Documentation

VulWitch is written using node.js

Go to https://github.com/drboyle2/DBDSW-VulWitch and download the code.

Running the Vulwitch server locally requires Node.js 12 or higher to be installed, along with the following apt packages:

- Flawfinder
- Wkhtmltopdf
- npm

Also must install the following npm packages:

- bcrypt
- body-parser
- Path
- Cors
- mysql2
- jsonwebtoken
- Express
- Pdfkit
- Pdf-lib
- Puppeteer

In a directory, there is also sub-directories:

- app.js
- database.js
- Package.json
- Package-lock.json
- README
- verifytoken.js
- <directory> Public
    - script.js
    - Index.html
    - register.html
- <directory> models
    - users.js

From the command line in this directory, run the command

```
npm start
```

or,

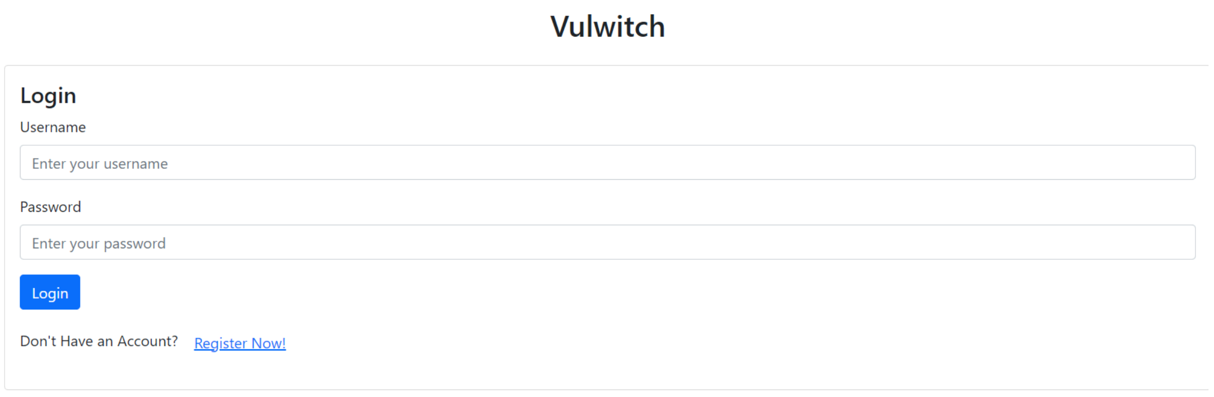
```
node app.js
```

The following lines should be printed on the console

```
> DBDSWVulwitch@1.0.0 start
> node app.js

Server is running on http://localhost:3000
Connected to AWS RDS MySQL database as ID ###
```

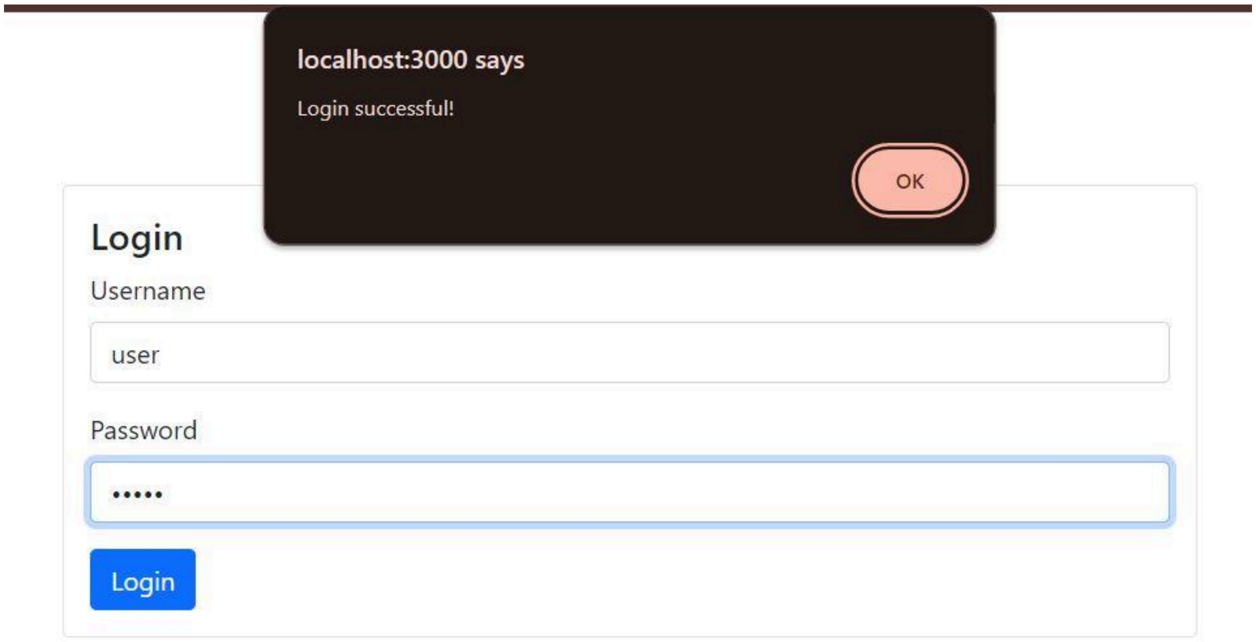
To use the Vulwitch web app:

