

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.

Contents

1	Introduction	1
1.1	Background	1
1.2	Purpose and Objective	2
1.2.1	Research Questions	2
1.2.2	Contribution	2
1.3	Method	2
1.4	Target group	3
1.5	Report structure	4
2	Theoretical Background	5
2.1	Vulnerabilities	5
2.1.1	SQL Injection	5
2.1.2	Cross-Site Scripting - XSS	8
2.1.3	Request forgery	10
2.1.4	Authentication and Session Management	14
2.2	Spring Specific Vulnerabilities	15
3	Implementation	18
3.1	Proposed solution	19
3.1.1	Cross-site scripting (XSS)	19
3.1.2	Injection	21
3.1.3	Authentication and Session Management	22
4	Analysis of the Results	25
4.1	Related Work	25
4.2	Discussion	26
4.3	Limitations	27
5	Conclusions	28
5.1	Future work	28
	Bibliography	29
	Glossary	31

Acronyms	32
Appendices	34
A Java source code	35
A.1 Web	35
A.2 ConFig	42
A.3 Source	45
B Online resources	60
B.1 Prototype	60
B.2 Videos	60
B.2.1 Exploit	60
B.2.2 Secure	61
B.3 Report	61
B.4 Presentation	61

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

3.4	Secure - Reflected XSS - Nothing generates	20
3.5	Secure - Reflected XSS - The script not generated	20
3.6	Secure - Reflected XSS for all browsers	20
3.7	Secure - Reflected XSS for all browsers	21
3.8	Secure - SQL Injection:Development model	21
3.9	Secure - SQL Injection:Authorization	22
3.10	Secure - SQL Injection:Authorization	22
3.11	Secure - SQL Injection - Encrypted Login 1	23
3.12	Secure - SQL Injection- Encrypted Login 2	23

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, 5, 13] discussed in section 4.1

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[6, 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 1.1. 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 section4 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.

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.

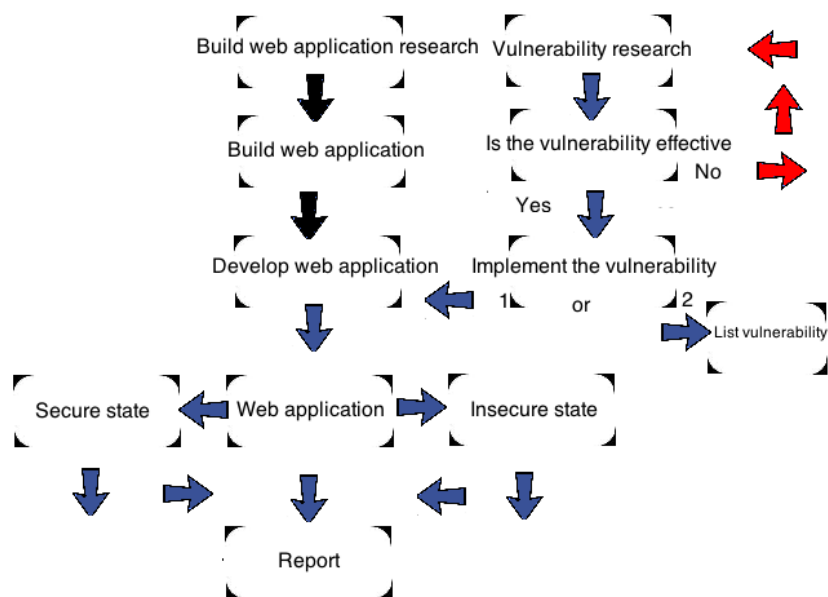


Figure 1.1: Progress method

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 follows5.1, such as acronyms, terminology, references and appendix5.1.

Chapter 2

Theoretical Background

This chapter show how to exploit the vulnerabilities, implemented in the prototype. It is of importance to know how the exploits takes place, to better understand the solutions in next chapter3 , 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[14, 15]. 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[14, 15] and books[13].

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 listing2.1.1 the sequence flow is presented, for authentication on data storage. Viewed from an attackers 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 attacker's misuse model and the normal intended use model.

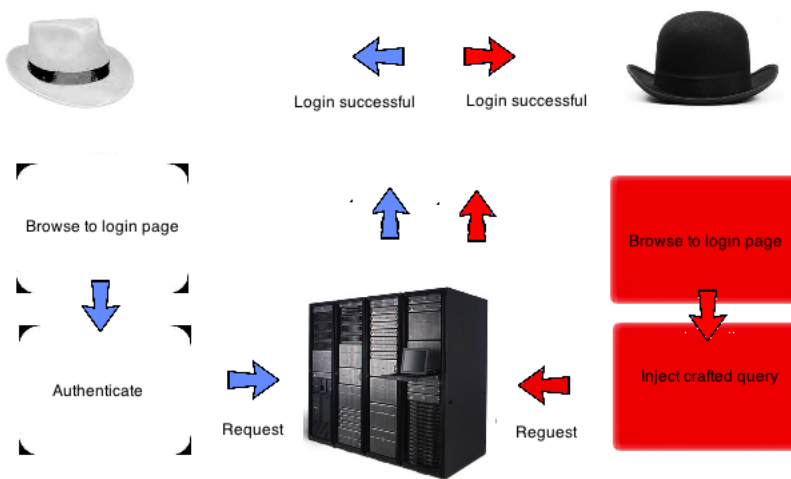


Figure 2.1: Misuse case - SQL injection:Authentication

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.



Figure 2.2: Exploit - SQL Injection:Authentication

In Listing 2.1 the login controller is displayed.

```
1 @RequestMapping(value = "/login", method = RequestMethod.POST)
   public ModelAndView login(@ModelAttribute("loginAttribute")
   Administrator admin, HttpServletRequest request) {
3
   Boolean auth = administratorService.authenticate(admin);
5
   if(auth) {
```

2.1. VULNERABILITIES CHAPTER 2. THEORETICAL BACKGROUND

```
7         WebUtils.setSessionAttribute(request, "user", admin.  
9             getAdmin());  
11     }  
13     else  
14         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 listing2.2(Line 6,7,8).

```
2 public Boolean authenticate(Administrator admin){  
3     Boolean auth;  
4  
5     Session session = sessionFactory.getCurrentSession();  
6     Query query = session.createQuery("select admin,password  
7         from administrator where admin='"+admin.getAdmin()+"'  
8         and password='"+admin.getPassword()+"'");  
9  
10    if(query.list().size()==1)  
11        auth = true;  
12    else  
13        auth=false;  
14    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.

```
2 Query query = session.createQuery("select admin,password  
3     from administrator where admin='\${admin}'--(  
4     everything here is now a comment)\}' and password='"+  
5     admin.getPassword()+"'");  
6 \}
```

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.



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.



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.

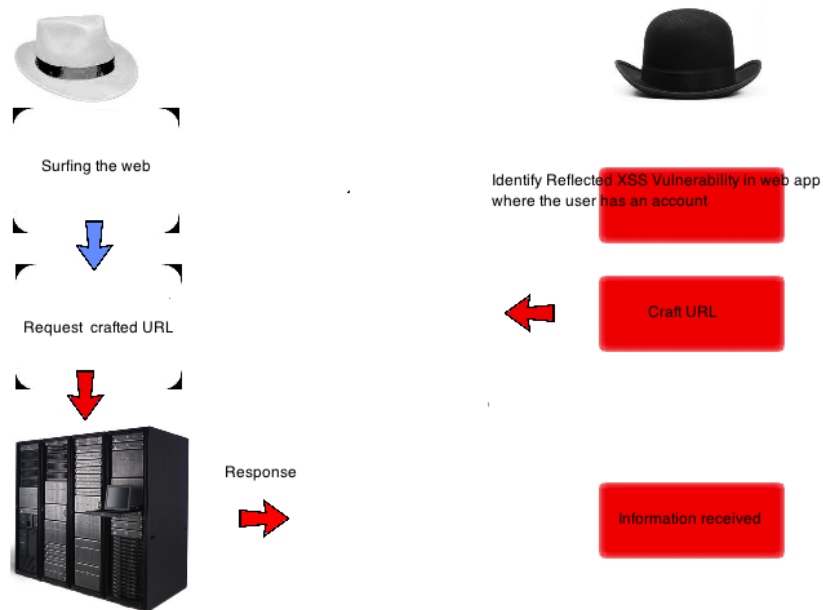


Figure 2.5: Misuse case - Cross-site Scripting:Reflected

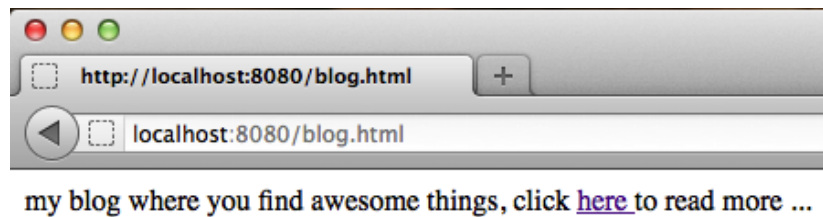


Figure 2.6: Exploit - Cross-site Scripting:Reflected

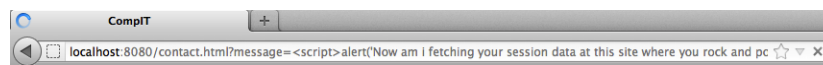


Figure 2.7: Exploit - Cross-site Scripting:Reflected:URL

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].

