

University of West Bohemia

Faculty of Applied Sciences

Department of Computer Science and Engineering

# **MASTER'S THESIS**

Pilsen, 2013

Jan Froněk

University of West Bohemia  
Faculty of Applied Sciences  
Department of Computer Science and Engineering

## **Master's Thesis**

# **Security in EEG/ERP Portal**

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 14. května 2013

Jan Froněk, .....

# Acknowledgements

I would like to thank my supervisor, Ing. Petr Jěžek, Ph.D., for all his time he dedicated to me and for his guidance and help to complete this thesis.

I would also like to acknowledge my loyal friend Mates, the best dog in the world, who never fails to cheer me up. Most ideas used in this thesis came to me during our walks in the park.

And finally, I want to thank my parents for their support during my studies. This thesis could never be completed without them.

# Abstract

The main goal of this thesis is to investigate current security threats in field of web applications. Especially, in relation to EEG/ERP Portal of neuroinformatics group at the University of West Bohemia. First part of this thesis investigates and describes the most spread threats together with ways of their removal and prevention. The investigated risks are based on the "Top Ten" list published by The Open Web Application Project, a well known and respected group in field of internet security. Second part tests and identifies security weaknesses of EEG/ERP Portal while the third part suggests their removal and future prevention. Last part of this thesis introduces design and implementation of secured e-shop for EEG/ERP Portal. This store's security design is based on knowledge gained in previous parts. The security measures are implemented according to suggestions given in third part of this thesis. It practically demonstrates presented approach.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Neuroinformatic Research Group . . . . .	2
1.2	EEG/ERP Portal . . . . .	2
1.3	Motivation for this Thesis . . . . .	3
1.4	Technologies in EEG/ERP Portal . . . . .	4
1.4.1	Hibernate . . . . .	4
1.4.2	Spring . . . . .	4
1.4.3	Spring Security . . . . .	5
1.4.4	Spring Social . . . . .	5
1.4.5	Apache Wicket . . . . .	5
<b>2</b>	<b>Existing Security Standards</b>	<b>7</b>
2.1	Open Web Application Security Project . . . . .	7
2.2	Top Ten Security Risks . . . . .	7
<b>3</b>	<b>Security Risks</b>	<b>9</b>
3.1	Injection . . . . .	9
3.1.1	Preventing Injections . . . . .	10

3.2	Cross-Site Scripting . . . . .	11
3.2.1	Preventing Cross-Site Scripting . . . . .	11
3.3	Broken Authentication and Session Management . . . . .	12
3.4	Insecure Direct Object References . . . . .	13
3.5	Cross-Site Request Forgery . . . . .	13
3.5.1	Preventing Cross-Site Request Forgery . . . . .	14
3.6	Security Misconfiguration . . . . .	17
3.7	Insecure Cryptographic Storage . . . . .	18
3.8	Failure to Restrict URL Access . . . . .	19
3.9	Insufficient Transport Layer Protection . . . . .	20
3.10	Unvalidated Redirects and Forwards . . . . .	20
3.11	Clickjacking . . . . .	21
3.11.1	Preventing Clickjacking . . . . .	21
<b>4</b>	<b>Current State of Security in EEG/ERP Portal</b>	<b>24</b>
4.1	Injection Vulnerability . . . . .	24
4.2	Cross-Site Scripting Vulnerability . . . . .	25
4.3	Authentication and Session Management . . . . .	25
4.4	Securing Direct Object References . . . . .	26
4.5	Cross-Site Request Forgery Vulnerability . . . . .	26
4.6	Security Configuration . . . . .	28
4.7	Cryptographic Storage . . . . .	28
4.8	Restricting URL Access . . . . .	28
4.9	Transport Layer Protection . . . . .	29

---

4.10	Validation of Forwards and Redirects . . . . .	29
4.11	Clickjacking Vulnerability . . . . .	29
<b>5</b>	<b>Security Improvements in EEG/ERP Portal</b>	<b>30</b>
5.1	Preventing Injections . . . . .	30
5.2	Preventing Cross-Site Scripting . . . . .	31
5.3	Authentication and Session Management . . . . .	32
5.4	Cross-Site Request Forgery Protection . . . . .	32
5.4.1	Enabling CryptoMapper . . . . .	33
5.5	Data Storage Encryption . . . . .	35
5.5.1	Existing Solutions . . . . .	35
5.5.2	Oracle Transparent Data Encryption . . . . .	36
5.5.3	Setting-up Transparent Data Encryption . . . . .	37
5.5.4	Choosing Data for Encryption . . . . .	39
5.5.5	Impact of Encryption on Database Response Time . . . . .	41
5.6	Clickjacking Prevention . . . . .	43
<b>6</b>	<b>Building Secured E-Shop</b>	<b>46</b>
6.1	Current State of EEG/ERP Portal . . . . .	46
6.2	Shopping Cart Functions . . . . .	47
6.3	Payment Options . . . . .	47
6.4	Shopping Cart Design . . . . .	48
6.5	Example Purchase Scenario . . . . .	49
6.6	Processing a Payment . . . . .	55
6.7	Securing the Purchase Process . . . . .	59



6.7.1	Total Price Protection . . . . .	60
6.7.2	Two-step Confirmation . . . . .	60
6.7.3	Encryption . . . . .	62
6.8	Description of Implemented E-Shop . . . . .	62
<b>7</b>	<b>Conclusion</b>	<b>64</b>

# Chapter 1

## Introduction

Providing sufficient security measures is part of application development, especially in field of web applications. The main goal of this thesis is to investigate and improve security measures in EEG/ERP Portal developed by neuroinformatics research group at the University of West Bohemia. The EEG/ERP Portal contains private data collected from experiments performed by the research group. Therefore all steps necessary to secure such private information should be taken as required by Czech law.

Following this goal, Chapter 3 of this thesis investigates and describes the current most common threats in field of web applications. Each such threat is described and a general solution of its removal and prevention is given. The investigated security risks were chosen according to list of top 10 web application security issues published by The Open Web Application Security Project, a well known and respected authority in field of web security.

In Chapter 4, the EEG/ERP Portal is tested for presence of aforementioned risks and weaknesses followed by solutions suggested in Chapter 5. When suggesting solutions to improve level of security in EEG/ERP Portal and eliminate any weakness found, the possibilities offered by already used frameworks are preferred. In addition to securing the application layer of EEG/ERP Portal, the ways of securing physical data storage, containing

collected private information, are discussed and implemented.

In Chapter 6, a prototype of secured online store is introduced, designed and implemented. The security measures taken in this store are based on knowledge gained in previous parts of this thesis. The developed store allows EEG/ERP Portal visitors to purchase an experimental data. To provide a secure way of online payment, the PayPal's Express Checkout service is used.

This thesis provides the EEG/ERP Portal application with security measures necessary to prevent the most spread and severe threats to avoid any private information exposure or theft.

## 1.1 Neuroinformatic Research Group

At the University of West Bohemia there is a research group performing experiments in a field of neuroinformatics. Its research contains a lot of experiments with human subjects, such as hospital patients, children or university students. These experiments usually measure subject's EEG waves and stores them for further processing.

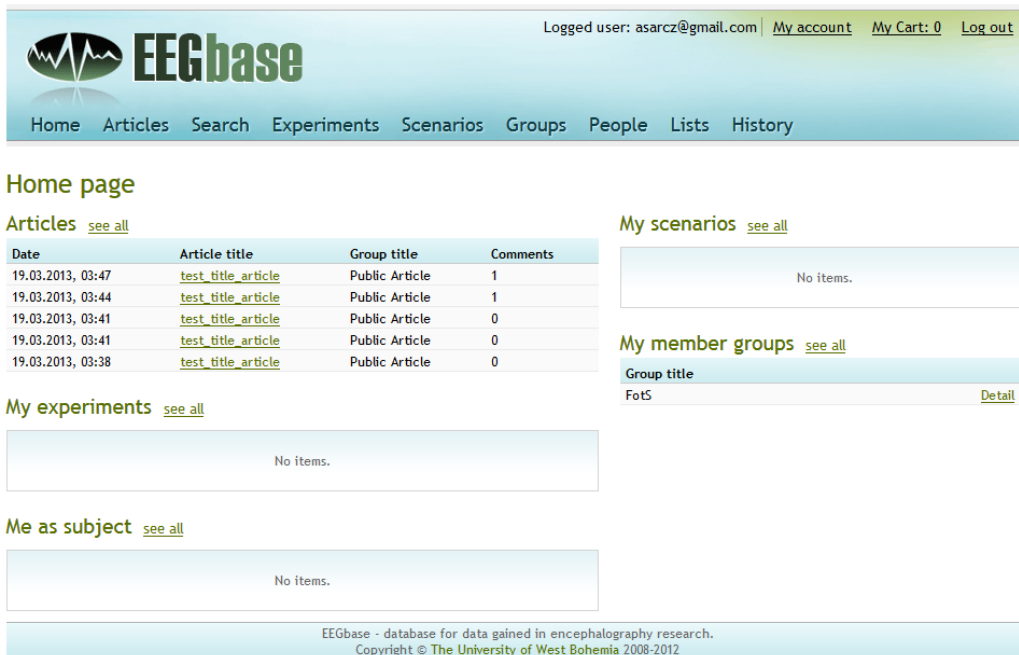
## 1.2 EEG/ERP Portal

To easily share experimental data a project of web portal<sup>1</sup> is being developed. This portal application allows members to upload, store and share experimental data and scenarios, publish articles or comments and share them on LinkedIn. Users are organized into smaller groups based on current projects they work on. By doing so, they can easily restrict published information to be shared only within such group.

You can see an example of EEG/ERP Portal application in Figure 1.1.

---

<sup>1</sup><http://eegdatabase.kiv.zcu.cz>



EEGbase

Logged user: asarcz@gmail.com | [My account](#) | [My Cart: 0](#) | [Log out](#)

Home Articles Search Experiments Scenarios Groups People Lists History

### Home page

**Articles** [see all](#)

Date	Article title	Group title	Comments
19.03.2013, 03:47	<a href="#">test_title_article</a>	Public Article	1
19.03.2013, 03:44	<a href="#">test_title_article</a>	Public Article	1
19.03.2013, 03:41	<a href="#">test_title_article</a>	Public Article	0
19.03.2013, 03:41	<a href="#">test_title_article</a>	Public Article	0
19.03.2013, 03:38	<a href="#">test_title_article</a>	Public Article	0

**My scenarios** [see all](#)

No items.

**My member groups** [see all](#)

Group title	Detail
Fot5	<a href="#">Detail</a>

**My experiments** [see all](#)

No items.

**Me as subject** [see all](#)

No items.

EEGbase - database for data gained in encephalography research.  
Copyright © The University of West Bohemia 2008-2012

Figure 1.1: EEG/ERP Portal example.

## 1.3 Motivation for this Thesis

As a result of numerous experiments, EEG/ERP Portal contains private information provided by participating subjects, such as their name, date of birth or contact information. According to Czech law [1], everyone who collects, processes or stores such information is obliged to undertake any steps necessary to secure such data and restrict access to them. This is one reason to investigate and evaluate current security measures taken in EEG/ERP Portal, improve them and eliminate any security risks found. The referred law [1] also mentions a duty to document all security measures taken. Therefore, this thesis is intended to serve this purpose as well.

Leak of private information could harm and discredit the research group's credibility. This could be a serious issue considering ongoing collaboration with INCF<sup>2</sup> organization and possibility of joining their DataSpace project<sup>3</sup>.

