Project Report - SWE 681

Paurav Surendra
Nitin Kaveriappa Udiyanda Muthanna
2016/12/10

"Anything that can go wrong, will go wrong." -Edward Aloysius Murphy Jr.

1. Introduction

As both of us are very avid gamers we decided to implement Fanorona-Tsivy for our SWE 681 Project. Both of us had played Assassin's Creed which is a popular game from Ubisoft and as part of it's gameplay there are many mini games, one of which is Fanorona-Tsivy, wherein lies the source of our project topic.

Fanorona is a strategy board game for two players and is indigenous to Madagascar. Fanorona-Tsivy is the most popular amongst all the versions. Black and white pieces, twenty-two each, are arranged on all points but the center. The objective of the game is to capture all the opponents pieces. The game is a draw if neither player succeeds in this. Capturing is done by either approaching or withdrawing from opponent's pieces.

Project Name	-	Fanorana-Tsivy
Collaborators	-	Paurav Surendra Nitin Kaveriappa Udiyanda Muthanna
Class	-	SWE 681
Date	-	2016/12/10
Github Link	-	https://github.com/nitinkaveriappa/Project-Fan (Will be made public after the last date for project submission)
Website Link	-	https://nitinkaveriappa.pro/fanorana/ (Up and running and hopefully error free)

2. Design/Architecture

The project was designed to be as simple as it could be, so that it can successfully be implemented in the given time frame. The project has three layers to it: Client Side, Server Side and Database Management.

Database Management:

Technology used: MySQL DB was chosen to store and manage the data. It was chose for its simplicity, ease of use and free to use licence. The database contains the following tables:

Player master table which stores all the registered players with email ids and password
Player stats table which stores the win, loss and draw count for each player
Flag list which has flags for player logged in, in the lobby and playing a game
Game master table which stores the list of games created with timestamps and players
Game state table that stores the state of the game during a match.
Verify list table stores the list of players registered yet to verify.

Client Side:

Technologies Used: HTML5, CSS, JavaScript/JQuery (with Ajax)

The client can access the game from the browser. The pages displayed are written in HTML5 with CSS for styling. JQuery is used to perform AJAX calls to server to fetch data during a game. Client can create an account, login, view win/loss stats and play the game with other players online.

Server Side:

Technologies Used: Apache 2.4 Server, PHP 5.6

The website is hosted on an Apache web server. PHP was chosen as the language to code because of its object oriented capabilities, security features, ease of use for handling sessions and database connectivity.

3. Installation Instructions

To run the project on your localhost in a Linux OS

1. Install the latest Apache Web Server
2. Ensure the server has port 443 enabled for SSL/TLS
3. Configure the server with a (self)signed certificate and enable SSL/TLS
4. Install the PHP 5.6 (latest stable version)
5. Install the PDO and SSL modules for PHP
6. Install SSMTP and configure with a valid email address
7. Link the SSMTP location in PHP configuration file
8. Install MySQL and set up a database server
9. Create the database with the provided script
10. Place the source code in /var/www/fanorona
11. Modify the config file with values respective to the setup above

For a detailed explaination on how to setup a LAMP environment to run websites: https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-on-ubuntu-16-04

4. Operating Instructions

To play the game open browser and go to https://www.nitinkaveriappa.pro/fanorona or to run game on your localhost with above set up go to https://localhost/fanorona

1. You will have to first create an account. Click on Register Now and enter your details
2. Wait for the verification email and click on the link in the email
3. Login to the site with your credentials
4. For test purposes, you can do steps 1-3 in another browser for a different account
5. To start a new game click on new game
6. To join this game and click on join game
7. Once the game starts, the turn switches between both the players until a winner is declared.
8. The win/loss count of your games can be seen in the home page.

5. Game Rules

There exist variations of the rules; this is the variant we have implemented. You may notice that we have bent the rules ever so slightly when compared to the original rules so as to conform to our program logic.

The Fanorona-Tsivy board consists of lines and intersections, creating a grid with 5 rows and 9 columns subdivided diagonally to form part of the tetrakis square tiling of the plane. A line represents the path along which a stone can move during the game. There are weak and strong intersections. At a weak intersection it is only possible to move a stone horizontally and vertically, while on a strong intersection it is also possible to move a stone diagonally. A stone can only move from one intersection to an adjacent intersection.

- Players alternate turns, starting with Red and then Blue.
- There are two kinds of moves: non-capturing and capturing. A non-capturing move is called a paika move.
- A paika move consists of moving one stone along a line to an adjacent intersection.
- Capturing moves are obligatory and have to be played in preference to paika moves.
- Capturing implies removing one or more pieces of the opponent, in one of two ways:
  - Approach—moving the capturing stone to a point adjacent to an opponent's stone, which must be on the continuation of the capturing stone's movement line.
  - Withdrawal—the capturing stone moves from a point adjacent to the opponent's stone, away from the stone along the continuation of the line between them.
- When an opponent stone is captured, all opponent pieces in line beyond that stone (as long as there is no interruption by an empty point or an own stone) are captured as well.
- An approach capture and a withdrawal capture cannot be made at the same time – if an approach capture is given preference to the withdrawal capture.
- As in checkers, the capturing piece is allowed to continue making successive captures, with these restrictions:
  - The piece is not allowed to arrive at the same position twice.
  - It is not permitted to move twice consecutively in the same direction (first to make a withdrawal capture, and then to make an approach capture) as part of a capturing sequence.
- However, unlike in checkers, continuing the capturing sequence is optional.
- The game ends when one player captures all stones of the opponent. If neither player can achieve this—for instance if the game reaches a state where neither player can attack the other without overly weakening their own position—then the game is a draw.

