



Bacheloroppgave

IBE610 Informasjonsbehandling

Web security report

Rubén Rubio Barrera

Totalt antall sider inkludert forside: 129

Molde, 30/05/2011



Mandatory statement

Each student is responsible for complying with rules and regulations that relate to examinations and to academic work in general. The purpose of the mandatory statement is to make students aware of their responsibility and the consequences of cheating. Failure to complete the statement does not excuse students from their responsibility.

<p>Please complete the mandatory statement by placing a mark <u>in each box</u> for statements 1-6 below.</p>		
1.	I/we hereby declare that my/our paper/assignment is my/our own work, and that I/we have not used other sources or received other help than is mentioned in the paper/assignment.	<input checked="" type="checkbox"/>
2.	<p>I/we hereby declare that this paper</p> <ol style="list-style-type: none"> 1. Has not been used in any other exam at another department/university/university college 2. Is not referring to the work of others without acknowledgement 3. Is not referring to my/our previous work without acknowledgement 4. Has acknowledged all sources of literature in the text and in the list of references 5. Is not a copy, duplicate or transcript of other work 	<p>Mark each box:</p> <ol style="list-style-type: none"> 1. <input checked="" type="checkbox"/> 2. <input checked="" type="checkbox"/> 3. <input checked="" type="checkbox"/> 4. <input checked="" type="checkbox"/> 5. <input checked="" type="checkbox"/>
3.	I am/we are aware that any breach of the above will be considered as cheating, and may result in annulment of the examinaion and exclusion from all universities and university colleges in Norway for up to one year, according to the Act relating to Norwegian Universities and University Colleges, section 4-7 and 4-8 and Examination regulations section 14 and 15.	<input checked="" type="checkbox"/>
4.	I am/we are aware that all papers/assignments may be checked for plagiarism by a software assisted plagiarism check	<input checked="" type="checkbox"/>
5.	I am/we are aware that Molde University college will handle all cases of suspected cheating according to prevailing guidelines.	<input checked="" type="checkbox"/>
6.	I/we are aware of the University College`s rules and regulation for using sources	<input checked="" type="checkbox"/>

Publication agreement

ECTS credits: 15

Supervisor: Hans Fredrik Nordhaug

Agreement on electronic publication of master thesis

Author(s) have copyright to the thesis, including the exclusive right to publish the document (The Copyright Act §2).

All theses fulfilling the requirements will be registered and published in Brage HiM, with the approval of the author(s).

Theses with a confidentiality agreement will not be published.

I/we hereby give Molde University College the right to, free of charge, make the thesis available for electronic publication: yes no

Is there an agreement of confidentiality? yes no

(A supplementary confidentiality agreement must be filled in)

- If yes: **Can the thesis be online published when the period of confidentiality is expired?** yes no

Date: 30/05/2011

Contents

1	Introduction	8
1.1	Motivation	8
1.2	Report's structure	9
2	Website's description	10
2.1	Description	10
2.2	Software and programming languages used	11
2.3	Technical description	12
2.3.1	Front-end	12
2.3.2	Back-end	14
2.4	Database	14
2.4.1	Entity-relationship diagram	14
2.4.2	Logical diagram	20
2.5	Class diagram	25
3	Objectives	27
4	SQL injection	29
4.1	Description	29
4.2	Example	30

4.3	Issue solution	39
4.3.1	Prepared statements	39
4.3.2	Escape user input	41
4.3.3	Additional defenses	41
4.4	Conclusion	44
5	XSS (Cross Site Scripting)	45
5.1	Description	45
5.1.1	Rule #1: HTML escaping before inserting untrusted data into HTML context	46
5.1.2	Rule #2: Attribute escape before inserting untrusted data into HTML common attributes	47
5.1.3	Rule #3: Javascript escape before inserting untrusted data into HTML Javascript data values	48
5.1.4	Rule #4: CSS escape before inserting untrusted data into HTML style property values	48
5.1.5	Rule #5: URL escape before inserting untrusted data into HTML URL parameter values	49
5.1.6	Rule #0: never insert untrusted data except in allowed locations	49
5.2	Example	50
5.3	Solution	57
5.4	Conclusion	59
6	Broken authentication and session management	60
6.1	Description	60
6.1.1	Authentication	60
6.1.2	Session management	63
6.2	Examples	64
6.2.1	Password strength	65

6.2.2	Authentication responses	67
6.3	Issues solution	69
6.3.1	Password strength	69
6.3.2	Password recovery	75
6.3.3	Authentication responses	80
6.3.4	Account lockout	81
6.3.5	Ensure session ID's	83
6.3.6	Timeout	84
6.4	Conclusion	85
7	Insecure direct object references	87
7.1	Description	87
7.2	Example	87
7.3	Issue solution	88
7.4	Conclusion	90
8	CSRF (Cross-site request forgery)	91
8.1	Description	91
8.2	Example	91
8.3	Solution	93
8.4	Conclusion	95
9	Failure to restrict URL access	97
9.1	Description	97
9.2	Example	97
9.3	Solution	99
9.4	Conclusion	101
10	Insufficient transport layer protection	102

10.1 Description	102
10.2 Secure server design	103
10.3 Server certificate and protocol configuration	105
10.4 Conclusion	106
11 Conclusion	107
A Source code	109
A.1 Source code to check password strength and password verification	109
A.2 Password recovery source code	113
Bibliography	119
List of figures	123
List of tables	126
List of listings	127

Chapter 1

Introduction

1.1 Motivation

A family member wants to open a new business using new technologies, so he thought about to create an online shop. As IT students, we proposed to build the website, but he was worried about the security: how he can be secure that nobody will steal the data stored, that nobody attacks the website obtaining profit of that...

At this point, we thought that web security is a good point to study, because it includes almost all points of a current web: dynamic content, work with databases, user accounts with personal data, online payments...

Though we will define some concrete objectives related to web security in chapter 3, this report's main objective is to build features to make an existent website secure against web main weaknesses.

We know that, since the Internet is a complex system and it is in continuous change, it is impossible to build a hundred percent secure site. But we can try to enforce our security in order to difficult the access to it, and then only the expert people can succeed.

Therefore, this website will allow us study security issues (how they work and how can be exploited), implement the solutions and finally verify if those solutions are effective and avoid illegal access to the website.

1.2 Report's structure

Aside from introduction, this report structure is as described below.

First of all, it has a description of the website where is explained how it works, what can a user make and what is it for. It also explains how the website is made, with a technical explanation.

After that, it has a definition of the report's objectives, scilicet, issues that exists on the website and must be solved. That definition is given after a research along The Web, consulting security paperwork and websites.

Then, the work's most important part, that is issues' study inside the website, looking for the issues described in the previous point and maybe others unlisted. This part contents a technical description of the issues and examples of each vulnerability in the website. For each issue, it also explains the solution of the issue on the website and a conclusion.

Finally, the last point is a conclusion, describing the objectives achieved, a little summary and a personal valuation of the work.

To complete report's structure, we have to add that to write this report, we have used the following software:

- To write this report, OpenOffice.org Writer[1] and TexShop[2] have been used.
- To draw diagrams, OmniGraffle Professional[2] has been used.

Website's description

2.1 Description

As it has already been said, the website is an online shop where an user can make an account and then look for products along the site, to add them to the cart and after buy them; the user can also change his personal information and check his order's status.

But the website also has an administration section, where shop's owner can manage all over the shop, like adding, modifying or deleting products, support companies, users... Unfortunately, the shop doesn't have any accounting section, but it can be implemented in future releases.

The shop works with promotions: each product is inside one promotion, and promotions last for a concrete period of time. It is possible to include in future releases navigation by brand, namely, a client can buy products with discounts inside a promotion during a period of time, or always out of a promotion.

We will need to save some information into a database to have all these features. For each client, we would like to save an ID, his email and his password to access to the shop; his personal data, like his first name, his second name, his address and his phone number; and we would like to have his register date.

The shop will have some support companies that will provide it products and will deliver orders. From these companies we would like to save an ID, its name and address, and a contact email and phone and the first and the last

name of the contact person. For the carriers, we would also like to save its geographic area and its shipping cost.

Some of these companies provide products to the shop. From these products we would like to save its reference, name, brand, description and a picture; we would also like to save provider price, other's shops price and our shop's price, and this product stock on the shop.

Products are inside a promotion. From promotions we would like to save an ID, its name, its start and finish date and a picture.

We also want to save client's order data. From orders we would like to save its status, total amount, all dates: order date, shipping date and delivery date; another address, if the delivery address is different from the client's address, and comments about the order, if the clients want to do it.

Finally, it is necessary to say that to pay an order the shop uses an integration with PayPal, so all the information related to the order is sent to PayPal and they are who manage the collection.

2.2 Software and programming languages used

The website has been developed using free tools:

- As a server software, Apache[4] has been used.
- To present the website, HTML[5] has been used.
- As a dynamic programming language, PHP[6] has been used.
- As a database engine, MySQL[7] has been used, and for executing initial queries, phpMyAdmin[8] has been used though queries has been written before in a plain text file.
- To check forms, Javascript[9] has been used.
- To improve the administration section, AJAX[10] has been used.

All the code has been written as a plain text, without using any other support software. To write the code, Kate[11] and TextMate[12] have been used.

To test the website, three different browsers have been used: Safari[13], Mozilla Firefox[14] and Opera[15].

The website appearance is not beautiful because any style has been applied due it was a lot of work that did not belong to this report and it made the source code more complicated to understand. This way, everybody who wants to use the website can apply his own unique style.

2.3 Technical description

2.3.1 Front-end

This is the website's section that everybody can see, but we have to differentiate between common users and registered users. The use case diagram of this section is described in figure 2.1.

As we can see, a common user can only do the following actions:

- *View products on the shop*: to see if there is something that interests him, before he registers.
- *Register*: to become a registered user.

But a registered user can do more actions, as is described below:

- *Login*:to access to his account.
- *View products on the shop*: to search for products of his interest.
- *Add products to the chart*:so he can buy those products.
- *Buy*: to buy products in the chart
- *Manage orders*: a registered user can check his orders' status and cancel an order if it is still not prepared
- *Modify personal data*:to change some data as password, email, address...

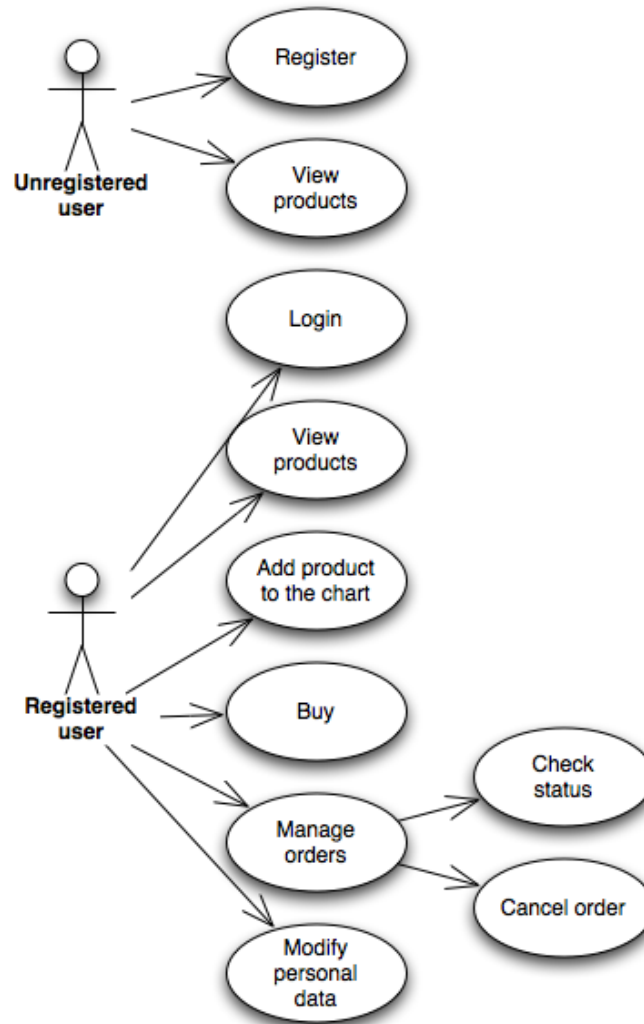


Figure 2.1: Front-end use case diagram

This way, the website uses less resources for unregistered users because, anyway, if someone wants to buy something, he has to be registered first, so the shop can have his name and address to send him the orders.

2.3.2 Back-end

This website section is reserved for shop's owner, and nobody else should have access to it. The use case diagram of this section is described in figure 2.2.

Actions that the administrator can do are:

- *Login*: to obtain access to the administration section.
- *Manage clients*: to add, modify, delete or view a client, if some of those actions are necessary.
- *Manage providers*: to add, modify, delete or view a provider, if some of those actions are necessary.
- *Manage carriers*: to add, modify, delete or view a carrier, if some of those actions are necessary.
- *Manage products*: to add, modify, delete or view a product, if some of those actions are necessary.
- *Manage orders*: to view, delete or modify the status of an order; an order can be deleted only if its status is received.
- *Manage promotions*: to add, modify, delete or view a promotion, if some of those actions are necessary, and to add or delete a product from a promotion.

2.4 Database

2.4.1 Entity-relationship diagram

The database structure is designed to cover all the actions described in the use case diagrams. The entity-relationship diagram is described in figure 2.3.

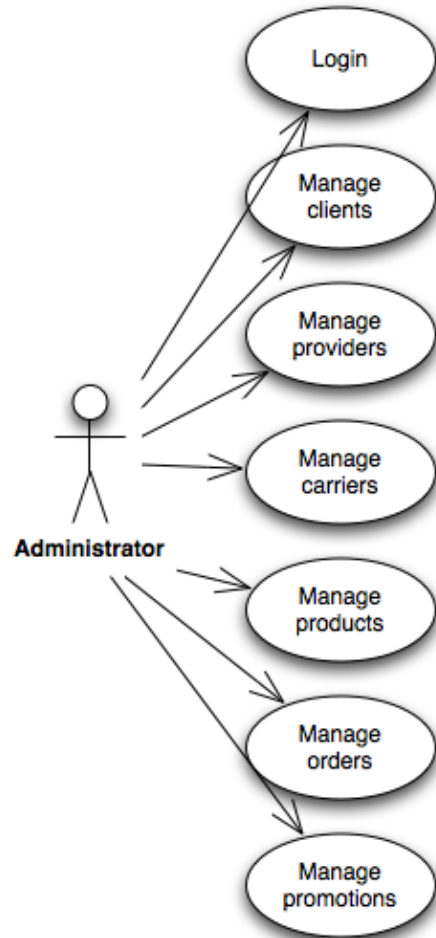


Figure 2.2: Back-end use case diagram

It is necessary to say that the entities only have their key attributes in order to make the diagram more readable.

Entities list

- Client: represents shop's clients. Its attributes are:
 - *ID*: it is client's ID; it is primary key and numeric type.
 - *Email*: it is client's contact email; it is alphanumeric type.
 - *Password*: it is client's password to access the shop; it is alphanumeric type.
 - *Register date*: it is client's register date on the shop; it is date type.
 - *First name*: it is client's first name; it is alphanumeric type.
 - *Last name*: it is client's last name; it is alphanumeric type.
 - *Address*: it is client's address; it is alphanumeric type.
 - *Postal code*: it is client's postal code; it is numeric type.
 - *Town*: it is client's town; it is alphanumeric type.
 - *Area*: it is client's area; it is alphanumeric type.
 - *Telephone*: it is client's phone number; it is numeric type.
- Order: represents client's orders. Its attributes are:
 - *ID*: it is order's ID; it is primary key and numeric type.
 - *Status*: it is order's status; it is numeric type and it can only be: 0 for received orders; 1 for prepared orders; 2 for sent orders; and 3 for delivered orders.
 - *Amount*: it is order's total amount; it is numeric type.
 - *Order date*: it is order's order date; it is date type.
 - *Shipping date*: it is order's shipping day; it is date type.
 - *Delivery date*: it is order's delivery date: it is date type.
 - *Address*: it is delivery's address, if it is different from the client's address; it is alphanumeric type.

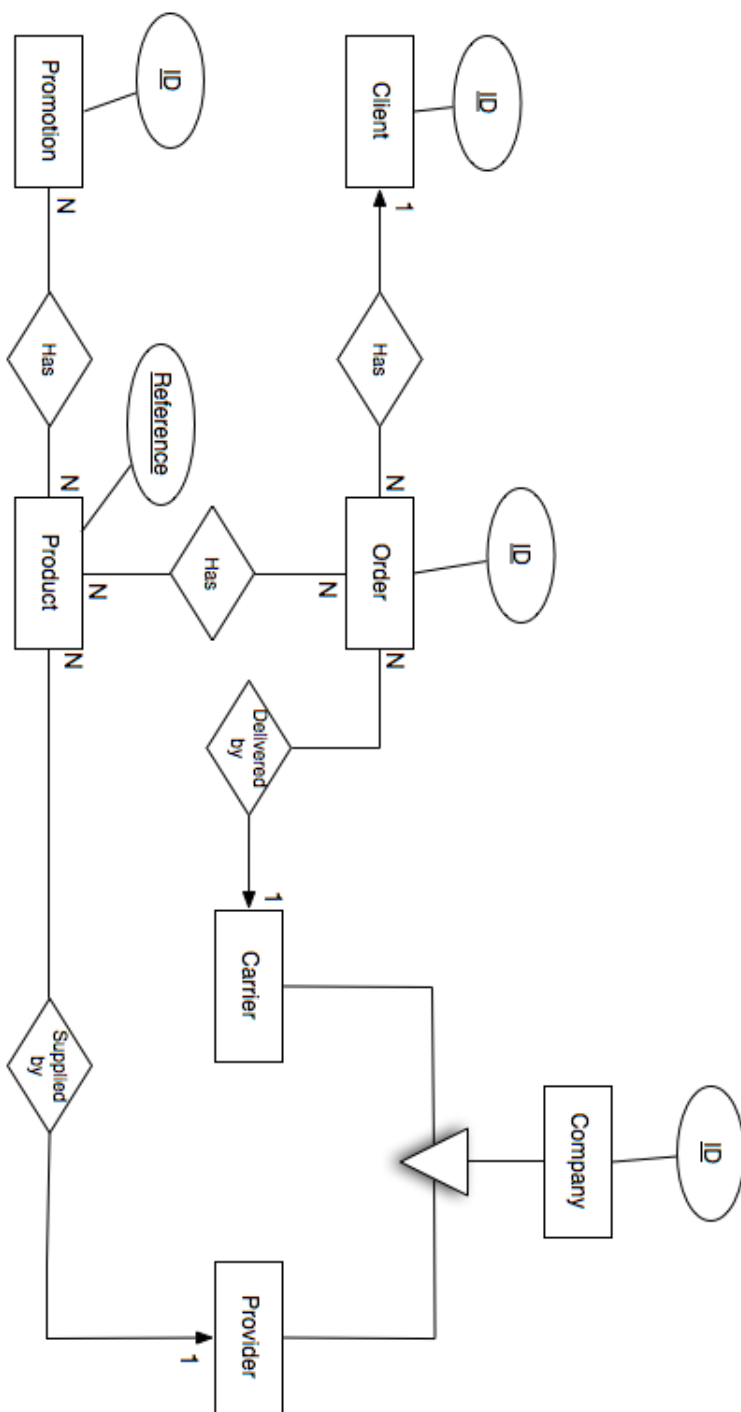


Figure 2.3: Database entity-relationship diagram

- *Postal code*: it is delivery's postal code, if it is different from the client's postal code; it is numeric type.
- *Town*: it is delivery's town, if it is different from the client's town; it is alphanumeric type.
- *Area*: it is delivery's area, if it is different from the client's area; it is alphanumeric type.
- *Comments*: it is order's comments, if user wants to make some; it is alphanumeric type.
- Support company: represents shop's support companies. Its attributes are:
 - *ID*: it is company's ID; it is primary key and numeric type.
 - *Name*: it is company's name; it is alphanumeric type.
 - *Address*: it is company's address; it is alphanumeric type.
 - *Postal code*: it is company's postal code; it is numeric type.
 - *Town*: it is company's town; it is alphanumeric type.
 - *Area*: it is company's area; it is alphanumeric type.
 - *Contact email*: it is company's contact email; it is alphanumeric type.
 - *Contact phone*: it is company's contact phone number; it is numeric type.
 - *Contact first name*: it is company's contact person's first name; it is alphanumeric type.
 - *Contact last name*: it is company's contact person's last name; it is alphanumeric type.

