PBA Web Development

KEA - Copenhagen School of Design and Technology

Security in Web Development

So Much Trivia

Web Application

Teachers:
Dany Kallas

Ralf Patrik Blaga
Johannes Otto Skjærbæk
Group 15

26 - MAY - 2017

Table of Contents

Introduction

The goal of the exam project was to develop a prototype of a web page/application, focusing specifically on the security challenges inherent to developing for the web. The following report will detail our thought processes, choices, and reasons for making those choices in detail.

Analysis

Project Description

In an ever growing and highly sociable world, the Internet is becoming one of the most desired utilities, even trumping heat or water sometimes. With so many teenagers having access to the Internet, Team So Much Trivia has set out to create an interactive and easy to use party application.

We plan on developing a web application that is built as an entertainment system that combines the uses of multiple gadgets, such as laptops and smartphones. We plan on creating a platform that will allow us to quickly connect players with each other, so they can start playing as fast as possible.

The players will have to create accounts to be able to set up or join an already existing gaming room. Once a gaming room is created, they will be able to invite other users to join their room via a room id. Once more people join the room, people will be able to chat with their competitors before actually starting the game, and our system will keep track of the scored points during the game.

Furthermore, the user will be able communicate with all users of the application, including the administrators of the game (our group), and he will also be able to submit questions that could potentially be added to the main series of questions in the quiz.

The delivered product will be feasible and functional. A minimum viable prototype, that will, due to time constraints, be focused on security aspects rather than adding bells and whistles to the game experience. Our project is very modular, which allows us to build a fully functional product with an easy basic background and, if time allows it, we can expand further with different add-ons and new game modes.

Requirements

An advantage of the trivia game project is that it can be made with a modular approach, fitting each new element, or several new elements, into one iteration so that we always have a working prototype.

To reflect this nature, we can divide the project using the MoSCoW prioritization technique which allows us to divide the project into the 4 groups presented by the technique, Must have, Should have, Could have and Won't have. Into the first and second priority groups, or "Need To Have" and "Nice To Have", go what we must have for the project to have basic functionality, and what would be nice to have to improve on this very basic version of the project.

1st Priority: MUST have

The first priority of the team is to be able to deliver the minimum usable subset of our game. The must have list is critical for the delivery timebox, every feature on this list must be completed in order to consider the project a success. The website must have the following features:

- Multilevel login with backend authentication
- New User Registration
- Data stored in cookie
- CMS-like interface for user input
- User Chats
- Image Upload
- Integration with external service
- Security against SQL injection, XSS, CSRF and Client side manipulation

2nd Priority: Should have

Once the above features of the platform are implemented, the team will move on and work on the features presented on the should have list. The features presented on this list are important but not crucial to the development of our prototype. The website should have the following features:

- Based on a room name/code, accept users into a common “room” and let them set their user name.
- In this room, display information (questions) to the users in the room, and allow them to pick between provided answers.
- Display whether the user’s answer was correct or not, keep score in the room of who has answered how many questions correctly.
- The option to submit more questions, so they can be added to the game.

3rd Priority: Could have

Due to the limited timebox several bells and whistles could be implemented into the product, however, they are not necessary to the proper use of our application. The team will try to incorporate a few element from this list if the time and resources permit. The website could have the following features:

- A log page for the administrators of the page
- An online leaderboard in which players can see their scores
- A series of themed questions
- Responsiveness

4th Priority: Won’t have

Items labeled as won’t have are the items that are out of scope in the current state of the project. Just because a bullet point is out of scope, does not mean that the team did not consider it as an option to improve the overall general feel and aesthetic of the application. The Won't Have list is not too long, but crucial on a larger scale of the application, especially for a modular application. The current version of the website will not have:

- A mobile application specific for the game
- Support for Browsers on Smart TV's and gaming consoles

Risk Analysis

Every project has its own risks and dangers it can face, and this is not exclusive to Web development and security. Knowing that these things happen, we decided to go for a Minimum Risks Maximum Rewards policy. This means that the team will try to avoid risks and try to defend against the 5 most crucial security issues according to the 2013 OWASP report[1].

While we cover more security risks in the Risk Analysis[2], we will talk specifically about the top 5 in this section. For coding specifics of how they were solved, please refer to the Implementation section of this report.

Injection

In our case, specifically SQL Injection. The danger is that, through user input, people gain indirect access to the database, and this can obviously be used for malicious purposes if steps are not taken to mitigate the problem. Not only can SQL injection result in data loss or corruption, it can even potentially lead to complete host takeover.

Broken Authentication and Session Management

Especially for a system such as the one we have made, where authentication and session management has to be built from the ground up, it would be arrogant to assume that

[1] https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project#tab=OWASP_Top_10_for_2013
[2] See: Appendix 2: Risk Sheet

every eventuality and possibility is accounted for. This same issue can be encountered on many pages on the web, in many variations.

Examples include: Exposing the session ID in links, not setting a session timeout or setting one for an unduly long period, or even something as simple as not properly protecting passwords in the database should someone gain access. It is safer to assume that someone will, eventually, and thus passwords need to be properly hashed to ensure that this eventuality is protected against as well.

This issue also covers session fixation attacks, where a session ID is taken over by the attacker to use a user's privileges with malicious intent.

