Web Security

01/8/2015

KEA Medie/IT

Jimmi Nielsen & Mikkel Plejdrup

Table of contents

Introduction....3
Project features....4-6
Techology stack....6
Biggest threats....6
Client-side manipulation....7
XSS....8
CSRF....8 - 9
SQL Injection....9-10
General security measures....10-11
Code examples....11-12
Prepared statements....12
Encryption....13
Validator, sanitize & encoding....12 - 14
Reflection....14
Discussion....15
Conclusion....15 - 16
References....17-18
Appendix....18 - 19

Introduction

Since almost nothing is secure on the internet, shown just as recent as in the Sony hack event (Geigner 2014). Learning about how you can protect yourself and web related data on the net is essential for us as web-developers. We need to be able to prevent common security issues, because they happen all the time (Info Security 2014). Because they happen all the time, we have to know about how to prevent them, or at least prevent the biggest and most common security threats.
You cannot be 100% secure all the time, but at least you can make the life of potential malicious hacker as hard as possible. There is different kind of hackers, for example there are black hats and white hats (Wikipedia Hacker 2014), black hats are the ones you have to be careful for, because they are the ones who try to steal your data illegally. White hats on the other hand are paid employees, who are paid to try to hack and test a site.
In this module we are taught how to build a secure website and also common ways of attacking it, thus teaching us to be white hats. This will help us develop our project, since we then will be able to test and correct ourselves.
The assignment is to make a website, with enough content, data and possible security flaws for it to be hacked. Plus we have to make it as secure as we possible can. We have chosen to use PHP in our project, since that was what was taught and also because it is a pretty common language and is fitting for these types of smaller websites. It can be discussed whether or not an OOP language like JAVA, could be more secure (Lowmiller 2013).
The projects should have different features such as picture uploading, Facebook-like wall and creation of user profiles. These are some of the main things we have to secure.

Problem

How can we build a secure website system, where the content, functions and all of the data is secured against possible attacks?

Facebook wall

In order to make the site exploitable it was required to have some sort of Facebook wall with a comment section so you could test your newly learned hacking skills.
In our case we made a comment section to each image.
When the user clicks on the desired image, you will be taken to a comment page with a generated random nr, which is generated when the topic is been posted by the admin, more on that afterwards.

```
$commentNr = $_GET['comment'];
$resultcomment = $mysqli->query("select randomnr from content where randomnr ='" . $commentNr . "'");
$numberOfRowscomment = mysqli_num_rows($resultcomment);
if ($numberOfRowscomment == 0) {
    header('Location: 404.html');
}
if (!$commentNr) {
    header('Location: 404.html');
}
$sLength = strlen($commentNr);
if ($sLength != 7) {
    header('Location: 404.html');
}
if (!is_numeric($commentNr)) {
    header('Location: 404.html');
}
```

In order to check if the user is trying to get to the comment site without login or trying to use SQL injection through the URL we made some security measures where we check for all possible hits.
We used a WYSIWYG editor, NicEdit[1] because of all the extra functions it has but also for the build-in sanitizing and encoding, this will elaborated later in the report.

Post topic

In order to be able to post topics you have to have admin rights.
A topic consists of an image and a title people can discuss.
We sanitized the input even though it requires an admin login to get to the admin site.

Logging

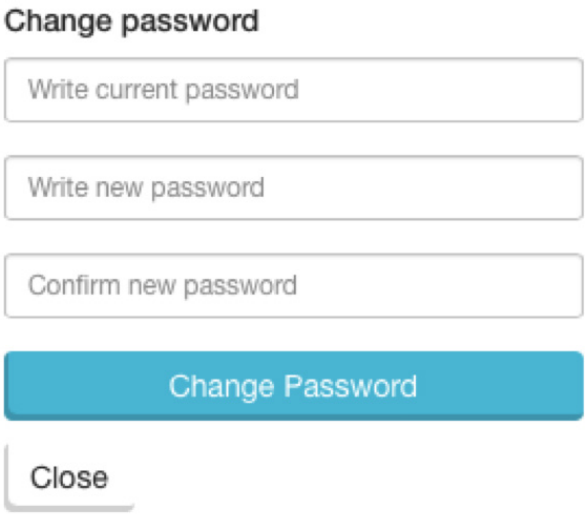
A smart way of keeping track of your users is to log every login entry both if it is success or fail.
In our case we logged every login-try based on the IP address. If the IP address appeared more than 3 times we checked how long time since last login try. The user is banned for 5 min before he/she can make a new attempt. It is discussed if best practice

