

████ - Candidate Assessment

Web Application Penetration Test

Written by Nathan Schwartz

Executive Summary

Synopsis

On Thursday, April 13th, █████ gave us access to their Vulnerable Web Application. █████ has given us until Friday, April 21st to perform a security assessment of the target. In order to access the environment, we were given multiple instances of the web application for effective testing.

Scope

The scope for this engagement is the web application provided by █████. Multiple instances of the web application were hosted for us to access. This is a time-bound black-box assessment in which we were given a deadline alongside a brief description of the target.

In scope:

- The web application itself (various instances provided)



- The MySQL database underlying the web application

Key Findings

The assessment uncovered multiple critical issues, including but not limited to:

Remote Code Execution via eval(): Any unauthenticated user can execute code on the server, completely compromising the confidentiality, integrity, and availability of the web application and server.

Unauthenticated Admin Console Access: The admin console can be accessed without any authentication, allowing for unauthorized admin account creation, user deletion, and asset management.

Default or Weak Credentials: A high-privileged user was found using insecure, easily guessable passwords.

Strategic Recommendations

In general, ensure that secure coding standards are adopted and adhered to by all developers. This includes following best practices for input validation, output encoding, error handling, and access control. Implement security controls such as firewalls, intrusion detection/prevention systems (IDS/IPS), and web application firewalls (WAFs) as these can help detect various attacks. Conduct code reviews and static analysis to identify potential vulnerabilities and ensure that code is free from common security issues. Finally, performing regular security assessments, including vulnerability scans and penetration testing, can help detect any new vulnerabilities and allow for prompt remediation.

Table of Findings

Page #	Vulnerability	Location	Rating
3	Remote Code Execution via eval()	POST /Auth/signUp	Critical
4	Lacking Principle of Least Privilege	POST /Auth/signUp	Critical
5	Unauthenticated Admin Console Access	GET /Admin/console POST /Admin/user/ GET /Admin/user/:user DELETE /Admin/asset/:id	Critical
6	Default or Weak Credentials	GET /Secret.txt	Critical
7	Stored Cross-Site Scripting	POST /view/Comments	High
8	Systematic SQLi Vulnerabilities	POST /views/Comments POST /Admin/login /app/src/sql/sql.js	High
9	Directory Traversal	GET /Secret.txt GET /anotherXml.xml GET /nodejs-express-web-applications.zip GET /:asset-name	High
10	Unauthenticated Asset Management	GET /Assets/:id POST /Assets/search/ GET /Assets/delete/:id	High
11	Reflected Cross-Site Scripting	POST /Assets/search/	Medium
12	User Impersonation via WebSockets	GET /support/chat	Medium
13	Lacking CSRF Protections	POST /Admin/user/ GET /Admin/user/:user DELETE /Admin/asset/:id	Medium
14	Lacking Content Filter for Assets	POST /Assets/upload	Low
15	Lacking Logout Mechanisms	GET /logout	Low
16	Accessing robots.txt	GET /robots.txt	Informational

Critical

Remote Code Execution via eval()

Location

POST /Auth/signUp

Impact

Any unauthenticated user can execute code on the server, completely compromising the confidentiality, integrity, and availability of the web application and server.

Description

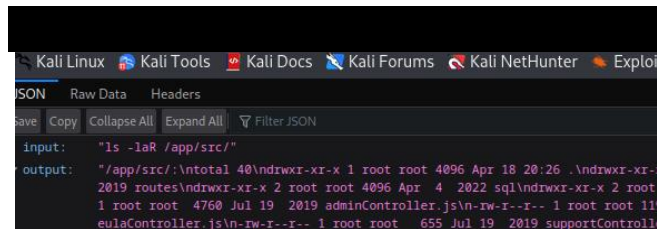
A Remote Code Execution (RCE) vulnerability was uncovered within the captcha functionality of the sign-up page. The captcha is implemented such that user-provided input, including arbitrary commands, will be evaluated with `eval()` in NodeJS.

