

Introduction

A vulnerability is a flaw in a web application that could allow an attacker to take control of resources or perform actions in a way that compromises the security of the system. For example, some websites have been hacked by hackers and had their passwords given out to other users on their website. In this case, when a Client login on they would now see another user's password. There are many types of vulnerabilities, making it difficult to identify which ones are present within a website.

For this particular project, an employee management system developed using PHP, JavaScript, and CSS was chosen. The admin section and the user (visitor or employee) section are included in the system's features. The admin section handles all editing, updating, project management, and employee management; employees can submit work and request leaves as needed. This system's layout is straightforward to prevent user troubles when using it.

This report is about identifying vulnerabilities in this website and fixing them. To identify vulnerabilities ZAP (Zed Attack Proxy) web app scanner was used. Eighteen vulnerabilities were identified in the original application. SQL Injection, Absence of Anti CSRF Tokens, not setting connect security Policy header, Cross-domain misconfigurations, missing anti-clickjacking header, parameter tempering, and Directory browsing are some of the vulnerabilities that were found using the ZAP scanner application.

Potential Vulnerabilities



1. SQL Injection

SQL injection is a type of cyber attack in which an attacker inserts malicious code into a website's input field for execution in the database. This can allow the attacker to gain unauthorized access to the database and potentially sensitive information stored within it.

SQL injection attacks can be difficult to detect and prevent, as they often involve manipulating input fields that are not adequately sanitized. To prevent SQL injection attacks, it is important to use prepared statements and parameterized queries, and to properly escape special characters in user input.

2. Anti-CSRF

Anti-CSRF (Cross-Site Request Forgery) tokens are used to mitigate the risk of CSRF attacks in web applications. CSRF attacks occur when a malicious actor is able to trick a user into making requests to the server without their knowledge or consent. An Anti-CSRF token is a random string that is generated by the server and then included as a hidden parameter in all requests sent to the server. When the server receives the request, it validates the token to ensure that it matches the token generated by the server. The request is trusted and processed as normal if the token validates. If the token is not valid, then the request is rejected and no action is taken. Anti-CSRF tokens help to ensure that malicious requests are blocked, thus protecting the application from CSRF attacks.

3. Directory Browsing

Directory browsing on a website vulnerability is when a malicious user is able to access and view the contents of a web server's directory structure. This can allow the attacker to gain access to sensitive information such as source code, configuration files, passwords, and other sensitive data. The use of directory browsing can also result in large amounts of bandwidth being consumed.

4. Content Security Policy

Content Security Policy (CSP) is a security measure used to protect websites from malicious code injection attacks, such as cross-site scripting (XSS). CSP allows webmasters to specify a set of rules that browsers should enforce when loading content from a website. It defines which resources (scripts, images, fonts, etc.) should be loaded from which domains, and also which types of requests should be allowed or blocked. By providing a CSP, webmasters can be sure that only approved content is being loaded, thus preventing malicious code from being injected.

5. Missing Anti-click Jacking

Missing Anti-click Jacking is a big security gap that can lead to malicious users hijacking clicks intended for legitimate users. This type of attack occurs when malicious users embed malicious

code into a website or application, allowing them to intercept user clicks and redirect them to other malicious websites or applications. This attack can be used to steal user login credentials, hijack sessions, or redirect users to malicious websites containing malware. To protect against this type of attack, web developers and application developers should implement anti-click jacking measures such as disabling the ability to open links in new windows, setting a strict X-Frame-Options header, and implementing CAPTCHA verification on all sensitive forms. Additionally, web developers should be aware of the dangers of hosting untrusted content on their sites, as this can open them up to click-jacking attacks.

6. Server Leaks Information

Server links information in website vulnerabilities refers to the links found within a website that are used to access or communicate with a server. These links can be used to exploit a website or gain access to sensitive information, such as passwords and other private data. By identifying and exploiting these links, hackers can gain control of a website and use it for malicious purposes.

