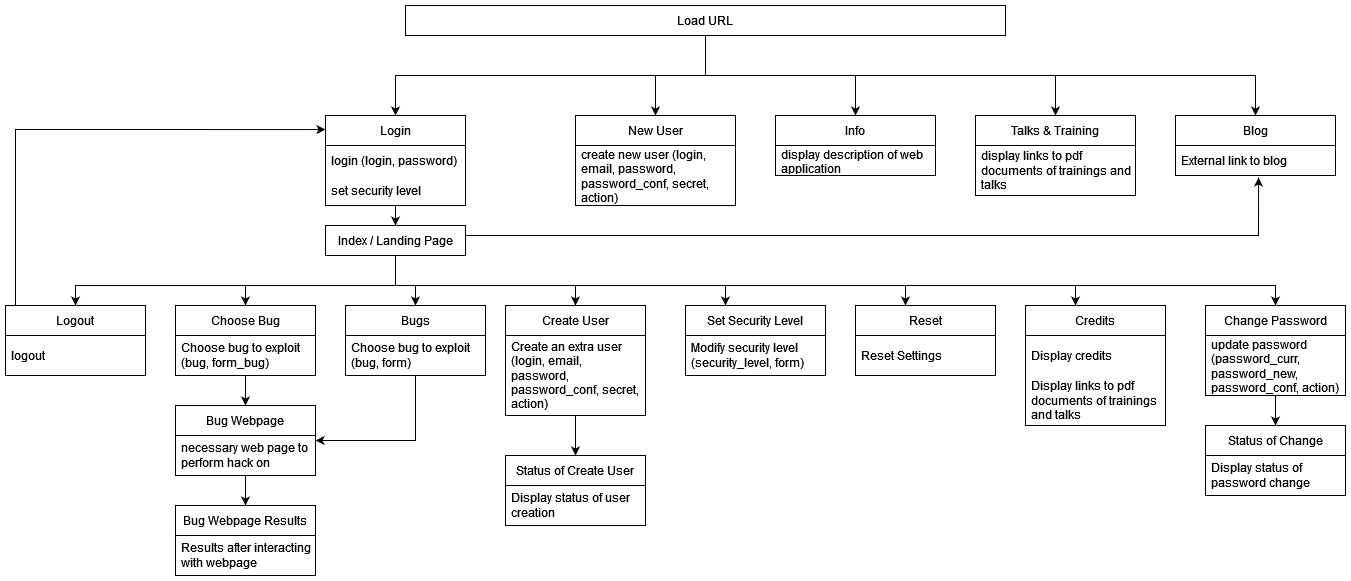
# Group Project 1

(Group 1)

