**SURE TRUST ORGANISATION**

**Domain : Cyber Security**

**Trainer : Mr. Himaneesh sir**

**Done by :**

**L . Bhavani Shankar**

**DAMN VULNERABLE WEB APPLICATION**

**Introduction :**

DVWA, or Damn Vulnerable Web Application, is a purposely designed web application created for educational and training purposes in the field of cybersecurity. Developed by security professionals, DVWA is intentionally filled with various vulnerabilities, allowing individuals to practice identifying, exploiting, and mitigating common web application security issues in a controlled environment.

**Key Objectives of DVWA:**

**Educational Tool:** DVWA serves as a practical, hands-on tool for individuals seeking to enhance their understanding of web application security. It provides a safe space to experiment with different vulnerabilities without any legal concerns.

**Security Training:** Security professionals, ethical hackers, and students use DVWA to sharpen their skills in identifying and addressing security vulnerabilities commonly found in real-world web applications.

**Vulnerability Exploration:** The application offers a range of security vulnerabilities, including SQL injection, cross-site scripting (XSS), file inclusion, and more. Users can explore these vulnerabilities to gain insights into how they work and their potential impact.

**Security Levels:** DVWA includes different security levels, allowing users to gradually increase the difficulty of challenges as they become more proficient. This helps individuals tailor their learning experience based on their skill level.

**Open Source:** DVWA is an open-source project, encouraging collaboration and contributions from the cybersecurity community. Its source code is freely available, enabling users to review, modify, and contribute to the project's development.

Downloading DVWA on Windows:

Requirements:

Install a web server stack like XAMPP or WampServer. Ensure PHP is installed and configured.

Steps:

* Download DVWA from the official repository on GitHub.
* Extract the downloaded zip file into the web server's root directory (e.g., htdocs in XAMPP).
* Configure the config.inc.php file with database settings.
* Open a web browser, navigate to http://localhost/DVWA/setup.php, and follow the setup instructions. After setup, access DVWA at <http://localhost/DVWA/index.php>.

Downloading DVWA on Linux:

Requirements:

* Install LAMP (Linux, Apache, MySQL, PHP) or a similar web server stack.

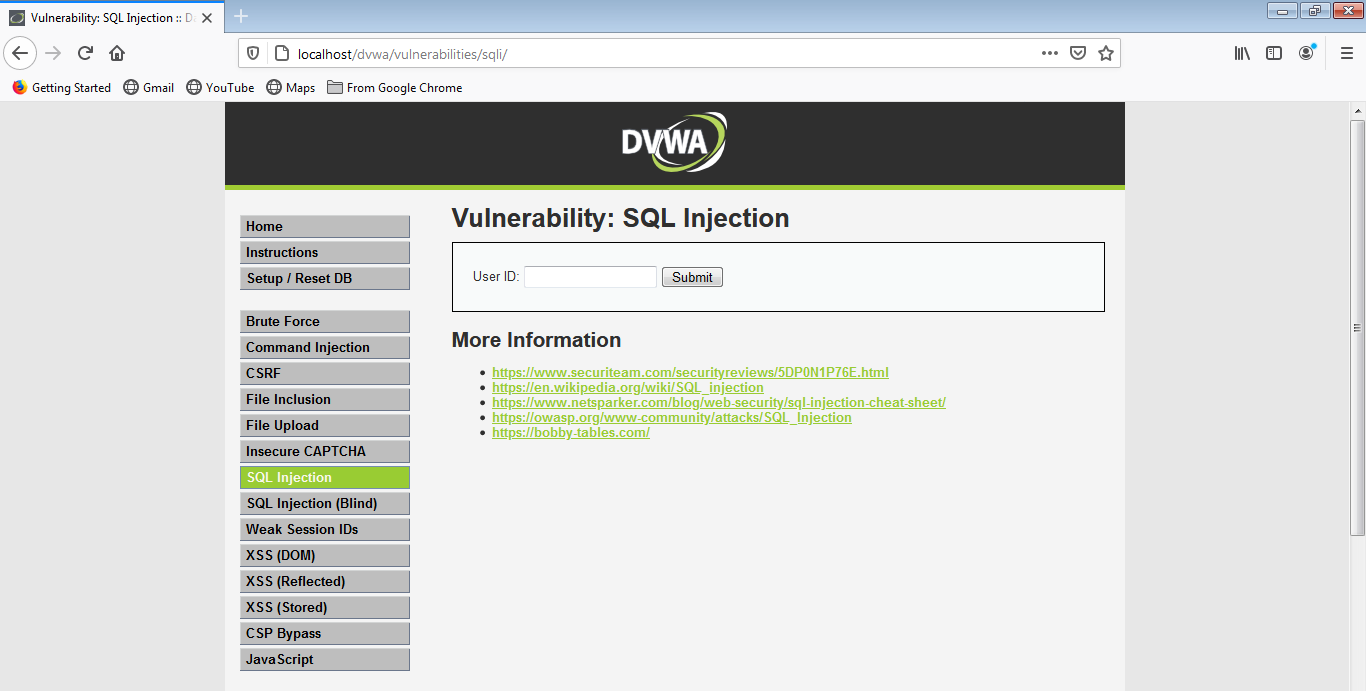
Steps:

* Clone DVWA from the official GitHub repository using the command:
* bash Copy code git clone <https://github.com/ethicalhack3r/DVWA.git>
* Move the DVWA directory to the web server's root (e.g., /var/www/html/).
* Configure the config.inc.php file with database settings.
* Open a web browser, navigate to http://localhost/DVWA/setup.php, and
* complete the setup.
* Access DVWA at <http://localhost/DVWA/index.php>.

If you took some time getting familiar with application, you might had already noticed that it has four difficulty levels:

* Low – if you have a difficulty level set to low, most of the vulnerabilities will be easy to exploit. There are no security measures implemented.
* Medium – this difficulty level illustrates how the security measures can be implemented in a bad way.
* High – this level will be a little bit trickier to exploit and it will provide a level of difficultness similar to the CTF (capture the flag) competitions. Impossible – if you think this is a challenge for you, well.. good luck. This level is secure and is created as an example of how the secure code should look versus insecure.

Naturally, you should start by low level and increase the difficulty while proceeding with it. The purpose of setting the DVWA security level to low is to investigate how the vulnerability could be exploited in the worst case – when there are no security measures at all.



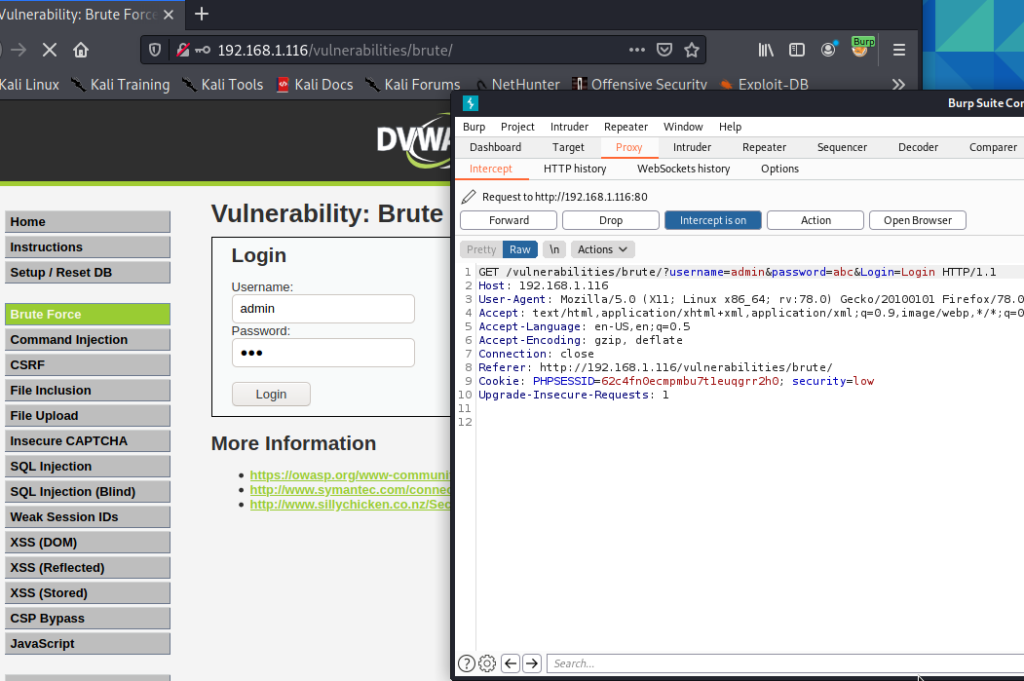
Brute Force

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.

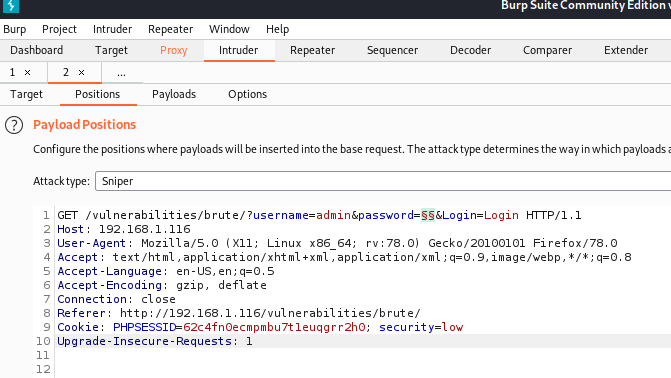