Cross-Site Scripting (XSS)

XSS refers to injecting hostile client-side code into the system, such that it is stored in the database and later displayed for users of the site to cause them issues. Examples include redirection to another site with cookie information if the cookie is accessible via Javascript, and just general injection of annoying or malicious code. Alerts, ads, etc.

Insecure Direct Object References

Allowing users of the site to change variables so that they might access an unintended part of site functionality. This occurs of authorization level is not properly checked when someone attempts to gain access to a certain part of the site. For example, if an admin function is hidden only by virtue of the only public call available being made from an admin-only page, then it is insecure, as someone could refer to the function from anywhere and, if no authorization check is made, perform the given admin action.

Security Misconfiguration

General problems with security configurations across the entire stack of the web page, from server to actual presentation. This can include things like displaying directory contents publicly, default accounts and passwords being used at any level of the server or front end, public and too-detailed error reporting, etc. Given that we are not experts at server setup and configuration, this is difficult to protect against in full, but we can at least try to do what we can.

Design Principles

Since we want to create an application with lasting security, as much as such a thing is realistic, we are going to focus on Defense in Depth, Securing the Weakest Link, and Failing Securely. The idea is obfuscation, and it is also a continuation of the modular approach mentioned above. If we partition our application into many different “security modules”, we can create security for each of these imaginary modules, such that the application will be difficult to truly break.

Since the base case for any function will be no access, and only granting access that is needed, there is no single point of failure. If a function only needs read permission, it only has read permission. If this function should be exposed, we have not simultaneously given the attacker permission to create, update or delete.

With the above security premise laid out, we will cover, in brief, why we have chosen not to focus on some security design principles, and why we have chosen to focus on those we do focus on in greater detail at the end.

Lower Priority Principles

As we cannot focus on all design principles at the same time, some will naturally fall by the wayside. This does not imply total ignorance, but rather that some principles will be observed only in brief or on a more surface layer. Though we do not, for example, focus on Least Privilege, we don’t plan to simply spew privileges everywhere.

It is common sense, and not particularly difficult or time-consuming, to limit privileges to what makes sense. The lack of focus, in such a case, simply means that we will not pour the majority of our efforts into these principles, but rather observe them in a more cursory fashion.

We live in a world where we have to assume that security failures are an inevitability rather than a possibility, and this simple fact means that we simply cannot outright ignore any principles. Of course, to reduce workload in the case of this particular school project, we are forced to prioritise, and we cannot implement some of the more odious principles to a satisfactory degree while holding to our focused principles, but we may still try.

Encrypting all data we hold, and perhaps never holding some critical data, goes some way towards implementing the principle of Never Assuming That Your Secrets Are Safe, even though we, again, are not focused on this principle. Rather than continue to recount how we will in some small way be implementing many design principles in part, though, we will consider our case made and move on to the principles that we have chosen to focus on.

Priority Principles

Defense in Depth and Securing the Weakest Link partner up quite well, since we can assess the weakest links in a progressive chain and thus build up the depth of our defense. A very simplified example is the idea of essential data never being stored as plain information. Even if data is stolen, provided the encryption is not woefully out of date, we have not lost control of all of our data.

Even more powerfully, of course, is us never having certain critical data in the first place. A by-now very widespread example of this is password recovery going out of style. The reason for this is that many services now simply do not store your password anywhere, so they cannot recover it for you. They will instead allow you to generate a new one, which of course opens another can of security worms, but it does at least lessen/eliminate the risk of catastrophic data theft events (most recently, as of this report being written, Yahoo! and Sony were victims of this).

The Failing Securely principle fits into this approach beautifully, since it supports the modular approach we have chosen to focus on. Restricting access as the default case forces us to evaluate what each functionality of the application really needs, and it helps prevent a situation where a breach in one section of the application is necessarily a breach to all of it.

If the base case is to deny access, this not only eliminates a whole host of often difficult to detect exploits that would otherwise fall back on granting access, but also helps in the general development and debugging of code, as we will immediately be able to address a code problem if it fails to access data it needs.

Implementation of these 3 design principles is something we see as a progressive process. We have a base case - lack of access - and this should take care of a lot of basic flaws. From this basis, we can focus specifically on our weakest links, our highest vulnerabilities, and address them in multiple successive layers, from the front end and all the way to the database, and back again.

Because restriction is the baseline, we do not have to worry about a single breach granting undue access. Even if someone, just as an example, was to gain access to a user's specific questions, this does not have to mean that they also have access to the room in which these would be displayed. In this case, the functions are, for the regular user, tied together visually, but technically separate.

A function that submits specific user questions does not need to know, or have access to, a user's specific room, or any room at all. By de-escalating access, decoupling functionality from broad and vulnerable functionality and thus implementing multiple points of potential privilege failure, and working with specific weak points (such as data submission to the database or a file, in this case), we implement each of our focused principles, and create a robust security solution that allows for inevitable failures without totally breaking down in the process.

In the above case, even if user-specific questions should somehow suffer a breakdown, this does not mean that all questions are down for the count. Games can continue. We failed securely, and thus continue to operate, even if one element or module of the application suffered a breach.

Design

Architectural Considerations

To a certain degree, system architecture was decided for us. We knew that we were going to work with a linux server, and due to guidance being focused on this and XAMPP for setup in an unfamiliar environment, these were chosen as the base building blocks. We made, essentially, a crude MVC-based system.