- Open the web browser and go to http://localhost:3000
- This should display the login page for Vulwitch
- **Our website includes rate limiting on HTTP requests from a session, preventing a DDoS attack. This limit is 10 http requests per session**



UserAuthentication:

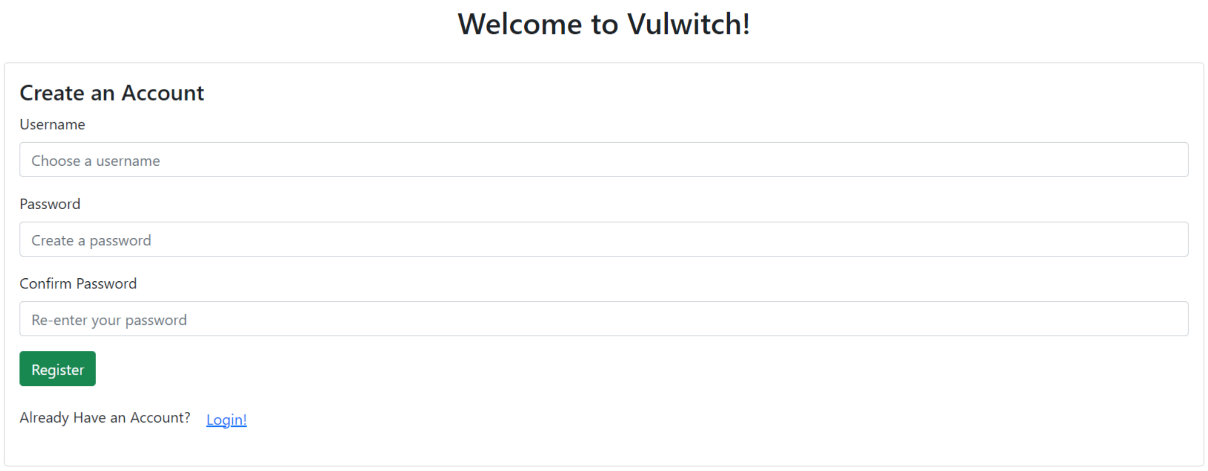
- Enter any registered "Username" and "Password" to log in.
- **If a user is unable to log in correctly within 10 tries, it will temporarily lock the account to prevent brute force attacks.**
- On successful login, a JWT token will be generated and stored in the browser's local storage.


localhost:3000 says

Login successful!

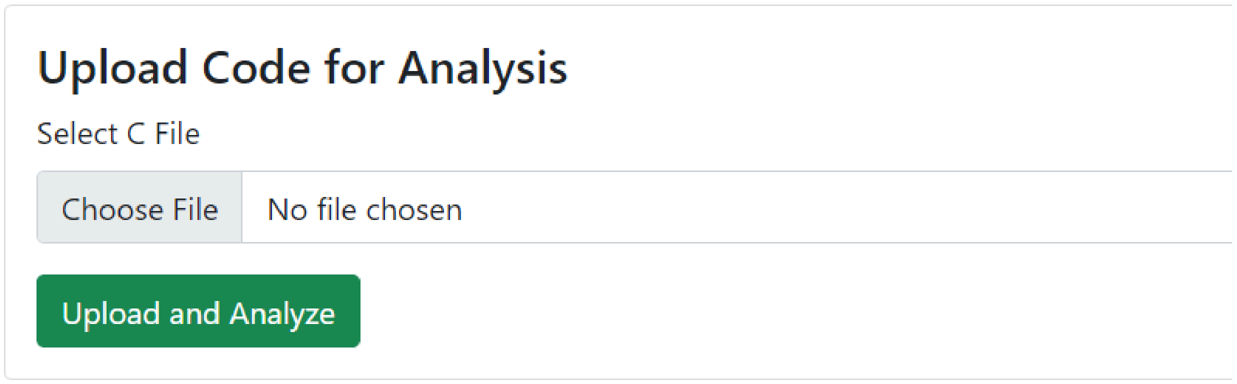
Adding New Users:

- In the current version a user can be added by entering any “Username” and “Password”.
- These credentials can be used to login as a user after they have been added.
- The new user will be added to the database with the password securely hashed using bcrypt

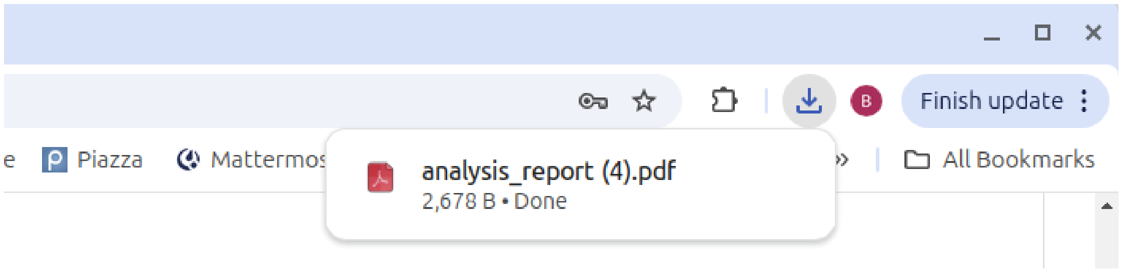


C File Upload:

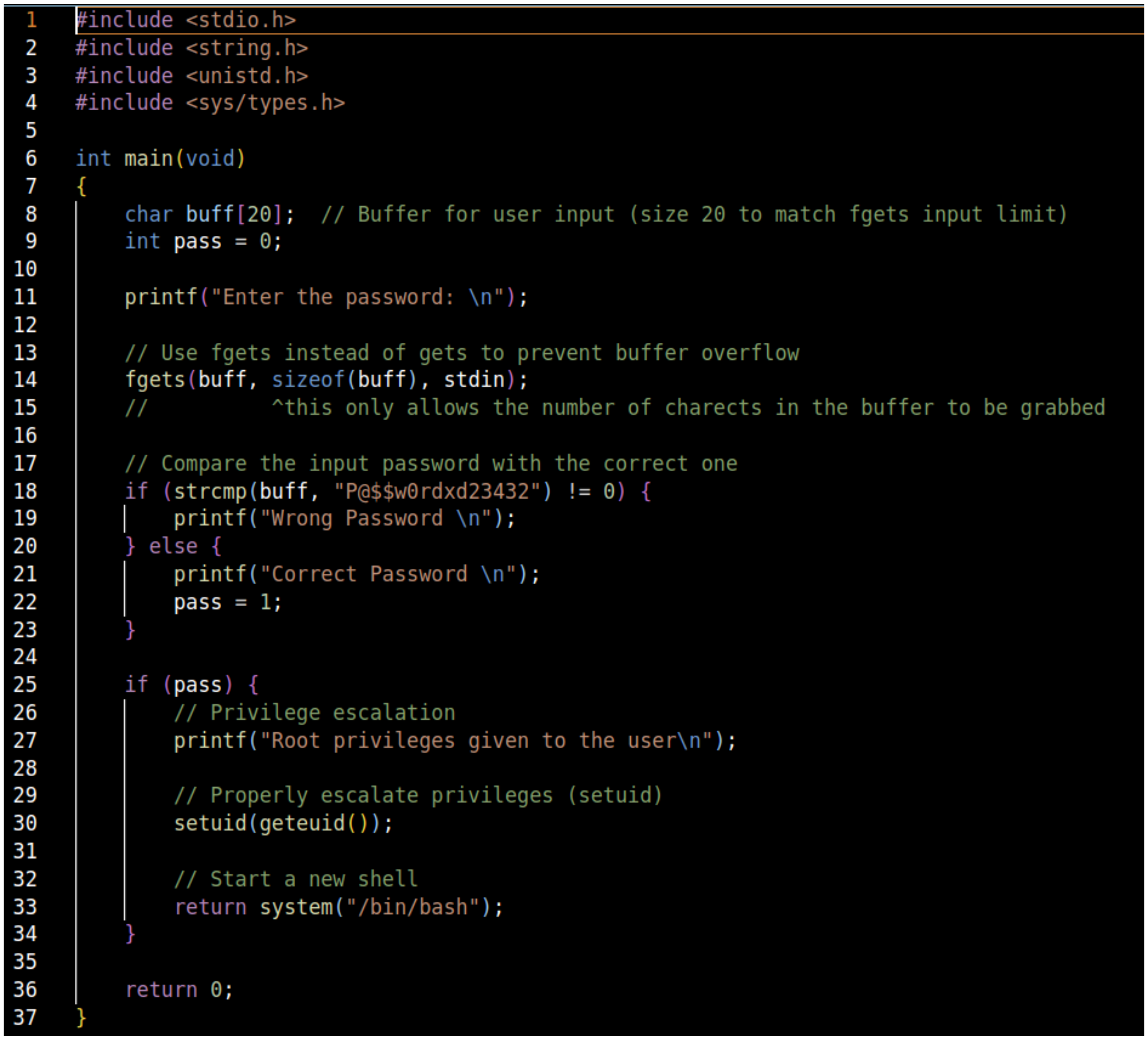
- Upon successful login, different things will appear depending on the account type
- Users can select a C file and click "Upload and Analyze"
- **The code submissions are sanitized, to prevent users from injecting files containing code in their names.**
- This would then show “Analysis complete! PDF downloaded.” below the Upload and Analyze button.



