**Introduction to Cyber Security  
CS 329**

Project Report  
Term Project 1  
bWAPP Vulnerabilities

A picture containing emblem, symbol, trademark, badge

Description automatically generated

Group Members:

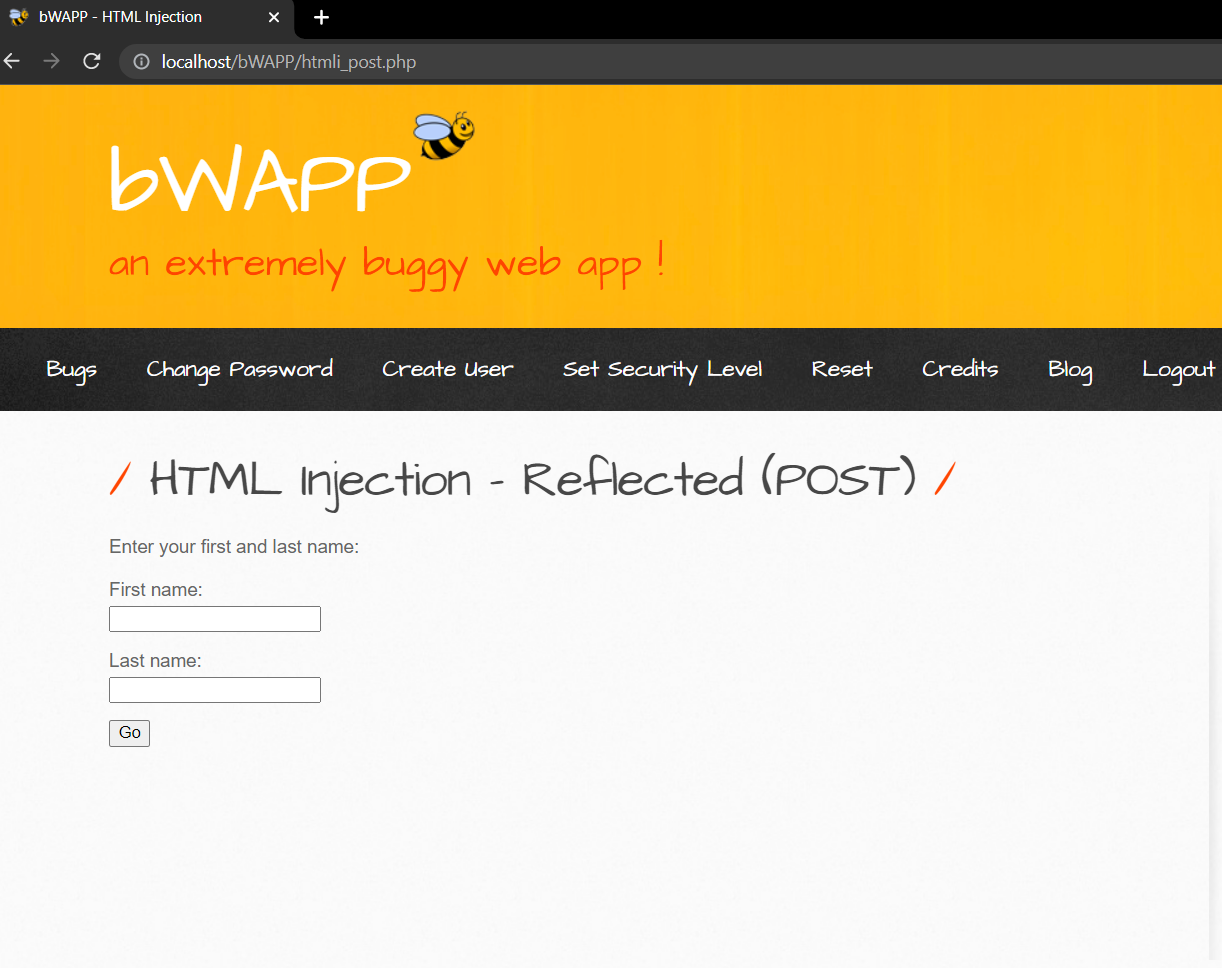
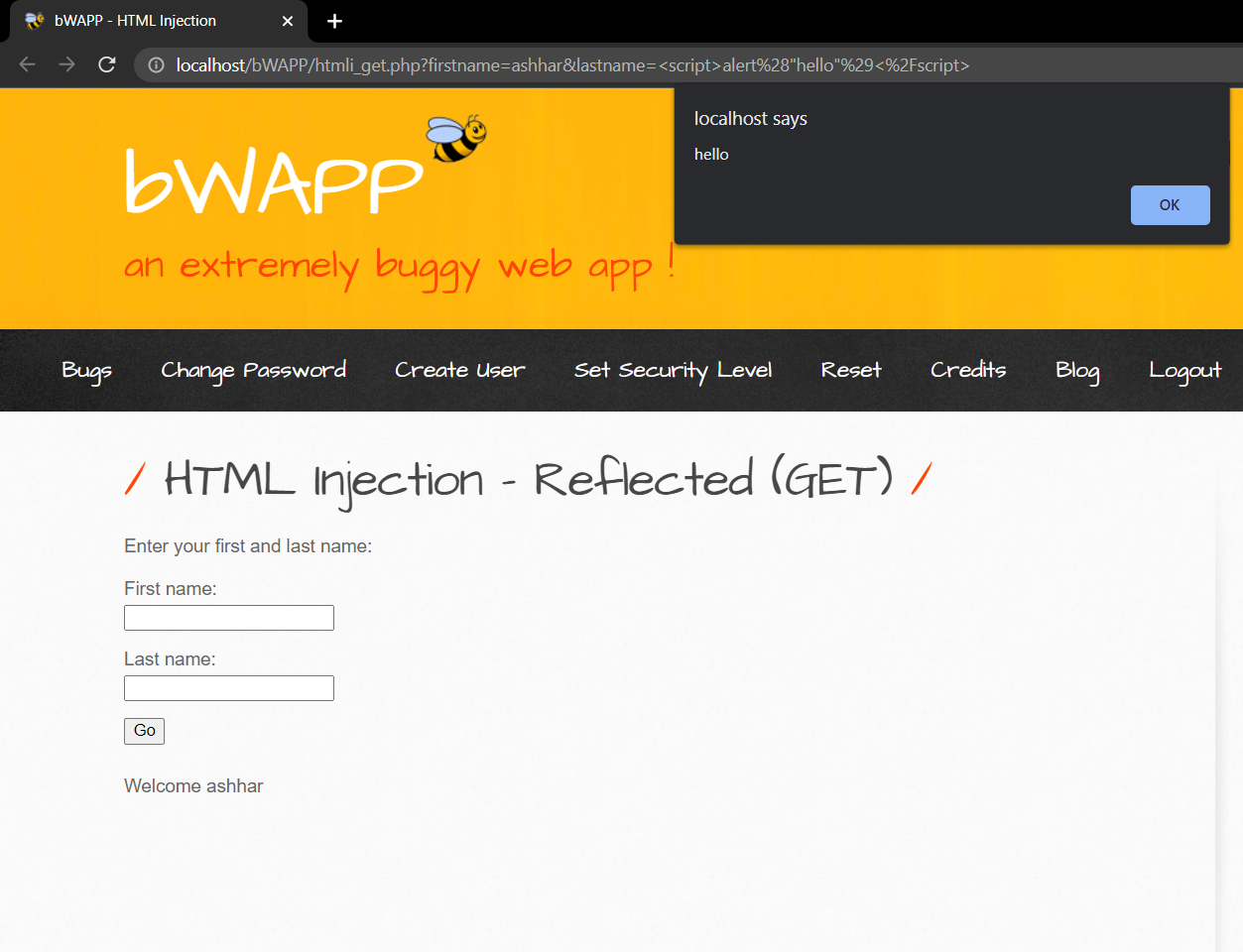
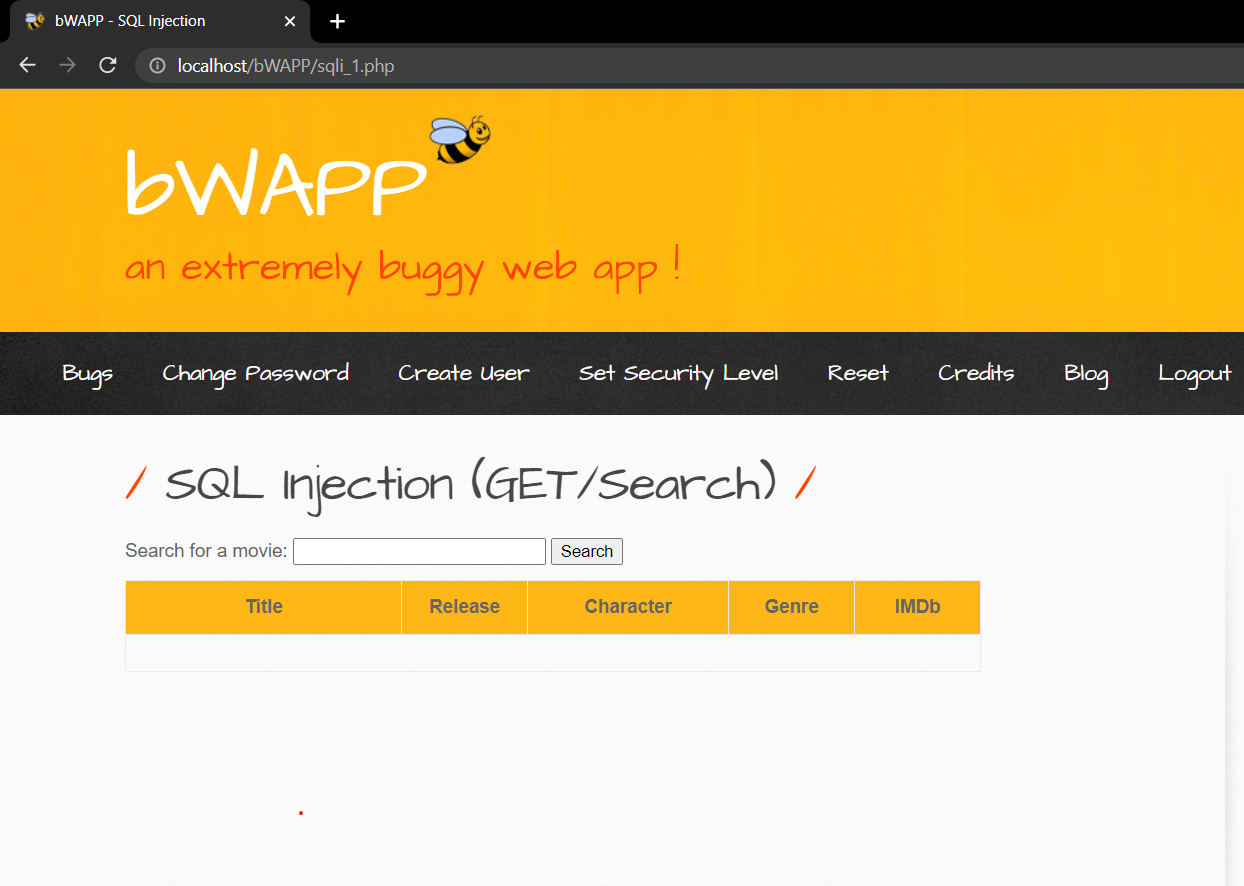
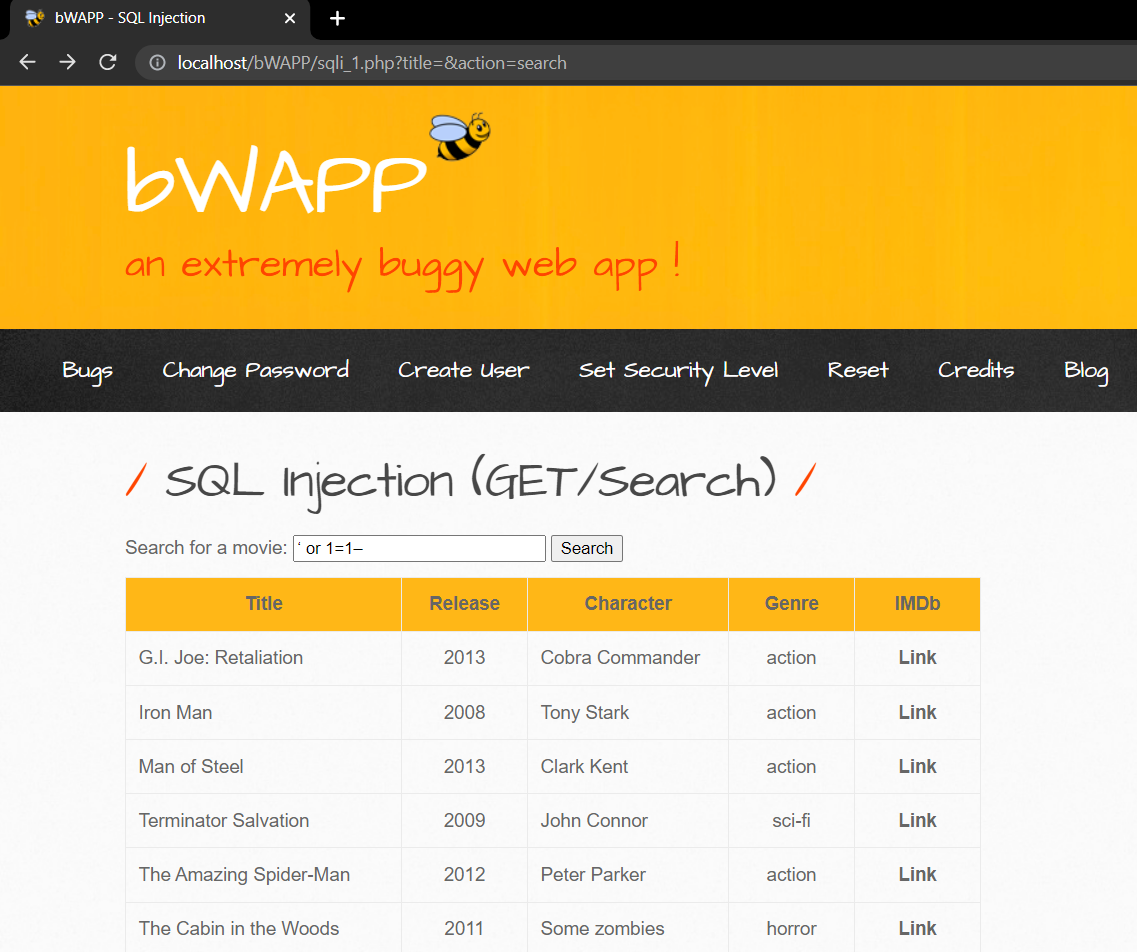
1. Syed Muhammad Ashhar Shah, 2020478
2. Khizar Ali Shah, 2020196