**Low**

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 principe of attack is pretty straightforward – you identify the request (GET request in this case) that sends login credentials, you use a dictionary with different words, and perform many requests. Then you review the responses and check if a password was identified during the attack.

If you want to use Burp for this purpose, what you should do is to run the Burp Suite, configure the proxy, then intercept the request from the DVWA brute force page.



After that, use a dictionary and try to brute force the password. In [Kali Linux](https://www.kali.org/), there are dictionaries in different locations, but you might use this one – **/usr/share/dict/wordlist-probable.txt**. As you might see in the picture below, the password was found pretty fast. If you paid attention to the response length, you will see that the request where **password**was used as a password, the response is longer than requests with the incorrect password.



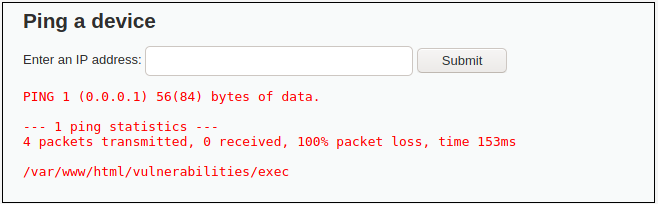
### Command Injection

Command injection is an attack that focuses on injecting and executing commands on OS. This should not be mistaken as code injection. Attack has potentially devastating effects – if a hacker can execute commands on the operating system, we can control the web application itself.

**Low**

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.

For the purpose of the example, let’s try to inject the **pwd** command. So that we could get the view of directory structure.



As you might see from the picture above, command injection was successful – server returned us path to the current directory – **/var/www/html/vulnerabilities/exec**.