6. Why we believe it's secure?

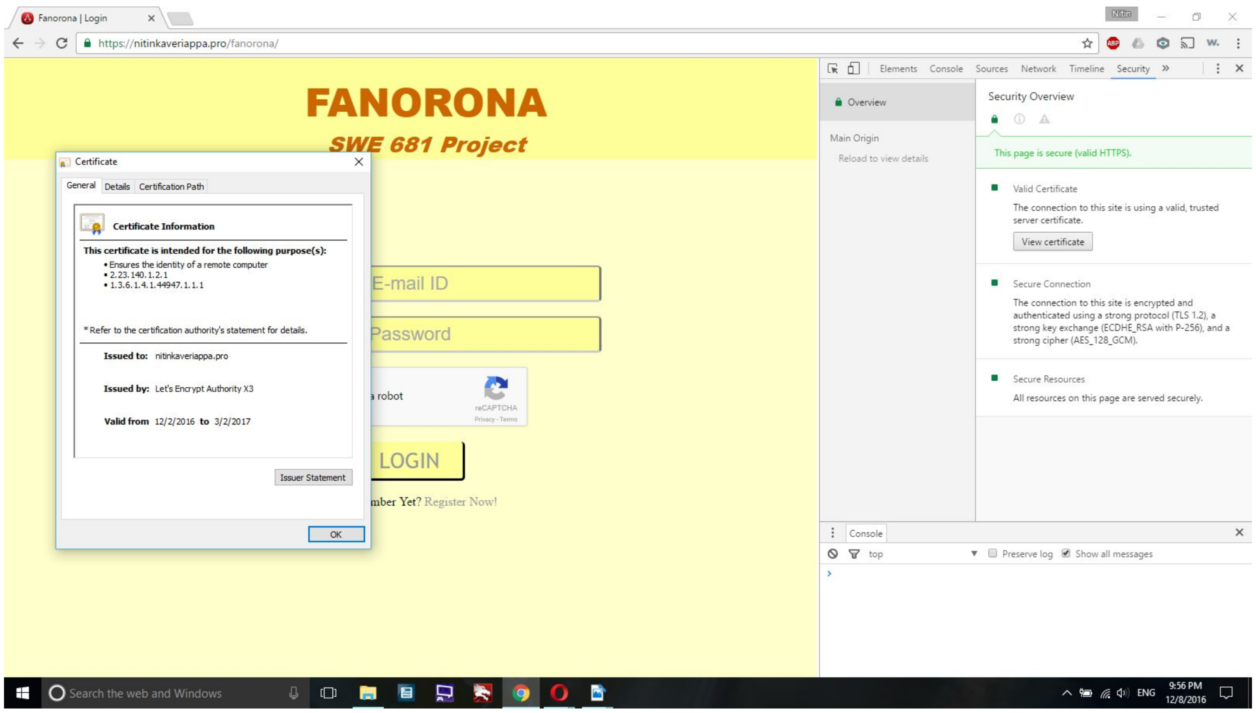
The main goal of this project has been security. Everything that is done, from database schema to language used to content displayed, has security as its core. The following are the features adapted or provided to ensure the game can be played in a secure environment.

Input validation: All the data that is directly taken from the user is checked using input validation. The data is filtered using regular expressions and whitelists. These set of accepted inputs prevent the attackers from sending harmful data to server, that can cause Overflows and XSS attacks. This is then further sanitized by escaping special characters before storing the data into the database (prepared statements). This prevents any SQL Injection. The data that is retrieved from the database to be displayed is escaped for any content that might be harmful for the client.

```
//Checks if the user name is acceptable
if(isset($_POST['userName'])                                                    &&
preg_match('/^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{1,4}$/',$_POST['userNa
me'])  && strlen($_POST['userName']) < 50)
 {
  $userName = $_POST['userName'];
 }
//Escape special characters from data retrieved
$this->player_name = htmlspecialchars($result['player_name']);
```

TLS/SSL connection :The connection to the local server version of the game was secured using OpenSSL. OpenSSL is an open source project that provides a robust, commercial-grade, and full-featured toolset for the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols.

The connection to this online game site is encrypted and authenticated using a strong protocol (TLS 1.2), a strong key exchange (ECDHE_RSA with P-256), and a strong cipher (AES_128_GCM) issued by Let's Encrypt. Let's Encrypt is a free, automated, and open certificate authority (CA), run for the public's benefit. It is a service provided by the Internet Security Research Group (ISRG). The server has been configured such that all the connections to it are forced to be served securely.



Certificate

General Details Certification Path

Certificate Information

This certificate is intended for the following purpose(s):

- Ensures the identity of a remote computer
- 2.23.140.1.2.1
- 1.3.6.1.4.1.44947.1.1.1

* Refer to the certification authority's statement for details.

Issued to: nitinkaveriappa.pro

Issued by: Let's Encrypt Authority X3

Valid from 12/2/2016 **to** 3/2/2017

Security Overview

This page is secure (valid HTTPS).

Valid Certificate

The connection to this site is using a valid, trusted server certificate.