<sup>2</sup><http://www.incf.org>

<sup>3</sup><http://www.incf.org/resources/data-space>

This project offers a possibility to share data collected from experiments among INCF member organisations worldwide. The responsibility to secure shared data remains on each participating organization. The research group should therefore secure its data before considering joining such project. This doesn't apply only to the database containing data to be shared but to the whole application of EEG/ERP Portal which accesses this database as well. Any security measures suggested in this thesis shall eliminate or at least minimize a possibility of compromising stored data or its theft.

## 1.4 Technologies in EEG/ERP Portal

The implementation of EEG/ERP Portal is programmed in Java language and uses many technologies commonly used in development of contemporary Enterprise applications.

### 1.4.1 Hibernate

Application's data are stored in Oracle database. Since the EEG/ERP Portal is object oriented based, there is a need to map data stored in relational database to data objects. This is done using Hibernate<sup>4</sup> ORM (Object Relational Mapping) technology. Therefore, the database queries are performed using HQL (Hibernate Query Language), it is an object oriented modification of SQL (Structured Query Language).

### 1.4.2 Spring

Spring<sup>5</sup> is an open source framework for Java Enterprise application development. It offers configuration of components, managing life cycle of application's objects, object creation and wiring dependent objects together.

---

<sup>4</sup><http://www.hibernate.org>

<sup>5</sup><http://www.springsource.org>

### 1.4.3 Spring Security

Spring Security<sup>6</sup> is an open source framework providing means to secure developed application. It offers authentication and authorization tools for identifying users and managing privileges by set of roles - e.g. admin, user, etc.

### 1.4.4 Spring Social

The EEG/ERP Portal provides possibility of user authentication using user's Facebook or LinkedIn credentials. This feature is implemented using corresponding version of Spring Social<sup>7</sup> framework. Both frameworks, for Facebook and LinkedIn, use OAuth<sup>8</sup> (Open Standard for Authorization) protocol to exchange confirmation token between social network's login service and developed application.

### 1.4.5 Apache Wicket

Since early 2013, the EEG/ERP Portal is developed using Apache Wicket<sup>9</sup> which mostly replaces formerly used Spring MVC framework. Apache Wicket is a component-based web application framework that varies a bit from the standard MVC frameworks. Standard MVC framework uses controller objects and maps incoming requests to particular controller's methods with desired actions. In contrast, Wicket application is based on a tree of components and listeners serve the incoming HTTP requests. Apache Wicket is often compared to GUI framework Swing for its use of listeners. In both cases, the components are pure Java objects. While Swing is used for desktop applications development, Apache Wicket is a web application framework. Both of them uses similar principles. Therefore, it is considered easy to adopt

---

<sup>6</sup><http://www.springsource.org/spring-security>

<sup>7</sup><http://www.springsource.org/spring-social>

<sup>8</sup><http://oauth.net>

<sup>9</sup><http://wicket.apache.org>

usage of Wicket once you know the usage of Swing. Apache Wicket and its listeners also offer an advantage of eliminating extensive usage of XML configurations required by other similar frameworks (e.g. aforementioned Spring Web MVC).

For further details on Apache Wicket framework and security measures it offers see sections 4.5 and 5.4 on pages 26 and 32.

# Chapter 2

## Existing Security Standards

### 2.1 Open Web Application Security Project

The Open Web Application Security Project, mostly known by its abbreviation OWASP, is a non-profit organisation focused on providing security advice and standards for web application development. This community group realizes numerous projects such as Application Security Verification Standard or Application Development guide.

### 2.2 Top Ten Security Risks

In 2010, OWASP published a list of Top 10 most critical security risks [2] along with recommendations about their removal and future avoidance. Even though this list is almost three years old, it is still a good start when trying to develop a secure application. The most critical threats it describes still exist nevertheless. Therefore, it is still beneficial to take them into consideration. Beside the listed threats, this work includes description and solution for another security threat, the Clickjacking, that was not listed in aforementioned list. Its occurrence has rapidly risen and its security risk is severe and therefore needs to be looked at and solved.



The OWASP Top Ten List is recommended to follow and implement by U.S. Trade Commission. It was adopted as a part of Payment Card Industry standard and is used by many companies like HP, IBM, Citibank, Microsoft or Oracle.

The published list [2] consists of these risks:

1. Injection
2. Cross-Site Scripting (XSS)
3. Broken Authentication and Session Management
4. Insecure Direct Object References
5. Cross-Site Request Forgery (CSRF)
6. Security Misconfiguration
7. Insecure Cryptographic Storage
8. Failure to Restrict URL Access
9. Insufficient Transport Layer Protection
10. Unvalidated Redirects and Forwards

Not every one of these items describes an actual threat or particular attack technique. Some of them are about general principles required to provide sufficient security level.

OWASP Top ten Lists were published with a three year difference, in 2004, 2007 and 2010. A list of candidates for new version was introduced in March, 2013 and final version is due May, 2013. In comparison, the 2010 and 2013 versions only differ in minor aspects. Several risks changed positions, "Security Misconfiguration" was divided into two risks while "Insecure cryptographic storage" and "Insufficient Transport Layer" were merged together. Full list of 2013 candidates can be found at [3]. Considering the 2013 list is not final yet and the minor changes compared to 2010 version, this thesis elaborates with the 2010 version.

# Chapter 3

## Security Risks

This chapter goes through aforementioned threats, describes them and suggests a general solutions. Note that another threat - Clickjacking (see section 3.11) was added besides the OWASP Top ten list threats.

### 3.1 Injection

The application or system can be compromised with injection when it is possible to insert and execute unsecured code or command by submitting user input. It can be either injecting command to interpreter like PHP or query injection when user input is used as a parameter in database query. These exploits can cause possible loss of data, data corruption or personal data can be compromised or stolen. The business impact of this exploit is considered severe because of possible loss or theft of stored data.

An instance of injection attack [4, 5] can be demonstrated on an application which uses URL

`.../show_article&id=x`

to display article identified by id. This id, when submitted, is used as a parameter to query the database. An attacker can modify the URL to

`.../show_article&id=' or '1'='1`

to see whether the application is vulnerable to injection [6] or not.

If application builds SQL queries by simply appending parameters from requests as in listing 3.1 its weakness can be exploited to insert and execute any SQL command, for instance deleting all users data by modifying URL to  
.../show\_article&id=;DELETE \* FROM users.

Listing 3.1: Unprotected Query.

```
1 String query = "SELECT * FROM articles WHERE id='" +  
2 request.getParameter("id") + "'";
```

### 3.1.1 Preventing Injections

Injections can be prevented by escaping any special character such as ; ', and /. As regards SQL query, injections can be eliminated by using so called Prepared Statements. Prepared Statement eliminates any need to escape special characters manually and input is used only as parameter for prepared query. This makes it impossible to sneak in and execute another command within created query.

See an example of Prepared Statement in listing 3.2.

Listing 3.2: Protected Query.

```
1 String query = "SELECT * FROM articles where id=?";  
2 PreparedStatement stmt =  
3     connection.prepareStatement(query);  
4 stmt.setInt(1, requested_id);
```

## 3.2 Cross-Site Scripting

This attack is done by including a malicious script into user input which is then stored in the application, for instance into a text reply in web forum. Whenever any user views attacked page, the web browser interpreter is tricked into executing included script leading into modifying either page's appearance or including unwanted `<script>` tags. These scripts can either steal user's credentials, include unwanted advertisements or redirect the user to some malicious web page. For example a post in simple web forum is described by tags `<div><p>Text of post</p></div>`.

If text of the post isn't properly escaped, an attacker might easily submit a post containing something like:

Text of post `<script>document.location=malicious_URL</script>`.

The user can be totally unaware he/she became a victim of an Cross-Site Scripting attack. XSS scripts can simply make user's browser to send user's cookie file to attacker's web page without raising any suspicion. Such attack results into compromising session id and complete account loss with all its impacts. For instance, compromising user's account in some online store can result in losing money in unwanted purchases.

### 3.2.1 Preventing Cross-Site Scripting

The danger of Cross-Site Scripting [7, 8] can be partially avoided by making sure any user input is properly escaped when sent back to the browser and possibly escaped before storing in database. Still, this solution should be supported by proper defence of user's cookie by making them HTTP only. HTTP only cookies cannot be viewed by client-side scripts. A cookie can be set to HTTP only by adding a flag into its HTTP header. In Java EE applications this is usually done by modifying `WEB-INF/web.xml` configuration file with parameters shown in listing 3.3.

Listing 3.3: Session-config example.

```
1 <session-config>
2     <session-timeout>
3         60
4     </session-timeout>
5     <cookie-config>
6         <http-only>true</http-only>
7     </cookie-config>
8 </session-config>
```

### 3.3 Broken Authentication and Session Management

This topic depicts common mistakes in security setting that can result in account theft or either password or session ID exposure. Such mistakes allow attacker to impersonate other user and perform any action under his/her identity in order to abuse his/her privileges.

There is no method that would assure 100% safety against this kind of exploit but a general set of rules and recommendations to make it less likely to happen exists.

- Session ID should be protected by not exposing it as a part of URL.
- Passwords shall be created as a combination of lower and upper case letters and digits to be considered strong.
- Passwords should be also stored in encrypted form.
- Possibility that an attacker could guess a valid credentials should be limited by allowing only several attempts to login.
- All credentials shall be transferred through protected communication (e.g. via SSL).

- User shall remain authenticated only for a limited time by setting session time-out.

## 3.4 Insecure Direct Object References

This security weakness occurs when application uses objects IDs as parameters. For instance, article id is used as request parameter without following test whether the user is authorized to view requested article or not. The attacker can exploit this flaw by simply modifying URL parameter and access an article or a resource with restricted admission. For instance, URL `.../show_article&id=x` where x as originally viewed article might be altered to refer to another article the user wasn't intended to be allowed to view.

To avoid this weakness all direct references need to be validated for authorized access or direct references shall be replaced by indirect ones wherever it is possible. This is rather general recommendation since it heavily relies on used technologies and might differ with used programming languages.

## 3.5 Cross-Site Request Forgery

This attack technique, also known by its abbreviation CSRF [9], relies on forging false HTTP requests and tricking users into submitting them. Doing so, user unwillingly performs some action as though he/she performed it himself knowingly. When user was previously logged in, an vulnerable application can't distinguish whether the action was really requested by the user himself. These false links or image tags usually refer to some actions user needs to be authorized for, such as thread creation or deletion, subscription, etc.

### 3.5.1 Preventing Cross-Site Request Forgery

To be protected against CSRF users should follow these rules [9]:

- Log out before leaving the web site. Since CSRF attack requires the user to be logged in, logging out before leaving a web site eliminates the risk of becoming a victim of CSRF attack.
- Avoid using the "remember me" options when logging in and do not let your browser to store usernames and passwords.
- Use different browser for internet browsing and for accessing sites of high importance, such as internet banking.

However, users prefer higher comfort in exchange for lowering their security [10] and usually expect the application takes care of all security measures without their participation.

To prevent this flaw within an application it is necessary to exchange some secret information other than user's id, credentials or session id to ensure request's validity, for instance when a form is submitted. This is usually done by adding invisible secret field into <form> element which contains some secret key. This key is then, upon request reception, compared with generated key the form was created with. There are basically three possible ways to perform such exchange and comparison. The key can be either stored in a cookie file, session object or database. All possibilities are vulnerable in some ways. Cookie needs to be protected against client side access (see HTTP Only cookies in section 3.2.1). In case cookies are disabled the solution doesn't work at all or the whole form cannot be accepted and processed. The other approach, storing generated key into session object, requires secure session handling so its id isn't exposed and therefore vulnerable to exploiting. Some solutions suggest storing generated keys into database but this possibility is rather slow compared to those mentioned previously. Plus, it also makes managing keys more difficult when user uses multiple tabs in his browser.