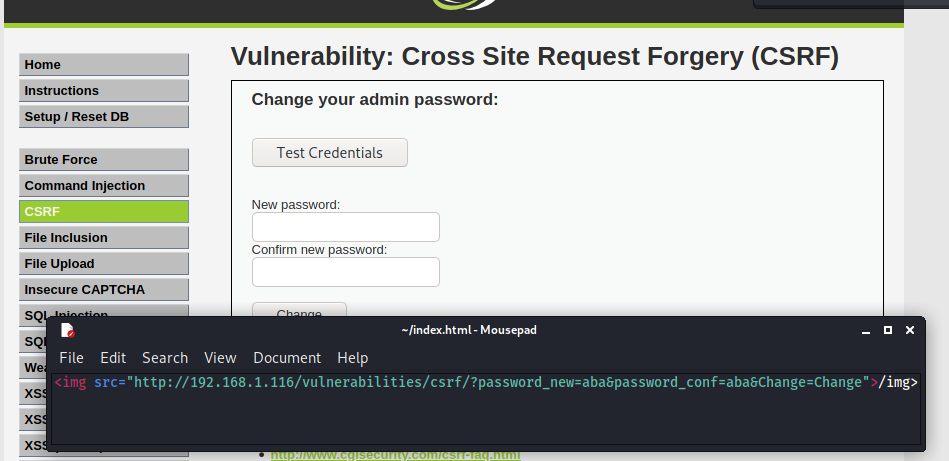
While this command was not that devastating, imagine, hacker, that managed to find command injection vulnerability, could completely control your OS. This includes stealing data or deleting every single file.

### 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. We can create a simple HTML file with an image element. That element will have a source, that we can set as a password changing request. In the image below you can see how the HTML file might look like. Just make sure you **set the** **IP of your Damn Vulnerable Web Application installation instead of the 192.168.1.116**.



After you created the file and saved it in a HTML format, the next step would be to run the HTML in your browser. After it is executed, the password will change for the admin user.

### 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.

**Low**

As you might see on the page, there are three files – **file1.php, file2.php**, and **file3.php**. All of them reside on a local server.



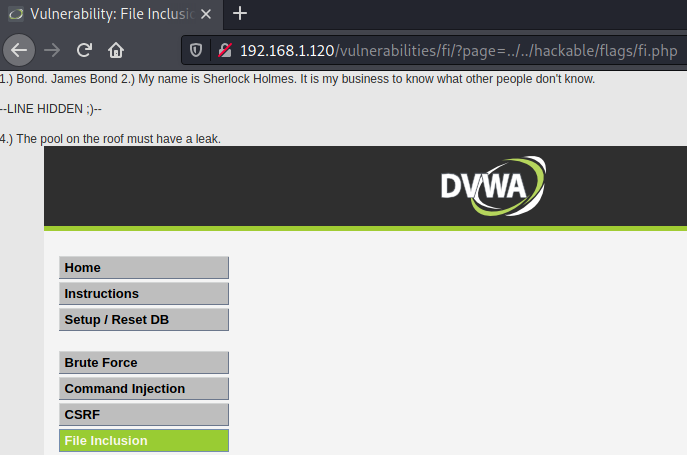
Now click on one of them, and pay attention to how the URL looks like. It loads file1.php like this: **?page=file1.php**.

URL Structure of the File Inclusion Vulnerability

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.

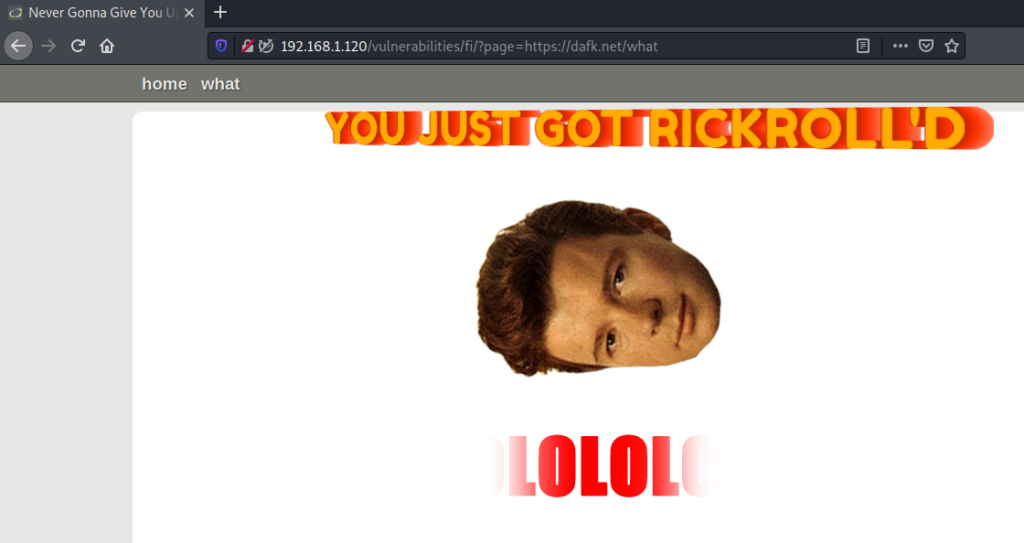
In the file inclusion DVWA vulnerability example there is a specific task – you have to access a specific PHP file. If you would try to access it manually, by visiting a page, you would get an error: **Nice try ;-). Use the file include next time!**But as there is another way to load a page, this problem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