Injection

It is a vulnerability which allows an attacker to inject malicious code such as SQL and HTML snippets in different fields of a web page, they can be used to compromise a regular users system. Furthermore, injection of scripts can be used to steal user data such as cookies. An attacker can also use the injection to modify a user’s data or take complete control over an application hence can use it to carry out malicious activities and target users of the web site.

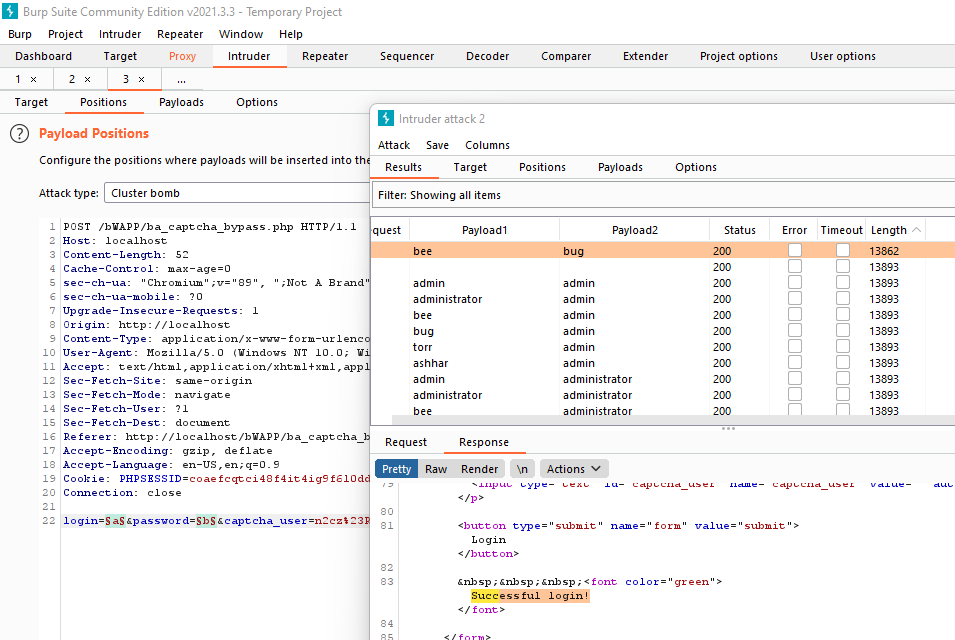
1. **HTML Injection (Reflected GET)**
   * Route: We need to select the bug from the drop-down list located at the top right of the web page. From there is a web page with two fields, one for the first name and second for the last name. Here we can inject HTML code into the input fields.
   * A screenshot of a web application