7. X-content Type

X-Content-Types is a security feature in HTTP headers that allows webmasters to prevent certain types of content from being served on a web page. It is used to prevent certain attacks that rely on exploiting the type of content a website serves. For example, if a website serves HTML documents as text or HTML, but an attacker is able to inject a malicious script, then the website could be vulnerable to a cross-site scripting attack. X-Content-Type-Options can be used to prevent this by ensuring that only certain types of content are served, such as application/x-www-form-urlencoded, multipart/form-data, text/plain, or application/JSON. This helps reduce the risk of an attacker exploiting the website.

8. Application Error Disclosure

A type of security vulnerability that can occur in software applications when error messages reveal sensitive information. This information can include details about the server configuration, system vulnerabilities, and other sensitive details that can be used by attackers to exploit the system. The vulnerability is usually caused by poor error handling practices, such as relying on the default error messages generated by the system or not properly securing error logs. Attackers can exploit this vulnerability by analyzing error messages and using the information they reveal to launch attacks against the system.

9. Hidden File Found

A type of security vulnerability that occurs when a web application reveals the existence of hidden files or directories that were intended to be kept confidential. These files or directories may contain sensitive information such as configuration files, backups, or private data, and their

exposure can pose a significant security risk. This vulnerability is often caused by misconfigured web servers or the lack of proper access controls, which allow unauthorized users to access hidden files or directories. Attackers can exploit this vulnerability by using common tools to scan a website and identify the presence of hidden files.

10. Vulnerable JS Library

A type of security vulnerability that occurs when a web application uses a JavaScript library with known vulnerabilities. JavaScript libraries are collections of pre-written code that developers can use to add functionality to their web applications, but if the library contains a vulnerability, the entire application can be at risk. This type of vulnerability is often caused by the use of outdated or unsupported libraries or by the failure to update libraries when new vulnerabilities are discovered. Attackers can exploit these vulnerabilities by crafting malicious input that takes advantage of the flaw in the library and can potentially compromise the entire web application.

Fixing the vulnerabilities

1. SQL Injection

```
// prepared statement for login
$sql = "SELECT * from `alogin` WHERE email = ? AND password = ?";

// create a prepared statement
$stmt = $conn->prepare($sql);
$stmt->bind_param("ss", $email, $password);

$email = checkInput($_POST['mailuid']);
$password = checkInput($_POST['pwd']);

// $result = mysqli_query($conn, $sql);
$stmt->execute();
$result = $stmt->get_result();

if (mysqli_num_rows($result) == 1) {
    //echo ("logged in");
    header("Location: ../aloginwel.php");
} else {
    header("Location: ../alogin.php?error=Invalid Email or Password");
}
```

```
// remove white spaces and unwanted special characters
function checkInput($data)
{
    $data = trim($data);
    $data = stripslashes($data);
    $data = htmlspecialchars($data);
    return $data;
}
```

- Used Prepared Statements: Prepared statements (also known as parameterized queries) are the safest way to prevent SQL injection in PHP. A prepared statement is a precompiled SQL statement that is stored in the database server. When the prepared statement is executed, user input is passed as a parameter, rather than as part of the SQL query. This prevents the user input from being interpreted as part of the SQL query.
- Used the `mysqli_real_escape_string()` Function: The `mysqli_real_escape_string()` function was used to create a legal SQL string for use in a query. This function escapes special characters in a string for use in an SQL statement.

- Used the htmlspecialchars() Function: The htmlspecialchars() function was used to convert special characters to HTML entities. This prevents any malicious code from being executed.
- Used the stripslashes() Function: The stripslashes() function was used to remove the backslashes added by the addslashes() function. This function can be used to clean up data retrieved from a database or from an HTML form.