General solution for keys stored in a session object in Java applications is shown in listings 3.4, 3.5 and 3.6. An incoming request goes through a filter where the secret key is generated and stored into a session object. Depending on used frameworks, secret token can be generated inside standard HTTP doFilter method, inside controller code or done by handlerInterceptor. The example in listings 3.4 and 3.6 explains principles of Cross-Site Request Forgery prevention usage on doFilter method but the code it contains would stay nearly unchanged for other implementations (controller, handlerInterceptor) as well.

Listing 3.4: CSRF Protection - creating token [11].

```
1 public void doFilter(Request request, Response
   response, FilterChain chain){
2     Cache<String, Boolean> csrfPreventionSaltCache =
3     (Cache<String, Boolean>)httpReq.getSession()
4     .getAttribute("csrfPreventionSaltCache");
5
6     if(csrfPreventionSaltCache == null){
7         csrfPreventionSaltCache = CacheBuilder
8         .newBuilder()
9         .maximumSize(5000)
10        .expireAfterWrite(20, TimeUnit.MINUTES)
11        .build();
12        request.getSession().setAttribute(
13            "csrfPreventionSaltCache",
14            csrfPreventionSaltCache);
15    }
16    String salt = RandomStringUtils.random(20, 0, 0,
17        true, true, null, new SecureRandom());
18    csrfPreventionSaltCache.put(salt, Boolean.TRUE);
19    request.setAttribute("csrfPreventionSalt", salt);
20    chain.doFilter(request, response);
}
```



Most available solutions have a limitation that they generate and include only one secret key per user. Such limitation is restricting users to only open and use the application in only one tab. If user opened the application in another tab, the secret key would be overwritten resulting in all but last opened form being refused upon submission. Therefore, described solution operates with a cache object, actually a Map<sup>1</sup>, allowing multiple keys to be stored in it. In addition, cache used in listing 3.4 adds an attribute of time-out after which is the stored key considered invalid. The aforementioned cache uses Guava CacheBuilder<sup>2</sup> implementation but generally any implementation of a Map could be used.

After the secret key is generated, the request is passed to a page where a form is created. The Secret key is then stored in a hidden field of a form as shown in listing 3.5.

Listing 3.5: CSRF Protection - hidden field in form [11].

```
1 <form action="/requiredAction" method="post">
2   <input type="hidden" name="csrfPreventionSalt"
3     value="<c:out value='${csrfPreventionSalt}'/>
4   "/>
5 </form>
```

When a form is filled in and submitted back to the server, the incoming request is passed to a filter where the received key is compared to the generated one. If the incoming secret value matches the one stored in session cache, the request is passed for further processing. Otherwise, the request is refused to be served. In such case, some kind of alert might be raised or CSRF attempt logged into sever log file.

An example of incoming form validation is shown in listing 3.6. If a CSRF attack is detected the request is not passed to another filter or servlet. As a result the attack is prevented from performing any action.

---

<sup>1</sup>Abstract Data Type organising stored data into pairs of keys and values

<sup>2</sup><http://docs.guava-libraries.googlecode.com>

Listing 3.6: CSRF Protection - receiving token [11].

```
1 public void doFilter(Request request, Response
  response, FilterChain chain){
2     String salt = (String)request.getParameter(
3         "csrfPreventionSalt");
4     Cache<String, Boolean> csrfPreventionSaltCache =
5         (Cache<String, Boolean>)request.getSession()
6         .getAttribute("csrfPreventionSaltCache");
7
8     if(csrfPreventionSaltCache != null
9         && salt != null
10        && csrfPreventionSaltCache
11           .getIfPresent(salt) != null){
12        chain.doFilter(request, response);
13    }
14    else{
15        //CRF occurred
16        //Application should log a CSRF detection or
17        raise any other alert
18    }
```

## 3.6 Security Misconfiguration

The Security Misconfiguration on OWASP's list of threats describes common mistakes in managing privileges and admissions to files or objects. It includes flaws not only in application design but in used frameworks as well. Therefore, it is necessary to keep all frameworks, libraries up to date to ensure an attacker cannot abuse any known weakness of an outdated part of application. The danger of using outdated libraries and frameworks should not be treated lightly. Anytime a new version of library is released, its patch

notes contain information about fixed issues and therefore any flaw of previous version become publicly known and documented. Especially, in case of an open source solution where anyone can view the source code, knowing both, issue and a way it was fixed, can provide a solid amount of knowledge necessary to construct an attack scenario.

OWASP gives another piece of advice about Security misconfiguration. It is to give as few information as necessary when informing user about application's error or when denying access to any source. This includes disabling stack trace to be displayed to user upon error occurrence. Basically, any information should be considered private and any server details, such as class names or file locations, should be kept secret.

### 3.7 Insecure Cryptographic Storage

Protecting application's data shall be a main goal of any project or business that somehow collects information about users. The value of customer's personal details and contact information is priceless in any business. Therefore, losing such data or violating its safety could have potentially lethal impact on any business.

For instance, if a company loses all or even just a part of its data it may delay or even prevent their deals to be fulfilled. Any lost data need to be replaced or recollected and any inconvenience could affect customer's favour. Therefore, the potential financial loss comes from two sources - additional expenses on restoring business data and a loss of income as a result of loss of customers. In addition, if an information about a company being a victim of private data theft becomes public, its reputation might be shattered up to a point of losing more customers and losing its credibility.

To avoid or at least minimize any risk of losing or compromising business data every important data should be encrypted when stored into database. However, this security measure should always be complemented with physical

data storage protection as well as with secure transport channel.

The possible ways of encrypting a data storage are highly dependent on software and hardware parts the data storage consists of. Some database solutions offer additional security features to be purchased such as Oracle Transparent Data Encryption (see section 5.5 on page 35).

### 3.8 Failure to Restrict URL Access

Unauthorized access liability is caused by not verifying the user's privilege to access any page after requesting it's URL. Attacker may simply overwrite requested page name if he/she either knows or can guess it. For instance an attacker could change the `.../home` URL to `.../admin_settings` to get access a page he/she is not authorized to view.

This can be prevented by validating users privileges upon entering requested page by assigning several different roles with different privileges - e.g. admin, registered user, guest, etc. User role management can be handled by security frameworks, such as Spring Security which is also used in EEG/ERP Portal.

A typical example of vulnerable application are sites that perform authorization only when creating links and other content for displayed web page. In such case, a registered user is viewing menus with different options than site administrator but when some action is requested no authorization is performed. Therefore, an attacker can try to guess names and parameters for restricted pages and operations. In such case, an attacker is allowed to view such pages and perform such actions without any restrictions as though he/she was authorized user or administrator himself. For instance, enter and use URL `.../admin_settings` and its features.

## 3.9 Insufficient Transport Layer Protection

The Insufficient Transport Layer Protection risk on OWASP's list is about unprotected exchange of information between a user and an application server. Even though the application might use or implement multiple security measures, the data are exposed when transferred through a communication channel. For instance, a user's cookie might possibly be stolen when accessing a page via wi-fi connection.

To prevent any private data exposure, application shall communicate via secured channel, for instance using the SSL with a valid certificate.

## 3.10 Unvalidated Redirects and Forwards

If application handles forwards and redirects by specifying destination as a request parameter this weakness can be exploited into redirecting user to malicious pages by links that appear as links of trusted web page.

Recommended solution is not to use redirects and forwards. Application's security is strongly dependent on its design. An unsecured redirects and forwards should be eliminated from the very beginning. When trying to improve security of already existing application it is not always possible to totally eliminate this flaw. If there is a need to keep redirects and forwards within application's logic, these shall avoid setting destination page as a parameter. If parameters cannot be avoided then such parameters always need to be validated to refer only to such pages that are valid and requesting user is authorized to view them.

## 3.11 Clickjacking

Clickjacking is a very powerful technique which was firstly used in 2008. Attacked facebook users were lured to give "likes" to chosen pages unwillingly. Since abusing this weakness is very easy, the Clickjacking technique has spread very quickly. This technique is performed by including exploitable web page into malicious page in an iframe tag. This iframe is then set to 100% opacity via CSS so visitors cannot see it unless they inspect the HTML code. Since most users do not possess the knowledge of HTML there is no way they could suspect their accounts or identities being abused.

Visitors of a such pages are forced to click somewhere on the page, for instance on some Next button or an image they want to view. But these elements are positioned exactly under the see-through iframe resulting in user clicking onto some submit button, link or "Like" button. These iframes can be either placed on some fixed position or can be always moving under user's cursor using JavaScript. If the user was logged in the exploitable application and his/her browser still contains this information, clicking onto such elements has same results as if the user clicked on them willingly.

Potential risk of this exploit is severe. It can lead from simple "Like" exploit into posting some advertisements, illegal pictures or making unwanted purchases.

### 3.11.1 Preventing Clickjacking

There are two possible ways to deal with Clickjacking. First approach, used in the past, suggests disabling the `<body>` element from being rendered and re-enabling it via JavaScript under condition it is not included in other page or element (see listing 3.7). The web page contains a `<style>` element with assigned ID. This element prevents the web page from being rendered and needs to be removed when the page is not included in an `<iframe>` element. This is done using JavaScript. The script either removes the afore-

mentioned `<style>` element using its ID or forces the browser to render current frame's URL by setting `top.location = self.location`. This technique is called "frame-breaker".

This solution is currently recommended only for legacy browsers that does not support the second approach introduced in this section.

Listing 3.7: Clickjacking Protection - variant 1 [12].

```
1 <style id="antiClickjack">
2   body{display:none !important;}
3 </style>
4
5 <script type="text/javascript">
6   if(self === top){
7     var antiClickjack = document.getElementById(
8       "antiClickjack");
9     antiClickjack.parentNode.removeChild(
10      antiClickjack);
11   }
12   else{
13     top.location = self.location;
14   }
15 </script>
```

Second approach to defend against Clickjacking is setting so called X-FRAME-OPTIONS parameter into page's HTTP header. This solution was introduced in 2009 and is currently supported by all contemporary browsers. By specifying this option to values DENY or SAMEORIGIN page will not be rendered when included in any iframe tag at all or in any iframe outside its origin site.

An example of setting X-FRAME-OPTIONS parameter in Wicket application is shown listing 3.8. Note, that this example is platform dependent and therefore may differ for other programming languages and frameworks.

Listing 3.8: X-FRAME-OPTIONS settings.

```
1 @Override
2 protected void setHeaders(WebResponse response) {
3     super.setHeaders(response);
4     response.setHeader("X-Frame-Options", "deny");
5 }
```



# Chapter 4

## Current State of Security in EEG/ERP Portal

After a theoretical base of possible attacks and threats this chapter investigates the level of security in EEG/ERP Portal. It describes performed tests and their results. Following actions taken in order to improve overall security are introduced in chapter 5 on page 30.

### 4.1 Injection Vulnerability

Since EEG/ERP Portal is developed in Java language and does not use any direct code input to interpreter, it is immune against any Code Injections in general.

Application's data are stored in Oracle database and accessed using the Hibernate framework. Special kind of SQL injections for Hibernate framework is called HQL injection. Injection Prevention in EEG/ERP Portal is ensured by parameter binding, a technique described in section 3.1.1 on page 10.

## 4.2 Cross-Site Scripting Vulnerability

In current implementation, application is secured against Cross-Site Scripting attacks. This is ensured by Apache Wicket framework, which escapes any output by default. Such behaviour needs to be explicitly disabled [13] every time the applications needs to skip the escaping process by declaring `.setEscapeModelStrings(false)` for particular label, panel, etc.

However, user's cookie files were not protected against script admission and therefore were vulnerable to such attack. This issue was resolved as described in section 5.2 on page 31.