This entity has two specializations:

- *Provider*: represents provider's companies; it doesn't have attributes.
- *Carrier*: represents carrier's companies; its attributes are:
 - * *Geographic area*: it is company's geographic delivery area; it is alphanumeric type.

* *Shipping cost*: it is company's standard shipping cost; it is numeric type.

- Product: represents products to be sell in the shop. Its attributes are:
 - *Reference*: it is product's reference; it is primary key and alphanumeric type.
 - *Name*: it is product's name; it is alphanumeric type.
 - *Brand*: it is product's brand; it is alphanumeric type.
 - *Description*: it is product's description; it is alphanumeric type.
 - *Picture*: it is product's picture's path inside the server; it is alphanumeric.
 - *Price*: it is product's selling price; it is numeric.
 - *Shop price*: it is product's price on other shops; it is numeric.
 - *Stock*: it is product's current stock; it is numeric;
 - *Provider price*: it is product's provider price; it is numeric.
- Promotion: represents promotions. Its attributes are:
 - *ID*: it is promotion's ID; it is primary key and alphanumeric type.
 - *Name*: it is promotion's name; it is alphanumeric type.
 - *Start date*: it is promotion's start day; it is date type.
 - *Finish date*: it is promotion's finish day; it is date type.
 - *Picture*: it is promotion's picture's path inside the server; it is alphanumeric.

Interrelations list

- *Client – order*: 1 to N relationship; one client can have more than one order, but an order can only belong to one client.
- *Order – product*: N to N relationship; one product can belong to more than one order, and one order can have more than one product. It has an associate attribute, which is Amount, the amount of this product in this order.

- *Order – carrier*: 1 to N relationship; one order can only be delivered by one carrier, but one carrier can have more than one order to deliver.
- *Product – provider*: 1 to N relationship; one product can only be supplied by one provider, but one provider can supply more than one product.
- *Promotion – product*: N to N relationship; one product can be at more than one promotion (not at the same time), and one promotion can have more than one product.

2.4.2 Logical diagram

To get the correct structure to implement the database into the database engine, it is necessary to convert the entity-relationship diagram into a logical (or table) diagram. The shop's logical diagram is described in figure 2.4.

Tables list

- Client: represents shop's clients; this table comes from the client entity and its attributes are:
 - *ID*: it is client's ID; it is primary key and numeric type.
 - *Email*: it is client's contact email; it is character type.
 - *Password*: it is client's password to access to the shop; it is character type.
 - *Register_date*: it is client's register date on the shop; it is date type.
 - *First_name*: it is client's first name; it is character type.
 - *Last_name*: it is client's last name; it is character type.
 - *Address*: it is client's address; it is character type.
 - *Postal_code*: it is client's postal code; it is numeric type.
 - *Town*: it is client's town; it is character type.
 - *Area*: it is client's area; it is character type.

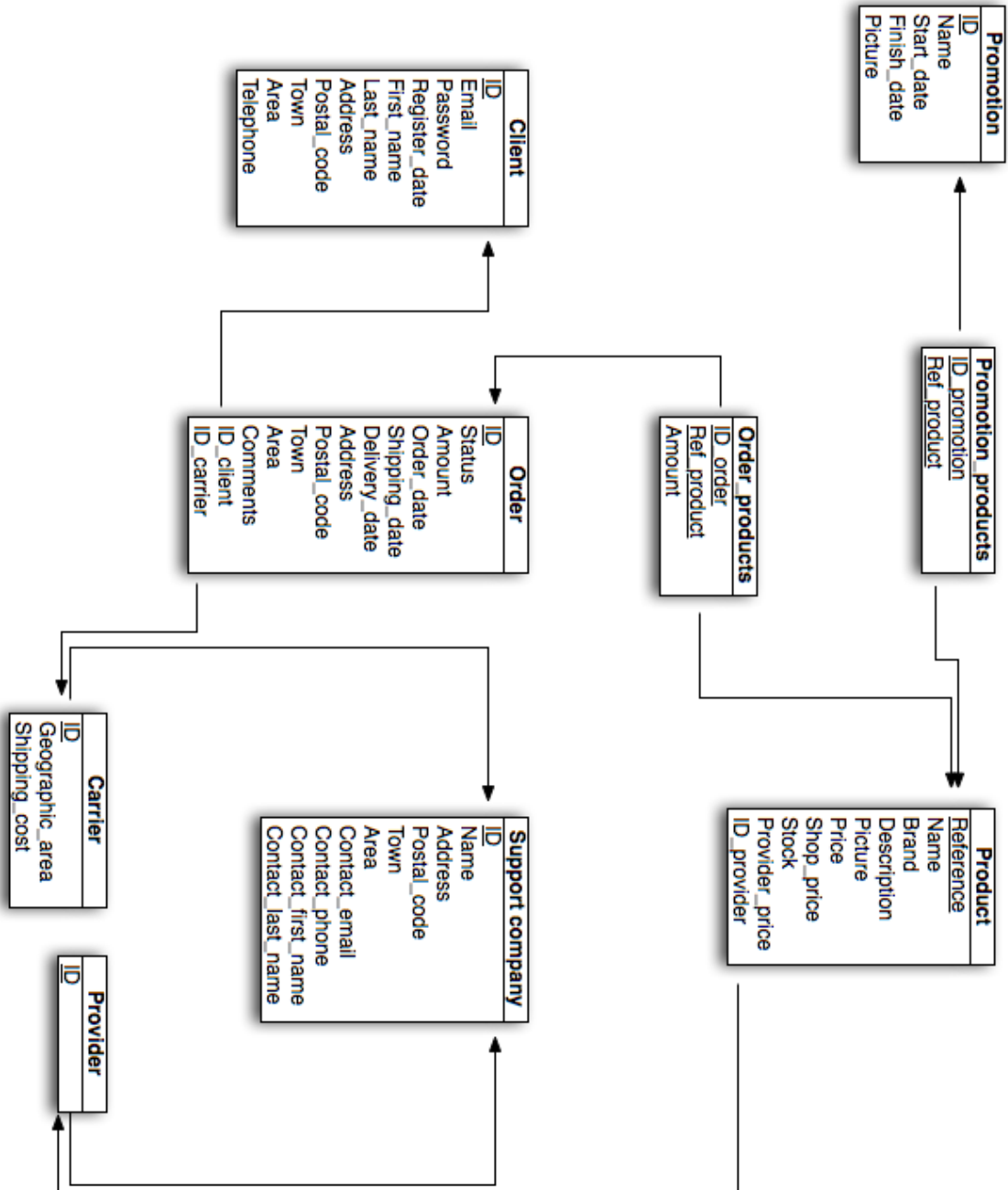


Figure 2.4: Database logical diagram

- *Telephone*: it is client's phone number; it is numeric type.
- Order: represents client's orders; this table comes from the order entity and its attributes are:
 - *ID*: it is order's ID; it is primary key and numeric type.
 - *Status*: it is order's status; it is numeric type and it can only be: 0 for received orders; 1 for prepared orders; 2 for sent orders; and 3 for delivered orders.
 - *Amount*: it is order's total amount; it is numeric type.
 - *Order_date*: it is order's order date; it is date type.
 - *Shipping_date*: it is order's shipping day; it is date type.
 - *Delivery_date*: it is order's delivery date: it is date type.
 - *Address*: it is delivery's address, if it is different from the client's address; it is character type.
 - *Postal_code*: it is delivery's postal code, if it is different from the client's postal code; it is numeric type.
 - *Town*: it is delivery's town, if it is different from the client's town; it is character type.
 - *Area*: it is delivery's area, if it is different from the client's area; it is character type.
 - *Comments*: it is order's comments, if user wants to make some; it is character type.
 - *ID_client*: it is buyer's ID; it is numeric type.
 - *ID_carrier*: it is carrier's ID for this order; it is numeric type.
- Support_company: represents the support companies; this table comes from the Support company entity and its attributes are:
 - *ID*: it is company's ID; it is primary key and numeric type.
 - *Name*: it is company's name; it is character type.
 - *Address*: it is company's address; it is character type.
 - *Postal_code*: it is company's postal code; it is numeric type.
 - *Town*: it is company's town; it is character type.

- *Area*: it is company's area; it is character type.
- *Contact_email*: it is company's contact email; it is character type.
- *Contact_phone*: it is company's contact phone number; it is numeric type.
- *Contact_first_name*: it is company's contact person's first name; it is character type.
- *Contact_last_name*: it is company's contact person's last name; it is character type.
- Provider: represents shop's providers; this table comes from the Provider entity, which is a specialization of the Support company entity; its single attribute is:
 - *ID*: it is provider's ID; it is numeric type.
- Carrier: represents carriers; this table comes from the Carrier entity, which is a specialization of the Support company entity; its attributes are:
 - *ID*: it is provider's ID; it is numeric type.
 - *Geographic_area*: it is company's geographic delivery area; it is character type.
 - *Shipping_cost*: it is company's standard shipping cost; it is numeric type.
- Product: represents the shop's products; this table comes from the Product entity and its attributes are:
 - *Reference*: it is product's reference; it is primary key and character type.
 - *Name*: it is product's name; it is character type.
 - *Brand*: it is product's brand; it is character type.
 - *Description*: it is product's description; it is character type.
 - *Picture*: it is product's picture's path inside the server; it is character.
 - *Price*: it is product's selling price; it is numeric.

- *Shop_price*: it is product's price on other shops; it is numeric.
- *Stock*: it is product's current stock; it is numeric;
- *Provider_price*: it is product's provider price; it is numeric.
- *ID_provider*: it is product's provider ID; it is numeric.
- Promotion: represents the shop's promotions; this table comes from the Promotion entity and its attributes are:
 - *ID*: it is promotion's ID; it is primary key and character type.
 - *Name*: it is promotion's name; it is character type.
 - *Start_date*: it is promotion's start day; it is date type.
 - *Finish_date*: it is promotion's finish day; it is date type.
 - *Picture*: it is promotion's picture's path inside the server; it is character.
- Order_products: represents the relationship between order and products, namely, products that are in a order; this table comes from the interrelation N to N between the Order and the Product entities and its attributes are:
 - *ID_order*: it is order's ID in the relationship; it is primary key and numeric type.
 - *Ref_product*: it is product's ID in the relationship; it is primary key and character type.
 - *Amount*: it is product amount in this order; it is numeric type.
- Promotion_products: represents the relationship between promotions and products, namely, products that are in a promotion; this table comes from the interrelation N to N between the Promotion and the Product entities and its attributes are:
 - *ID_promotion*: it is promotion's ID in the relationship; it is primary key and numeric type.
 - *Ref_product*: it is product's ID in the relationship; it is primary key and character type.

2.5 Class diagram

We can see in figure 2.5 the class diagram for the website. The first that we can see is that there is an abstract object called *Collection*. The function of this object is to save a list of objects of the same type, each one in its own object, in order to present them all together. For example, when the administrator wants to edit a client, with these objects he can obtain all the clients in the shop.

Then we can see that, except *Chart*, all the other objects are the same as we saw in the database description. The relationships between these objects are the same as described in that section, so we do not need to explain them now. All these objects have functions to interact with the data stored in the database: to read, to modify, to insert and to delete, and they have their own functions.

Finally, the last object, *Chart*, represents the shop's chart. A chart belongs to an user, and it can not exist without the user. And it has products in it, and it can not exist without products.

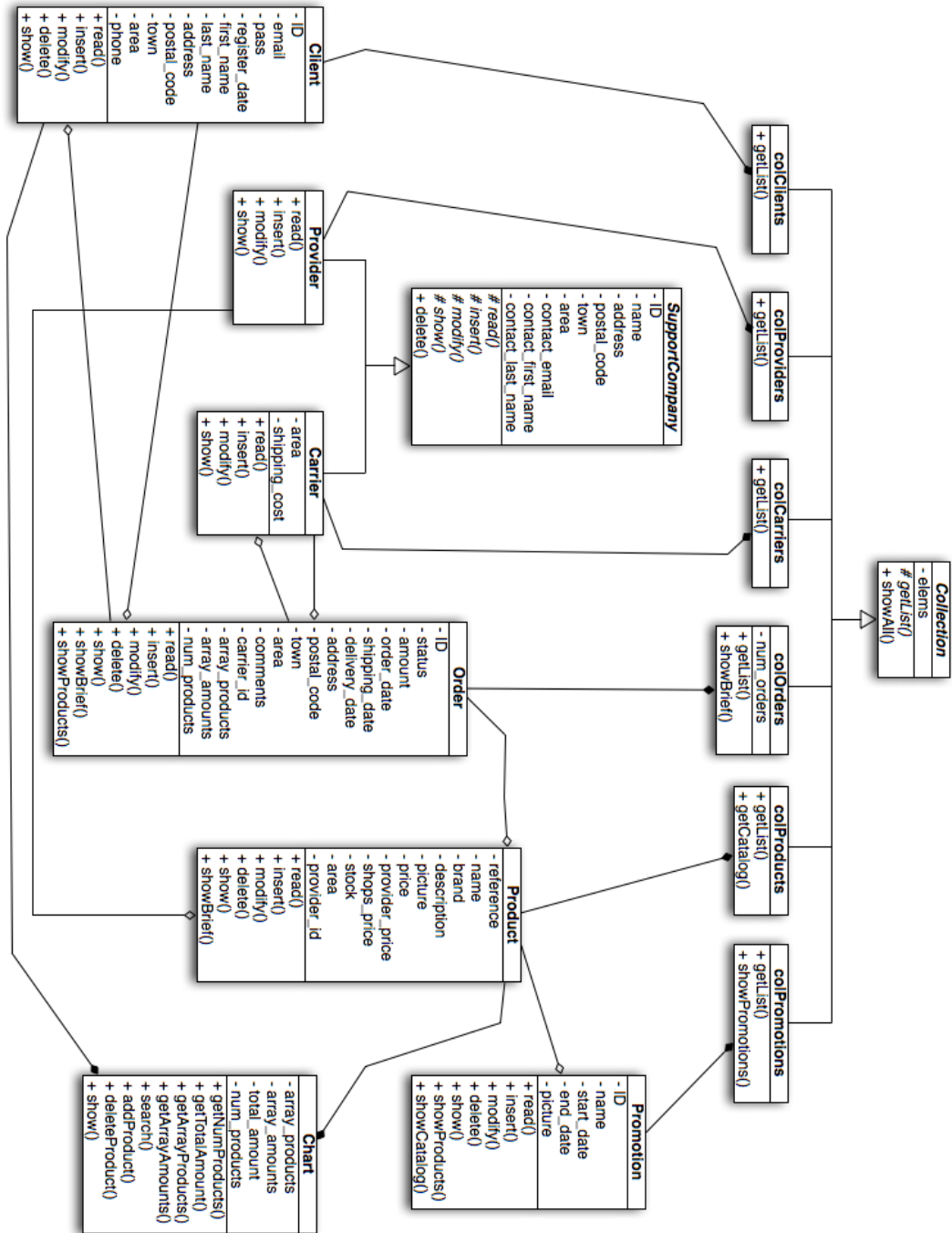


Figure 2.5: Class diagram of the website

Chapter 3

Objectives

Though The Web is a changing environment and because of that it is difficult to establish the main security problems, the OWASP¹[16] develops a ranking with those problems each three years.

The OWASP is a foundation that collects and produces documentation about security issues, and several professionals in this area work for that foundation, that is open, scilicet, everybody can collaborate with it and all its information is open source.

So, using the OWASP 2010 ranking is a good starting point to establish this report objectives due to a lot of experts collaborate with it and they group and rate the issues.

The OWASP top 10 application security risks are described in table 3.1. The ranking is ordered from the most common to the less common issue.

In the next chapters we will study these issues on the website and we will solve them. However, we will not study the following objectives:

- *Security misconfiguration*: this issue solution consists of keeping all the software, libraries used... updated, and we will not study it because it is not really related with web development and it depends on the platform used.
- *Insecure cryptographic storage*: this issue solution consists of encrypting the data in the database. We will not study it because the data that has to be encrypted is different depending on the country where the

¹Open Web Application Security Project

database is hosted, and because how to do it depends on the platform and the database engine used.

- *Unvalidated redirects and forwards*: this issue solution consists of avoiding using redirects in the URL. We will not study it because it does not exist in the webpage.

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

Table 3.1: Objectives table

Chapter 4

SQL injection

4.1 Description

SQL injection is one of the most important issues in web security, because it is really common along the web, it is really easy to exploit and attackers can get access to the database's critical data, which contains attractiveness data.

This attack occurs when programmers create dynamic database queries that requires user input. A malicious user can exploit it introducing SQL code in the data inputs, resulting in a new query different from the original one, that has bad intentions. The most used places to inject SQL code is on forms and on URLs.

This kind of attack requires to know how the database is designed, but it is possible to perform first a not successful SQL injection attack, so the database engine will show an error message with the affected tables and then a malicious user will know something about the database. Repeating this attack several times with different inputs will let the malicious user know how the database is designed.

With this attack, it is possible from to obtain critical information from the database to erase it or to gain access to the website with another user account. Those things are really critical and should not be possible to perform them.

In the next section we will see a concrete example of this vulnerability on the website.

4.2 Example

We will see a concrete example of SQL injection in our website. The page affected in this case is `user_login.php`, that let users to access to their accounts.

This page has two forms as we can see in figure 4.1¹, one to introduce the email and another one to introduce the password. When the accept button is pressed, it redirects to the same page and the input is saved in the super-global PHP variable `$_POST`. The email is saved in `$_POST['email']` and the password in `$_POST['pass']`.

The source code for the validation process is as follows:

```

$db = new AuxDB();
$db->connect();

$sql = "select *
from CLIENT
where EMAIL = '" . $_POST["mail"] . "' and
PASSWORD = '" . $_POST["pass"] . "'";

$rst = $db->executeSQL($sql);
if($rst != false) {
    $row = $db->nextRow($rst);

    $user = new Client();
    $user->read(null, $row['EMAIL']);
    $_SESSION["user"] = $user;
    $_SESSION["login"] = 1;

    echo "<h2>You have successfully logged into your account, "
        . $_SESSION["user"]->getName() . "</h2>";

    <p>You will be redirected to your personal
    menu in 5 seconds</p>;

    echo "<meta http-equiv='Refresh' content='5;URL=user.php' />";
} else {
    echo "<h2>Email address or password are wrong.</h2>";
}

```

¹It is necessary to say that in order to present the pictures, the password field is treated as a normal text field

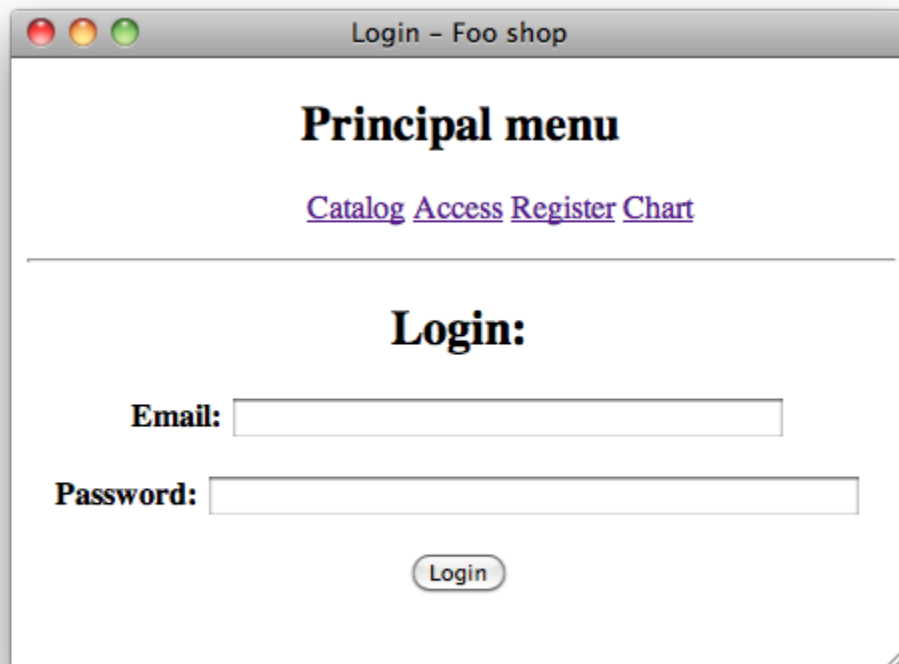


Figure 4.1: Website login page

```
<p><a href='user_login.php'>Try again</a></p>";
}
```

Listing 4.1: Login check source code

As we can see, the sql query to check if the email and the password are correct is stored in the `$sql` variable, and is

```
$sql = "select *
from CLIENT
where EMAIL = '" . $_POST["email"] . "' and
PASSWORD = '" . $_POST["pass"] . "'";
```

Listing 4.2: Login check query source code

In a normal case, if an user with the email `test@test.com` and the password `123456` exists and he tries to access to his account, the query will be as the programmer wants:

```
select *
from CLIENT
where EMAIL = 'test@test.com' and PASSWORD = '123456';
```

Listing 4.3: Normal query

But imagine an attacker who wants to gain access with another user's account. We have two scenarios: if the malicious user knows an existing email in the database, or if he does not know any existing email.

