WEB APPLICATION SECURITY IN THE JAVA ENVIRONMENT

Kristoffer Wanderydz
Email:kw@wcom.se

June 6, 2012
Supervisor: Edgar Alonso Lopez-Rojas
Email: edgar.lopez@bth.se
Department: Information Technology - Security Karlskrona, Sweden
Copyright

Copyright protects the results of creative, intellectual work.

The Copyright Act (8 July 1961/404) provides copyright to a creator of a written or an artistic work (Copyright Act 1:1).

The Copyright Act provides protection for, for example, oral presentations, works of film and photographs, maps, drawings and computer programs.

Bachelor’s theses are public.

They can be protected by copyright only if they are sufficiently independent and original.

Bachelor’s theses do not automatically satisfy this requirement.

A work protected by copyright may be quoted and cited in review and to the extent required by the purpose.

Individuals may also produce a few copies for private use, but this does not, however, apply to computer programs (Copyright Act 2:12).

On the basis of the right of free presentation, the work may be used in connection with, for example, teaching (Copyright Act 2:14).

The author is the original holder of copyright.

Copyright is not restricted, for example, by the fact that an employee has created the work as part of his/her work duties, received payment for the work and used his/her employer’s equipment.

This does not, however, apply to computer programs (Copyright Act 3:40b).

Copyright may be transferred either wholly or in part (Copyright Act 3:27), but an agreement must be made in the case of such a transfer.

12 Other immaterial rights such as patents, (Patent Act 1967/550), protection of designs (Act on the Protection of Designs 1971/221), and trade marks (Trademark Act 1964/7) must also be taken into consideration.

©Kristoffer Wanderydz
Abstract

This project concerns some vulnerabilities in web applications, the most common exploits has been collected and implemented in a prototype. The prototype is developed with this project to produce results, suitable for the examples that is used in this project to address the exploits.

Each vulnerability collected in this project, was exploited and secured in the prototype. The vulnerabilities are presented in two shapes, one secure and one insecure. The prototype ran on a Tomcat web server, and was developed with frameworks such as Web, Spring and Hibernate. Connected to one PostgreSQL data source.

All vulnerabilities were successfully implemented in Spring framework, and they were all exploited. Every vulnerability was also secured, with different tools and methods from earlier mentioned frameworks. As a result, real examples from the prototype is used for demonstration in the project, both in a secure and an insecure state.

The result views Spring as a framework with good security potential. Most of the Spring specific vulnerabilities, are logical design flaws from developers that can be avoided. Vulnerabilities not related to Spring, such as the one collected for this project. Could be prevented by using methods from the Spring framework or intelligent programming.

Which leads to conclusions. Web applications are always exposed to attacks, no matter the framework in use. Creative hackers search to discover new vulnerabilities, and update old ones all the time. Developers has a responsibility, towards the web applications users. Web applications can not just developed for normal use, but also against possible misuse. Frameworks with good reputation and well processed models, is a good ground for developing a secure application.
Acknowledgements

I would like to thank the supervisor of this project Edgar Alonso Lopez-Rojas at BTH for guidance in concept of this project and chiefly the report, and all the others who had related material published online in this area that made a contribution to this project in setting up the environment or provided facts.
List of Figures

1.1 Progress method ............................................. 3
2.1 Misuse case - SQL injection:Authentication .................. 6
2.2 Exploit - SQL Injection:Authentication ........................ 6
2.3 Exploit - SQL injection:Bypassing authorization .............. 8
2.4 Sequence - Cross-site Scripting:Reflected .................... 8
2.5 Misuse case - Cross-site Scripting:Reflected .................. 9
2.6 Exploit - Cross-site Scripting:Reflected ..................... 9
2.7 Exploit - Cross-site Scripting:Reflected:URL .................. 9
2.8 Exploit - Cross-site Scripting:Reflected:Generated .......... 10
2.9 Exploit - Cross-site Scripting:Reflected:Message .......... 10
2.10 Sequence - Cross-site Scripting:Stored ..................... 10
2.11 Misuse case - Cross-site Scripting:Stored ................... 11
2.12 Identify vulnerability - Cross-site Scripting:Stored .......... 11
2.13 Exploit vulnerability - Cross-site Scripting:Stored .......... 12
2.14 Exploit vulnerability - Cross-site Scripting:Stored .......... 12
2.15 Exploit vulnerability - Cross-site Scripting:Stored .......... 12
2.16 Exploit vulnerability - Cross-site Scripting:Stored .......... 12
2.17 Misuse case - Request Forgery ............................. 13
2.18 Exploit - Request Forgery .................................... 13
2.19 Exploit - Request Forgery .................................... 14
2.20 Exploit - Request Forgery .................................... 14
2.21 Exploit - Request Forgery .................................... 14
2.22 Exploit - Request Forgery .................................... 15
2.23 Exploit - Request Forgery .................................... 15
2.24 Sequence - Authentication management ....................... 16
2.25 Misuse case - Authentication management .................... 16
2.26 Exploit - Authentication management ........................ 17
2.27 Exploit - Authentication management ........................ 17
2.28 Exploit - Authentication management ........................ 17
3.1 Spring MVC:Model View Controller .......................... 18
3.2 Secure - Reflected XSS .................................... 19
3.3 Secure - Reflected XSS - Redirected URL ..................... 19
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4</td>
<td>Secure - Reflected XSS - Nothing generates</td>
<td>20</td>
</tr>
<tr>
<td>3.5</td>
<td>Secure - Reflected XSS - The script not generated</td>
<td>20</td>
</tr>
<tr>
<td>3.6</td>
<td>Secure - Reflected XSS for all browsers</td>
<td>20</td>
</tr>
<tr>
<td>3.7</td>
<td>Secure - Reflected XSS for all browsers</td>
<td>21</td>
</tr>
<tr>
<td>3.8</td>
<td>Secure - SQL Injection: Development model</td>
<td>21</td>
</tr>
<tr>
<td>3.9</td>
<td>Secure - SQL Injection: Authorization</td>
<td>22</td>
</tr>
<tr>
<td>3.10</td>
<td>Secure - SQL Injection: Authorization</td>
<td>22</td>
</tr>
<tr>
<td>3.11</td>
<td>Secure - SQL Injection - Encrypted Login 1</td>
<td>23</td>
</tr>
<tr>
<td>3.12</td>
<td>Secure - SQL Injection - Encrypted Login 2</td>
<td>23</td>
</tr>
</tbody>
</table>
Listings

2.1 Detailed Error Message Response ................................ 6
2.2 insecure Authentication Query .................................... 7
2.3 Exploited Query ...................................................... 7
3.1 Characters of Importance to Escape ............................... 20
3.2 Insecure Message Output .......................................... 20
3.3 Secure Message Output ............................................. 20
3.4 Secure Authentication Query ...................................... 21
3.5 Safe Data Transmission ............................................. 23
A.1 article.jsp ............................................................ 35
A.2 articleadd.jsp ....................................................... 36
A.3 blog.jsp .............................................................. 36
A.4 comments.jsp ....................................................... 36
A.5 contact.jsp .......................................................... 37
A.6 login.jsp ............................................................. 37
A.7 loginSafe.jsp ....................................................... 38
A.8 news.jsp ............................................................. 38
A.9 computero.css ....................................................... 39
A.10 navigation.jsp ...................................................... 40
A.11 footer.jsp .......................................................... 41
A.12 header.jsp .......................................................... 41
A.13 taglibs.jsp .......................................................... 41
A.14 home.jsp ............................................................ 41
A.15 index.jsp ............................................................ 42
A.16 web.xml .............................................................. 42
A.17 applicationContext.xml ............................................ 42
A.18 dispatcher-servlet.xml ............................................. 43
A.19 hibernate-context.xml ............................................. 43
A.20 hibernate.cfg.xml .................................................. 44
A.21 logging.properties ................................................. 45
A.22 spring.properties .................................................. 45
A.23 AdministratorService.java ....................................... 45
A.24 ArticleService.java ............................................... 48
A.25 CommentService.java ............................................ 50
A.26 Administrator.java ................................................. 52
A.27 Article.java ........................................................ 53
A.28 Comment.java .................................................. 55
A.29 SuperController.java ................................. 56
Chapter 1

Introduction

The security in web applications is an important issue to attract users, still large number of applications gets exploited. The developers have the responsibility for satisfying security, but they do not know about the vulnerabilities or lack security awareness, because it seems to be a recurring problem. A trustworthy application is up-to-date and does not contain known vulnerabilities.