**Task 1:** Gathering information on a target

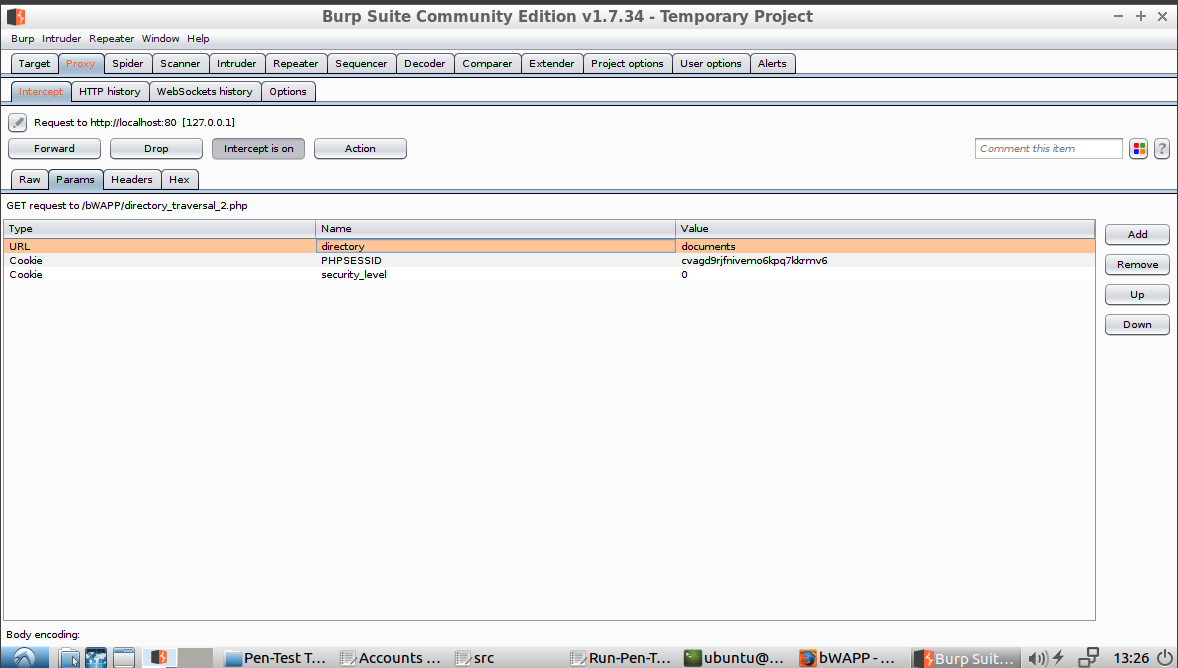
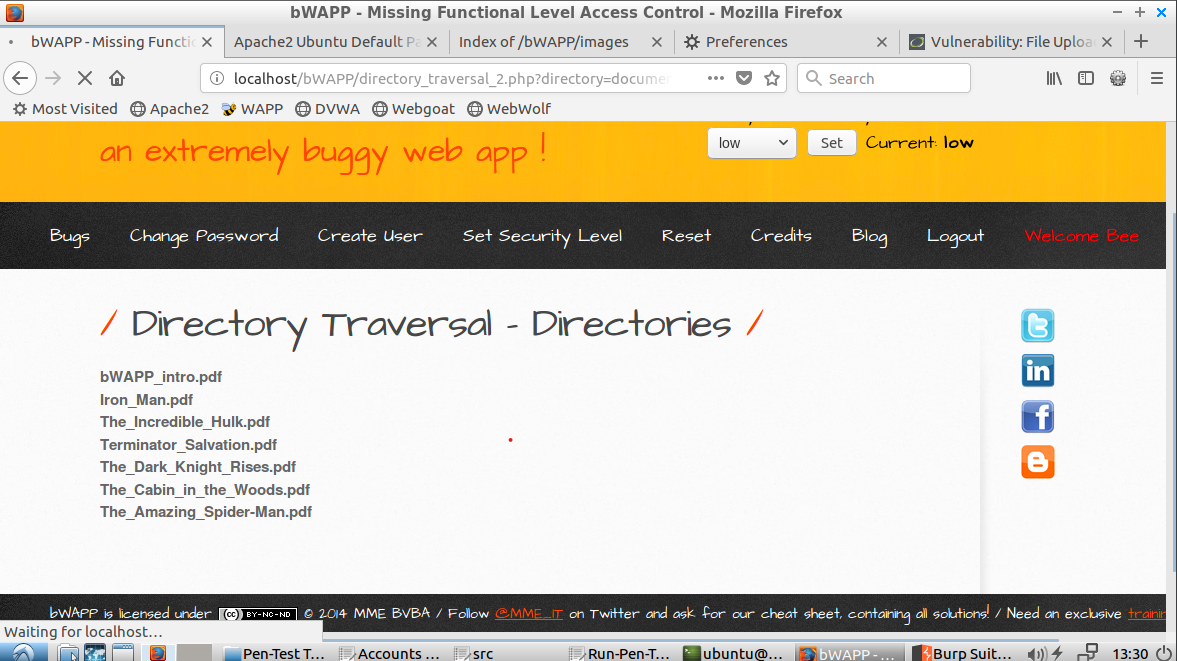
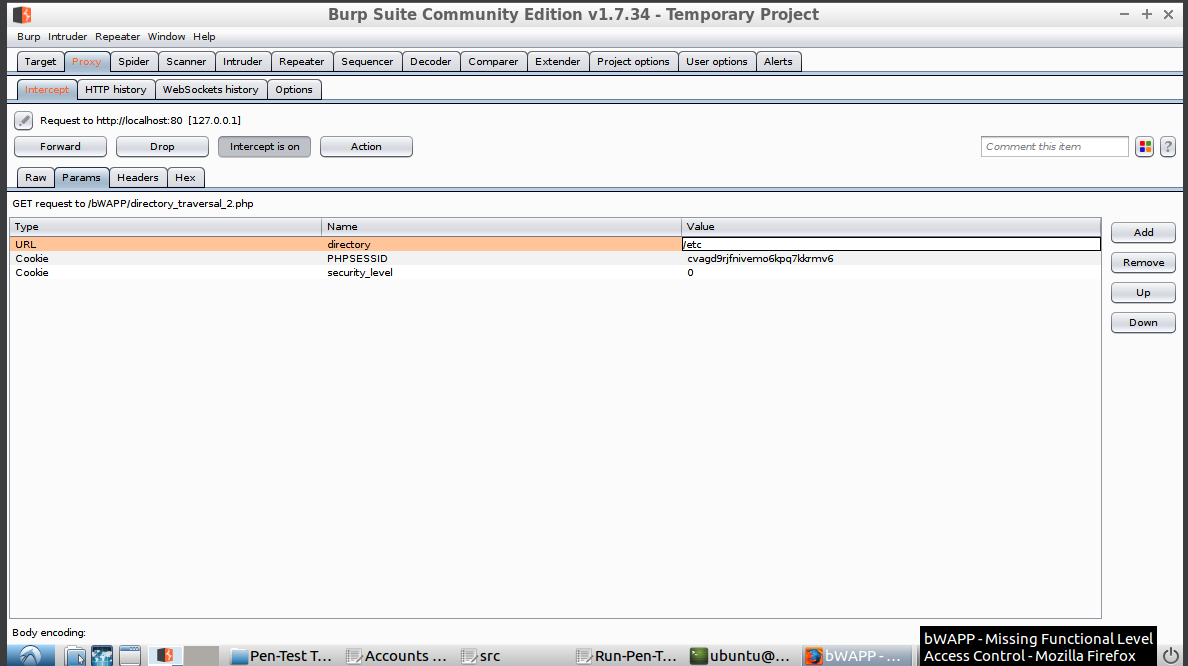
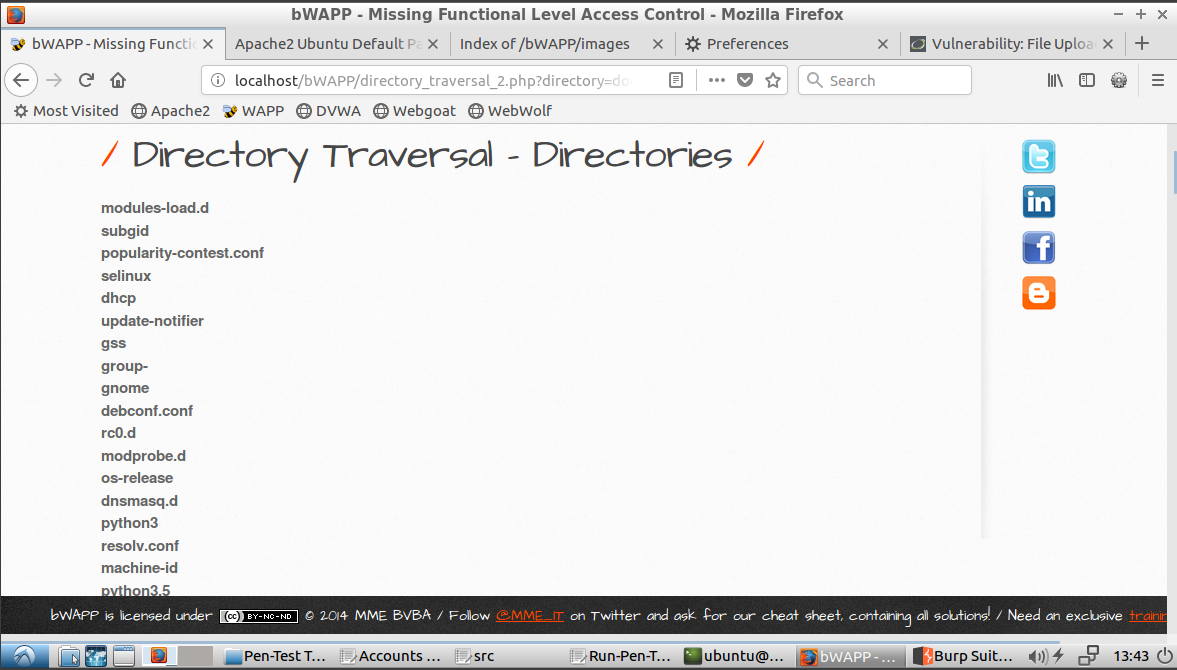
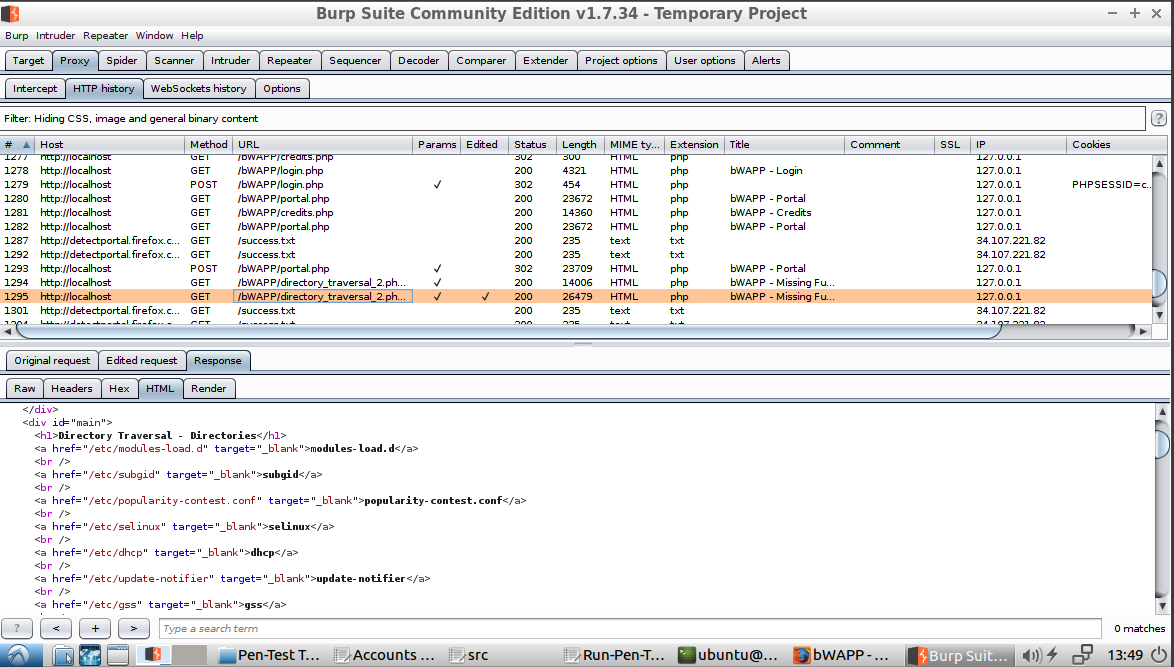
We will be exploring bWAPP web page to find vulnerabilities that can be exploited. First, we will explore the web application to gain a better understanding of the different pages and their relationships. To speed up the process, we began with an automated spider crawl of the application using Burp Suite. This exposed most of the initial pages and the requests that generate them. However, there were still more pages yet to be discovered by the spider that require manual crawling. This included the results of the pages after creating a user, clicking on the Blog link, and changing the current user password. This had to be done manually since the automated crawler provided by Burp Suite isn’t intelligent enough to inject the necessary data into the requests. Based on the information gathered in this process, we were able to create a diagram to visualize the application and its different pages, as shown in Figure 1 below.

