavallari Militelli. *OWASP ASDR: The Application Security Desk Reference*. OWASP, 2008.

[MIT99] MITRE. CVE database, 1999. `http://cve.mitre.org` (accessed 2015-02-09).

[MKC08] Mateo Meucci, E Keary, and D Cuthbert. Owasp testing guide v3, 2008. `https://www.owasp.org/index.php/OWASP_Testing_Guide_v3_Table_of_Contents` (accessed 2015-02-09).

[MSW09] Michael Meike, Johannes Sametinger, and Andreas Wiesauer. Security in open source web content management systems. *IEEE Security & Privacy*, (4):44–51, 2009.

[Nie01] Jakob Nielsen. How to conduct a heuristic evaluation, 2001. `http://www.useit.com/papers/heuristic/heuristic_evaluation.html` (Accessed 2015-02-09).

[Ö10] Serkanz Özkan. cvedetails.com, 2010. http://www.cvedetails.com (accessed 2015-02-09).

[OWAa] OWASP. Wiki: Attacks. https://www.owasp.org/index.php/Category:Attack (accessed 2015-02-09).

[OWAb] OWASP. Wiki: Vulnerabilities. https://www.owasp.org/index.php/Category:Vulnerability (accessed 2015-02-09).

[OWA14a] OWASP. Owasp risk rating methodology, 2014. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology (accessed 2015-02-09).

[OWA14b] OWASP. Owasp testing guide v4, 2014. https://www.owasp.org/index.php/OWASP_Testing_Guide_v4_Table_of_Contents (accessed 2015-02-09).

[Pel08] Carlo Pelliccioni. Owasp backend security project, 2008. https://www.owasp.org/index.php/Category:OWASP_Backend_Security_Project (accessed 2015-02-10).

[QS15] Q-Success. Usage of server-side programming languages for websites, 2015. http://w3techs.com/technologies/overview/programming_language/all (accessed 2015-02-14).

[Ret94] M Rettig. Prototyping for tiny fingers; commun. acm. *Communications Of The Acm, 1994 Apr, Vol.37(4), pp.21-27*, 1994.

[SCY10] Hui-zhong Shi, Bo Chen, and Ling Yu. Analysis of web security comprehensive evaluation tools. In *Networks Security Wireless Communications and Trusted Computing (NSWCTC), 2010 Second International Conference on*, volume 1, pages 285–289. IEEE, 2010.

[TXX11] Lijo Thomas, Weifeng Xu, and Dianxiang Xu. Mutation analysis of magento for evaluating threat model-based security testing. In *Computer Software and Applications Conference Workshops (COMPSACW), 2011 IEEE 35th Annual*, pages 184–189. IEEE, 2011.

[VdSCC+08] A Van der Stock, D Cruz, J Chapman, D Lowery, E Keary, M Morana, D Rook, J Williams, and P Prego. Owasp code review guide, v1. 1. *The OWASP Foundation, Nov*, 2008.

[vdSWCS05] Andrew van der Stock, Adrian Wiesmann, Mark Curphey, and Ray Stirbei. Guide to building secure web applications and web services 2.0. *Black Hat edition*, 2005.

[VM09] Ganesh Vaidyanathan and Steven Mautone. Security in dynamic web content management systems applications. *Communications of the ACM*, 52(12):121–125, 2009.

[VR11] Jaya Vijayan and G Raju. A new approach to requirements elicitation using paper prototype. *International Journal of Advanced Science and Technology*, 28:9–16, 2011.

[w3t15] w3techs.com. w3techs.com cms statistics, 2015. `http://w3techs.com/technologies/overview/content_management/all` (accessed 2015-02-15).

[WD12] James Walden and Maureen Doyle. Savi: Static-analysis vulnerability indicator. *Security & Privacy, IEEE*, 10(3):32–39, 2012.

[WDWW09] James Walden, Maureen Doyle, Grant A Welch, and Michael Whelan. Security of open source web applications. In *Proceedings of the 2009 3rd international Symposium on Empirical Software Engineering and Measurement*, pages 545–553. IEEE Computer Society, 2009.

[WKO14] WKO, 2014. `http://wko.at/Statistik/KMU/WKO-BeschStatK.pdf` (accessed 2015-04-16).

[WRL05] Henning Wolf, Stefan Roock, and Martin Lippert. *eXtreme Programming*. dpunkt Verlag, 2005.

[WW13] Jeff Williams and Dave Wichers. Owasp top 10 2013. 2013. `https://www.owasp.org/index.php/Top_10_2013-Top_10` (accessed 2015-04-18).