This project focuses on the secure web development in the Java environment and the Spring framework. This project is based on security in the computer science area, where vulnerabilities and exploits in a web applications are the main focus. An interesting point of view would be to analyze how vulnerable Java is, and what is the potential Java possess to secure various vulnerabilities.

Correlative to the collection of vulnerabilities, a prototype is built in this project, and take two shapes. One shape is the secure state, and the other one is insecure. figure1.1 depicts a visual explanation. The insecure state is addressing vulnerabilities and presents how they get exploited, the secure state demonstrates how to secure the vulnerabilities, in other words how to prevent these exploits.

In many reports, journals and books similar aspects of vulnerabilities have been analyzed, introducing exploits and fixes in web applications. Most of the work addresses vulnerabilities in hardware, software and development proposals such as mentioned by[14, 7, 12, 15, 9, 13] discussed in section4.1

1.1 Background

Web development has an old history in computer science, but history has always repeated itself when it comes to state of security. The hacking exposed series 7 - 12, indicates the need of security trough time.

In the beginning web pages were static and the purpose of a web page was only to convey information to visiting clients. Vulnerabilities were rarely discovered in the application itself, rather than the environment the application ran on. As the years passed by the requirements on web applications grew and
matured with response and requests methods such as AJAX\[2\], extended protocols, scripting, frameworks and development kits. Web applications became more modern and dynamical than before\[8, 1. Introduction\].

New types of user interaction emerged, sites with possibilities to shop(Ebay), chat(Facebook), search(Google), gamble(Party Poker) and do bank errands(SEB). They became a more flexible alternative for people. Interaction with dynamical web applications is done through a web browser described in\[4\].

User based web applications seem to benefit with a good reputation in security, they could be appealing for users who must store confidential data.

1.2 Purpose and Objective

The purpose of this project is to test the prerequisites of a web application developed in Java environment with focus on the Spring framework against the most exploited vulnerabilities affecting web applications today.

A web application prototype was developed according to the model\[4,1\]. Demonstrating what the Java framework Spring provide to prevent common vulnerabilities, and which vulnerabilities that can be exploited in a web application built with Spring, in the Java environment. The purpose of the prototype is to serve on educational basis, and feature as a proof-of-concept.

The result section\[4\] provide an answer to, if a web application developed in Spring. Can be secure enough, to stand against the most exploited vulnerabilities.

1.2.1 Research Questions

Is Spring framework in the Java environment vulnerable to the most common vulnerabilities? Can the Java environment secure the vulnerabilities?

1.2.2 Contribution

The first contribution of this project is secure development in Spring framework, presenting security in vulnerabilities and exploits.

The second contribution is the web application prototype. The purpose is to pedagogically show the vulnerabilities and how to exploit them, but also how to prevent them. The web application prototype is meant to serve for educational purpose such as enlighten target groups.

This project covers the most common vulnerabilities in web applications, according to Trustwave\[14\] among others. The project demonstrates exploits and how to cover them up, with real examples from the prototype.

1.3 Method

This project uses agile, experimental and empirical research methods to answer the research questions.
1.4. TARGET GROUP

The most effective vulnerabilities are implemented in the prototype, followed by an attempt to secure the vulnerabilities. The result is documented and builds the report, with solutions in how to secure the application. Empirical but also science based conclusions can be drawn from the result.

Figure 1.1 explains a typical scenario of the intended work process. The red arrows creates an agile iteration, which later on is implemented in an experimental method to generate a result.

1.4 Target group

This project content may be of interest to web application developers and penetration testers, it may also be of interest to teachers and students for educational purposes.

Developers can use this project to avoid pitfalls, meanwhile penetration testers can use the project to find pitfalls.

Teachers can use the project for educational purposes, where students will gain knowledge about vulnerabilities and how they get exploited. Students can also learn how to configure and set up a java web application.
1.5 Report structure

This project is partitioned into different following chapters, sections and subsections to ease the reading. An overview is found in the table of content.

Chapter 1 tells what the project covers and which answers you will get by reading the rest of the chapters.

Chapter 2 presents the collected vulnerabilities and how they are exploited. With models and examples from the prototype.

Chapter 3 demonstrates how to secure the vulnerabilities from chapter 2.

Chapter 4 presents the result from 3 and reflects over the result in a discussion.

In chapter 5 conclusions are drawn, based on the discussion. The projects whereabouts is covered.

After these chapters, support sections follow such as acronyms, terminology, references and appendix.
Chapter 2

Theoretical Background

This chapter shows how to exploit the vulnerabilities, implemented in the prototype. It is of importance to know how the exploits take place, to better understand the solutions in next chapter, why and how they can occur. In the end of this chapter are specific vulnerabilities for Spring listed.

2.1 Vulnerabilities

This section covers all collected vulnerabilities, the most common vulnerabilities according to the reports. Hundreds of applications were successfully penetrated, a classification was made by the success rate for an exploit. That is how a vulnerability makes the cut.

The vulnerabilities are also implemented in the prototype, demonstrating the exploits. The prototype is viewed from an insecure perspective.

2.1.1 SQL Injection

Injection is an attack against a web application’s data storage, by modifying the query that the application sends to the data source to perform certain actions such as authentication, fetch articles or add articles. The attacker can exploit the data communication between the data source and the web application.

It is more to be found on attacks against data storage in articles and books.

SQL injection: Authentication

To identify and eventually perform a successful injection attack on the authentication data storage, the attack normally have to occur in some kind of login form.

In listing the sequence flow is presented, for authentication on data storage. Viewed from an attacker’s perspective.
2.1. VULNERABILITIES CHAPTER 2. THEORETICAL BACKGROUND

1. Identify the login form
2. Send unexpected data
3. Take notes from the behavior and the response from the server
4. Try to make use of the information from the response
5. Take action

Figure 2.1 demonstrates the attackers misuse model and the normal intended use model.

Figure 2.1: Misuse case - SQL injection: Authentication

Figure 2.2 presents a login attempt with a blank password. The web application gives the response that the password is wrong, so the attacker can assume that the username is correct.

```
admin
```

Figure 2.2: Exploit - SQL Injection: Authentication

In Listing 2.1 the login controller is displayed.

```
@RequestMapping (value = "/login", method = RequestMethod.POST)
public ModelAndView login (@ModelAttribute("loginAttribute") Administrator admin, HttpServletRequest request) {
    Boolean auth = administratorService.authenticate(admin);
    if (auth) {
```
2.1. VULNERABILITIES CHAPTER 2. THEORETICAL BACKGROUND

```java
WebUtils.getSessionAttribute(request, "user", admin.getAdmin());
return new ModelAndView("home", "Message", "Authorized");
else
return new ModelAndView("home", "messageFail", "Forgot your password?");
```

Listing 2.1: Detailed Error Message Response

The attacker knows that the user he tested is legit, because the error message only addressed the password. The attacker performs a SQL injection on the user attribute, with the query "admin'--" which comments out the rest of the query. As shown in listing 2.2 (Line 6,7,8).

```java
public Boolean authenticate(Administrator admin){
    Boolean auth;
    Session session = sessionFactory.getCurrentSession();
    Query query = session.createQuery("select admin, password from administrator where admin='" + admin.getAdmin() + "' and password='" + admin.getPassword() + "' ");
    if (query.list().size()==1)
        auth = true;
    else
        auth=false;
    return auth;
}
```

Listing 2.2: Insecure Authentication Query

The administrator object passes from the login-form to this authentication process, where the primary key "admin" compares to the administrators in the table "administrator" which holds all administrators.

