Why is security still an issue?
– A study comparing developers’ software security awareness to existing vulnerabilities in software applications

Varför är säkerhetshål i mjukvara fortfarande ett problem?

Lars Backman

Supervisor : Azeem Ahmad
Examiner : Ola Leifler
Upphovsrätt


Copyright

The publishers will keep this document online on the Internet – or its possible replacement – for a period of 25 years starting from the date of publication barring exceptional circumstances. The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/her own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility. According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement. For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: http://www.ep.liu.se/.

© Lars Backman
Abstract

The need for secure web applications grows ever stronger the more sensitive, personal data makes its' way onto the Internet. During the last decade, hackers have stolen enormous amounts of data from high profile companies and social institutions. In this paper, we answer the question of why security breaches still occur; Why do programmers write vulnerable code? To answer this question, we conducted a case study on a smaller software development company. By performing penetration tests, surveys and interviews we successfully identified several weaknesses in their product and their way of working, that could lead to security breaches in their application.

We also conducted a security awareness assessment and found multiple contributing factors to why these weaknesses occur. Insufficient knowledge, misplaced trust, and inadequate testing policies are some of the reasons why these vulnerabilities appeared in the studied application.
Acknowledgments

First I would like to thank my thesis advisor Azeem Ahmad at the Department of Computer and Information Science at Linköping University. He has helped me wholeheartedly and always met me with a smile on his lips. His support has been crucial for the success of this work.

I would also like to thank the classmates who helped me by testing my interview questions before I could send them to my subjects. Their comments and insights into the structure of my survey question helped a lot.

I must not forget Dennis Persson, my roommate. When I was stuck with the SSL error thrown by the ZAP Web Proxy, it was his idea to use an automated mouse and keyboard recorder that could automate the testing procedure and saved me hours of work. Thank You!

At last I must express my very profound gratitude to my mom, for always believing in me and picking me up when I was feeling unmotivated. Your encouragement throughout these years has been one of the main reasons why I have succeeded.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>iii</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>iv</td>
</tr>
<tr>
<td>Contents</td>
<td>v</td>
</tr>
<tr>
<td>List of Figures</td>
<td>vii</td>
</tr>
<tr>
<td>List of Tables</td>
<td>viii</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Background</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Aim and Motivation</td>
<td>2</td>
</tr>
<tr>
<td>1.3 Research Questions</td>
<td>2</td>
</tr>
<tr>
<td>1.4 Delimitations</td>
<td>3</td>
</tr>
<tr>
<td>2 Theory</td>
<td>4</td>
</tr>
<tr>
<td>2.1 Cybersecurity</td>
<td>4</td>
</tr>
<tr>
<td>2.2 Why do programmers write vulnerable code?</td>
<td>5</td>
</tr>
<tr>
<td>2.3 Risks and Prevention Techniques</td>
<td>6</td>
</tr>
<tr>
<td>2.4 Discovering Potential Vulnerabilities</td>
<td>8</td>
</tr>
<tr>
<td>3 Method</td>
<td>11</td>
</tr>
<tr>
<td>3.1 Penetration testing</td>
<td>11</td>
</tr>
<tr>
<td>3.2 Interviews and Survey</td>
<td>13</td>
</tr>
<tr>
<td>4 Survey and Interview Results</td>
<td>19</td>
</tr>
<tr>
<td>4.1 Survey</td>
<td>19</td>
</tr>
<tr>
<td>4.2 Interviews</td>
<td>26</td>
</tr>
<tr>
<td>5 Penetration Test Results</td>
<td>28</td>
</tr>
<tr>
<td>6 Discussion</td>
<td>30</td>
</tr>
<tr>
<td>6.1 Psychological Factors</td>
<td>30</td>
</tr>
<tr>
<td>6.2 Technical Factors</td>
<td>31</td>
</tr>
<tr>
<td>6.3 Real-World Factors</td>
<td>31</td>
</tr>
<tr>
<td>6.4 Responsibility Factors</td>
<td>31</td>
</tr>
<tr>
<td>6.5 Work In A Wider Context</td>
<td>31</td>
</tr>
<tr>
<td>7 Limitations &amp; Future Work</td>
<td>33</td>
</tr>
<tr>
<td>7.1 Internal Validity</td>
<td>33</td>
</tr>
<tr>
<td>7.2 External validity</td>
<td>34</td>
</tr>
<tr>
<td>8 Conclusion</td>
<td>35</td>
</tr>
</tbody>
</table>
Bibliography

Appendices

1 Appendix 1 ................................................................. 39
2 Appendix 2 ................................................................. 47
List of Figures

1.1 Flowchart over the infrastructure behind the CRM-system ........................................ 2

2.1 Illustration of how an attacker can utilize unsanitized input to construct malicious
queries. The vulnerable variable is highlighted in red, and the malicious input has
been URL encoded ........................................................................................................ 7

3.1 ZAP asking about keeping the session persistent ...................................................... 11

3.2 Launching a preconfigured version of Firefox from ZAP ......................................... 12

3.3 Showing Firefox proxy settings .................................................................................. 12

3.4 Creating a new context from ZAPs Site view ............................................................. 12

3.5 Creating a new context from ZAPs Site view ............................................................. 13

3.6 Example question examining the subjects knowledge regarding injection attacks..... 14

3.7 Example of more elaborate interview questions ......................................................... 15

5.1 Alert box altered to show cookie content ................................................................. 29
List of Tables

3.1 Table displaying examples on how coding and classification was done to analyze quantitative interview data ................................................................. 17

4.1 Survey results from the survey sections of Background, Security Related Education and Self Assessment. With anonymized labels for the developers .......... 20

4.2 Survey results from the survey section “Cybersecurity Knowledge” part 1 .... 21

4.3 Survey results from the survey section “Cybersecurity Knowledge” part 2 .... 22

4.4 Survey results from the survey section “Security Within the Application” part 1 23

4.5 Survey results from the survey section “Security Within the Application” part 2 24

5.1 OWASP ZAP proxy test results ................................................................. 28
The expansion of the Internet has brought with it an explosion of new services hosted online by companies and governmental institutions. The Internet has simplified many bureaucratic processes, but the popularity of the Internet has also attracted troublemakers and criminals trying to utilize the still premature nature of the Internet to their advantage. In the last couple of years, many high-profile companies like Equifax, Yahoo, Deloitte, and Verizon have been subjected to substantial compromises of their systems.

Equifax, a consumer credit reporting agency was the subject of one of the most significant data breaches last year with personal data of more than 145 million American consumers leaked. The leaked data contained credit histories, social security numbers, and residential addresses. This data, now in the hands of criminals, can be used to get credit cards and mortgages in the name of the unsuspecting victims of this attack. Even after Equifax recognized the attack, Krebs, a blogger specialized in cybersecurity, could show how Equifax continued to have significant security issues. In the blog post Krebs shows how he could access the username and passwords of employees by utilizing the "admin/admin" username and password combination.

The example with Equifax shows the rising need for good web application security. This need has brought both political actions like the new European General Data Protection Regulation (GDPR) and industry efforts like the Open Web Application Security Project (OWASP). OWASP's core purpose is to "Be the thriving global community that drives visibility and evolution in the safety and security of the world’s software." and has since its start in 2001 tried to collect and centralize the knowledge of the world’s security experts. One of the outcomes from OWASP is the OWASP Top 10, representing the most critical web application security risks. The list was first designed to raise awareness regarding risks surrounding web applications but has today grown to become a standard used in web application security research for classifying web application vulnerabilities.

However, simple vulnerabilities continue to appear even with these commitments. Why? Blyth argues that this is because of multiple factors divided into three main groups of technological, psychological or real-world factors.

We want to expand on this research and examine the software security awareness of web developers in a small organization that has had problems with vulnerabilities in their products, as well as conduct experiments on their application to identify existing vulnerabilities.
1.1 Background

We will not name the company subjected to the research in this thesis due to some of the sensitive information we will present. It is, however, a small software developing company that produces applications for business to business sales representatives in the Fast Moving Customer Goods (FMCG) industry. The company has recently started to develop a new Customer Relationship Management system (CRM). Due to pressure from customers and GDPR, the company has felt the need for improving their security-related work. The application itself is intended to be used via an iPad application, but the application is web-based for better cross-platform compatibility. The main functionalities of the CRM-system include order management, the ability to calculate prices and sales margin, visits scheduling, contact information, and budget tracking.

The application in question is built using a JavaScript frontend based on third-party dependencies like FullCalendar [12], Chart.js [4], jQuery [33], and moment.js [19]. The front-end is served using a PHP backend written with the support of the Yii Framework [38]. The client accesses the application via an infrastructure built using the Amazon Web Services (AWS) [1]. The infrastructure consists of a pool of web servers that manages requests served by load balances and a Content Distribution Network (CDN). The web servers are configured to be stateless, meaning that all dynamic content is served by the database servers connected to the web servers. There are always two database servers online, one master and one online replica that ensures redundancy if the primary web server fails. A flowchart of the infrastructure is shown in Figure 1.1.

![Flowchart over the infrastructure behind the CRM-system](image)

**Figure 1.1:** Flowchart over the infrastructure behind the CRM-system

1.2 Aim and Motivation

The company subjected to this study has earlier had problems with vulnerabilities in their products. We aim to utilize the OWASP Top 10 to evaluate if there are any fundamental vulnerabilities present in their new CRM-system. If we discover any vulnerabilities, we want to analyze why these vulnerabilities exist so that we can understand the company’s weaknesses and recommend changes that will improve their security-related work and reduce the possibility of more security-related issues in the future.

1.3 Research Questions

**RQ1** What vulnerabilities, mentioned by the OWASP list of the top 10 most critical web application security risks can be identified in the CRM system using the ZAP web-proxy?