1 http://www.nicedit.com/

is through IP address or by username. If you are on a shared network where every computer has the IP address everybody would be blocked. If you check by username you will block only that user no matter what connection he/she uses.

Multilevel authentication

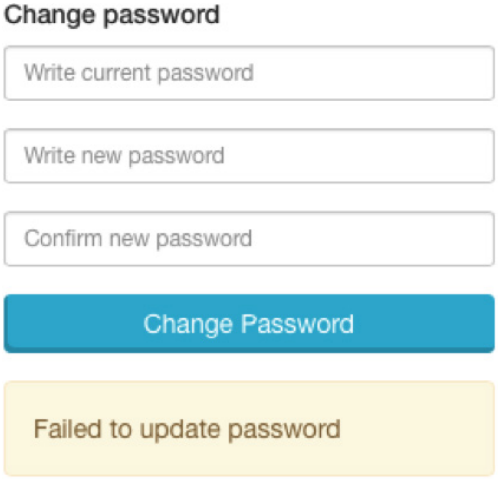
We decided not to have multi level login, because we did not want the login process to be too long. So instead of the multilevel login we made the function to change your password, which works in a similar way.



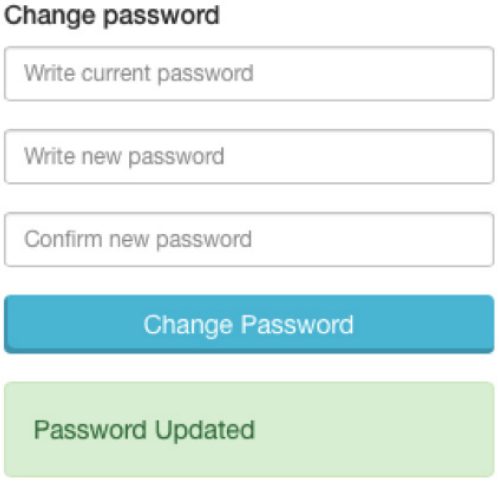
To change the password you cannot simply write a new one, you have to take certain measures in order for it to work. This also complicates things for an attacker who might be trying to change your password.

To change the password: 1. You must know the current password and 2. You have to write the new password twice. This creates different levels of security, where every parameter must be filled in order for it to work. If you do not meet the requirements you will receive an error message (picture 1.), if you do meet the requirements you will receive a success message (picture 2).

1.



2.



This is done like this in the code, where all have to be true to be executed:

```
if ($oldPasswordFromDatabase === $salted_hashed_check_old_password && $cNewPassword == $cNewPassword2) {
```

More details about the password code will be explained in "Code Examples".

Technology stack

In order to focus more on the functions and the security that comes with them we used Twitter bootstrap[1]. Which gives a nice intuitive design and easy frontend functionality. The framework is ideal for a web-developer looking to ease and optimize their work.

As server side language/backend we used PHP both because the lecture examples were in PHP and of its widely use in web pages and applications which makes it more easy to find help and documentation.
We used MySQL to store data. When you put them together it opens countless possibilities for your web application.

Biggest Threats

Based on the lectures and based on the risk analysis we have made, we will now explain the biggest threats against our system.
Our risk analysis is based on the list from OWASP which describes the top ten biggest threats (OWASP top 10 2014), but it is also based what threats are most common for a system that has the features that our system has. Features like several input fields, user creation and outputting of data from the database.