     Description automatically generated with low confidenceScreenshot Before Action
   * A screenshot of a computer

     Description automatically generated with medium confidenceScreenshot After Action
   * Observation: From the images above, we see that we can inject HTML code snippets into the fields of the webpage and when executed they are processed as HTML tags rather than regular text. This works for low level security level and not for medium and high level. In this example we have injected an h1 heading html tag which is successfully injected into the body of the web page and can be viewed. This vulnerability can be exploited by an attacker to inject malicious scripts which can run on a user’s system when he views a certain webpage and compromise his system.
   * Preventions:
     + 1. Validate all input data which is entered in a input field to ensure structure integrity.
       2. Any output that is sent back to the user should be properly encoded so that an injected script or html snippet is unable to run on the user’s system.
       3. Save the users input to the database in a text only format so that when it is re-rendered it is done so as a text rather than a html tag.
2. **HTML Injection (Reflected POST)**
   * Route: We need to select the bug from the drop-down list located at the top right of the web page. From there a web page with two fields, one for the first name and second for the last name. Here we can inject HTML code into the input fields. Furthermore we can also see create a link and directly imbed the malicious piece of code or script directly into the link of the web application to formulate the attack, example: <http://localhost/bWAPP/htmli_get.php?firstname=ashhar&lastname=%3Cscript%3Ealert%28%22hello%22%29%3C%2Fscript%3E>
   * Screenshot Before Action
   * Screenshot After Action
   * Observation: We see that when we provide an input in the fields, it is also sent to the URL also not just the body of the HTML document. Hence when the URL is processed to load the web page the embedded HTML tag is ran on the browser of a victim. This vulnerability can be used to embed malicious script in the URL of a web page that may look authentic but does something else that an attacker may want the scripty to do such as steal a user’s passwords or account information. The security level for this demonstration is selected to be low as this does not run on medium and high ones.
   * Preventions:
     1. Validate all input data which is entered in a input field to ensure structure integrity.
     2. Any output that is sent back to the user should be properly encoded so that an injected script or html snippet is unable to run on the user’s system.
     3. Use security tokens to ensure that each POST request is a valid request before it is processed so that a malicious script cannot be injected.
3. **SQL Injection (GET / Search)**
   * Route: We need to select the bug named SQL Injection (GET/Search) from the drop-down list located at the top right of the web page. From there a web page with an input field for a movie is displayed. In the input field we see that we can inject SQL queries which when run gives us access to data in the database which we should not have.
   * Screenshot Before Action
   * Screenshot After Action
   * Observation: We see that we can inject SQL code into the input fields of the web site which can in turn give us access to restricted data which we should not be able to view. This is a high-level vulnerability as an attacker can steal sensitive data from a web application's database as well as modify it and can take complete control of a web application server and use it to execute commands causing a complete system compromise.

Broken Authentication

When an application's authentication procedures are not appropriately implemented, it creates a sort of security vulnerability known as broken authentication that enables an attacker to get around authentication and access restricted sections of the application or system. Insecure password reset features, weak passwords, session management difficulties, credential stuffing, and other techniques can all be used to take advantage of this vulnerability. A successful attack can have catastrophic effects, including breach of data, system compromise, and damage to the application’s reputation.