Fig2.10 presents the sequence flow, and Fig2.11 demonstrates the misuse

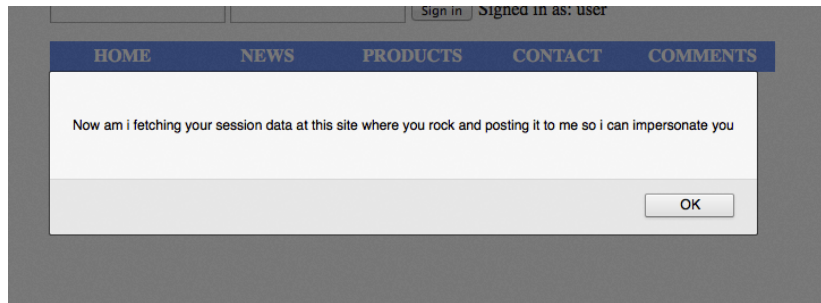


Figure 2.8: Exploit - Cross-site Scripting:Reflected:Generated



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

case of a stored Cross-site scripting attack.



1. Attacker identify a user input possibility to store his attack
2. Attacker store his malicious code
3. User visits the web page where the stored attack is stored
4. Attack executes
5. Attack succeeds

Figure 2.10: Sequence - Cross-site Scripting:Stored

The attack starts with a posted script in the guest book, shown in Fig2.13. The content of the script are invisible, so the user is unaware of that the script, actually is executed and running as viewed in Fig2.16.

An alert is generated for demonstration, and generates for every user who views the guest book. Shown in Fig2.14.

Firebug shows the hidden and auto generated javascript, shown in Fig2.15.

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

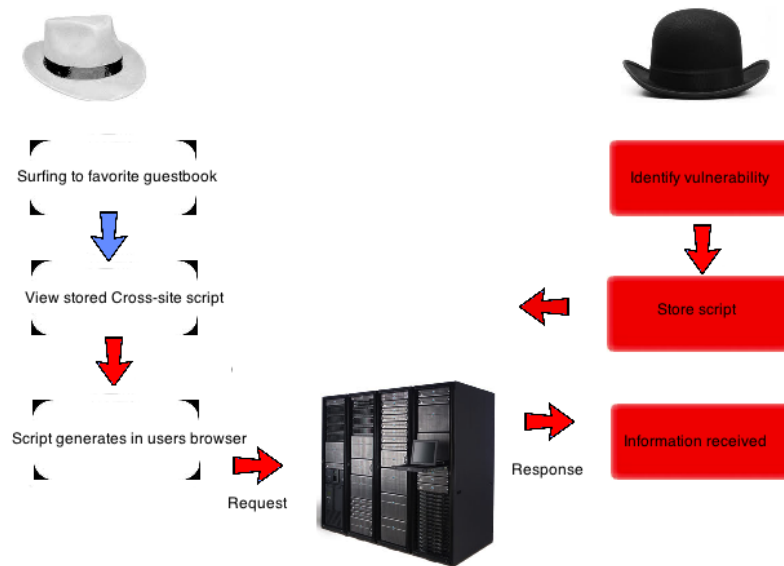


Figure 2.11: Misuse case - Cross-site Scripting:Stored

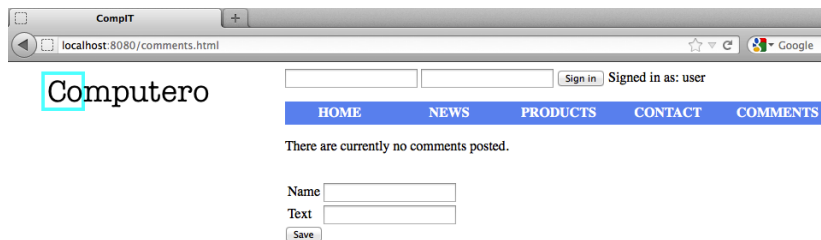


Figure 2.12: Identify vulnerability - Cross-site Scripting:Stored

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

Signed in as: user

HOME **NEWS** **PRODUCTS** **CONTACT** **COMMENTS**

There are currently no comments posted.

Name

Text

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

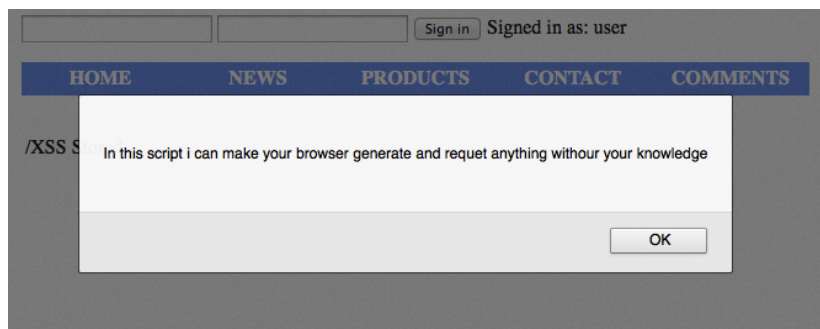


Figure 2.14: Exploit vulnerability - Cross-site Scripting:Stored

```
<td>/XSS Stored</td>
</tr>
<tr>
<td><script>alert("In this script i can make your browser generate and requet anything without
your knowledge");</script></td>
</tr>
</tbody>
</table>
```

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

Signed in as: user

HOME **NEWS** **PRODUCTS** **CONTACT** **COMMENTS**

/XSS Stored

Name

Text

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

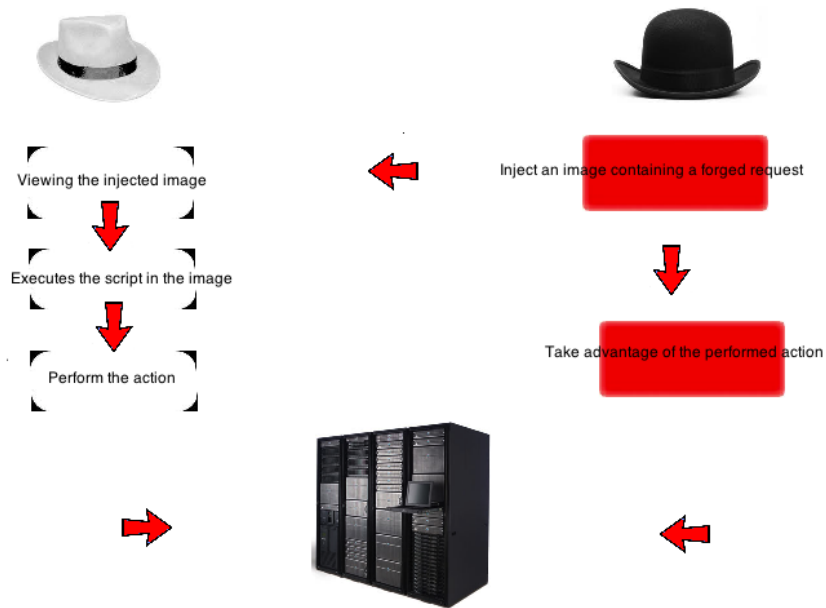


Figure 2.17: Misuse case - Request Forgery

The attacker identify an article of interest, and notice that the form contains the variables Category, Name and Price. As shown in Fig2.18.

<input type="text"/>	<input type="text"/>	<input type="button" value="Sign in"/>	Signed in as: user	
HOME	NEWS	PRODUCTS	CONTACT	COMMENTS
<input type="button" value="SHOW"/>				
Category	Name	Price		
Desktop	Mouse	199		
Vehicle	Segway	60000		

Figure 2.18: Exploit - Request Forgery

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 Fig2.19. An administrator normally has higher privilege than normal users, and can therefore take actions not normal users can.