## 4.3 Authentication and Session Management

The level of security in relation to Authentication and Session Management was revised according to recommendations given in section 3.3 on page 12.

Currently, EEG/ERP Portal uses email address and password as login credentials. The chosen password needs to have at least six alpha-numerical characters, but there are no other requirements - e.g. how many numbers, upper or lower case letters it must contain. Passwords in database are hashed, not stored as a plain text.

In a case that a user needs to change his/her password, the original one is required. This provides sufficient protection in situation when user's cookie was stolen and attacker might want to overtake user's account by changing its credential. The "forgotten password" function generates new password which is then sent to given email address. If an attacker knew user's email address, used in the register process, and wanted to steal his account using the "forgotten password" function, the attacker would have to gain access into user's email account first.

The amount of times user can attempt to log into EEG/ERP Portal is not anyhow limited or restricted. The reasons on keeping such state are discussed in section 5.3 on page 32.

The session ID is not exposed as a part of URL and credentials are transferred via SSL protected channel. However, no time period after which the session is invalidated was set.

## 4.4 Securing Direct Object References

The Portal application pages were tested for presence of an URL parameter that could be possibly exploited to gain access to a restricted resource. The tested pages and their results are shown in Table 4.1 on page 27.

All pages were proved secure. In many cases, the content is generated and displayed based on user's role, so even if a resource is accessed by parameter value modification, a view matching user's role is provided. In current state, no actions were needed to be taken.

## 4.5 Cross-Site Request Forgery Vulnerability

Before using the Apache Wicket framework, the EEG/ERP Portal did not use any security measures against Cross-Site Request Forgery and was therefore vulnerable to such technique. The Apache Wicket framework offers two measures to prevent CSRF attacks from being executed. The first one, called Session-relative URL, is used by default and therefore Portal application was already provided with this protection.

The Session-relative URL means that there is a numerical counter bound to session that contains the number of actions user took from the beginning of the session. This counter is then used as a part of URL which looks, for instance, as: *http://147.228.64.172:8080/experiments-list?9*. This number is contained also in generated forms and compared upon their submission.

Figure 4.1: Object References Security Test.

Page	Parameters	Safe
AccessDeniedPage	no	yes
AccountOverViewPage	no	yes
ArticlesPage	no	yes
ConfirmPage	no	yes
DataFileDetailPage	yes	yes
ExperimentsDetailPage	yes	yes
ForgottenPasswordPage	no	yes
HistoryPage	no	yes
HomePage	no	yes
ChangePasswordPage	no	yes
ListExperimentsPage	yes	yes
ListPersonPage	no	yes
ListResearchGroupsPage	yes	yes
ListScenariosPage	no	yes
ListsPage	no	yes
MyGroupsPage	no	yes
PersonDetailPage	yes	yes
RegistrationPage	no	yes
ResearchGroupsDetailPage	yes	yes
ScenarioDetail	yes	yes
SocialNetworksPage	no	yes
UnderConstructPage	no	yes
WelcomePage	no	yes

However, this solution provides only a basic protection against CSRF, the exchanged value isn't encrypted and could potentially be guessed. In order to improve application's protection against CSRF, second measure, offered by Apache Wicket framework, was implemented and is described in section 5.4 on page 32.

## 4.6 Security Configuration

The level of security in EEG/ERP Portal in relation to Security Misconfiguration was revised as described in section 3.6 on page 17. Application uses default error page and printing StackTrace to visitors is disabled. It uses Spring Security framework providing a system of roles to restrict access to resources. There was no weakness found and therefore no action taken. Only recommendation given is to keep using most up to date libraries and web server.

## 4.7 Cryptographic Storage

Application's database, containing private information collected from test subject, did not use any encryption beside of storing passwords as hash imprints. Therefore, a set of private information was chosen and secured using data encryption as described in section 5.5 on page 35.

## 4.8 Restricting URL Access

To restrict access to certain pages for registered users only a system of roles is used. Currently the system defines a role of an administrator, a registered user and anonymous visitor. This is ensured using Spring Security framework and authorization annotations defining access requirements for particular pages. No action were necessary to improve current system.

## 4.9 Transport Layer Protection

One of recommendations given in Thesis [14] investigating security measures in EEG/ERP Portal in 2010, was to provide sufficient protection of communication channel using SSL protocol. The EEG/ERP Portal currently uses such measure therefore the information exchanged between client and server is not exposed and potentially vulnerable to be exploited.

## 4.10 Validation of Forwards and Redirects

The EEG/ERP Portal does not use forwards or redirects that could be directly influenced by URL parameter modifications. Therefore, application is safe against such threat and no improvement was necessary.

## 4.11 Clickjacking Vulnerability

The EEG/ERP Portal did not have any security measure against Clickjacking and was vulnerable to such exploit. The implemented security improvement is described in section 5.6 on page 43 along with test performed to verify this solution.

# Chapter 5

## Security Improvements in EEG/ERP Portal

### 5.1 Preventing Injections

In current implementation, data stored in database are accessed using Data Access Objects. Some of their methods use queries in form shown in listing 5.1, some in form shown in listing 5.2.

Listing 5.1: Hibernate HQL example 1.

```
1 List<String> list = getSessionFactory()  
2     .getCurrentSession()  
3     .createQuery(hqlQuery).list();
```

Listing 5.2: Hibernate HQL example 2.

```
1 List<String> list = getHibernateTemplate()  
2     .findByNameParam(hglQuery,  
3     "groupId", groupId);
```

According to Hibernate documentation:

*As of Hibernate 3.0.1, transactional Hibernate access code can also be coded in plain Hibernate style. Hence, for newly started projects, consider adopting the standard Hibernate3 style of coding data access objects instead, based on `SessionFactory.getCurrentSession()`. [15]*

it is now supported creating queries by invoking `SessionFactory` methods and using `HibernateTemplate` is no longer recommended. This means newly added DAOs and their methods should follow this rule and current implementation should be unified so all queries are created and used in a same way.

This is not an imminent security risk, but unified approach should be used in order to simplify any potential transformation in case any security flaw is found.

## 5.2 Preventing Cross-Site Scripting

To improve protection against XSS attacks as well as protection against reading user's cookie file via such scripts, two measures were taken. First, all user input is escaped when outputted on a page. This is ensured by Apache Wicket framework, which escapes any output by default as described in section 4.2 on page 25. Second, application's cookies were set to be HTTP-only, and therefore readable only through HTTP requests, as described in chapter 3.2.1 on page 11.



### 5.3 Authentication and Session Management

As described in section 4.3 on page 25, there is no limitation on how many attempts to login can be made. After consideration of possible solutions and their impact on both, application security and user comfort, no actions were made. If login attempts were bound to username, attacker could simply guess valid usernames by being locked out after several unsuccessful login attempts leaving him/her with a list of valid usernames. If users were simply locked out for another login attempt for certain period of time, an attacker could exploit such measure to limit real account owners from logging in as well, resulting in inability to use the service. If login attempts were counted per session or IP address, attacker could simply restart or change them before each attempt leaving him in same position as though as there were no limitations. In addition, guessing a matching pair of username and password using brute force based attack would take an excessive amount of time even for one single user.

Considering the above-mentioned reasons and a fact, that any measures taken might have impact on user's comfort of using EEG/ERP Portal, no actions to limit login attempts were taken.

### 5.4 Cross-Site Request Forgery Protection

The CSRF protection implementation introduced in section 3.5.1 on page 14 would be convenient to use in former version of EEG/ERP Portal using SpringWeb MVC. But since it now uses the Apache Wicket, more suitable and complex solution was at hand. Apache Wicket provides a so called CryptoMapper [16] allowing to encrypt any URL generated by the server. The encryption algorithm generates a private key for each user and stores it in his/her session. In addition, generated key is changed for every session.

An example of encrypted URL is shown in Figure 5.1.

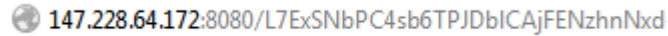


Figure 5.1: Encrypted URL example.

Providing native CSRF protection is quite unique among other web page frameworks. It was desired by web developers during the Wicket's development phase.

### 5.4.1 Enabling CryptoMapper

Wicket's CryptoMapper can be activated by adding a command shown in listing 5.3 to *init* method of Application's main class - EEGDataBaseApplication.java. Once the CryptoMapper is activated, all generated URLs are encrypted. To keep some pages accessible by their symbolical name (e.g. /homepage) a *mountPage* method is used (see listing 5.4).

CryptoMapper can be used in multiple ways, all described in Table 5.2. Pages mounted after enabling cryptoMapper stay accessible by their mount name.

The main disadvantage of using CryptoMapper is that a page secured by encrypting URL is not bookmarkable [17] or accessible by its name. To keep a maximum level of user's comfort while providing sufficient security to most important pages a multi-level security was made. From list of pages, already shown in Table 4.1 on page 27, only those processing private information were chosen to be protected by encrypting their URL. Those are: ForgottenPasswordPage, ChangePasswordPage and RegistrationPage. For other pages, protection using session relative URL is considered sufficient.

Listing 5.3: Enabling CryptoMapper.

```
1 setRootRequestMapper(new CryptoMapper(  
    getRootRequestMapperAsCompound(), this));
```

Figure 5.2: Multi-level Security using Wicket's cryptoMapper.

Order	Advantages	Disadvantages
CryptoMapper No pages mounted	Best security option. Parameters cannot be guessed nor used for CSRF.	Pages are not accessible by their symbolical name and therefore cannot be bookmarked.
Mount Pages CryptoMapper	Medium security option. All pages are encrypted but are still accessible by their symbolical name and therefore bookmarkable. Parameter names remain hidden from the user.	Pages can be accessed by their name including parameters which could be guessed and potentially abused by attacker.
CryptoMapper Mount Pages	Mounted pages are not encrypted at all meaning the URL is clean and pages are bookmarkable. User's orientation on page might be smoother.	Weaker security option. Pages and their parameters are not protected against possible attacks.

Listing 5.4: Mounting page to its name.

```
1 mountPage("home-page", HomePage.class);
```

## 5.5 Data Storage Encryption

[18] EEG/ERP Portal project is focused on neuroinformatics science and serves as a data store of experimental data. The subjects participating in performed experiments provide their basic credentials for their identification, such as name, surname, email address or date of birth. According to the Czech law [1] this information is considered private. Anyone who collects, keeps or in any way manipulates data containing private information is bound by law to perform all actions necessary to protect this data and restrict any possible access by unauthorized persons.

Considering the fact that private information stored in database is not encrypted, one goal of this thesis is to find and implement such security measures to provide strong and sufficient protection of collected data.

### 5.5.1 Existing Solutions

[18] Considering that the project of EEG/ERP Portal uses the Oracle database it was obvious to investigate related work and possible solution. Oracle provides support for a DICOM<sup>1</sup> format [19]. It is commonly used to store medical data in the field of neuroinformatics experiments or medical experiments in general. The format's header includes private information about patients similar to information used and stored in EEG/ERP Portal. Therefore, similar methods to secure this data are to be expected.

When storing medical data containing private information about the patient there are two possible approaches. The first one assumes that the information, such as patient's name, will no longer be needed and therefore

---

<sup>1</sup><http://medical.nema.org>

can be replaced with unique hash imprint that still identifies the patient but no longer allows to decrypt their full name. Hash algorithms as SHA1 or MD5 are commonly used for this purpose. The second approach requires the private information to be securely stored with possibility of fully decrypting the stored information, for example to contact the patient etc. In such case, hash algorithms need to be replaced with encryption algorithms.