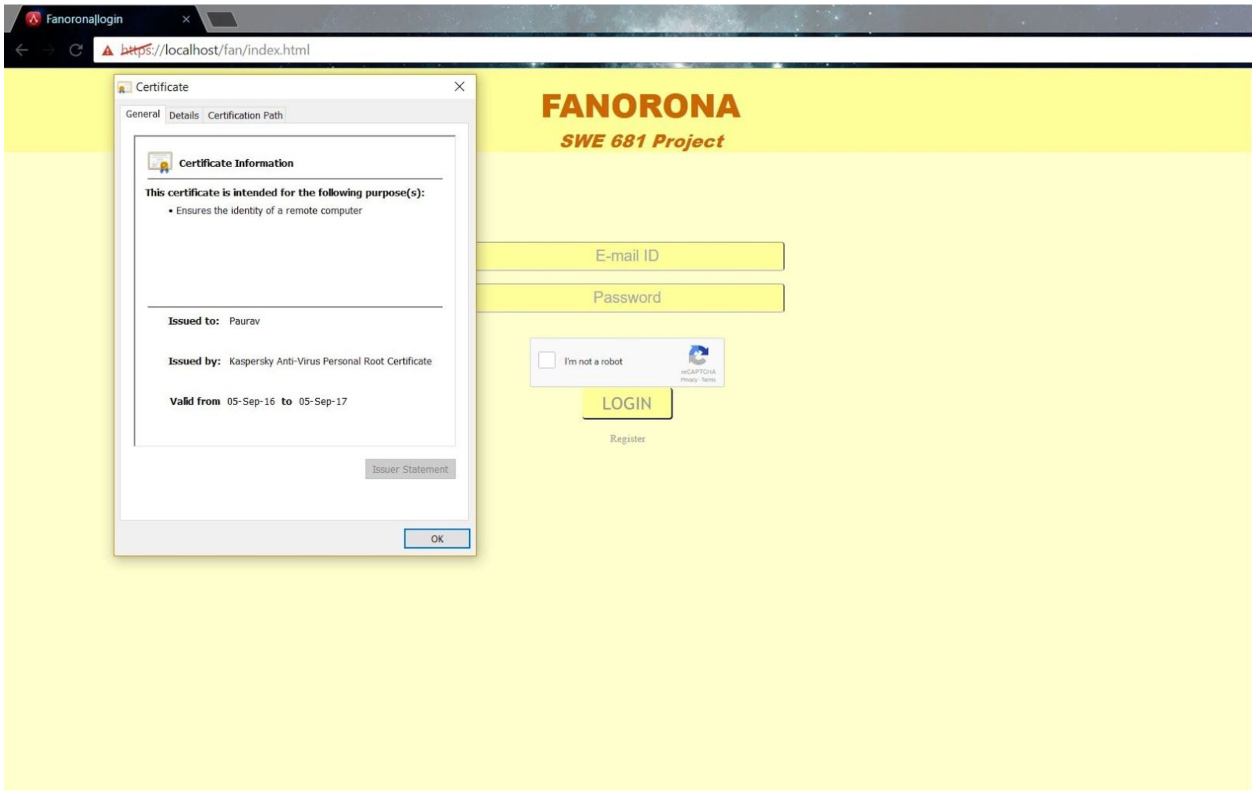
Secure Connection

The connection to this site is encrypted and authenticated using a strong protocol (TLS 1.2), a strong key exchange (ECDHE_RSA with P-256), and a strong cipher (AES_128_GCM).

Secure Resources

All resources on this page are served securely.

Certificate from Let's Encrypt on the website



Certificate

General Details Certification Path

Certificate Information

This certificate is intended for the following purpose(s):

- Ensures the identity of a remote computer

Issued to: Paurav

Issued by: Kaspersky Anti-Virus Personal Root Certificate

Valid from 05-Sep-16 **to** 05-Sep-17

Self Signed Certificate on localhost

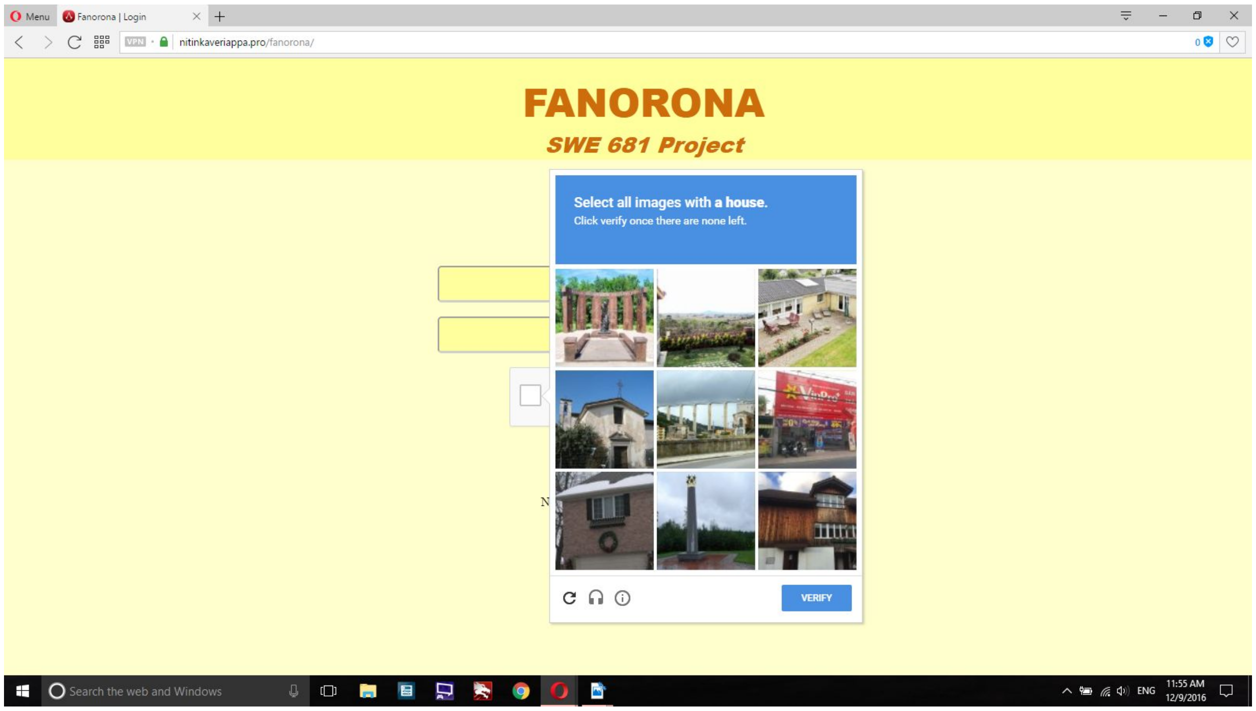
PDO: PHP provides an extension to interact with databases called PHP Data Objects (PDO). PDO provides a level of abstraction for data access between the database and code. Therefore the syntax and functions used for all database management systems are the same with only a driver specific to that DBMS.

PDO provides a good way to overcome SQL Injection attacks with Prepared Statements. Prepared statements are queries with placeholders for conditions. These conditions are usually determined by the user and thus will have to be sanitized before entered in the query. The bind parameter function provided by PDO does exactly that, eliminating any SQL injection threats.