**RQ2** Does any relationship exist between the vulnerabilities found in the CRM-system and the developers’ web application security awareness?
1.4 Delimitations

The security aspects of a web application are only as strong as the weakest link. All aspects must, therefore, be taken into account when conducting a full security audit of an application. Due to the time limitations of this paper, a full security audit will not be possible. This audit will instead focus on the developers’ awareness of software security and the security of the CRM application with regards of the OWASP Top 10, and not take into consideration network security or server configuration, which would be necessary to give a full insight into the security of an application.
To be able to answer \textbf{RQ2} we must first understand the basic concepts of software security and realize what makes software developers write insecure code.

\section{Cybersecurity}

The terms information security and cybersecurity is easily mixed even though they do not always represent the same idea. Solms and Solms \cite{solms2010cybersecurity} worked on constructing a simple definition of cybersecurity and defined it as "...the part of information security which specifically focuses on protecting the confidentiality, integrity, and availability of digital information assets against any threat, which may arise from such assets begin compromised via the Internet". The three topics of confidentiality, integrity, and availability are also known as the CIA-triad and stand in the center of the definition of both information security and cybersecurity. The three topics are defined as follows:

\textbf{Confidentiality}

Confidential information is only to be disclosed to authorized parties and needs protection against unauthorized people, entities or processes \cite{solms2010cybersecurity}. Confidentiality is especially important in systems that handle secretive information like personal information like credit card numbers and social security numbers. Encryption is an essential tool when it comes to maintaining confidentiality. With strong encryption, we can ensure that the confidential data cannot be interpreted by a third party, even if they have access to it. A breach of confidentiality occurs when the sensitive data can be read and interpreted by unauthorized people. As an example the Equifax data breach led to a violation of confidentiality when personal information leaked from their database.

\textbf{Integrity}

Confidentiality is an important cornerstone; however, if critical data is changed or destroyed, it can be just as problematic as if the data leaked. Integrity is defined as a "property of accuracy and completeness" and ensures that vital information cannot be altered when data is stored or transferred between storage locations \cite{solms2010cybersecurity}. One way hashing is a common technique to ensure
the integrity of a message in transit. The one-way hash is calculated in beforehand and sent together with the message. The recipient then computes the hash again and compares it to the one received. If the hashes match, the message has stayed intact.

Availability

Too much effort spent on ensuring the confidentiality or integrity of some data can hinder the intended use of that data. A driver's license, for example, is a personal document that needs to be kept safe. To ensure the safety of the document it should be stored in a secured container, preferably a safe. If the driver's license is stored in such a container it would be harder for criminals to steal the owner's identity, but it would be so much harder to carry around, and each time a police officer would like to see the licenses one would need to open the safe. Many people will, therefore, prefer to use a wallet instead of a safe. It is not as safe, but it is much more practical. This example shows why there is a need for balance among confidentiality, integrity, and availability. Availability is defined as "property of being accessible and usable upon demand by an authorized entity" and ensures that the people who are authorized to read or modify the data can do so without unnecessary obstacles.

2.2 Why do programmers write vulnerable code?

With the CIA-triad in mind, it might seem obvious that all programmers should strive to keep the three cornerstones in balance. After all, a vast majority of programmers want to write good code. So, why are security vulnerabilities so common in applications today? Blyth suggests that there are three main factors at play when good programmers write bad code:

1. Technical factors; The complexity of the underlying system is too great.
2. Psychological factors; Mental hindrances, like problems with risk assessment and faulty mental models.
3. Real-world factors; Financial and social factors that work against software quality.

Xie et al. studied these factors and found that while their interview results highlight the same real-world problems, they also discovered that another critical influencer to secure software development practices was an "it is not my problem" attitude. They noticed that developers exhibit a firm reliance on other people, processes or technologies to take care of software security. The authors explain this attitude with the many concerns and pressures already felt by software developers. These concerns make it natural for the developers to pass on this responsibility to someone, or something, else. While this attitude was found to be encouraged by some organizational policies, it can be dangerous if there exist holes in this security net.

To evaluate the first two factors; Technical and Psychological, the knowledge and awareness of web application security issues can be analyzed to see what tools the developers have to combat these factors. Rahim et al. conducted a systematic review of methods used to assess cybersecurity awareness, and they found that we cannot solely base our judgment of human behavior on quantitative research; and therefore assessment of cybersecurity awareness requires a combination of methodologies. The authors identify two studies Rezgui and Marks, and McCormac et al. which utilize a combination of observations, survey questionnaires, and interviews to evaluate knowledge, attitude, and behavior regarding cybersecurity issues. Both papers highlight the importance of collecting data from multiple sources when performing awareness assessments.

When conducting a quantitative evaluation of cybersecurity awareness, Kruger et al. found that there is a significant relationship between the knowledge of concepts (vocabulary)
2.3 Risks and Prevention Techniques

To be able to answer RQ1 we have to define and understand the risks that affect the CRM-system in question. The application is built using the PHP programming language on the server side, and this will define what kind of vulnerabilities we are looking for. The following two sections will describe the risks most likely to affect a modern PHP application as described by OWASP Top10\textsuperscript{23} and OWASP PHP security cheat sheet\textsuperscript{11}.

PHP design and security issues

PHP is an open source programming language with framework like features already built-in. With these built-in features, it is important to evaluate both the language aspects as well as the framework aspects when trying to secure a PHP application\textsuperscript{11}. The language also suffers from a collection of security pitfalls like unreliable built-ins, configuration issues and the built-in URL-routing\textsuperscript{11}. In this section, we want to highlight PHP’s weak typing architecture since it is one of the language features that has caused severe security issues in high profile applications\textsuperscript{21}.

PHP’s weak typing means that PHP will convert data types automatically when needed. This feature - typically referred to as type juggling - might seem attractive since it unloads some workload from the developer, but it can cause unexpected behaviors since it can change the value of a variable during the type juggling process. In section 3.2 we describe an example of how type juggling can cause such unexpected behavior.

Critical web application security risks

Although language specific features can affect the security of a web application, the most critical risks come from programming practices and design choices made by the application’s developers. We will use The OWASP Top 10\textsuperscript{23} as a guide for the most critical risks that can affect a modern web application. OWASP constructed this list by examining data from companies specialized in cybersecurity, and surveys from experts. The list presents the following risks:

1. **Injection** flaws allow for untrusted data to be sent to an interpreter as a command or query. Malicious data can then trick the interpreter to execute unintended commands or access confidential data. To prevent these kinds of attacks OWASP recommends using either a secured API, which avoids the use of the interpreter entirely or provides a parameterized interface. One specific type of injection attack that is highly relevant in the context of web application security is SQL-injection attacks. Databases play a critical role in serving dynamic content on the modern web, and the Structured Query Language (SQL) is one of the primary tools used when manipulating the database. The web server often builds SQL queries with the help of user provided input, and if the developers do not adequately sanitize this input, the user can manipulate this input to construct new queries like the example in Figure 2.1.

2. **Broken Authentication** can give an attacker access to confidential information, and can also allow the attacker to impersonate another user, either temporarily or perma-
2.3. Risks and Prevention Techniques

Figure 2.1: Illustration of how an attacker can utilize unsanitized input to construct malicious queries. The vulnerable variable is highlighted in red, and the malicious input has been URL encoded.

- OWASP recommends the use of multi-factor authentication, and implementation of weak-password checks together with password length, complexity, and rotation policies.

3. **Sensitive Data Exposure** might happen whenever an application does not adequately protect sensitive data. Adequately protected data needs to be encrypted with updated encryption algorithms when it is both in transit between hosts and when it is in rest. OWASP especially encourages always to have data in transit encrypted with secure protocols, such as TLS. Passwords should be encrypted using robust adaptive and salted hashing functions with a delay factor.

4. **XML External Entities (XXE) vulnerabilities** can affect some older XML processors that still process external entities. These external entities can then be used to disclose internal files, execute remote code and perform internal port scannings. Therefore one should, whenever possible, use less complex data formats and avoid serializing sensitive data. If that is not possible, one should patch or upgrade all XML processors and libraries and disable XML external entities and document type definition processing.

5. **Broken Access Control** can be exploited to access unauthorized functionalities and data. Poorly implemented restrictions on what authenticated users are allowed to access will result in access control vulnerabilities. When designing access control mechanisms, OWASP suggests always to deny access by default (except for public resources), log access control failures, and notify admins when necessary.

6. **Security Misconfiguration** is, according to OWASP, one of the most commonly seen issues and often the result of either insecure default configurations, faulty configuration, or random ad-hoc configurations. It is essential to always keep all operating systems and tools up to date. To do this, OWASP recommends the use of minimal platforms, repeatable processes for deploying new, identical platforms, and automated processes for verifying proper configurations.

7. **Cross-Site Scripting (XSS) attacks** can happen wherever a website includes untrusted data, which has not been adequately validated or escaped in a website. There exist two kinds of XSS attacks; reflected and persistent. Reflected XSS attacks happen when a
malicious request forces the web server to respond with HTML output that has been infected with trusted JavaScript code. A user, usually, has to interact with a malicious link (constructed by the attacker) for the attack to be executed. Persistent XSS attacks, however, utilize a weakness which allows for untrusted JavaScript code to be stored in the application. The code can then be inserted once, and run on the client without any involvement of the victim. OWASP recommends the usage of frameworks that automatically escape XSS by design. If one cannot implement such frameworks, one must escape untrusted data manually before the server sends it to the client.

8. **Insecure Deserialization** can lead to remote code execution, injection attacks, and privilege escalation attacks. Integrity checks (like digital signatures) and strict type constraints during deserialization are two useful tools when securely deserializing data.

9. **Using Components With Known Vulnerabilities** can result in severe data loss or server takeover. It is therefore essential to continuously monitor if the third party components in use are secure, and up to date. Sources like the National Vulnerability Database should be monitored to see if new vulnerabilities are discovered.