The second approach has been chosen and a way to encrypt and decrypt private data has been found for the needs of EEG/ERP Portal. Oracle database encrypts private information in aforementioned DICOM format [19] using its TDE - Transparent Data Encryption [20, 21] feature. Same approach can be used for encrypting the private data stored in EEG/ERP Portal.

### 5.5.2 Oracle Transparent Data Encryption

[18] Oracle's Advanced Security pack, which is possible to use with Enterprise edition of Oracle database, includes methods for data encryption and decryption. This can be achieved in two ways:

- Tablespace Data Encryption
- Column Data Encryption

The first option encrypts the whole database tablespace with all its data. This approach should be, according to the Oracle's manual, slightly faster and more CPU efficient compared to the column encryption because all the encryption is made in a cache memory during the select queries. On the other hand, using this kind of encryption on already existing database requires its cloning and transferring into newly made encrypted tablespace. The second option is to encrypt only chosen columns by adding an "ENCRYPT" attribute to their definition. Considering the need of transferring existing database and the fact that only a small percentage of all stored data needs to be encrypted, the second choice appears to be more convenient and fitting.

When using Column Data Encryption, every table containing an encrypted column it is provided a specific key shared by all columns within it. All this keys assigned to tables are all encrypted by so called Master key which is stored in an Oracle wallet. The Oracle wallet can be either external security device or a file. The Master key is stored outside the database and therefore can be accessed only by security administrator(who can be completely different from a database administrator). Besides storing the Master key, Oracle wallet is also responsible for the process of data encryption and decryption. The default database wallet can be used or another one can be created by specifying its location in SQLNET.ORA file.

Both ways of data encryption support commonly used encryption algorithms [22], such as AES 128, 192, 256 and 3DES 168 (the number value specifies the length of used encryption key). The Column data encryption uses the AES 192 algorithm by default. This is considered slightly faster than 3DES and its key length is sufficient. This algorithm is considered safe and its complexity of breaking the cipher by brute force is estimated to be  $2^{189}$  for the 192 bit key length.

### **5.5.3 Setting-up Transparent Data Encryption**

Column version of Transparent Data Encryption can be set and enabled in few steps [23]:

1. Create new Oracle wallet (if doesn't exist) by specifying its location in SQLNET.ORA file using ENCRYPTION\_WALLET\_LOCATION parameter.
2. Create new Master Key: ALTER SYSTEM SET ENCRYPTION KEY IDENTIFIED BY password
3. Set the wallet to be automatically opened after the database is restarted or shut down: ALTER SYSTEM SET WALLET OPEN IDENTIFIED BY password

4. It is also highly recommended to set a "sticky bit" to the wallet file so it cannot be accidentally deleted which would cause all the encrypted data to be permanently lost since there would be no way of decrypting them without the Master key. Sticky bit can be set using command: `CHMOD +t oracle_wallet_file`.

When all the steps above are performed it is possible to either create a new table or alter an existing one by specifying which columns are supposed to be encrypted. This can be done by executing command shown in listing 5.5.

Listing 5.5: Alter Table script - structure.

```
1 ALTER TABLE
2   table_name
3   modify(
4   column_name ENCRYPT ,
5   ...
6  );
```

Specific script used to encrypt personal information stored in EEG/ERP Portal database is described in listings 5.6 and 5.7 on pages 40 and 41. Beside the ENCRYPT attribute it is also possible to specify following options:

- SALT / NO SALT
- USING 'encryption\_algorithm'
- 'MAC' / 'NOMAC'

SALT option defines whether or not to use a random string to strengthen the cipher. Using salt makes identical values appear as different. It is convenient to use if data in column are not restricted to contain only unique values. Once a column is encrypted using SALT option the salt can be removed by altering the column to ENCRYPT NOSALT re-encrypting the data automatically without forcing the user to do it manually.

USING option defines the used encryption algorithm. Values 3DES168, AES128, AES192, AES256 can be used, but all columns within one table must use the same encryption algorithm.

NOMAC option specifies if integration check should be performed or not. According to Oracle documentation using NOMAC option can save up to 20B per encrypted value but it is recommended to use integrity check unless the amount of encrypted data is excessive. By default 'SHA-1' algorithm is used.

### 5.5.4 Choosing Data for Encryption

After analysis of data stored in database it was decided to encrypt only specific columns in PERSON table (see Figure 5.3).





PERSON		
P *	PERSON_ID	NUMBER
	GIVENNAME	VARCHAR2 (50 BYTE)
*	SURNAME	VARCHAR2 (50 BYTE)
	DATE_OF_BIRTH	DATE
*	GENDER	CHAR (1 BYTE)
U	EMAIL	VARCHAR2 (50 BYTE)
	PHONE_NUMBER	VARCHAR2 (20 BYTE)
	NOTE	VARCHAR2 (255 BYTE)
U	USERNAME	VARCHAR2 (50 BYTE)
	PASSWORD	VARCHAR2 (80 BYTE)
	AUTHORITY	VARCHAR2 (50 BYTE)
	DEFAULT_GROUP_ID	NUMBER
	REGISTRATION_DATE	DATE
	CONFIRMED	NUMBER
	AUTHENTICATION	VARCHAR2 (50 BYTE)
U	FB_UID	NUMBER
*	LATERALITY	CHAR (1 BYTE)
*	EDUCATION_LEVEL_ID	NUMBER
 PERSON__IDX (PERSON_ID)		
 PERSON__IDXv1 (EMAIL)		
 PERSON__IDXv2 (USERNAME)		
 PERSON__IDXv3 (FB_UID)		

Figure 5.3: Definition of PERSON table.



All values that could possibly lead to identifying a real person were taken into consideration of encrypting them. From above-mentioned columns it is name, surname, date of birth and possibly also email address, Facebook id and phone number that are all easily traceable and connectible with real person.

All columns that require unique values can be encrypted with the NO SALT option, while SALT option was used with other columns to ensure higher security by all values appearing different from each other. To alter the table definition, following script showed in listing 5.6 was used.

Listing 5.6: Alter Table script - encryption.

```
1 ALTER TABLE
2 PERSON
3 modify
4 (
5     NAME          encrypt ,
6     SURNAME       encrypt ,
7     DATE_OF_BIRTH encrypt ,
8     email         encrypt  no salt ,
9     fb_uid        encrypt  no salt ,
10    phone_number  encrypt
11 );
```

In case this encryption ever needs to be removed, similar script can be used to turn the encryption off (see listing 5.7).

Listing 5.7: Alter Table script - decryption.

```
1 ALTER TABLE
2 PERSON
3 modify
4 (
5     NAME            decrypt ,
6     SURNAME         decrypt ,
7     DATE_OF_BIRTH  decrypt ,
8     email           decrypt ,
9     fb_uid          decrypt ,
10    phone_number    decrypt
11 );
```

### 5.5.5 Impact of Encryption on Database Response Time

[18] To compare the difference in database performance with or without using the Transparent Data Encryption a simple test case was prepared.

A table with exactly the same definition as PERSON table was created, once using the encryption and once without it. Then this table was repeatedly filled with certain amount of data - 100, 1000 and 10000 rows. For each of this number a full select over the table was performed and value of response time was measured. This was done 100 times for each table size without data encryption and repeated for the same amount but on the table with encrypted data.

From these measured times an average value, which is shown in graph in Figure 5.4 on page 42, was taken.

Also a SELECT including WHERE clause was tested using the same table and scenario. The results for query SELECT \* from test\_table WHERE email = 'test\_subject@gmail.com' are shown in Figure 5.5 on page 42.

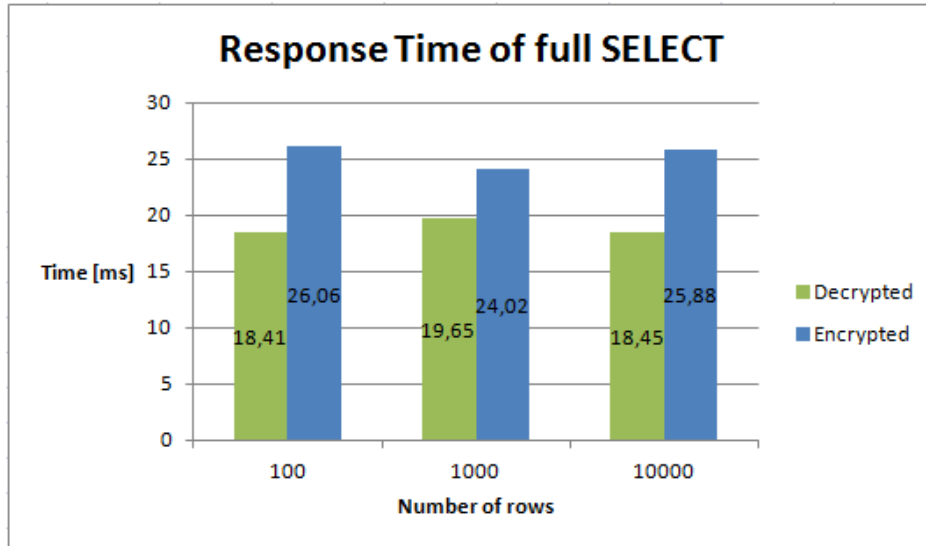


Figure 5.4: Graph of response times with/out TDE.



Figure 5.5: Graph of response times for SELECT including WHERE clause.

These values prove that there is some increase in response time, yet it has to be taken into consideration that due to Oracle's query optimisation and result caching its exact value doesn't necessarily have to depict reality. That is why this was tested on bigger amount of data while the application usually fetches only a single row. In addition, the measured times also include the amount of time necessary to transport the results over the internet which might have influenced their values.

Considering the results provided by aforementioned test it is very easy and convenient to use Oracle's Transparent Data Encryption for securing the stored private data without any significant impact on database performance. It should be still kept in mind that securing data with Oracle TDE only protects physically stored data against direct access of unauthorized person but has no influence on possible abuse of web application or monitoring an ongoing information exchange between client and application server. This security measure is therefore supported by other improvements, such as by using secure encrypted communication on web via SSL protocol.

## 5.6 Clickjacking Prevention

To protect EEG/ERP Portal against Clickjacking, the second approach, using the X-FRAME-OPTIONS, described in section 3.11.1 on page 21, was used. X-FRAME-OPTIONS parameter can be added into page's HTTP header by overriding *setHeaders* method in *BasePage* class definition (see example in listing 5.8).

Listing 5.8: X-FRAME-OPTIONS settings.

```
1 @Override
2 protected void setHeaders(WebResponse response) {
3     super.setHeaders(response);
4     response.setHeader("X-Frame-Options", "deny");
5 }
```

This solution was tested using simple web page code shown in listing 5.9. For demonstration purposes, *home-page* was left without Clickjacking protection while *experiments-list* had the X-FRAME-OPTIONS parameter set to "DENY". As you can see in Figure 5.6, the unprotected page is rendered as usual while the iframe with protected page remains blank.

Listing 5.9: X-FRAME-OPTIONS test.