In the first case, the attacker knows that the email account exists in the database, so he can try to avoid the password verification. He can write as it is shown in figure 4.2, so the query will be like:

```
select *
from CLIENT
where EMAIL = 'test@test.com' AND /* ' and PASSWORD = '*/ ''='';
```

Listing 4.4: Case 1: attack knowing an email address

All written between `/*` and `*/` is a comment, so it does not count to the query, and `" = "` is always true (empty equals empty), so the attacker avoid the password verification and he obtains access as the user `test@test.com`, as we can see in figure 4.3.

To complete the information, the query that the server will execute is:

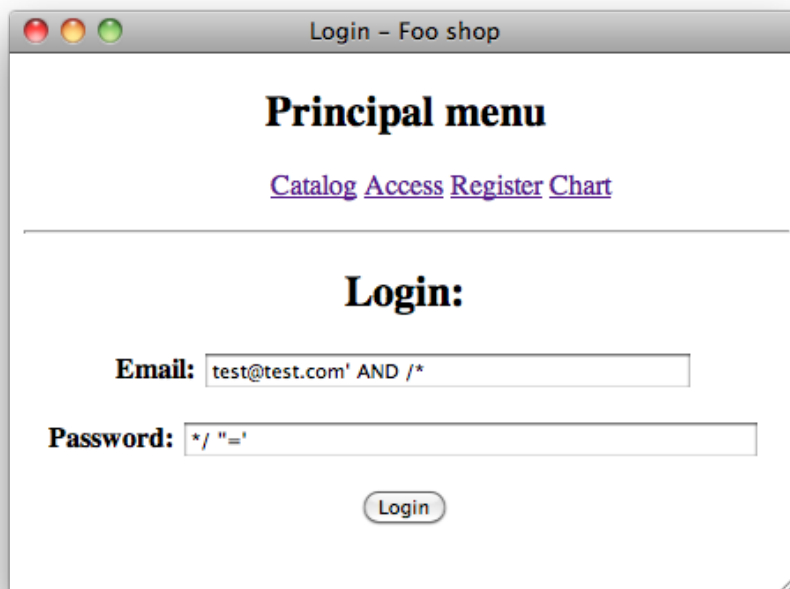


Figure 4.2: Example 1: data introduced

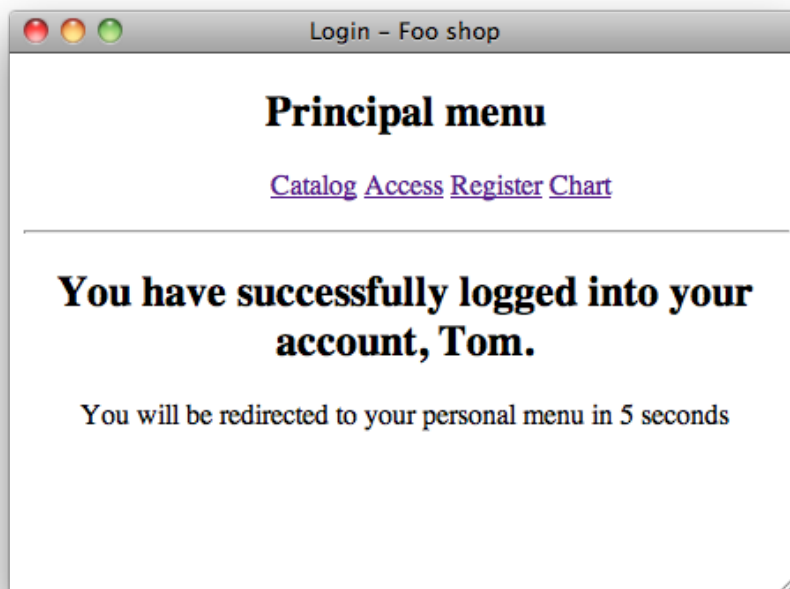


Figure 4.3: Example 1: login succesful

```
select *  
from CLIENT  
where EMAIL = 'test@test.com' AND ''='';
```

Listing 4.5: Case 1: executed query

Imagine now that the attacker does not know any email that exists in the database. In this case, he has to avoid both verifications, for the email and for the password. Changing some information in the input forms, as is shown in figure 4.4, he can obtain access to the website as an arbitrary user. Result query will be:

```
select *  
from CLIENT  
where EMAIL = 'foo@foo.com' or ''=''' and PASSWORD = '' OR ''=''';
```

Listing 4.6: Case 2: attack unknowing any data

In this case, the attacker needs that the AND's both sides inside the WHERE clause are true, and the attacker obtains it using an OR. In the left side, the email can exist or not in the database, but it is sure that empty equals empty, so this side will be true. It is similar to the right side: the password can be empty, but empty, for sure, equals empty, so this side will be true too. This way, if the given email does not exist in the database, the query will return all records, and the attacker will access as the first user in the list, as it is shown in figure 4.5; if the given email exists in the database, the attacker will get access as that email's user, so this is an alternative method to obtain access as a known user in the database.

It is necessary to say that in this case, as the website is written, it is not possible to get the database information using wrong queries as it was said in the description, because the PHP object that connects with the database only shows a generic error, without saying details.

This object uses mysql PHP extension, that does not allow to execute multiple queries, which is safer because does not let an attacker to execute more than one query, that can be dangerous too. Let's do an hypothetical example.

Imagine that an attacker write an email and a password as shown in figure 4.6. The resultant query will be:

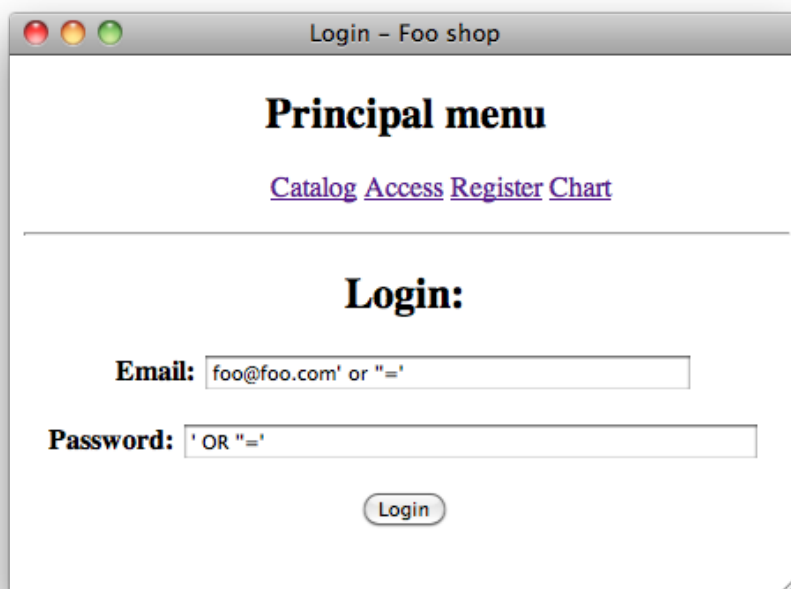


Figure 4.4: Example 2: data introduced

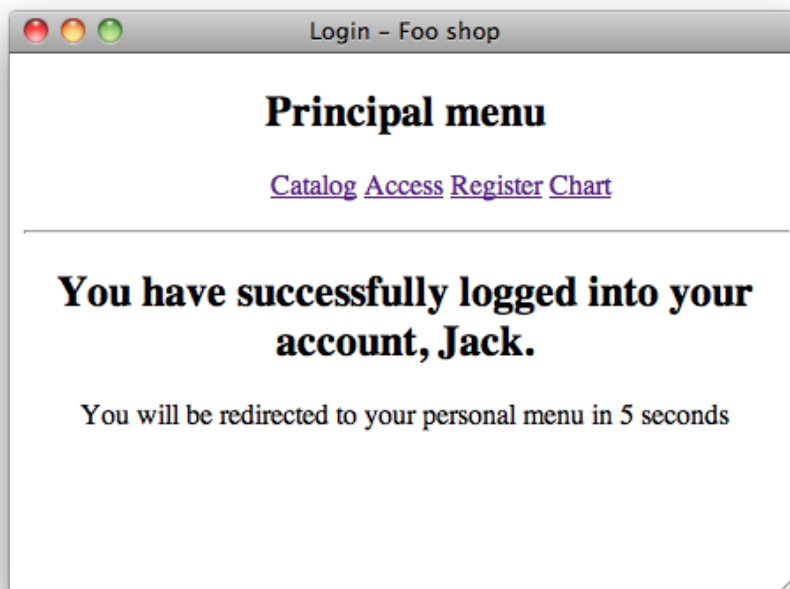


Figure 4.5: Example 2: login succesful

```
select *
from CLIENT
where EMAIL = 'foo@foo.com' and PASSWORD = '123456';

delete
from client
where 1 or username='';
```

Listing 4.7: Case 3: hypothetical attack deleting data

If it were possible to execute multiple queries, this access would delete all the data from the table client, regardless if the first query is successful or not.

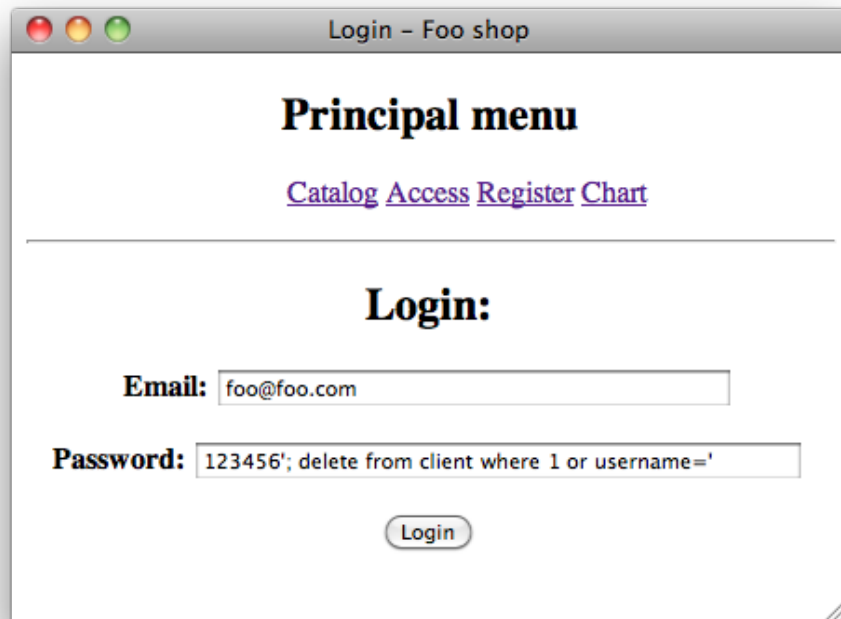


Figure 4.6: Example 3: data introduced

4.3 Issue solution

Although SQL injection is one of the most important issues in web security, it is really easy to prevent this attack, so all web developers should prevent their websites against it.

There are different techniques to prevent SQL injection, but we only studied two of them that are enough to prevent SQL injection attacks. We also studied some additional techniques that intensify the security to the techniques used.

4.3.1 Prepared statements

This technique, also called parametrized queries, consists of, first, define the SQL query and then pass in each parameter to the query. This way, the database can distinguish between the query and the input data, no matter what is that input data.

So, an attacker can not change the query because it is stored before, and all SQL commands that he writes as a input will be treated as a string.

We implemented this solution in the webpage, although we had to write a new database connection object to make it work with parametrized queries, because it needs a different driver to connect to the database, PDO instead of mysql. This solution is really effective and it avoids all the attacks described in section 4.2.

We can try to make one attack of that section, the second one for example, to see how the defense works. We introduce the same data as it is shown in figure 4.4. But now, instead of gain access as another user, we are rejected, as it is shown in figure 4.7. Let's see how the defense worked taking a look at the executed query:

```
select *
from CLIENT
where EMAIL = 'foo@foo.com\' or \'\'=\'\'
and PASSWORD = '\ ' OR \'\'=\'\';
```

Listing 4.8: Rejected attack

Database searched an email that is equal to `foo@foo.com' or ''='` with a

password equals to ' OR ''=' and it did not find a match, so the webpage denied the access.

Finally, the simplified code for this part is:

```
$sql = "select *  
from CLIENT  
where EMAIL = ? and PASSWORD = ?";  
  
$privateStatement->bindParam(1, $_POST["mail"], PDO::PARAM_STR);  
$privateStatement->bindParam(2, $_POST["pass"], PDO::PARAM_STR);  
  
$privateStatement->execute();
```

Listing 4.9: Prepared statements source code

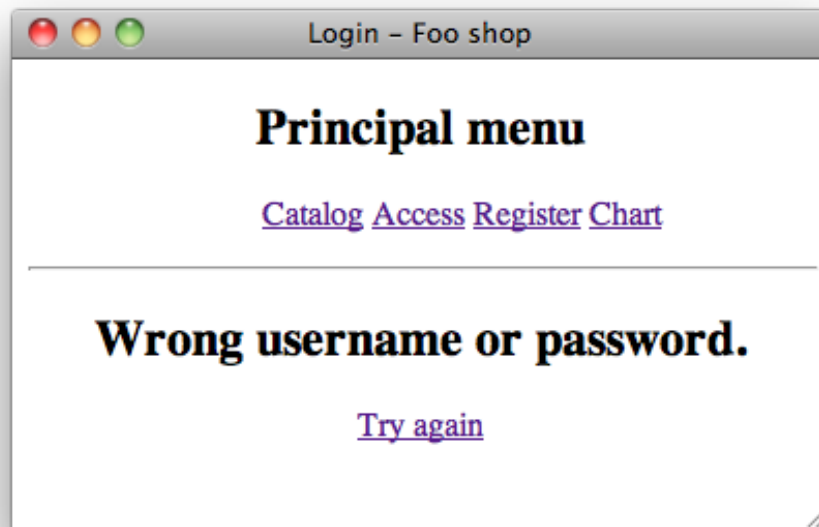


Figure 4.7: Rejected attack

4.3.2 Escape user input

This technique consists of escape all the critical characters in the user data input. In SQL injection attacks is usual to use characters like ' or \ to close the normal data input and append the SQL malicious code behind it, but this technique prevents against it, because it escapes these characters, making them strings, so the result is the same as using prepared statements.

We implemented this solution in the webpage too, but using the original object to connect to the database, because the use of the escaping function is not compatible with that driver, PDO. The solution works too, avoiding illegal accesses to the website.

If we try to make the attack shown in figure 4.2, we will get rejected as is shown in figure 4.7. The resultant query is the same as shown in listing 4.8, but the source code is not so different from the original and, in fact, the only changed line is the one where query definition is, and it is as follows:

```
$sql = "select *
from CLIENT
where EMAIL =
'" . mysql_real_escape_string($_POST["mail"]) . "'
and PASSWORD =
'" . mysql_real_escape_string($_POST["pass"]) . "'";
```

Listing 4.10: Escape user input source code

As we can see, the `mysql_real_escape_string()` function is which escapes the input data.

4.3.3 Additional defenses

Using the techniques described before are enough to prevent SQL injection attacks, but we can add some other defenses to make the website more secure.

Least privileges

This is not a technique in itself, but it can prevent against SQL attacks and other issues. It consists of use for web applications a database user with less privileges or making some database users that can only access to some

database tables.

In the first case, for example, if we only need to read data from the database, to show the catalog, it is not necessary to have other privileges as writing or removing data from the database.

In the second case, we can imagine that we have in the same database website's data and shop's accounting data. In this case, we should use two different database users: one who has access to the database and another one who has access to the accounting side too.

We did not change any user from the database because it does not belong to web security report, but to the database security, and we will come over later, in a different issue.

Checking forms

It is always better to prevent attacks from the beginning, from user's side, so we can check all the data input before it is sent to the web server.

If we continue with the same example, `user_login.php`, we can make some changes in the input form: limit the input size and check that the input data is really the kind of data that we want.

Normally, SQL injection attacks require a lot of characters, and the email address is shorter than 40 characters and the password shorter than 16, so we can limit these fields to those values, so the SQL code has less probabilities to fit in that space. The HTML code to obtain that is:

```
<p><strong>Email: </strong>
  <input name='mail' type='text' id='mail' maxlength='40' />
</p>
<p><strong>Password: </strong>
  <input name='pass' type='password' id='pass' maxlength='16' />
</p>
```

Listing 4.11: Limit input data source code

In the second case, we can check with a regular expression using Javascript that the input data is really an email, and not any other thing, so we avoid that an attacker introduces SQL code. We can see it in figure 4.8, and the javascript code is:

```

function formValidator() {
    var mail = document.getElementById('mail');
    if (isMail(mail, "Wrong email address")) {
        return true;
    }
    return false;
}

function isMail(elem, helperMsg) {
    var emailFilter = /^[a-zA-Z0-9_\. \-]+\@(([a-zA-Z0-9 \-]
+\.)+([a-zA-Z0-9]{2,4})+)$/;
    if (!(emailFilter.test(elem.value))) {
        alert(helperMsg);
        return false;
    }
    else {
        return true;
    }
}
}

```

Listing 4.12: Check input data source code

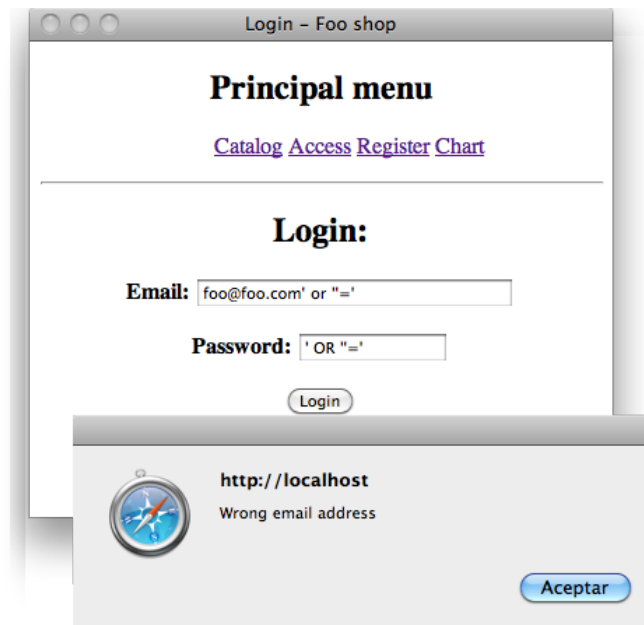


Figure 4.8: Checking wrong email address

4.4 Conclusion

As we could see, SQL injection attacks are really dangerous due to it is possible to do several malicious actions like extract critical data or erase all the data in a table or in the whole database. And this weakness is really extended along the Internet, so malicious users can perform much damage.