*Figure 1*

Note that in this diagram, the data fields available to the user on each web page are included in the parenthesis. Also, after choosing a bug and submitting the bug form, there is a new page for each bug. Since there are over 150 different vulnerabilities to select from, including each unique we page in the diagram would have rendered it less effective due to the amount of information being conveyed. Therefore, we generalized those pages as Bug Webpage and Bug Webpage Results.

**Task 2.1:** Scenario 1 of bypassing a client-side control

The first instance of bypassing a client-side control was done on bWAPP, using Burp Suite as a tool. Intercept is turned on. The following steps will show how we were able to successfully bypass the client controls.

1. We selected a link from the Bug dropdown. Upon loading the URL for the page, Burp Suite intercepted the request (Figure 2.1). You can see that the request is a GET to /bWAPP/directory\_traversal\_2.php. It is passing a URL parameter and two Cookies.  
    Mario Amaro  
   Sushruti Bansod  
   Joshua Fishman  
   Nick Forleo  
   *Figure 2.1*
2. To see the default behavoir of the request, we sent the request to the server without any modifications (Figure 2.2). It displays the content a directory on the web page.   
     
   *Figure 2.2*
3. Based on the webpage, it looks like the client is defaulting the directory of files to be displayed. It doesn’t allow the user to change the contents through the web page interface. This means the client is attempting to control what the user sees. However, in step 1 we saw that the directory is being passed as a URL parameter in the GET request. We can attempt to bypass this directory and enter our own by modifying the request, as shown in Figure 2.3 below. We will enter the /etc directory. There are many important files in that directory on linux machines.  
     
   *Figure 2.3*
4. You can see now that the web page is displaying the contents of the /etc directory (Figure 2.4). We were able to bypass the client-control of what directory is being displayed. You can also see that the response to modified request include this information as well (Figure 2.5).  
     
   *Figure 2.4 - Content displayed on web page includes /etc directory.*  
     
     
   *Figure 2.5 - Response of modified request with /etc directory content*

**Task 2.2:** Scenario 2 of bypassing a client-side control

The target for this scenario was the Authentication Flaws > Authentication Bypasses page of the WebGoat web application using WebScarab as the penetrating test tool.

1. The challenge was to verify an account so that the user can change the password, but a secondary verification method was needed, using questions, but the answers were ‘unknown’. The task was to find a way to bypass the questions so that the user can verify the account. User ID was previously verified. A screenshot of a computer