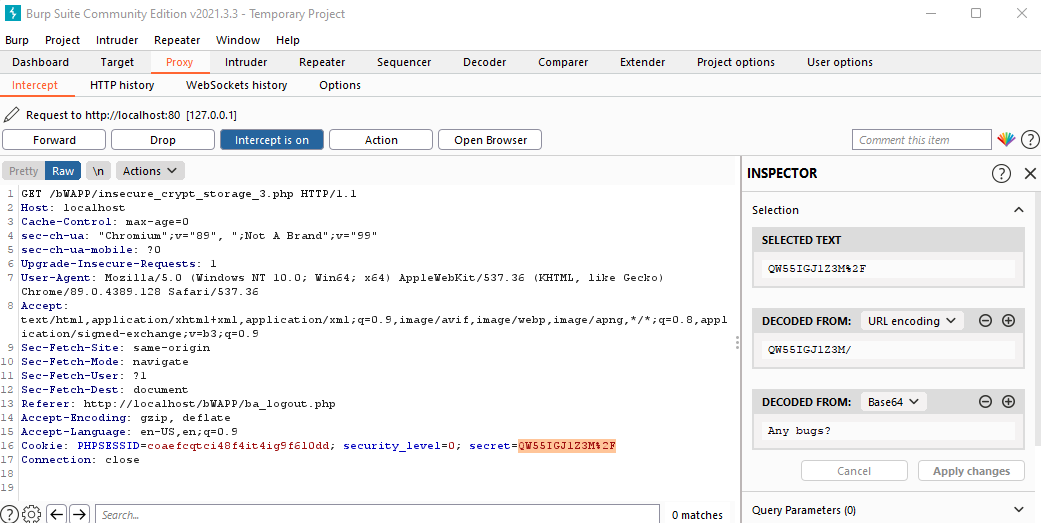
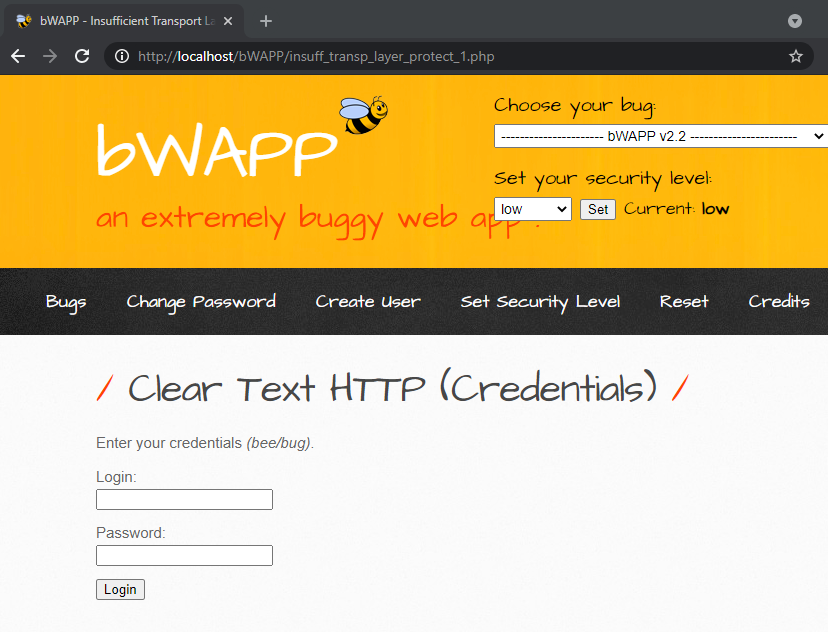
1. **CAPTCHA Bypassing**
   * Route: We need to select the bug from the drop-down list located at the top right of the web page. From there a web page is loaded with three fields, namely login, password, and CAPTCHA. Here we check when we enter a credentials and valid CAPTCHA, we can login, when we enter invalid credentials and valid CAPTCHA, we cannot login and if both the credentials and CAPTCHA are invalid, we again cannot login. However, we see that the only way to bypass the authentication is through entering a valid CAPTCHA and valid credentials. Suppose if we do not have the credentials but we can use the CAPTCHA to bypass the authentication using brute force attack as the CAPTCHA does not change on an unsuccessful attempt.
   * Screenshot Before Action
   * Screenshot After Action
   * Observation: We see that using an application like burp suite we can carry out a brute force attack by using different combinations of logins and passwords on the application successfully bypassing the CAPTCHA by checking the response packets sent back to the application and searching for a successful login in the body of the packet. This means that there is a vulnerability in the CAPTCHA authentication system of the application that can be exploited to gain unrestricted access to a web application by getting the hold of an administrators account using attack techniques such as brute force attack and executing commands on the account owner’s behalf remaining undetected in the system while simultaneously targeting the users of the application.
   * Preventions:
     1. Make sure that a user’s account has a strong password upon initial signup.
     2. Use of multi factor authentication, 2FA so that only a login name and password is not enough.
     3. Monitor number of failed attempts and place a restriction.
     4. Use of secure communication channels so that the packets which are sent and received are encrypted properly.
2. **Logout Management**
   * Route: We need to select the bug from the drop-down list located at the top right of the web page. From there a web page is loaded that gives us a href text link that allows us to use the logout function of the application. When we click the link, a popup is displayed which asks us to confirm our logout and upon clicking OK it takes us back to the login page. However, when we click on the button of the client browser that allows us to go back to the previous web page we observe that we can in fact again go back to the logged in session and navigate around the web application which we should not be able to do and hence the login functionality of this web application is broken. This only works for the low security level. On the medium and high-level security when we click the back button we are still prompted to the login page and cannot go back to the session.
   * Screenshot Before Action
   * A screenshot of a computer

     Description automatically generatedScreenshot After Action
   * Observation: We observe that we can go back to a logged-out session by simply clicking the back button of the client browser which reloads the previous page, and we can navigate as a logged in user which indicates improper logout management. Furthermore, if we open another separate web page hence two web pages of the application and login to both, if we logout from one of the tabs we should be able to logout of the next one also however it is not the case which further adds to the improper logout management function. This can be exploited by attackers to access sensitive data or functionality, such as financial transactions or personal information, by hijacking a user's active session.
   * Preventions:
     1. Implement a brief timeout period so that inactive users will be automatically logged out after a predetermined amount of time.
     2. Make sure you delete any session tokens or cookies linked with a user's session after they log out.
     3. To prevent client-side scripts from accessing or intercepting session data, use HTTP-only and secure cookies to store session information.

Sensitive Data Exposure

A particular kind of security flaw in online applications referred to as sensitive data exposure vulnerability lets unauthorized individuals access confidential information. This may happen for several reasons, including unsecured data storage, inadequate encryption, or poor processing of user input. Passwords, bank information, personal identifying data, and other sensitive information can all be compromised, which has been known to result in identity theft, financial loss, reputational harm, and legal consequences.