As a result, you will get the secret quotes that previously were not visible.



However, there is more that can be exploited in this situation. You can easily get the passwd file with the local file inclusion: **?page=../../../../../../etc/passwd**.

But we covered only local, but not remote file inclusion. In case of a RFI, you can load an external script or page:



Included page will load on top of the DVWA page, and you will get RickRoll’D.

### File Upload

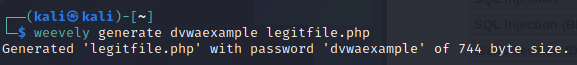
File upload vulnerability is one of the most dangerous ones. The reason for this is that uploaded files might be exploited in many ways: by making the server run a malicious script, or executing the script in the user’s browser. This can all potentially lead to hazardous compromise of a server and even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

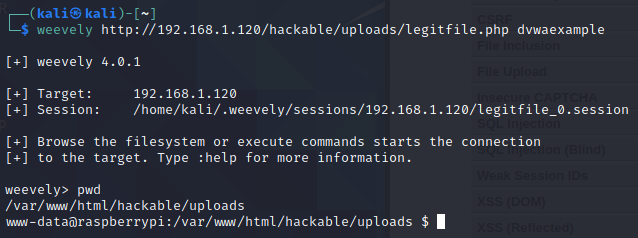
By default, Kali Linux comes with a reverse shell called weevely. The first step would be to generate an agent, and this can be done from the command line: **weevely generate your-password legitfile.php.**



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**.



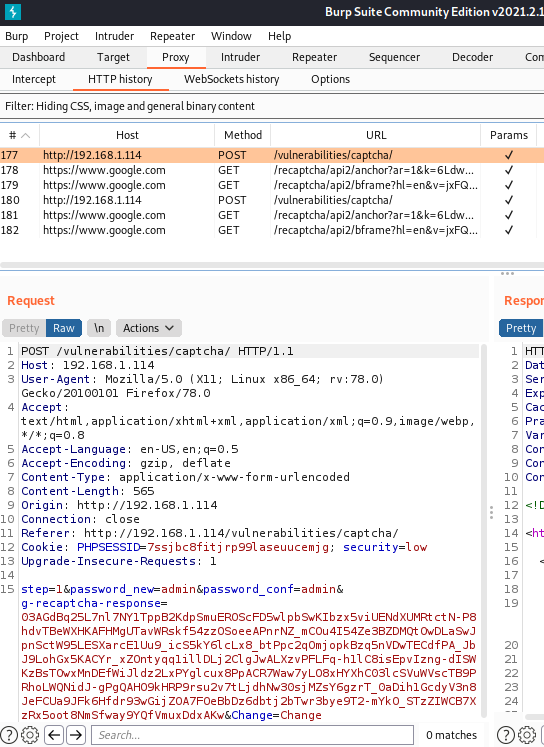
If everything worked out, you should get access. In my case, a connection with a **www-data** user of DVWA instance, which is located on Raspberry Pi, was gained. From this point, external actors might do a lot of harm.