Because the server is running using NodeJS and Express, the response object can be modified to give the adversary direct feedback on any commands run or files listed. While reverse shells and bind shells cannot be created on the web server due to its firewall configuration, a makeshift webshell can be implemented using this vulnerability.

Reproduction

Using the /Auth/signUp page, any unauthenticated user can insert the following command in the captcha and receive a response from the server:

```
const cmd='[command here]';
const execSync = require('child_process').execSync;
code = execSync(cmd).toString();
res.send({input: cmd, output: code});
```



This was used to download the source code for the web application which confirmed the existence of multiple other findings, extract MySQL credentials from `sql.js`, and discover a hidden message in files such as `/app/public-folder/admin_file.jpg`.

Remediation

The use of `eval()` is inherently dangerous and should be avoided. The captcha implementation should be updated to use a safer approach. In addition, user input should be properly sanitized and validated before being used in any server-side processing. This can help prevent injection attacks and other vulnerabilities.

Critical

Lacking Principle of Least Privilege

Location

POST /Auth/signUp

Impact

This web application is being run as the root user, which leads to the complete compromise of the web server in cases of command injection. In general practice, service accounts should be used such as www-data or node which have only permissions necessary for the web applications basic function.

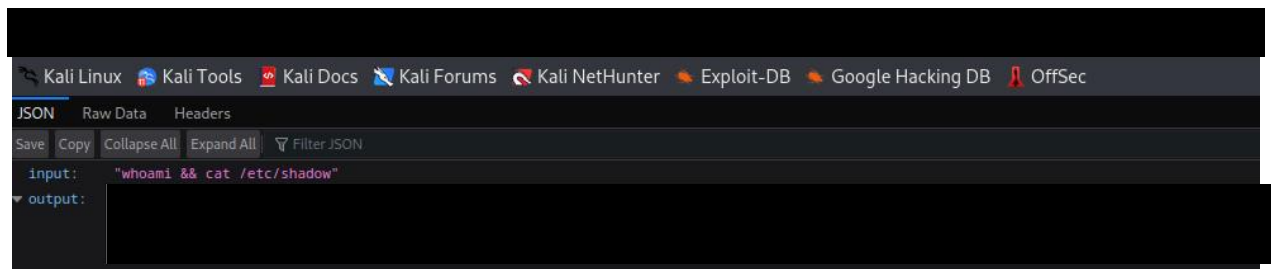
Description

This vulnerability was discovered as a product of the Remote Code Execution via eval() vulnerability. While service accounts such as www-data and node exist on the web server, it appears that the application is running as the root user. This user has the permissions necessary to do anything on the server, completely compromising confidentiality, integrity, and availability of the web application.

As a general rule, the root user should never be logged in or run web applications. Administrator, developer, and service accounts are a safer alternative as they are only given the permissions necessary to perform their duties. This limits the usefulness of any command injection vulnerabilities being leveraged by an adversary.

Reproduction

Using the Remote Code Execution via eval() vulnerability, any unauthenticated user can see that their commands are being run as the root user:



Remediation

It is recommended to use administrator, developer, and service accounts on the web server. Using the www-data or node accounts to run the web application would mitigate the damages of any future command injection vulnerabilities.

Critical

Unauthenticated Admin Console Access

Location

GET /Admin/console
POST /Admin/user/
GET /Admin/user/:user
DELETE /Admin/asset/:id

Impact

Any unauthenticated user can access the admin console at /Admin/console, completely bypassing any login mechanisms such as the admin login page. This allows them to create new admin users with elevated privileges and delete important assets or user accounts.

Description

Normally, the admin console could be accessed by providing the correct username and password in the login page at /Admin/login. However, there exists a lack of access controls on various routes within the web application which allows an unauthenticated user full access to administrator functions.

The admin console is a page only intended for use by administrators, as it has access to admin creation, asset viewing and removal, and user deletion. Allowing an unprivileged user access to these functions could lead to the disruption of the web applications normal operations.