The Model tier was our MySQL database in XAMPP, the Controller tier was made with PHP, and the View tier was made with HTML, CSS (SASS) and Javascript.

This setup was arrived at due in part to familiarity with this specific stack setup, and from the pre-determined setup provided to us by KEA. The MVC pattern contributes, as well, to the modularity of the application, allowing us, if we so choose, to exchange one tier for another. The Controller tier could be replaced with another type of architecture if we so desired, as could the Model, or the View.

For external frameworks, we primarily relied on frontend technology. JQuery, Bootstrap, Material UI, Google's reCaptcha and the like. All of them are large and very widely-used frameworks, and we felt, as such, that they were relatively secure from severe intrusion or lack of timely updates relating to security.

Security Requirements

The specific requirements are listed below:

- SQL Injection
- XSS
- CSRF
- Client side manipulation

These are, of course, not the only security issues when it comes to web technology, but they were a starting point. One can, to some degree, begin pulling at any given security thread and never really stop going down the rabbit hole. This is where a cost/benefit analysis, or a risk analysis, comes in handy.

For example, setting headers so that our page cannot be displayed in iframes prevents a method of circumventing CSRF protection via tokens, but these headers may be ignored in older browsers. Is it worth it to go down this path? Is there an easy alternative? And so on.

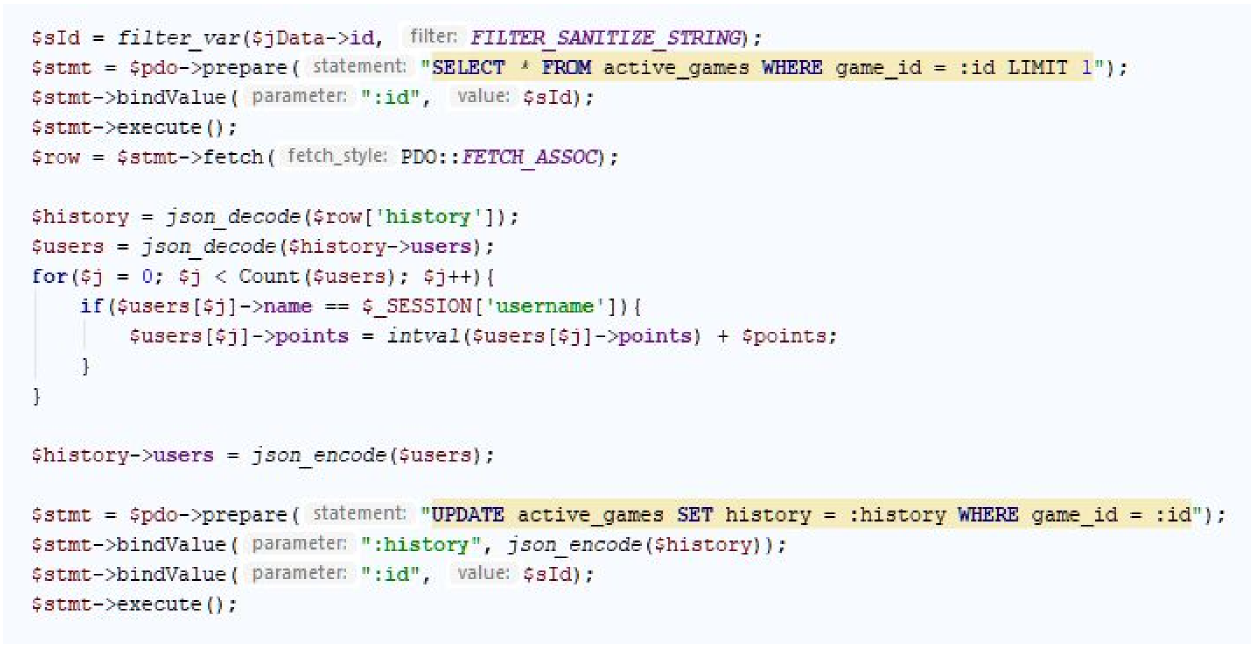
As our experience in this field is limited, it is hard for us to make this call, nor are we even, most likely, aware of all possible attack vectors that we could, with relative ease, protect against. Of course, given that one of our design principles is Securing the Weakest Link, our development methodology supports this in some fashion. As we become aware of more vulnerabilities, they make their way to the top of our priority queue.

Implementation

SQL injection

By using prepared statements, we effectively prevent 1st order SQL injection, and 2nd order SQL injection can be handled simply by resolving never to really trust data that has been, in part or entirely, in the hands of a user. As such, even if we are pulling data from the database and using it to access another part of the database, as long as we continue to use prepared statements, we continue to secure ourselves against SQL injection.