### Insecure CAPTCHA

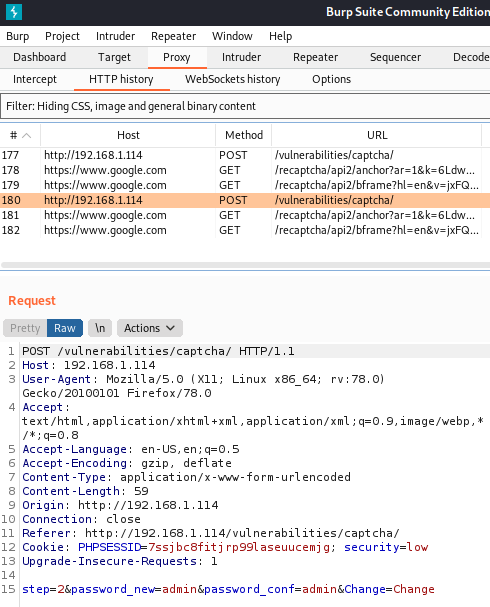
Captcha is a security mechanism against automated bots (robots). In order to prevent automatic actions, users should solve a simple task. This task can be solved only by a human and usually it is a great countermeasure against robots.

**Low**

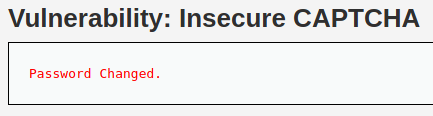
Even though a **captcha** is set for the low level, and you must confirm it before proceeding, it is not implemented properly. After submitting the **captcha** with a new **password, with a Burp proxy,** you can see that there are two requests that are mainly responsible for submiting the new password and captcha response. This is how the first one looks like:



However, the second request to the **/vulnerabilities/captcha/** page has no **g-recaptcha-response** in the POST body:



And in conclusion, we can repeat the second request that requires no reCaptcha response. This will bypass the captcha implementation and will change the password.



### SQL Injection

SQL injection is probably one of the most disastrous injection types. As this vulnerability is able to affect the most important part of the system – data, sequences can be devastating. This can vary from altered data to the full data loss after it was deleted by a malicious actor.

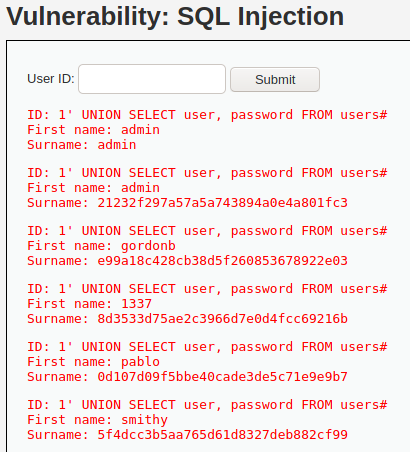
**Low**

As the task for the DVWA SQL Injection vulnerability is to get the passwords of the users, this query will return the passwords of existing users.

1' UNION SELECT user, password FROM users#

PRO TIP: if you get an error with this query, write the symbol ‘ by hand on the SQL Injection DVWA page.

As the input field requires user ID, we give it to the server, but additionally, we append our own command (that is constructed accordingly to the database type and version). Command reaches databases and returns us the users list. Returned passwords are hashed. They can be de-hashed by using Kali Linux[tools](https://tools.kali.org/password-attacks/findmyhash).



### SQL Injection (Blind)

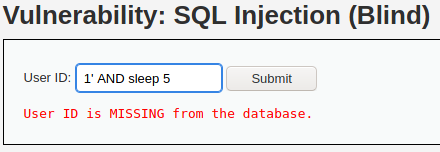
It might be trickier to check if an input field leads to the **blind SQL injection**. It does differ from the classical SQL injection by the fact that it does not show directly the results of injection.

By executing the query and checking if there are any errors on the page is one of the ways to test for blind SQLi.