1 www.getbootstrap.com

Client-side manipulation

One of the first steps you can take when securing your website, is limiting what can be changed and manipulated on the client-side. A malicious client-side manipulation could be if an attacker changed a price or a cookie, simply just by opening the console and write in a different value. If he could do that he could possible login as different users and he could pay 1 kr. for an item that normally costs 2000 kr. just by changing the value. Basically you should try to prevent the attacker from being able to change anything on the client-side, which could be sent to the server, that was not intended to be changed. (OWASP CSRM 2014) Our project has a lot of input fields, so it would be sensible to make them more secure. For that we have used JQuery Validation, which is JQuery plugin that contains method for validating input fields (JQuery validate 2014). So if the input field is defined as an email input, you have to write a proper email address for the containing form to be submitted. Plus you can set an input field to be required, which means that it can not be empty when you submit the form. This does supply some security, but is only one step of securing your site from attackers trying to hack your site through the input fields. Other methods will be discussed later in the report.
Client-side manipulation applies to the JavaScript files too, if the attacker easily can see what the JavaScript file contains he has higher chance of changing it. To prevent that from happening you can obfuscate the JavaScript, which means you transform your JavaScript into obfuscated code, which is very difficult for humans to decode (Wikipedia Obfuscation 2014).

That means your JavaScript will be changed from something like this:

```
$(document).ready(function () {
    $(document).on("click", "#logOut", function () {
        event.preventDefault();
        $(".logOut").submit();
    });
```

To this (This example are both the beginning of the file main.js):

```
var _0x10aa=["\x63\x6C\x69\x63\x6B","\x23\x6C\x6F\x67\x4F\x75\x74",
```

XSS (Cross Site Scripting)

A big threat we need to consider is XSS, since it is such a big part of all attacks and even 75% of all US government sites suffers from it (McCaney 2011). Cross-site scripting allows hackers to inject code into your site; either from input fields or possible through upload forms. For example they could upload a JavaScript script that sends a request with users cookie to their own sites, and then retrieving the cookie, in the error log. There are two types of XSS; Reflected and Stored (OWASP XSS 2014). In order for reflected XSS to work, the user must be logged in and press a link that the malicious attacker has made. On the other hand Stored XSS is when the code is stored in the database and can cause greater harm since it affects everyone who views the output from the database.
There are several ways of preventing this from happening, for example you can sanitize the input fields and encode the output, we will explain this with examples later in this report in Code Examples Sanitize and Encoding (p. 88).
Other measure could be to set a low time for the session and setting up a firewall.
Example of XXS from OWASP:

```
<SCRIPT type="text/javascript">
var adr = '../evil.php?cakemonster=' + escape(document.cookie);
</SCRIPT>
```

CSRF (Cross site request forgery)

Unlike XSS, Cross-site request forgery does not actually steal the cookie or the data, instead it uses the end-users to execute request, without them knowing. (OWASP CSRF 2014) That way the attacker can make changes using someone else authentication, to change user info like their password, or if a person has admin rights, change the system. The way an attacker can get an end-user to do this is to make them click a link, which leads them to an unsafe server with a page that generates the request. The link must be an enticing link for a user to want to click on it; this is done on Facebook sometimes, where it usual-

ly links to a video (Doshi 2011). There is also OSRF, which is very similar, but is instead stored on a server, similar to Stored XSS.
If you have protected yourself against XSS attacks, you have also protected yourself against OSRF, when you sanitize and encode the output for example. You could still be vulnerable to CSRF though, so other measures you could follow is using a token, a token that is stored in a session and then checked every time a request is received and then changed. This makes sure that the end-user is the one making the request.
Example of CSRF from OWASP:

```
<a href="http://bank.com/transfer.do?acct=MARIA&amount=100000">View my Pictures!</a>
```