The report automatically downloads to your machine

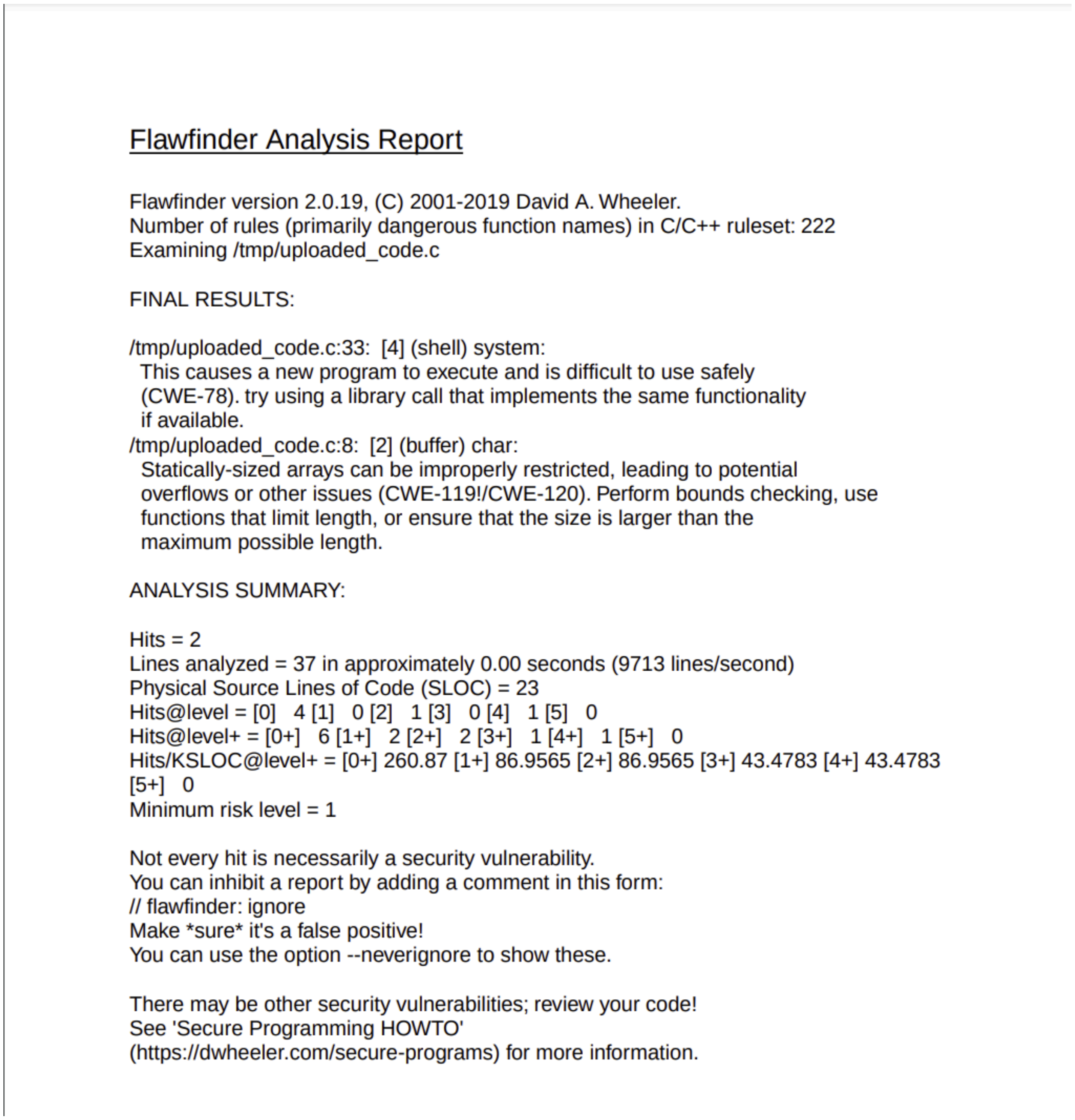


This was the code submitted to our tool



```c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>

int main(void)
{
    char buff[20];  // Buffer for user input (size 20 to match fgets input limit)
    int pass = 0;

    printf("Enter the password: \n");

    // Use fgets instead of gets to prevent buffer overflow
    fgets(buff, sizeof(buff), stdin);
    //          ^this only allows the number of charects in the buffer to be grabbed

    // Compare the input password with the correct one
    if (strcmp(buff, "P@$$w0rdxd23432") != 0) {
        printf("Wrong Password \n");
    } else {
        printf("Correct Password \n");
        pass = 1;
    }

    if (pass) {
        // Privilege escalation
        printf("Root privileges given to the user\n");

        // Properly escalate privileges (setuid)
        setuid(geteuid());

        // Start a new shell
        return system("/bin/bash");
    }

    return 0;
}
```