Reproduction

Navigating to /Admin/console, any unauthenticated user can access the following page:

The screenshot displays the Admin Console dashboard. At the top, there are two summary cards: 'Total Users' with a value of 3 and 'Total Assets' with a value of 3. Below these, the 'Users' section contains a table with columns for Username, Role, and Delete User. The table lists three users: 'admin' (role: admin), 'user' (role: user), and 'unauth' (role: admin). Each user has a red 'Delete User' button next to it. To the right of the Users table is a form titled 'Add Administrator' with fields for Full name, Username, Password, Confirm Password, and Email, along with a 'Create Admin User' button. At the bottom, the 'Assets' section shows a table with columns for Asset Name, Owner, Date Uploaded, and ID. The table contains one entry: 'admin' (owner: admin, date: Mon Aug 04 1987 00:14:00 GMT+0000 (UTC), id: 7797).

Remediation

Access Controls should be implemented to restrict access to the admin console only to authorized users. Authentication should be required for any access to sensitive functionality such as user management and asset deletion. In addition, monitoring user activity for suspicious behavior such as creating or deleting admin accounts could mitigate damages and prevent further exploitation.

Critical

Default or Weak Credentials

Location

GET /Secret.txt

Impact

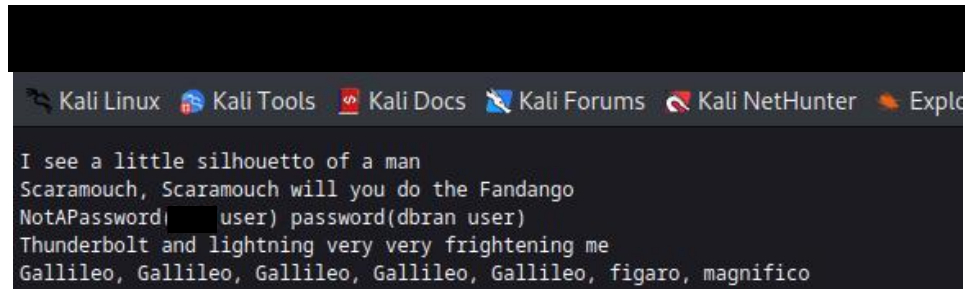
A high-privileged user was found using insecure, easily guessable passwords. Any unprivileged user can gain control of an admin account using the credentials dbran:password.

Description

High-privileged accounts should use secure passwords and two-factor authentication to prevent unauthorized individuals from accessing them. In this case, the admin user, dbran, was using an easily guessable password, password, to secure their account.

Alongside using weak credentials, this users password was found on the website in plaintext on the /Secret.txt page. This page can be discovered by any unauthenticated user through an existing directory traversal or admin console access vulnerability. This page also stores insecure credentials for the [REDACTED] user.

Reproduction



```
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploits

I see a little silhouette of a man
Scaramouch, Scaramouch will you do the Fandango
NotAPassword([REDACTED] user) password(dbran user)
Thunderbolt and lightning very very frightening me
Gallileo, Gallileo, Gallileo, Gallileo, Gallileo, figaro, magnifico
```

Any unprivileged user can access the admin console using the credentials dbran:password. This allows them to create new admin users with elevated privileges and delete important assets or user accounts.

Remediation

Ensure that users choose a unique, secure, 16+ character password when creating high-privileged accounts. Alternatively, high-privileged users should be using a password manager to create secure passwords for their accounts. In general, Two-Factor Authentication (2FA) is an effective means of securing user accounts as they require more than one mechanism in order to log into an account.

High

Stored Cross-Site Scripting

Location

POST /view/Comments

Impact

Any unprivileged user can store javascript code on a webpage in order to steal user data, redirect users to malicious websites, and damage the reputation of the affected website and result in loss of trust from its users.