SQL Injection

Topping the list of the OWASP list is SQL injection, which is when an attacker injects an SQL query from an input field from the client to the server (OWASP SQL Injection 2014).
The reason it tops the list is because it is a dangerous attack and also because it is an attack that happens a lot. (Ragan, 2012)
These kind of attacks can be severe, if the attacker succeeds, he can basically control the database, performing Insert, Update and Delete on database content and control the database in general.
There are different ways you can try to prevent this, for example you can use "mysql_real_escape_string()", which basically skips over the special characters that might be used in a SQL injection.(PHP.net 2014) Sanitizing your input fields also helps prevent this attack, since it strips out characters that might have been used in a SQL query.
Another more effective way is using prepared statements, prepared statements separates the actual SQL query from the input fields. For this you can use PHP data Objects, which have some very useful methods for prepared statements (Wurzer, 2012). We will explain this with examples later in this report in Code

Example of an SQL injection:

```
SELECT * FROM Users WHERE UserId = 105 or 1=1
```

General security measures

Firewall
An essential security measure is a firewall. Our server is a linux server which includes a standard firewall, IPTables. The firewall checks the network traffic with a set of rules that defines the characteristics that a package must have to match.(Beginners guide to IPTables) Thereafter decide the action that should be taken. We implemented a set of simple rules that allows a set of ports through TCP, HTTPS and SSH.

SSL
Secure Sockets Layer(SSL) is a technology that secures the connection between a server and a client, typically a web server and a browser.(What is SSL) This allows us to put https in front of our ip address making it a trusted connection.

PHP Config
When developing in Php there are some security measures you have to disable in the php.ini file. Such as including a url instead of a local file, this could be a threat if the developers owning the file changed the content and used it for hacking. Also 'exec' and 'system' should also be disabled. They are used to execute external programs. If enabled a user can enter any command and execute on your server.

Encryption of data
Encryption is essential when building secure sites.
There are two ways of encrypting, symmetric and asymmetric. In symmetric encryption you use the same encryption key to encrypt and decrypt. Therefore all decoders must have the same key.
In asymmetric encryption also called public-key cryptography you have both a private and a public key. So each user has a public key which they can send out and will be used by other

users to encrypt the message that will be send back. Each user also have a private key that they save and is used to decrypt the message that they will receive. The key for encrypting is not the same for decrypting.

Code Examples

Salting and hashing

When you have a website, that deals with user creation and user accounts, it is important to secure the passwords, in the case of your database getting hacked (Geigner 2014) Sony for example did not do this, they stored their passwords as plain-text, which in effect put a lot of their users private info in jeopardy (Geigner 2014). To secure the password you can use Salting and Hashing. Hashing a password means that you turn your password into a fixed length set of random characters, which cannot be reversed. (Defuse Seucrity 2014) These hashes are different, even with tiniest change in the password, For example "password123" output will be different from "password122". This will help you protect your password a great deal, but these hashes can be cracked, for example with a brute force attack(Defuse Seucrity 2014). Where the attacker simply will go through all the possible combinations up to a certain length. This does require a lot of computer strength though, and also depends on the type of hashing. We used Sha256 to hash our passwords as you can see in this example.

```
$passwordHashed = hash('sha256', $saltedpassword);
```

Sha256 creates a string of 64 random characters.
To try to mitigate this problem, we can effectively use Salt. A salt, in this case, is generated random number, which should be the same length as the total hashed password (64 characters) (Defuse Seucrity 2014). The salt is then concatenated with the password, which is then hashed, as the example above. The salt is then stored in the database, so that it can be used later as key to compare the password with the stored password.
In this example, you can see how the salt is created, concatenated with the password, hashed and then stored in the database:

```
$newSalt = generateSalt();
$saltedpassword = $cPassword . $newSalt;
$passwordHashed = hash('sha256', $saltedpassword);
$mysqli->query("INSERT INTO users(first_name,last_name,email,salt,password) VALUES('$newFirst_name','$newLast_name','$newEma
```