10. **Insufficient Logging And Monitoring** can allow attackers to continue to have access to vulnerable systems and allow them to attack other systems further. It is important to act quickly if an attacker compromises the system, and without proper logging and monitoring attacks can go unnoticed. All access control failures, server-side input validation failures should, therefore, be logged with sufficient user context to be able to identify malicious accounts. The logs should be generated in a format that can be consumed by a log management solution, and active monitoring should alert when suspicious activities are detected.

### 2.4 Discovering Potential Vulnerabilities

We have now defined a set of vulnerabilities but how can we assess if our application is vulnerable to any of the risks outlined in section with our restricted time-frame? Wu and Zhao divides software security testing into two main categories: Black-box and White-box testing. The grey-scale gradient within these names describes the level of insight in the application that the tester obtains. According to Wu and Zhao, Black-box testing infers that the tester has no previous inside knowledge of the application or access to the source code. In Black-box testing, the tester derives the tests from the application’s behaviors and functions. In contrast, White-box testing methods rely on the developers understanding of the underlying logic of the application. Further insight into the applications inner workings allows the tester to build more structured test cases from the objects source code. Khan and Khan made a comparison between Black-box, White-box, and Grey-box testing - a mix of the two main strategies. They found that since there exists no conflict between Black-box and White-box strategies a Grey-box testing approach which mixes the strengths of both Black-box and White-box testing can be preferable in all instances. Austin and Williams also researched the strengths and weaknesses of different testing methodologies based on both Black-box and White-box approaches. They found that the use of one method is often not enough to find a sufficient amount of different kinds of vulnerabilities. Automated penetration tests were excellent in finding many vulnerabilities in a short amount of time. However, automated penetration tests cannot find any design flaws, for that, a manual approach is needed. Austin and Williams do recommend a mixture of structured manual penetration testing, and automated penetration testing for the best time-efficient approach.
2.4. Discovering Potential Vulnerabilities

Manual penetration testing

Due to its dependency on time and expertise of the test, manual penetration testing is an expensive technique. Austin and Williams divides manual penetration testing into two main groups; exploratory testing, and systematic testing. Exploratory testing is when the manual penetration tests are done without a test plan, while systematic testing utilizes a predefined test plan. In their comparison of different penetration testing techniques, Austin and Williams found that the systematic penetration testing technique was more effective in finding vulnerabilities than exploratory penetration testing.

Automated penetration tests

Austin and Williams argue that while automated penetration testing is useful in finding a lot of vulnerabilities in a short amount of time. The automated penetration tests failed at finding a large number of vulnerabilities, leading the authors to recommend to use automated penetration tests as an addition to systematic manual penetration testing to speed up the repetitive stages of the manual approach. This mix should, according to the authors, provide the most time effective method. Many vulnerability testing tools exist on the market, both commercial and open-source, but when Idrissi et al. conducted their performance evaluation, they found no correlation between cost and performance. They conducted a study on eleven Web Application Vulnerability Scanners (WAVS). Five of which were under a commercial license and six of them were open-source. The study compared the true-positives (found vulnerabilities), false-negatives (missed vulnerabilities) and false-positives (discovered vulnerabilities that do not exist) between the WAVS on four different vulnerabilities; SQL-injection, XSS, Remote File Inclusion (RFI) and Local File Inclusion (LFI). What they did notice was that commercial scanners often have better features for finding vulnerabilities in web2.0 technologies such as AJAX and JavaScript, while open-source tools are better at LFI and RFI. However, they also found that the individual performance between different scanners varies a lot, and the authors recommend choosing a scanner based on what vulnerabilities are of interest. In the test cases presented by the authors, the ZAP web proxy and Vega were the two best performing WAVS. ZAP succeeded in finding all SQL-injection vulnerabilities, and all RFI vulnerabilities while discovering 95% of all XSS vulnerabilities, and 72% of all LFI vulnerabilities. Vega, on the other hand, found all of the SQL-Injection points, XSS vulnerabilities, RFI vulnerabilities, and 63% of the LFI vulnerabilities.

Vega

The open source security company Subgraph develops the Vega WAVS. Subgraph employs five security experts with extensive experience in web application security. The company markets Vega as a cross-platform vulnerability scanner that is designed to validate SQL Injection, XSS, disclosed sensitive information and other vulnerabilities. The application has three modes; Automated, Manual, and Hybrid. When using the automated scanner, Vega uses a crawler to find resources and automatically scans them for vulnerabilities. The manual mode utilizes an intercepting proxy which allows the user to manually inspect every HTTP packet sent from the client via the proxy. The hybrid mode replaces the automated scanner with the actions of the user. Instead of using the scanner, the proxy relies on the manual exploration of the website by the user to get resources. The proxy then automatically scans the resource for vulnerabilities.

However, the development of Vega has stopped, and with the last commit being pushed over two years ago the application can be expected to miss many of the newer vulnerabilities.
ZAP

The Zed Attack Proxy (ZAP) is a WAVS developed by the OWASP community. OWASP markets ZAP as one of the worlds most popular free security tools, and 91 registered contributors are actively maintaining their GitHub repository. The ZAP project has over 6000 recorded commits and is continuously updated [10].

ZAP comes with an extensive arsenal of tools and has a marketplace for add-ons if needed. ZAP and Vega are very similar since both applications are man-in-the-middle proxies which can be used to inspect websites manually. However, ZAP also has tools for automating testing procedures like a fully automated vulnerability scanner, a traditional spider, and an AJAX spider to help scan more modern, JavaScript-based applications.

Based on that ZAP and Vega had pretty much the same performance but that OWASP is keeping ZAP more recently maintained than Vega made it so that our choice in this project was to utilize the ZAP Attack Proxy.
3 Method

The following chapter will explain the selection of methodology in this paper. Two main topics will be covered; (1) Penetration testing to answer RQ1, and (2) interviews and surveys to answer RQ2.

3.1 Penetration testing

The penetration tests had to be conducted during a short time slot on the developers test-server, which runs a duplicate instance of the CRM system in production. This was needed due to the malicious data inserted into the system. To not interfere with the developers work, the tests were run during a weekend so that the developers could reset the database and continue to work the following Monday. Since we did not have access to any system specification from which we could derive a test plan as Austin and Williams did we chose to only rely on the results from the automated tests.

The tools used in the penetration test was ZAP and Firefox. The ZAP test suit took quite a long time, 7 - 10 hours to be exact. The test suit also allocated more resources than was available on the host, which resulted in the final test-results being a composition of several runs. The penetration test can be broken down into four sections; 1. Manual exploration, 2. Spider, 3. Attack, 4. validation. The first part of the penetration test, manual exploration, means to explore the website manually with ZAP intercepting the traffic. By doing this, we can capture important packets and identify the login/ logout procedures to ZAP so that this can be automated later in the ZAPs testing suite. The second part, the spider, is fully automated by ZAP and means to identify every resource on the site. ZAP utilizes both a traditional

![Figure 3.1: ZAP asking about keeping the session persistent.](image)
3.1. Penetration testing

For a more in depth test you should explore your application using your browser or automated regression tests while proxying through ZAP.

### Figure 3.2: Launching a preconfigured version of Firefox from ZAP.

![Launching a preconfigured version of Firefox from ZAP](image)

**Figure 3.2:** Launching a preconfigured version of Firefox from ZAP.

### Figure 3.3: Showing Firefox proxy settings.

![Showing Firefox proxy settings](image)

**Figure 3.3:** Showing Firefox proxy settings.

### Figure 3.4: Creating a new context from ZAPs Site view.

![Creating a new context from ZAPs Site view](image)

**Figure 3.4:** Creating a new context from ZAPs Site view.

spider and a more modern AJAX spider to fully explore the site. ZAP then uses the found resources in the third phase. In the attack phase ZAP tries to infect every resource and entry point found by the spider. ZAP analyzes the results and raises alerts if it finds any potential vulnerability. These alerts are summarized into a report which needs manual validation since they can be false-positives.

#### Setting Up The Penetration test

Due to the time restrictions, the choice was made to keep the settings of the scanner as default as possible. With this in mind, some settings needed to be defined before even the most basic scan could be executed.

The pop-up is shown in Figure 3.1 is the first thing that opens when ZAP is started. It asked if the user wants to persist the ZAP Session. Since we wanted to be able to go back to our attack results at a later time, the session was chosen to persist with a name based on the current timestamp. Next, the ZAP interface was used to open the Firefox web browser (See: Fig 3.2). This action opens a Firefox instance with the ZAP web proxy already configured as seen in Figure 3.3. Once Firefox was opened it was used to connect to the application. The domain will appear in ZAPs Site view where it can be expanded to show the resources discovered in that domain.
3.2. Interviews and Survey

Next ZAP needs a Context to limit the spider and attack functions. The Context is a concept in ZAP to indicate which domains are interesting, and which are not. We then configured ZAP to use the test user we were granted while performing the tests. To do this, we used the button shown in Figure 3.4 to create a new Context. Once this was done, we used the Firefox browser which was configured with the ZAP web proxy to log in with the test user. We, then, used the sites view to locate the login request, that was sent from the browser to the server. The request was selected and included in the context as a Form-based Auth Login Request. Once the Session properties window had appeared with the information from the login request a user was defined via the Users tab (See Figure 3.5). ZAP knows how to log in to the system; now it just needs to know when it has logged itself out. We clicked the logout button and recorded the traffic. We opened up the response to our logout request and highlighted it in ZAP. By doing this we could tell ZAP how to recognize the logout procedure.

ZAP now has a general understanding of how to log in to the application and how to know when it has logged itself out. With these basic configurations, the penetration tests started. The tests began with a run-through with the traditional spider, then the AJAX spider, and last the active scan. No additional settings were defined for either of the spiders, but some more parameters were adjusted for the active scan. Under the settings tab for the active scan window, we limited the use of technology to that of what the CRM-system was using. Since the application is written PHP and runs on an Ubuntu server, this meant deselecting ASP, C, and XLM as languages; MacOS and Windows as operating systems; and IIS and Tomcat as web servers.