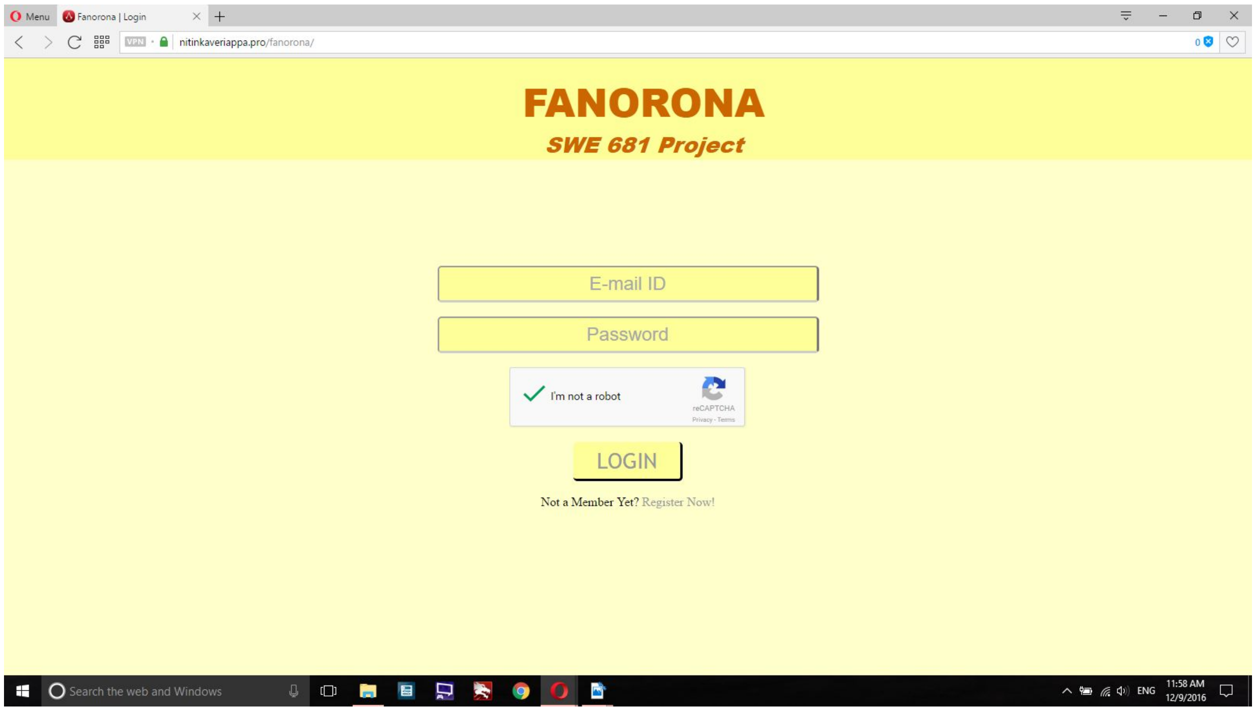
```
$playerEmail = $_REQUEST["mail"];
$verifyQuery  =  $connection->prepare("SELECT player_name  FROM  pl_mst  WHERE
player_email=:mailid;");
$verifyQuery->bindParam(':mailid',$playerEmail);
$verifyQuery->execute();
```

Recaptcha: To prevent against DEnial of Service attacks, we have implements Google's Recaptcha on the Login and Register pages. Google's Recaptcha is a simple but effective tool in verifying human interaction. The user has to click a simple checkbox and if the number of attempts to login increases, the user has to select images from a list of images as described by the tool. This will prevent any bots from overloading the server with large number of requests.

```
//get captcha secret key from config file
 $config = parse_ini_file('config.php');
 $secret = $config['secret'];

//Validates the captcha value from google server
$response=file_get_contents("https://www.google.com/recaptcha/api/siteverify?se
cret=$secret&response=".$captcha."&remoteip=".$_SERVER['REMOTE_ADDR']);
 $data = json_decode($response);

//IF its a bot it redirects to back to login page
 if($data->success==false)
 {
   header("Location:index.html?type=err");
 }
```


FANORONA
SWE 681 Project
Select all images with a house.
Click verify once there are none left.
VERIFY

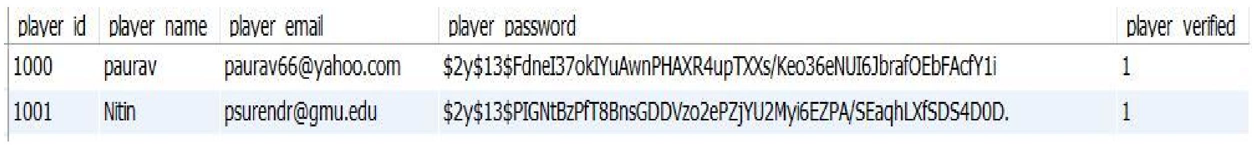
Requests user to select appropriate images



Checkbox ticked if selected images are correct

Password storage: Each player must create an account to play the game. So each player has a unique user name (email id) and password to login to their account. Attackers can login to other players account if they get hold of the player's' password. To prevent this, the passwords are stored in the database as hashes. To increase the level of security, the passwords are hashed with salt which prevents rainbow tables and brute force attacks to identify the password. PHP provides built-in function to hash data with salt called bcrypt. It uses blowfish algorithm with a random salt generator to create salted hashes 60 characters long irrespective of the data length. The function allows programmers to set the cost i.e. number of rounds the hashing should occur.