1. **Base64 Encoding (Secret)**
   * Route: We need to select the bug from the drop-down list located at the top right of the web page. From there a web page is loaded which states *“Your secret has been stored as an encrypted cookie!”*. We will now open burp suite to capture the request that is sent so that we can figure out a way to access the encrypted cookie. We see from the request packet we have a cookie in the header which contains an encrypted secret. When we copy the secret, we go to the decoder of the burp suite application. We will then use the base64 decoding functionality of the burp suite application to decode the secret that was present inside the cookie header.
   * Screenshot Before Action
   * A screenshot of a computer

     Description automatically generated with medium confidenceScreenshot After Action
   * Observation: We see that we can successfully decode the secret by using the base64 decode functionality of the burp suite application by copying the secret from the cookie header of the request packet and upon successfully decoding it prints “Any bugs?” which shows that the encryption used for the cookie secret was weak. This shows that anyone with access to the encrypted data may quickly and easily decode it to get the original data. Furthermore, Attackers can utilize Base64 encoding to conceal harmful data, sneak around input validation checks, and carry out various kinds of attacks on application users.
   * Preventions:
     1. Use a strong encryption algorithm other than Base64 to make sure that the values cannot be decrypted easily.
     2. Input validation should be done on both the encoded and decoded data.
     3. When sending sensitive data over the internet, use a secure transport layer.
     4. To guarantee that only authorized people have access to sensitive data, implement access control.
2. **Clear Text HTTP (Credentials)**
   * **Route****:** We need to select the bug from the drop-down list located at the top right of the web page. From there a web page is loaded which displays two input fields, one for login and one for password. Now we will use a packet tracer application like burp suite to check if the login and password data that is sent back to the sever for the authentication process is properly encrypted or not.
   * Screenshot Before Action
   * A screenshot of a computer

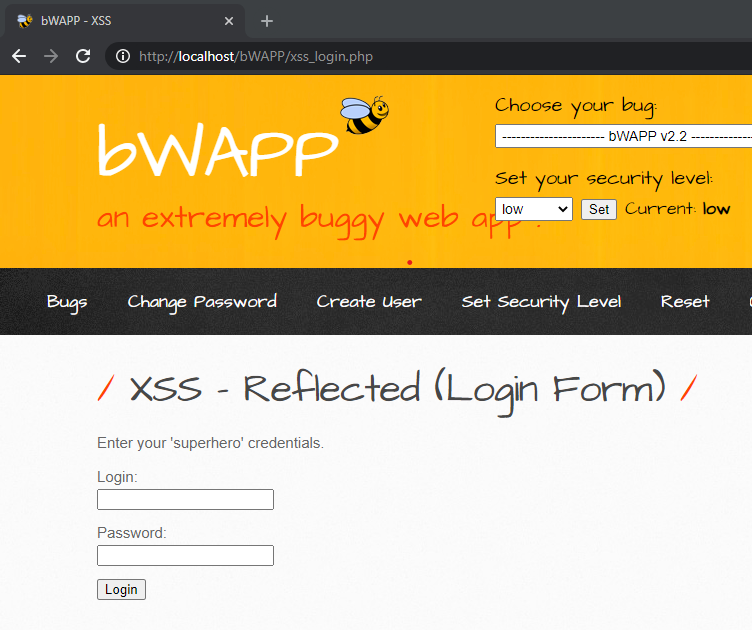
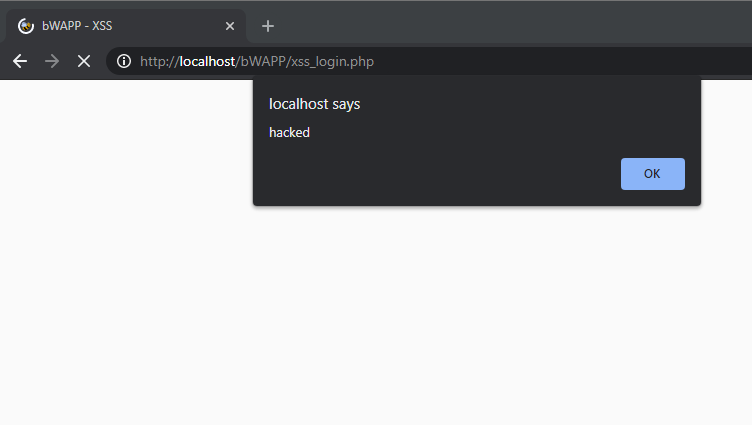
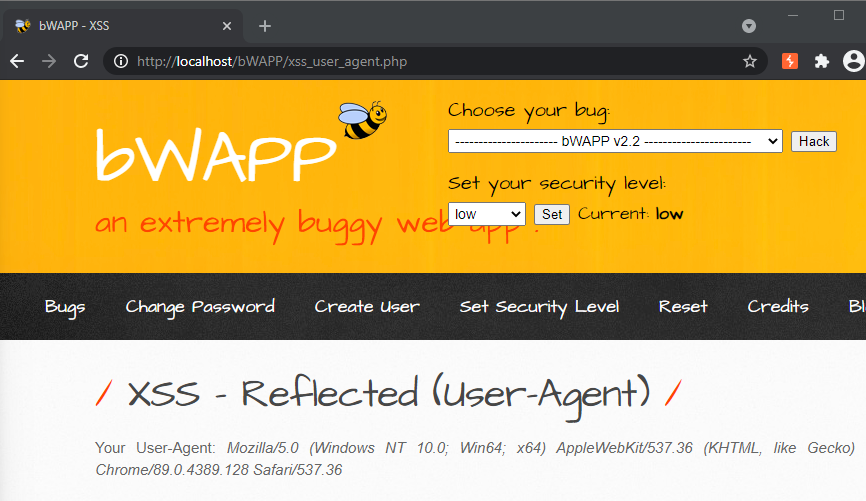
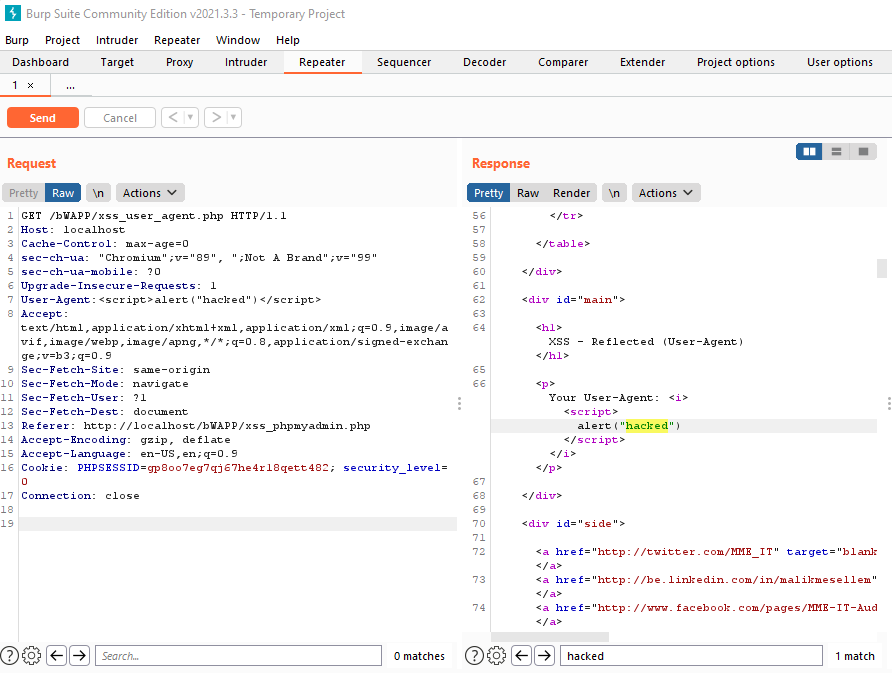
     Description automatically generatedScreenshot After Action
   * Observation: We see that the login and password inputs that are sent back to the server in the request packet are in fact in the form of clean text and can be read by the intruder in the system that is carrying out packet sniffing. This is a big issue in the security of the web application as confidential data may be intercepted and seen by attackers while it's being sent to the server or back from the server, which might have disastrous results like account breach and identity theft.
   * Preventions:
     1. To encrypt all data sent between the client and the server, always utilize HTTPS encryption.
     2. Protect user credentials from exposure by using secure password storage techniques like hashing and salting.
     3. Using two-factor authentication to increase the security of user accounts.