3.2 Interviews and Survey

This paper bases its methodology regarding interviews and surveys on the work done by McCormac et al. [18] while examining cyber security awareness in the Australian government. We developed a survey to assess the knowledge and behavior regarding web application security among the developers. Since McCormac et al. [18] assessed general cybersecurity awareness, and not specifically web application security, we had to redesign the focus areas of the surveys and interviews. We derived a list of six focus areas from the OWASP Top 10 Most Critical Web Application Security Risks [23]. Some topics from the list did not qualify to become focus
3.2. Interviews and Survey

Figure 3.6: Example question examining the subjects knowledge regarding injection attacks areas in our study since the technologies they cover is not present in the application audited in this paper. The final list of focus areas was the following:

1. SQL-Injection attacks
2. Authentication
3. Sensitive Data
4. Cross-Site Scripting
5. External Components with Known Vulnerabilities
6. Insufficient Logging & Monitoring

Survey

The web-based survey that was constructed with these focus areas in mind, contained seven distinct sections:

**Introduction**  The survey started with a short introduction text with information about the survey and highlighted the fact that the survey was totally anonymous so that the developers would feel more inclined to provide truthfull answers, even though they might be uncomfortable.

**Background**  We collected information surrounding the backgrounds of our subjects to understand its role in their knowledge and attitude toward security. We, therefore, asked questions regarding the subject’s years of experience as a software developer, how many years they have been with the company, and their area of responsibility.

**Security-related education**  To analyze if previous education had resulted in a higher security awareness we had the subjects described previous security-related education.

**Self-assessment**  Sometimes people overestimate their capacity which might make them blind to their shortcomings. We wanted to examine if this was the case regarding cybersecurity topics and therefore had the subjects perform a self-assessment on their knowledge regarding web application security.
3.2. Interviews and Survey

Cybersecurity knowledge We constructed a vocabulary test to evaluate the subjects knowledge regarding cybersecurity and web application security topics. This section was constructed based on the work of Kruger et al. [17]. The vocabulary test examined the subjects knowledge regarding OWASP; the differences between a threat, vulnerability, and risk; how attacks like SQL-injection and XSS works; how to perform secure password storage; and where to find information about known vulnerabilities in applications and libraries. Fig 3.6 shows an example question from the survey designed to assess the subjects knowledge regarding injection attacks.

Security within the application We created a section to understand the work conducted by the subjects and their view on the security issues surrounding the application. This section asked the subjects to describe their view on what the most critical security risks were, what the most sensitive resources were in regards of confidentiality, what security mechanisms they had implemented, and if there existed any vulnerabilities at the time of questioning. This section goes on to ask about the subjects knowledge regarding previous penetration tests, testing procedures, and restrictions that hinders the subject from performing security-related tasks.

End section allowed the subject to send in further questions or comments on the subject. The actual questions, used during the survey can be found in Appendix 1.

Before the survey was released to the subjects the questions were sent and tested on some colleagues, all of which had studied computer engineering on at least a bachelor level. The test subjects provided feedback if any question was ambiguous, incorrect, or confusing. When the questions were approved by these test subjects they were sent out to the developers.

Interview

The interviews were conducted via voice-chat using the Telegram application because the interviewer was on another continent than the subjects. Appendix 2 shows the questions used during the interview. The questions in the interview allowed for more elaborate answers than those in the survey. Instead of merely having to describe a specific attack, the subjects were asked to elaborate on how the attacks are executed and describe prevention techniques that can be used in the context of the CRM-system they were developing. An example of such a question can be found in Figure 3.7. To examine more of the subject’s behavior, we asked questions regarding testing procedures, how they performed risk assessments and policies regarding password management. The interview also had a PHP specific security question.

We recorded the interviews so that the interviewer could focus on the interview instead of taking notes. The following part of this section will describe how we utilized the guide written by Gorden [13] to analyze our answers. As an example on the classes and categories were derived we will use the following quote from one of the developers when asked about what the most sensitive resources are that they store in their application. The developer said:

“Because we are a CRM system we have our customers partners, and customer information and we have all of their order purchasing information, and also all customers have their own personal information like email and names.”
3.2. Interviews and Survey

The first step in the process is to define the coding categories. According to Gorden[13],
these categories are to be "All-inclusive and mutually exclusive", meaning that the categories
are to be defined in such a way that it can be applied on all relevant answers while it must
be defined clearly enough, that a concrete answer cannot fall into two categories at the same
time. From this quote we derived four categories:

1. Customer information
2. Customer relationships
3. Order history
4. Sales associates contact information

When this step is done, one can assign symbols to the categories which will help with
summarizing and condensing the categories before one classifies relevant information. The
point of classification is to select the most important categories and relate them to each other.
In the example given above, the categories were classified in the following manner:

1. Customer information → Customer information
2. Customer relationships → Customer information
3. Purchasing information → Order information
4. Sales associates contact information → Sales associates information

More examples on how coding and classification was performed to analyze the quantitative
data collected from interviews can be seen in Table 3.1. Coding and classifying transcripts
from the interviews helped a great deal when comparing answers between the interviews.

PHP Type Juggling

Since the CRM-application is built using the PHP programming language, we found it neces-
sary to examine the developers’ knowledge about a PHP specific security topic. The topic we
chose to examine was how PHPs dynamic type casting (also known as type juggling). Type
juggling is one of the basic functionalities of PHP, and can cause security vulnerabilities if
not handled correctly. The developers were therefore presented with a snippet of PHP code
that was a part of a security bug in WordPress, the worlds most popular online publishing
platform, back in 2014[21].

```php
$hash = hash_hmac('md5', $username . '|' . $expiration, $key);
if ($hmac != $hash)
{
    // Report invalid cookie
}
// Valid cookie. Continue...
```

The code snippet contains a non-strict comparison when validating a cookie which could
allow for an attacker to forge authentication cookies by exploiting PHP’s type juggling system.
In this case, the user can manipulate the $username and $expiration variables to force the
MD5 algorithm to produce a string with the format "0e..." which, when compared to a $hmac
variable, that is an integer with the value zero, would be type-juggled to a zero as well, allowing
for the attacker to bypass the cookie validation mechanism.
Table 3.1: Table displaying examples on how coding and classification was done to analyze quantitative interview data

<table>
<thead>
<tr>
<th>Subject</th>
<th>Quote</th>
<th>Codes</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dev1</td>
<td>We use a web framework called Yii and there is actually a lot of wrappers and layers for protecting input data in the system. We use all of them. When we have data input we use integrated data layer and they take care about all of the input data and SQL injection. Protection for XSS and CSRF attacks exists as well. I feel secure when using this framework.</td>
<td>Yii</td>
<td>Utilizes the Yii Framework</td>
</tr>
<tr>
<td></td>
<td></td>
<td>XSS</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSRF</td>
<td></td>
</tr>
<tr>
<td>Dev2</td>
<td>Yes, I think I know what Cross-Site scripting means. It’s when you have a backend, and client side that sends request to the back-end. This could happen when something that you have not created makes request to you back-end. By default this Cross-Site scripting is disabled. It’s not a big problem. But, this could happen if you don’t take precautions.</td>
<td>Client-Server communication</td>
<td>Explained Cross-Site Request Forgery rather than XSS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSRF</td>
<td>Relaxed attitude</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Not a big problem</td>
<td></td>
</tr>
<tr>
<td>Dev3</td>
<td>This will be noticed in logs that are manually checked daily. There are no automated services implemented to do this. There was a project with implementing a WAF (Web Application Firewall) but the project has been halted. Implementing a WAF could easily be done but we need coordination with the development team so that we can perform testing. Otherwise the implementation of a WAF could harm the application. Here exists some communication problems between management, DevOps and Development for this to happen.</td>
<td>Manual checking</td>
<td>Manual inspection</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Implement WAF halted</td>
<td>Motivated to improve</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Communication issues</td>
<td>Hindered by management</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Communication issues</td>
</tr>
<tr>
<td>Dev4</td>
<td>Umm, well, for now, we do not implement any automated tests. So basically we test it out on the application itself. For example, before uploading to production, we have our dev-server (development). We first upload all of our code to our dev-server and we test out all of our new features there. And if we don’t find any problems, breaks or other things, we make sure about it, and we also ask our guys there in Sweden to test. If they agree that everything is fine, then we update our latest features, our latest code to the production server.</td>
<td>Manual testing</td>
<td>Manual and unstructured testing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No test-plan</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Testing with managers</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dev-server</td>
<td></td>
</tr>
</tbody>
</table>
When presented with the code snippet the developers were asked to find a security issue in the code. If the subject could not answer the question within five minutes, they were asked if they understood the algorithm, and got any question regarding the code answered. If they still could not answer the question regarding what was wrong with the algorithm, they were given a hint telling them that type juggling was the issue. If they still could not answer the question, they got the issue explained.
Survey and Interview Results

The following chapter contains the results from the interviews and surveys held with the developers.

4.1 Survey

The survey collected results from all three of the developers, and the system administrator responsible for the infrastructure setup. We designed the survey surrounding four subsections; background, self-assessment, cybersecurity knowledge, and the cybersecurity status of the application. These are all presented in the four tables labeled Table 4.1 to 4.4. Due to the small group of developers that were available for questioning it would be easy for anyone who knows the developers by person to understand what developer said what based on the answers given in Table 4.1. With this in mind we have chosen to mix the developers answers around and change their labels in this particular table. Table 4.2 and Table 4.5 shows the answers to the section regarding Cybersecurity Knowledge. Since this section was designed to assess the developer’s knowledge of cybersecurity-related topics we color-coded the answers to indicate when the developers answered correctly. Green answers are correct, and Red answers are incorrect. Table 4.4 and Table 4.5 shows the answers provided to the section we chose to call “Security Within the Application”. This section contained questions more specific to the CRM-system we were examining. Here there is no right or wrong; the purpose was to collect the opinions of the developers. Since we do not wish to disclose the name of the system (due to the sensitive information in this report) we replaced the name of the application with stars.