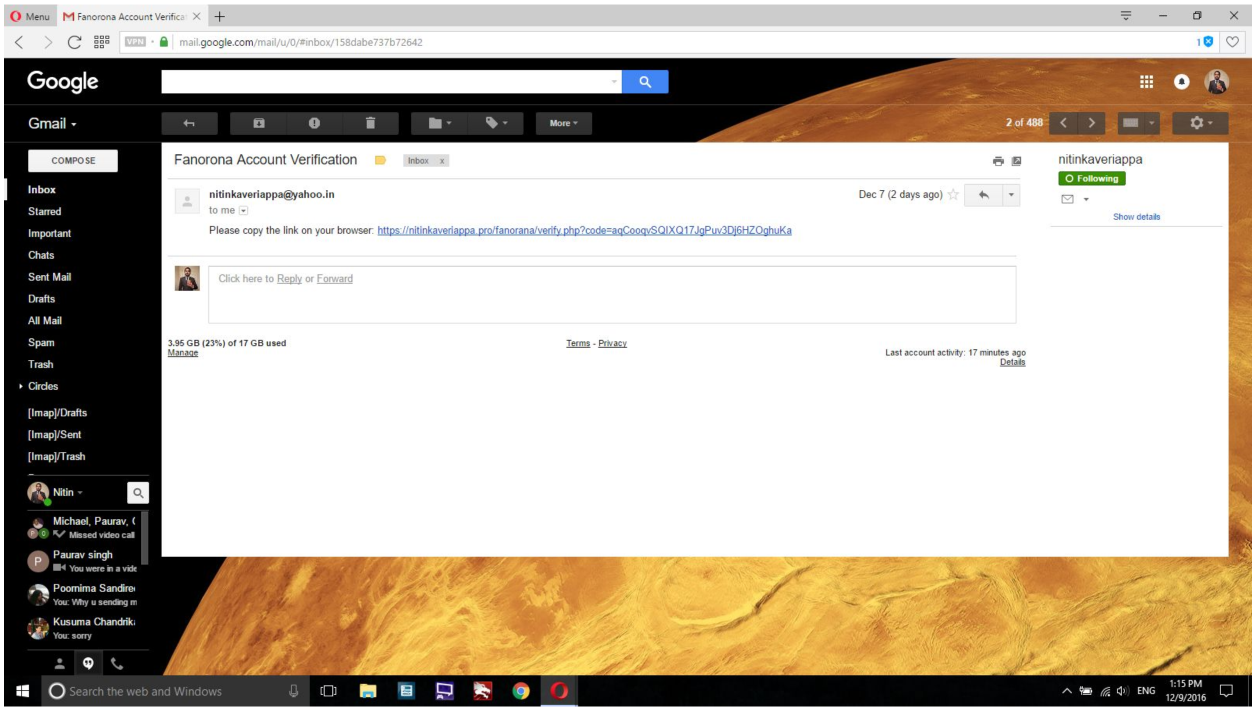
```
$options = array("cost"=>13);
$playerPassword = password_hash($playerPassword,PASSWORD_BCRYPT, $options);
```



player id	player name	player email	player password	player verified
1000	paurav	paurav66@yahoo.com	$2y$13$FdneI37okIYuAwnPHAXR4upTXXs/Keo36eNUI6JbrafOEbFAcfY1i	1
1001	Nitin	psurendr@gmu.edu	$2y$13$PIGNtBzPfT8BnsGDDVzo2ePZjYU2Myi6EZPA/SEaqhLXfSDS4D0D.	1

Email verification: An email verification process has been implemented to prevent individuals from creating dummy accounts and overloading the server with game requests. The advantage of having verified emails also helps in identifying attackers based on the account from which attacks originated. The verification process is simple, once the user registers he/she receives an email from the server with an account activation link. This link contains a code generated specific to the details of that account which is stored in the database. Once the player is verified, player can login.

```
//Generates a code with email id
 $encryptemail = password_hash($playerEmail,PASSWORD_BCRYPT, $options);
 $code = substr($encryptemail,29,31);

//Add verification code with player id into fanodb
$addVerifyQuery = $connection->prepare("INSERT INTO vr_ls
(player_id,verify_code) VALUES (:playerId,:code);");
$addVerifyQuery->bindParam(':playerId',$playerId);
$addVerifyQuery->bindParam(':code',$code);
$addVerifyQuery->execute();

//Sends the verification url to member
$this->sendVerification($playerEmail,$code);

//On receiving the link from user
$code = $_REQUEST['code'];

//Get player ID for the code from database
$getPlayerQuery = $connection->prepare("SELECT player_id FROM vr_ls WHERE
verify_code=:code;");
$getPlayerQuery->bindParam(':code',$code);
$getPlayerQuery->execute();

//IF code is valid Update the verify flag in player master
if($getPlayerQuery->rowCount() == 1)
{
 $result = $getPlayerQuery->fetch(PDO::FETCH_ASSOC);
 $id = $result['player_id'];
 $verifyPlayerQuery = $connection->query("UPDATE pl_mst SET player_verified=1
WHERE player_id=$id;");

//Remove the tuple with verify code for the player
$deleteVerifyQuery = $connection->query("DELETE FROM vr_ls WHERE
player_id=$id;");
 header("LOCATION:index.html");
}
else
{
echo "VERIFICATION FAILED";
}

//Server checks if the player is verified during login
if(password_verify($userPassword, $cryptedPassword) && $verifiedFlag == 1 &&
$loginFlag == 0)
   { ...
```


mail.google.com/mail/u/0/#inbox/158dabe737b72642

Fanorona Account Verification

nitinkaveriappa@yahoo.in
to me
Dec 7 (2 days ago)

Please copy the link on your browser: https://nitinkaveriappa.pro/fanorana/verify.php?code=aqCooqvSQIXQ17JgPuv3Dj6HZOghuKa

Click here to Reply or Forward

3.95 GB (23%) of 17 GB used
Manage

Terms - Privacy

Last account activity: 17 minutes ago
Details

Account Verification Email

Single Login: The server keeps track of players that have logged in and will not allow the same player to login from multiple systems. This prevents the players from playing multiple games at the same time and overload the database with mixed information. Single login also prevents players from playing games against themselves. Most importantly if an attacker gets hold of the player credentials, he will not be able to login if the player is already logged on. In the reverse scenario, if a player is unable to login, it indicates the account is being used from a different location and can be tracked down.

```
//Server checks if the player is not already logged in
if(password_verify($userPassword, $cryptedPassword) && $verifiedFlag == 1 &&
$loginFlag == 0)
   { …
```

Session handling: A session is started when a player logs into their account and it exists till they log out. To prevent attacks with session id, the server has a session limit of 5 minutes for a given session id. And the session id for a session gets regenerated every 2 minutes if the player is active. Therefore an attacker cannot use a session id for more than 2 minutes to attack the server. The session is used to hold player data and it is destroyed once the player logs out.