2. Anti-CSRF

```

Run terminal Help
alogin.php - 3/3project - Visual Studio Code
alogin.php U X
alogin.php > html > head > meta
U
28 <div class="loginbox">
29 
30 <h1>Login Here</h1>
31
32 <?php
33 $token = bin2hex(random_bytes(32));
34 $_SESSION['csrf_token'] = $token;
35 ?>
36
37 <form action="process/aprocess.php" method="POST">
38 <p>Email</p>
39 <input type="text" name="mailuid" placeholder="Enter Email Address" required="required">
40 <p>Password</p>
41 <input type="password" name="pwd" placeholder="Enter Password" required="required">
42 <input type="hidden" name="csrf_token" value="{?php echo $token; ?}>"
43 <input type="submit" name="login-submit" value="Login">
44
45 </form>
46
47 </div>
48 </body>
49 </html>
U
U
M
M
U
M
M

```

```

alogin.php U
aprocess.php M X
process > aprocess.php > ...
1 <?php
2
3 require_once('dbh.php');
4 require_once('checkInput.php');
5
6 $csrf_token = checkInput(isset($_POST['csrf_token']) && $_POST['csrf_token']);
7
8 // check if csrf token is valid
9 if (!isset($_SESSION['csrf_token']) || !hash_equals($_SESSION['csrf_token'], $csrf_token)) {
10     header("Location: ../403.html#error=csrf");
11     exit;
12 }
13
14 // $sql = "SELECT * from `login` WHERE email = '$email' AND password = '$password'";
15
16 // prepared statement for login
17 $sql = "SELECT * from `login` WHERE email = ? AND password = ?";
18
19 // create a prepared statement

```

A 32-byte-long randomly generated token was set to the session and it was passed as a hidden field through a form to the backend. Once the backend received the token we check whether the hash value is equal to the token that was received through the form. If it is not equal the page is redirected to a page that shows an error message and if it is equal the content in the aprocess.php is shown to the user.

3. Directory Browsing

```
244 # below.
245 #
246 #
247 #
248 # DocumentRoot: The directory out of which you will serve your
249 # documents. By default, all requests are taken from this directory, but
250 # symbolic links and aliases may be used to point to other locations.
251 #
252 DocumentRoot "D:/XAMPP/htdocs/ems/370project"
253 <Directory "D:/XAMPP/htdocs/ems/370project">
254 #
255 # Possible values for the Options directive are "None", "All",
256 # or any combination of:
257 #   Indexes Includes FollowSymLinks SymLinksifOwnerMatch ExecCGI MultiViews
258 #
259 # Note that "MultiViews" must be named *explicitly* --- "Options All"
260 # doesn't give it to you.
261 #
262 # The Options directive is both complicated and important. Please see
263 # http://httpd.apache.org/docs/2.4/mod/core.html#options
264 # for more information.
265 #
266 # Options Indexes FollowSymLinks Includes ExecCGI
267 Options -Indexes +FollowSymLinks
268 #
269 #
270 #
271 # AllowOverride controls what directives may be placed in .htaccess files.
272 # It can be "All", "None", or any combination of the keywords:
```

Added "Indexes" and
"FollowSymLinks" options to the
directory that contains the files of the
web application
(D:/XAMPP/htdocs/ems/370project).