Description

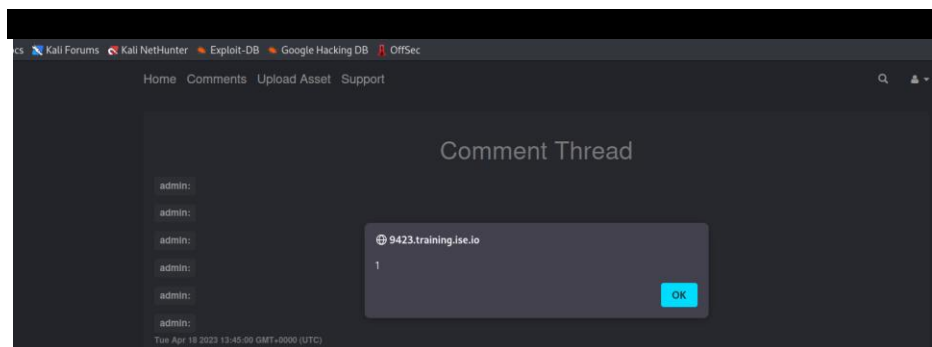
Stored Cross-Site Scripting (XSS) is a type of web application vulnerability that allows an adversary to inject malicious code into a website, which is then stored and displayed to all users who view the affected page. Unlike reflected XSS, which occurs immediately after the malicious code is injected, stored XSS remains in the web application and can continue to impact users who view the page until it is removed.

This can have a significant impact on the security and functionality of a web application. Adversaries can utilize these vulnerabilities to steal session cookies and impersonate users, implement javascript keyloggers to gather credentials, and redirect users to malicious clones of the affected website.

Reproduction

Using the /view/Comments page, inputting the following shows a proof of concept for Stored XSS:

```
<audio src/onerror=alert(1)>
```



Remediation

While the current approach taken is blacklisting certain words, this can be bypassed via alternate javascript execution methods. It is recommended to sanitize all input on arrival based on what is expected or valid. Encoding data on output is effective in preventing XSS vulnerabilities. Finally, using a detailed Content Security Policy (CSP) can help mitigate any additional XSS issues that could still occur.

High

Systematic SQLi Vulnerabilities

Location

POST /views/Comments
POST /Admin/login
/app/src/sql/sql.js

Impact

Any unprivileged user can read, modify, or delete data within the MySQL database. This compromises the confidentiality, integrity, and availability of all sensitive information stored by this web application thus damaging its overall reputation.

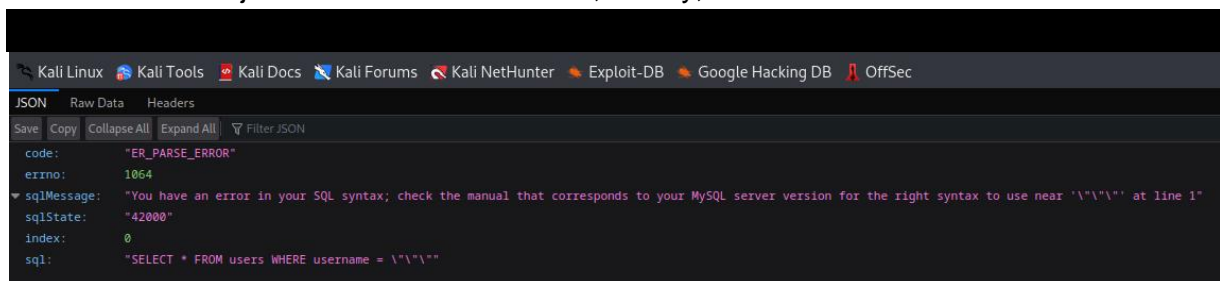
Description

SQL injection (SQLi) is a type of vulnerability that allows an attacker to manipulate SQL statements used by an application to communicate with a database. Systematic SQLi vulnerabilities refer to a pattern of SQL injection across multiple points in the application. This can lead to unauthorized access to sensitive data, modification of data, or disruption of the system's normal operations.

These vulnerabilities can lead to unauthorized access to sensitive data or functionality, including the theft or modification of sensitive data. As a result of this, it can cause disruption of the system's normal operations. An adversary can use this access to extract valuable information, perform malicious activities, or launch further attacks against the system.