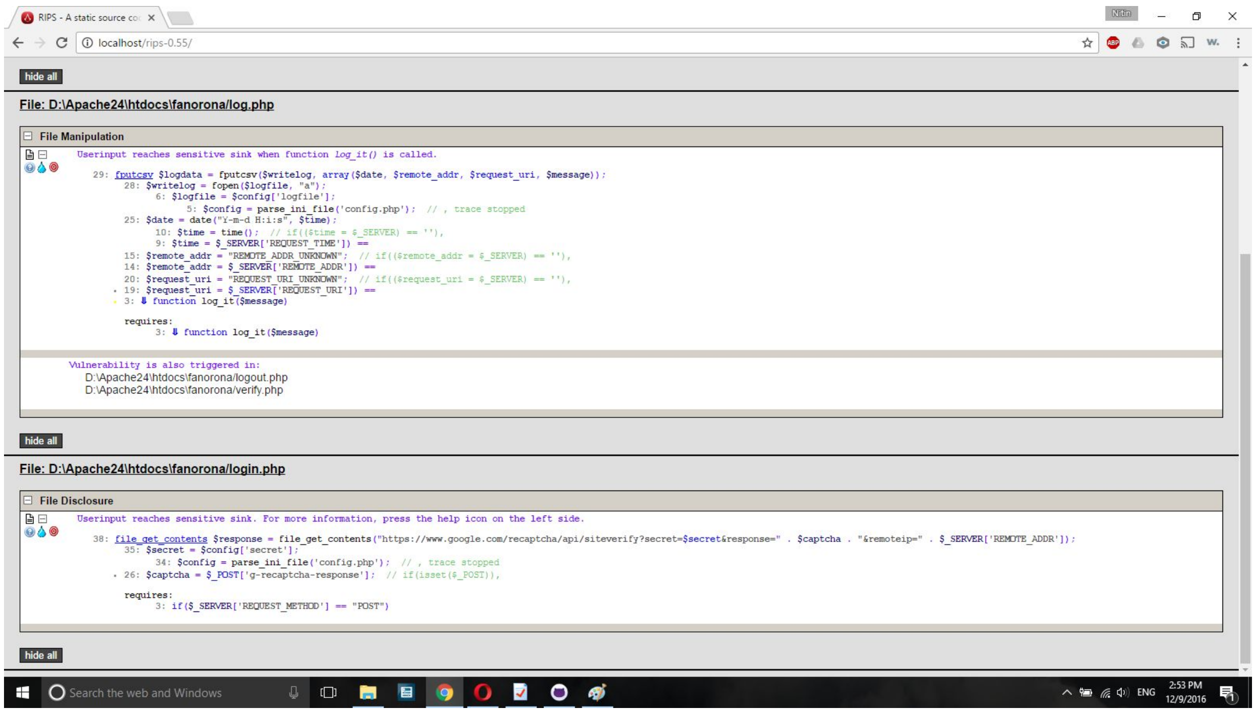
```
if ($idle > 300)
{
 header('Location:logout.php');
}
else if($idle > 120)
{
      session_regenerate_id(true);
      $_SESSION['created'] = time();
}
//Session is destroyed and deleted during logout
 // Unset all of the session variables
 $_SESSION = array();
 // Delete the session cookie
 if (ini_get("session.use_cookies"))
 {
       $params = session_get_cookie_params();
         setcookie(session_name(),  '',  time()  -  42000,  $params["path"],
$params["domain"], $params["secure"], $params["httponly"]);
 }
 //Destroy session
    if(session_destroy()) {
      header("Location:index.html");
   }
```

Static Code Analysis

Static code analysis is the analysis of computer software that is performed without actually executing programs. There are not many good free static analysis tools available for free for PHP code analysis. But nevertheless we have implemented RIPS and Visual Code Grepper static analysis tools in order to find and fix vulnerabilities.

RIPS is a tool written in PHP to find vulnerabilities in PHP applications using static code analysis. By tokenizing and parsing all source code files RIPS is able to transform PHP source code into a program model and to detect sensitive sinks (potentially vulnerable functions) that can be tainted by user input (influenced by a malicious user) during the program flow.

There are 3 plausible vulnerabilities suggested by RIPS: 2 file disclosures alerts and 1 file manipulation. The file disclosure alerts are pointed towards the response link for Google's recaptcha result. This a publicly available link and can be obtained in matter of seconds. But to obtain the result from this link, a site key and secret key are required both of which is obtained from a config file. Therefore having this link exposed does not pose a threat to the security of the site. The file manipulation is done by the server to add logs to the log file. This file is inaccessible outside the server and the location of the file never disclosed to the public. Also, the logs entered are predefined set of strings, therefore log forging is prevented.



File: D:\Apache24\htdocs\fanorona/log.php

File Manipulation

```
Userinput reaches sensitive sink when function log_it() is called.

  29: fputcsv $logdata = fputcsv($writelog, array($date, $remote_addr, $request_uri, $message));
      28: $writelog = fopen($logfile, "a");
          6: $logfile = $config['logfile'];
             5: $config = parse_ini_file('config.php');  // , trace stopped
      25: $date = date("Y-m-d H:i:s", $time);
          10: $time = time();  // if(($time = $_SERVER) == ''),
          9: $time = $_SERVER['REQUEST_TIME']) ==
      15: $remote_addr = "REMOTE_ADDR_UNKNOWN";  // if(($remote_addr = $_SERVER) == ''),
      14: $remote_addr = $_SERVER['REMOTE_ADDR']) ==
      20: $request_uri = "REQUEST_URI_UNKNOWN";  // if(($request_uri = $_SERVER) == ''),
    • 19: $request_uri = $_SERVER['REQUEST_URI']) ==
    • 3: ⇓ function log_it($message)

      requires:
          3: ⇓ function log_it($message)

Vulnerability is also triggered in:
   D:\Apache24\htdocs\fanorona/logout.php
   D:\Apache24\htdocs\fanorona/verify.php
```