We could see that actually it is not difficult to prevent this kind of attack, so all web designers should write their websites avoiding it.

Expert people from OWASP[17] recommend to use prepared statements technique because it is the safest one, so if someone starts to write a website with database queries, he should use that technique.

But if the website is already written without using prepared statements, it is faster, less complex and really secure to use the escape user input technique.

And it is recommended to use with both techniques the additional defenses: to use a good user privilege policy in the database and to check all the input data. With that combination, website will be safe against SQL injection attacks.

XSS (Cross Site Scripting)

5.1 Description

As SQL injection, XSS is a really important web issue, due to an attacker can make another user to do things without knowledge, such as password extraction, phishing... It usually does not work automatically, but requires user interaction.

Also as SQL injection, XSS works with the user input data but it consists of introduce some malicious code that will be stored in the database, so when someone view that information, the code will be executed. This code is normally Javascript inserted in HTML, and it is executed according to some event, as loading an image, for example.

This attack requires to break out a data context¹ and to switch into a code context, using special characters in the used interpreter. The interpreter in these attacks is the browser, and the malicious code is inserted into the HTML code.

We can find two types of XSS:

- Injection UP: is the most common way, and consists of close the current context and start a new context. For example, the attacker closes first a context with `\>` and then he starts a new context with `<script>` to start the malicious context.
- Injection DOWN: is the less common way, and consists of insert code

¹A context is, for example, `<div>Data context</div>`

without closing the current context. For example, the attacker changes `` into ``, and he does not have to break the current context.

But there is another way to inject the code without changing of context, and it is simply write the code directly in the input forms, because there are a lot of sites that do not verify input data.

As we can see, XSS is really potential, and it allows attackers to perform a lot of malicious actions, and nowadays, when all sites are more interconnected, all untrusted data² should be verified to not harm other sites.

To prevent XSS attacks, we have to consider that all input data is malicious and, therefore, we must validate all the input.

Traditionally, input validation has been the preferred technique for check untrusted data, but actually it is not a complete solution. In first place, because the validation is done when the input data is received, and at that moment we do not know which characters will be significant in the target place. In second place, because we need to write all characters, even if they are potentially harmful. For example, the character `'` is special in SQL, but we should let use this character because there are some names with it, like O'Brien.

However, input validation is recommended and is a good complement for the prevention techniques that we will describe below. The generic examples showed are extracted from OWASP[17].

Those techniques are rules, and it is not necessary to add all of them to the website. According to OWASP, a website will be secure using only rules 1 and 2.

5.1.1 Rule #1: HTML escaping before inserting untrusted data into HTML context

This rule is for when the web developer wants to put untrusted data in somewhere in the HTML body. The tags included are normal tags, like `div`,

²Untrusted data is all data that comes from inputs and that is not verified

p, td... Of course, this rule is not sufficient, because there are other HTML contexts, that will need another rules.

Generic examples of this rule are:

```
<body>...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...</body>
<div>...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...</div>
```

Listing 5.1: Rule #1 generic examples

The character conversion is:

- & (ampersand) becomes &
- " (double quote) becomes "
- ' (single quote) becomes '
- <(less than) becomes <
- >(greater than) becomes >

The PHP function `htmlspecialchars` converts special characters into HTML entities.

5.1.2 Rule #2: Attribute escape before inserting untrusted data into HTML common attributes

This rule is for putting untrusted data into common HTML attributes, like `width`, `name`... This rule should not be used for complex attributes, as `href`, `src` or any of the event handler like `onmouseover`. For the event handler, use Rule #3.

Generic examples of this rule are:

```
<div attr=...ESCAPE UNTRUSTED DATA BEFORE PUTTING
  HERE...>content</div>      inside Unquoted attribute

<div attr='...ESCAPE UNTRUSTED DATA BEFORE PUTTING
  HERE...'>content</div>    inside single quoted attribute

<div attr="...ESCAPE UNTRUSTED DATA BEFORE PUTTING
  HERE...">content</div>  inside double quoted attribute
```

Listing 5.2: Rule #2 generic examples

5.1.3 Rule #3: Javascript escape before inserting untrusted data into HTML Javascript data values

This rule is for putting untrusted data in Javascript events handler that are specified in some HTML elements. The only safe place to put data in this case is on quoted data value; including it in another location is very dangerous.

Generic examples are:

```
<script>alert ('...ESCAPE UNTRUSTED DATA BEFORE PUTTING
  HERE...')</script>      inside a quoted string

<script>x='...ESCAPE UNTRUSTED DATA BEFORE PUTTING
  HERE...'</script>      one side of a quoted expression

<div onmouseover="x='...ESCAPE UNTRUSTED DATA BEFORE PUTTING
  HERE...'"></div>      inside quoted event handler
```

Listing 5.3: Rule #3 generic examples

5.1.4 Rule #4: CSS escape before inserting untrusted data into HTML style property values

This rule is for when the web developer wants to put untrusted data into a stylesheet or a style tag. CSS can be used for several attacks. The only safe place to put untrusted data is in a property value.

Generic examples are:


```
<style>selector { property: ...ESCAPE UNTRUSTED DATA BEFORE  
  PUTTING HERE...; } </style>      property value  
  
<style>selector { property: "...ESCAPE UNTRUSTED DATA BEFORE  
  PUTTING HERE..."; } </style>    property value  
  
<span style="property: ...ESCAPE UNTRUSTED DATA BEFORE PUTTING  
  HERE...">text</style>         property value
```

Listing 5.4: Rule #4 generic examples

5.1.5 Rule #5: URL escape before inserting untrusted data into HTML URL parameter values

This rule is for when web developer wants to put untrusted data into an HTTP GET parameter value.

A generic example is:

```
<a href="http://www.somesite.com?test=...ESCAPE UNTRUSTED DATA  
  BEFORE PUTTING HERE...">link</a >
```

Listing 5.5: Rule #5 generic example

5.1.6 Rule #0: never insert untrusted data except in allowed locations

This rule is to deny all, namely, put untrusted data only in the slots defined in Rule #1 through Rule #5. The reason for this rule is because there are several strange contexts in HTML that makes difficult to know the list of escaping rules.

So, generic examples are:

<code><script>...NEVER PUT UNTRUSTED DATA HERE...</script></code>	directly in a script
<code><!--...NEVER PUT UNTRUSTED DATA HERE...--></code>	inside an HTML comment
<code><div ...NEVER PUT UNTRUSTED DATA HERE...= test /></code>	in an attribute name
<code><NEVER PUT UNTRUSTED DATA HERE... href="/test" /></code>	in a tag name

Listing 5.6: Rule #0 generic examples

5.2 Example

We will see an example of XSS in our website. The affected pages in this case are `./admin/login.php`, that allows shop's owner to manage all the topics related with the shop, `prepayment.php`, that allows clients to add some information about the delivery, and `./admin/view_order.php`, that allows shop's owner to view all orders made in the shop by clients.

`prepayment.php`, as we can see in figure 5.1, has 4 text inputs to add delivery address if it is different from the user's address, and another input to add some information about the delivery, such as delivery's preferred time; it also has a drop-down list to select the preferred carrier.

`view_order.php` works with AJAX, and has a drop-down list to select the order's status; once it is selected, another drop-down list is charged with all the orders with the selected status. When an order is selected, all the data about the order is charged in the page. We can see it in figure 5.2

`login.php`, has two forms as we can see in figure 5.3, one to introduce the email and another one to introduce the password. Once the data has been introduced and the login button pressed, it redirects to the same page and verifies that the introduced data is right.

Imagine an attacker that wants to obtain the administrator username and password. He can use XSS to get them.

First of all, he has to make an order buying a product. It does not matter,

The screenshot shows a web browser window with the title "Información del pedido - Foo shop". The address bar contains "http://localhost/~ruben/eshop/prepayment.php" and a search bar with "Google". The page content includes a navigation menu with links for "Catalog", "My account", "Close session", and "Chart". The main heading is "Add some information about your order:". Below this, there are three sections: "Address" with input fields for "Address:", "Postal code:", "Town:", and "Area:"; "Comments" with a large text area; and "Carrier" with a dropdown menu labeled "Choose carrier:" and "TDN". At the bottom, there is an "Accept" button.

Figure 5.1: Prepayment page

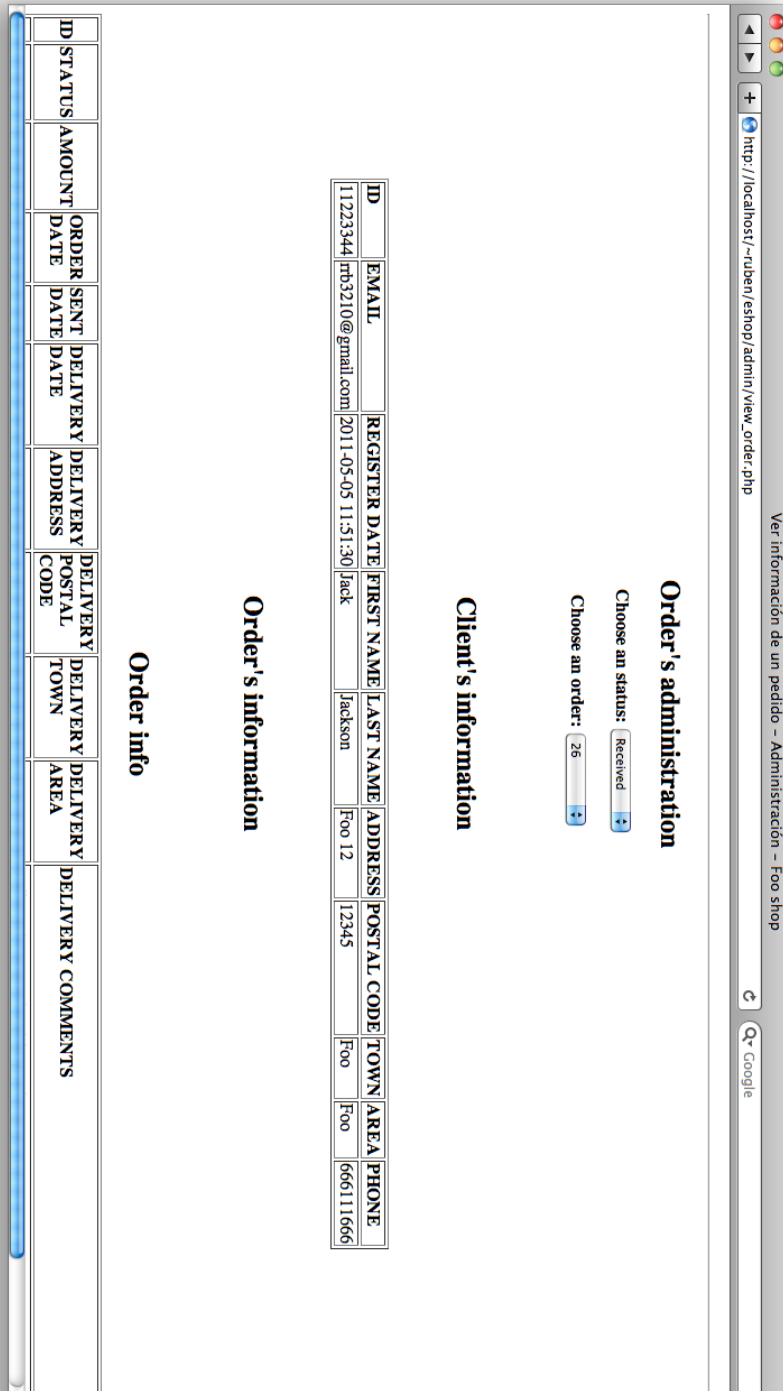


Figure 5.2: View order page

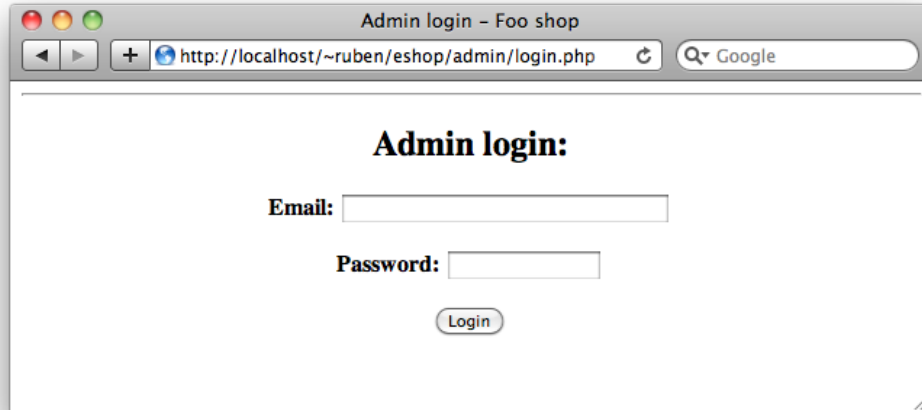


Figure 5.3: Admin login page

because he can delete the order later. When he arrives to `prepayment.php`, he writes on comments a type of injection down, as we can see in figure 5.4:

```
Delivery between 5 and 8 pm <img src='http://localhost/~ruben/eshop/imagenes/white.jpg' onload='window.location="http://localhost/~ruben/eshop/hack/login.php"' />
```

Listing 5.7: Inserted malicious comment

If we analyze this line, we can see mainly two parts, a text to make more credible the attack, and some code. This code loads an image, that in this case is just a white pixel, the same color as the background, and, when the image is loaded, it redirects automatically to another page, that is an administrator login page fake copy, with a message that the session expired. When somebody tries to view this order's comments, he will be redirected to that login page, which code is as follows:

```

<?
if(isset($_POST["Submit"])) {
    //Process data
    echo "<meta http-equiv='Refresh' content=
        '0;URL=http://localhost/~ruben/eshop/admin/pedidos.php'
        />";
} else {
?>
<html>
  <head>
  <title>Admin login - Foo shop</title>
  </head>
  <body>
<center>
<HR width=100% align='center'>
  <h1>Session timeout</h1>
  <h2>Admin login:</h2>
  <form method='post' action='login.php'>
  <p><strong>Email: </strong>
  <input name='mail' type='text' id='mail' maxlength='40'
    size='40' />
  </p>
  <p><strong>Password: </strong>
  <input name='pass' type='password' id='pass' maxlength='16'
    size='16' />
  </p>

  <p><input class='boton' type='Submit' name='Submit'
    value='Login' /></p>
</form>
</body>
</html>
<?
}

```

Listing 5.8: Fake login page source code

Once he finishes writing it, he follows the steps and he pays as if he were a normal user. After that, he just has to wait.

What happens now? When the shop's owner checks for new orders, he will access to this order and, suddenly, he will be redirected to the administrator login page. But this login page, though it looks the same as the original page, it is not the same, as we can check in the URL, in figure 5.5. The shop's owner will think that his session expired, so he will write his username and

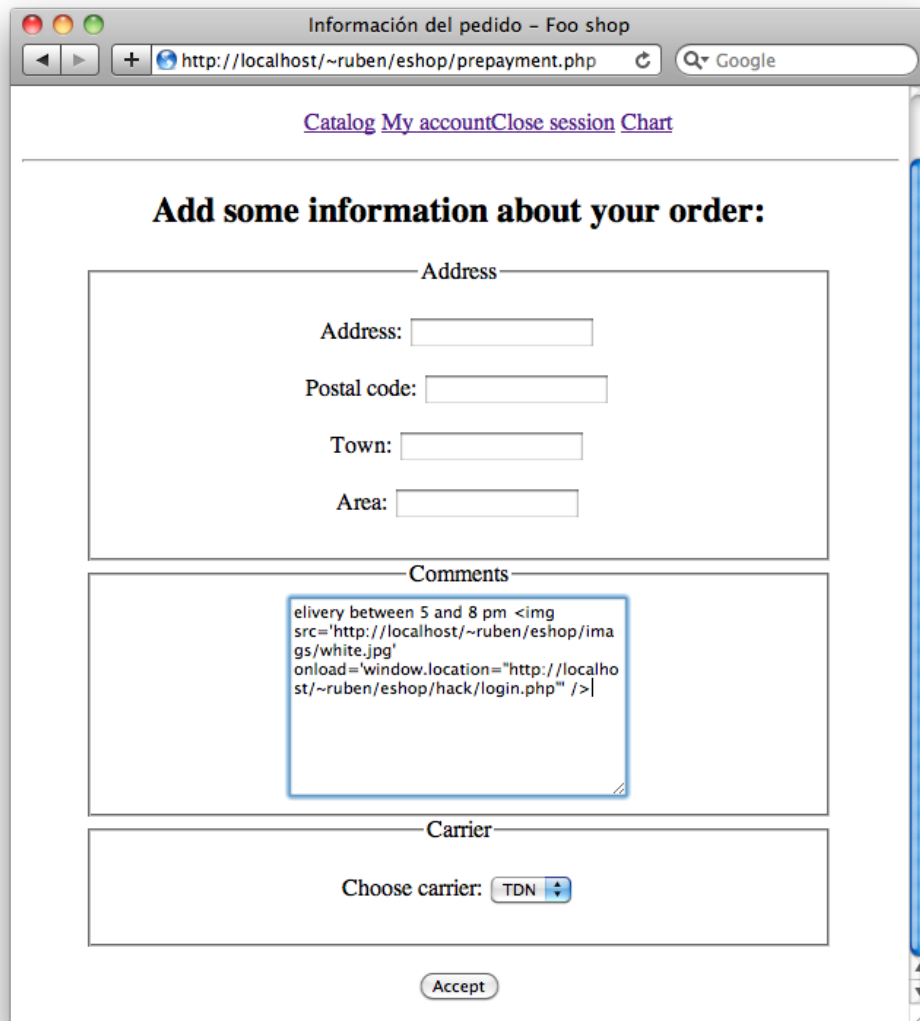


Figure 5.4: Attack on prepayment

password, that will be processed by the attacker, and he will return to the website, without knowing that he just gave his access data to an attacker.

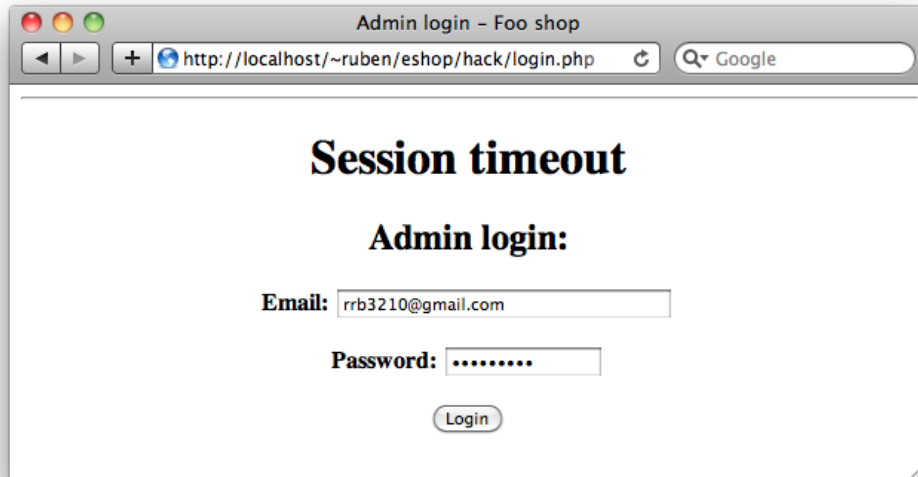


Figure 5.5: Fake admin login page

We have to say that this attack, to make it more credible, requires to know how the website is made and it is not easy, but most pages are designed in the same way, so an attacker can guess it.

We can also see that this attack is not very smart, because it requires that the attacker gives his personal data and pays, so if the shop's owner notices, he will be able to denounce him. But it illustrates how easy is to make an effective XSS attack.