We use prepared statements many places in our code, one example being the below:



```
$sId = filter_var($jData->id,  filter: FILTER_SANITIZE_STRING);
$stmt = $pdo->prepare( statement: "SELECT * FROM active_games WHERE game_id = :id LIMIT 1");
$stmt->bindValue( parameter: ":id",  value: $sId);
$stmt->execute();
$row = $stmt->fetch( fetch_style: PDO::FETCH_ASSOC);

$history = json_decode($row['history']);
$users = json_decode($history->users);
for($j = 0; $j < Count($users); $j++){
    if($users[$j]->name == $_SESSION['username']){
        $users[$j]->points = intval($users[$j]->points) + $points;
    }
}

$history->users = json_encode($users);

$stmt = $pdo->prepare( statement: "UPDATE active_games SET history = :history WHERE game_id = :id");
$stmt->bindValue( parameter: ":history", json_encode($history));
$stmt->bindValue( parameter: ":id",  value: $sId);
$stmt->execute();
```

$sId contains data received from the front end, which is sanitized to remove HTML, XML and PHP tags. We could optionally have set flags to strip or encode ASCII characters with values outside the 32 - 127 range. In the second line, we prepare the database with an SQL statement, and then in the third line, bind the value of $sId to :id, which escapes it and invalidates any SQL that anyone might have tried to use. On the fourth line, we execute the prepared statement with the value put in, and then fetch the resulting row on the fifth line.

We could have used bindParam instead of bindValue if we anticipated having to potentially manipulate the value between it being bound and the prepared statement's execution, but this was not (at any point in our code) necessary.

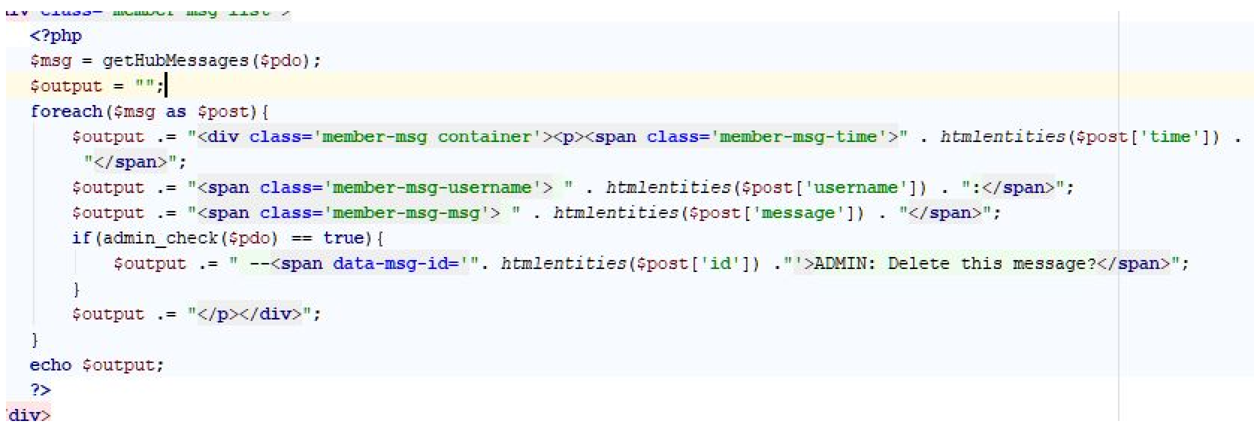
In the middle section, we manipulate some data, and then in the final 5 lines, to prevent potential 2nd order SQL, we bind not only the external id, but the internal $history.

Cross Site Scripting - XSS

Protecting against cross-site scripting fully comes down to a lot of small, spread out measures. Sanitizing, for example, will not necessarily eliminate cleverly nested script tags unless that sanitization continues to iterate over the code. Likewise, escaping output comes down to escaping output every time it is shown. No matter if a single sanitization/escaping function is made, or something built in - like htmlentities() - is relied on, it ultimately comes down to the developer to use it.

In this instance, a case could have been made that we could have made generic functions to handle certain SQL calls and always escaped output. Any time we have to build something custom, however, this once again falls apart. We save stringified JSON in the database, and this requires at times multiple decodes into objects, something which a generic set of CRUD SQL functions could not handle. Again, it falls to the individual developer to properly implement protection against this, something which we, more than once during development, found that we were not perfect at. And that was while specifically working with security in mind.

An example of output that escapes frontend tags can be seen below:

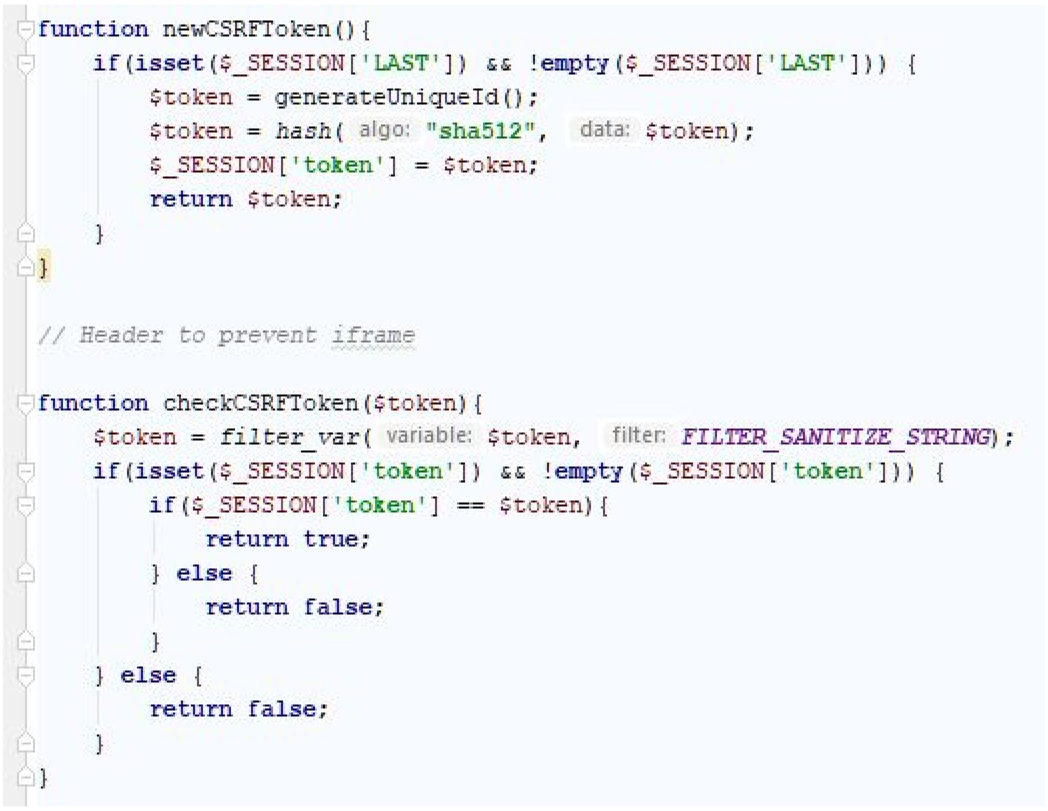


```
  <?php
  $msg = getHubMessages($pdo);
  $output = "";
  foreach($msg as $post){
      $output .= "<div class='member-msg container'><p><span class='member-msg-time'>" . htmlentities($post['time']) .
       "</span>";
      $output .= "<span class='member-msg-username'> " . htmlentities($post['username']) . ":</span>";
      $output .= "<span class='member-msg-msg'> " . htmlentities($post['message']) . "</span>";
      if(admin_check($pdo) == true){
          $output .= " --<span data-msg-id='". htmlentities($post['id']) ."'>ADMIN: Delete this message?</span>";
      }
      $output .= "</p></div>";
  }
  echo $output;
  ?>
div>
```

Cross Site Request Forgery - CSRF

The idea behind CSRF is that another website can exploit users that have a session running on the site they want to exploit. This is done by making the user's browser make calls to server endpoints where the session is running. This can be mitigated by including a token with each server request, such that attempting to use the server endpoint from a page that doesn't generate a valid token and sets it in the user's session for comparison's sake simply will not work. Token comparison fails.

Instead of the above, a clever hacker might open a hidden iframe on their page, and fire a javascript event to have the user's browser click on a submission button. There will now be a valid token included with the request. This, in turn, is mitigated by including headers in the page that disallow being loaded in iframes, or only allow it if the server source of the iframe is the server of the website itself.



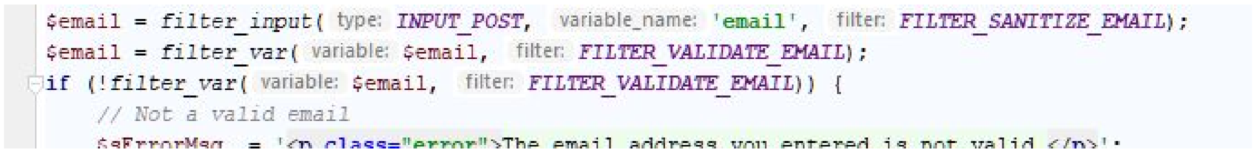
```
function newCSRFToken(){
    if(isset($_SESSION['LAST']) && !empty($_SESSION['LAST'])) {
        $token = generateUniqueId();
        $token = hash( algo: "sha512",  data: $token);
        $_SESSION['token'] = $token;
        return $token;
    }
}

// Header to prevent iframe

function checkCSRFToken($token){
    $token = filter_var( variable: $token,  filter: FILTER_SANITIZE_STRING);
    if(isset($_SESSION['token']) && !empty($_SESSION['token'])) {
        if($_SESSION['token'] == $token){
            return true;
        } else {
            return false;
        }
    } else {
        return false;
    }
}
```

Above, we have the two functions running our CSRF token generation and validation. In the first function, a token is generated by hashing a unique id (which is not really unique, it is 2 uniqid stapled together) and setting this as the current token in the active session, and also returning it. This way, when the function is called on a page, the session's token is set, and the same token can be echoed into a hidden input field of a form.

Client side manipulation

The ability for someone to influence the server by changing undesired variables front end. This can generally be protected against by only exposing what needs to be exposed, and by obfuscating slightly damning data by hashing it. This is something we protect against with server-side validation in most cases. For example, you cannot log in as someone else by changing some secret value (because there is none) in the login form. You cannot circumvent the CSRF protection by changing the token. All you will do is stop yourself from doing what you wanted to do.

An example of validating server-side can be seen here, for the registration process:



```
$email = filter_input( type: INPUT_POST,  variable_name: 'email',  filter: FILTER_SANITIZE_EMAIL);
$email = filter_var( variable: $email,  filter: FILTER_VALIDATE_EMAIL);
if (!filter_var( variable: $email,  filter: FILTER_VALIDATE_EMAIL)) {
    // Not a valid email
    $sErrorMsg  = '<p class="error">The email address you entered is not valid.</p>';
```

The email is first sanitized, and then validated, and in the part of the iff statement we see, an error is generated if the gathered email is not valid.