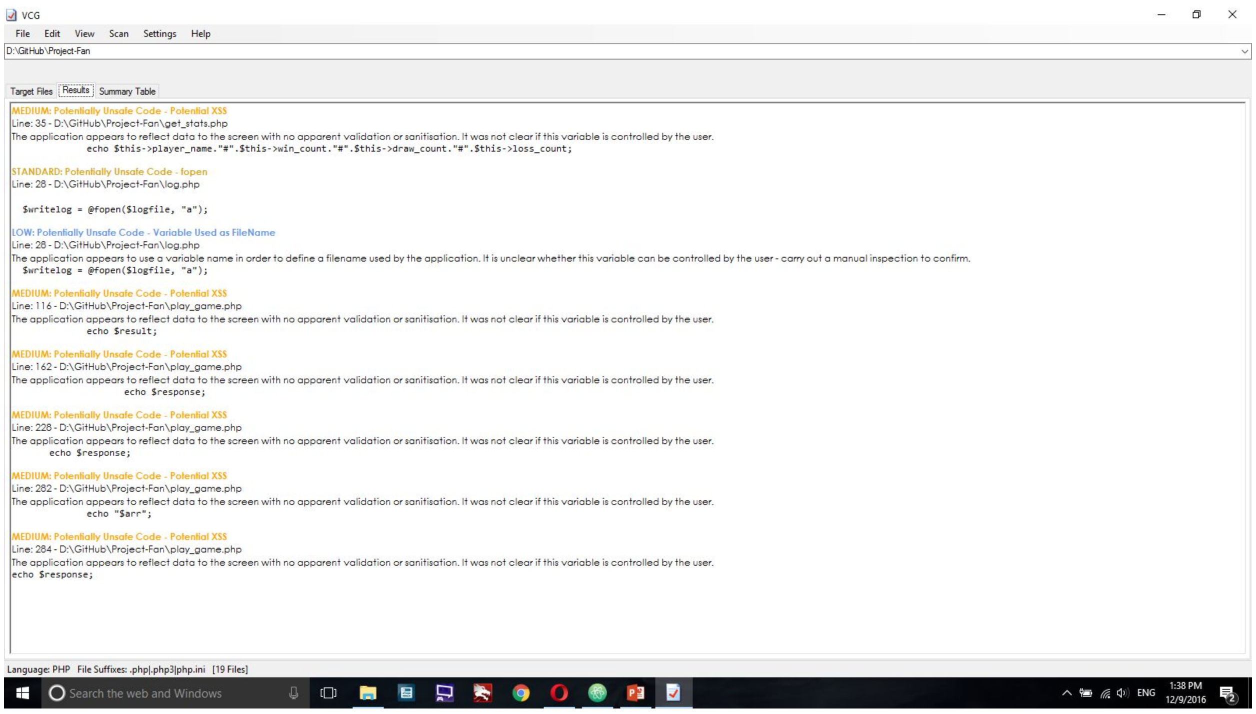
File: D:\Apache24\htdocs\fanorona/login.php

File Disclosure

```
Userinput reaches sensitive sink. For more information, press the help icon on the left side.

  38: file_get_contents $response = file_get_contents("https://www.google.com/recaptcha/api/siteverify?secret=$secret&response=" . $captcha . "&remoteip=" . $_SERVER['REMOTE_ADDR']);
      35: $secret = $config['secret'];
          34: $config = parse_ini_file('config.php');  // , trace stopped
    • 26: $captcha = $_POST['g-recaptcha-response'];  // if(isset($_POST)),

      requires:
          3: if($_SERVER['REQUEST_METHOD'] == "POST")
```

RIPS Results

Visual Code Grepper is an automated code security review tool which is intended to drastically speed up the code review process by identifying bad/insecure code. There are multiple vulnerabilities shown as Potential XSS. But these potential issues have been prevented by various measures. None of the responses sent to the client is being displayed. Also these response are used as conditions to perform specific actions based on their values. The values for these responses obtained from database are sanitized before they are sent to the client. Therefore we have eliminated any possible XSS attack at the client. The low priority unsafe code is again for writing the log file which is explained above.



```
MEDIUM: Potentially Unsafe Code - Potential XSS
Line: 35 - D:\GitHub\Project-Fan\get_stats.php
The application appears to reflect data to the screen with no apparent validation or sanitisation. It was not clear if this variable is controlled by the user.
            echo $this->player_name."#".$this->win_count."#".$this->draw_count."#".$this->loss_count;

STANDARD: Potentially Unsafe Code - fopen
Line: 28 - D:\GitHub\Project-Fan\log.php

  $writelog = @fopen($logfile, "a");

LOW: Potentially Unsafe Code - Variable Used as FileName
Line: 28 - D:\GitHub\Project-Fan\log.php
The application appears to use a variable name in order to define a filename used by the application. It is unclear whether this variable can be controlled by the user - carry out a manual inspection to confirm.
  $writelog = @fopen($logfile, "a");

MEDIUM: Potentially Unsafe Code - Potential XSS
Line: 116 - D:\GitHub\Project-Fan\play_game.php
The application appears to reflect data to the screen with no apparent validation or sanitisation. It was not clear if this variable is controlled by the user.
            echo $result;

MEDIUM: Potentially Unsafe Code - Potential XSS
Line: 162 - D:\GitHub\Project-Fan\play_game.php
The application appears to reflect data to the screen with no apparent validation or sanitisation. It was not clear if this variable is controlled by the user.
                  echo $response;

MEDIUM: Potentially Unsafe Code - Potential XSS
Line: 228 - D:\GitHub\Project-Fan\play_game.php
The application appears to reflect data to the screen with no apparent validation or sanitisation. It was not clear if this variable is controlled by the user.
      echo $response;

MEDIUM: Potentially Unsafe Code - Potential XSS
Line: 282 - D:\GitHub\Project-Fan\play_game.php
The application appears to reflect data to the screen with no apparent validation or sanitisation. It was not clear if this variable is controlled by the user.
            echo "$arr";

MEDIUM: Potentially Unsafe Code - Potential XSS
Line: 284 - D:\GitHub\Project-Fan\play_game.php
The application appears to reflect data to the screen with no apparent validation or sanitisation. It was not clear if this variable is controlled by the user.
echo $response;
```

VCG
File Edit View Scan Settings Help
D:\GitHub\Project-Fan
Target Files
Results
Summary Table
Language: PHP File Suffixes: .php|.php3|php.ini [19 Files]
Search the web and Windows
ENG
1:38 PM
12/9/2016

Visual Code Grepper Results

Dynamic Code Analysis

Dynamic program analysis is the analysis of computer software that is performed by executing programs on a real or virtual processor. We have run OWASP's Zed Attack Proxy dynamic analysis tool on our localhost and web hosted application to find and fix vulnerabilities.

OWASP Zed Attack Proxy(ZAP) dynamic analysis tool was used to test our web application. ZAP is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications.