Figure 2.19: Exploit - Request Forgery

The attacker wrote a script based on the analysis of the web application, the script is shown in Fig2.20.

```
<tr>
<td>/Request Forgery</td>
</tr>
<tr>
<td><form method="post" action="http://localhost:8080/addarticle.html?category=veichle&name
=segway&price=3"><input type="submit" value="Admin click here"></form></td>
</tr>
```

Figure 2.20: Exploit - Request Forgery

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 Fig2.21.



Figure 2.21: Exploit - Request Forgery

Fig2.22 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 Fig2.23.

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



Figure 2.22: Exploit - Request Forgery



Figure 2.23: Exploit - Request Forgery

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] .

Figure2.24 show a sequence flow, and fig2.25 present a misuse case how sniffing can occur.

Fig2.26 demonstrates an admin signing in, and Fig2.27 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 Fig2.28, 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 in3.1.3.

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 information2.2.

1. Spring MVC:ModelView Injection[9]

2.2. SPRING SPECIFIC VULNERABILITIES THEORETICAL BACKGROUND



Figure 2.24: Sequence - Authentication management

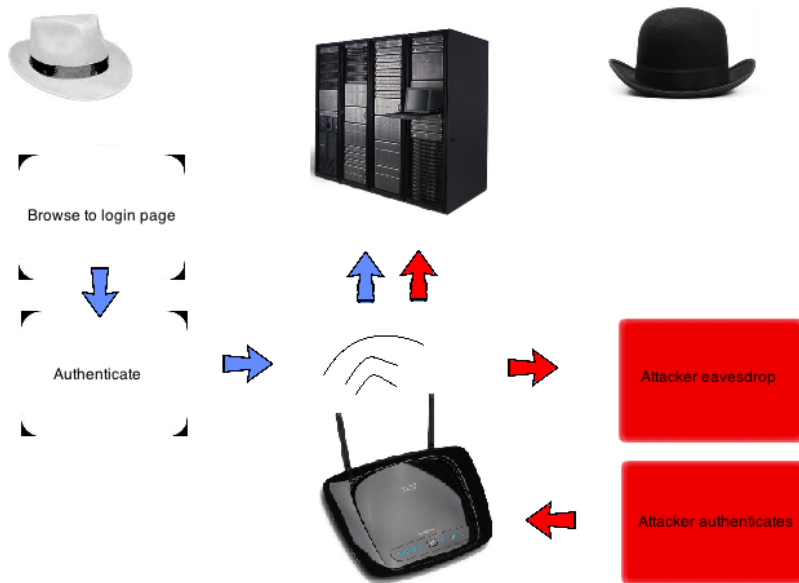


Figure 2.25: Misuse case - Authentication management

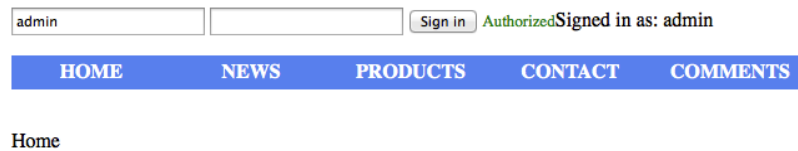


Figure 2.26: Exploit - Authentication management

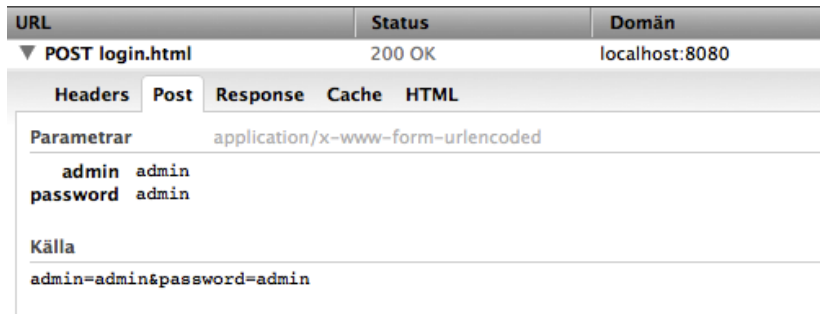


Figure 2.27: Exploit - Authentication management

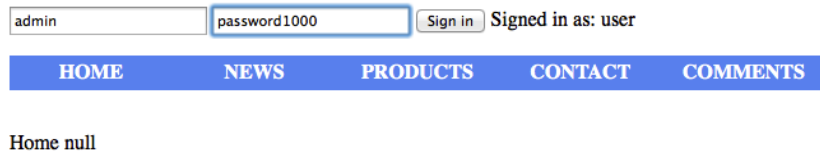


Figure 2.28: Exploit - Authentication management

2. Spring MVC:Data Submission to Non-Editable Fields[9]
3. Spring Framework:Remote Denial of Service Vulnerability[10]
4. Spring Framework:Execution of arbitrary code[10]
5. Spring Framework:Information disclosure[10]
6. Spring Framework:Serialization-based remoting vulnerabilities[10]
7. Spring Security:Bypass of security constraints[11]
8. Spring Security:Privilege escalation when using RunAsManager[11]
9. Spring Security:Header injection vulnerability[11]
10. Spring Security:Serialization-based remoting vulnerabilities[11]

Chapter 3

Implementation

Following sections in this chapter propose solutions to secure the vulnerabilities from chapter2, and a short description of the prototype. It is important to understand Chapter2 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 Fig3.1. The prototype was built to test the security in Spring and provide real cases.

The prototype is available externallyB.1, but also listed in this project5.1.

The infrastructure is visible inB.1 and a short description is given in the appendix5.1 for each section of the prototype.

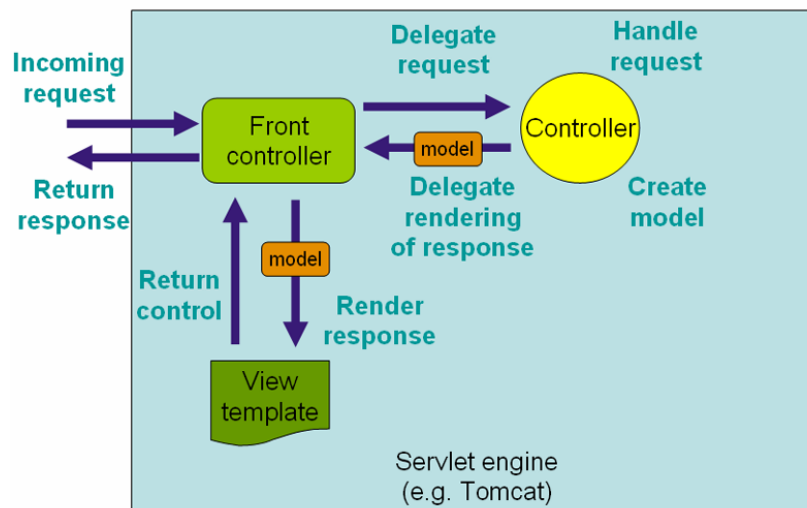


Figure 3.1: Spring MVC:Model View Controller

Figure3.1[1] explains the work flow in the prototype very well.

Incoming requests to the web application passes on by the web.xml A.1, to the dispatcher-servlet.xmlA.1. Requests are later managed by the SuperController.javaA.3 for navigation in the web application.

The controller models the view from the request, resulting in a response in form of a jspA.1. Jsp contains HTML, usually interpreted by the browser that sent the request.

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.

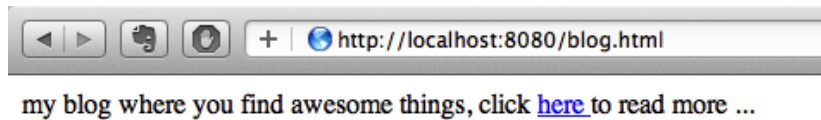


Figure 3.2: Secure - Reflected XSS

In Fig3.2 the web browser IE with the XSS-filter gets attacked, with one reflected Cross-site scripting attack as shown in section2.1.2.

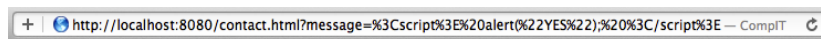


Figure 3.3: Secure - Reflected XSS - Redirected URL

The filter is active by default. The script shown in Fig3.5 will be requested, but wont get generated as shown in Fig3.3. Fig3.4 shows no alert window popping up and the vulnerability is secured.

Solution 2

Browsers such as Firefox, are still vulnerable as we saw in Fig2.8. The vulnerability was demonstrated in this section2.1.