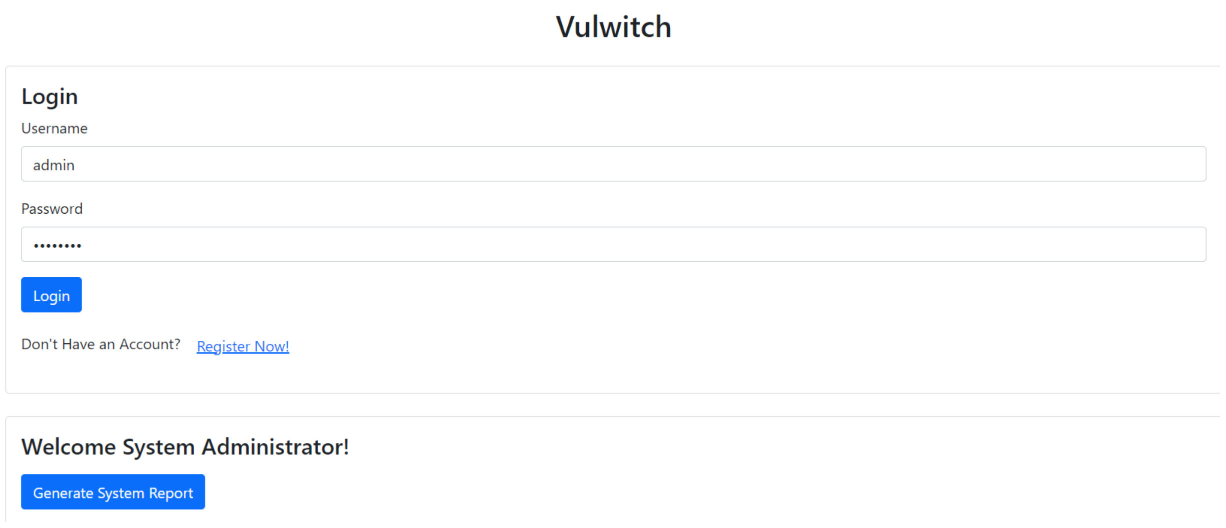
Here is the output from the scan



Flawfinder Analysis Report

```
Flawfinder version 2.0.19, (C) 2001-2019 David A. Wheeler.
Number of rules (primarily dangerous function names) in C/C++ ruleset: 222
Examining /tmp/uploaded_code.c

FINAL RESULTS:

/tmp/uploaded_code.c:33:  [4] (shell) system:
  This causes a new program to execute and is difficult to use safely
  (CWE-78). try using a library call that implements the same functionality
  if available.
/tmp/uploaded_code.c:8:  [2] (buffer) char:
  Statically-sized arrays can be improperly restricted, leading to potential
  overflows or other issues (CWE-119!/CWE-120). Perform bounds checking, use
  functions that limit length, or ensure that the size is larger than the
  maximum possible length.

ANALYSIS SUMMARY:

Hits = 2
Lines analyzed = 37 in approximately 0.00 seconds (9713 lines/second)
Physical Source Lines of Code (SLOC) = 23
Hits@level = [0]   4 [1]   0 [2]   1 [3]   0 [4]   1 [5]   0
Hits@level+ = [0+]   6 [1+]   2 [2+]   2 [3+]   1 [4+]   1 [5+]   0
Hits/KSLOC@level+ = [0+] 260.87 [1+] 86.9565 [2+] 86.9565 [3+] 43.4783 [4+] 43.4783
[5+]   0
Minimum risk level = 1

Not every hit is necessarily a security vulnerability.
You can inhibit a report by adding a comment in this form:
// flawfinder: ignore
Make *sure* it's a false positive!
You can use the option --neverignore to show these.

There may be other security vulnerabilities; review your code!
See 'Secure Programming HOWTO'
(https://dwheeler.com/secure-programs) for more information.
```