If the administrator is found the process compares the password sent from the login-form, and if it is a double-match the query returns a list with one administrator and the authorization process returns valid login.

```java
Query query = session.createQuery("select admin, password from administrator where admin='" + admin.getAdmin() + "' and password='" + admin.getPassword() + "' ");
```

Listing 2.3: Exploited Query

If the password now is a comment, the list will always return the value one as long as the username is correct.

If the name sent from the login-form exists, the crafted query will return a list of one administrator and the authentication will get bypassed.
2.1. VULNERABILITIES

CHAPTER 2. THEORETICAL BACKGROUND

Figure 2.3: Exploit - SQL injection: Bypassing authorization

2.1.2 Cross-Site Scripting - XSS

Cross-site Scripting is the most common web application attack, the attack exists in several different states and is further explained in articles such as [15].

Cross-site Scripting: Reflected

Reflected XSS is what you call an attack when an attacker makes a crafted request, and feed a victim with the request. The request exploits a reflection vulnerability in a web application. Reflected Cross-site scripting is described further in literature such as [14, 15, 13].

Figure 2.4 is an example of a normal sequence flow of an reflected attack.

![Figure 2.4: Sequence - Cross-site Scripting: Reflected](image)

1. User is surfing the web
2. Attacker provide the user with a crafted url
3. The user browser requests the crafted url
4. The malicious javascript gets executed
5. The javascript sends the response from the request to the attacker

Figure 2.4: Sequence - Cross-site Scripting: Reflected

Figure 2.5 is an example of an attacker misusing the intended normal use case. The exploits from the developed prototype, follows in upcoming Figures starting with Fig 2.6.

The user surfs in to a blog of interest as in Fig 2.6. The blog site is actually made by an attacker, who presumes that if the content in the blog is appreciated. The visitor probably have an account at the site, where the attacker has found a vulnerability.

The crafted request is visible in Figure 2.7 and generates the alert 2.8 the attacker is running.

remember that this does not have to be done so visible, as in Fig 2.8. The same process 2.9 can occur in the background without the users knowledge.
2.1. VULNERABILITIES CHAPTER 2. THEORETICAL BACKGROUND

Cross-site Scripting:Reflected

Figure 2.5: Misuse case - Cross-site Scripting:Reflected

Figure 2.6: Exploit - Cross-site Scripting:Reflected

Figure 2.7: Exploit - Cross-site Scripting:Reflected:URL

Cross-site Scripting:Stored

Stored XSS is when an attacker, exploits the possibility for users of storing data. The attacker can for example store a script in a guest book.

The attacker posts a comment into a guestbook, which contain an invisible script. Every user that takes a look at the same guest book, will now get this script generated in their browser. Other examples can be found in [14, 15, 13]. Fig 2.10 presents the sequence flow, and Fig 2.11 demonstrates the misuse...
2.1. VULNERABILITIES CHAPTER 2. THEORETICAL BACKGROUND

Figure 2.8: Exploit - Cross-site Scripting:Reflected:Generated

Figure 2.9: Exploit - Cross-site Scripting:Reflected:Message

case of a stored Cross-site scripting attack.

Figure 2.10: Sequence - Cross-site Scripting:Stored

The attack starts with a posted script in the guest book, shown in Fig 2.13. The content of the script are invisible, so the user is unaware of that the script, actually is executed and running as viewed in Fig 2.16.

An alert is generated for demonstration, and generates for every user who views the guest book. Shown in Fig 2.14.

Firebug shows the hidden and auto generated javascript, shown in Fig 2.15. this example is a proof-of-concept and generates a warning. The script could have been doing a buy or retrieve credentials from the user "user" account, without the users knowledge.

2.1.3 Request forgery

Also known as session riding, the attacker exploits a user. Not by knowing the token, but by manufacturing and analyzing the web applications normal
2.1. VULNERABILITIES CHAPTER 2. THEORETICAL BACKGROUND

Figure 2.11: Misuse case - Cross-site Scripting:Stored

Figure 2.12: Identify vulnerability - Cross-site Scripting:Stored

behavior. Request forgery has a type called on-site and further information can be found in the articles [14, 15] and the book [13, page 502-510].

On-site
If a web application is secured, and escapes possible inputs. It is still possible for an attacker to manufacture this kind of attacks.

It is still a possibility to make damage for an attacker, by adding items for other users to view that is hard to html escape. For example adding an image.

1. Investigate the web application for user input data storage
2.1. VULNERABILITIES CHAPTER 2. THEORETICAL BACKGROUND

Figure 2.13: Exploit vulnerability - Cross-site Scripting:Stored

Figure 2.14: Exploit vulnerability - Cross-site Scripting:Stored

Figure 2.15: Exploit vulnerability - Cross-site Scripting:Stored

Figure 2.16: Exploit vulnerability - Cross-site Scripting:Stored
2.1. VULNERABILITIES CHAPTER 2. THEORETICAL BACKGROUND

2. Identify if the user input is active content for other users using the web application

3. The vulnerability is often exploited when the data is inserted to a hyperlink or other URL within the page

4. If the web application is vulnerable look for a suitable request to target in the exploit

![Diagram](image)

Figure 2.17: Misuse case - Request Forgery

The attacker identify an article of interest, and notice that the form contains the variables Category, Name and Price. As shown in Fig. 2.18.

![Table](image)

Figure 2.18: Exploit - Request Forgery
2.1. VULNERABILITIES  CHAPTER 2. THEORETICAL BACKGROUND

An attacker could then forge a request, by analyzing the web applications behavior. Then wait for an administrator to request the page, where the administrator has stored the script, as shown in Fig.2.19. An administrator normally has higher privilege than normal users, and can therefore take actions not normal users can.

![Figure 2.19: Exploit - Request Forgery](image1)

The attacker wrote a script based on the analysis of the web application, the script is shown in Fig.2.20.

![Figure 2.20: Exploit - Request Forgery](image2)

If an administrator takes a look at the guest book, the forged request Will be posted as shown in the url. In this case an article gets updated, as shown in Fig.2.21.

![Figure 2.21: Exploit - Request Forgery](image3)

Fig.2.22 presents the parameters the attacker made the administrator post, when visiting the guest book.

The attacker can now buy the segway for 3 units shown in Fig.2.23.

### 2.1.4 Authentication and Session Management

Successful attacks on authentication are more neatly described in the book[13](ch6). Attacks on the authentication are breaches in the login process, meanwhile
attacks against session management [13] (ch7), indicate unsafe management of stored credentials and how the communication of data between the layers takes place.

Sniffing in Wireless net

The attacker sniffs the Wireless net to pick up the users credentials, described in more detailed manners [13] page 50, 18, 159-161).

Figure 2.24 show a sequence flow, and fig 2.25 present a misuse case how sniffing can occur.

Fig 2.26 demonstrates an admin signing in, and Fig 2.27 show the credentials submitted unencrypted, when signing in. The attacker can collect the sent data, when someone signs in. This is possible through a sniff attack.

After the attacker has collected the sign in data, it is possible for the attacker to sign in illegal.

In Fig 2.28 number of login attempts is used as a salt. The attacker can still collect the salted credentials, and login.

One way to solve a sniffing attack, is to make the sign in process polymorphic. As proposed in 3.1.3.

2.2 Spring Specific Vulnerabilities

Vulnerabilities in Spring are listed. There is more to find out about vulnerabilities in the article [8]. Each listed vulnerability contains a referenced link with more information.

1. Spring MVC: ModelView Injection [9]
2.2. SPRING SPECIFIC VULNERABILITIES

CHAPTER 2. THEORETICAL BACKGROUND

Figure 2.24: Sequence - Authentication management

Figure 2.25: Misuse case - Authentication management
2. Spring MVC: Data Submission to Non-Editable Fields [9]
5. Spring Framework: Information disclosure [10]
Chapter 3

Implementation

Following sections in this chapter propose solutions to secure the vulnerabilities from chapter 2 and a short description of the prototype. It is important to understand Chapter 2 to make use of this chapter.