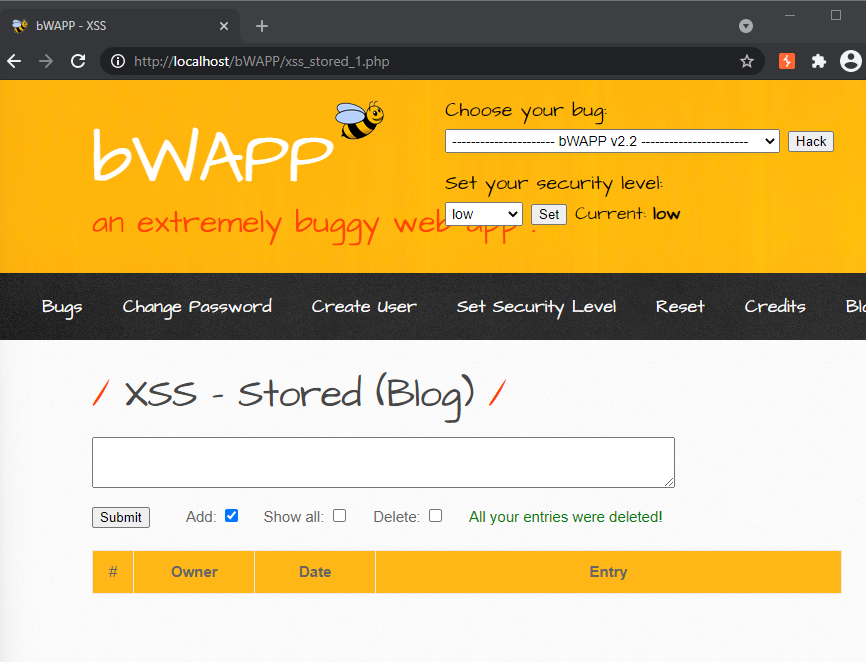
Cross-Site Scripting (XSS)

A security flaw known as cross-site scripting (XSS) allows an attacker to insert malicious code into a web page that is being viewed by other users. The injected malware may be used to steal sensitive data, including cookies, session tokens, and login passwords.

There are two types of cross site scripting attacks:

* Reflected XSS: In this kind of attack, the URL or content that is sent and returned to the user by the server contains harmful code. The browser runs the malicious code when the user accesses the compromised web page, enabling the attacker to obtain access to confidential information or take actions on the user's behalf.
* Stored XSS: In this kind of attack, the malicious code is kept on the server, usually in a database or other storage space, and executed every single time a user views the compromised web page. Because it may impact several users and continue long after the attacker's access has been revoked, this kind of attack is more harmful once carried out successfully by the attacker.

1. **XSS – Reflected (Login Form)**
   * Route: We need to select the bug from the drop-down list located at the top right of the web page. From there a web page is loaded which displays two input fields, one for login and one for password. Here we are prompted to enter your superhero credentials. If we enter invalid credentials the login fails and prints invalid credentials. However if we input a special character ‘ we are prompted with a message regarding an SQL syntax error which means we can perform SQL injection in the input fields which show that there is a vulnerability in the input fields. We can use the SQL error to carry out a XSS attack.
   * Screenshot Before Attack
   * Screenshot After Attack
   * Observation: We see that we can bypass login form by ending the SQL query with a statement that will always be true, for example ‘1=1; once we enter this in the input field, we can bypass the login form and then inject the malicious code from there. In this demonstration I have executed the following piece of code in the login input field ‘1=1; <script>alert(“hacked”)</script> which has executed successfully on the clients browser and this means we can use the SQL injection vulnerability to carry out cross site scripting attack and gain access to a user’s confidential information.
   * Preventions:
     1. Using input validation techniques such as input filtering, data type validation, and length and range checking.
     2. Prior to the data being shown to the user make sure to cleanse it to make sure it is free of any harmful code that could compromise the system.
2. **XSS – Reflected (User – Agent)**
   * **Route:** We need to select the bug from the drop-down list located at the top right of the web page. From there a web page is loaded which displays the user agent header that is sent in the request packet when we load the page. It contains data about the client system like the OS version and the browser type and version. If we use a packet tracer like burp suite to read the contents of the packet, we see that we can change the values of the user agent header in the request packet, we also observe that we can edit the header to add a script and send it back to the server which is processed and is a XSS reflected vulnerability.
   * Screenshot Before Action
   * A screenshot of a computer