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <h1>Test X-FRAME-OPTIONS</h1>
6
7 <p>home-page</p>
8 <p><iframe src="http://localhost:8080/home-page">
9     </iframe></p>
10
11 <p>experiments-list</p>
12 <p><iframe src="http://localhost:8080/experiments-
13     list"></iframe></p>
14 </body>
15 </html>
```

## Test X-FRAME-OPTIONS

home-page



experiments-list



Figure 5.6: Results of X-FRAME-OPTIONS test.

# Chapter 6

## Building Secured E-Shop

Providing strong security measures is a main goal of any online store development. This chapter depicts design and implementation of secured online store for EEG/ERP Portal. The application must ensure that the whole process of purchase, including the payment processing, is secured and cannot be exploited. The EEG/ERP Portal is intended to share its collected data with public community, therefore part of this thesis suggests security measures needed to provide a possibility of selling the collected experimental data.

### 6.1 Current State of EEG/ERP Portal

In current situation, EEG/ERP Portal offers a possibility to share and download scenarios and experiments among registered users. The research group, developing the Portal project, has a vision of allowing Portal visitors to purchase data collected from performed experiments. Such goal will require labelling published data as either public, downloadable for free, or private, purchasable for a fee.

The experiments should be grouped into purchasable packs, each containing several experiments or scenarios. These packs will contain only

anonymized data, so it will be impossible to anyhow identify the subjects measured during these experiments.

The package structure, amount of experiments it will contain and its pricing is recently being discussed. Therefore the developed store operates with single experiments and serves as a prototype or proof of concept.

## 6.2 Shopping Cart Functions

The designed online store is required to provide following functions:

- browse store of available experiments
- add experiments to the cart
- review the cart content
- remove experiments from the cart
- check out current cart content as an order
- pay for an order online

## 6.3 Payment Options

When building an online store, it is important to offer various payment methods. Since each method requires its own implementation and particular payment methods are being subject of current discussion, it was decided to implement only the PayPal payment option. In addition to PayPal services, application could, in the future, accept payments with credit cards or other online payment services similar to PayPal, such as eGold, MoneyBookers or Stormpay.

PayPal service was chosen as one of the most used and known online payment services, and so a high level of security is to be expected.



## 6.4 Shopping Cart Design

The design of EEG/ERP Portal's shopping cart is based upon example of Apache Wicket tutorial book [13]. Since the shopping cart object is unique for each user and his session, the object of shopping cart is placed in the *EEGApplicationSession* object, which offers methods for cart creation and admission. The shopping cart is destroyed together with session when it cease to exist.

For more details on Shopping cart and Session class, such as attribute and method list, see UML diagram in Figure 6.1.

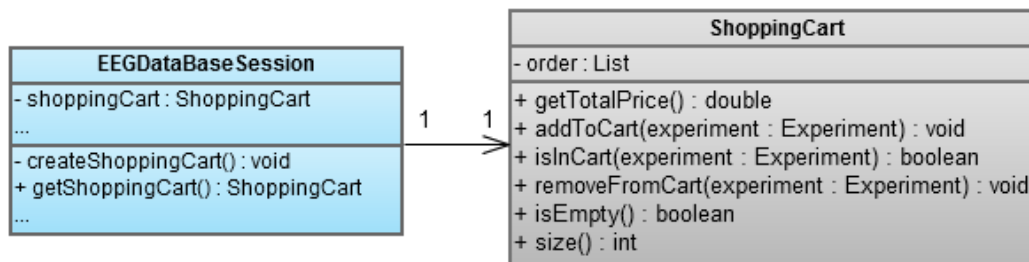


Figure 6.1: Shopping Cart UML diagram.

Shopping cart is a wrapper object to manage user's order - add items to the cart, remove them, compute the total price of order and perform the process of checking out and paying.

The very order is implemented as an ArrayList of Experiments. Since user can buy an experiment only once, there was no need to deal with quantity of purchased items. Therefore, object of Map, usually used for managing orders, could be replaced with a List of items.

To ensure each experiment can be placed inside a list of ordered items only once, the shopping cart's addToCart method always checks experiment's presence before adding it into order. In addition, this restriction is also supported by making "Add to Cart" links invisible for every experiment already placed in an order. In future, when the EEG/ERP Portal's store goes live, this rule should be extended even for experiments user already bought.

## 6.5 Example Purchase Scenario

This section describes the process of online purchase from user's point of view. The purchase use case is shown to introduce the context of payment processing. The security measures taken to secure each of the following purchase steps are described in section 6.7 on page 59.

1. In order to view a list of purchasable experiments, user has to be logged in.

See an example of Experiment store in Figure 6.2 on page 50.

2. While browsing through available experiments, user can simply add desired experiment into his/her cart by clicking the "Add to Cart" link. See an "Add to Cart" link in Figure 6.3 on page 50.

Note, that the user can always see how many items are there in a cart by checking the number shown next to "My Cart" link placed in top right navigation menu.

Top right menu can be seen in Figure 6.4 on page 50.

3. When done adding experiments in the cart, the user can view its current content by clicking the "My Cart" link placed in top right navigation menu.

Top right menu can be seen in Figure 6.4 on page 50.

4. On the "My Cart" page, the user can review details of selected experiments, remove them from the cart, see total price of order and select a payment method.

An example of "My Cart" page can be seen in Figure 6.5 on page 52.

EEGbase

Logged user: asarcz@gmail.com | [My account](#) | [My Cart: 0](#) | [Log out](#)

Home Articles Search Experiments Scenarios Groups People Lists History

All experiments

My experiments

Me as subject

Search experiment

Add experiments

Services results

### All experiments

Showing 1 to 20 of 216  
 << < 1 2 3 4 5 6 7 8 9 10 > >>

No.	Scenario title	Date	Gender	Year of birth	Services	Detail	Download	Buy
0	Vliv únavy na řízení automobilu	18.2.02 10:20	M	1974	<a href="#">Services</a>	<a href="#">Detail</a>	<a href="#">Download</a>	<a href="#">Add to Cart</a>
12	ahoj	4.4.10 9:45	M	1982	<a href="#">Services</a>	<a href="#">Detail</a>	<a href="#">Download</a>	<a href="#">Add to Cart</a>
13	Vliv	4.4.10 9:45	M	1984	<a href="#">Services</a>	<a href="#">Detail</a>	<a href="#">Download</a>	<a href="#">Add to Cart</a>
14	Vliv únavy na řízení automobilu	16.4.09 9:45	M	1982	<a href="#">Services</a>	<a href="#">Detail</a>	<a href="#">Download</a>	<a href="#">Add to Cart</a>
15	omg	16.4.09 9:45	M	1990	<a href="#">Services</a>	<a href="#">Detail</a>	<a href="#">Download</a>	<a href="#">Add to Cart</a>
16	Reakce na významnou událost	16.4.09 9:45	M	1982	<a href="#">Services</a>	<a href="#">Detail</a>	<a href="#">Download</a>	<a href="#">Add to Cart</a>
17	meri mozrove vlny	16.4.09 9:45	F	1982	<a href="#">Services</a>	<a href="#">Detail</a>	<a href="#">Download</a>	<a href="#">Add to Cart</a>
18	measurement	16.4.09 9:45	M	1982	<a href="#">Services</a>	<a href="#">Detail</a>	<a href="#">Download</a>	<a href="#">Add to Cart</a>
19	xml soubor	26.4.09 9:45	M	1976	<a href="#">Services</a>	<a href="#">Detail</a>	<a href="#">Download</a>	<a href="#">Add to Cart</a>
20	Reakce na významnou událost	5.12.09 12:30	F	1982	<a href="#">Services</a>	<a href="#">Detail</a>	<a href="#">Download</a>	<a href="#">Add to Cart</a>
21	Reakce na významnou událost	5.12.09 12:30	F	1982	<a href="#">Services</a>	<a href="#">Detail</a>	<a href="#">Download</a>	<a href="#">Add to Cart</a>
22	measurement	5.8.10 15:00	M	2011	<a href="#">Services</a>	<a href="#">Detail</a>	<a href="#">Download</a>	<a href="#">Add to Cart</a>
23	measurement	5.8.10 15:00	M	1982	<a href="#">Services</a>	<a href="#">Detail</a>	<a href="#">Download</a>	<a href="#">Add to Cart</a>
24	xml soubor	1.5.09 12:00	M	1982	<a href="#">Services</a>	<a href="#">Detail</a>	<a href="#">Download</a>	<a href="#">Add to Cart</a>

Figure 6.2: Experiment list example.

14	Vliv únavy na řízení automobilu	16.4.09 9:45	M	1982	<a href="#">Services</a>	<a href="#">Detail</a>	<a href="#">Download</a>	<a href="#">Add to Cart</a>
----	---------------------------------	--------------	---	------	--------------------------	------------------------	--------------------------	-----------------------------

Figure 6.3: Add to Cart link.

Logged user: asarcz@gmail.com | [My account](#) | [My Cart: 5](#) | [Log out](#)

Figure 6.4: My Cart menu.

5. After reviewing the current order and choosing the payment method, user is redirected to PayPal's web site in order to authorize the payment.
  - (a) User is prompted to log in using his PayPal credentials.

See an example of PayPal login screen in Figure 6.6 on page 52.
  - (b) Upon successful login, user is shown his/her order's description along with total price and vendor's name, EEG Database, in this case. In this step, the user is asked to authorize the payment by clicking the "Continue" button. The user can also cancel current transaction. In such case, the user is redirected back into his/her shopping cart on EEG/ERP Portal web site.

See an example of payment authorization page in Figure 6.7 on page 53.
6. When the payment is authorized, the user is redirected to EEG/ERP Portal's payment confirmation page. On this page, the user is asked to do a final revision of authorized payment's total price and a list of ordered items. The user can either confirm or cancel his order. Upon cancelling the order, the user is redirected back to his/her cart, where he/she can modify the items placed in the order.

See an example of Order confirmation page in Figure 6.8 on page 53.
7. Upon confirming the payment, user's PayPal account is billed for amount of order's total price and user is then redirected to "My Downloads" page. There, user is provided with successful payment confirmation message along with a list of purchased items.

See an example of My Downloads page in Figure 6.9 on page 54.
8. In case an error occurs when processing the payment, the user is redirected to an Error page. The Error page contains a brief information that the order could not have been completed and offers a possibility to continue either to user's cart or EEG/ERP Portal's home page. See an example of Error page in Figure 6.10 on page 54.

EEGbase

Logged user: asarcz@gmail.com | [My account](#) | [My Cart: 5](#) | [Log out](#)

[Home](#) [Articles](#) [Search](#) [Experiments](#) [Scenarios](#) [Groups](#) [People](#) [Lists](#) [History](#)

My Cart

My Downloads

### My Cart

No.	Scenario title	Date	Gender	Year of birth	Detail	Remove
0	Vliv únavy na řízení automobilu	18.2.02 10:20	M	1974	<a href="#">Detail</a>	<a href="#">Remove</a>
18	measurement	16.4.09 9:45	M	1982	<a href="#">Detail</a>	<a href="#">Remove</a>
20	Reakce na významnou událost	5.12.09 12:30	F	1982	<a href="#">Detail</a>	<a href="#">Remove</a>
24	xml soubor	1.5.09 12:00	M	1982	<a href="#">Detail</a>	<a href="#">Remove</a>
180	ScenarioWithAvg	18.9.11 10:06	M	2010	<a href="#">Detail</a>	<a href="#">Remove</a>

Total Price: 25.0 €

[Check out with PayPal](#)  
The safer, easier way to pay

EEGbase - database for data gained in encephalography research.  
Copyright © The University of West Bohemia 2008-2012

Figure 6.5: My Cart page.

### EEG Database's Test Store

**Your order summary**

Descriptions

Experiments for total of: 15.0 GBP

You'll be able to see your order details before you pay.

**Choose a way to pay**

▼ **Pay with my PayPal account**

Log in to your account to complete the purchase

Email  
user11@zcu.cz

PayPal password  
●●●●●●●●

**Log In**

[Forgotten your email address or password?](#)

► **Pay with a debit or credit card**

(Optional) Sign up to PayPal to make your next checkout faster

[Cancel and return to EEG Database's Test Store.](#)

Figure 6.6: PayPal login screen.

EEG Database's Test Store

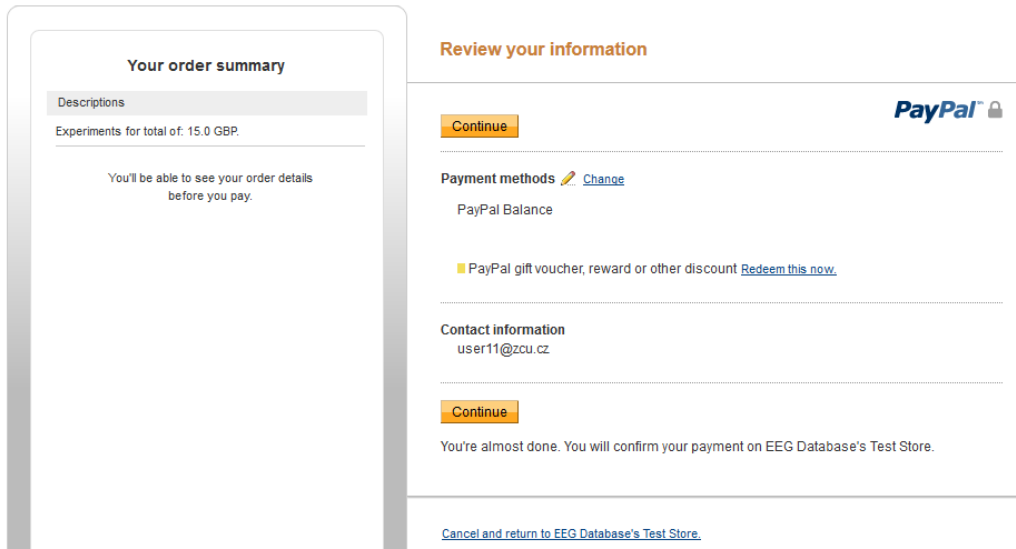


Figure 6.7: PayPal payment authorization page.

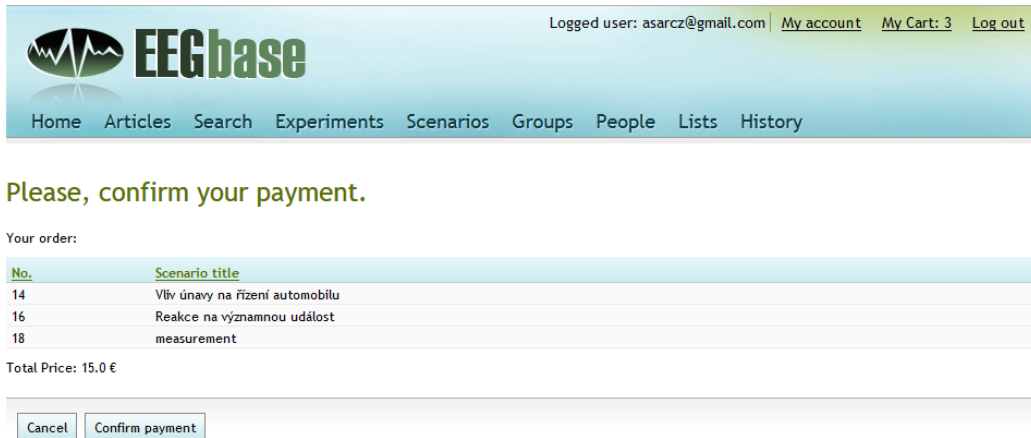


Figure 6.8: Order confirmation page.

Logged user: asarcz@gmail.com | [My account](#) | [My Cart: 0](#) | [Log out](#)

Home Articles Search Experiments Scenarios Groups People Lists History

My Cart  
My Downloads

### My Downloads

Congratulations, You have just bought:

No.	Scenario title	Download
14	Vliv únavy na řízení automobilu	<a href="#">Detail</a>
16	Reakce na významnou událost	<a href="#">Detail</a>
18	measurement	<a href="#">Detail</a>

EEGbase - database for data gained in encephalography research.  
Copyright © The University of West Bohemia 2008-2012

Figure 6.9: My Downloads page.

Logged user: asarcz@gmail.com | [My account](#) | [My Cart: 1](#) | [Log out](#)

Home Articles Search Experiments Scenarios Groups People Lists History

### Payment Error

An error occurred when processing your payment. Please try again later.

[Go to homepage](#) [Back to My Cart](#)

EEGbase - database for data gained in encephalography research.  
Copyright © The University of West Bohemia 2008-2012

Figure 6.10: Error page.

## 6.6 Processing a Payment

Since EEG/ERP Portal store deals only with digital goods, there is no need to bother customers with a need to fill in unnecessary shipping details. For this purpose PayPal offers a so called "Express Checkout" service which is a modification of standard checkout process but eliminating the need of filling any additional information. The Express Checkout process is shown in Figure 6.11 on page 56 and each phase is described in this section.

To use PayPal's payment services within an application developers are provided with PayPal SDK libraries<sup>1</sup> along with application credentials tied to a business account. Every PayPal business account receives 3 keys to identify and authorize vendor's application when billing customer's account as shown in listing 6.1.

Payment processing consist of following steps:

User chooses to perform an Express Checkout with PayPal. Vendor application then needs to build a request to be sent to PayPal's server. This *setExpressCheckout* request must contains following information as shown in listing 6.2.

Listing 6.1: PayPal credentials example.