Background The subjects were found to have a wide range of work experience with two reporting 3-6 years of experience, while the other two reported 6-9 and 10+ years of experience respectively. Half of the team is relatively new to the organization and have no more than two years of experience within the company, and only one subject had been with the company for more than four years. There is also no real separation of responsibility in the organization. They have one person in charge of the infrastructure and server configuration, but among the developers, there is no division between front-end or back-end development. All developers responded that they do a bit of both.
Table 4.1: Survey results from the survey sections of Background, Security Related Education and Self Assessment. With anonymized labels for the developers.

<table>
<thead>
<tr>
<th>Developer</th>
<th>Years of Experience</th>
<th>Years at Work</th>
<th>Areas of Responsibility</th>
<th>Have You Taken</th>
<th>Type of Security Education</th>
<th>Education Details</th>
<th>General Knowledge of Web Application Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>DevA</td>
<td>10+ years</td>
<td>2-4 years</td>
<td>DevOps, SysOps</td>
<td>Yes</td>
<td>Passed AWS SysOps</td>
<td>Certification</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DevB</td>
<td>3-6 years</td>
<td>0-2 years</td>
<td>General software</td>
<td>No</td>
<td>-</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>development</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DevC</td>
<td>6-9 years</td>
<td>4+ years</td>
<td>General software</td>
<td>No</td>
<td>-</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>development</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DevD</td>
<td>3-6 years</td>
<td>0-2 years</td>
<td>General software</td>
<td>Yes</td>
<td>Mainly online: blogs,</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>development</td>
<td></td>
<td>books, and best practices</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 4.2: Survey results from the survey section “Cybersecurity Knowledge” part 1.

<table>
<thead>
<tr>
<th>Dev1</th>
<th>Yes</th>
<th>The ten most critical web application security risks</th>
<th>A newly discovered incident with the potential to do harm to a system.</th>
<th>A known weakness of a resource that can be exploited by an attacker.</th>
<th>The potential for loss or damage when a system is compromised.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dev2</td>
<td>No</td>
<td>I don’t know</td>
<td>A known weakness of a resource that can be exploited by an attacker.</td>
<td>A newly discovered incident with the potential to do harm to a system.</td>
<td>The potential for loss or damage when a system is compromised.</td>
</tr>
<tr>
<td>Dev3</td>
<td>Yes</td>
<td>The ten most critical web application security risks</td>
<td>A newly discovered incident with the potential to do harm to a system.</td>
<td>A known weakness of a resource that can be exploited by an attacker.</td>
<td>A newly discovered incident with the potential to do harm to a system.</td>
</tr>
<tr>
<td>Dev4</td>
<td>No</td>
<td>The ten most critical web application security risks</td>
<td>The potential for loss or damage when a system is compromised.</td>
<td>A known weakness of a resource that can be exploited by an attacker.</td>
<td></td>
</tr>
</tbody>
</table>
Table 4.3: Survey results from the survey section “Cybersecurity Knowledge” part 2.

<table>
<thead>
<tr>
<th>What is an injection attack?</th>
<th>What does it mean to parameterize SQL-queries?</th>
<th>What is an XSS attack?</th>
<th>Select the techniques you find necessary to use when storing passwords in a database.</th>
<th>What is TLS?</th>
<th>Where can you find information about known vulnerabilities?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Malicious input is given to the system that is later processed as a command or query which alters the programs execution.</td>
<td>It’s when database queries are done through stored procedures, and the arguments are sent as parameters to the stored procedure.</td>
<td>It allows an attacker to target different users, and run malicious code on their clients.</td>
<td>Hashing, Salting</td>
<td>A protocol for transferring encrypted information over the internet.</td>
<td><a href="https://www.us-cert.gov/ncas/bulletins">https://www.us-cert.gov/ncas/bulletins</a></td>
</tr>
<tr>
<td>Malicious input is given to the system that is later processed as a command or query which alters the programs execution.</td>
<td>It’s when SQL statements are sent to and parsed by the database server separately from it’s parameters.</td>
<td>It allows an attacker to target different users, and run malicious code on their clients.</td>
<td>Hashing, Salting</td>
<td>I don’t know</td>
<td>I don’t know</td>
</tr>
<tr>
<td>All of the above</td>
<td>All of the above</td>
<td>All of the above</td>
<td>Hashing, Salting</td>
<td>A protocol for transferring encrypted information over the internet.</td>
<td><a href="http://www.owasp.org">www.owasp.org</a></td>
</tr>
<tr>
<td>All of the above</td>
<td>when database queries are done through stored procedures, and the arguments are sent as parameters to the stored procedure.</td>
<td>It allows an attacker to target different users, and run malicious code on their clients.</td>
<td>Hashing, Salting, Hash stretching</td>
<td>A protocol for transferring encrypted information over the internet.</td>
<td><a href="http://www.nvd.nist.gov">www.nvd.nist.gov</a></td>
</tr>
</tbody>
</table>
### Table 4.4: Survey results from the survey section “Security Within the Application” part 1.

<table>
<thead>
<tr>
<th>Developer</th>
<th>What are the most critical security risks about ****?</th>
<th>What are the most sensitive resources stored in the **** database?</th>
<th>What security mechanisms have been put in place to minimize these risks?</th>
<th>Do you know of any vulnerabilities in the **** application as of today?</th>
<th>If you answered “Yes” on the question above, what are they?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dev1</td>
<td>I'm not sure about the web app code</td>
<td>Users’ personal data</td>
<td>We aimed to be GDPR compliant, so we will implement mechanisms related to this compliance</td>
<td>No</td>
<td>-</td>
</tr>
<tr>
<td>Dev2</td>
<td>No CRITICAL risks</td>
<td>User information, sales, and distribution data</td>
<td>Servers can be accessed by specific IP addresses, and access to servers is gained over several security steps.</td>
<td>No</td>
<td>-</td>
</tr>
<tr>
<td>Dev3</td>
<td>We have a data of different companies and any leaks of this information can cause a damage to our customers</td>
<td>Customer’s Orders history and personal information of their business partners</td>
<td>We have used all the available security libraries and mechanisms of Yii Framework</td>
<td>No</td>
<td>-</td>
</tr>
<tr>
<td>Dev4</td>
<td>DDoS attacks, injection, xss</td>
<td>customer financial data, personal information, company information</td>
<td>application is written on secure framework - Yii which handles security issues such as: XSS, CSRF, prevention of attacks via cookies etc.</td>
<td>Yes</td>
<td>Weak passwords</td>
</tr>
</tbody>
</table>
Table 4.5: Survey results from the survey section “Security Within the Application” part 2.

<table>
<thead>
<tr>
<th>**** has earlier had one of their customers perform a penetration test on one of their other applications. Do you know which application they tested, and what vulnerabilities were found?</th>
<th>Before releasing a new update to the **** application do you, personally, perform any security related tests?</th>
<th>What hinder you from performing more security focused work?</th>
<th>Other related questions or comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dev1</td>
<td>No</td>
<td>No</td>
<td>It’s quite difficult to coordinate and organize procedures and tasks related to the security with the development phases</td>
</tr>
<tr>
<td>Dev2</td>
<td>No</td>
<td>No</td>
<td>Time</td>
</tr>
<tr>
<td>Dev3</td>
<td>I think I didn't understand a question</td>
<td>No</td>
<td>Time</td>
</tr>
<tr>
<td>Dev4</td>
<td>No</td>
<td>No</td>
<td>Time</td>
</tr>
</tbody>
</table>
Security Related Education  Two subjects answered that they had had some previous security focused education. One responded to have trained to and have passed the AWS SysOps certification exam that has a specific section designated for security. While this can be considered formal education, the other subject answered to have studied blogs, books, and best practices, which we cannot consider to be any formal education.

Self-assessment  They all assessed that they had good general knowledge about security with no one grading themselves lower than three on a 1 - 5 scale.

Cybersecurity Knowledge  Even though nearly all of the responders acknowledged that they had heard about the vulnerabilities composing the OWASP Top 10 list, no one had heard about the list itself. The questions, assessing the subject’s understanding of the concepts of "Threat", "Vulnerability" and "Risk", found that there is some disagreement in the organization with the meaning of these words. Only half of the subjects knew the meaning of a "Threat" while two-thirds knew the meaning of "Vulnerability" and "Risk." When questioned about the different web hacking techniques half of the subjects knew about the goal of general injection attacks. However, when asked about how parameterization of SQL-queries work, only one person knew that parameterization is when the parameters and query are sent separately to the server. All respondents know that XSS - or Cross-Site Scripting - attacks allow an attacker to target different users by running malicious code on their client. When questioned about the techniques necessary for storing passwords in a database it was agreed upon by all subjects that Hashing and Salting are necessary techniques, but only one recognized Hash stretching as another necessary technique. Three out of four knew that TLS is a protocol for transferring encrypted information over the internet, but one thought that it was a predecessor to the deprecated SSL protocol.

Security Within the Application  When questioned about security related to their CRM-system, the answers became inconsistent; Two subjects answered that the users’ data is the most sensitive data, while others found that the customer order history, business partners, and financial data are the most critical resources if they were leaked. When answering the question about the most critical security risks with the application are, one responded that no critical risks are surrounding the application. While one expressed concern for DDoS attacks and injection attacks, another said that:

"We have data of different companies, and any leaks of this information can cause damage to our customers."