Browsers that lack the Cross-site scripting filter, must escape the html tags manually. Such as those in Fig3.1.

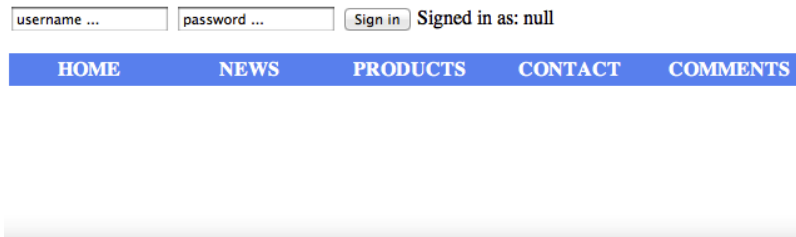


Figure 3.4: Secure - Reflected XSS - Nothing generates

```

<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.

```

1  ${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.

```

1  <c:out value="${message}"/>

```

Listing 3.3: Secure Message Output

```

localhost:8080/contact.html?message=<script>alert("Now am I fetching your session data at this site where you rock and pc ☆ ▾ ©

```

Figure 3.6: Secure - Reflected XSS for all browsers

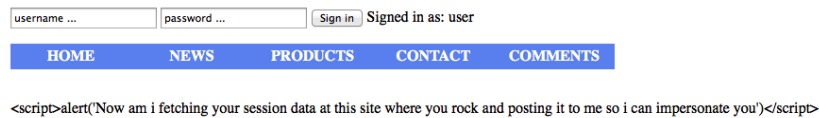


Figure 3.7: Secure - Reflected XSS for all browsers

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.

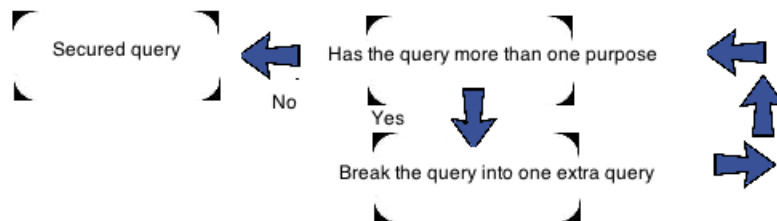


Figure 3.8: Secure - SQL Injection:Development model

```

1 public Boolean authenticateSafe(Administrator admin)
2     {
3         Boolean auth;
4
5         Session session = sessionFactory.getCurrentSession();
6
7         //One query to fetch the administrator
8         Query queryName = session.createQuery("select admin from
9             administrator where admin='"+admin.getAdmin()+"'");
10
11        //One query to fetch the password
12        Query queryPassword = session.createQuery("select
13            password from administrator where admin='"+queryName.
14                list().get(0)+"'");
15
16        //One authentication process strictly related to the
17            administrator object, now the list size doesn't matter.
18            if(queryPassword.list().get(0).equals(admin.getPassword()))
19                to

```

```

15     auth = true;
17     else auth=false;
19     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.

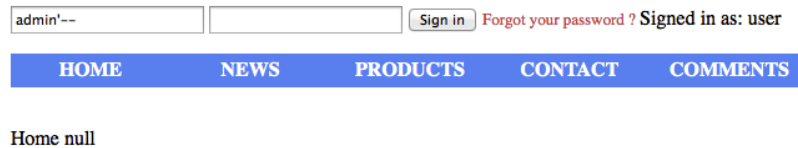


Figure 3.9: Secure - SQL Injection:Authorization

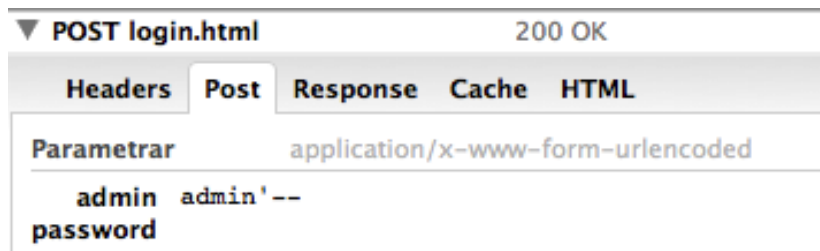


Figure 3.10: Secure - SQL Injection:Authorization

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 enumeration 3.1.3. Encrypt the posted password and the sniffer can

not tell what hides, as in Fig3.5 the password is encrypted with a MD5 hash.

```

1 <form name="myform" action="http://localhost:8080/login.html"
  method="POST">
  <input type="text" id="admin" name="admin" value="username
    ..." onclick="this.value=""/>
3 <input type="text" id="password" name="password" value="
  password ..." onclick="this.value=""/>
  <input type="button" value="Sign in" onclick="encrypt(
    password.value,admin.value);">
5 </form>
</div>
7 <script type="text/javascript">
9   function encrypt(pass, user) {
    encryptedPass = calcMD5(pass);
11
    window.location = "http://localhost:8080/login.html?admin="
      + user + "&password=" + encryptedPass;
13
  }
15 </script>

```

Listing 3.5: Safe Data Transmission

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

admin admin Sign in Signed in as: user

HOME NEWS PRODUCTS CONTACT COMMENTS

Home null

Figure 3.11: Secure - SQL Injection - Encrypted Login 1

URL	Status															
GET login.html?admin=a...a7438!	200 OK															
<table border="1"> <thead> <tr> <th>Params</th> <th>Headers</th> <th>Response</th> <th>Cache</th> <th>HTML</th> </tr> </thead> <tbody> <tr> <td>admin</td> <td>admin</td> <td></td> <td></td> <td></td> </tr> <tr> <td>password</td> <td>21232f297a57a5a743894a0e4a801fc3</td> <td></td> <td></td> <td></td> </tr> </tbody> </table>		Params	Headers	Response	Cache	HTML	admin	admin				password	21232f297a57a5a743894a0e4a801fc3			
Params	Headers	Response	Cache	HTML												
admin	admin															
password	21232f297a57a5a743894a0e4a801fc3															

Figure 3.12: Secure - SQL Injection- Encrypted Login 2

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

follows.

1. Lets say is the thousandt time the user login, the entered password "admin" will be appended to equal "admin1000" and sent encrypted "e2559927016276b8f01e7b13de6d5c41" 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[15][14]. 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 Work4.1. 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 fall 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 to much time. The vulnerabilities concerns all platforms and frameworks, but in development this project will use Java with framework 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-

veloper can pick the right environment, and take the right precautions. To customize a security level that match the applications purpose, and know which guarantees that 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 of value to compare 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
- [2] JJ Garrett. Ajax: A new approach to web applications. pages 18–21, 2005. 2
- [3] William Hau, Steve Taylor, Tim Smith, and A Russell. A Guide to Building Secure Web Applications. *The Open Web*, 2002. 25
- [4] DH Judson. Web browser with dynamic display of information objects during linking, 1996. 2
- [5] Mario Konecki and Hutinski. Secure web applications? *30th Jubilee International*, 2007. 1, 25
- [6] Jennifer KRISHNAMURTHY, Balachander REXFORD. *Web protocols and practice : HTTP/1.1, networking protocols, caching and traffic measurement*. 2001. 2
- [7] Stuart McClure, Joel Scambray, and George Kurtz. *Hacking Exposed*. Fourth edition, 2003. 1
- [8] Spring. All SpringSource security vulnerabilities, 2009. 15
- [9] Team Spring. Spring MVC, 2008. 15, 17
- [10] Team Spring. Spring Framework, 2009. 17
- [11] Team Spring. Spring Security, 2010. 17
- [12] McClure Stuart, Scambray Joel, and Kurtz George. *Hacking Exposed 7 Network Security Secrets & Solutions Seventh Edition: Network Security Secrets and Solutions*. 7 edition, 2012. 1
- [13] Dafydd Stuttard and Markus Pinto. *The Web Application Hackers Handbook*. Second edition, 2011. 1, 5, 8, 9, 11, 14, 15, 25
- [14] Trustwave. 2012 Global Security Report. 2012. 1, 2, 5, 8, 9, 11, 26
- [15] J Williams. OWASP Top 10 2010. *OWASP Foundation, April*, 2010. 1, 5, 8, 9, 11, 26

Glossary

1. Spring - Framework used for development of web applications in Java. <http://www.springsource.org/>
2. Tomcat - A Web server, manages the communication between web browser and web application. <http://tomcat.apache.org/>
3. Hibernate - Framework used to create objects in the web application, from the data source. <http://www.hibernate.org/>
4. PostgreSQL - Data source for storage of data. <http://www.postgresql.org/>

Acronyms

BTH = Blekinge Tekniska Hogskola <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 Fig3.1. They contain the HTML code, and is a part of the web applications dynamical behavior.