Finally, we have to say that there are a lot of XSS attacks, some more complex than others, and using different languages, but their operation mode is the same, being possible to inject code from URL, due to it is input data too.

5.3 Solution

The website is well constructed, so it is not necessary to apply all the rules. In fact, we only need Rule #1 because the website only puts untrusted data in HTML context. It does not put untrusted data in HTML attributes, Javascript nor URLs. And, referent to CSS, the website does not use it, so we do not have to worry about that.

As we said in section 5.1.1, there is a PHP function that escape HTML characters, so the only thing that we have to do is use it when the data arrives, before storing it in the database, and the result in the code is:

```
if (isset($_POST["obs"])) {  
    $order->setComments(htmlspecialchars($_POST["obs"],  
        ENT_QUOTES, "UTF-8"));  
}
```

Listing 5.9: Solution adopted

Then, when that data is shown, it will show the written code and it won't be executed.

We can see it if we repeat the same attack executed in section 5.2. An attacker makes a purchase and he writes on the order's comments the same as listing 5.7, and after that he pays. Now, when the shop's owner checks for new orders, as we can see in figure 5.6 he will see on the order's comments the code written by the attacker, instead the code executes, as happened in the example.

This way, we avoided XSS attacks in the website.

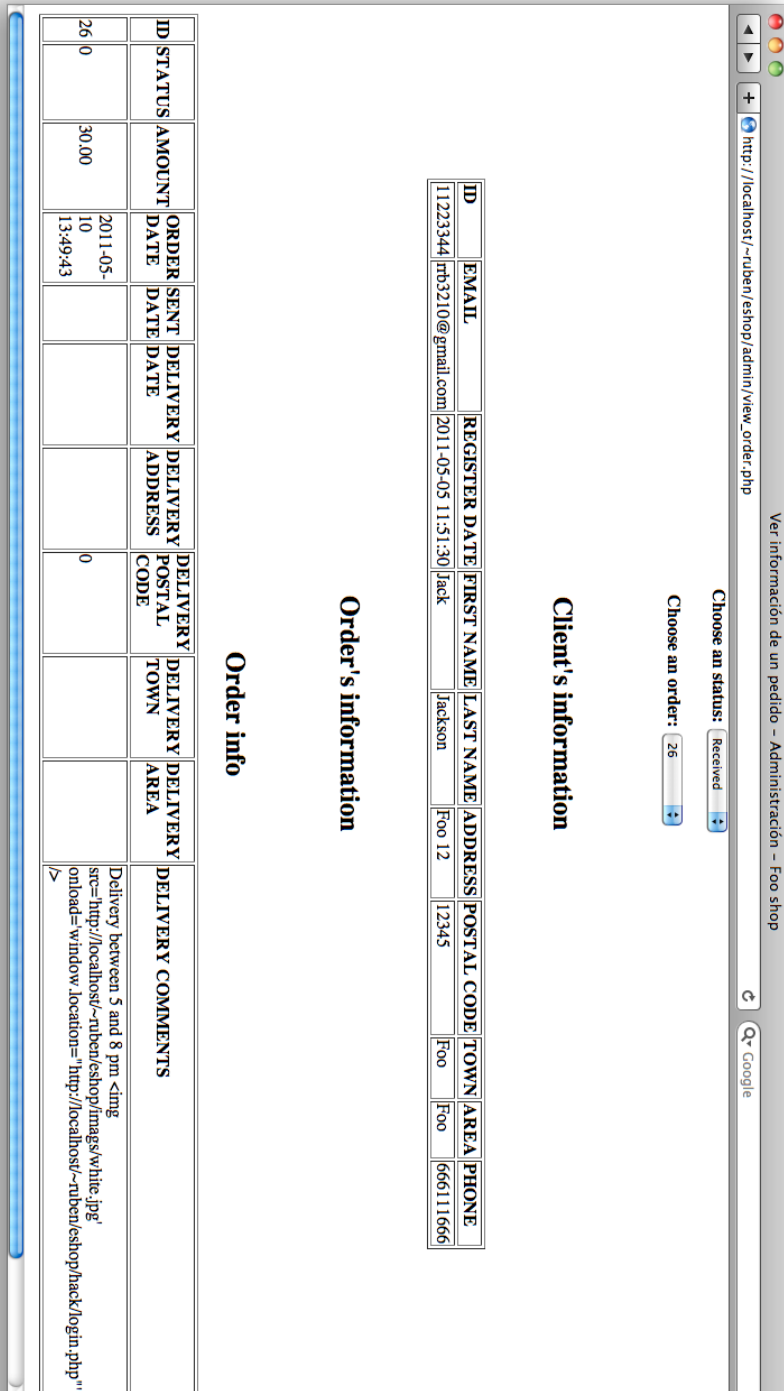


Figure 5.6: Failed attack

5.4 Conclusion

We could see that XSS attacks are potentially dangerous because it allows attackers to make malicious actions, from data theft to fraud. And this weakness is really extended along the Web, so attackers can take profit of it.

But actually, it is not difficult to prevent this kind of attacks, so all web developers should build their websites avoiding this weakness.

First, developers have to build a good websites, avoiding some potential security holes, writing secure code. And then, they have to check the rules described and apply whatever that are necessary to make their websites secure against XSS attacks.

And they should also use input validation, so some attacks will be filtered before they arrive until the server, where they will be completely filtered.

Using this combination properly, the website will be secure against XSS attacks.

Broken authentication and session management

6.1 Description

This issue is related with impersonation and it can affect users, not so much the server. It consists of breaking authentication systems, such as password, or breaking users' session, all to get access to the system with another user account.

We must differentiate between authentication and session management.

6.1.1 Authentication

Authentication is the process of verification that an user is really that user, and no one has supplanted him. This process of verification is done asking the user some data that only he can know, usually an ID and a password, but it is possible to ask him in more complex ways, as we will see.

Password strength

When we identify users using a password, it is necessary that the password is strong, so it will not be easy to attackers to break it, either using manual or automated means. We can force users to choose a good password applying these guidelines:

- Password length: it is more difficult to guess a password when it has more characters, because there are more possible combinations. According to OWASP[19], we can set up three classes according to the application:
 - Important applications: minimum of 6 characters length.
 - Critical applications: minimum of 8 characters length.
 - Highly critical applications: consider multi-factor authentication, explained in subsection 6.1.1.
- Password complexity: it is more difficult to guess a password when it is not trivial, namely, when it does not have dictionary words, sequences... According to OWASP[20], we should force users to create passwords according to this composition, or a variance of it:
 - at least: 1 uppercase character (A-Z)
 - at least: 1 lowercase character (a-z)
 - at least: 1 digit (0-9)
 - at least: 1 special character (!,*,\$,...)
 - no contiguous characters (e.g. 12lmno)
 - no more than 2 identical characters in a row (aaaa)

Password recovery

All applications should have a mechanism to let users recovery their account even if they forgot the data to access, such as password or ID.

Sites that have business relationships with their users should follow these steps to allow users recovery their passwords.

1. **Step #1: gather identity data.** This step consists of collect some user's data in order to ask him later his security questions. A minimum of 3 inputs is recommended, and some of them can be email address, last name, date of birth...
2. **Step #2: verify security questions.** This step consists of ask users their security questions that were asked when they registered. If any answer is incorrect, a generic error message should be displayed.

3. **Step #3: send a token over a side-channel.** This step consists of send via email or SMS a randomly-generated code with 8 or more characters to the user. This way, we establish an out of band communication and it would be really difficult for a hacker to overcome.
4. **Step #4: allow user to change password.** This last step consists of introduce the code sent in the previous step and allow the user to change the password, if the code is correct. This password, of course, should be checked to be strength, as we described in subsection 6.1.1, *password strength*.

Utilize multi-factor authentication

This kind of authentication asks for:

- Something you know (ID, password...).
- Something you have (tokens or mobile phones).
- Something you are (biometrics).

to login into a system. This kind of authentication is really secure, but as we said in subsection 6.1.1, *password strength*, is used only in highly critical applications, because it is really complex to the user to login.

Authentication responses

When there is a failed login, the application should answer with a generic error message, regardless if it was the ID or the password the incorrect input, and should not give any indication about the status of the account, if it exists or not. This way, we avoid attackers to know ID's that can be used to make another kind of attacks, like SQL, as we have seen.

A correct response is: *Login failed. Invalid ID or password*. It does not say if it was the ID or the password the wrong input, so attackers can not infer any data.

Transmit password only over TLS

This protection consists of transmit password only over TLS, so the data will be encrypted and no body will be able to see it.

We will see this protection in chapter 10.

Account lockout

This protection consists of let only do a established number of attempts to an account in a established period of time. This way, we avoid brute force attacks, that consist of introducing all password combinations for an account to get access to it.

6.1.2 Session management

Session management is a process by which the server maintains the state of an entity interacting with it. We use it for keep users logged in, so once they log in, they start their session. Sessions should be unique per user and difficult to predict.

Transmit session ID's only over TLS

Session ID's should only be transmitted over TLS, to avoid Man-in-the-Middle and surf jacking attacks.

We will see this protection in chapter 10.

Ensure session ID's

Session ID's should be large and random enough so they ca not be predicted, avoiding session hijacking and identity theft.

Session ID's should satisfy:

- Session ID's should be at least 128 bits, to avoid brute-force attacks.
- Session ID's should be random, so they can not be guessed.

- Session ID's should use the largest character set available to it.

Timeout

The application should implement session idle and absolute timeout to protect users if their ID's have been stoled. This way, attacker will have a limited time to perform his actions.

- **Session idle timeout:** sessions should be finished after a established period of inactivity.
- **Session absolute timeout:** session should be finished after a established period of time.

Cookie security

We can follow some procedures to ensure cookies.

- **Secure flag:** it consists of activate the secure flag over SSL operations.
- **HTTP only:** it consists of send to the browser the HTTP only directive, so it will protect the cookie from Javascript manipulation, avoiding XSS cookie attacks.
- **HTTP POST:** it consists of use HTTP POST instead of HTTP GET, due to last one can give some information to attackers along the traffic, even if SSL is used.

6.2 Examples

The website does not implement good measures in authentication and session management. We will show some examples to demonstrate the weaknesses in this area. However, we will not show examples of:

- Password recovery, multi-factor authentication, account lockout, secure session ID's and timeouts: because these features are not implemented

in the website although some of them will be implemented and showed in solutions section.

- Password and session transmission over TLS: because these features will be explained in chapter 10.
- Cookie security: because it is not implemented in the website.

6.2.1 Password strength

The website does not implement any control about password strength. We can register, via `new_user.php` page, using an insecure password, as we can see in figure 6.2¹. Once we press the accept button, we are registered with an insecure password, as we can see in figure 6.1.

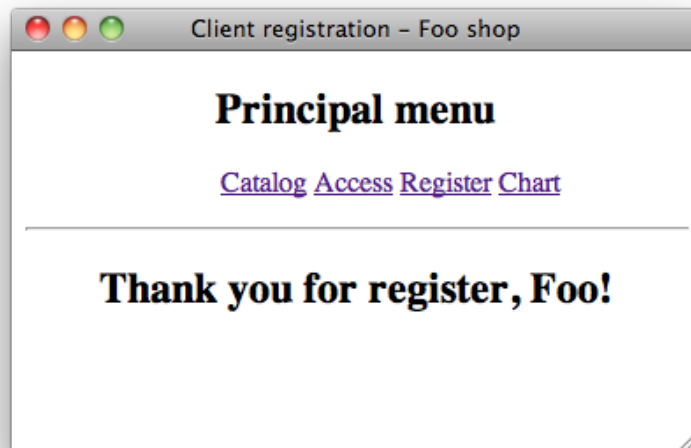


Figure 6.1: Registering succesful with a wrong passowrd

¹In order to present the pictures, the password field is treated as a normal text field

Client registration - Foo shop

ID:

Email:

First name:

Last name:

Address

Address:

Postal code:

Town:

Area:

Password

Password:

Password confirmation:

Contact

Phone number:

Accept

Figure 6.2: Registering using an insecure password

6.2.2 Authentication responses

In some of the pages, the website does not work correctly when it has to show error messages while authentication or registration. We can see in `new_user.php` two issues in error messages, one when we use an ID that is already in use (figure 6.3), and the other one when we use an email that is already in use (figure 6.4).

The source code for this section is:

```
$res_check = checkClient($_POST["id"], $_POST["mail"]);
if($res_check == 0) {
    $client = new Client(/* All data */);

    $client->insert();

    echo "<h2>Thank you for register, " . $_POST['name'] .
        "!</h2>";
} else {
    switch($res_check) {
        case 1: echo "<h2>The ID already exists in the
            database</h2>";
            break;

        case 2: echo "<h2>The email address already exists in the
            database</h2>";
            break;
    }
}
```

Listing 6.1: User verification and error messages

The showed message gives us information about the users: if the email or the ID exist in the website.

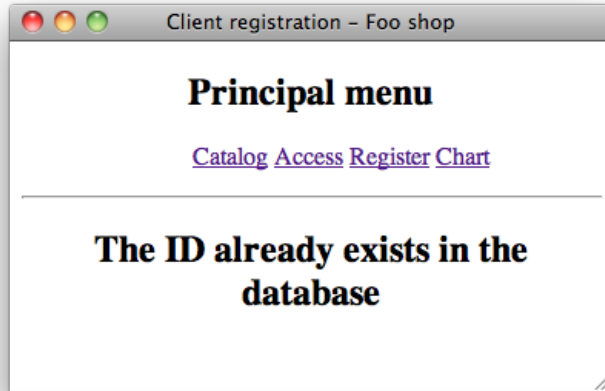


Figure 6.3: Wrong error message: it gives information about IDs in the database

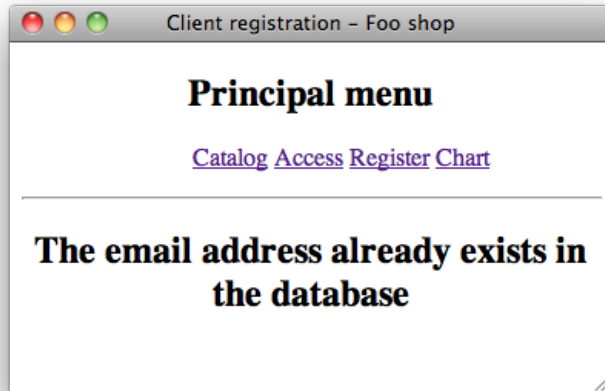


Figure 6.4: Wrong error message: it gives information about emails in the database

6.3 Issues solution

In this section we will show how we solved the mentioned issues and how we implemented new features to enforce our website authentication and session management.

We will show how we solved passwords weaknesses and error messages, and how we implemented new features: password recovery, account lockout and timeout.

We did not implement multi-factor authentication because we do not think that this kind of authentication must be used in an online shop, because it is tedious to the user, that usually wants speed.

We also did not implement cookies security, because it is a issue related to cookies, that are not implemented in the website.

6.3.1 Password strength

As we said, we need to assure that the website's users choose a good password to avoid some kind of attacks.

To solve this issue, we implemented some features in `new_user.php` webpage. We inserted two visual markers: one to show if the password is long enough and strength (figure 6.5), and another one to show if both password and password confirmation are equal (figure 6.6). And we added a control that does not allow users to register if the password is not long enough and if the password and the password confirmation are not equal.

We force users to write twice their password because they do not see what they are writing, and maybe they make a mistake writing the password and, then, they will not be able to access to the website. So, it is better to force users to write it twice, to ensure that they do not make mistakes.

To make the visual markers, we used an already written code found in the Internet[21]. It uses AJAX and JQuery[22] to show the verification in real-time, and it uses a simple algorithm to calculate the password complexity:

1. It checks the length of the input string. If it is longer than the minimum length, it gives base score of 50. If not, the base score is 0.

Client registration - Foo shop

ID:

Email:

First name:

Last name:

Address

Address:

Postal code:

Town:

Area:

Password

Secure!

Passwords match

Contact

Phone number:

Accept

Figure 6.5: Password verification: correct password and confirmed

Client registration - Foo shop

ID:

Email:

First name:

Last name:

Address

Address:

Postal code:

Town:

Area:

Password

Secure!

Password different

Contact

Phone number:

Accept

Figure 6.6: Password verification: correct password but not confirmed

2. It checks how many extra characters over the minimum the string has, and gives a bonus for each character.
3. It checks if the string has a combination of upper case letters, numbers and symbols or all three, and gives a bonus for it. It also gives a bonus for each presence.
4. It checks if the string only contains either lower case letters or numbers, and if it does, it penalizes.
5. Finally, it calculates the final score and decides the strength of the password.

The original code had set the minimum characters at 6, and it only had let view one input, but we modified it in order to change the minimum characters at 8 and to allow write twice the password and check if both are equal to verify it.

We chose a minimum of 8 characters because we consider the website as a critical application due to it allows to buy, to modify orders...

The source code can be found in the appendix A.1.

The other feature is a control that does not allow users to register if the password is not correct. We wrote it in Javascript and it checks, in this order:

1. That both passwords are equal (figure 6.7).
2. That password fields are not empty (figure 6.8).
3. That password has at least 8 characters (figure 6.9).

If one of these conditions is not satisfied, the user ca not register in the website.

The source code is:


```
function formValidator() {
    var pass1 = document.getElementById( 'pass1 ' );
    var pass2 = document.getElementById( 'pass2 ' );

    if(isEqual(pass1, pass2, "Passwords must be equal!")){
    if(notEmpty(pass1, "You must write a password!")){
    if(isPass(pass1, "Password must have at least 8
        characters!")){
        return true;
    }
    }
    }

    return false;
}

function isEqual(elem1, elem2, helperMsg){
    if(elem1.value == elem2.value){
        return true;
    }
    else {
        alert(helperMsg);
        return false;
    }
}

function notEmpty(elem, helperMsg){
    if(elem.value.length == 0){
        alert(helperMsg);
        return false;
    }
    return true;
}

function isPass(elem, helperMsg){
    if(elem.value.length >= 8){
        return true;
    }
    else {
        alert(helperMsg);
        return false;
    }
}
```

Listing 6.2: Password verification

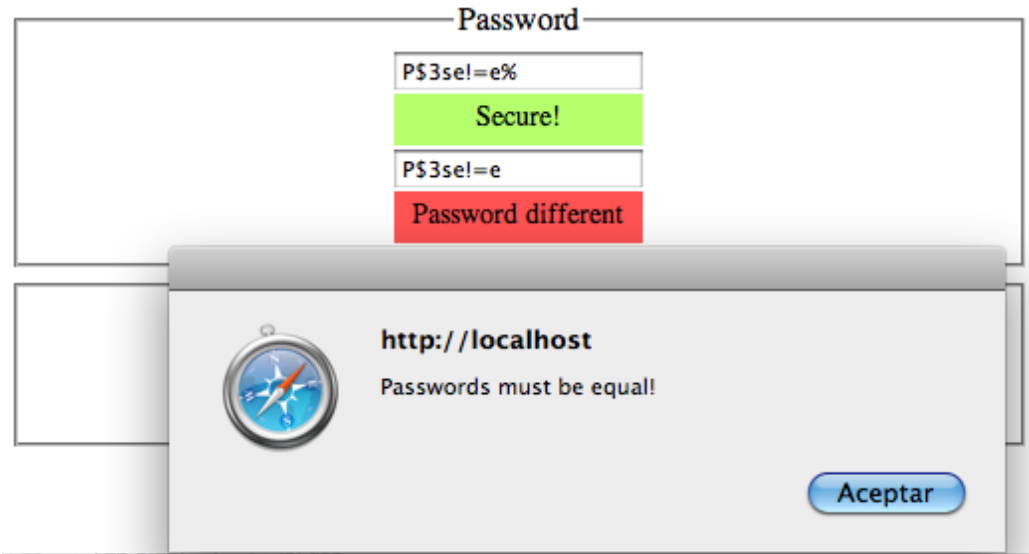


Figure 6.7: Password verification: password not confirmed



Figure 6.8: Password verification: password empty

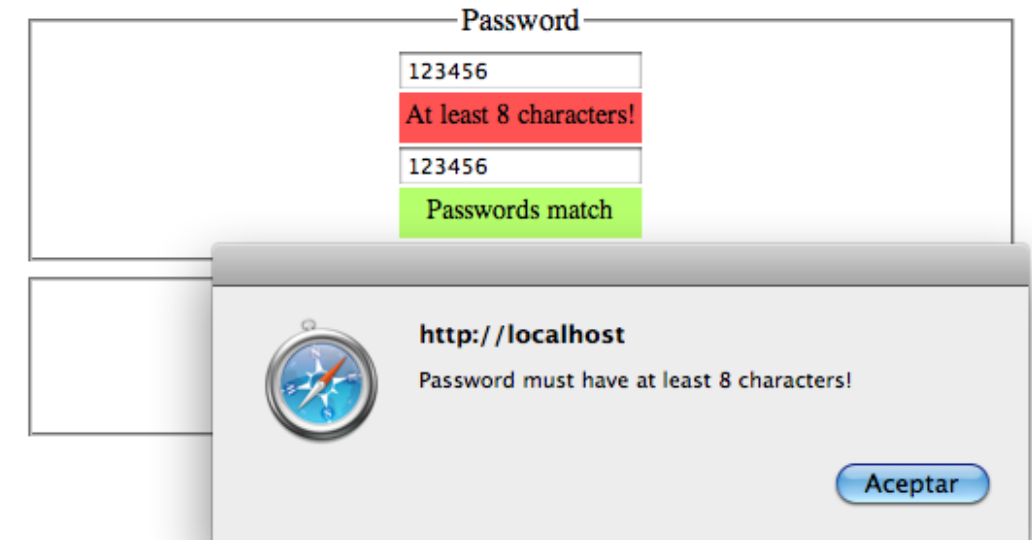


Figure 6.9: Password verification: password too short

6.3.2 Password recovery

To implement a password recovery system, we followed the steps described in subsection 6.1.1, *password recovery*, and we had to make some changes in the database.