Reproduction

Inserting the " character in the /views/Comments and /Admin/login pages demonstrates how SQL injection can be used to read, modify, or delete data:



```
code: "ER_PARSE_ERROR"
errno: 1064
sqlMessage: "You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '\"'\"' at line 1"
sqlState: "42000"
index: 0
sql: "SELECT * FROM users WHERE username = '\"'\"'"

```

Remediation

The use of prepared statements helps to prevent SQL injection attacks by preparing the query action followed by supplying the user input. This prevents users from inserting their own statements or queries, as the binary has only prepared a single action. In addition, using parameterized queries helps by separating user input from SQL statements. This is seen in various functions within the /app/src/sql/sql.js file, allowing those functions to be secure. Finally, sanitizing user input can prevent an adversary from injecting malicious SQL code into an application.

High

Directory Traversal

Location

```
GET /Secret.txt
GET /anotherXml.xml
GET /nodejs-express-web-applications.zip
GET /:asset-name
```

Impact

Any unauthenticated user can access various files stored within the `/app/src/assets/` directory. This allows for an adversary to uncover plaintext credentials and view assets without any mechanism for authentication.

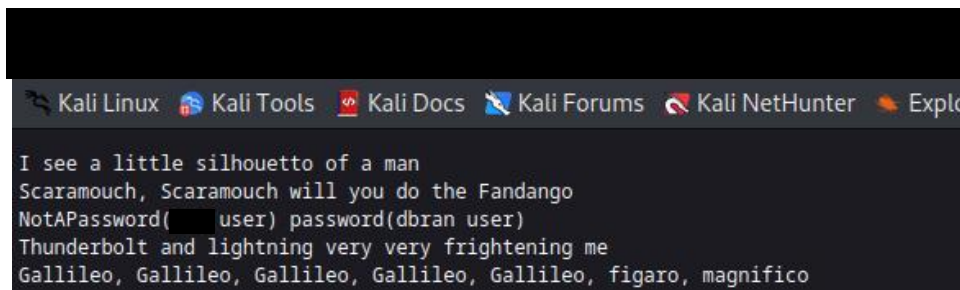
Description

Directory Traversal, also known as Path Traversal, is a web application vulnerability that allows an adversary to access files or directories outside of the web application's root directory.

This web application hosts static files located in the `/app/src/assets/` directory. While this appears to be intentional, several unused and sensitive files are accessible to any unauthenticated user via this file-hosting functionality.

Reproduction

Any unauthenticated user can access the `Secret.txt` asset, which contains plaintext credentials to various user accounts:

A screenshot of a Kali Linux terminal window. The terminal has a dark background with a light-colored text cursor. The top of the window shows a taskbar with icons for Kali Linux, Kali Tools, Kali Docs, Kali Forums, Kali NetHunter, and a folder icon labeled 'Expl'. The terminal output shows the contents of the Secret.txt file, which is a poem about Scaramouch and a list of passwords for various users.

```
I see a little silhouetto of a man
Scaramouch, Scaramouch will you do the Fandango
NotAPassword( user) password(dbran user)
Thunderbolt and lightning very very frightening me
Gallileo, Gallileo, Gallileo, Gallileo, Gallileo, figaro, magnifico
```

Remediation

When storing sensitive and unused files, avoid placing them in a folder which is freely accessible from a web application. The file-hosting functionality appears intended within `/app/app/js`, however we recommend disabling this feature as it does not comply with demonstrated business logic.

High

Unauthenticated Asset Management

Location

GET /Assets/:id
POST /Assets/search/
GET /Assets/delete/:id

Impact

Any unauthenticated user can view or delete assets owned by any other user. This vulnerability goes against the main functionality of the website and damages the reputation of the web application overall.