```
1 <%@include file=" ../../ taglibs .jsp " %>
2 <c:set var=" filter " value=" article "/>
3 <%@include file=" ../../ header .jsp " %>
4 <%@include file=" ../../ navigation .jsp " %>
5 <c:if test=" ${not empty articles} ">
6     <table width=" 320 ">
7         <thead style=" text-align : left ; border-spacing : 15px ">
8             <tr>
9                 <th>Category</th>
10                <th>Name</th>
11                <th>Price</th>
12            </tr>
13        </thead>
14        <c:forEach items=" ${articles} " var=" article ">
15            <tbody>
16
17                <tr>
18                    <td><c:out value=" ${article . category} "/></td>
19                    <td><c:out value=" ${article . name} "/></td>
20                    <td><c:out value=" ${article . price} "/></td>
21
22                </tr>
23            </c:forEach>
24        </tbody>
25    </table>
26</c:if>
27
28<c:if test=" ${empty articles} ">
29    There are currently no articles in the list .
30</c:if>
```

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

Listing A.1: article.jsp

```

1 <%@include file="../../taglibs.jsp" %>
2 <c:set var="filter" value="article"/>
3 <%@include file="../../header.jsp" %>
4 <%@include file="../../navigation.jsp" %>
5 <form:form action="http://localhost:8080/addarticle.html"
6   modelAttribute="articleAttribute" method="POST">
7   <table>
8     <tr>
9       <td><form:label path="category">Category</form:label></td>
10      <td><form:input path="category"/></td>
11    </tr>
12
13    <tr>
14      <td><form:label path="name">Name</form:label></td>
15      <td><form:input path="name"/></td>
16    </tr>
17
18    <tr>
19      <td><form:label path="price">Price</form:label></td>
20      <td><form:input path="price"/></td>
21    </tr>
22  </table>
23
24  <input type="submit" value="Save"/>
25 </form:form>
26 <%@include file="../../footer.jsp" %>
```

Listing A.2: articleadd.jsp

```

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

Listing A.3: blog.jsp

```

1 <%@include file="../../taglibs.jsp" %>
2 <c:set var="filter" value="comments"/>
3 <%@include file="../../header.jsp" %>
4 <%@include file="../../navigation.jsp" %>
5 <c:if test="{empty comments}">
6   There are currently no comments posted.
7 </c:if>
8 <c:forEach items="{comments}" var="comment">
9   <table style="margin-top:30px">
10    <tbody>
11      <tr>
```

```

13         <td><c:out value="{comment.sender}"/></td>
14     </tr>
15     <tr>
16         <td><c:out value="{comment.message}"/></td>
17     </tr>
18 </tbody>
19 </table>
20 </c:forEach>
21 <form:form modelAttribute="commentAttribute" method="POST" action="
22     http://localhost:8080/comments.html">
23     <table style="margin-top:30px;">
24         <tr>
25             <td><form:label path="sender">Name</form:label></td>
26             <td><form:input path="sender"/></td>
27         </tr>
28
29         <tr>
30             <td><form:label path="message">Text</form:label></td>
31             <td><form:input path="message"/></td>
32         </tr>
33     </table>
34
35     <input type="submit" value="Save"/>
36 </form:form>
37
38 <%@include file="../../footer.jsp" %>

```

Listing A.4: comments.jsp

```

1 <%—
2     Created by IntelliJ IDEA.
3     User: kristofferwanderydz
4     Date: 2012-maj-15
5     Time: 14:27:32
6     To change this template use File | Settings | File Templates.
7 —%>
8 <%@include file="../../taglibs.jsp" %>
9 <c:set var="filter" value="contact"/>
10 <%@include file="../../header.jsp" %>
11 <%@include file="../../navigation.jsp" %>
12     <c:out value="{message}"/>
13 <%@include file="../../footer.jsp" %>

```

Listing A.5: contact.jsp

```

1 <%—
2     Created by IntelliJ IDEA.
3     User: kristofferwanderydz
4     Date: 2012-maj-17
5     Time: 14:47:21
6     To change this template use File | Settings | File Templates.
7 —%>
8
9 <div style="margin-left:320px;">

```

```

11 <form:form action="http://localhost:8080/login.html" method="
    POST" modelAttribute="loginAttribute">
    <form:label path="admin"/>
    <form:input path="admin"/>
13 <form:label path="password"/>
    <form:password path="password"/>
15 <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>
17 </form:form>
</div>

```

Listing A.6: login.jsp

```

<%@ taglib prefix="form" uri="http://www.springframework.org/tags/
    form" %>
2 <%—
    Created by IntelliJ IDEA.
4    User: kristofferwanderydz
    Date: 2012-maj-17
6    Time: 14:47:21
    To change this template use File | Settings | File Templates.
8 —%>
10 <div style="margin-left:320px;">
    <form name="myform" action="http://localhost:8080/login.html"
        method="POST">
12 <input type="text" id="admin" name="admin" value="username
        ..." onclick="this.value=""/>
    <input type="text" id="password" name="password" value="
        password ..." onclick="this.value=""/>
14 <input type="button" value="Sign in" onclick="encrypt(
        password.value,admin.value);"><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>
16 </div>
<script type="text/javascript">
18
    function encrypt(pass, user) {
20        encryptedPass = calcMD5(pass);
22
        window.location = "http://localhost:8080/login.html?admin="
            + user + "&password=" + encryptedPass;
24
    }
26 </script>

```

Listing A.7: loginSafe.jsp

```

2 <%—
    Created by IntelliJ IDEA.

```

```

  User: kristofferwanderydz
4   Date: 2012-maj-15
   Time: 14:27:32
6   To change this template use File | Settings | File Templates.
  —%>
8  <%@include file="../../taglibs.jsp" %>
   <c:set var="filter" value="news"/>
10 <%@include file="../../header.jsp" %>
   <%@include file="../../navigation.jsp" %>
12 News
   <%@include file="../../footer.jsp" %>
```

Listing A.8: news.jsp

Non-functional jsps

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

```

1 #navi ul
  {
3   margin-left:320px;
   list-style-type:none;
5   padding:0;
   overflow:hidden;
7   }

9 #navi li
  {
11  float:left;
  }

13 #navi a:link,a:visited
  {
15  display:block;
   width:120px;
17  font-weight:bold;
   color:#FFFFFF;
19  background-color:#6495ed;
   text-align:center;
21  padding:4px;
   text-decoration:none;
23  text-transform:uppercase;
25  }

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

31 #main{
33   height:auto;
   margin-left:320px;
35 }
```

```

37 |
39 | #logo{
41 |     position: absolute;
43 | }
45 | .current a{
47 |     color:# fff;
        padding-bottom:12px;
    }

```

Listing A.9: computero.css

```

<div id="logo"><a href="http://localhost:8080/"></a></div>
2 | <%@include file="WEB-INF/jsp/login.jsp"%>
<div id="navi">
4 | <ul>
6 |     <li><a href="http://localhost:8080/home.html">Home</a></li>
7 |     <li><a href="http://localhost:8080/news.html">News</a></li>
8 |     <li><a href="http://localhost:8080/article.html">Products</a></
9 |     li>
10 |     <li><a href="http://localhost:8080/contact.html?message=Under+
11 |     construction">Contact</a></li>
12 |     <li><a href="http://localhost:8080/comments.html">Comments</a
13 |     ></li>
14 | </ul>
15 |
16 | <c:if test="${filter == 'article'}">
17 |     <div id="submenu">
18 |         <ul class="submenu">
19 |             <li><a href="http://localhost:8080/addarticle.html">Add
20 |             </a></li>
21 |             <li><a href="http://localhost:8080/article.html">Show</
22 |             a></li>
23 |         </ul>
24 |     </div>
25 | </c:if>
26 |
27 | <c:if test="${filter == 'news'}">
28 |     <div id="submenu">
29 |         <ul class="submenu">
30 |         </ul>
31 |     </div>
32 | </c:if>
33 |
34 | <c:if test="${filter == 'home'}">
35 |     <div id="submenu">
36 |         <ul class="submenu">

```

```

38         </ul>
        </div>
40 </c:if>
        </div>
42 <div id="main">