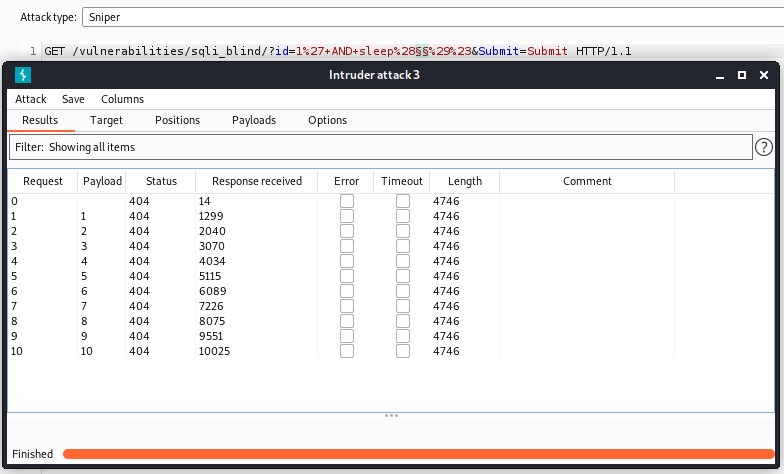
Another way, that will be used in our **DVWA blind SQL injection example, is trying to inject sleep operation into** the database. By comparing normal behavior to the application behavior after **sleep() injection**, we might tell if the blind SQL exists in the Damn Vulnerable Web Application.

**Low**

Just like the classic SQL injection, that we’ve covered in the previous section, a blind one requires a user ID. What can we do in order to test if the **DVWA blind SQL injection** exists, is to add the mentioned sleep function: **1′ AND sleep(5)#**.



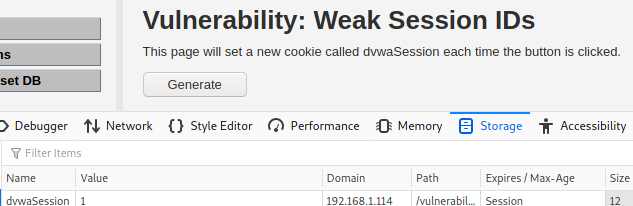
You might have noticed that this request took a while to execute. We can use [Burp Suite intruder](https://portswigger.net/burp/documentation/desktop/tools/intruder/using) to make it even more obvious. Let’s submit and intercept the request with the sleep function. Then we can send it to the intruder, add a list of payloads that consists of **numbers 1-10**, add the position for inserting value to **sleep() function**, and run it.



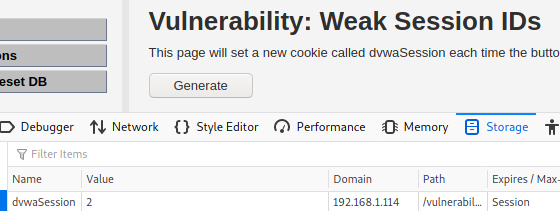
We can observe that the response time increases each time we use a higher number payload.

**Low**

By viewing the **browser’s developer tools’ Storage tab**, we can see that first time the session ID is equal to 1 – **dvwaSession** value is set to 1.



After clicking on **Generate** button for second time, we can see that the ID gets the value 2. From this we can state that the ID generation is incremental and it is easy to guess what session ID will be generated the next time.

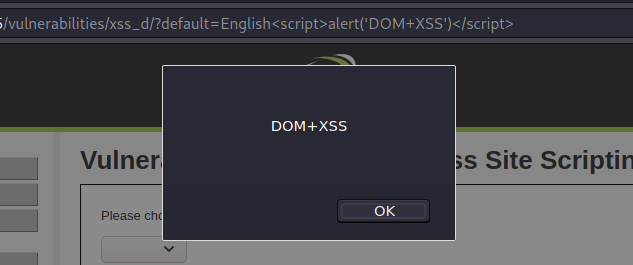


### XSS (DOM)

Cross-Site Scripting (XSS) is another injection attack. With this attack, malicious scripts are injected into the website and executed as legitimate ones. There are a few types of XSS, one of them is DOM-based XSS. This works differently than reflected or stored XSS, as DOM-based XSS happens because of the modification of a DOM environment by the client-side script.

**Low**

There is a select with different values as the language options. As there is no validation, we can append a script in this manner: **?default=Englishalert(‘DOM+XSS’)**.