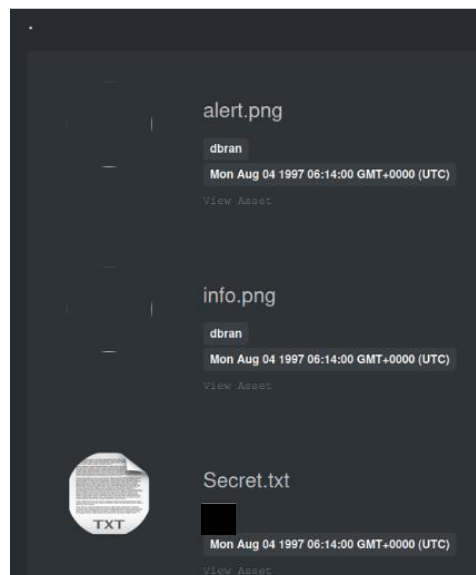
Description

Part of the business logic for this web application is that only an assets owner should be able to view, modify, or delete an asset. Despite this, there are ways for an unauthenticated, unprivileged user to access these resources.

Allowing assets to be viewed or deleted by non-owners demonstrates a compromise in the confidentiality and availability of those files. This can damage the overall business reputation as users discover that their private information is not secure or accessible.

Reproduction

Any unauthenticated user can search for assets with "." to discover other users files:



Remediation

Access controls should be put in place of these important functions to ensure that only the owner of an asset can access these resources. Ensure that a user is authenticated and authorized before displaying an asset or allowing them to appear in a search.

Medium

Reflected Cross-Site Scripting

Location

POST /Assets/search/

Impact

Any unprivileged user can send a maliciously crafted request in order to execute javascript. This allows an adversary to steal user data, redirect users to malicious websites, and damage the reputation of the affected website which could result in loss of trust from its users.

Description

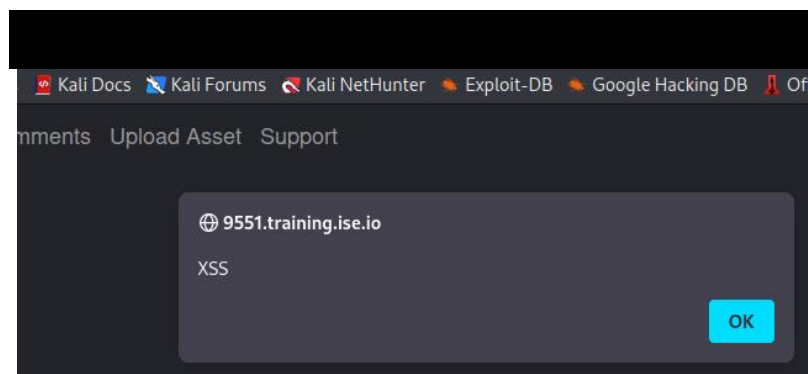
Reflected Cross-Site Scripting (XSS) is a web application vulnerability that occurs when user input is not properly sanitized and is reflected back to the user without proper encoding. An adversary can exploit this vulnerability by injecting malicious code into the web application, which is then reflected back to the user's browser and executed.

This can have a significant impact on the security and functionality of a web application. Adversaries can utilize these vulnerabilities to steal session cookies and impersonate users, implement javascript keyloggers to gather credentials, and redirect users to malicious clones of the affected website.

Reproduction

Using the search function, inputting the following shows a proof of concept for Reflected XSS:

```
<script>alert('XSS')</script>
```



Remediation

It is recommended to sanitize all input on arrival based on what is expected or valid. Encoding data on output is effective in preventing XSS vulnerabilities. Finally, using a detailed Content Security Policy (CSP) can help mitigate any additional XSS issues that could still occur.

Medium

User Impersonation via WebSockets

Location

GET /support/chat

Impact

Any unprivileged user can impersonate an administrator account via WebSockets. This violation of message integrity can lead to a deterioration of user trust in the web application.