The prototype were built for demonstration in this project, and developed according to the logical design in Spring. Explained in Fig 3.1. The prototype was built to test the security in Spring and provide real cases.

The prototype is available externally but also listed in this project. The infrastructure is visible and a short description is given in the appendix for each section of the prototype.

![Spring MVC:Model View Controller](image)

Figure 3.1: Spring MVC:Model View Controller

Figure 3.1 explains the work flow in the prototype very well.
3.1 Proposed solution

Following section contain real examples from the prototype 5.1, securing the vulnerabilities that got exploited from section 2.

3.1.1 Cross-site scripting (XSS)

Solution 1

Some web browsers such as IE and Google Chrome, contain Cross-site scripting (XSS) filters. One solution could be to forbid browsers that does not provide the needed filter.

![my blog where you find awesome things, click here to read more ...](http://localhost:8080/blog.html)

Figure 3.2: Secure - Reflected XSS

In Fig 3.2 the web browser IE with the XSS filter gets attacked, with one reflected Cross-site scripting attack as shown in section 2.1.2.

![my blog where you find awesome things, click here to read more ...](http://localhost:8080/contact.html?message=%3Cscript%3E%20alert%28%27SYC%20%27%29%3C/script%3E%20--%20CompT %3E)

Figure 3.3: Secure - Reflected XSS - Redirected URL

The filter is active by default. The script shown in Fig 3.2 will be requested, but wont get generated as shown in Fig 3.3. Fig 3.4 shows no alert window popping up and the vulnerability is secured.

Solution 2

Browsers such as Firefox, are still vulnerable as we saw in Fig 2.8. The vulnerability was demonstrated in this section 2.1.

Browsers that lack the Cross-site scripting filter, must escape the html tags manually. Such as those in Fig 3.1.
3.1. PROPOSED SOLUTION

CHAPTER 3. IMPLEMENTATION

Figure 3.4: Secure - Reflected XSS - Nothing generates

```html
<div id="main">
<script>alert("YES");</script>
</div>
```

Figure 3.5: Secure - Reflected XSS - The script not generated

 Listing 3.1: Characters of Importance to Escape

Solution 3

The best solution, is probably to manipulate the way the output from the application interprets by the browser.

In this Fig3.2 there is no action taken, therefore browsers without Cross-site scripting-filter is vulnerable.

```java
$message
```

Listing 3.2: Insecure Message Output

In this Fig3.3 action is taken. The request shown in Fig3.6 will make the browser escape correctly, shown in Fig3.7.

```java
<c:out value="$\{message\}"/>
```

Listing 3.3: Secure Message Output

Figure 3.6: Secure - Reflected XSS for all browsers
3.1. PROPOSED SOLUTION

CHAPTER 3. IMPLEMENTATION

3.1.2 Injection

The data storage vulnerability that took place in subsection 2.1.1 is here presented a proposed solution.

SQL Injection

Secure the query exploited earlier, shown in Fig 2.2. The administrator username and password are compared in the same query, it is more secure to break down the query to multiple processes. Therefore also more layers of protection.

```java
public Boolean authenticateSafe(Administrator admin) {
    Boolean auth;
    Session session = sessionFactory.getCurrentSession();
    //One query to fetch the administrator
    Query queryName = session.createQuery("select admin from administrator where admin='"+admin.getAdmin()+"'");
    //One query to fetch the password
    Query queryPassword = session.createQuery("select password from administrator where admin='"+queryName.list().get(0)+"'");
    //One authentication process strictly related to the
    //administrator object, now the list size doesn't matter.
    if (queryPassword.list().get(0).equals(admin.getPassword()))
```
3.1. PROPOSED SOLUTION

CHAPTER 3. IMPLEMENTATION

auth = true;
else auth=false;
return auth;

Listing 3.4: Secure Authentication Query

Do not have, independent authentication methods as in Fig2.2.
Instead more relevant comparisons should be used, that makes a direct dependency to the object shown in Fig3.4.

It is important to escape characters that can make damage, such as this comment example ”–“ which succeeded in Fig2.3.

Figure 3.9: Secure - SQL Injection:Authorization

Figure 3.10: Secure - SQL Injection:Authorization

The prototype is now secured against injection, bypassing the authentication process shown in Fig3.9. The parameters submitted will no longer get authorized as shown in Fig3.10.

3.1.3 Authentication and Session Management

The earlier example of an attack against Authentication and Session Management shown in 2.1.4 seemed hard to prevent when the attacker just repeated the sniffed request and gained access.

Sniffing in Wireless net

A solution to this vulnerability is to make the web applications behavior change, shown in enumeration3.1.3 Encrypt the posted password and the sniffer can
not tell what hides, as in Fig3.5 the password is encrypted with a MD5 hash.

```html
<form name="myform" action="http://localhost:8080/login.html" method="POST">
  <input type="text" id="admin" name="admin" value="username..." onclick="this.value=''">
  <input type="text" id="password" name="password" value="password..." onclick="this.value=''">
  <input type="button" value="Sign in" onclick="encrypt(password.value, admin.value);">
</form>
<script type="text/javascript">
  function encrypt(pass, user) {
    encryptedPass = calcMD5(pass);
    window.location = "http://localhost:8080/login.html?admin=" + user + '&password=' + encryptedPass;
  }
</script>
```

Listing 3.5: Safe Data Transmission

In Fig3.11 and in Fig3.12 an encrypted login scenario from the administrator is demonstrated.

![Figure 3.11: Secure - SQL Injection - Encrypted Login 1](image1)

![Figure 3.12: Secure - SQL Injection - Encrypted Login 2](image2)

Fig3.12 shows the password "admin" in encrypted form, posted in Fig3.11. It is still possible for the attacker to resend the credentials, a proposed solution
3.1. PROPOSED SOLUTION  

follows.

1. Let's say is the thousandth time the user login, the entered password "admin" will be appended to equal "admin1000" and sent encrypted "e25599270162 76b8f01e7b13de6d5c41" to the controller.

2. The controller decrypts the password and the salt 1000 will be updated to 1001.

3. So if the attacker reads the key "e2559927016276b8f01e7b13de6d5c41" and signs in with the encrypted key, the decrypted salt 1000 will not equal the new required salt 1001.
Chapter 4

Analysis of the Results

All the collected vulnerabilities were implemented, then exploited in the developed prototype. The vulnerabilities were proposed with an experimental solution. The prototype grew with each vulnerability, and generated science based examples.

Spring provides tools and methods for secure web development in Java, so Spring is a secure alternative for web application development.

The vulnerabilities were implemented in the prototype, and successfully exploited. The prototype show that vulnerabilities can occur in Spring, and that security actions needs to be taken.

All the vulnerabilities in the prototype were successfully secured, Spring provided methods and tools for a secure web development. Spring shows to be a secure framework with strong security potential, if used with precaution and consideration.

Spring show good security and leaves the responsibility to the developer. Spring provided different solutions to counter the exploits, and each exploit were proposed with a solution. The prototype developed in Spring was in the end secure, with other words not vulnerable to the most common vulnerabilities.

Web applications developed in Java, has the potential to be secure against famous exploits.

4.1 Related Work

The work of Konecki et al. on web application security called "Secure web applications?", is related because vulnerabilities are identified.[5]

The book "The Web Application Hackers Handbook" from Markus Pinto and Dafydd Stuttard is related because they identify the most exploited vulnerabilities of today and explain them [13].

The article "A Guide to Building Secure Web Applications" by OWASP is related from the perspective that they have focused on providing developers with knowledge of security that may be missed when developing a web application.[3]
These two reports are the result of two teams penetration testing many web applications, they list the most exploited vulnerabilities and a description of the vulnerability. This project did also contribute with The development of the prototype made this project unique, with real examples for exploits, and secured vulnerabilities in Spring. This project merges the statistics from the penetration reports with how to develop a secure web application. The project is a summary of important milestones in websecurity, and therefore suitable for introduction to security in web applications and for educational purposes.