First, we had to add two new attributes in the *Client* table, to add the security question and its answer. So, these attributes description are:

- Question: this is client's secret question; it is character.
- Answer: this is client's answer to his secret question; it is character.

After that, we had to let users to introduce both question and answer when they register, so we had to modify `new_user.php` page (figure 6.10) and the object.

Then, we had to add a new table in the database, called *Pass_recovery*. Its attributes are:

- ID: this is user's ID; it is numeric;

The image shows a registration form titled "Password". It contains the following elements:

- A text input field for the password.
- A button labeled "Enter your password".
- A second text input field for confirming the password.
- A button labeled "Confirm the password".
- A label "Security question:" followed by a text input field.
- A label "Security answer:" followed by a text input field.

Figure 6.10: Registering process: security question and answer added

- SID: this is session's ID, unique and generated randomly; it is primary key and character.
- Code: this is recovery code; it is character.

We had to create a new page, called `password_recovery.php`. Within this page we manage all password recovery.

First, it asks for the email address (figure 6.11). When the user writes one and submit, it checks if the email address is in the database; if it is not, it shows a generic error message (figure 6.12). But if the email exists in the database, it generates a randomly session ID and it stores it with the user's ID in the *Pass_recovery* table. Then, it sends an email² to the user with the link, that has both session ID and user ID, to resume the process.

When the user accesses to the link, the page asks the user his secret question (figure 6.13). When the user answers it, the page checks if the answer is correct, and if it is not, it shows a generic error message (figure 6.12). But if the answer is correct, it generates a code that stores in the *Pass_recovery* table with the session ID, and sends to the user another email with the link and the code.

²At the moment, to make easier the tests, it only shows the URL in the page

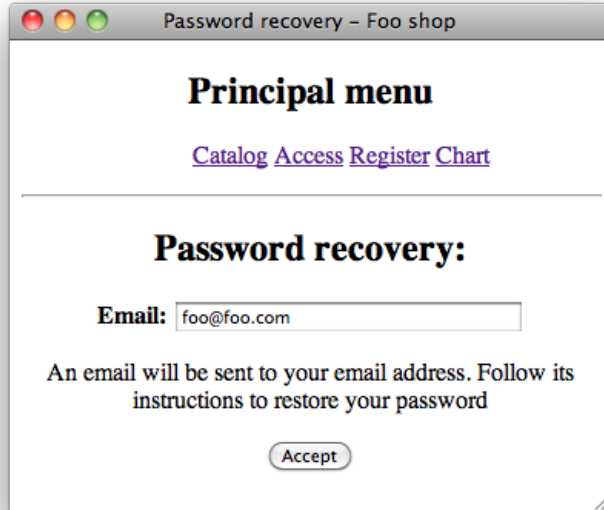


Figure 6.11: Password recovery: asking the email

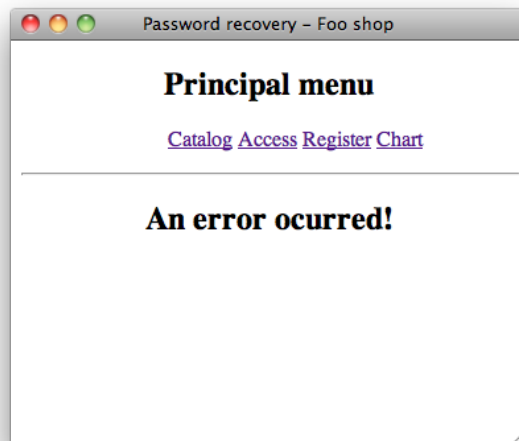


Figure 6.12: Generic error message showed when an error occurred



Figure 6.13: Password recovery: asking the secret question

When the user accesses to the new link, the page asks him for the code and a new password (figure 6.14). When he introduces the code and the password, that has the controls described in subsection 6.3.1, the page checks if the code is correct, and if it is not, it shows a generic error message (figure 6.12). But if it is correct, it changes the password and shows a message to the user (figure 6.15).

This way, we accomplished the steps described in subsection 6.1.1, *password recovery*, avoiding possible attacks.

Finally, we have to say that the table *Pass_recovery* should be emptied every day, to avoid long session IDs and to not take space in the database.

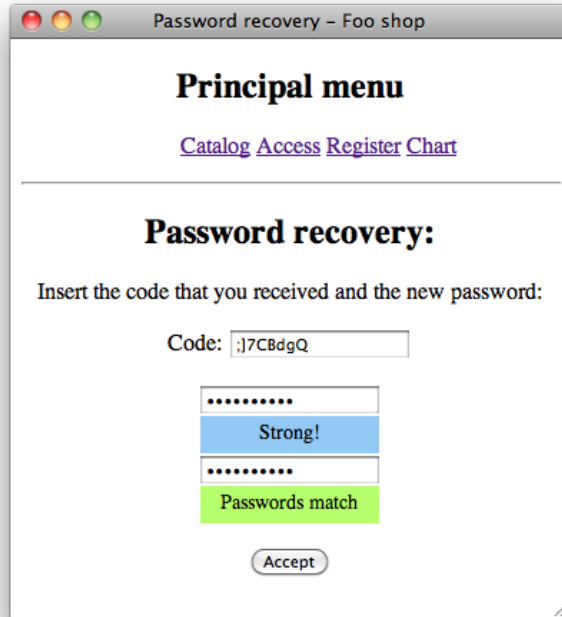


Figure 6.14: Password recovery: asking the code and the new password

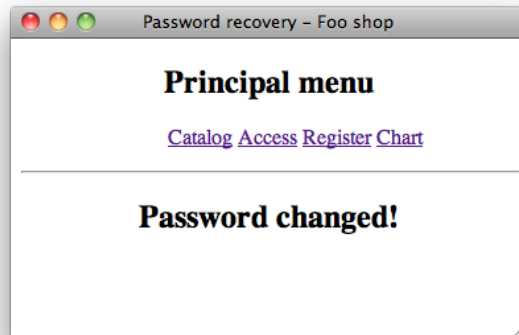


Figure 6.15: Password recovery: password successfully changed

6.3.3 Authentication responses

Fixing the issue described in the examples subsection 6.3.5, in the `new_user.php` page, is really easy, we only have to change the message showed to avoid this weakness. We can see the source code, that actually is almost the same:

```
$res_check = checkClient($_POST["id"], $_POST["mail"]);  
if($res_check == 0) {  
    $client = new Cliente(/* All data */);  
  
    $client->insert();  
  
    echo "<h2>Thank you for register, " . $_POST['name'] .  
        "!</h2>";  
} else {  
    echo "<h2>Wrong username or password.</h2>";  
}
```

Listing 6.3: Password verification

If we try to register now with an existing ID or email, we will see an error message like in figure 6.16.

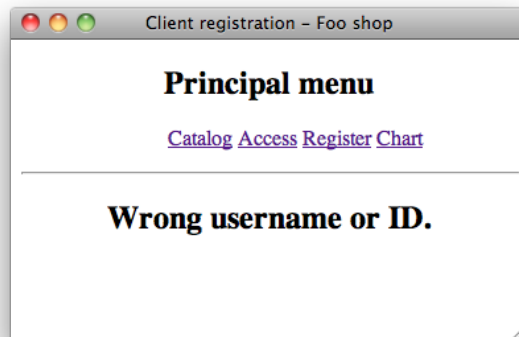


Figure 6.16: Generic error message showed when ID or email exists in the database

6.3.4 Account logout

We implemented the account logout in the `user_login.php` page, because is where attackers can make brute-force attacks.

To implement the account logout, we had to add a new table in the database, called *Lockout*. Its attributes are:

- Email: this is the email account used to the login attempt; it is primary key and character.
- Attempts: this is the number of attempts done to gain access to the account; it is numeric.
- Blocked: this attribute indicates if the account is blocked or not; it is boolean.

So, the algorithm used to implement account first checks if the used email is in the lockout table or not. If it is not, it checks if the username and the password are correct and, if they are, it does what is necessary to log in the user; if they are not correct, then it inserts the email in the lockout table, with one attempt and not blocked.

But if the given email is in the database, then it checks if it is blocked or not. If it is, it denies the access to the user with that email (figure 6.17).

But if not, then it checks if the username and password are correct, and if they are, it logs in the user and deletes the email from the lockout table. But if they are not, then it updates the table: if the attempts are 3, it also blocks the account (figure 6.18).

We can see that it does not say if the account exists in the database. It would contradict the authentication responses already implemented, so it just say that the email is blocked, so it is possible to block accounts that are not in the database.

The source code for this part is in the appendix A.2.

Fixing this vulnerability can cause DoS³ attacks. An attacker can block accounts juts trying to access with different emails. It is easy to automatize

³Deny of Service

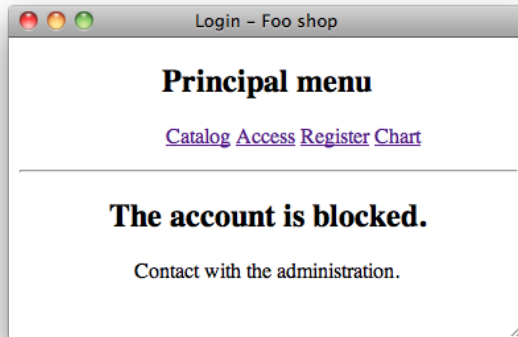


Figure 6.17: Blocked account

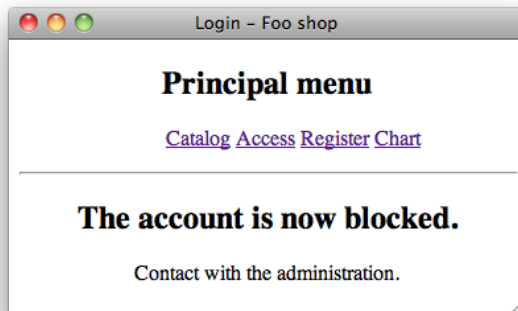


Figure 6.18: Blocked account after 3 attempts

this attack, so the attacker can block a lot of accounts in a brief period of time.

To avoid this new issue, we should implement the lockout for IP, but it can block the access to an entire company if an user does not remember his account, for example, so it is not a good solution. The best that we can do is implement a process that erases the table every concrete period of time, each 30 minutes, each hour... This way, the impact of the DoS attacks will be minimal.

6.3.5 Ensure session ID's

When we need to use session ID's, like when we described password recovery in subsection 6.3.2, we have to use an unique and unpredictable value to prevent identity theft.

To achieve this, we used a PHP code found in the Internet[23], that generates tokens, and it is possible to specify the total length and the usage of MD5. We can see the source code:

```
function genToken( $len = 32, $md5 = true ) {
    $chars = array('Q', '@', '8', 'y', '%', '^', '5', 'Z', '(',
        'G', '-', 'O', ',', 'S', '-', 'N', '<', 'D', '{', '}', '[',
        ']', 'h', ';', 'W', '.', '/', '|', ':', '1', 'E', 'L',
        '4', '&', '6', '7', '#', '9', 'a', 'A', 'b', 'B', '~', 'C',
        'd', '>', 'e', '2', 'f', 'P', 'g', ')', '?', 'H', 'i', 'X',
        'U', 'J', 'k', 'r', 'l', '3', 't', 'M', 'n', '=', 'o', '+',
        'p', 'F', 'q', '!', 'K', 'R', 's', 'c', 'm', 'T', 'v', 'j',
        'u', 'V', 'w', ' ', 'x', 'I', '$', 'Y', 'z', '*');
    $numChars = count($chars) - 1; $token = '';

    for ( $i=0; $i<$len; $i++ )
        $token .= $chars[ mt_rand(0, $numChars) ];

    if ( $md5 ) {
        $chunks = ceil( strlen($token) / 32 ); $md5token = '';
        for ( $i=1; $i<=$chunks; $i++ )
            $md5token .= md5( substr($token, $i * 32 - 32, 32) );
        $token = substr($md5token, 0, $len);
    } return $token;
}
```

Listing 6.4: Session ID generator

Every time we needed to use a unique session ID, we called this function to generate it.

6.3.6 Timeout

To implement session timeout, both idle and absolute, we added two new variables in the superglobal variable `$_SESSION` when the user logs into his account: one indicating what time was the session start, and one indicating what time was the last activity. We can see the source code:

```
$_SESSION["user"] = $user;  
$_SESSION["login"] = 1;  
$_SESSION["start_time"] = time();  
$_SESSION["last_activity_time"] = time();
```

Listing 6.5: Session timeout start

Of course, at the beginning both values will be the same, but it does not matter because we will update the value of last activity time.

Then, we had to check this time in each page that requires user authentication, to close the session if it is necessary. We implemented a function that does that, check if one of the times expired and, if it does, it closes the session:

```
function checkTimeout() {  
    $idle = 900;  
    $absolute = 3600;  
  
    if((time() - $_SESSION["start_time"]) > $absolute || (time()  
        - $_SESSION["last_activity_time"]) > $idle) {  
        session_unset();  
        return true;  
    }  
  
    $_SESSION["last_activity_time"] = time();  
    return false;  
}
```

Listing 6.6: Session timeout checking

Because we are building an online shop, we thought that the idle time should

be 15 minutes and the absolute time 1 hour⁴.

Now, we have to check the time in each page. We can see in figure 6.19 what message is shown when the session expires, in `user.php` page, and here is the source code template for it, that should be in each page that requires user authentication:

```
$ses_started = (isset($_SESSION["login"]));
$timeout = checkTimeout();
superior_menu();

if($timeout) {
    echo "<center><HR width=100% align='center'>";
    if($ses_started) {
        echo "<h2>Session timeout expired!</h2>";
    } else {
        echo "<h2>You don't have access to this page!</h2>";
    }

    echo "<meta http-equiv='Refresh' content='5;URL=index.php'
        /></center>";
} else {
    //Code if the user is authenticated
}
```

Listing 6.7: Session timeout template

This way, we implemented a session timeout, avoiding the related attacks.

6.4 Conclusion

We could see that there are a lot of aspects to consider about authentication and session management, and it is a very important part of the website, due to if what will keep users safe.

We noticed that is not easy to provide the security in this area, because there are too many things to manage, but luckily, the code and the logic is reusable in other projects.

First, web developers have to force users to choose strength passwords, in order to avoid brute-force attacks against their accounts, and referred to this

⁴Values in the variables are indicated in seconds.

kind of attacks, it is necessary to implement an account logout.

To make things easier to the user, it is also good to have a good system to recover the password, in case that the user forgot it. And it is also good to implement a timeout, so users don not have to worry about closing their sessions.

Finally, more to the server side, it is right to show generic error messages and to ensure session IDs, in order to avoid attacks.

With all these features, and some other that we will see in chapter 10, the website will have an optimal authentication system and session management.

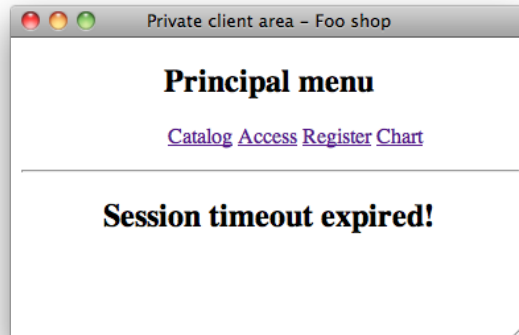


Figure 6.19: Session timeout

Insecure direct object references

7.1 Description

This kind of attack, though is common, is not critical, because the attacker can only access to the data related to the weakness, not to another data, so he can get access to critical that if we do not expose it.

An insecure direct object reference happens when we reference to an internal object, like a database record, as a URL. This happens, for example, when we try to access to a record by its primary key and we give the primary key through the URL. It is easy to see that an attacker can change that primary key and access to another information that not belongs to him.

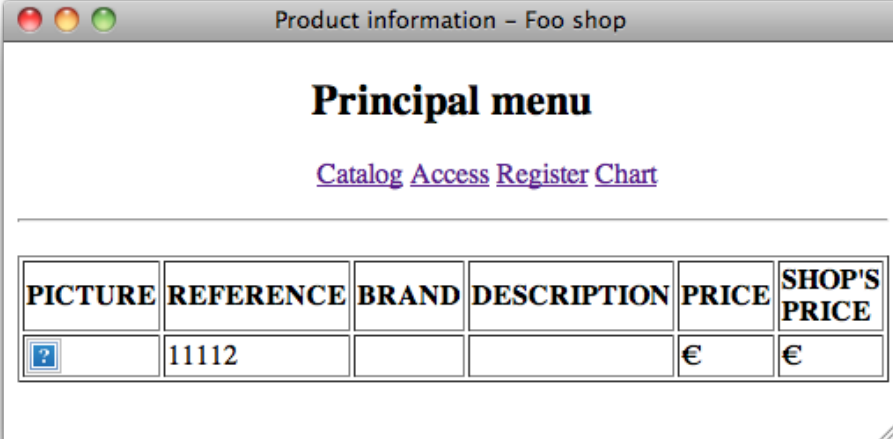
But we can also see that this attack is really limited, due to it can only access to another data of the same type, and if that data is not critical, the attack is practically useless.

We can avoid this weakness controlling what data is accessible depending of the user.

7.2 Example

In the website all parameters from a form are passed using the POST method, and the GET method is used in only two pages: `info_product.php` and

`info_promotion.php`. These pages show products' info and promotions' info, so an attacker can only have access to other products or promotions. This is not a big problem, precisely because in a shop interests that clients view the products, but maybe we do not want to show discontinued products or promotions, so we have to control the access, and to avoid things like it is shown in figure 7.1.




PICTURE	REFERENCE	BRAND	DESCRIPTION	PRICE	SHOP'S PRICE
	11112			€	€

Figure 7.1: Insecure reference: an empty record is fetched

What happens there is that we try to access to a reference that does not exist in the database, and because we do not control it, it shows an empty table, that should not be showed, and, instead of that, it should show an error message.

7.3 Issue solution

To solve this issue, in the case of the website, we only have to check if the requested value exists in the database. If it does, then we show it, but if not, we have to show an error message.