   Description automatically generated

*Figure 2.6 - Initial Authentication Bypasses page loaded in firefox*

1. Random answers were put in the form fields, and the application denied the answersA screenshot of a computer

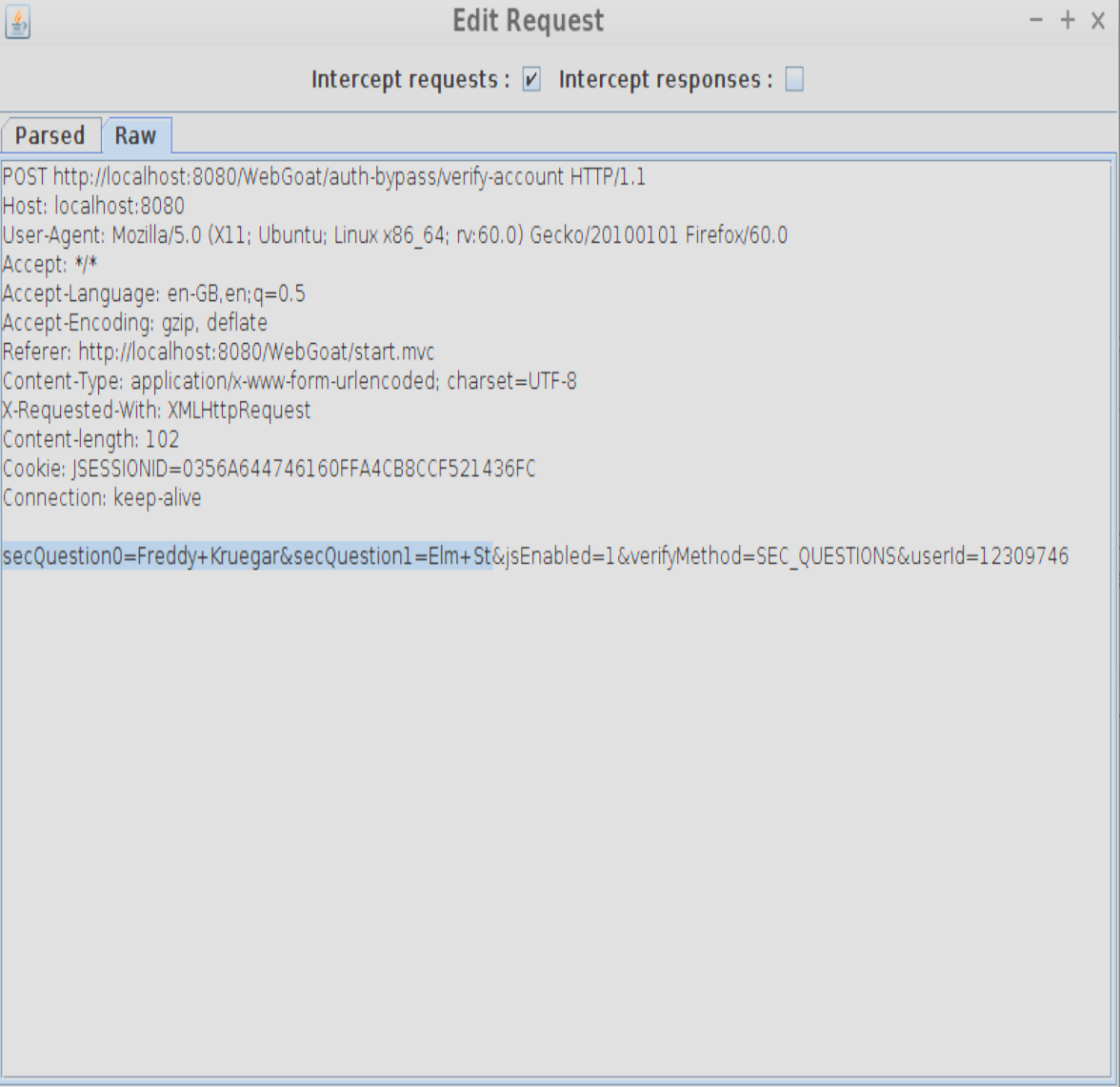
   Description automatically generated

*Figure 2.7 - Random security question answers added to the form shown in Figure 2.6*

1. The proxy used for WebScarab was port 8008 with WebGoat using port 8080. We initialized intercept request to pull the data before a response can be sent. A screenshot of a computer

   Description automatically generated

*Figure 2.8 - WebScarab Intercept configuration used to intercept POST requests*

1. We then submitted the same data in the answers field on the application. In WebScarab we were able to identify the data that requires verification through the SecQuestion script. So, the form would be looking for the answers in SeeQuestion0 and SeeQuestion1, but the only requirement is that SecQuestion and userid is verified. 

*Figure 2.9 - Unaltered intercepted POST request in WebScarab*

1. We changed the secQuestion to secQuestionA and secQuestionB, looking for any secQuestion that is ‘null’. If the value was not set or identified, then the form should allow to pass with any value since its nullified. A screenshot of a computer

   Description automatically generated*Figure 2.10 - Altered intercepted POST request in WebScarab*
2. We successfully bypassed the request because the secQuestion script was successfully verified with an acceptable value. A screenshot of a computer screen

   Description automatically generated*Figure 2.11 - Successful authentication bypass in WebGoat web application*

**Task 3:** Based on the found vulnerable issues, what should you do to prevent attackers from hacking your web application?

1. Protection/prevention for Scenario 1:  
    This exploit can be prevented or protected against by not including the directory in the request and having the contents of a specified directory returned directly from the server. This way there is no way for a user to modify the request to change the returned content. Another layer of security could be to include as–needed permissions for each file or at the lowest level directory necessary. For example, if the path is /documents/school/psu/insc561 and we only want to show the contents of insc561, we can open insc561, but only allow a specific user, such as admin, to access psu and above. This means the client won’t have access to those files.
2. Protection/prevention for Scenario 2:  
    To protect against the vulnerability of being able to bypass the secondary verification via manipulation of the security question responses, its crucial to implement robust server-side validation and encryption for all user inputs, especially for security questions. In this scenario, if the security questions did not exist in the request, the server did not fail the validation of the POST request (HTTP 400 response) but instead continued assuming to validate the request (HTTP 200 response) assuming that any missing security questions were correct. This is a broken authentication case that should be fixed simply by checking if the answers to the security questions exist and then checking that they are correct before continuing on to allow the user to reset their password.

This can be complemented with multi-factor authentication (MFA) to add an additional layer of security beyond just knowledge-based checks. Account lockout mechanisms can be used to deter brute force attempts. Additionally, security audits and penetration testing can help identify potential vulnerabilities. Moreover, logging and monitoring authentication attempts will aid in the early detection of suspicious activities, allowing for quick response to potential security breaches.

**Team peer-review table**

|  |  |
| --- | --- |
| Name | Contributing Efforts in this project (0 ~ 100%) |
| Nick Forleo | 100% |
| Mario Amaro | 100% |
| Sush Bansod | 100% |
| Joshua Fishman | 100% |