This project was partly made to conclude if Spring was a secure alternative for web development. Also to contribute with solutions on securing the most common vulnerabilities.

4.2 Discussion

Previous chapters show that a web application developed in Spring, can be secured against the most exploited vulnerabilities. The exploited web applications from the test teams, mentioned in Related Work, can be developed by developers unaware of the most common exploited vulnerabilities. The developers of the exploited applications may not know how easy it is, to encounter the vulnerabilities in a web application. The vulnerabilities in this project are open and common known knowledge.

The development of the prototype, made the project more unique than the other projects in related work. Developing in Spring was a challenge itself, also developing a secure web application came to be time consuming. Lucky Spring is a famous framework and well documented.

As the result points to, Spring has the potential to be a secure alternative for developing web applications.

The Spring framework and the Spring MVC model, introduce a safe development process. One of the important things of developing a secure web application, is for the developer to be aware of the normal use case and the misuse case. The developer must know which possibilities, the functionality in the web application leaves to an attacker. It is of importance to prevent unforeseen consequences.

You can still not yet declare Spring totally safe, there is certain many vulnerabilities not known. The hammer must fail on the developer, it seems to be of importance to test potential misuse cases.

The result of the project is from an objective point-of-view. The project demonstrates many proof-of-concepts, showing the security potential in the Spring framework. The prototype address collected vulnerabilities, exploits and fixes.

The result declares already discovered vulnerabilities, and the security level of the prototype. The prototypes security level is based on how well the vulnerabilities was secured.
4.3 Limitations

SSL is not discussed, because it is out of scope in Java development. SSL is a layer to layer protection, implemented in the web server-side of a running application.

Several Platforms and frameworks in the prototype are not implemented, because it takes too much time. The vulnerabilities concern all platforms and frameworks, but in development this project will use Java with frameworks Web, Spring and Hibernate.

Vulnerabilities that compromise the web application, but is not a part of the development stage. For example vulnerabilities in the latest version of the web server.
Chapter 5

Conclusions

Web applications are always exposed to attacks, no matter what framework is in use. With many possible vulnerabilities found and exploited by creative attackers, a web application has a responsibility towards its users whom stores sensitive data. The developers can not just develop the web application for normal use, but also against possible misuse.

If the most common vulnerabilities is secured in a web application. The application has a tolerant security classification and good basis for distribution.

There are many ways of defense for the prototype. The lack of knowledge of the existing vulnerabilities with the developer, seems to be the greatest risk against security.

Frameworks with good reputation and well processed models are a good ground for developing a secure application.

A collection of the top listed exploited vulnerabilities was presented, explained and demonstrated with live examples in two states. Both when the vulnerability got exploited and when the vulnerability got secured. The prototype managed to address all vulnerabilities, in both a secure and insecure state with live examples.

The vulnerabilities can exist in both Java and Spring, they could also get exploited as expected. The Java environment provided methods to secure the vulnerabilities. If a developer is unaware of the vulnerabilities existence, it is more likely to develop an application which contain the vulnerabilities. On the other hand if the developer knows about the vulnerability and how they get exploited, the Spring framework helps, by providing the tools for securing such vulnerabilities.

5.1 Future work

As new frameworks and updates to framework gets released. It is good to enlighten web developers about new vulnerabilities, exploits and protection based on other frameworks with focus on the security in web applications. The de-
5.1. FUTURE WORK

The developer can pick the right environment, and take the right precautions. To customize a security level that matches the application's purpose, and know which guarantees can be given to the users.

Other potential future work could be penetration testing, and find new vulnerabilities based on the techniques mentioned in this report. Read about the vulnerabilities and find new exploits.

This project can also be used as a ground, to find other solutions to the mentioned vulnerabilities. There would be value in comparing the solutions in this project, against potential new ones and point out pros and cons.
Bibliography

[1] The requesting processing workflow in Spring Web MVC (high level). 18


[6] Jennifer KRISHNAMURTHY, Balachander REXFORD. *Web protocols and practice: HTTP/1.1, networking protocols, caching and traffic measurement*. 2001. 2


[8] Spring. All SpringSource security vulnerabilities, 2009. 15


30
Glossary


3. Hibernate - Framework used to create objects in the web application, from the data source. [http://www.hibernate.org/](http://www.hibernate.org/)

Acronyms

BTH = Blekinge Tekniska Högskola [http://www.bth.se/]
API = Application Programming Interface
SSL = Secure Sockets Layer
MVC = Model View Controller
HTML = Hypertext Markup Language
IE = Internet Explorer
XSS = Cross-site Scripting
Appendices
The appendix contains all the code, the web application uses in this project. As a proof-of-concept and for demonstration. There is a more detailed description with each section in the appendix.
Appendix A

Java source code

A.1 Web

Functional jsps

Functional jsps are the response views, that models from the request according to the Fig.1. They contain the HTML code, and is a part of the web applications dynamical behavior.

```xml
<%@include file="../../../taglibs.jsp" %>
<c:set var="filter" value="article"/>
<%@include file="../../../header.jsp" %>
<%@include file="../../../navigation.jsp" %>
<c:if test="${not empty articles}">
  <table width="320">
    <thead style="text-align: left; border-spacing:15px">
      <tr>
        <th>Category</th>
        <th>Name</th>
        <th>Price</th>
      </tr>
    </thead>
    <c:forEach items="${articles}" var="article">
      <tbody>
        <tr>
          <td>${article.category}</td>
          <td>${article.name}</td>
          <td>${article.price}</td>
        </tr>
      </tbody>
    </c:forEach>
  </table>
</c:if>
<c:if test="${empty articles}">
  There are currently no articles in the list.
</c:if>
```
A.1. WEB APPENDIX A. JAVA SOURCE CODE

Listing A.1: article.jsp

```jsp
%@include file="../../../footer.jsp"
<c:set var="filter" value="article"/>
%@include file="../../../header.jsp"
<form:form action="http://localhost:8080/addarticle.html" modelAttribute="articleAttribute" method="POST">
    <table>
        <tr>
            <td><form:label path="category">Category</form:label></td>
            <td><form:input path="category"/></td>
        </tr>
        <tr>
            <td><form:label path="name">Name</form:label></td>
            <td><form:input path="name"/></td>
        </tr>
        <tr>
            <td><form:label path="price">Price</form:label></td>
            <td><form:input path="price"/></td>
        </tr>
    </table>
    <input type="submit" value="Save"/>
</form:form>
%@include file="../../../footer.jsp"
```

Listing A.2: articleadd.jsp

```jsp
my blog where you find awesome things, click <a href="http://localhost:8080/contact.html?message=<script>alert('Now am i fetching your session data at this site where you rock and posting it to me so i can impersonate you')</script>">
here</a> to read more ...
```

Listing A.3: blog.jsp

```jsp
%@include file="../../../taglibs.jsp"
<c:set var="filter" value="comments"/>
%@include file="../../../header.jsp"
%@include file="../../../navigation.jsp"
<c:if test="${empty comments}">
    There are currently no comments posted.
</c:if>
<c:forEach items="${comments}" var="comment">
    <table style="margin-top:30px">
        <tr>
```

36
### Listing A.4: comments.jsp

```jsp
<%@include file="../../../../footer.jsp" %>

<form:form modelAttribute="commentAttribute" method="POST" action="http://localhost:8080/comments.html">
  <table style="margin-top:30px;">
    <tr>
      <td>${comment.sender}</td>
    </tr>
    <tr>${comment.message}</tr>
  </table>
</form:form>

<%@include file="../../../../taglibsp.jsp" %>
<%@include file="../../../../header.jsp" %>
<%@include file="../../../../navigation.jsp" %>
<c:out value="${message}"/>
<%@include file="../../../../footer.jsp" %>
```