```

Listing A.10: navigation.jsp

```

</div>
2         <div style="text-align:center;margin-top:100%"><
            span style="font-weight:bold">Computero
            2012</span></div>
4
        </body>
6 </html>

```

Listing A.11: footer.jsp

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "
    http://www.w3.org/TR/html4/loose.dtd">
2 <%@ page contentType="text/html; ISO-8859-1" language="java" %>
<html>
4 <head>
    <title>CompIT</title>
6    <!--<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>
8    <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>-->
10    <script src="http://code.jquery.com/jquery-1.7.2.min.js" type="
        text/javascript"></script>
    <link rel="stylesheet" href="computero.css" type="text/css"/>
12 </head>
<body>

```

Listing A.12: header.jsp

```

1 <%@ taglib prefix="form" uri="http://www.springframework.org/tags/
    form" %>
    <%@ taglib uri="http://java.sun.com/jstl/core_rt" prefix="c" %>
3 <%@ taglib uri="http://java.sun.com/jstl/fmt_rt" prefix="fmt" %>
    <%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn"
        %>
5 <%@ taglib uri="http://www.springframework.org/tags" prefix="spring
    " %>

```

Listing A.13: taglibs.jsp

```

2 <%—
    Created by IntelliJ IDEA.
    User: kristofferwanderydz

```

```

4 |   Date: 2012-maj-15
   |   Time: 14:27:32
6 |   To change this template use File | Settings | File Templates.
   | —%>
8 | <%@include file=" ../.. / taglibs.jsp" %>
   | <c:set var=" filter" value="home" />
10 | <%@include file=" ../.. / header.jsp" %>
   | <%@include file=" ../.. / navigation.jsp" %>
12 | Home
   | <spring:message scope="{foo}" />
14 | <%@include file=" ../.. / footer.jsp" %>

```

Listing A.14: home.jsp

```

2 | <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
   | <c:redirect url="http://localhost:8080/home.html" />

```

Listing A.15: index.jsp

A.2 ConFig

The conFig section contains all the conFiguration files from the web application.

```

2 | <?xml version="1.0" encoding="UTF-8"?>
   | <web-app id="WebApp.ID" version="2.4" xmlns="http://java.sun.com/
   |   xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
   |   instance" xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
   |   http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
4 |   <servlet>
   |     <servlet-name>dispatcher</servlet-name>
6 |     <servlet-class>org.springframework.web.servlet.
   |       DispatcherServlet</servlet-class>
   |     <load-on-startup>1</load-on-startup>
8 |   </servlet>
10 |   <servlet-mapping>
   |     <servlet-name>dispatcher</servlet-name>
12 |     <url-pattern>*.html</url-pattern>
   |   </servlet-mapping>
14 |   <listener>
16 |     <listener-class>org.springframework.web.context.
   |       ContextLoaderListener</listener-class>
   |   </listener>
18 | </web-app>

```

Listing A.16: web.xml

```

1 | <?xml version="1.0" encoding="UTF-8"?>
   | <beans xmlns="http://www.springframework.org/schema/beans"
3 |   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   |   xmlns:context="http://www.springframework.org/schema/context"

```

```

5  xmlns:mvc="http://www.springframework.org/schema/mvc"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
7      http://www.springframework.org/schema/beans/spring-beans
      -3.0.xsd
      http://www.springframework.org/schema/context
9      http://www.springframework.org/schema/context/spring-
      context-3.0.xsd
      http://www.springframework.org/schema/mvc
11     http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd"
   >
13  <!-- Activates various annotations to be detected in bean classes
      -->
   <context:annotation-config />
15
   <!-- Scans the classpath for annotated components that will be
      auto-registered as Spring beans.
17   For example @Controller and @Service. Make sure to set the
      correct base-package-->
   <context:component-scan base-package="se.compit" />
19
   <!-- Configures the annotation-driven Spring MVC Controller
      programming model.
21   Note that, with Spring 3.0, this tag works in Servlet MVC only!
      -->
   <mvc:annotation-driven />
23
   <!-- Load Hibernate related configuration -->
25  <import resource="hibernate-context.xml" />
27 </beans>

```

Listing A.17: applicationContext.xml

```

1  <?xml version="1.0" encoding="UTF-8"?>
   <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:p="http://www.springframework.org/schema/p"
5     xsi:schemaLocation="http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans
      -3.0.xsd">
7
   <!-- Declare a view resolver -->
9   <bean id="viewResolver" class="org.springframework.web.servlet.
      view.InternalResourceViewResolver"
      p:prefix="/WEB-INF/jsp/" p:suffix=".jsp" />
11
   </beans>

```

Listing A.18: dispatcher-servlet.xml

```

   <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xmlns:p="http://www.springframework.org/schema/p"
     xmlns:tx="http://www.springframework.org/schema/tx"

```

```

6      xmlns:context="http://www.springframework.org/schema/
      context"
      xsi:schemaLocation="
8      http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-3.0.
      xsd
10     http://www.springframework.org/schema/tx
      http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
12     http://www.springframework.org/schema/context
      http://www.springframework.org/schema/context/spring-context
      -3.0.xsd
14     ">
16 <context:property-placeholder location="/WEB-INF/spring.
      properties" />
18     <!-- Enable annotation style of managing transactions -->
      <tx:annotation-driven transaction-manager="transactionManager" />
20     <!-- Declare the Hibernate SessionFactory for retrieving
      Hibernate sessions -->
22     <!-- See http://static.springsource.org/spring/docs/3.0.x/
      javadoc-api/org.springframework.orm.hibernate3/annotation/
      AnnotationSessionFactoryBean.html -->
      <!-- See http://docs.jboss.org/hibernate/stable/core/api/index.
      html?org/hibernate/SessionFactory.html -->
24     <!-- See http://docs.jboss.org/hibernate/stable/core/api/index.
      html?org/hibernate/Session.html -->
      <bean id="sessionFactory" class="org.springframework.orm.
      hibernate3.annotation.AnnotationSessionFactoryBean"
26         p:dataSource-ref="dataSource"
          p:configLocation="{hibernate.config}"
28         p:packagesToScan="se.compit"/>
30     <!-- Declare a datasource -->
      <bean id="dataSource" class="org.springframework.jdbc.
      datasource.DriverManagerDataSource" >
32 <property name="driverClassName" value="org.postgresql.Driver"/>
      <property name="url" value="jdbc:postgresql://localhost:5432/
      postgres"/>
34 <property name="username" value="postgres"/>
      <property name="password" value="trapt123"/>
36 </bean>
38     <!-- Declare a transaction manager-->
      <bean id="transactionManager" class="org.springframework.orm.
      hibernate3.HibernateTransactionManager"
40         p:sessionFactory-ref="sessionFactory" />
42 </beans>

```

Listing A.19: hibernate-context.xml

```

<?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE hibernate-configuration PUBLIC
      "-//Hibernate/Hibernate Configuration DTD 3.0//EN"

```

```

4      "http://hibernate.sourceforge.net/hibernate-configuration-3.0.
      dtd">
6 <hibernate-configuration>
  <session-factory>
8    <!-- We're using MySQL database so the dialect needs to MySQL as
      well -->
      <property name="hibernate.dialect">org.hibernate.dialect.
      PostgreSQLDialect</property>
10    <!-- Enable this to see the SQL statements in the logs -->
      <property name="show_sql">>false</property>
12    <!-- This will drop our existing database and re-create a new
      one.
      Existing data will be deleted! -->
14    <!-- <property name="hbm2ddl.auto">create</property -->
      <!-- <property name="hibernate.jdbc.batch.size">0</property>
      -->
16  </session-factory>
</hibernate-configuration>

```

Listing A.20: hibernate.cfg.xml

```

1 org.apache.catalina.core.ContainerBase.[Catalina].level = INFO
  org.apache.catalina.core.ContainerBase.[Catalina].handlers = java.
  util.logging.ConsoleHandler

```

Listing A.21: logging.properties

```

# database properties
2 app.jdbc.driverClassName=org.postgresql.Driver
  app.jdbc.url=jdbc:postgresql://localhost:5432/postgres
4 app.jdbc.username=postgres
  app.jdbc.password=postgres
6
#hibernate properties
8 hibernate.config=/WEB-INF/hibernate.cfg.xml

```

Listing A.22: spring.properties

A.3 Source

Service

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

```

package se.compit.service;
2
import org.hibernate.Query;
4 import org.hibernate.Session;
import org.hibernate.SessionFactory;
6 import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
8 import se.compit.model.Administrator;

```