```
1 Username: eegportal_api1.zcu.cz
2 Password: 1363432632
3 Signature: AFcWxV21C7fd0v3bYYYRCpSSRl31Am.91
4           JsLe8yAsbqHkYJ35J4tbGWk
```

---

<sup>1</sup><http://paypal.github.io>



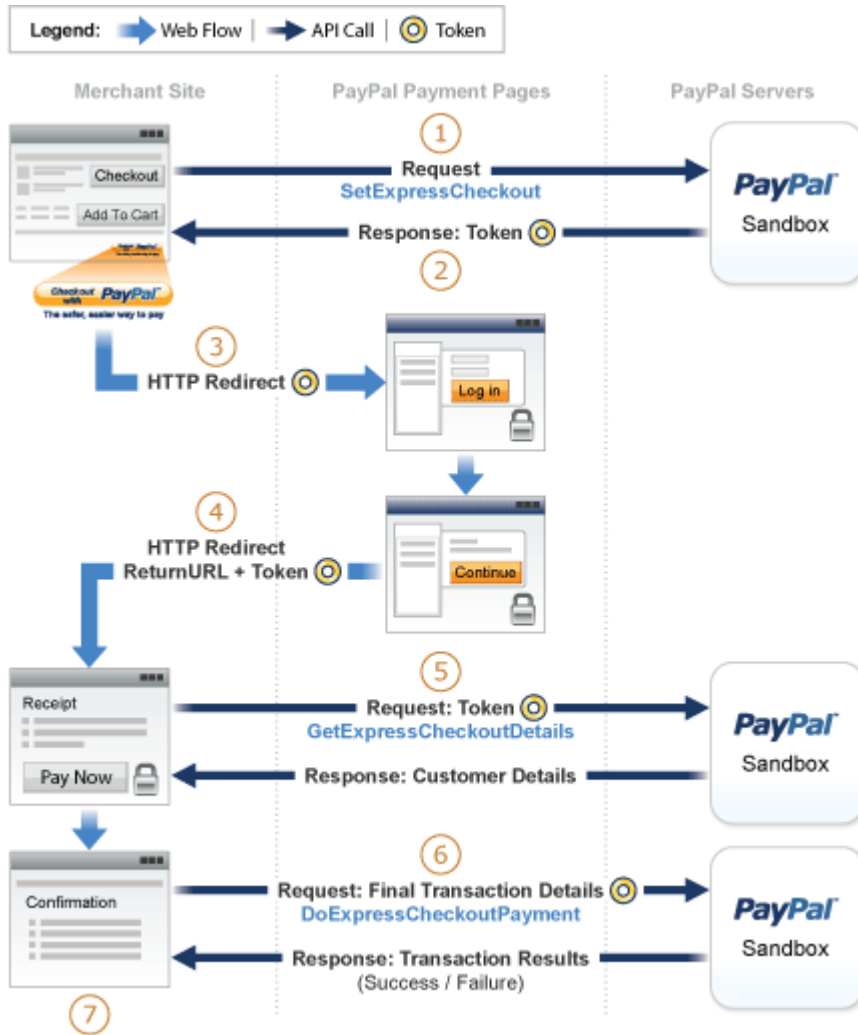


Figure 6.11: Express Checkout process [24].

Listing 6.2: `setExpressCheckout` request example [25].

```
1 "USER=<callerID> # User ID of the PayPal caller
   account
2 &PWD=<callerPswd> # Password of the caller account
3 &SIGNATURE=<callerSig> # Signature of the caller
   account
4 &METHOD=SetExpressCheckout
5 &VERSION=93
6 &PAYMENTREQUEST_0_PAYMENTACTION=SALE # type of
   payment
7 &PAYMENTREQUEST_0_AMT=19.95 # amount of transaction
8 &PAYMENTREQUEST_0_CURRENCYCODE=USD # currency of
   transaction
9 &RETURNURL=http://www.example.com/success.html # URL
   of your payment confirmation page
10 &CANCELURL=http://www.example.com/cancel.html # URL
   redirect if customer cancels payment"
```

When the request is completed, it is sent to PayPal's server that registers a payment with provided details and responds with a token identifying requested payment. The *setExpressCheckout* request is created and submitted using the aforementioned PayPal SDK's API call. When application receives requested token, it redirects the user to PayPal web site using received token's value as an URL parameter in form:

*[https://www.paypal.com/webscr?cmd=\\_express-checkout&token=<tValue>](https://www.paypal.com/webscr?cmd=_express-checkout&token=<tValue>)*

User is then prompted to log in using his PayPal credentials, review his order's total price and description and authorize the payment to be charged. Upon user's decision, he is redirected either to cancel or success URL specified in the *setExpressCheckout* request.

After authorizing a payment for PayPal, the user is requested to confirm the payment again for vendor application. When both confirmations, for PayPal and EEG/ERP Portal store, are given, application needs to get

user's PayPal ID in order to charge him/her. To receive the payer's ID, *getExpressCheckout* request, containing customer's payer ID, must be sent. The *getExpressCheckout* request must specify details needed as shown in listing 6.3.

Listing 6.3: *getExpressCheckout* request example [25].

```
1 "USER=<callerID>
2 &PWD=<callerPswd>
3 &SIGNATURE=<callerSig>
4 &METHOD=GetExpressCheckoutDetails
5 &VERSION=93
6 &TOKEN=<tokenValue>"
```

When application receives customer's PayPal payer ID, his account can be finally charged for his order's price. The billing is requested sending *doExpressCheckout* request containing details as shown in listing 6.4. Both these requests are created and submitted to PayPal's server using the provided SDK's API.

Listing 6.4: *doExpressCheckout* request example [25].