### Listing A.5: contact.jsp

```jsp
<%@include file="../../../../footer.jsp" %>

<div style="margin-left:320px;">

```
```

<%--
Created by IntelliJ IDEA.
User: kristofferwandumdz
Date: 2012–maj–17
Time: 14:47:21
To change this template use File | Settings | File Templates. --%>
<%--
<%@include file="../../../../taglibsp.jsp" %>
<c:set var="filter" value="contact"/>
<%@include file="../../../../header.jsp" %>
<%@include file="../../../../navigation.jsp" %>
<c:out value="${message}"/>
<%@include file="../../../../footer.jsp" %>
```
<form action="http://localhost:8080/login.html" method="POST" modelAttribute="loginAttribute">
  <form:label path="admin"/>
  <form:input path="admin"/>
  <form:label path="password"/>
  <form:password path="password"/>
  <input type="submit" value="Sign in"/>
  <span style="color:#cd5c5c; font-size:small;">${messageFail}</span>
  <span style="color:#228b22; font-size:small;">${Message}</span>
  <span>Signed in as: <%=session.getAttribute("user")%></span>
</form>

Listing A.6: login.jsp

<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

<div style="margin-left:320px;">
  <form name="myform" action="http://localhost:8080/login.html" method="POST">
    <input type="text" id="admin" name="admin" value="username..." onclick="this.value=''">
    <input type="text" id="password" name="password" value="password..." onclick="this.value=''">
    <input type="button" value="Sign in" onclick="encrypt(password.value, admin.value)">
  </form>
</div>

<script type="text/javascript">
  function encrypt(pass, user) {
    encryptedPass = calcMD5(pass);
    window.location = "http://localhost:8080/login.html?admin=" + user + "&password=" + encryptedPass;
  }
</script>

Listing A.7: loginSafe.jsp

<%>
  Created by IntelliJ IDEA.
<%>

38
Non-functional jsps

Non-functional jsps are the same as a functional jsp[A.1] but they are often a part of a functional jsp, to create an object oriented logic. The ”illusion” of a dynamical application.

```html
#navi ul
{
  margin-left:320px;
  list-style-type:none;
  padding:0;
  overflow:hidden;
}

#navi li
{
  float:left;
}

#navi a:link , a:visited
{
  display:block;
  width:120px;
  font-weight:bold;
  color:#FFFFFF;
  background-color:#6495ed;
  text-align:center;
  padding:4px;
  text-decoration:none;
  text-transform:uppercase;
}

#navi a:hover , a:active
{
  background-color:#4169e1;
}

#main
{
  height:auto;
  margin-left:320px;
}
```
A.1. WEB APPENDIX A. JAVA SOURCE CODE

Listing A.9: computero.css

```html
#logo{
  position:absolute;
}

.current a{
  color:#fff;
  padding-bottom:12px;
}

<%@include file="WEB-INF/jsp/login.jsp"%>

<div id="navi">
  <ul>
    <li><a href="http://localhost:8080/home.html">Home</a></li>
    <li><a href="http://localhost:8080/article.html">Products</a></li>
    <li><a href="http://localhost:8080/comments.html">Comments</a></li>
  </ul>
</div>

<c:if test="${filter == 'article'}">
  <div id="submenu">
    <ul class="submenu">
      <li><a href="http://localhost:8080/article.html">Show</a></li>
    </ul>
  </div>
</c:if>

<c:if test="${filter == 'news'}">
  <div id="submenu">
    <ul class="submenu">
    </ul>
  </div>
</c:if>

<c:if test="${filter == 'home'}">
  <div id="submenu">
    <ul class="submenu">
    </ul>
  </div>
</c:if>

<c:if test="${filter == 'contact'}">
  <div id="submenu">
    <ul class="submenu">
    </ul>
  </div>
</c:if>
```

Listing A.10: navigation.jsp

```html
<div style="text-align:center;margin-top:100%">
  <span style="font-weight:bold">Computero 2012</span>
</div>
</html>
```

Listing A.11: footer.jsp

```html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <title>CompIT</title>
  <!--
  <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js" type="text/javascript"></script>
  <script src="http://code.google.com/apis/ajaxlibs/documentation/index.html#jquery" type="text/javascript"></script>
  <script src="http://ajax.aspnetcdn.com/ajax/jQuery/jquery-1.7.2.min.js" type="text/javascript"></script>
  <script src="http://code.jquery.com/jquery-1.7.2.js" type="text/javascript"></script>
  <link rel="stylesheet" href="computero.css" type="text/css"/>
  -->
  <link rel="stylesheet" href="http://ajax.googleapis.com/ajax/libs/jqueryui/1.7.2.custom.min.css" type="text/css"/>
  <script src="http://ajax.googleapis.com/ajax/libs/jqueryui/1.7.2.custom.min.js" type="text/javascript"></script>
</head>
<body>
</body>
</html>
```

Listing A.12: header.jsp

```html
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%@ taglib uri="http://java.sun.com/jstl/core_rt" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jstl/fmt_rt" prefix="fmt" %>
<%@ taglib uri="http://www.springframework.org/tags" prefix="spring" %>
```

Listing A.13: taglibs.jsp

```html
<!--<
  Created by IntelliJ IDEA.
  User: kristofferwanderydz
-->
A.2. CONFIG

The config section contains all the configuration files from the web application.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>*.html</url-pattern>
    </servlet-mapping>

    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
</web-app>
```

Listing A.16: web.xml
A.2. CONFIG  APPENDIX A. JAVA SOURCE CODE

```
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:p="http://www.springframework.org/schema/p"

<!-- Activates various annotations to be detected in bean classes -->
<context:annotation-config />

<!-- Scans the classpath for annotated components that will be auto-registered as Spring beans. For example @Controller and @Service. Make sure to set the correct base-package -->
<context:component-scan base-package="se.compit" />

<!-- Configures the annotation-driven Spring MVC Controller programming model. Note that, with Spring 3.0, this tag works in Servlet MVC only! -->
<mvc:annotation-driven />

<!-- Load Hibernate related configuration -->
<import resource="hibernate-context.xml" />
```

Listing A.17: applicationContext.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

<!-- Declare a view resolver -->
<bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver"
     p:prefix="/WEB-INF/jsp/" p:suffix=".jsp" />
```

Listing A.18: dispatcher-servlet.xml
Listing A.19: hibernate-context.xml

```xml
<beans>
  <context:property-placeholder location="/WEB-INF/spring.properties"/>

  <!-- Enable annotation style of managing transactions -->
  <tx:annotation-driven transaction-manager="transactionManager" />

  <!-- Declare the Hibernate SessionFactory for retrieving Hibernate sessions -->
  <!-- See http://static.springframework.org/spring/docs/3.0.x/javadoc-api/org/springframework/orm/hibernate3/annotation/AnnotationSessionFactoryBean.html -->
  <bean id="sessionFactory" class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean">
    <p:configLocation>${hibernate.config}</p:configLocation>
    <p:packagesToScan="se.compit"/>
  </bean>

  <!-- Declare a datasource -->
  <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="org.postgresql.Driver"/>
    <property name="url" value="jdbc:postgresql://localhost:5432/postgres"/>
    <property name="username" value="postgres"/>
    <property name="password" value="trapt123"/>
  </bean>

  <!-- Declare a transaction manager -->
  <bean id="transactionManager" class="org.springframework.orm.hibernate3.HibernateTransactionManager">
    <p:sessionFactory-ref="sessionFactory"/>
  </bean>
</beans>
```
A.3. SOURCE

Source files are used to write the functions that execute between the data source and the web application.

```java
package se.compit.service;

import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import se.compit.model.annotation.Administrator;
```

Listing A.20: hibernate.cfg.xml

```xml
<hibernate-configuration>
    <session-factory>
        <!-- We're using MySQL database so the dialect needs to MySQL as well-->
        <property name="hibernate.dialect">org.hibernate.dialect.PostgreSQLDialect</property>
        <!-- Enable this to see the SQL statements in the logs-->
        <property name="show_sql">false</property>
        <!-- This will drop our existing database and re-create a new one. Existing data will be deleted! -->
        <!-- <property name="hbm2ddl.auto">create</property> -->
        <!-- <property name="hibernate.jdbc.batch_size">0</property> -->
    </session-factory>
</hibernate-configuration>
```

Listing A.21: logging.properties

```properties
# database properties
app.jdbc.driverClassName=org.postgresql.Driver
app.jdbc.url=jdbc:postgresql://localhost:5432/postgres
app.jdbc.username=postgres
app.jdbc.password=postgres

#hibernate properties
hibernate.config=/WEB-INF/hibernate.cfg.xml
```

Listing A.22: spring.properties

A.3. Source

Service

Service files are used to write the functions that executes between the data source and the web application.