```
10 import javax.annotation.Resource;
11 import java.util.List;
12 import java.util.logging.Logger;
13
14 /**
15  * Created by IntelliJ IDEA.
16  * User: kristofferwanderydz
17  * Date: 2012-maj-15
18  * Time: 19:05:04
19  * To change this template use File | Settings | File Templates.
20  */
21 @Service("administratorService")
22 @Transactional
23 public class AdministratorService {
24
25     protected static Logger logger = Logger.getLogger("service");
26
27     @Resource(name = "sessionFactory")
28     private SessionFactory sessionFactory;
29
30     /**
31      * Retrieves all administrators
32      *
33      * @return a list of administrators
34      */
35     public List<Administrator> getAll() {
36
37         // Retrieve session from Hibernate
38         Session session = sessionFactory.getCurrentSession();
39
40         // Create a Hibernate query (HQL)
41         Query query = session.createQuery("FROM Administrator");
42
43         // Retrieve all
44         return query.list();
45     }
46
47     public Boolean authenticate(Administrator admin)
48     {
49         Boolean auth;
50
51         Session session = sessionFactory.getCurrentSession();
52
53         Query query = session.createSQLQuery("select admin,password
54         from administrator where admin='admin'-- and password
55         ='"+admin.getPassword()+"'");
56
57         if(query.list().size()==1)
58             auth = true;
59         else auth=false;
60
61         return auth;
62     }
63 }
```

```
64     }
65
66     public Boolean authenticateSafe(Administrator admin)
67     {
68         Boolean auth;
69
70         Session session = sessionFactory.getCurrentSession();
71
72         Query queryName = session.createSQLQuery("select admin from
73             administrator where admin='"+admin.getAdmin()+"'");
74
75         System.out.print(queryName.list().get(0));
76
77         Query queryPassword = session.createSQLQuery("select
78             password from administrator where admin='"+queryName.
79             list().get(0)+"'");
80
81         System.out.print(queryPassword.list().get(0));
82
83         if(queryPassword.list().get(0).equals(admin.getPassword()))
84             auth = true;
85         else auth=false;
86
87         return auth;
88     }
89
90     /**
91     * Retrieves a single administrator
92     */
93     public Administrator get(Integer id) {
94         // Retrieve session from Hibernate
95         Session session = sessionFactory.getCurrentSession();
96
97         // Retrieve existing administrator first
98         Administrator administrator = (Administrator) session.get(
99             Administrator.class, id);
100
101         return administrator;
102     }
103
104     /**
105     * Adds a new administrator
106     */
107     public void add(Administrator administrator) {
108
109         // Retrieve session from Hibernate
110         Session session = sessionFactory.getCurrentSession();
111
112         // Save
113         session.save(administrator);
114     }
115
116 }
```

```
118
    /**
120     * Deletes an existing administrator
    *
122     * @param id the id of the existing administrator
    */
124     public void delete(Integer id) {

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

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

134         // Delete
136         session.delete(administrator);
    }

138     /**
    * Edits an existing administrator
    */
140     public void editPass(Administrator administrator) {

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

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

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

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

Listing A.23: AdministratorService.java

```
package se.compit.service;

2
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;

8
import javax.annotation.Resource;
```

```
import java.util.List;
12 import java.util.logging.Logger;

14 /**
   * Created by IntelliJ IDEA.
16 * User: kristofferwanderydz
   * Date: 2012-maj-16
18 * Time: 01:41:16
   * To change this template use File | Settings | File Templates.
20 */
   @Service("articleService")
22 @Transactional
   public class ArticleService {

24

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

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

30

32     /**
       * Retrieves all articles
       *
34     * @return a list of articles
       */
36     public List<Article> getAll() {

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

40

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

44

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

48     }

50     /**
       * Retrieves a single article
       */
52     public Article get(Integer id) {

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

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

58         return article;

60     }

62     /**
       * Adds a new article
       */
64     public void add(Article article) {

66         // Retrieve session from Hibernate
```

```

68     Session session = sessionFactory.getCurrentSession();
69
70     // Save
71     session.save(article);
72 }
73
74 /**
75  * Deletes an existing article
76  *
77  * @param id the id of the existing article
78  */
79 public void delete(Integer id) {
80
81     // Retrieve session from Hibernate
82     Session session = sessionFactory.getCurrentSession();
83
84     // Retrieve existing article first
85     Article article = (Article) session.get(Article.class, id);
86
87     // Delete
88     session.delete(article);
89 }
90
91 /**
92  * Edits an existing article
93  */
94 public void editPass(Article article) {
95
96     // Retrieve session from Hibernate
97     Session session = sessionFactory.getCurrentSession();
98
99     // Retrieve existing article via id
100    Article existingArticle = (Article) session.get(Article.
101        class, article.getId());
102
103    // Assign updated values to this article
104    existingArticle.setCategory(article.getCategory());
105    existingArticle.setName(article.getName());
106
107    // Save updates
108    session.save(existingArticle);
109 }
110 }
111 }
112 }

```

Listing A.24: ArticleService.java

```

package se.compit.service;
2
import org.hibernate.Query;
4 import org.hibernate.Session;
import org.hibernate.SessionFactory;
6 import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
8 import se.compit.model.Comment;

```

```
10 import javax.annotation.Resource;
11 import java.util.List;
12 import java.util.logging.Logger;
13
14 /**
15  * Created by IntelliJ IDEA.
16  * User: kristofferwanderydz
17  * Date: 2012-maj-16
18  * Time: 01:41:16
19  * To change this template use File | Settings | File Templates.
20  */
21 @Service("commentService")
22 @Transactional
23 public class CommentService {
24
25
26     protected static Logger logger = Logger.getLogger("service");
27
28     @Resource(name = "sessionFactory")
29     private SessionFactory sessionFactory;
30
31     /**
32      * Retrieves all comments
33      *
34      * @return a list of comments
35      */
36     public List<Comment> getAll() {
37
38         // Retrieve session from Hibernate
39         Session session = sessionFactory.getCurrentSession();
40
41         // Create a Hibernate query (HQL)
42         Query query = session.createQuery("FROM Comment");
43
44         // Retrieve all
45         return query.list();
46     }
47
48     /**
49      * Retrieves a single comment
50      */
51     public Comment get(Integer id) {
52         // Retrieve session from Hibernate
53         Session session = sessionFactory.getCurrentSession();
54
55         // Retrieve existing comment first
56         Comment comment = (Comment) session.get(Comment.class, id);
57
58         return comment;
59     }
60
61     /**
62      * Adds a new comment
63      */
64     public void add(Comment comment) {
```

```
66         // Retrieve session from Hibernate
68         Session session = sessionFactory.getCurrentSession();
69
70         // Save
71         session.save(comment);
72     }
73
74     /**
75     * Deletes an existing comment
76     *
77     * @param id the id of the existing comment
78     */
79     public void delete(Integer id) {
80
81         // Retrieve session from Hibernate
82         Session session = sessionFactory.getCurrentSession();
83
84         // Retrieve existing comment first
85         Comment comment = (Comment) session.get(Comment.class, id);
86
87         // Delete
88         session.delete(comment);
89     }
90
91     /**
92     * Edits an existing comment
93     */
94     public void editPass(Comment comment) {
95
96         // Retrieve session from Hibernate
97         Session session = sessionFactory.getCurrentSession();
98
99         // Retrieve existing comment via id
100        Comment existingComment = (Comment) session.get(Comment.
101            class, comment.getId());
102
103        // Assign updated values to this comment
104
105
106
107
108        // Save updates
109        session.save(existingComment);
110    }
111 }
```

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.

```
1 package se.compit.model;
```

```
3 import javax.persistence.*;
import java.io.Serializable;
5
7 /**
8  * Created by IntelliJ IDEA.
9  * User: kristofferwanderydz
10 * Date: 2012-maj-15
11 * Time: 18:55:23
12 * To change this template use File | Settings | File Templates.
13 */
14 @Entity
15 @Table(name = "administrator")
16 public class Administrator implements Serializable
17 {
18     @Id
19     @Column(name = "id")
20     @GeneratedValue
21     private Integer id;
22
23     @Column(name = "admin")
24     private String admin;
25
26     @Column(name = "password")
27     private String password;
28
29     public Integer getId() {
30         return id;
31     }
32
33     public void setId(Integer id) {
34         this.id = id;
35     }
36
37     public String getAdmin() {
38         return admin;
39     }
40
41     public void setAdmin(String admin) {
42         this.admin = admin;
43     }
44
45     public String getPassword() {
46         return password;
47     }
48
49     public void setPassword(String password) {
50         this.password = password;
51     }
52 }
```