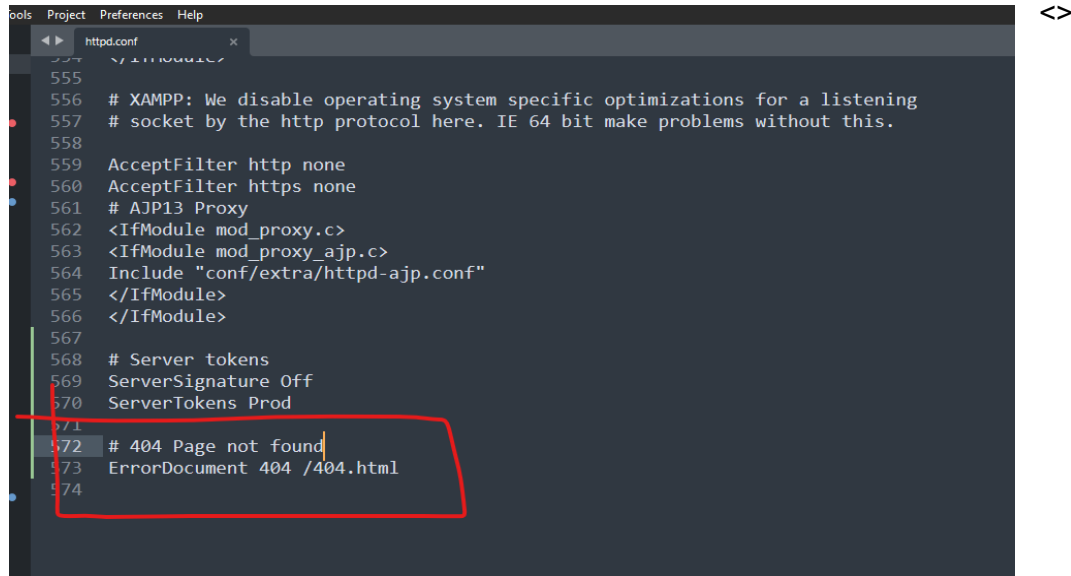
4. Content Security Policy

```
aprocess.php M index.html M X
index.html > html > body
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="utf-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1">
7   <meta http-equiv="Content-Security-Policy"
8     content="default-src 'self' https://fonts.googleapis.com; font-src 'self' https://fonts.gstatic.com;">
9   <title>XYZ Corporation</title>
10  <link href="https://fonts.googleapis.com/css?family=Lobster|Montserrat" rel="stylesheet">
11  <link rel="stylesheet" type="text/css" href="styleindex.css">
12 </head>
13
14 <body>
15   <header>
16     <nav>
17       <h1>XYZ Corp.</h1>
18       <ul id="navli">
19         <li><a class="homered" href="index.html">HOME</a></li>
20         <li><a class="homeblack" href="aboutus.html">ABOUT US</a></li>
```

```
589 # Disable server-status
590 <Location /server-status>
591   SetHandler server-status
592   Order deny,allow
593   Deny from all
594 </Location>
595
596 # Disable X-Frame-Option
597 Header set X-Frame-Options "DENY"
598
```

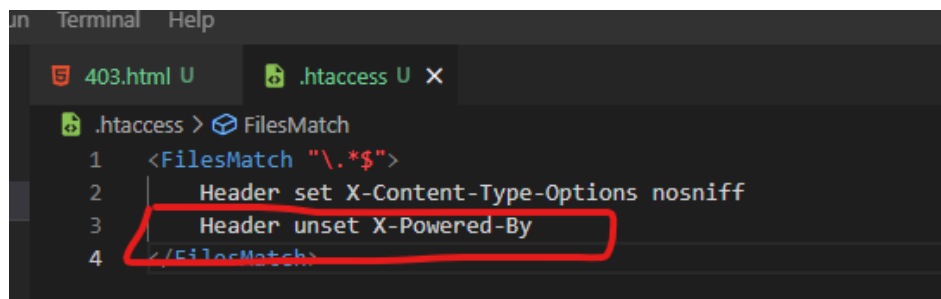
Added the Content Security Policy **file** to the header.