In this example, you can see how the stored salt is used to compare with stored salted and hashed password. Also a new salt is created along with the newly created password, for extra security.

```
while ($row = $result->fetch_row()) {
    $saltFromDatabase = $row[0];
    $oldPasswordFromDatabase = $row[1];
    $salted_check_old_password = $cOldPassword . $saltFromDatabase;
    $salted_hashed_check_old_password = hash('sha256', $salted_check_old_password);
    if ($oldPasswordFromDatabase === $salted_hashed_check_old_password && $cNewPassword == $cNewPassword2) {


        $newSalt2 = generateSalt();
        $salted_new_password = $cNewPassword . $newSalt2;
        $new_password_salted_hashed = hash('sha256', $salted_new_password);
        $mysqli->query("update users set password='" . $new_password_salted_hashed . "', salt='" . $newSalt2
```

Prepared Statements

As previously mentioned one way of avoiding SQL Injection is by using prepared statements. Prepared statements is a way of separating the input fields input, from the SQL query you perform on the server (Wurzer, 2012). That way you have a SQL query prepared with a variable, when you receive the data you then bind the data to that variable. This for example makes sure that an attacker can't write ""."$_POST["email"]."'" in the input field and get a successful query.
In further development we could implement prepared statements, but due to time constraint and other development priorities it is not implemented in this version.
When we test our site, it is already secure against SQL injection, so it could be discussed whether is it necessary.
To show an example of how to do it, we altered the function, where we get the picture belonging to a certain topic. First

you set up the PDO, and then the query is written, the incoming data is separated from the SQL query, and is bind to the parameter ":secretNr". After that, the whole thing is executed and finally the result is fetched and put into a variable.

```
$dbh=new PDO('mysql:host:localhost','root','');
$sth = $dbh->prepare('select * from content where randomnr= :secretNr');
$sth->bindParam(':secretNr',  $_POST['contentnr'],PDO::PARAM_INT);
$sth->execute();
$result = $sth->fetchAll();
```

Encryption

There is always a probability that your database will get compromised and then the data will be compromised as well plus the data are still accessible to the hosting company where the database is hosted. To add another layer of security we encrypted everything inserted in the database.
We used symmetric encryption means you only have one key for encryption that also works for decryption. It is very easy and fast to use but the downside is that you need to keep the encryption key secure.

```
function encryptThis($input) {
    $MASTERKEY = "Durant35";
    $td = mcrypt_module_open('tripledes', '', 'ecb', '');
    $iv = mcrypt_create_iv(mcrypt_enc_get_iv_size($td), MCRYPT_RAND);
    mcrypt_generic_init($td, $MASTERKEY, $iv);
    $crypted_value = mcrypt_generic($td, $input);
    mcrypt_generic_deinit($td);
    mcrypt_module_close($td);
    return base64_encode($crypted_value);
}

function decryptThis($input2) {
    $MASTERKEY = "Durant35";
    $td = mcrypt_module_open('tripledes', '', 'ecb', '');
    $iv = mcrypt_create_iv(mcrypt_enc_get_iv_size($td), MCRYPT_RAND);
    mcrypt_generic_init($td, $MASTERKEY, $iv);
    $decrypted_value = mdecrypt_generic($td, base64_decode($input2));
    mcrypt_generic_deinit($td);
    mcrypt_module_close($td);
    return $decrypted_value;
}
```

Validator, Sanitize and Encoding

When developing applications with user input such as login forms and text inputs that are connected to a database. In order to secure yourself against previously mentioned XSS and Sql injection you can do multiple things and still make it user friendly.
The first basic thing is using a validator so email should contain '@' then the zero tolerate way were you clean the value of the inputs if they contain certain words that could be associated with a Sql Query or inserting a script.

```php
function cleanInput($input) {
    $search = array(
        '@<script[^>]*?>.*?</script>@si', // Strip out javascript
        '@<[\/\!]*?[^<>]*?>@si', // Strip out HTML tags
        '@<style[^>]*?>.*?</style>@siU', // Strip style tags properly
        '@<![\s\S]*?--[ \t\n\r]*>@'         // Strip multi-line comments
    );
    $output = preg_replace($search, '', $input);
    return $output;
}
```

When your application contains a comment section you are not able to sanitize if the comment is about anything code related. Therefor you can encode the value meaning use special characters.