As a system administrator (there is only one; they cannot register), you have a different screen upon logging in:

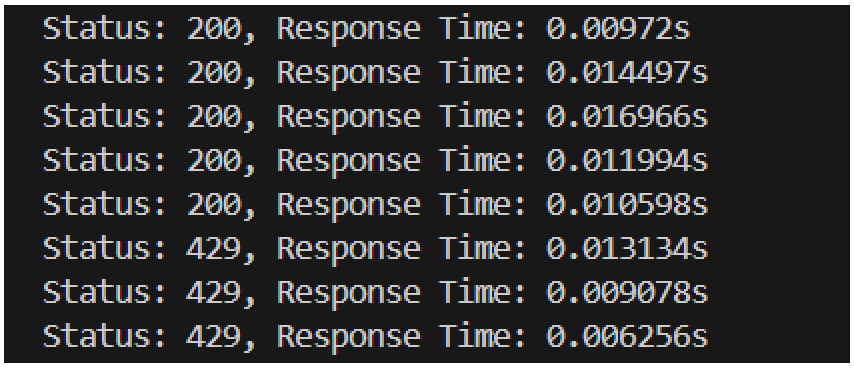


The “Generate System Report” button downloads a singular pdf of all scans to date. The system administrator can use this information in order to assess the overall security of the code being submitted

Milestone Report

Preventing DDoS via HTTP requests

As previously stated in the last deliverable, one way an attacker can take down a website is by spamming HTTP requests to a target website, flooding it to the point where the website would not be able to handle any legitimate requests. To prevent this, we implemented "rate limiting" to our website. Rate limiting is limiting the amount of requests to a website. Doing this will completely kill the ability to spam HTTP, as our website will begin to limit them.

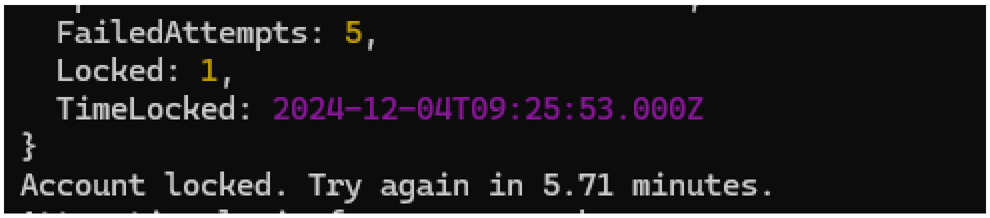


```
Status: 200, Response Time: 0.00972s
Status: 200, Response Time: 0.014497s
Status: 200, Response Time: 0.016966s
Status: 200, Response Time: 0.011994s
Status: 200, Response Time: 0.010598s
Status: 429, Response Time: 0.013134s
Status: 429, Response Time: 0.009078s
Status: 429, Response Time: 0.006256s
```

Here is the terminal output from our previous HTTP attack file. As you can see, we start by getting Status 200, which means that our request to the website has been successfully connected. However, shortly after, it begins to give Status 429, which indicates that the website has gotten too many requests, and will begin to block them.