Test/Performance/Attacks

We tested SQL injection against our solution, and found no working attacks. We also tested CSRF attacks, and found ourselves not to be vulnerable. The one vulnerability left is someone using an outdated browser that does not recognize the headers we have set to stop the page from being loaded in iframes. The solution to this problem would be to simply detect if the user's browser is out of date, and then not display the page if it is.

We tested for XSS vulnerabilities in many fields as well, but again, found no vulnerabilities. However, as detailed in the Implementation section of this report, defending against XSS attacks is an ongoing effort, as we certainly could have people attempt to attack us and fail, only for them to succeed later if we forget to properly escape output. This is why, even though it’s relatively easy to defend against, we still consider it a high threat. Sanitization (especially for nested tags) and escaping only has to fail once for an old XSS snippet to suddenly activate, and since it is simple text, it is hard to truly stop it from ever getting stored in the database.

Conclusion

conclude on whether you reached your goal in the PF (no new info should go here)

Web Security taught us about security threats that we were either not aware of, or only aware of in a broad, basic sense. Learning to defend against these attacks, though not necessarily complicated at the level we are at, was nevertheless an interesting experience. It was also eye-opening in that we learned what many websites do (or fail to do), and how relatively easy it really is to stop.

We succeeded in implementing all our must haves, almost all of the nice to haves, and even one or two of the could haves. It would be arrogant and a sign of hubris to claim that what we have produced is totally secure, though. In fact, we are going to continue to look into this simple solution to uncover security problems even beyond the hand-in date, if only to make future security challenges easier for us to understand and deal with.

Reflections

What we have attended is not a linux administration course, and because of this, we are unlikely to have caught the problems associated with our server setup. One glaring issue, for example, is that all of our server-side setup is done with the root user, which means that if a breach should happen and an attacker gain access, they would have root access to the server.

In general, this project suffered from being our first concerted effort at implementing security in a web application. Many things could, and likely should, be made with more generic functions that can be reused, rather than writing new code for every little thing we do. These are not new lessons for us, but the focus here was on learning and bugfixing new elements of code, rather than optimizing and perfecting the code implementation for use and reuse.

Appendix

Appendix 1: Security policy: Password Policy

Overview

The password policy was created and put into effect to ensure that the passwords used for the application have a minimum strength.

Purpose

The purpose of the policy is to guide all users of the system to creating viable and strong passwords.

Scope

The scope of the policy is application-wide. Good password policy does not change too much from the frontend to the backend, and the passwords outlined here also function as an absolute minimum for backend passwords.

Target Audience

All users of the application, whether internal (developers, designers, etc.) or users.

Policies

1.0: The password should only ever be specified by and known in plain text to the user to whom it is tied.

1.1: The password must be a minimum of 8 characters.

1.2: The password must consist of both lower case and upper case letters.

1.3: The password must contain at least one number.

1.4: The password must contain at least one special character.

Definitions

Lower case: Letters such as: a b c d e f g …

Upper case: Letters such as: A B C D E F G …

Number: 1 2 3 4 5 6 7 8 9 0

Special character: One of these, though others apply as well: ! # ¤ % & ? ^

Version

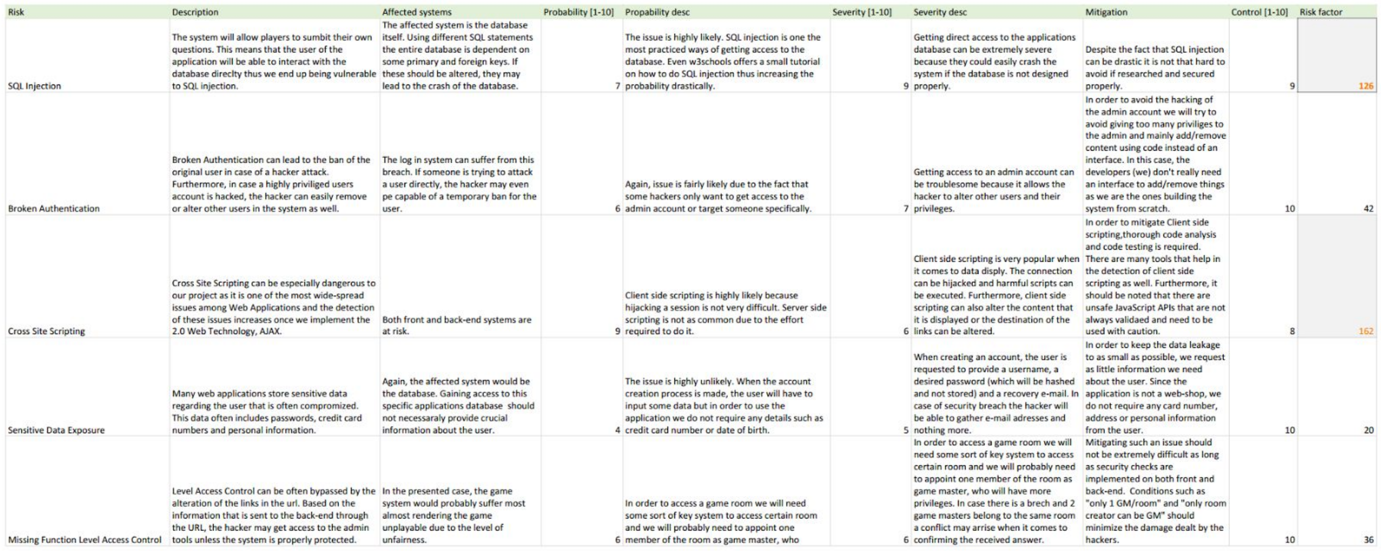
1.0: (24/05/2017): Policy created.