Here someone has tried to make a simple <script> to see if he/she was able to insert a script directly to the page.
We used a wysiwyg (What you see is what you get) editor, NicEdit which had done all the hard work for us and encoding every possible threats.

Reflection

This course has added a lot to our mindset when developing a web application. With security threats becoming more and more common, a security aspect will always come in handy.

Discussion

When developing a web application and has security as the main goal you see the importance and all the aspects you have to deal with. After seeing how easy a script or similar can be used as an attack, we would, when developing from now on, never not use sanitizing. So simple can prevent so much. The Sony hack made it obvious for us that salting and hashing is a must, even for small web applications like ours. A key factor in security is encryption which makes it that much harder for the hacker to get your data. You must focus on user experience while making the site secure. The WYSIWYG editor, NicEdit we used made it so much easier and does a good job securing the inputs. Plenty has tried to insert script or sql without any harm done.

Conclusion

In our process of completing this project, we learned that there are several layers that exists, when you want to secure a website and its content. Because of that you have to make a choice of how many security measurements you want or need to take in order for you to secure your specific site. In our case our project dealt with user accounts and a high degree of data flowing back and forth between the client and the server. That meant we needed to secure the different input fields, upload form and the places where the data is outputted. We achieved this with sanitizing and encoding, which helped secure us against XSS and CSRF. This also helped us to prevent SQL injections in those areas. Another layer of security was more general security measures, such as firewall and SSL. By implementing SSL we made sure it was a secure connection between the client and the server. By implementing the firewall, we achieved control over which ports could access the site. We also learned that it is a highly iterative process, where testing and refactoring is big part of the process. In this case it means you implement a security measure and then you play the role of the attacker and you try to hack your own site. But our ability to test our site is limited to our skills as attackers,

therefore to make sure the data was safe, if we were attacked, we encrypted the data and we salted and hashed the passwords. This made sure our (fictional) users data was secure, in the case of a breach in security.

References

Web articles

Beginners guide to IPTables
http://www.howtogeek.com/177621/the-beginners-guide-to-iptables-the-linux-firewall/
Download 6/1/15

What is SSL
https://www.digicert.com/ssl.htm
Download 5/1/15

Doshi, 2011, 'Please send me your facebook anti csrf token' Symantec.
http://www.symantec.com/connect/blogs/please-send-me-your-facebook-anti-csrf-token
Download 21/12/14

Wurzer, 2012, 'Why you should be using PHP's PDO for the database access' Tutsplus.
http://code.tutsplus.com/tutorials/why-you-should-be-using-phps-pdo-for-database-access--net-12059
Download 21/12/14

Geigner, 2014, 'Shocking sony learned no password lessons after 2011 PSN hack', Tech Dirt.
https://www.techdirt.com/articles/20141204/12032329332/shocking-sony-learned-no-password-lessons-after-2011-psn-hack.shtml
Download 21/12/14

Defuse Seucrity, 2014 'Hashing Security ', Crackstation.
https://crackstation.net/hashing-security.htm
Download 21/12/14

Links

OWASP Top 10
https://www.owasp.org/index.php/Top_10_2013-Top_10
Download 21/12/14

OWASP CSRF
https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)
Download 21/12/14

OWASP SQL Injection
https://www.owasp.org/index.php/SQL_Injection
Download 21/12/14

PHP.net
http://php.net/manual/en/function.mysql-real-escape-string.php
Download 21/12/14

Wikipedia Sha-2
http://en.wikipedia.org/wiki/SHA-2
Download 21/12/14

Wikipedia Obfuscation
http://en.wikipedia.org/wiki/Obfuscation_(software)
Download 21/12/14

OWASP CSRM
https://www.owasp.org/index.php/Testing_for_Client_Side_Resource_Manipulation_(OTG-CLIENT-006)
Download 21/12/14