When asked about what security mechanisms are used to minimize the risks concerning the application, the majority relied on the use of the Yii Framework to mitigate attacks like XSS and SQL-injection. To the question "Do you know of any vulnerabilities in the application as of today?" There was only one respondent, which expressed that weak passwords are a vulnerability concerning the application. The survey also revealed that there are no security specific tests performed before a new version of the software is released to the production servers. When asked about what stops the developers from focusing more on security-specific work three out of four said that they do not have the time to prioritize security-related work, while one said that it is:

"...difficult to coordinate and organize [security related] procedures and tasks [within the organization]"

\footnote{A common protection technique used to defend against SQL-injection attacks}
4.2 Interviews

We succeeded in arranging interviews with all the developers in charge of the application. Overall it was found that there is a lack of knowledge about security-related topics, confusion regarding what resources are risks and prioritization, and a lack of testing procedures.

Background and Working Environment

Even though no one had heard about the OWASP organization before most of them acknowledged that they knew about the security risks regarding web applications that was listed on the OWASP top 10.

The interviews revealed that there is a firm reliance on the developers on other people and technology to take care of software security. Many of the developers rely on the Yii framework to handle input validation and injection attack mitigation. One of the subjects expressed this in our interview when asked about how they handle injection attack mitigation in the application:

“We use a web framework called Yii, and there is a lot of wrappers and layers for protecting input data in the system. We use all of them. When we have data input, we use integrated data layer, and they take care about all of the input data and SQL injection. Protection for XSS and CSRF attacks exists as well. I feel secure when using this framework.”

Another developer relied on the system administrator to have performed a risk assessment when asked if any risk assessment had been done on the application:

“From our side, no, but I think it probably was done by our server administrator, that we have. We have a separate guy doing the administration of our servers, and I think that is probably going to be a question for him.”

Risks

When we asked the subjects about what the most sensitive resources are that are handled by the application it became clear that no one had done any kind of risk assessment. The subjects presented us with a wide range of different kinds of data, ranging from users personal data to orders, and financial statistics. These answers do not conform to the answers from the managers who say that pricing could be the most damaging data for their customers if it were to leak from the application.

No developer knows how they could notice if someone attacked their system. There is no automated system, like a Web Application Firewall (WAF) in place that could send notifications if an attack is detected. One of the subjects in the interview stated that there had been plans to implement a WAF into the infrastructure, but the plans had been stagnated because of the lack of communication and drive between management, system administrator, and developers. The success of the WAF implementation relies on good cooperation due to the high likelihood that the implementation could disturb the application before the WAF have been configured appropriately.

Knowledge Regarding Attacks & Prevention Techniques

From the interviews we can conclude that the developers possess a good general understanding of SQL-injection attacks and countermeasures, there is, however, a lack of understanding in how to prevent XSS attacks, and some developers seem to confuse XSS attacks with Cross-Site Request Forgery (CSRF) attacks. Here’s a part of the answer given in the interview by one of the developers when asked how to mitigate XSS attacks:
“As I said before, The Yii Framework generates this CSRF-token so that our backend knows that requests are being sent from the content we produce. This token is constructed every time a request is made. So if there is no CSRF-token, or an invalid CSRF-token is provided, then the request is rejected. That is one of the basic precautions we make.”

This explanation does not conform with the OWASP guidelines for XSS attack prevention, which recommends escaping HTML characters before writing data to the HTML document[24] but instead explains a technique used to prevent CSRF attacks. Further, when presented with the code snippet showing a PHP type juggling bug from WordPress in 2014 none of the developers could accurately describe the issue with the code. When informed that the issue regarded PHP type juggling, none of the developers knew about the concept of type juggling.

Testing

The organization utilizes a manual testing approach where, if the CRM-system is to be patched or updated with new features, the new source code is uploaded to a dedicated test server (regarded by the developers as the dev-server). When the code has been uploaded to the test-server, the developer (or developers, depending on the size of the update) test the new features. The performed tests are based on the functionality specified for the new feature. Even though there exists no documented specifications, the developers follow the supposed release-notes for that update to see if everything works as expected. If the release is a part of a larger update or feature release, they incorporate some of the managers to help with testing. Despite this, the developers acknowledged that they do not perform any kind of security-specific testing. The testing procedure was considered as being one of the largest security weaknesses in the organization by one of the more experienced developers:

“Subject: No, actually this is maybe one of the weaknesses in our system. We don’t have any unit testing, and yeah, actually, we had the plan that right now we need a proof of concept, later we can build a better system with automatic unit testing, and we should do it later, and now the system is already so big it’s hard to implement unit testing in this stage.

Interviewer: So you are doing manual testing?

Subject: Yeah, we are doing manual testing, and if its part of some bigger release we have a development server, and we release everything to our dev-server and do manual testing there if everything works fine, then we merge to the release branch which takes it to the production servers.”

Since there exists no unit tests, no test plan, and no security specific test cases are performed there exists a large possibility of vulnerabilities going unnoticed into production. Other than allowing for security weaknesses, the lack of proper testing procedures can also allow for bugs to be released into the production environment.
Penetration Test Results

As stated in Section 3.1, we performed the penetration test in four steps: manual exploration, spider, attack, and validation. We did not find much during the manual stage of the penetration test. The only thing we found was that the application saved unencrypted user data in one of its cookies. The main point of the manual exploration was to record traffic in ZAP and help ZAP with recognizing the login procedure. When the proxy had been instructed to recognize the login procedure, we started the scans.

The spiders mapped the application and signaled for some XSS vulnerabilities, but it was the attack phase that found the majority of the alerts we later validated. Table 5.1 shows the results of the scan. ZAP also tested a set of other vulnerabilities, but since they are out of the scope for our study, we chose not to include them in our results. The tests we chose to focus our attention to were the Cross-Site Scripting tests, Injection tests (SQL-injection, Server Side Code Injection, and Remote OS Command Injection), and deserialization tests (Path traversal, Directory Browsing, and Parameter Tampering). Since we were using the testing server for our penetration test, no developer could work during the testing phases. The penetration tests were, therefore, conducted during a weekend and the short time interval limited us to how many times we could redo the tests in case anything went wrong. This can be noted in the amount of progress made on the SQL Injection tests. As we ran into a problem with SSL

<table>
<thead>
<tr>
<th>Name</th>
<th>Progress</th>
<th>Elapsed (min:sec)</th>
<th>Requests</th>
<th>Alerts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross Site Scripting (Reflected)</td>
<td>100 %</td>
<td>29:38</td>
<td>20400</td>
<td>190</td>
</tr>
<tr>
<td>Cross Site Scripting (Persistent) - Prime</td>
<td>100 %</td>
<td>09:23</td>
<td>6465</td>
<td>0</td>
</tr>
<tr>
<td>Cross Site Scripting (Persistent) - Spider</td>
<td>100 %</td>
<td>01:48</td>
<td>859</td>
<td>0</td>
</tr>
<tr>
<td>Cross Site Scripting (Persistent)</td>
<td>100 %</td>
<td>02:45</td>
<td>138</td>
<td>2</td>
</tr>
<tr>
<td>SQL Injection</td>
<td>38 %</td>
<td>97:19</td>
<td>67169</td>
<td>21</td>
</tr>
<tr>
<td>Server Side Code Injection</td>
<td>100 %</td>
<td>04:49</td>
<td>5200</td>
<td>0</td>
</tr>
<tr>
<td>Remote OS Command Injection</td>
<td>100 %</td>
<td>11:36</td>
<td>12477</td>
<td>0</td>
</tr>
<tr>
<td>Path Traversal</td>
<td>100 %</td>
<td>114:36</td>
<td>106721</td>
<td>168</td>
</tr>
<tr>
<td>Directory Browsing</td>
<td>100 %</td>
<td>00:35</td>
<td>277</td>
<td>0</td>
</tr>
<tr>
<td>Parameter Tampering</td>
<td>100 %</td>
<td>05:54</td>
<td>5340</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>278:23</td>
<td>220240</td>
<td>381</td>
</tr>
</tbody>
</table>
during our tests that made ZAP make invalid SSL requests to the server. The server then answered with 502 Bad Gateway errors and we had to cancel the SQL Injection test and never had the time to redo the test.

The tests resulted in a total of 381 alerts, all of which we manually inspected to evaluate if the alert was a false-positive or not. While all SQL-Injection alerts and all Path Traversal alerts were shown to be false positives we identified two persistent XSS vulnerabilities and two reflected XSS vulnerabilities. Persistent XSS vulnerabilities exist in fields belonging to order types and region names, while the reflected XSS vulnerabilities exist in fields belonging to date and store id. When we combine the persistent XSS vulnerability with the fact that unencrypted user information was saved in one of the site’s cookie we can use this XSS vulnerability to access the session cookie and steal it. Figure 5.1 shows how we could access the content via this XSS vulnerability. Even though we tested this on a test server, we still consider the content of that cookie to be sensitive. We, therefore, chose to blur the picture.

Figure 5.1: Alert box altered to show cookie content
During our tests, we found a set of critical vulnerabilities that an attacker could use to infiltrate organizations present in the CRM system we audited. The following section will describe an approach which can be used by an assailant to gain administrative control over the customers’ instance in the CRM-system.

The attack begins by gaining control over a sales associates account. As we see it there are two primary ways of doing this; 1) construct a fake login-page of the CRM-system. The link to the artificial login-page is then sent in a phishing mail designed to fool the user to login using the fake webpage. 2) Due to the insufficient password policies enforced by the application, there is a high possibility of weak passwords being present in the database. The attacker can use the weak passwords to guess the password of a given user easily. Once the attacker has control of a sales associates account, it would be easy to utilize the persistent XSS attack we demonstrated in Figure 5.1. The payload of the attack can sniff for cookies of the active users browsing the webpage and send these to a server owned by the attacker. By having control over the session cookies, the attacker can now impersonate every user in the organization, including administrators. This attack is, however, isolated to individual organizations within the application. We found no way that an attacker can become a global administrator, and in that way access all organizations registered to the system. This attack would also probably go unnoticed by the company as well since the logging and monitoring solutions implemented are insufficient to find these kinds of attacks.