Preventing Brute Force Password Attacks

Brute force password attacks is when the user tries to guess the password by brute force. This can cause issues in the website and can give a malicious user access to the website. In the last deliverable we mentioned a Selenium based brute force attack. In order to prevent this attack; we can timeout the user/ Limit the entries from any user. This way they cannot try as many passwords as possible and brute force guess the password. In this case we can timeout the user after a set number of tries (5); this would mean they cannot try more till the timeout is over. This would prevent the Brute Force password attacks on the website and make it more secure.



```
  FailedAttempts: 5,
  Locked: 1,
  TimeLocked: 2024-12-04T09:25:53.000Z
}
Account locked. Try again in 5.71 minutes.
```

When the user correctly enters their password the failed attempts counter is set back to 0. Users will have to wait 15 minutes after 5 failed password attempts this will substantially slow down any brute force of user passwords.

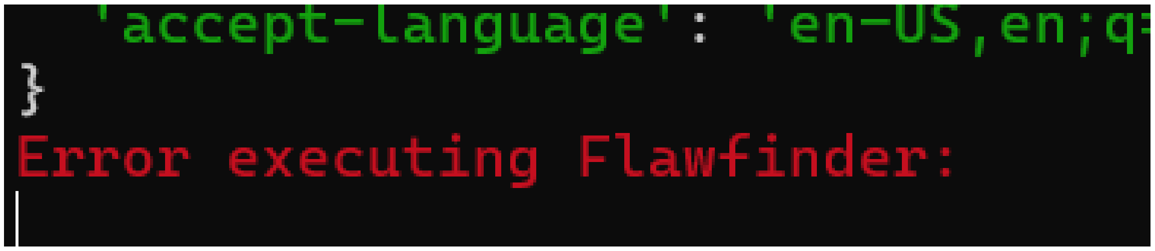
Preventing Command Injection through file submission

Previously mentioned in the documents there, a way the attacker can attack the website would be via injection attacks. It happens when dangerous and malicious inputs are provided by the user/attacker and then it get executed unintentionally by the program and gives access to unauthorized data. The attack example was

"test & del app.js & del database.js & del report.pdf & del package.json & del package-lock.json & del verifyToken.js & del README.md & del nothing.txt"

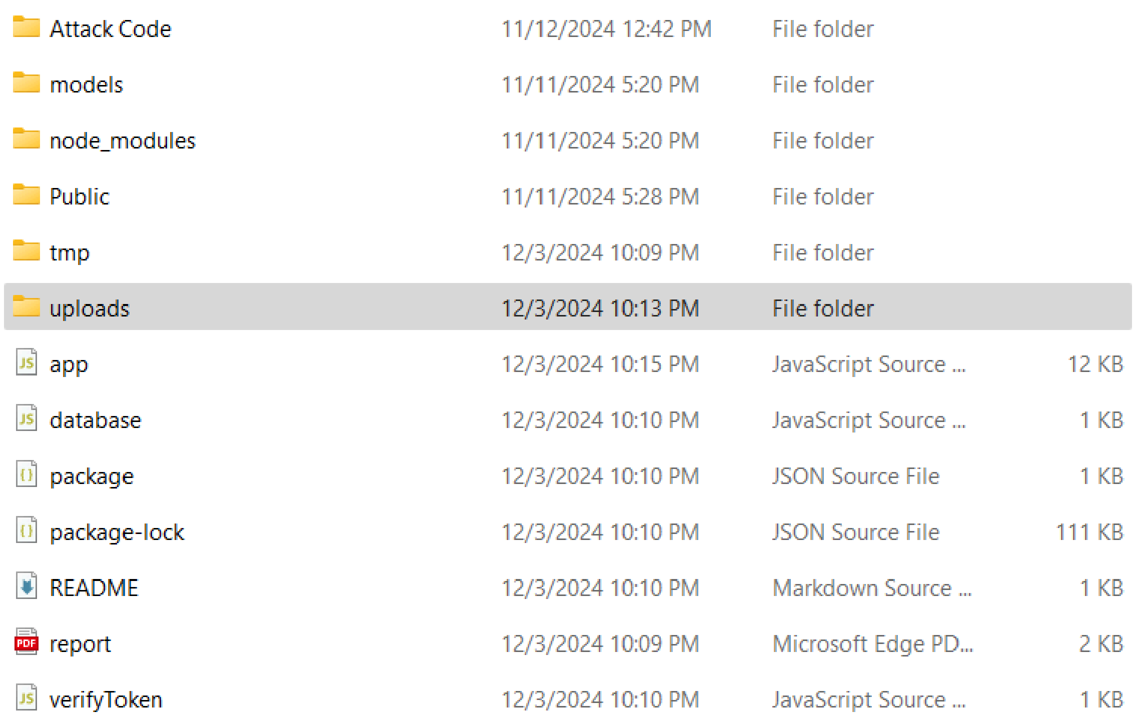
This input tries to execute a command that deletes important files. To stop this we can have different approaches. One of them is to Input Sanitization. We can remove dangerous characters/words from the input. This would stop the attack from happening. We can use libraries that allow to sanitize inputs and stop execution of dangerous inputs. To further implement security we can also verify and give validation to only some characters/words so that only the ones allowed will be bypassed and everything else would not be valid; allowing the stopping of Injection attacks.