```
1 "USER=<callerID>
2 &PWD=<callerPswd>
3 &SIGNATURE=<callerSig>
4 &METHOD=DoExpressCheckoutPayment
5 &VERSION=93
6 &TOKEN=<tokenValue>
7 &PAYERID=<payerID> # customer's unique PayPal ID
8 &PAYMENTREQUEST_0_PAYMENTACTION=SALE # payment type
9 &PAYMENTREQUEST_0_AMT=19.95 # transaction amount
10 &PAYMENTREQUEST_0_CURRENCYCODE=USD # transaction
    currency, e.g. US dollars"
```

## 6.7 Securing the Purchase Process

The process of purchasing experiments is divided into several phases:

- Order review
- Checkout
- Payment authorization on PayPal
- Payment confirmation in EEG/ERP Portal
- Executing transaction
- Transaction success confirmation
- Informing about error during payment processing

The security measures required were taken into consideration for each of the listed phases. The store must ensure, that no one can alter the content of the cart or its pricing, neither an attacker nor user himself. The following security measures were implemented:

- The cart does not store the information about order's total price. The price is computed whenever requested.
- The process of checking out requires two-step confirmation.
- All URLs involved in payment processing are encrypted with user and session specific key.
- For security reasons, the error page does not specify the error occurred. Revealing any details could lead into exposing a possibly exploitable weakness.

### 6.7.1 Total Price Protection

As was already stated, the shopping cart does not store the information about order's total price. For security reasons, the total price is computed whenever requested to ensure the price always matches the cart's content.

### 6.7.2 Two-step Confirmation

After authorizing a payment for PayPal, the user is requested to confirm the payment again for vendor application. The Two-step confirmation scheme is shown in Figure 6.12 on page 61. The blue highlighted part depicts the first step - payment authorization on PayPal web site. The grey highlighted part shows the second step - payment confirmation in vendor application. This two-step confirmation is required by PayPal's service policy [25, 24]. In addition, this rule provides a security measure when the user browse vendor application in multiple tabs at once. In such case, a situation could occur when the user performs checkout on PayPal in one tab while altering the order in another. To solve such exploitable weakness, application can either implement the aforementioned two-step confirmation or lock the shopping cart to prevent any changes of order during the checkout process. To implement and manage such lock is difficult in order to prevent its potential deadlock. Vendor application would have to keep track of all opened tabs. In case the user performed a checkout in one of multiple tabs, then closed this tab and tried to perform checkout from another tab, the shopping cart would still be locked by the first, already closed tab. This would result in an inability to perform checkout process at all. To avoid this issue, application would have to be aware of any tab being closed and check whether it was keeping the cart locked or not. On the other hand, when using the two step confirmation rule, a checkout process can only result in an error when altering order in one tab, while performing checkout on payment service provider's site from another tab.



Figure 6.12: Two-step confirmation scheme [24].

For instance, when creating a checkout request, application needs to specify the total price of order. When user authorizes the payment and confirms the order in vendor application, another request, to bill the user's account, is created and sent. This request contains the total price again. In case, order has been altered, these two prices would differ and PayPal would refuse to process such payment. In that case, PayPal would answer the vendor application with a negative response indicating an error during the payment process. If such situation occurs in current EEG/ERP Portal's shopping cart, the user is redirected to Error page with information about unsuccessful payment. The user can then try to perform the whole checkout process again.

Following the OWASP's recommendation (see section 3.6 on page 17), EEG/ERP Portal uses default Error page for any error that occurs during payment processing in order not to expose unnecessary and possibly exploitable error details.

### 6.7.3 Encryption

Every URL involved in shopping process is encrypted with user and session bound secret key. In addition, these pages have no symbolical name and are therefore inaccessible by guessing their URL names. This ensures that there is no URL an attacker could abuse to modify shopping cart content or its total price. It also provides overall immunity against CSRF attacks.

The encryption is ensured using the Apache Wicket's CryptoMapper described in section 5.4 on page 32.

## 6.8 Description of Implemented E-Shop

Any visitor, who is authorized to view a list of shared experiments is also allowed to purchase them. When viewing said list, user can add desired experiment into his cart. Any experiment can be added into the shopping cart only once. This is ensured in two steps. First, the implementation of shopping cart always checks, upon calling the *addToCart* method, whether the requested experiment isn't in the cart already. Second, the "Add to Cart" link is disabled and not shown for experiments that are already in the cart.

When user is done selecting experiments, he/she can then proceed to review content of his order by clicking "My Cart" link placed in top right navigation menu next to his/her profile management link. While browsing his/her order, user can remove experiments from the cart, view chosen experiment's details or checkout his/her order and proceed to pay. Since the prototype only operates with single experiments instead of packages and the pricing wasn't decided yet, every experiment has a fixed price of 5€.

The left menu of shopping cart is designed to allow the user to either view the current content of his/her order or let him/her view a list of previous purchases, allowing him/her to re-download them. For aforementioned reasons, the "My downloads" page currently serves only for viewing the content of order upon its successful realization. The application currently doesn't store

any information about purchased experiments and therefore users cannot view their purchase history.

The shopping cart currently offers only one payment method - PayPal's Express Checkout. The other payment methods can be added in the future. In addition, all PayPal payments currently take place in PayPal's Sandbox [26] - test server for developers.



# Chapter 7

## Conclusion

In this thesis, I investigated and described current security threats identified by The Open Web Application Project in field of web applications. Each such threat was described along with general solution for its removal and prevention.

In Chapter 4, the current level of security in EEG/ERP Portal is tested in order to ensure the project provides sufficient security measures to protect private information collected from performed experiments. Investigating and improving security measures is required by Czech law whenever private data are collected, stored or processed. Security level was evaluated and recommendations for security improvements were given and implemented in Chapter 5.

In Chapter 6, I designed and implemented a prototype of secured online store for EEG/ERP Portal web site. Its design is based on security measures implemented in previous parts of this thesis. The e-shop offers an ability to browse a list of available experiments, add them to cart, checkout and pay the order using PayPal's online payment services. An access to shopping cart is secured using Apache Wicket's CryptoMapper, encrypting all related URLs. The checkout process follows the two-step confirmation rule to ensure the order cannot be anyhow exploited or altered during the payment processing.

The implemented online store is prepared to be extended in the future by adding more payment options.

# List of Abbreviations

CSRF	Cross-Site Request Forgery
EEG	Electroencephalography
ERP	Event-Related Potential
GUI	Graphical User Interface
HQL	Hibernate Query Language
MVC	Model View Controller
ORM	Object Relational Mapping
OWASP	The Open Web Application Security Project
SQL	Structured Query Language
SSL	Secure Sockets Layer
TDE	Transparent Data Encryption
URL	Uniform Resource Locator
XML	Extensible Markup Language
XSS	Cross-Site Scripting

# References

- [1] Czech law about protecting private information. [online], cited: 1. 1. 2013. <http://business.center.cz/business/pravo/zakony/ooou/cast1h2.aspx>.
- [2] The Open Web Application Security Project. Top 10 list - 2010. [online], cited: 17. 4. 2013. [owasptop10.googlecode.com/files/OWASPTop10-2010.pdf](https://owasptop10.googlecode.com/files/OWASPTop10-2010.pdf).
- [3] The Open Web Application Security Project. Top 10 list - 2013. [online], cited: 17. 4. 2013. [https://www.owasp.org/index.php/Top\\_10\\_2013-T10](https://www.owasp.org/index.php/Top_10_2013-T10).
- [4] Tsung-Po Lin. Shih-Kun Huang. D.T. Lee. Sy-Yen Kuo. Yao-Wen Huang., Chung-Hung Tsai. A testing framework for Web application security assessment. *Computer Networks*, Volume 48(Issue 5):Pages 739–761, 5 August 2005.
- [5] GOLLMAN Dieter. Securing Web applications. *Information Security Technical Report*, Volume 13(Issue 1):Pages 1–9, 2008.
- [6] MORGAN David. Web application security – SQL injection attacks. *Network Security*, Volume 2006(Issue 4):Pages 4–5, April 2006.
- [7] Hee Beng Kuan Tan. Lwin Khin Shar. Automated removal of cross site scripting vulnerabilities in web applications. *Information and Software Technology*, Volume 54(Issue 5):Pages 467–478, May 2012.

- [8] RITCHIE Paul. The security risks of AJAX/web 2.0 applications. *Network Security*, Volume 2007(Issue 3):Pages 4–8, March 2007.
- [9] The Open Web Application Security Project. Testing for CSRF. [online], cited: 17. 4. 2013. [https://www.owasp.org/index.php/Testing\\_for\\_CSRF\\_\(OWASP-SM-005\)](https://www.owasp.org/index.php/Testing_for_CSRF_(OWASP-SM-005)).
- [10] SANDHU Ravi. Speculations on the science of web user security. *Computer Networks*, Volume 56(Issue 18):Pages 3891–3895, December 2012.
- [11] ZUASTI Ricardo. Preventing CSRF in Java web apps. [online], cited: 17. 4. 2013. <http://ricardozuasti.com/2012/preventing-csrf-in-java-web-apps/>.
- [12] The Open Web Application Project. Clickjacking Defense Cheat Sheet. [online], cited: 19. 4. 2013. [https://www.owasp.org/index.php/Clickjacking\\_Defense\\_Cheat\\_Sheet](https://www.owasp.org/index.php/Clickjacking_Defense_Cheat_Sheet).
- [13] HILLENIUS Eelco. DASHORST Martijn. *Wicket in Action*. Manning, Greenwich, 2009. ISBN 1-932394-98-2.
- [14] VLAŠIMSKÝ Jiří. System oprávnění v EEG/ERP portálu. Master's thesis, University of West Bohemia. Department of Computer Science and Engineering, Pilsen, 2011.
- [15] HibernateTemplate Class Documentation. [online], cited: 17. 4. 2013. <http://static.springsource.org/spring/docs/2.5.x/api/org/springframework/orm/hibernate3/HibernateTemplate.html>.
- [16] VAYNBERG Igor. *Apache Wicket Cookbook*. Packt, Birmingham, 2011. ISBN 978-1-849511-60-5.
- [17] Wicket und CSRF. [online], cited: 17. 4. 2013. <http://codepitbull.wordpress.com/2012/04/10/wicket-und-csrf-deutsch/>.

- [18] FRONĚK Jan. Data Encryption in Oracle Database. Student's project, University of West Bohemia. Department of Computer Science and Engineering, Pilsen, 2013.
- [19] Oracle Database 11g DICOM Medical Image Support. [online], September 2009. <http://www.oracle.com/technetwork/products/multimedia/overview/dicom11gr2-wp-medingsupport-133109.pdf>.
- [20] Oracle TDE documentation. [online], cited: 1. 1. 2013. [http://docs.oracle.com/cd/B19306\\_01/network.102/b14268/asotrans.htm](http://docs.oracle.com/cd/B19306_01/network.102/b14268/asotrans.htm).
- [21] Oracle Advanced Security Transparent Data Encryption Best Practices. [online], March 2012. <http://www.oracle.com/technetwork/database/security/twp-transparent-data-encryption-bes-130696.pdf>.
- [22] Oracle Advanced Security with Oracle Database 11g Release 2. [online], October 2010. <http://www.oracle.com/technetwork/database/owp-security-advanced-security-11gr-133411.pdf>.
- [23] Oracle Advanced Security Administration documentation. [online], cited: 1.1.2013. <http://students.kiv.zcu.cz/doc/oracle/network.112/e10746/toc.htm>.
- [24] PayPal. Getting Started With Express Checkout. [online], cited: 17. 4. 2013. <https://www.x.com/developers/paypal/documentation-tools/express-checkout/integration-guide/ECGettingStarted>.
- [25] PayPal. How to Create One-Time Payments Using Express Checkout. [online], cited: 17. 4. 2013. [https://www.x.com/developers/paypal/documentation-tools/express-checkout/how-to/ht\\_ec-singleItemPayment-curl-etc](https://www.x.com/developers/paypal/documentation-tools/express-checkout/how-to/ht_ec-singleItemPayment-curl-etc).
- [26] PayPal. Paypal Sandbox Overview. [online], cited: 17. 4. 2013. [https://www.x.com/developers/paypal/documentation-tools/ug\\_sandbox](https://www.x.com/developers/paypal/documentation-tools/ug_sandbox).

# Appendix A

## CD content

This thesis has a CD attached which contains following folders:

- bin - contains an executable file EEGBase.war with complete project of EEG/ERP Portal. The .war file is executable on any common web container, such as Jetty or Apache Tomcat.
- doc - contains electronic version of this thesis in a .pdf file
- src - contains source codes of EEG/ERP Portal project