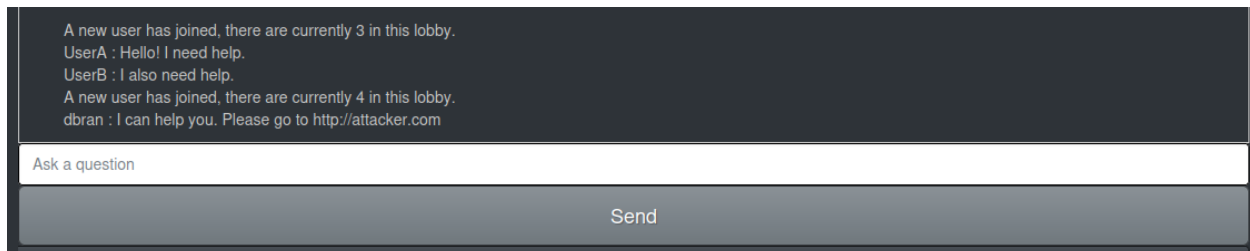
Description

WebSockets is a protocol that provides a bidirectional communication channel between a client and a server over a single TCP connection. While WebSockets can enhance the real-time communication capabilities of a web application, they can also introduce security vulnerabilities if not implemented correctly.

Because the client sends their name alongside their message, an adversary can intercept the request and change their username in order to impersonate someone. This can be utilized during a phishing campaign to gain the trust of a target and redirect them to a malicious resource.

Reproduction

In the following example, UserB modified their message to look like an admin account (dbran) joined the lobby and directed them to the attackers website:



The screenshot shows a chat interface with a dark background. The chat history contains the following messages:

- A new user has joined, there are currently 3 in this lobby.
- UserA : Hello! I need help.
- UserB : I also need help.
- A new user has joined, there are currently 4 in this lobby.
- dbran : I can help you. Please go to <http://attacker.com>

Below the chat history is a text input field with the placeholder text "Ask a question". At the bottom of the interface is a grey "Send" button.

Remediation

Improve the WebSockets implementation such that a users identity in the chat is tied to the identity they logged in with. Users should not be able to modify chat information stored on the server such as who sent a message and how many users are in a lobby. Finally, monitor the chat room for suspicious activity. If users are pretending to be moderators or administrators, investigate the issue and resolve it as soon as possible.

Medium

Lacking CSRF Protections

Location

POST /Admin/user/
GET /Admin/user/:user
DELETE /Admin/asset/:id

Impact

An adversary can craft a request allowing them to execute unauthorized actions on behalf of a victim user, such as changing their password, uploading unauthorized assets, or even deleting their account. This can lead to data theft, financial loss, reputational damage, and legal liabilities.

Description

Cross-Site Request Forgery (CSRF) is an attack in which an adversary exploits the trust of a victim user's browser to perform unauthorized actions on a vulnerable website. CSRF attacks work by tricking a victim user into clicking on a specially crafted link or visiting a malicious website while they are logged into the vulnerable website. The adversary can then perform actions on the victim user's behalf without their knowledge or consent.

Reproduction

An adversary could craft a website with the following HTML on it:

```

```

If an admin user navigated to this website, they would inadvertently delete a specific user.

Remediation

Implement CSRF tokens to ensure that requests made to the server originate from a trusted source. A CSRF token is a unique token generated by the server and included in the HTML form or request. The server then verifies that the token is present and matches the expected value before processing the request.

In addition, Same-Site cookies help to ensure that cookies are only sent with requests that originate from the same website. This can prevent CSRF attacks from external websites.

Finally, ensure that sensitive actions are performed using POST instead of GET requests. This can help prevent an adversary from crafting URLs that perform unauthorized actions.

Low

Lacking Content Filter for Assets

Location

POST /Assets/upload

Impact

Being able to upload or download the EICAR file may indicate a vulnerability in the system that could allow an adversary to upload or download actual malware, potentially causing serious harm to the system and its users.