     Description automatically generated with medium confidenceScreenshot After Action
   * Observation: We see that we can successfully inject malicious code into the request packet header, and it will run on the clients browse which can be verified by viewing the response packet, This shows that XSS can be carried out by an attacker who is sniffing for packets in the network and can compromise systems to gain access to important confidential information like credit card numbers and account information’s.
   * Preventions:
     1. Encrypt all data sent between the client and server using HTTPS. Data is transferred safely and cannot be intercepted by or read by intruders thanks to HTTPS encryption.
     2. To stop XSS attacks, use anti-XSS frameworks or libraries that automatically encrypt or sanitize input and output data.
     3. Put in place a Content Security Policy (CSP) that enables web engineers to stop malicious code from being executed on a website.
3. **XSS – Stored (Cookies)**
   * Route: We need to select the bug from the drop-down list located at the top right of the web page. From there a web page is loaded which displays a input field where we can enter text which is stored in the database as a blog and can be used by other users. Here we can check if we can store a malicious script in the input field and when is displayed to a user the script is executed which results in the system being compromised.
   * Screenshot Before Action
   * A screenshot of a computer

     Description automatically generated with medium confidenceScreenshot After Action

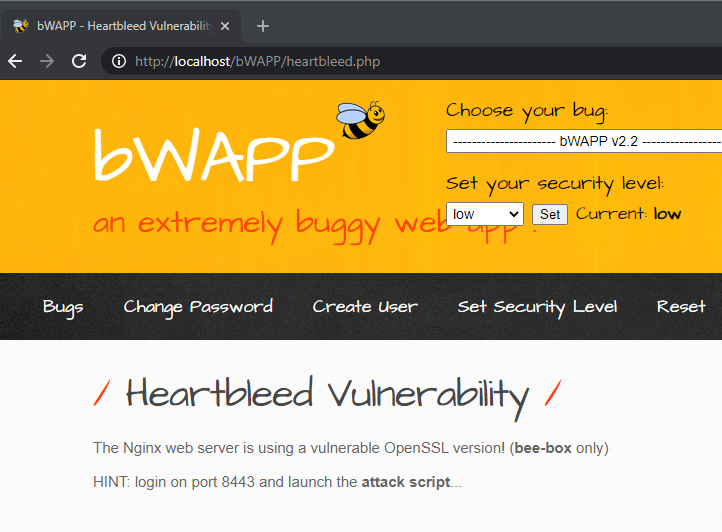
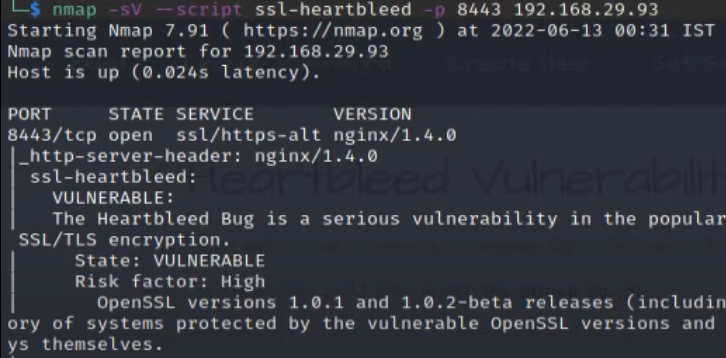
A screenshot of a computer

Description automatically generated with medium confidence

* + Observation: We see that we can successfully inject a script into the input field which is stored in the blog database. When a user requests the blog webpage the script is successfully ran and the user’s system is compromised. An attacker can use this bug to steam sensitive information from a user like his bank details or session cookies, it can also be used to inject malware via the script.
  + Preventions:
    1. To stop attackers from executing lengthy scripts, user input fields should be kept to a reasonable length so that huge scripts are not ran due to buffer overflow.
    2. To guarantee that only authorized users may access and alter stored data, put strong authorization, and access control measures in place.
    3. Use input validation techniques as well as sanitize the input and output when it is being stored or sent back from the database.

Using Components with Known Vulnerabilities

A sort of attack where an attacker makes use of flaws in software components used in an application is called using components with known vulnerabilities. This may occur if a software developer incorporates out-of-date or unpatched software libraries or components into their program. Attackers can access the application or its data by taking advantage of known weaknesses in these components.

1. **Heartbleed Vulnerability**
   * Route: We need to select the bug from the drop-down list located at the top right of the web page. From there a web page is loaded which prompts us to logon port 8443 to run the heartbleed attack script and check if there is a heartbleed vulnerability. When we run the heartbleed script on our bee box server we see that there is indeed a heartbleed vulnerability as the data returned by the script is more than it should be.
   * Screenshot Before Execution
   * Screenshot After Execution
   * Observation: We see that after running the python script the server is indeed Heartbleed vulnerable. This means an attacker exploiting the OpenSSL vulnerability can read a server’s memory without leaving a trace of their activities on the server. This flaw can allow an attacker to send the server a specifically constructed request, which would cause the server to return a memory chunk that might include sensitive data which can be read by the attacker.
   * Preventions:
     1. Make sure to always use the up-to-date OpenSSL version that has bug fixes.
     2. Monitor the systems regularly for any signs of unusual activity.