```java
package se.compit.service;

import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import se.compit.model.annotation.Administrator;
```
```java
import javax.annotation.Resource;
import java.util.List;
import java.util.logging.Logger;

/**
 * Created by IntelliJ IDEA.
 * User: kristofferwanderyd
 * Date: 2012-maj-15
 * Time: 19:05:04
 * To change this template use File | Settings | File Templates.
 */
@Service("administratorService")
@Transactional
public class AdministratorService {

  protected static Logger logger = Logger.getLogger("service");

  @Resource(name = "sessionFactory")
  private SessionFactory sessionFactory;

  /**
   * Retrieves all administrators
   * @return a list of administrators
   */
  public List<Administrator> getAll() {
    // Retrieve session from Hibernate
    Session session = sessionFactory.getCurrentSession();

    // Create a Hibernate query (HQL)
    Query query = session.createQuery("FROM Administrator");

    // Retrieve all
    return query.list();
  }

  public Boolean authenticate(Administrator admin) {
    Boolean auth;

    Session session = sessionFactory.getCurrentSession();

    Query query = session.createQuery("select admin, password from administrator where admin='admin' and password ="+admin.getPassword()+" "");

    if (query.list().size()==1)
      auth = true;
    else auth=false;

    return auth;
  }
}
A.3. SOURCE

APENDIX A. JAVA SOURCE CODE

```java
A.3. SOURCE

APPENDIX A. JAVA SOURCE CODE

public Boolean authenticateSafe(Administrator admin) {
    Boolean auth;
    Session session = sessionFactory.getCurrentSession();
    Query queryName = session.createQuery("select admin from administrator where admin='"+admin.getAdmin()+"'");
    System.out.println(queryName.list().get(0));
    Query queryPassword = session.createQuery("select password from administrator where admin='"+queryName.list().get(0)+"'");
    System.out.println(queryPassword.list().get(0));
    if(queryPassword.list().get(0).equals(admin.getPassword()))
        auth = true;
    else auth=false;
    return auth;
}

/**
 * Retrieves a single administrator
 */
public Administrator get(Integer id) {
    // Retrieve session from Hibernate
    Session session = sessionFactory.getCurrentSession();
    // Retrieve existing administrator first
    Administrator administrator = (Administrator) session.get(
        Administrator.class, id);
    return administrator;
}

/**
 * Adds a new administrator
 */
public void add(Administrator administrator) {
    // Retrieve session from Hibernate
    Session session = sessionFactory.getCurrentSession();
    // Save
    session.save(administrator);
}
```
A.3. SOURCE APPENDIX A. JAVA SOURCE CODE

```java
/**
 * Deletes an existing administrator
 * @param id the id of the existing administrator
 */
public void delete(Integer id) {

    // Retrieve session from Hibernate
    Session session = sessionFactory.getCurrentSession();

    // Retrieve existing administrator first
    Administrator administrator = (Administrator) session.get(Administrator.class, id);

    // Delete
    session.delete(administrator);
}

/**
 * Edits an existing administrator
 */
public void editPass(Administrator administrator) {

    // Retrieve session from Hibernate
    Session session = sessionFactory.getCurrentSession();

    // Retrieve existing administrator via id
    Administrator existingAdministrator = (Administrator) session.get(Administrator.class, administrator.getId());

    // Assign updated values to this administrator
    existingAdministrator.setPassword(administrator.getPassword());

    // Save updates
    session.save(existingAdministrator);
}
```

Listing A.23: AdministratorService.java

```java
package se.compit.service;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import se.compit.model.Article;
import javax.annotation.Resource;
```
import java.util.List;
import java.util.logging.Logger;

/**
 * Created by IntelliJ IDEA.
 * User: kristofferwanderydz
 * Date: 2012-maj-16
 * Time: 01:41:16
 * To change this template use File | Settings | File Templates.
 */
@Service("articleService")
@Transactional
public class ArticleService {

    protected static Logger logger = Logger.getLogger("service");

    @Resource(name = "sessionFactory")
    private SessionFactory sessionFactory;

    /**
     * Retrieves all articles
     * @return a list of articles
     */
    public List<Article> getAll() {
        // Retrieve session from Hibernate
        Session session = sessionFactory.getCurrentSession();

        // Create a Hibernate query (HQL)
        Query query = session.createQuery("FROM Article");

        // Retrieve all
        return query.list();
    }

    /**
     * Retrieves a single article
     */
    public Article get(Integer id) {
        // Retrieve session from Hibernate
        Session session = sessionFactory.getCurrentSession();

        // Retrieve existing article first
        Article article = (Article) session.get(Article.class, id);

        return article;
    }

    /**
     * Adds a new article
     */
    public void add(Article article) {
        // Retrieve session from Hibernate
    }
}
A.3. SOURCE
APPENDIX A. JAVA SOURCE CODE

Listing A.24: ArticleService.java

```java
package se.compit.service;

import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import se.compit.model.Comment;
```

```java
Session session = sessionFactory.getCurrentSession();

// Save
session.save(article);
}

/**
 * Deletes an existing article
 * @param id the id of the existing article
 */
public void delete(Integer id) {
    // Retrieve session from Hibernate
    Session session = sessionFactory.getCurrentSession();

    // Retrieve existing article first
    Article article = (Article) session.get(Article.class, id);

    // Delete
    session.delete(article);
}

/**
 * Edits an existing article
 */
public void editPass(Article article) {
    // Retrieve session from Hibernate
    Session session = sessionFactory.getCurrentSession();

    // Retrieve existing article via id
    Article existingArticle = (Article) session.get(Article.class, article.getId());

    // Assign updated values to this article
    existingArticle.setCategory(article.getCategory());
    existingArticle.setName(article.getName());

    // Save updates
    session.save(existingArticle);
}
```

50
import javax.annotation.Resource;
import java.util.List;

import java.util.logging.Logger;

/**
 * Created by IntelliJ IDEA.
 * User: kristofferwanderydz
 * Date: 2012-maj-16
 * Time: 01:41:16
 * To change this template use File | Settings | File Templates.
 */
@Service("commentService")
@Transactional
public class CommentService {

protected static Logger logger = Logger.getLogger("service");

@Resource(name = "sessionFactory")
privateSessionFactory sessionFactory;

/**
 * Retrieves all comments
 *
 * @return a list of comments
 */
public List<Comment> getAll() {
    // Retrieve session from Hibernate
    Session session = sessionFactory.getCurrentSession();
    // Create a Hibernate query (HQL)
    Query query = session.createQuery("FROM Comment");
    // Retrieve all
    return query.list();
}

/**
 * Retrieves a single comment
 */
public Comment get(Integer id) {
    // Retrieve session from Hibernate
    Session session = sessionFactory.getCurrentSession();
    // Retrieve existing comment first
    Comment comment = (Comment) session.get(Comment.class, id);
    return comment;
}

/**
 * Adds a new comment
 */
public void add(Comment comment) {
}
// Retrieve session from Hibernate
SessionFactory session = sessionFactory.getCurrentSession();

// Save
session.save(comment);

/**
 * Deletes an existing comment
 *
 * @param id the id of the existing comment
 */
public void delete(Integer id) {

    // Retrieve session from Hibernate
    Session session = sessionFactory.getCurrentSession();

    // Retrieve existing comment first
    Comment comment = (Comment) session.get(Comment.class, id);

    // Delete
    session.delete(comment);
}

/**
 * Edits an existing comment
 */
public void editPass(Comment comment) {

    // Retrieve session from Hibernate
    Session session = sessionFactory.getCurrentSession();