5. Missing Anti-click Jacking

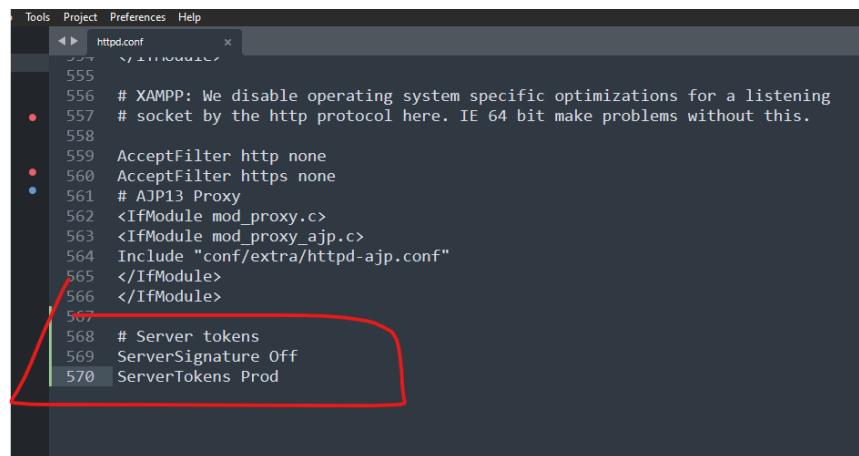


```
555
556 # XAMPP: We disable operating system specific optimizations for a listening
557 # socket by the http protocol here. IE 64 bit make problems without this.
558
559 AcceptFilter http none
560 AcceptFilter https none
561 # AJP13 Proxy
562 <IfModule mod_proxy.c>
563 <IfModule mod_proxy_ajp.c>
564 Include "conf/extra/httpd-ajp.conf"
565 </IfModule>
566 </IfModule>
567
568 # Server tokens
569 ServerSignature Off
570 ServerTokens Prod
571
572 # 404 Page not found
573 ErrorDocument 404 /404.html
574
```

6. Server Leaks Information



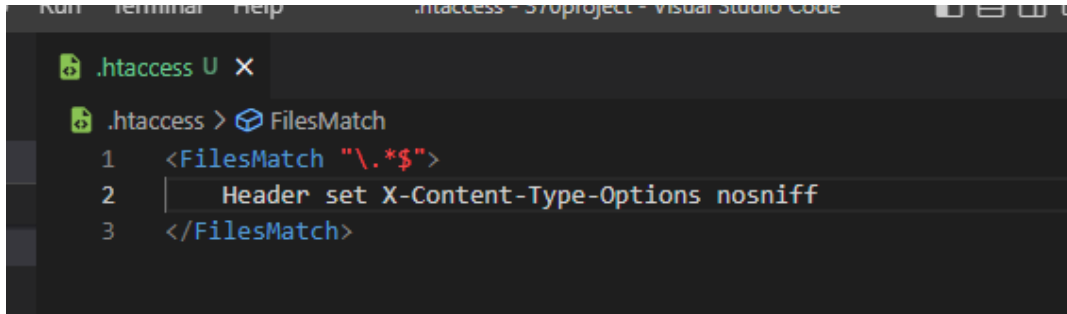
```
403.html U  .htaccess U X
.htaccess > FilesMatch
1 <FilesMatch "\.*$">
2   Header set X-Content-Type-Options nosniff
3   Header unset X-Powered-By
4 </FilesMatch>
```



```
Tools Project Preferences Help
httpd.conf
555
556 # XAMPP: We disable operating system specific optimizations for a listening
557 # socket by the http protocol here. IE 64 bit make problems without this.
558
559 AcceptFilter http none
560 AcceptFilter https none
561 # AJP13 Proxy
562 <IfModule mod_proxy.c>
563 <IfModule mod_proxy_ajp.c>
564 Include "conf/extra/httpd-ajp.conf"
565 </IfModule>
566 </IfModule>
567
568 # Server tokens
569 ServerSignature Off
570 ServerTokens Prod
```

Since the response page shows details of the server, we unset the header.

7. X-content Type



```
.htaccess > FilesMatch
1 <FilesMatch "\.*$">
2   Header set X-Content-Type-Options nosniff
3 </FilesMatch>
```

Added X-Content-Type-Options which blocks all request if there "style" MIME-type is not text/css and JavaScript MIME-type. Plus it enables the cross origin if there MIME-Type text/html, text/plain, text/json, application/json and any type of xml extension.

8. Application Error Disclosure

There were couple of places in the code that throws errors while executing. In order to address this vulnerability we fixed these errors.

9. Hidden File Found

Fixed this vulnerability by deleting the file that was created when initiated git.

Vulnerabilities which are not fixed

1. Vulnerable JS Library

This library is used for the datepicker and therefore, this vulnerability can not be solved.