Appendix 2: Risk Sheet

The risk sheet can be viewed at a much better quality by accessing the link below:

https://goo.gl/utQGGl

Please visit the link in order to read the risk analysis. Please note that in order to maintain the aesthetics of the document we have only provided a screenshot of the analysis. For precise information please refer to the above posted link.



Risk	Description	Affected systems	Probability [1-10]	Propability desc	Severity [1-10]	Severity desc	Mitigation	Control [1-10]	Risk factor
SQL Injection	The system will allow players to sumbit their own questions. This means that the user of the application will be able to interact with the database direclty thus we end up being vulnerable to SQL injection.	The affected system is the database itself. Using different SQL statements the entire database is dependent on some primary and foreign keys. If these should be altered, they may lead to the crash of the database.	7	The issue is highly likely. SQL injection is one the most practiced ways of getting access to the database. Even w3schools offers a small tutorial on how to do SQL injection thus increasing the probability drastically.	9	Getting direct access to the applications database can be extremely severe because they could easily crash the system if the database is not designed properly.	Despite the fact that SQL injection can be drastic it is not that hard to avoid if researched and secured properly.	9	126
Broken Authentication	Broken Authentication can lead to the ban of the original user in case of a hacker attack. Furthermore, in case a highly priviliged users account is hacked, the hacker can easily remove or alter other users in the system as well.	The log in system can suffer from this breach. If someone is trying to attack a user directly, the hacker may even pe capable of a temporary ban for the user.	6	Again, issue is fairly likely due to the fact that some hackers only want to get access to the admin account or target someone specifically.	7	Getting access to an admin account can be troublesome because it allows the hacker to alter other users and their privileges.	In order to avoid the hacking of the admin account we will try to avoid giving too many priviliges to the admin and mainly add/remove content using code instead of an interface. In this case, the developers (we) don't really need an interface to add/remove things as we are the ones building the system from scratch.	10	42
Cross Site Scripting	Cross Site Scripting can be especially dangerous to our project as it is one of the most wide-spread issues among Web Applications and the detection of these issues increases once we implement the 2.0 Web Technology, AJAX.	Both front and back-end systems are at risk.	9	Client side scripting is highly likely because hijacking a session is not very difficult. Server side scripting is not as common due to the effort required to do it.	6	Client side scripting is very popular when it comes to data disply. The connection can be hijacked and harmful scripts can be executed. Furthermore, client side scripting can also alter the content that it is displayed or the destination of the links can be altered.	In order to mitigate Client side scripting,thorough code analysis and code testing is required. There are many tools that help in the detection of client side scripting as well. Furthermore, it should be noted that there are unsafe JavaScript APIs that are not always validaed and need to be used with caution.	8	162
Sensitive Data Exposure	Many web applications store sensitive data regarding the user that is often compromized. This data often includes passwords, credit card numbers and personal information.	Again, the affected system would be the database. Gaining access to this specific applications database should not necessaraly provide crucial information about the user.	4	The issue is highly unlikely. When the account creation process is made, the user will have to input some data but in order to use the application we do not require any details such as credit card number or date of birth.	5	When creating an account, the user is requested to provide a username, a desired password (which will be hashed and not stored) and a recovery e-mail. In case of security breach the hacker will be able to gather e-mail adresses and nothing more.	In order to keep the data leakage to as small as possible, we request as little information we need about the user. Since the application is not a web-shop, we do not require any card number, address or personal information from the user.	10	20
Missing Function Level Access Control	Level Access Control can be often bypassed by the alteration of the links in the url. Based on the information that is sent to the back-end through the URL, the hacker may get access to the admin tools unless the system is properly protected.	In the presented case, the game system would probably suffer most almost rendering the game unplayable due to the level of unfairness.	6	In order to access a game room we will need some sort of key system to access certain room and we will probably need to appoint one member of the room as game master, who	6	In order to access a game room we will need some sort of key system to access certain room and we will probably need to appoint one member of the room as game master, who will have more privileges. In case there is a brech and 2 game masters belong to the same room a conflict may arrise when it comes to confirming the received answer.	Mitigating such an issue should not be extremely difficult as long as security checks are implemented on both front and back-end. Conditions such as "only 1 GM/room" and "only room creator can be GM" should minimize the damage dealt by the hackers.	10	36

Collaboration Agreement

A collaboration agreement is a tool for producing effective group processes. We will present our expectations to the group work and to each other. That way we can discuss our different wishes and decide how we will work as a group. The group contract should document all our decisions about our group work.