    // Retrieve existing comment via id
    Comment existingComment = (Comment) session.get(Comment.class, comment.getId());

    // Assign updated values to this comment

    // Save updates
    session.save(existingComment);
}

Listing A.25: CommentService.java

Model

The objects used in the web application are defined in these model files, they are also the link to the data source.

package se.compit.model;
import javax.persistence.*;
import java.io.Serializable;

/**
 * Created by IntelliJ IDEA.
 * User: kristofferwanderydz
 * Date: 2012-maj-15
 * Time: 18:55:23
 * To change this template use File | Settings | File Templates.
 */

@Entity
@Table(name = "administrator")
public class Administrator implements Serializable {
  private Integer id;
  
  private String admin;
  
  private String password;

  public Integer getId() {
    return id;
  }

  public void setId(Integer id) {
    this.id = id;
  }

  public String getAdmin() {
    return admin;
  }

  public void setAdmin(String admin) {
    this.admin = admin;
  }

  public String getPassword() {
    return password;
  }

  public void setPassword(String password) {
    this.password = password;
  }
}

Listing A.26: Administrator.java
/**
 * Created by IntelliJ IDEA.
 * User: kristofferwanderydz
 * Date: 2012−maj−16
 * Time: 01:37:12
 * To change this template use File | Settings | File Templates.
 */

@Entity
@Table(name = "article")
public class Article implements Serializable {
    @Id
    @GeneratedValue
    private Integer id;

    @Column(name = "category")
    private String category;

    @Column(name = "name")
    private String name;

    @Column(name = "price")
    private String price;

    public String getPrice() {
        return price;
    }

    public void setPrice(String price) {
        this.price = price;
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getCategory() {
        return category;
    }

    public void setCategory(String category) {
        this.category = category;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
Listing A.27: Article.java

```java
package se.compit.model;

import javax.persistence.*;
import java.io.Serializable;

/**
 * Created by IntelliJ IDEA.
 * User: kristofferwanderydz
 * Date: 2012-09-16
 * Time: 17:02:02
 * To change this template use File | Settings | File Templates.
 */
@Entity
@Table(name = "comment")
public class Comment implements Serializable {

    @Id
    @GeneratedValue
    private Integer id;

    @Column(name = "sender")
    private String sender;

    @Column(name = "message")
    private String message;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getSender() {
        return sender;
    }

    public void setSender(String sender) {
        this.sender = sender;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
}
```
Listing A.28: Comment.java

Controller

The controller is the internal part of the dynamical behavior (the functional jsp's are the external A.11) as shown in Fig 3.1

```java
package se.compit.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAndView;
import org.springframework.web.util.WebUtils;
import se.compit.model.Administrator;
import se.compit.model.Article;
import se.compit.service.AdministratorService;
import se.compit.service.ArticleService;
import se.compit.service.CommentService;
import javax.annotation.Resource;
import javax.servlet.http.HttpServletRequest;
import java.util.List;
import java.util.logging.Logger;

/**
 * Created by IntelliJ IDEA.
 * User: kristofferwanderydz
 * Date: 2012-maj-15
 * Time: 15:40:54
 * To change this template use File | Settings | File Templates.
 */
@Controller
@SessionAttributes("admin")
public class SuperController {

    /** ModelView Injection. Spring MVC */
    @RequestMapping("/universal")
    public String look_what_much_time_i_save_on_codeing(
        @RequestParam(required = true, value = "navi", defaultValue = "home") String string)
    {
        System.out.print("universal");
        return string;
    }

    /** END ModelView Injection. Spring MVC */

    @ModelAttribute("loginAttribute")
```

56
public Administrator getLogInAttribute()
{
    Administrator admin = new Administrator();

    return admin;
}

protected static Logger logger = Logger.getLogger("controller");

@Resource(name="administratorService")
private AdministratorService administratorService;

@Resource(name="articleService")
private ArticleService articleService;

@Resource(name="commentService")
private CommentService commentService;

@RequestMapping("/home")
public String homeView()
{
    // This will resolve to /WEB-INF/jsp/home.jsp
    return "home";
}

@RequestMapping(method = RequestMethod.GET)
public String main()
{
    return "home";
}

@RequestMapping(value = "/article", method = RequestMethod.GET)
public String articleView(Model model)
{
    List<Article> articles = articleService.getAll();
    model.addAttribute("articles", articles);

    return "article";
}

@RequestMapping(value = "/addarticle", method = RequestMethod.POST)
public ModelAndView addArticle(@ModelAttribute("articleAttribute") Article article)
{
    articleService.add(article);
}
return new ModelAndView("home","Message","Product added!");
}

@RequestMapping(value = "/comments", method = RequestMethod.GET)
public String viewComments(Model model)
{
    List<Comment> comments = commentService.getAll();

    model.addAttribute("comments", comments);
    model.addAttribute("commentAttribute", new Comment());

    return "comments";
}

@RequestMapping(value = "#/comments", method = RequestMethod.POST)
public ModelAndView addComment(@ModelAttribute("commentAttribute") Comment comment)
{
    commentService.add(comment);

    return new ModelAndView("home","Message","Comment added!");
}

@RequestMapping(value = "/addarticle", method = RequestMethod.GET)
public String addArticle(Model model)
{
    model.addAttribute("articleAttribute", new Article());

    return "articleadd";
}

@RequestMapping("/contact")
public ModelAndView contactView(@RequestParam(required=false, value="message") String message)
{
    ModelAndView mav = new ModelAndView("contact");

    mav.addObject("message", message);

    return mav;
A.3. SOURCE

APPENDIX A. JAVA SOURCE CODE

```
/* @RequestMapping ("/contact")
public ModelAndView contactView () {
    ModelAndView mav = new ModelAndView("contact");
    mav.addObject("message","Under construction");
    return mav;
}
*/

@RestController
@RequestMapping ("/news")
public ModelAndView newsView () {
    return new ModelAndView("news");
}

@RestController
@RequestMapping ("/blog")
public ModelAndView blogView () {
    return new ModelAndView("blog");
}

@RestController
@RequestMapping (value = "/login", method = RequestMethod.POST)
public ModelAndView login (@ModelAttribute("loginAttribute") Administrator admin, HttpServletRequest request) {
    Boolean auth = administratorService.authenticateSafe(admin);
    if (auth) {
        WebUtils.getSessionAttribute(request, "user", admin.getAdmin());
        return new ModelAndView("home", "Message", "Authorized");
    } else
        return new ModelAndView ("home", "messageFail", "Forgot your password?");
}
```

Listing A.29: SuperController.java

59
Appendix B

Online resources

This file will be kept online at least for a period of 3 years.

B.1 Prototype

http://code.google.com/p/web-application-security/

B.2 Videos

B.2.1 Exploit

Authentication and Session Management
http://www.youtube.com/watch?v=8bEwkW-A0ZQ&feature=plcp

Cross-Site Scripting: Stored
http://www.youtube.com/watch?v=Lo5vVVeys0w&feature=plcp

Cross-Site Scripting: Reflected
http://www.youtube.com/watch?v=ROzdRqChXxg&feature=plcp

Request Forgery
http://www.youtube.com/watch?v=IqwpSEzLx0o&feature=plcp

SQL Injection
http://www.youtube.com/watch?v=ieS1CiTIkzw&feature=plcp
B.3. REPORT

APPENDIX B. ONLINE RESOURCES

B.2.2 Secure Authentication and Session Management

http://www.youtube.com/watch?v=OVlWAxjwNnE&feature=plcp

Cross-Site Scripting: Stored

http://www.youtube.com/watch?v=Bv-8E-0_-kM&feature=plcp

Cross-Site Scripting: Reflected

http://www.youtube.com/watch?v=ONOswkDW3lU&feature=plcp

Request Forgery

http://www.youtube.com/watch?v=IqwpSEzlLxo&feature=plcp

SQL Injection


B.3 Report

B.4 Presentation