The ZAP report revealed several alerts and we tried to solve as many as we could. The following were the ones that were solved successfully have been described below:

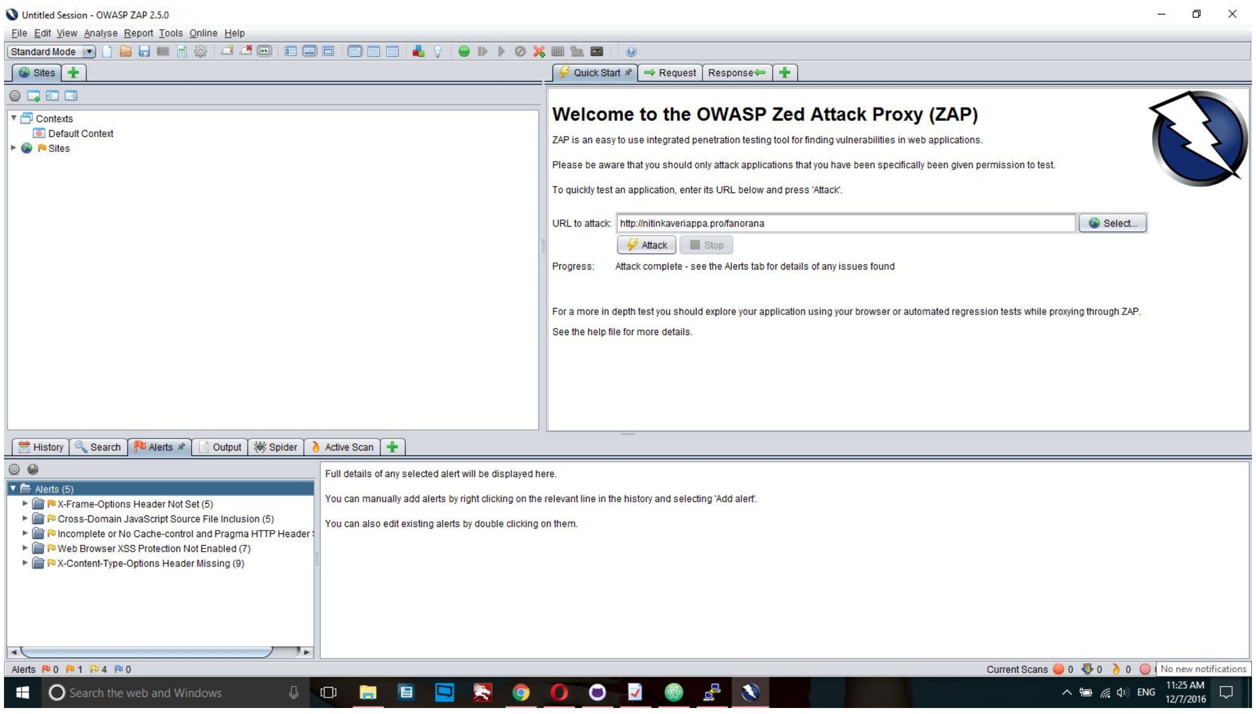
- The Directory Listings was enabled which allowed direct URL access to view subfolders and the files inside them by giving the direct URL. This is mentioned in CWE-548 Information Exposure through Directory Listing which describes it as a directory listing is inappropriately exposed, yielding potentially sensitive information to attackers. This issue was fixed by modifying the Apache2 config file to disallow directory listing.
- The AUTOCOMPLETE attribute is not disabled on an HTML FORM/INPUT element containing password type input. This is mentioned in CWE-525 Information Exposure Through Browser Caching where Passwords may be stored in browsers and retrieved. This issue was fixed by fixing the vulnerable code by setting the AUTOCOMPLETE attribute in forms or individual input elements containing password inputs by using AUTOCOMPLETE='OFF'.
- The Cookie was set without the secure flag and without the HttpOnly flag, which means the cookie can be accessed via unencrypted connections and that the cookie can be accessed by

JavaScript. This is mentioned in CWE-614 Sensitive Cookie in HTTPS session without the 'Secure' attribute where it could cause the user agent to send those cookies in plaintext over an HTTP session and in CWE-16 Configuration where weakness is typically due to the configuration of the software. This issue was fixed by setting the flags permanently to True in php.ini file on the server and also forcing all connections with the server to be encrypted i.e. https.

- The Report listed Cross-Domain JavaScript source file inclusion as one of the alerts which can be neglected as it points to the use of Google reCaptcha JavaScript file.

The ZAP report can be viewed by following this link:
https://nitinkaveriappa.pro/fanorona/ZAPReport.html

Below is a screenshot of the ZAP result with the remaining unsolved low priority issues which should technically not be a security threat. Most of these alerts are about headers that do not adversely affect our application. The two plausible threats given below are Cross Domain Javascript call and Web browser xss protection not enabled. The cross domain javascript is the javascript file for Google Recaptcha which is explained further below. Web browser xss protection not enabled is an alert when the XSS protection header is not set. But all the browsers since few years have this enabled by default and do not need any explicit command. Therefore, it is safe to say, based on the ZAP feedback we have further secured our website from all possible attacks.



Welcome to the OWASP Zed Attack Proxy (ZAP)

ZAP is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications.

Please be aware that you should only attack applications that you have been specifically been given permission to test.

To quickly test an application, enter its URL below and press 'Attack'.

URL to attack: http://nitinkaveriappa.pro/fanorana

Progress: Attack complete - see the Alerts tab for details of any issues found

For a more in depth test you should explore your application using your browser or automated regression tests while proxying through ZAP.

See the help file for more details.

Alerts (5)
- X-Frame-Options Header Not Set (5)
- Cross-Domain JavaScript Source File Inclusion (5)
- Incomplete or No Cache-control and Pragma HTTP Header
- Web Browser XSS Protection Not Enabled (7)
- X-Content-Type-Options Header Missing (9)

Full details of any selected alert will be displayed here.

You can manually add alerts by right clicking on the relevant line in the history and selecting 'Add alert'.

You can also edit existing alerts by double clicking on them.

Alerts 0 1 4 0 Current Scans 0 0 0 0 No new notifications

11:25 AM 12/7/2016

ZAP Results