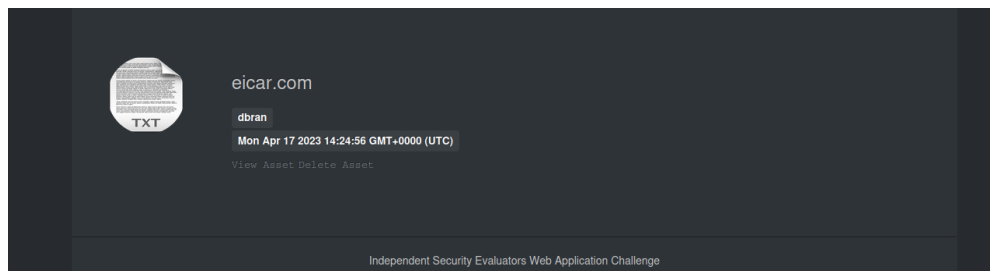
Description

The EICAR file is a standard test file used to test antivirus software and other security products. It is not malicious and cannot harm your computer. However, being able to upload or download the EICAR file can be a sign of a vulnerability in the system that could be exploited by adversaries to upload or download actual malware.

Uploading or downloading the EICAR file may indicate that the system does not properly filter or validate file uploads and downloads, allowing arbitrary files to be uploaded or downloaded. Adversaries can take advantage of this vulnerability by uploading malicious files disguised as harmless files, or by downloading sensitive files from the system.

Reproduction

The following is an example of a successfully uploaded EICAR file:



Remediation

Ensure that the system properly filters and validates file uploads and downloads. This can include checking file extensions, file types, file sizes, and blocking any suspicious files.

Low

Lacking Logout Mechanisms

Location

GET /logout

Impact

The logout button does not invalidate a session cookie, potentially allowing an adversary to continue using the user's session.

Description

When the logout button does not invalidate a session cookie, it may indicate a problem with the session management functionality of the application. This can occur if the application is not properly clearing session cookies or if the server-side code is not properly handling the logout process. This can result in session cookies remaining active even after the user has clicked the logout button, allowing an adversary to continue using the user's session.

This can leave the user's account vulnerable to attacks such as session hijacking, where an adversary can take control of the user's account by using the session cookies that remain active. This can allow the adversary to access sensitive information and perform unauthorized actions on behalf of the user, potentially causing significant harm to the user and the system.

Reproduction

Any unprivileged user can note their session cookie, logout, and navigate to a page using their session cookie.

Remediation

Ensure that session cookies are being cleared on both the server-side and client-side when logging out a user. In addition, CSRF tokens can help prevent session hijacking attacks as it ensures that the user's session is not being used by an adversary.

Informational

Accessing robots.txt

Location

GET /robots.txt

Impact

The file robots.txt is used to give instructions to web robots, such as search engine crawlers, about locations within the web site that robots are allowed, or not allowed, to crawl and index.

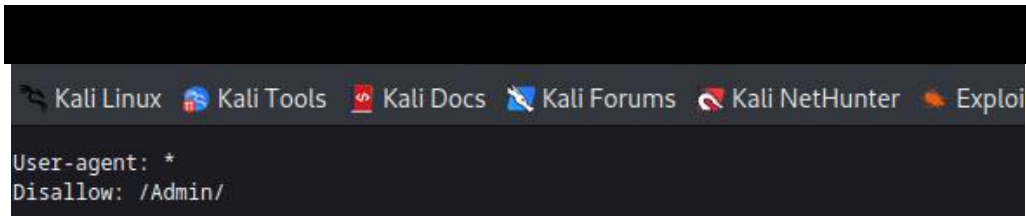
Description

The presence of the robots.txt does not in itself present any kind of security vulnerability. However, it is often used to identify restricted or private areas of a site's contents. The information in the file may therefore help an adversary to map out the site's contents, especially if some of the locations identified are not linked from elsewhere in the site. If the application relies on robots.txt to protect access to these areas, and does not enforce proper access control over them, then this presents a serious vulnerability.

In this case, robots.txt demonstrates the existence of the /Admin/ route, leading users to find the /Admin/login page.

Reproduction

Any unauthorized user can access the /robots.txt page shown below:



```
User-agent: *
Disallow: /Admin/
```

Remediation

The robots.txt file is not itself a security threat, and its correct use can represent good practice for non-security reasons. You should not assume that all web robots will honor the file's instructions. Rather, assume that adversaries will pay close attention to any locations identified in the file. Do not rely on robots.txt to provide any kind of protection over unauthorized access.