In the page `info_product.php`, we solved it just checking if the reference exists in the database before reading it, so we can now know if the attacker tries to access to a non-existent product, and if he does, he will see an error message (figure 7.2). The source code is:

```

if(isset($_GET["ref"])) {
    $db = new AuxDB();
    $db->connect();

    $sql = "SELECT * from PRODUCT where REFERENCE = ?";
    $params[] = array($_GET["ref"], PDO::PARAMINT);

    $rst = $db->executeSQLs($sql, $params);

    if($db->numRows($rst) > 0) {
        //Code when product exists
    } else {
        echo "<h2>An error ocurred!</h2>";
        echo "<meta http-equiv='Refresh' content='5;URL=index.php' />";
    }
}

```

Listing 7.1: Session timeout template

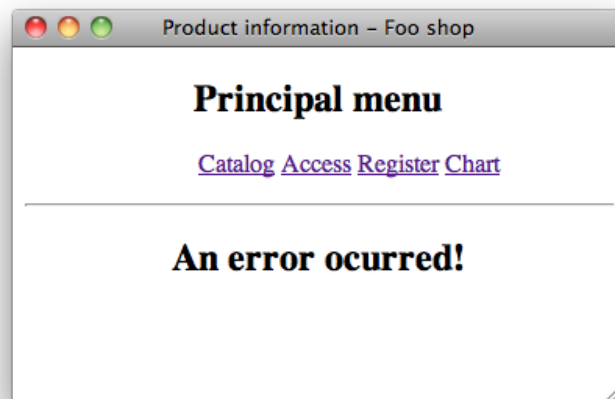


Figure 7.2: Insecure reference when trying to access to an inexistent reference

7.4 Conclusion

Insecure direct object references, as we could see, is not a big problem if the website is well builded, so to avoid it web designers should not use references to private objects in the URL.

But if they have to do it, they have to be sure that they use it with data that is not critical, and controlling it as we saw. If they accomplish these guidelines, the website will be secure against insecure direct object references attacks.

CSRF (Cross-site request forgery)

8.1 Description

CSRF attacks are not especially critical, due to they require, in first place, that the user makes an action, like follow a link, and in second place they require that the user is logged in in the website where attackers want to reference.

This kind of attack consists of inducting the victim to access a page that contains a malicious request. It is malicious because it inherits victim's identity and his privileges to perform an undesired action, like changing victim's personal data or password.

We can see that this attack impact depends on victim's role. If the victim is a normal user, the attack can only compromise end-user data, but if the victim is the website administrator, the attack can compromise the entire website.

8.2 Example

We will see an example of a CSRF attack in the website, in which the attacker makes the victim changes his password without notice it.

The affected page in this case is `modify_user.php`. This page allows the web-

site users to change their data, like personal data, address data or password data, using to choose one or another a value passed through the URL.

The template source code for this section is:

```

if(isset($_POST["Submit"])) {
    switch($_GET["option"]) {
        case 0:
            //Modify user's personal data
            break;

        case 1:
            //Modify user's address data
            break;

        case 2:
            if(strlen($_POST["pass1"]) > 0) {
                $user->setPass($_POST["pass1"]);
            }
            break;
    }
    $user->modify();

    echo "<h2>Data correctly modified!</h2>";
    echo "<meta http-equiv='Refresh' content='5;URL=user.php' />";
} else {
    //Input form
}

```

Listing 8.1: modify_user.php template source code

We can see that an attacker can induce a victim to submit a form from another page that redirects to `modify_user.php`, passing all POST parameters, and that will change the user's data. For example, we have this script:

```

echo "
<form name='hack' method='post'
      action='../modify_user.php?option=2'>
  <input type='hidden' name='pass1' value='qwerty'>
  <input type='hidden' name='Submit' value='1'>
</form>
<script
  language='JavaScript'>document.hack.submit();</script>" ;

```

Listing 8.2: Malicious script source code

This script will execute automatically when it is accessed, and it will set the POST variables necessary to change the personal data. We can see that it will set the password to qwerty, instead of the user's original.

This way, the attacker, using social engineering, can trick the victim to access that script, embedding its URL in a picture link, so when the victim tries to access the image, will be redirected to the script, that will change his password.

8.3 Solution

To solve this issue, we must verify that the form has been called from the website, and not from other page. To achieve that we have to generate a unique token, using the function described in subsection 6.3.5, and assign it to the SESSION variable when the users login into the website. Then, when we show the form, we have to assign the token as a hidden parameter, and when the user submits the form, check that the token is correct.

We implemented this solution in the webpage. First, we had to generate the token and store it when the user logs in. We just had to add the following line to the `user_login.php`:

```
$_SESSION["token"] = genToken();
```

Listing 8.3: Line to add to user_login.php

Then, we had to add the token as an input parameter in the `modify_user.php` form, and before changing user's data, we had to verify that both token from the form and session token are the same and then, modify the data. The

template source code is:

```

if(isset($_POST["Submit"])) {
    switch($_GET["option"]) {
        case 0:
            //Modify user's personal data
            break;

        case 1:
            //Modify user's address data
            break;

        case 2:
            if($_SESSION["token"] == $_POST["CSRFToken"]) {
                if(strlen($_POST["pass1"]) > 0) {
                    $user->setPass($_POST["pass1"]);
                    $msg = "<h2>Data correctly modified!</h2>";
                } else {
                    $msg = "<h2>An error occurred!</h2>";
                }
            }
            break;
    }
    $user->modify();

    echo $msg;
    echo "<meta http-equiv='Refresh' content='5;URL=user.php' />";
} else {
    switch($_GET["opcion"]) {
        case 0:
            //User's personal data's form
            break;
        case 1:
            //User's address data's form
            break;
        case 2:
            echo "<h2>Modify your password</h2>";
            <form method='post' action='modify_user.php?option=2'>

            <p>Password:
            <input name='pass1' type='password' id='pass1'
                maxlength='16' />
            </p>

            <p>Confirm your password:
            <input name='pass2' type='password' id='pass2'

```

```
        maxlength='16' />
    </p>

    <input type='hidden' name='CSRFToken' value='" .
        $_SESSION['token'] . "'>

    <p><input type='Submit' name='Submit' value='Modify'
        /></p>
    </form>";
    break;
}
}
```

Listing 8.4: modify_user.php template source code with the issue solved

Now, when the victim access to the script described in the example, he will see an error message (figure 8.1) and his data will not be modified.

8.4 Conclusion

We could see that, using social engineering, is really easy for the attacker to make a CSRF attack, inducting the victim to access a script that performs the malicious action.

But the victim has to be logged in in the website and has to access to the script, so it depends on the victim if the attack is successful or not.

Although, it is really easy to prevent this attack, we only have to generate a session token and check it every time that the users submit a form, so web developers should protect their websites against this attack.

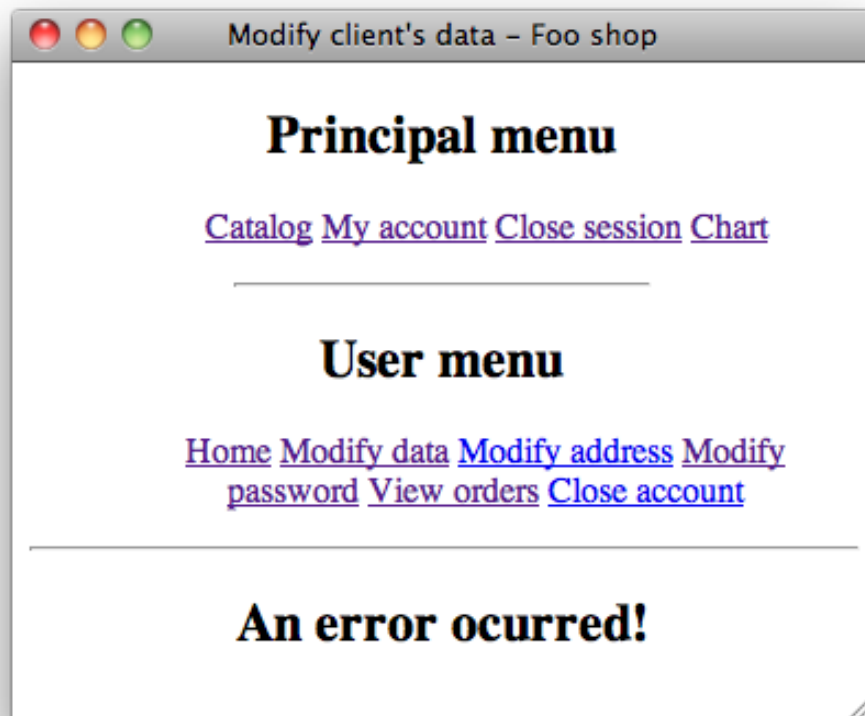


Figure 8.1: Message displayed when the token is not verified

Failure to restrict URL access

9.1 Description

This kind of attacks are critical depending on user's privileges, because attacker takes victim's privileges, so it is not the same if the attacker gets access to a normal user account than if he gets access to an admin account.

This attack does not require any technical knowledge, and it only consists of change manually the URL of the website to access to an unauthorized page. If there is not a good privilege control, the attacker will get access to forbidden pages.

We can see that if the victim is a normal user, the impact of the attack will not be critical, but it will be if the victim is the administrator.

To avoid this weakness, it is necessary to implement privileges control.

9.2 Example

In the website, we can see an example of this attack in the administration section. Although there is a login page, the whole administration section does not have any privileges control, so an attacker can just write the URL of a section and he will get access as an administrator to manage everything.

For example, he can get access to the clients section (figure 9.1), and he will be able to manage all about the clients, adding new clients or modifying, deleting or viewing the existing clients.

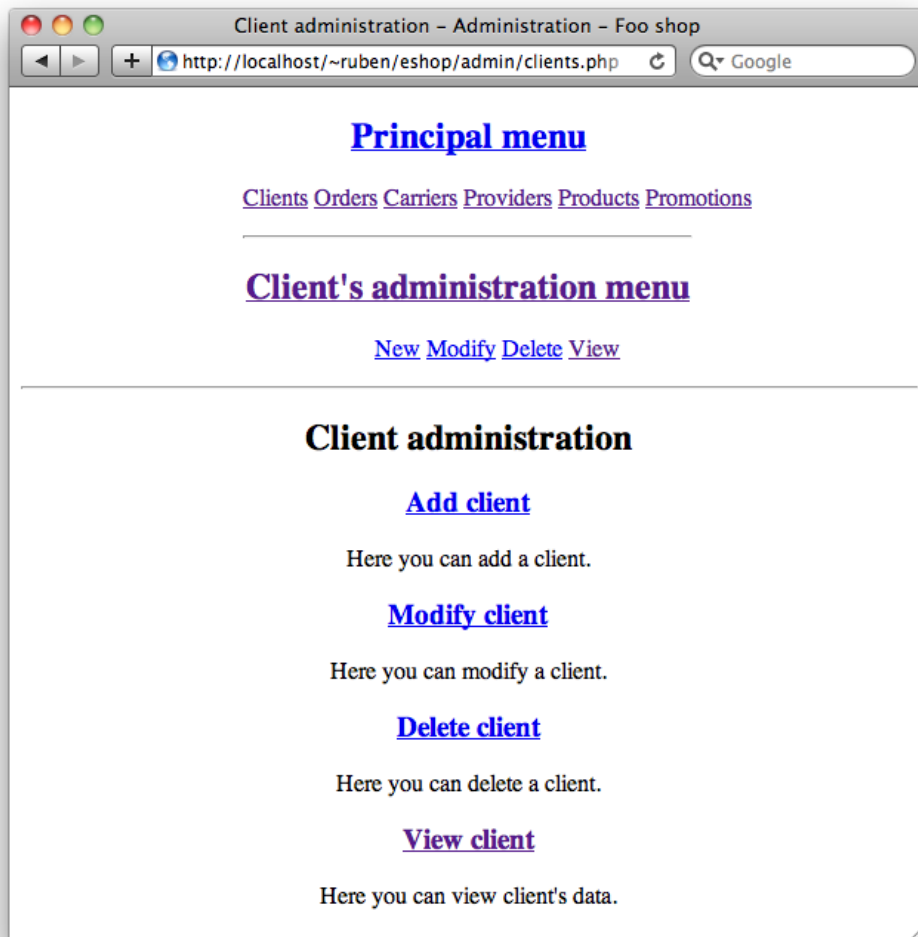


Figure 9.1: Access to a forbidden page without authenticate

9.3 Solution

To avoid this issue, we have to implement a permission control system to allow only logged in users to access the forbidden pages. We had to implement some new features in the website to solve the issue.

First of all, we had to add a new table to the database to save all administrator users. We could use the common user table, but, besides we did not need so many fields, it is better to separate users from administrators.

The new table is called *Admin* and its attributes are:

- ID: it is admin's ID; it is primary key and numeric.
- Email: it is admin's email; it is character.
- Password: it is admin's password; it is character.

We identify admins by their email, so we can have more than one admin if it is necessary in the future.

After that, we had to change the login page, and add to it the control to check if the username and the password are correct and, if they are, give access to the user. The value to control the session login is 2, because number 1 it is already in use for the normal web section. The template source code is:

```
if($_SESSION["login"] == 2) {
    //User already logged in, so redirect to the administration
    index
} else {
    if(isset($_POST["Submit"])) {
        $db = new AuxDB();
        $db->connect();

        $sql = "select * from ADMIN where EMAIL = ? and PASSWORD =
            ?";
        $params[] = array($_POST["mail"], PDO::PARAM_STR);
        $params[] = array($_POST["pass"], PDO::PARAM_STR);

        //echo $sql;
        $rst = $db->executeSQLs($sql, $params);
```

```

if($db->numRows($rst) > 0) {
    $_SESSION["login"] = 2;

    echo "<br><h2>Log in successful!.</h2>";
    echo "<meta http-equiv='Refresh '
        content='5;URL=index.php' />";
} else {
    echo "<br><h2>Error logging in.</h2>";
    echo "<meta http-equiv='Refresh '
        content='5;URL=login.php' />";
}
} else {
    //Form to write username and password
}
}

```

Listing 9.1: Login admin user template source code

We have to say that we should implement all features described in section 6.3 to have a good login system, except the account lockout, because if not, an attacker could block all the admin accounts and then the admins would not be able to unlock any account. In this situation we assume that the admins do not have technical knowledge to access to the database and erase the blocked accounts

After that, we had to implement the privileges control in all administration pages to allow only authenticated users to access the pages. The template source code is:

```

if($_SESSION["login"] != 2) {
    echo "<center><h2>Access forbidden!</h2></center>";
} else {
    //Code if the user has access to the page
}

```

Listing 9.2: All admin pages template source code

Now, if an unauthorized user tries to access a forbidden page, we will get an error message (figure 9.2).

9.4 Conclusion

As we could see, an access to a forbidden page by an unauthorized user can become into a critical attack if the target account is the admin account, but the impact is not so big if the page is from a normal user, for example.

All web developers should avoid this weakness, and to do that, it is necessary to check every page and decide if it requires privileges control or not, and if it does, implement it.



Figure 9.2: Error message when trying to access to a forbidden page without authenticate

Insufficient transport layer protection

10.1 Description

This issue, although is not entirely related with web development, is important while it assures a private data interchange between the website users and the server, so nobody can get access to that information, besides the involved parts.

Imagine a situation where an user logs into a website without using Transport Layer Security¹. All his data will be sent as a plain text, including the password, so an attacker can intercept that data and obtain it. The use of TLS encrypts the communication, so all the data sent will be encrypted and if an attacker intercepts the message, he will not have access to it.

Nowadays, the terms SSL² and TLS are used indistinctly, and, actually, SSL v3.1 is equivalent to TLS v1.0, but we will refer to the technology as TLS. Both protocols are supported by all current browsers, and users can notice that they are using TLS because the protocol used is HTTPS instead of HTTP.

TLS uses digital certificates to authenticate the server, so users can trust the communication. It is possible to authenticate the client too, but that is usually done in the server using an username and a password. The connection

¹from now TLS

²Secure Socket Layer

establishment is done using a 3-step algorithm:

1. Negotiation between client and server to choose the algorithm to use during the connection.
2. Public key interchange and authentication using digital certificates.
3. Traffic encryption using symmetric-key encryption.

This 3-step algorithm is automatically done by the server, so web developers do not have to worry about that, and it is server administrator task to configure the server correctly to use TLS. That is why, although we implemented TLS in the server, we will not explain it, because it does not behoove to the report.

But we will explain some rules that are necessary to configure TLS correctly, making it secure. These rules are a summary extracted from OWASP[24], and for seeing more information and other rules not explained here, consult directly to the OWASP web.

10.2 Secure server design

Rule: use TLS for all login pages and all authenticated pages

The login page and all authenticated pages must work only over TLS, including the initial login page, called *login landing page*. If the *login landing page* is not served using TLS, an attacker can modify the login form action and users credentials can be posted to an arbitrary location. If authenticated pages are not served using TLS, an attacker can view the unencrypted session ID and compromise the session.[25]

Rule: use TLS on any networks (external and internal) transmitting sensitive data

All networks, both external and internal, that transmit data must use TLS. It is not enough to say that the internal network is restricted to employees, because if an attacker cracks that network, he can see all the data sent on the internal network.[26]

Rule: do not provide non-TLS pages for secure content

All pages available over TLS must not be available without using TLS. If a user type an URL to an HTTP page within the authenticated portion of the application, the response and all the data will be sent without using TLS, in plain text.[27]

Rule: do not perform redirects from non-TLS page to TLS login page

Usually, web developers redirect users from a non-TLS version of the login page to a TLS version. This creates an additional attack vector for a man in the middle attack.[28]

Rule: do not mix TLS and non-TLS content

A page available over TLS must not contain any data that is transmitted without using TLS. An attacker can intercept that unencrypted data and inject malicious content into the user page.[29]

Rule: use secure Cookie Flag

The secure flag must be set for all user cookies. If it is not set, an attacker can access the session cookie by tricking the user's browser into submitting a request to an unencrypted page.[30]

Rule: keep sensitive data out of the URL

Sensitive data must not be sent via URL arguments. Better places to store sensitive data are in a server side repository or in user session. The URL arguments are encrypted using TLS, but there are two scenarios where the data is exposed[31]:

- In the browser's history. The entire URL is cached in it, so everybody with access to the browser can view it.
- When the user clicks on a link to another HTTPS site.

Rule: prevent caching of sensitive data

TLS provides confidentiality for the transmitted data, but not if the data is saved at the client. Server should force the client and the intermediary proxies to not cache the data.[32]

10.3 Server certificate and protocol configuration

Rule: use an appropriate certificate authority for the application's user base

An application must never be presented with a warning that the certificate was signed by an unknown or untrusted authority. To achieve that, the administrator has to purchase a certificate from a recognized certificate authority. Self signed certificates, obviously, must never be used.[33]

Rule: only support strong cryptographic ciphers

TLS can be compromised if a weak cipher is used. The server must only support strong ciphers and to use sufficiently large key sizes. The following should be observed[34]:

- Use AES, 3DES for encryption
- Use CBC mode
- Use SHA1 for digest
- MD5 may be used within the TLS protocol
- Do not provide support for NULL ciphersuites
- Do not provide support for anonymous Diffie-Hellman
- Support ephemeral Diffie-Hellman key exchange

Rule: only support strong protocols

Some weaknesses have been identified with older SSL protocols. The best practice is only provide support for the TLS protocols.[35]

Rule: use strong keys and protect them

The private key used must be strong enough for the anticipated life time of the private key and the corresponding certificate. The current best practice is to select a key size of at least 2048. The private key must be stored in a location that is protected from an unauthorized access.[36]

10.4 Conclusion

We could see that using TLS, although is not web developer's work, but server administrator, is really important to assure the communication between the client and the server, and to authenticate this one. This way, we can trust the communication, making it private and, therefore, nobody is able to read the information exchanged.