So why do the vulnerabilities which enable this attack to exist? We found a set of psychological, technical, real-world, and responsibility factors that contribute to this situation.

6.1 Psychological Factors

As the results in section Figure 5.1 show; the developers inhibit a good amount of knowledge regarding SQL-injection attacks, but a lack of knowledge surrounding what XSS attacks are, and how to prevent them. When asked about how to avoid XSS attacks, many of the developers instead referred to techniques used in order to avoid CSRF attacks, which are entirely different from XSS attacks. Without sufficient knowledge regarding what these attacks are, and how to avoid them there is no way for the developers to protect against them, mainly since they perform the testing manually by themselves.
There is no unity in the organization either, regarding what risks the application is subjected to. Each developer had a different answer to what critical resources are stored in the database. This lack of consensus can be contributed to that no one had done any risk assessment regarding the risks surrounding the application. This leads the developers to be unaware of what resources they need to be more careful with while developing their application. This leads to developers underestimating the risks that threaten the CRM-system.

6.2 Technical Factors

The technical factors contributing to the vulnerabilities are the lack of testing procedures. When the company only focuses on unstructured, manual testing, they run a significant risk in overseeing bugs or vulnerabilities leaking out into their production environment. The tool used in this research, ZAP Web Proxy, can be implemented into a continuous integration solution. This means that all of the errors we found, could have been detected before the vulnerabilities ever found their way onto the production servers. The lack of any structured testing policy is what we determine to be one of the major contributors to why these vulnerabilities exist since, with a better testing policy, these could have easily been found.

6.3 Real-World Factors

In our surveys and interviews, we found that the developers have the motivation to do more security-related work. But real-world factors are hindering them from executing their wishes. Some developers argued that there exists a communication gap between the different departments, and some argued that the focus on continuously releasing new features has hindered them from assuring quality, and testing their security.

These factors do not directly contribute to our findings of vulnerabilities in the CRM-system, but they explain why the insufficient testing procedures exist. The quote in Section 4.2 proves this point.

6.4 Responsibility Factors

We did also find evidence of responsibility issues. The developers focus a lot of trust into the Yii Framework to handle security-related issues. This trust results in a relaxed attitude towards security issues. Some developers find that since Yii takes care of these issues, there is no need for testing against these vulnerabilities. Unfortunately, since we found some significant vulnerabilities, this trust is misplaced. If the vulnerabilities show weaknesses in the data management of the Yii Framework, or if these vulnerabilities are signs of the developers not fully implementing the Yii security features is not something we can answer. To analyze that we need access to the source-code, which we did not have.

When we asked the developers about risk assessments and logging we found that the developers trusted this responsibility to the server administrator even though, the security of the application layer falls on the developers, rather than the server administrator.

6.5 Work In A Wider Context

With these factors in mind, we recommend that the company invests in better testing procedures, security-focused education for their developers and to conduct a risk assessment regarding the application to get to terms with the disagreements surrounding risks and what data is sensitive to a breach in confidentiality or not. As we found that many developers exhibited a willingness to focus on code quality and security we also recommend that the company look into reducing the workload or to hire more personal. A more aware workforce that has the time to focus on code quality and security would increase the security of the application, and
minimize the risk of sensitive data exposure for the company’s customers which can be very expensive for the company.
Limitations & Future Work

While the findings of this research are discussed in Chapter 6. It is important to analyze the factors that can invalidate our findings. This chapter is dedicated to analyzing the factors that can affect the internal, and external validity of our project.

7.1 Internal Validity

The following section will discuss factors that can cause a problem with the study’s internal validity. We will discuss our choice of methodology and how these can interfere with the validity of our results.

Validity of Interviews & Surveys

When assessing the security awareness of our subjects we were restricted by the physical distance between us and the subjects. With us being stationed in Sweden, and the subjects in Uzbekistan it proved hard to arrange for us to travel to Uzbekistan to perform the security assessment face-to-face during our short time-frame. Our choice of phone interviews and surveys can allow for the subjects to be dishonest in their answers. Since they have to be on a computer when answering our questions they can easily use the computer to look-up information before providing an answer. We tried to address this by keeping the interviews and survey results anonymous. In this way, they would not feel threatened by complications if they provided an incorrect answer. To also motivate the subjects to provide accurate answers we highlighted that their opinions will be take up with management, meaning that this was an opportunity for them to raise awareness of problems they might not raise otherwise.

It came as a surprise that Skype was blocked in Uzbekistan during the time for this project. It limited us to use Telegram without video when we performed our interviews. Phone interviews and surveys can also increase the risk of misunderstandings between the interviewer and the subject because of the lacking visual queues. This could have an even greater effect on our study since neither the interviewer or the subjects were native English speakers. Even though the English quality of both the interviewer and subjects seemed sufficient enough to execute successfully, the interviews and surveys both were designed to end with open questions which asked if the subject had any questions, comments, or had problems understanding anything
in the conversation. We also tested our questions on students with similar education as our subjects to ensure that the questions were easy to understand.

Validity of Penetration Tests

Our choice of only performing automated tests excluded the possibility of finding design flaws as pointed out by Austin and Williams. Further investigations are needed to evaluate the status of design flaws in the system. With this said, however, there exists no silver bullet, and one cannot be completely certain if all vulnerabilities have been found, even after extensive testing.

We did encounter some problems in the testing phase that could have resulted in some SQL-injection vulnerabilities being left unfound. When first launching the application against our target site, the CRM-system, an error occurred. The server answered with a 502 Bad Gateway response because of a handshake_failure. According to the ZAP FAQ page, the handshake failure is caused by a bug in the Java Runtime Environment (JRE), and ZAP has set up recommendations for solutions to this problem. However, none of these proposed solutions resolved our problem. What was noticed however was that when toggling the TLS1.2 setting in the Connections tab of ZAPs Options window it temporarily solved the issue. After days of work trying to resolve this matter, a workaround was set up using the Macintosh Macro Recorder by MurGaa. The macro recorder was used to record the keypresses and mouse clicks used to open up the settings tab, click the TLS1.2 check-box and save the new settings. The macro recorder then looped these interactions with a 30-second interval. With this setup, the tests could continue without interruptions.

When the tests then continued, we noticed that the computer running the tests started to slow down. This was because ZAP allocating more RAM than available on the host. Eventually, the computer became so slow that we risked crashing the computer and lose all progress. The tests had then been running for seven hours, and since we were in a tight time-slot, we choose to terminate the tests and rerun the test-cases that were left. The SQL-injection test was only completed to 40% before a choice was made to stop the test, and due to our tight time-slot, we did not have time to rerun that particular test. During this test ZAP only found false-positives, and it is possible that ZAP could have found some True-positives, had ZAP been allowed to finish the test.

7.2 External validity

This paper has shown how automated WAVS can be used to quickly get an insight into the security status of any web application, we then tried to explain this status by assessing the security awareness of the applications developers. The method we used to assess security awareness has been proved to be effective in assessing security awareness among other IT-related professions. With some of our questions being specific to the organization subjected to our research, we suggest redefining these questions before subjecting other organizations to the same evaluation. With these questions redefined according to the appropriate context, one can easily apply the same methodology of vocabulary test and interviews to assess the security awareness among developers in other domains than just web applications. However, one needs to reconsider the use of vulnerability test approach, since our use of automated WAVS is specific to web applications.
Conclusion

**RQ1**: Asked what vulnerabilities mentioned by the OWASP top 10 list can be identified in the CRM system using the ZAP web-proxy. With the help of the ZAP web proxy, we found that there exists XSS vulnerabilities (Both reflective and persistent). With the help of interviews and survey, we found that there also existed vulnerabilities in the areas of:

1. **Broken Authentication**: Bad password policies, and insufficient password hashing
2. **Sensitive Data Exposure**: Unencrypted user data existing in session cookie
3. **Using components with known vulnerabilities**: Insufficient policies regarding control of vulnerabilities in third party components
4. **Insufficient logging and monitoring**: Manual monitoring of web server request logs

**RQ2**: asked if there exists a relationship between the vulnerabilities found, and the web application security awareness among the developers. In this case, we can find a relationship between the discovered XSS vulnerabilities and the lacking knowledge of the developers in mitigating XSS attacks. This does however not tell the whole story of why these vulnerabilities exist. The developers describe a working environment with a lot of pressure on producing new features for the application, which makes them unable to properly test and validate the application. An “It’s not my responsibility” mentality exist as well among the developers. Their trust in the Yii-framework is especially strong. This mentality together with the found disagreement regarding risks and valuable resources contained within the application sends an alarming signal, and signals that there is a big chance that some vulnerabilities might slip through just because someone thinks it is another persons responsibility, while that person does not value the sensitivity of that mechanism.

Overall there exists a lack of knowledge among the developers when it comes to web application security. The existing testing policy is insufficient and could result in new security vulnerabilities and/or bad code quality. The developers are pressured on time to develop new features, which hinders them from conducting security specific work.

We have, in this paper, found some serious vulnerabilities and found indications that application security is somewhat neglected during the development cycle of the CRM-system in question. This needs to be improved, otherwise, there is a real potential that the application will be hacked in the future.
Bibliography


[8] OWASP Foundation. *About The Open Web Application Security Project - OWASP*. URL: https://www.owasp.org/index.php/About%7B%5C_%7DThe%7B%5C_%7DProject%7B%5C_%7DCore%7B%5C_%7DPurpose (visited on 05/30/2018).