Listing A.26: Administrator.java

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

```
5
6
7 /**
8  * Created by IntelliJ IDEA.
9  * User: kristofferwanderydz
10 * Date: 2012-maj-16
11 * Time: 01:37:12
12 * To change this template use File | Settings | File Templates.
13 */
14 @Entity
15 @Table(name = "article")
16 public class Article implements Serializable
17 {
18     @Id
19     @Column(name = "id")
20     @GeneratedValue
21     private Integer id;
22
23     @Column(name = "category")
24     private String category;
25
26     @Column(name = "name")
27     private String name;
28
29     @Column(name = "price")
30     private String price;
31
32     public String getPrice() {
33         return price;
34     }
35
36     public void setPrice(String price) {
37         this.price = price;
38     }
39
40     public Integer getId() {
41         return id;
42     }
43
44     public void setId(Integer id) {
45         this.id = id;
46     }
47
48     public String getCategory() {
49         return category;
50     }
51
52     public void setCategory(String category) {
53         this.category = category;
54     }
55
56     public String getName() {
57         return name;
58     }
59
60     public void setName(String name) {
61         this.name = name;
62     }
63 }
```

```
63 }  
}
```

Listing A.27: Article.java

```
1 package se.compit.model;  
3  
4 import javax.persistence.*;  
5 import java.io.Serializable;  
7 /**  
8  * Created by IntelliJ IDEA.  
9  * User: kristofferwanderydz  
10 * Date: 2012-maj-16  
11 * Time: 17:02:02  
12 * To change this template use File | Settings | File Templates.  
13 */  
14 @Entity  
15 @Table(name = "comment")  
16 public class Comment implements Serializable {  
17     @Id  
18     @Column(name = "id")  
19     @GeneratedValue  
20     private Integer id;  
21  
22     @Column(name = "sender")  
23     private String sender;  
24  
25     @Column(name = "message")  
26     private String message;  
27  
28     public Integer getId() {  
29         return id;  
30     }  
31  
32     public void setId(Integer id) {  
33         this.id = id;  
34     }  
35  
36     public String getSender() {  
37         return sender;  
38     }  
39  
40     public void setSender(String sender) {  
41         this.sender = sender;  
42     }  
43  
44     public String getMessage() {  
45         return message;  
46     }  
47  
48     public void setMessage(String message) {  
49         this.message = message;  
50     }  
51 }  
}
```

Listing A.28: Comment.java

Controller

The controller is the internal part of the dynamical behavior (the functional jsps are the external A.1) as shown in Fig 3.1

```

2 package se.compit.controller;
3
4 import org.springframework.stereotype.Controller;
5 import org.springframework.ui.Model;
6 import org.springframework.web.bind.annotation.*;
7 import org.springframework.web.servlet.ModelAndView;
8 import org.springframework.web.util.WebUtils;
9 import se.compit.model.Administrator;
10 import se.compit.model.Article;
11 import se.compit.model.Comment;
12 import se.compit.service.AdministratorService;
13 import se.compit.service.ArticleService;
14 import se.compit.service.CommentService;
15
16 import javax.annotation.Resource;
17 import javax.servlet.http.HttpServletRequest;
18 import java.util.List;
19 import java.util.logging.Logger;
20
21 /**
22  * Created by IntelliJ IDEA.
23  * User: kristofferwanderydz
24  * Date: 2012-maj-15
25  * Time: 15:40:54
26  * To change this template use File | Settings | File Templates.
27  */
28 @Controller
29 @SessionAttributes("admin")
30 public class SuperController {
31
32     /* ModelAndView Injection.Spring MVC */
33     @RequestMapping("/universal")
34     public String look_what_much_time_i_save_on_codeing(
35         @RequestParam(required = true, value = "navi", defaultValue =
36             "home")String string)
37     {
38         System.out.print("universal");
39         return string;
40     }
41
42     /* END ModelAndView Injection.Spring MVC */
43
44     @ModelAttribute("loginAttribute")

```

```
46     public Administrator getLogInAttribute()
47     {
48         Administrator admin = new Administrator();
49
50         return admin;
51     }
52
53
54
55     protected static Logger logger = Logger.getLogger("controller")
56     ;
57
58     @Resource(name="administratorService")
59     private AdministratorService administratorService;
60
61     @Resource(name="articleService")
62     private ArticleService articleService;
63
64     @Resource(name="commentService")
65     private CommentService commentService;
66
67
68     @RequestMapping("/home")
69     public String homeView() {
70
71         // This will resolve to /WEB-INF/jsp/home.jsp
72         return "home";
73     }
74
75     @RequestMapping(method = RequestMethod.GET)
76     public String main()
77     {
78
79         return "home";
80     }
81
82
83     @RequestMapping(value = "/article",method = RequestMethod.GET)
84     public String articleView(Model model) {
85
86         List<Article> articles = articleService.getAll();
87
88         model.addAttribute("articles", articles);
89
90         return "article";
91     }
92
93
94
95     @RequestMapping(value = "/addarticle", method = RequestMethod.
96     POST)
97     public ModelAndView addArticle(@ModelAttribute("
98     articleAttribute") Article article)
99     {
100         articleService.add(article);
```

```
100
102     return new ModelAndView("home", "Message", "Product added !");
104 }
106 @RequestMapping(value = "/comments", method = RequestMethod.GET
107 )
108 public String viewComments(Model model)
109 {
110     List<Comment> comments = commentService.getAll();
111
112     model.addAttribute("comments", comments);
114     model.addAttribute("commentAttribute", new Comment());
116
117     return "comments";
118 }
120 @RequestMapping(value = "/comments", method = RequestMethod.
121     POST)
122 public ModelAndView addComment(@ModelAttribute("
123     commentAttribute") Comment comment)
124 {
125     commentService.add(comment);
126
127     return new ModelAndView("home", "Message", "Comment added !");
128 }
130 @RequestMapping(value = "/addarticle", method = RequestMethod.
131     GET)
132 public String addArticle(Model model)
133 {
134     model.addAttribute("articleAttribute", new Article());
135
136     return "articleadd";
137 }
138 @RequestMapping("/contact")
139 public ModelAndView contactView(@RequestParam(required=false,
140     value="message") String message) {
141
142     ModelAndView mav = new ModelAndView("contact");
143
144     mav.addObject("message", message);
145
146
147     return mav;
148 }
150 }
```

```
152  /* @RequestMapping("/contact")
153     public ModelAndView contactView() {
154
155         ModelAndView mav = new ModelAndView("contact");
156
157         mav.addObject("message", "Under construction");
158
159
160
161         return mav;
162     }
163     */
164     @RequestMapping("/news")
165     public ModelAndView newsView() {
166
167         return new ModelAndView("news");
168     }
169
170     @RequestMapping("/blog")
171     public ModelAndView blogView() {
172
173         return new ModelAndView("blog");
174     }
175
176     @RequestMapping(value = "/login", method = RequestMethod.POST)
177     public ModelAndView login(@ModelAttribute("loginAttribute")
178         Administrator admin, HttpServletRequest request) {
179
180         Boolean auth = administratorService.authenticateSafe(admin)
181             ;
182
183         if(auth) {
184
185             WebUtils.setSessionAttribute(request, "user", admin.
186                 getAdmin());
187
188             return new ModelAndView("home", "Message", "Authorized");
189         }
190         else
191             return new ModelAndView("home", "messageFail", "Forgot
192                 your password ?");
193     }
194
195 }
196
197
198
199
200
201 }
```

Listing A.29: SuperController.java

Appendix B

Online resources

This files 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-AGZQ&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=R0zdRqChXxg&feature=plcp>

Request Forgery

<http://www.youtube.com/watch?v=IqwpSEzLLxo&feature=plcp>

SQL Injection

<http://www.youtube.com/watch?v=ieS1CiTIqkw&feature=plcp>

B.2.2 Secure

Authentication and Session Management

<http://www.youtube.com/watch?v=0V1WaxjwNhE&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=0N0sWk0W31U&feature=plcp>

Request Forgery

<http://www.youtube.com/watch?v=IqwpSEzLLxo&feature=plcp>

SQL Injection

1. <http://www.youtube.com/watch?v=fiJJohyfKQM&feature=plcp>
2. <http://www.youtube.com/watch?v=pHHmHXF87pE&feature=plcp>

B.3 Report

B.4 Presentation

<http://prezi.com/gxx1-no8fkuc/web-application-security-in-java-environment/>