But using TLS is not enough, because TLS strength is its cipher strength, and it is necessary to choose strong ciphers. Additionally, there are some other issues that can make the connection insecure and the server administrator has to take care about all of them.

Finally, we have to say that it is appropriate to purchase a digital certificate from a recognized certificate authority, so users will not see in their browsers any alert message.

Chapter 11

Conclusion

We could see that web security is a really important area inside computing. An unique issue can compromise not only the website, allowing attackers to take advantage of the website, buying products cheaper, for example, but also users personal data, that is really serious and in some countries is punished by the law.

To write this report, we needed a lot of work, several researches along the Web, because books about web security are obsolete at the time that they go on sale, so we could not consult any book. First, we had to search what are the main issues in web security, to define report's objectives. That was not easy due to there are different opinions, but luckily we found the OWASP, that every three years develops a list of the most common issues along the web.

Then, for each objective, we had to research how it works in order to understand it and to test the issue in the website. That was a hard part, because the examples found were simple and we could not try them on the website, so we had to make up other new ways to attack the website. Once it was done, we had to implement optimal solutions in the website and, finally, to write the corresponding part in the report, illustrating it properly.

We defined seven objectives in chapter 3, that each one corresponds to a web security issue, and we had to study all of them and solve them in the website. We can say that we achieved these objectives, because we studied and solved all of them.

We can also say that we achieved the report main objective, that was, as we say in the introduction (chapter 1), to make secure an already built online

shop against the main weaknesses of the web, so the family member who wants to open the shop is sure that the website will avoid the main issues. We described and implemented features that avoid those issues. We can say that the online shop is now safe against the main attacks.

This report can be useful for the future, to new web developers that ignore security issues in the Web, because it describes the main issues and how to solve them. It can also be useful to compare, from here a few years, how web weaknesses evolve, how new issues appear and how current issues disappear or reduce their impact.

Issues explained here are not usually taught at university although they are really important, therefore we feel really good about having learnt it. It has been a grateful work and really, really useful.

Appendix A

Source code

A.1 Source code to check password strength and password verification

```
$(document).ready(function ()
{
    var strPassword;
    var strPassword2;
    var charPassword;
    var charPassword2;
    var complexity = $("#complexity");
    var comparison = $("#comparison");
    var minPasswordLength = 8;
    var baseScore = 0, score = 0;

    var num = {};
    num.Excess = 0;
    num.Upper = 0;
    num.Numbers = 0;
    num.Symbols = 0;

    var bonus = {};
    bonus.Excess = 3;
    bonus.Upper = 4;
    bonus.Numbers = 5;
    bonus.Symbols = 5;
    bonus.Combo = 0;
    bonus.FlatLower = 0;
    bonus.FlatNumber = 0;
```

```
    outputResult ();
    $("#pass1").bind("keyup", checkVal);
    $("#pass2").bind("keyup", checkVal);

function checkVal()
{
    init ();

    if (charPassword.length >= minPasswordLength)
    {
        baseScore = 50;
        analyzeString ();
        calcComplexity ();
    }
    else
    {
        baseScore = 0;
    }

    outputResult ();
}

function init ()
{
    strPassword= $("#pass1").val ();
    charPassword = strPassword.split ("");

    strPassword2= $("#pass2").val ();
    charPassword2 = strPassword2.split ("");

    num.Excess = 0;
    num.Upper = 0;
    num.Numbers = 0;
    num.Symbols = 0;
    bonus.Combo = 0;
    bonus.FlatLower = 0;
    bonus.FlatNumber = 0;
    baseScore = 0;
    score =0;
}

function analyzeString ()
{
    for (i=0; i<charPassword.length; i++)
    {
```

```

    if (charPassword[i].match(/[A-Z]/g)) {num.Upper++;}
    if (charPassword[i].match(/[0-9]/g)) {num.Numbers++;}
    if (charPassword[i].match(/(.*[!,@,#,$,%,^,&*,?,-,~])/))
        {num.Symbols++;}
}

num.Excess = charPassword.length - minPasswordLength;

if (num.Upper && num.Numbers && num.Symbols)
{
    bonus.Combo = 25;
}

else if ((num.Upper && num.Numbers) || (num.Upper &&
    num.Symbols) || (num.Numbers && num.Symbols))
{
    bonus.Combo = 15;
}

if (strPassword.match(/^[\sa-z]+$))
{
    bonus.FlatLower = -15;
}

if (strPassword.match(/^[\s0-9]+$))
{
    bonus.FlatNumber = -35;
}
}

function calcComplexity()
{
    score = baseScore + (num.Excess*bonus.Excess) +
        (num.Upper*bonus.Upper) + (num.Numbers*bonus.Numbers) +
        (num.Symbols*bonus.Symbols) + bonus.Combo +
        bonus.FlatLower + bonus.FlatNumber;
}

function outputResult()
{
    if ($("#pass1").val() == "")
    {
        complexity.html("Enter your password").removeClass("weak
            strong stronger strongest").addClass("default");
    }
}

```

```

}
else if (charPassword.length < minPasswordLength)
{
    complexity.html("At least " + minPasswordLength+ "
        characters!").removeClass("strong stronger
        strongest").addClass("weak");
}
else if (score < 50)
{
    complexity.html("Weak!").removeClass("strong stronger
        strongest").addClass("weak");
}
else if (score >= 50 && score < 75)
{
    complexity.html("Average!").removeClass("stronger
        strongest").addClass("strong");
}
else if (score >= 75 && score < 100)
{
    complexity.html("Strong!").removeClass("strongest")
        .addClass("stronger");
}
else if (score >= 100)
{
    complexity.html("Secure!").addClass("strongest");
}

if ($("#pass2").val() == "")
{
    comparison.html("Confirm the password").removeClass("weak
        strong stronger strongest").addClass("default");
} else {
    if ($("#pass1").val() != $("#pass2").val())
    {
        comparison.html("Password different").removeClass("weak
            strong stronger strongest").addClass("weak");
    }
    if ($("#pass1").val() == $("#pass2").val()) {
        comparison.html("Passwords match").removeClass("weak
            strong stronger strongest").addClass("strongest");
    }
}
}
}
}
);

```


A.2 Password recovery source code

```
<?
//index.php
include("header.php");
cabecera("Password recovery - Foo shop");
superior_menu();
?>
<link type="text/css" href="password/css/style.css"
      rel="stylesheet" />
<script type="text/javascript"
        src="password/js/jquery.js"></script>
<script type="text/javascript"
        src="password/js/mocha.js"></script>

<script type='text/javascript'>

function formValidator(){

    var pass1 = document.getElementById('pass1');
    var pass2 = document.getElementById('pass2');

    if(isEqual(pass1, pass2, "Passwords must be equal!")){
    if(notEmpty(pass1, "You must write a password!")){
    if(isPass(pass1, "Password must have at least 8
        characters!")){
        return true;
    }
    }
    }
    return false;
}

function notEmpty(elem, helperMsg){
    if(elem.value.length == 0){
        alert(helperMsg);
        return false;
    }
    return true;
}

function isEqual(elem1, elem2, helperMsg){
```

```

    if(elem1.value == elem2.value){
        return true;
    }
    else {
        alert(helperMsg);
        return false;
    }
}

function isPass(elem, helperMsg){
    if(elem.value.length >= 8){
        return true;
    }
    else {
        alert(helperMsg);
        return false;
    }
}
</script>
<?

echo "<center>
<HR width=100% align='center'>";

switch($_GET["step"]) {
    case 1:
        if($_SESSION["login"] == 1) {
            echo "<br><h2>You have already logged in into your
            account.</h2>
            <p>You'll be redirected to your personal menu in 5
            seconds</p>";
            echo "<meta http-equiv='Refresh' content='5;URL=user.php'
            />";
        }
        else {
            if(isset($_POST["Submit"])) {
                $db = new AuxDB();
                $db->connect();

                $sql = "SELECT ID from CLIENT where EMAIL = ?";
                $params [] = array($_POST["mail"], PDO::PARAM_STR);

                $rst = $db->executeSQLs($sql, $params);
                if($db->numRows($rst) > 0) {
                    $row = $db->nextRow($rst);

```

```

    $id = $row["ID"];
    $sid = genToken(32, true);

    //Send an email with the address:
    password_recovery.php?step=2&id=$id&sid=$sid

    $db->freeResources($rst);
    unset($params);

    $sql = "INSERT INTO PASS_RECOVERY(ID, SID, CODE)
           VALUES (?, ?, ?)";
    $params[] = array($id, PDO::PARAM_INT);
    $params[] = array($sid, PDO::PARAM_STR);
    $params[] = array(null, PDO::PARAM_STR);

    $rst = $db->executeSQLs($sql, $params);
} else {
    echo "<h2>An error occurred!</h2>";
}
} else {
    echo "<h2>Password recovery:</h2>
    <form method='post'
          action='password_recovery.php?step=1'>
    <p><strong>Email: </strong>
    <input name='mail' type='text' id='mail' maxlength='40'
          size='40' />
    </p>
    <p>An email will be sent to your email address. Follow
          its instructions to restore your password</p>

    <p><input class='boton' type='Submit' name='Submit'
          value='Accept' /></p>
    </form>";
}
}
break;

case 2:
    if(isset($_GET["id"]) && isset($_GET["sid"])) {
        if(isset($_POST["Submit"])) {
            $user = new Client();
            $user->leer($_GET["id"]);

            if(strcasecmp($_POST["answer"], $user->getAnswer()) ==

```

```

    0) {
echo "<h2>Correct answer!</h2>";

    $code = genToken(8, false);
    //Send an email with the address:
        password_recovery.php?step=3&sid=" . $_GET["sid"]

    $db = new AuxDB();
    $db->connect();

    $sql = "UPDATE PASS_RECOVERY SET CODE = ? WHERE SID =
        ?";
    $params [] = array($code, PDO::PARAMSTR);
    $params [] = array($_GET["sid"], PDO::PARAMSTR);

    $rst = $db->executeSQLs($sql, $params);

    } else {
        echo "<h2>Wrong answer!</h2>";
    }
} else {
    $db = new AuxDB();
    $db->connect();

    $sql = "SELECT * from PASS_RECOVERY where ID = ? and
        SID = ?";
    $params [] = array($_GET["id"], PDO::PARAMINT);
    $params [] = array($_GET["sid"], PDO::PARAMSTR);

    $rst = $db->executeSQLs($sql, $params);
    if($db->numRows($rst) > 0) {
        $row = $db->nextRow($rst);

        $user = new Client();
        $user->leer($row['ID']);

        echo "<h2>Password recovery:</h2>
        <p><b>Security question:</b></p>
        <p>" . $user->getQuestion() . "</p>
        <form method='post'
            action='password_recovery.php?step=2&id=" .
            $_GET["id"] . "&sid=" . $_GET["sid"] . "'>
        <p><strong>Answer: </strong>
        <input name='answer' type='text' id='answer' />
        </p>

```

```

        <p><input type='Submit' name='Submit' value='Accept'
        /></p>";
    } else {
        echo "<h2>An error occurred!</h2>";
    }
}
} else {
    echo "<h2>An error occurred!</h2>";
}
break;

case 3:
    if (isset($_GET["sid"])) {
        if (isset($_POST["Submit"])) {
            $db = new AuxDB();
            $db->connect();

            $sql = "SELECT CODE, ID from PASS_RECOVERY where SID =
            ?";
            $params[] = array($_GET["sid"], PDO::PARAM_STR);

            $rst = $db->executeSQLs($sql, $params);
            if ($db->numRows($rst) > 0) {
                $row = $db->nextRow($rst);

                if (strcmp($row["CODE"], $_POST["code"]) == 0) {
                    $client = new Client();
                    $client->leer($row["ID"]);

                    $client->setPass($_POST["pass1"]);
                    $client->modify();

                    echo "<h2>Password changed!</h2>";
                } else {
                    echo "<h2>An error occurred1!</h2>";
                }
            } else {
                echo "<h2>An error occurred2!</h2>";
            }
        } else {
            echo "<h2>Password recovery:</h2>
            <p>Insert the code that you received and the new
            password:</p>
            <form method='post' onsubmit='return formValidator()'

```

APPENDIX A. SOURCE CODE

```
        action='password_recovery.php?step=3&sid=' .
        $_GET["sid"] . "'>

<p>Code: <input name='code' id='code' /></p>

<div class='block'>
    <input name='pass1' type='password' id='pass1' />
    <div id='complexity' class='default'>Enter a random
        value</div>
</div>
<div class='block'>
    <input name='pass2' type='password' id='pass2' />
    <div id='comparation' class='default'>Enter a random
        value</div>
</div>

<p><input type='Submit' name='Submit' value='Accept'
    /></p>";
    }
} else {
    echo "<h2>An error ocurred!</h2>";
}
break;

default:
    echo "<h2>An error ocurred!</h2>";
    break;
}

echo "</center>";

include("footer.php");
?>
```

Bibliography

- [1] OpenOffice.org site: <http://www.openoffice.org>
- [2] TexShop site: <http://pages.uoregon.edu/koch/texshop/>
- [3] OmniGraffle Professional site: <http://www.omnigroup.com/products/omnigraffle>
- [4] Apache web server site: <http://www.apache.org>
- [5] w3schools HTML site: <http://www.w3schools.com/html/default.asp>
- [6] PHP site: <http://www.php.net>
- [7] MySQL site: <http://www.mysql.com>
- [8] phpMyAdmin site: <http://www.phpmyadmin.net>
- [9] w3schools Javascript site: <http://www.w3schools.com/js/default.asp>
- [10] w3schools AJAX site: <http://www.w3schools.com/ajax/default.asp>
- [11] Kate site: <http://kate-editor.org>
- [12] TextMate site: <http://macromates.com>
- [13] Safari site: <http://www.apple.com/safari>
- [14] Mozilla Firefox site: <http://www.mozilla-europe.org>

- [15] Opera site: <http://www.opera.com>
- [16] The OWASP 2010 top ten: https://www.owasp.org/index.php/Top_10_2010-Main
- [17] OWASP: SQL injection prevention cheat sheet:
https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet#Defense_Option_1:_Prepared_Statements_.28Parameterized_Queries.29
- [18] OWASP: XSS prevention rules: [https://www.owasp.org/index.php/XSS_\(Cross_Site_Scripting\)_Prevention_Cheat_Sheet#XSS_Prevention_Rules](https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet#XSS_Prevention_Rules)
- [19] OWASP: password length recommendation: https://www.owasp.org/index.php/Authentication_Cheat_Sheet#Password_Length
- [20] OWASP: password complexity: https://www.owasp.org/index.php/Authentication_Cheat_Sheet#Password_complexity
- [21] NetTus+: Build a simple password strength checker:
<http://net.tutsplus.com/tutorials/javascript-ajax/build-a-simple-password-strength-checker/>
- [22] jQuery site: <http://jquery.com/>
- [23] itnewb: Generating session IDs and random passwords with PHP:
<http://www.itnewb.com/v/Generating-Session-IDs-and-Random-Passwords-with-PHP>
- [24] OWASP: Transport Layer Cheat Sheet:
https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet
- [25] OWASP: Use TLS for all login pages and all authenticated pages:
https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet#Rule_-_Use_TLS_for_All_Login_Pages_and_All_Authenticated_Pages

- [26] OWASP: Use TLS on any networks transmitting sensitive data:
https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet#Rule_-_Use_TLS_on_Any_Networks_.28External_and_Internal.29_Transmitting_Sensitive_Data
- [27] OWASP: do not provide non-TLS pages for secure content:
https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet#Rule_-_Do_Not_Provide_Non-TLS_Pages_for_Secure_Content
- [28] OWASP: do not perform redirects from non-TLS pages to TLS login page:
https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet#Rule_-_Do_Not_Perform_Redirects_from_Non-TLS_Page_to_TLS_Login_Page
- [29] OWASP: do not mix TLS and non-TLS content:
https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet#Rule_-_Do_Not_Mix_TLS_and_Non-TLS_Content
- [30] OWASP: use secure Cookie Flag:
https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet#Rule_-_Use_.22Secure.22_Cookie_Flag
- [31] OWASP: keep sensitive data out of the URL:
https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet#Rule_-_Keep_Sensitive_Data_Out_of_the_URL
- [32] OWASP: prevent caching of sensitive data:
https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet#Rule_-_Prevent_Caching_of_Sensitive_Data
- [33] OWASP: use an appropriate certificate authority for the application's user base:
https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet#Rule_-_Use_an_Appropriate_Certificate_Authority_for_the_Application.27s_User_Base

- [34] OWASP: only support strong cryptographic ciphers:
https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet#Rule_-_Only_Support_Strong_Cryptographic_Ciphers
- [35] OWASP: only support strong protocols:
https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet#Rule_-_Only_Support_Strong_Protocols
- [36] OWASP: use strong keys and protect them:
https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet#Rule_-_Use_Strong_Keys_.26_Protect_Them

List of Figures

2.1	Front-end use case diagram	13
2.2	Back-end use case diagram	15
2.3	Database entity-relationship diagram	17
2.4	Database logical diagram	21
2.5	Class diagram of the website	26
4.1	Website login page	31
4.2	Example 1: data introduced	33
4.3	Example 1: login succesful	34
4.4	Example 2: data introduced	36
4.5	Example 2: login succesful	37
4.6	Example 3: data introduced	38
4.7	Rejected attack	40
4.8	Checking wrong email address	43
5.1	Prepayment page	51
5.2	View order page	52
5.3	Admin login page	53
5.4	Attack on prepayment	55
5.5	Fake admin login page	56

5.6	Failed attack	58
6.1	Registering succesful with a wrong passowrd	65
6.2	Registering using an insecure password	66
6.3	Wrong error message: it gives information about IDs in the database	68
6.4	Wrong error message: it gives information about emails in the database	68
6.5	Password verification: correct password and confirmed	70
6.6	Password verification: correct password but not confirmed	71
6.7	Password verification: password not confirmed	74
6.8	Password verification: password empty	74
6.9	Password verification: password too short	75
6.10	Registering process: security question and answer added	76
6.11	Password recovery: asking the email	77
6.12	Generic error message showed when an error occurred	77
6.13	Password recovery: asking the secret question	78
6.14	Password recovery: asking the code and the new password	79
6.15	Password recovery: password successfully changed	79
6.16	Generic error message showed when ID or email exists in the database	80
6.17	Blocked account	82
6.18	Blocked account after 3 attempts	82
6.19	Session timeout	86
7.1	Insecure reference: an empty record is fetched	88
7.2	Insecure reference when trying to access to an inexistent reference	89
8.1	Message displayed when the token is not verified	96

LIST OF FIGURES

9.1 Access to a forbidden page without authenticate 98

9.2 Error message when trying to access to a forbidden page without authenticate 101

List of Tables

3.1 Objectives table	28
--------------------------------	----

Listings

4.1	Login check source code	30
4.2	Login check query source code	32
4.3	Normal query	32
4.4	Case 1: attack knowing an email address	32
4.5	Case 1: executed query	35
4.6	Case 2: attack unknowing any data	35
4.7	Case 3: hypothetical attack deleting data	38
4.8	Rejected attack	39
4.9	Prepared statements source code	40
4.10	Escape user input source code	41
4.11	Limit input data source code	42
4.12	Check input data source code	43
5.1	Rule #1 generic examples	47
5.2	Rule #2 generic examples	48
5.3	Rule #3 generic examples	48
5.4	Rule #4 generic examples	49
5.5	Rule #5 generic example	49
5.6	Rule #0 generic examples	50
5.7	Inserted malicious comment	53
5.8	Fake login page source code	54

5.9	Solution adopted	57
6.1	User verification and error messages	67
6.2	Password verification	73
6.3	Password verification	80
6.4	Session ID generator	83
6.5	Session timeout start	84
6.6	Session timeout checking	84
6.7	Session timeout template	85
7.1	Session timeout template	89
8.1	modify_user.php template source code	92
8.2	Malicious script source code	93
8.3	Line to add to user_login.php	93
8.4	modify_user.php template source code with the issue solved . .	94
9.1	Login admin user template source code	99
9.2	All admin pages template source code	100