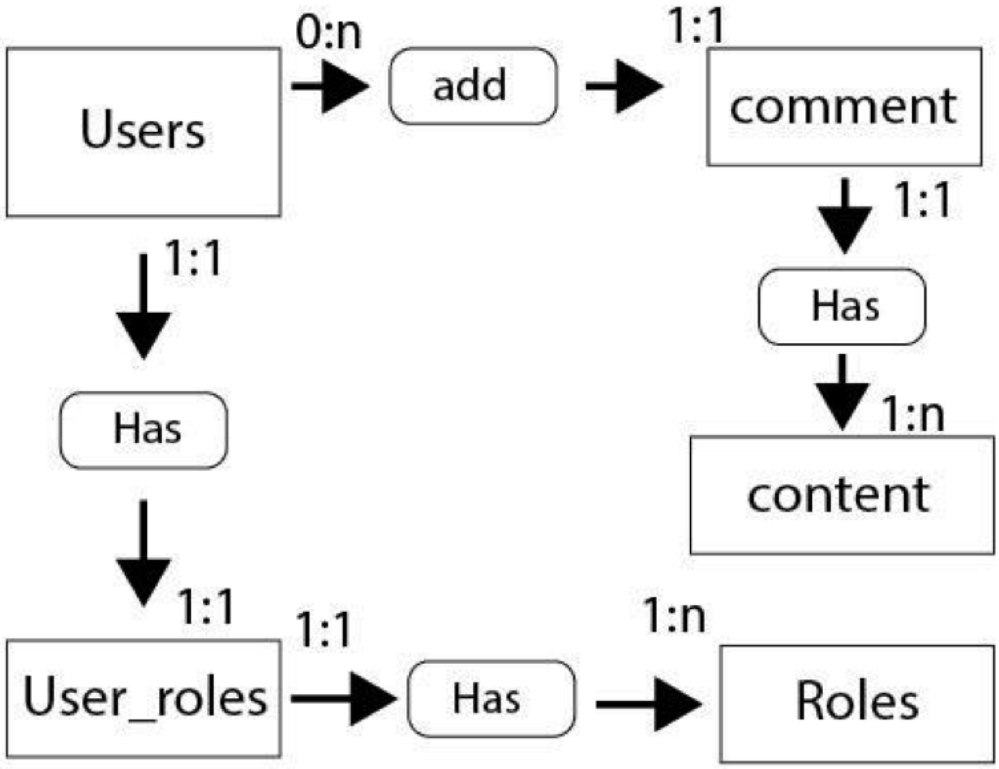
OWASP XSS
https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
Download 21/12/14

JQuery validate.
http://jqueryvalidation.org
Download 21/12/14

Wikipedia Hacker
http://en.wikipedia.org/wiki/Hacker_(computer_security)
Download 21/12/14

Appendix

ER-diagram



Risk analysis

Risk	Description	Affected systems	Propability [1-10]	Propability desc	Severity [1-10]	Severity desc	Mitigation	Control [1-10]	Risk factor
SQL Injection	sent to interpreter as a command	Server, All Data, website	10	The biggest threat	8	Get all the data	Prepared Statement	7	560
Broken authentication	Session managment etc implent	Website, user info, users	7	System design	6	Get user data	Proper session man.	9	
XSS	Excute scripts in victims browser	website, user info, session	6	Human error	6	get user data	Sanitize encoding	10	
Insecure references	reference to an internatl fil	server, data,	5	Human error	9	Get data	Proper references	9	
Security Misconfig.	Misconfigured security settings	Server, data, website	5	System design	8	Get data	Proper configuration	10	
Not encrypting data	Not encrypting sensitive data	User data	4	Data handling	8	get user data	encrypt data	10	
Access control	Control request sent for functions	website user info	4	System design	5	get user data	control access	9	
CSRF	Hacker forges request from users	website, user info	4	System design	5	user data, data	encode, control acc.	8	
using unsafe framewor	unsafe component in own system	server, all data website	3	System design	8	get all the data	proper design	7	
Unvalidated redirects	hackers can redirect to other sites	user info,	3	System design	4	user data	Validate redirects	6	
Not using SSL	easier to hack if its http	website, user info	3	System design	5	data	SSL	9	