The group contract is relevant because it will help us ensure a good group process. A good group process determines a good individual learning process. Some examples of advantages of group work are:

- We grow and learn along with the other students
- We learn to co-operate and get insight into ours and others' strengths and weaknesses
- We get continuous sparring and discussions with other students
- We can learn from people from other backgrounds other than our own
- We will have a small group of people to relate to

When our group is not working some of the disadvantages are:

- Less learning
- Poorer results
- A lot of frustration

Group members' full names	**E-mail addresses**	**Phone numbers**
1.Ralf Patrik Blaga	ralf.patric@yahoo.com	50321332
2.Johannes Otto Skjærbæk	significantowlowl@gmail.com	71787502

Working Terms and Agreements

Topic	Explanation	Group's decision
Group's purpose	What assignment – in brief, is the group going to solve	We are planning on working together on the Web Security mandatory

		assignments and exam project. The main purpose is to build an interactive and easy to use party web application.
Group’s goals	What goals – except solving the assignment – is the group going to put focus on: learning goals, process/social goals etc.	The main group goal is to improve our skills in both defending and attacking websites and web applications. Our secondary goal is to be able to hack into as many projects as we can during the hacking period.
Group’s ambition	Are you going to be the best in class – or are you going for just reaching the essentials	We will try to find a balance between defending and attacking websites. We do not work to impress others, we work to impress ourselves and to improve as developers.
Individual group members’ goals/ambition	Have you got any individual agendas; ex. part(s) of your individual semester learning goal, that you want to achieve within this group work.	Ralf is more interested in the social engineering, data extraction and cross site scripting.
		John is more interested in all of the technical/coding aspects of web security, and less so the social engineering.
What skills does the group wish to develop?	Have you got any specific skills that you as a group want to focus on; programming tools, Sensitive Data Exposure, etc.	Both members of the group are interested in Cross Site Scripting and SQL Injection.
Group’s strategy	How will the group ensure that each member develops the desired skills; are you going to meet a lot, read a lot, share knowledge a lot, teach each other etc.	In order to make sure that all the group members manage to achieve their goals we will try to distribute the tasks in a way, so that we work towards our own goals as well.
How many hours of daily group work? How many hours of daily individual work?		The hours of group work will vary based on the mandatory assignments we receive and the school and work schedule the group members have.
What activities outside the studies will each member prioritize?	Have you got any “outside the school obligations” that you want to tell your group members about; i.e. children, work, other that is of importance to you and something you want to-/have	Ralf: Mondays: Potential tutoring at MMD, L16 Wednesdays: Tutoring at MMD, L16 Fridays: Potential tutoring at MMD L16 John: Tuesdays/Fridays: Hobby stuff 20 - 24

	to prioritize – perhaps even above the group work.	Wednesdays: Work/Tutoring from 9 - 16
When will the group meet for group work?	Make a schedule!	The group will mostly meet in the evenings and weekends. As we live 2 blocks away from each other, meeting in the evenings is not a problem. Tuesday: 18:00-20:00 Friday: 16:00-18:00 Saturday: 12:00-16:00

How will the group divide the task(s)?	Are you going to do everything sitting together, are you going to work one and one, are you going to solve things in pairs?	The group will divide the tasks based on area of interest and work capabilities. We would prefer to complete the tasks together but if the time constraint does not allow it, working separately is an option we are willing to take.
How are you going to take decisions in the group?	Are you going to vote and/or use other “democratic tools? Are you aiming for unity and willing discuss each and every issue until everybody agrees or are you going to appoint responsibility?	We are aiming for unity and we are willing to discuss the issues. In case of disagreement and inability of choice, we will ask for coaching from our teacher and reconsider.
What problems are likely to arise in the group process?	Have you any pre knowledge of potential “working together issues”. Try to have some explicit considerations regarding who you are as “team player types”	The biggest issue our group will face is lack of presence. We have had previous experience with this problem and it is something that arises in almost any project. We are trying to overcome this problem in every project.
How are you going to organize the group?	Are you going to build a group hierarchy with a leader and other appointed “organizational tasks” If so, how are you going to appoint the leader/ decide about the hierarchy? Or are you going to work in a very horizontal and partly “anarchistic” group structure	The group will not have a hierarchy nor a leader. All of the members have an equal status in the group. The group is formed out of 2 people thus resulting in a 50-50. This has proven, based on past experience, not to be a problem.
How will group evaluate the group process the	Will you evaluate every day using ex. SCRUM meetings or will you evaluate otherwise	The group will be evaluated in format similar to SCRUM but on a weekly basis instead of a daily basis as it is not

group’s ability to reach it’s goals?		necessary or possible due to time constraints. There will be no scrum master, yet there will be a Kanban chart/To do list on what we want to do and when.
How and when will you punish violation of the group norms?	How often are you going to violate the rules in order to get punished? What types of punishment will you use; baking a cake, split up the group, expelling a team member? If you are going to expel a team member, how will you do it?	In case of violation of the terms of agreement, the violating party will be required to pay breakfast and give a good reason for not showing up. On another note, in terms of chances, 3 strikes and the violating member will be thrown out from the group.

Detailed requirements

The project, once a database with the details to be found in the db_access.php file is set up, works out of the box when unzipped in your localhost. We have included a database dump called quizgame.sql, but importing it is actually not required.

The only specific non-solution element to be aware of is that a member’s rank must manually be set to 999 in the database if they are to access admin functions.