[23] OWASP. “OWASP Top 10 - The Ten Most Critical Web Application Security Risks”. In: Oasw (2017), p. 22. URL: https://www.owasp.org/images/7/72/OWASP%7B%5C_%7DTop%7B%5C_%7D0%7B%5C_%7DD%7B%5C_%7DDay%7B%5C_%7D1%7B%5C_%7D10%7B%5C_%7D1%7B%5C_%7D%7B%5C_%7D0%7B%5C_%7D1%7B%5C_%7D28%7B%5C_%7D28%7B%5C_%7D%7B%5C_%7D0%7B%5C_%7D%7D2%7B%5C_%7D29%7B%5C_%7D29.pdf.pdf.


[37] Yahoo. Yahoo. URL: https://se.yahoo.com/?p=us%7B%5C&%7Dguccounter=1 (visited on 06/18/2018).

[38] Yii. Yii PHP Framework. URL: https://www.yiiframework.com/ (visited on 06/20/2018).
Appendicies

.1 Appendix 1
Hello,

This survey is constructed as a part of a bachelor thesis project designed to examine the security surrounding the **** CRM system. The goal of this survey is to provide that first insight into the **** organization and see how security issues are handled today. Due to the nature of some of the questions, the survey is anonymous. No one will be tied to their answers.

The survey is constructed of 4 sections:
1. Background
2. Self-assessment in how good, you perceive your knowledge about cybersecurity to be
3. Knowledge of certain security concepts
4. Cybersecurity questions regarding the **** application.

This survey will be accompanied by a follow-up interview. You can read more about that interview in the mail that was sent to you.

* Required

Background

The point of this section is to collect some basic information about you, the developers. The test is, as we've stated, anonymous.

1. How many years of experience do you have as a software developer? *
   
   Mark only one oval.
   
   □ 0-3 years of experience
   □ 3-6 years of experience
   □ 6-9 years of experience
   □ 10+ years of experience

2. How many years have you been working for ****? *
   
   Mark only one oval.
   
   □ 0-2 years
   □ 2-4 years
   □ 4+ years
3. What are your areas of responsibility at ****? *

Mark only one oval.

- General software development
- Testing
- Dev-Ops
- Front-end
- Back-end
- Application security
- Other:

Security related education
In these two questions we want to know about if you have any education within security. With security focus we mean education, courses or seminars that have had security as a main topic.

4. Have you taken any security focused education, courses or seminars? *

Mark only one oval.

- Yes
- No

5. If you answered "Yes" on the question above. What kind of security focused education have you taken?

________________________________________________________________________
________________________________________________________________________
________________________________________________________________________

Self assessment
In this section, we ask you to assess your own knowledge in the field of web application security.

6. How good is your general knowledge about web application security? *

Mark only one oval.

1 2 3 4 5
Rookie ☐ ☐ ☐ ☐ ☐ Expert

Cybersecurity knowledge
In this section, we ask you to answer some cybersecurity-related questions. Try to be as truthful as possible, and don't look up any answers. Remember that the survey is anonymous, and we will not hold you accountable if you give an incorrect answer to any of the questions.
7. Have you ever heard of the Open Web Application Security Project (OWASP)? *
   Mark only one oval.
   - Yes
   - No

8. What is the OWASP Top 10 list composed of? *
   Mark only one oval.
   - The ten best cyber web application security best-practices.
   - The ten most vital web application security mechanisms
   - The ten most critical web application security risks
   - All of the above
   - I don't know

9. What is a "Threat" in the context of web application security? *
   Mark only one oval.
   - A newly discovered incident with the potential to do harm to a system.
   - A known weakness of a resource that can be exploited by an attacker.
   - The potential for loss or damage when a system is compromised.
   - I don't know.

10. What is a "Vulnerability" in the context of web application security? *
    Mark only one oval.
    - A newly discovered incident with the potential to do harm to a system.
    - A known weakness of a resource that can be exploited by an attacker.
    - The potential for loss or damage when a system is compromised.
    - I don't know.

11. What is a "Risk" in the context of web application security? *
    Mark only one oval.
    - A newly discovered incident with the potential to do harm to a system.
    - A known weakness of a resource that can be exploited by an attacker.
    - The potential for loss or damage when a system is compromised.
    - I don't know.
12. **What is an injection attack?** *  
*Mark only one oval.*  
- A technique used where an attacker supply untrusted input to inject invalid user data to the system.  
- A technique designed to extract data from databases.  
- Malicious input is given to the system that is later processed as a command or query which alters the programs execution.  
- All of the above  
- I don't know

13. **What does it mean to parameterize SQL-queries?** *  
*Mark only one oval.*  
- It's when database queries are done through stored procedures, and the arguments are sent as parameters to the stored procedure.  
- It's when SQL statements are sent to and parsed by the database server separately from it's parameters.  
- All of the above  
- I don't know

14. **What is an XSS attack?** *  
*Mark only one oval.*  
- It allows an attacker to run malicious code on the webserver.  
- It allows an attacker to target different users, and run malicious code on their clients.  
- A way of extracting data from a database.  
- All of the above  
- I don't know

15. **Select the techniques you find necessary to use when storing passwords in a database.** *  
*Check all that apply.*  
- Hashing  
- Encryption  
- Encoding  
- Salting  
- Serialization  
- Hash stretching  
- I don't know
16. What is TLS? *

*Check all that apply.*

- A protocol for transferring encrypted information over the internet.
- A predecessor to SSL.
- Trusted Layer Software; a certificate approving secure websites.
- A library designed for high-performance websites.
- Hash stretching
- I don't know

17. Where can you find information about known vulnerabilities? *

*Mark only one oval.*

- www.nsa.gov
- www.nvd.nist.gov
- www.owasp.org
- www.defense.gov
- www.pentest-standard.org
- I don't know
- Other: ________________________________

**The security within the **** application**

In this section, we ask you to answer our questions from your own point of view, i.e. answer the following questions in regards to your own knowledge and experience.

18. What are the most critical security risks about ****? *

________________________
________________________
________________________
________________________

19. What are the most sensitive resources stored in the **** database? *

________________________
________________________
________________________
________________________
20. What security mechanisms have been put in place to minimize these risks? *

________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________

21. Do you know of any vulnerabilities in the **** application as of today? *
   Mark only one oval.
   ( ) Yes
   ( ) No

22. If you answered "Yes" on the question above, what are they?

________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________

23. **** has earlier had one of their customers perform a penetration test on one of their other applications. Do you know which application they tested, and what vulnerabilities were found? *

________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________

24. Before releasing a new update to the **** application do you, personally, perform any security related tests? *

________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
25. **What hinder you from performing more security focused work? * Mark only one oval.**

- [ ] Time
- [ ] Knowledge
- [ ] Management
- [ ] No restrictions exists
- [ ] Other: ________________________________

**Done**

Thank you for our participation, is there anything we've missed or anything you'd like to comment on. Please enter your questions or comments in the box below, or send us a mail at larba462@student.liu.se

26. **Other related questions or comments**

_________________________________________________________________

_________________________________________________________________

_________________________________________________________________

_________________________________________________________________

_________________________________________________________________
### Basic Questions

**Q1: Had you ever heard about OWASP before our survey?**
- □ Yes  □ No

**Notes...**

**Q2: Do you know what a risk assessment is?**
- □ Yes  □ No

**Q2 a): Have any kind of risk assessment has been done to the _____ application?**
- □ Yes, it has been done  □ Yes, it has not been done  □ No idea

**Notes...**

### Injection attacks

**Q3: Do you know what an injection attack is?**
- □ Yes  □ No

**Notes...**

**Q4: How does an injection attack work?**

**Notes...**

**Q5: How are injection attacks prevented in the _____ application?**

**Notes...**
<table>
<thead>
<tr>
<th>Authentication</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Q6:</strong> How are passwords stored in the _____ database?</td>
</tr>
<tr>
<td>Notes...</td>
</tr>
<tr>
<td><strong>Q7:</strong> What hashing algorithm is used for hashing the passwords?</td>
</tr>
<tr>
<td>Notes...</td>
</tr>
<tr>
<td><strong>Q8:</strong> Do you do regular checks for weak passwords in your database?</td>
</tr>
<tr>
<td>Notes...</td>
</tr>
<tr>
<td><strong>Q9:</strong> What technique do you use to remember your own passwords?</td>
</tr>
<tr>
<td>Notes...</td>
</tr>
</tbody>
</table>
### Sensitive Data

**Q10:** If _____ were to be compromised, what information would be leaked in a worst-case scenario?

Notes...

### Cross-Site Scripting

**Q12:** What's a Cross-Site Scripting attack?

Notes...

**Q13:** How do you defend against Cross-Site Scripting attacks in the _____ application?

Notes...

**Q14:** DO you allow weak passwords in your database?

Notes...
### Vulnerable External Components

**Q15:** What third-party libraries are used in the ________ application?

Notes...

**Q16:** Do you check for vulnerabilities in these libraries? If so how do you do this?

Notes...

**Q17:** How do you manage new releases of third party libraries? How often is this done?

Notes...

### Logging & Monitoring

**Q18:** If ________ is targeted by hackers. How would you notice?

Notes...
<table>
<thead>
<tr>
<th>PHP security</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q19: Take a look at the following piece of code. It’s taken from a former WordPress authentication bypass bug. It allowed unauthorized attackers to impersonate any user in the WordPress installation. What is the problem with the snippet of code? <a href="https://gist.github.com/zeNkan/9654f9cdd9946659f9d27a75917f628f">https://gist.github.com/zeNkan/9654f9cdd9946659f9d27a75917f628f</a></td>
</tr>
<tr>
<td>Notes...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Testing procedures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q20: How do you perform tests on code that’s going live?</td>
</tr>
<tr>
<td>Notes...</td>
</tr>
<tr>
<td>Q21: Do you have any security specific test cases?</td>
</tr>
<tr>
<td>Notes...</td>
</tr>
</tbody>
</table>