Of these options, we chose input sanitization. When we go to run flawfinder via the command line, our code removes any special characters that would affect the command line (\, &, `, |). These characters would not belong in any typical C file, and are thus removed. Upon this occurring, the file is not found, due to the different name, and we get an error:

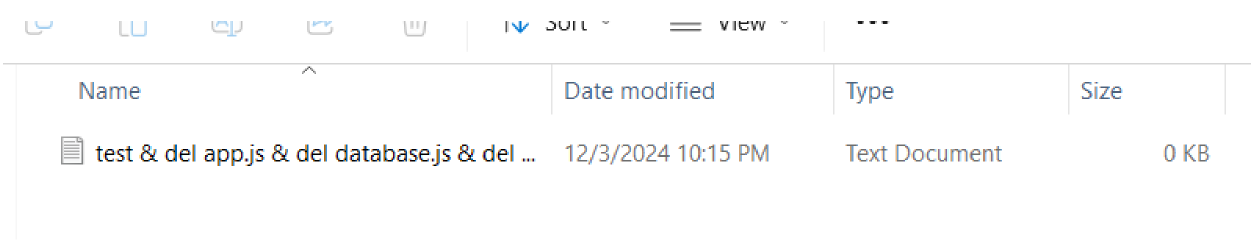


```
  'accept-language': 'en-US,en;q
}
Error executing Flawfinder:
```

Additionally, the uploaded file is not removed from the uploads directory (like a normal file usually is). This allows the system administrator to periodically check and see what files were not deleted (essentially flagging these files) as potential security threats. Should the system admin see a file that was not deleted, but appears to be valid, they can manually change the name and reupload the file.



Attack Code	11/12/2024 12:42 PM	File folder	
models	11/11/2024 5:20 PM	File folder	
node_modules	11/11/2024 5:20 PM	File folder	
Public	11/11/2024 5:28 PM	File folder	
tmp	12/3/2024 10:09 PM	File folder	
uploads	12/3/2024 10:13 PM	File folder	
app	12/3/2024 10:15 PM	JavaScript Source ...	12 KB
database	12/3/2024 10:10 PM	JavaScript Source ...	1 KB
package	12/3/2024 10:10 PM	JSON Source File	1 KB
package-lock	12/3/2024 10:10 PM	JSON Source File	111 KB
README	12/3/2024 10:10 PM	Markdown Source ...	1 KB
report	12/3/2024 10:09 PM	Microsoft Edge PD...	2 KB
verifyToken	12/3/2024 10:10 PM	JavaScript Source ...	1 KB



Name	Date modified	Type	Size
test & del app.js & del database.js & del ...	12/3/2024 10:15 PM	Text Document	0 KB

As can be seen above, the system files are not deleted (as they were before), and the malicious file is stored in uploads for review.