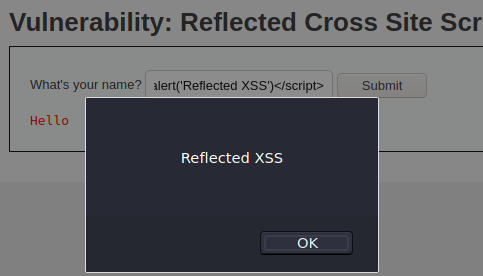


### XSS (Reflected)

Reflected is another type of the XSS. This injection is not persistent, one of the examples of how this can be exploited is when the user is tricked to click on a malicious link.

**Low**

Exploiting reflected XSS on low level is easy – all you need to do is to submit a **<script>alert(‘Reflected XSS’)</script>**.



### XSS (Stored)

Stored XSS is permanently stored on the website and malicious scripts can be executed every time user visits the page. For example, if a website is vulnerable and a script is injected into comments form, every users’ browser will execute it on page visit.

**Low**

Stored XSS does not differ from the reflected XSS by its nature. Payloads used in the previous section would work for stored XSS.

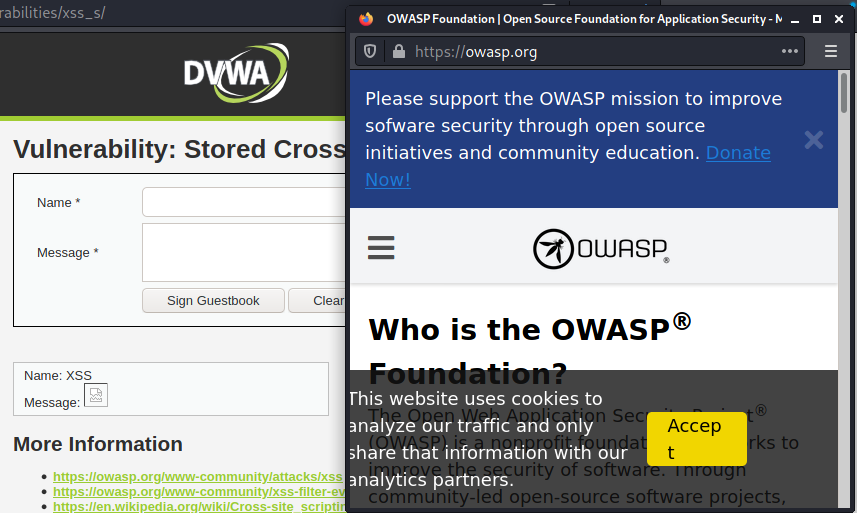
But the task for the stored XSS DVWA vulnerability is to make a redirect to an external page. We can do this in different ways, one of them is to add an **<img>** element that, on error, opens a page:

<img src="https://nonexisting.url" onerror=window.open("https://www.owasp.org","xss",'height=500,width=500');>

As the text field has a max size of 50 characters, we have to extend it, otherwise our payload won’t fit the field.

Increasing Max Length of the Field

After submitting the **img element** it should be saved in the guestbook. Each time user visits this page, another browser window will be opened with owasp.org. Keep in mind that for the first time **you will get an alert that the browser blocked popups.** After allowing the popups, you will see this:



### CSP Bypass

[Content Security Policy (CSP)](https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP)defines what resources can be used on the page. If the policy is set and it disallows external resources, then no external script could be loaded and executed.

**Low**

For the DVWA, checking if the CSP is implemented, is easy. Actually, this is the same for any case – the server responds with a **Content-Security-Policy** header that states what external resources are allowed.

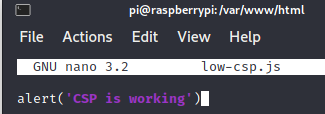
**Content-Security-Policy: script-src ‘self’ https://pastebin.com hastebin.com example.com code.jquery.com https://ssl.google-analytics.com ;**

And in this case, **self**, **pastebin, hastebin, example, code.jquery**, and **ssl.google-analytics** are allowed.

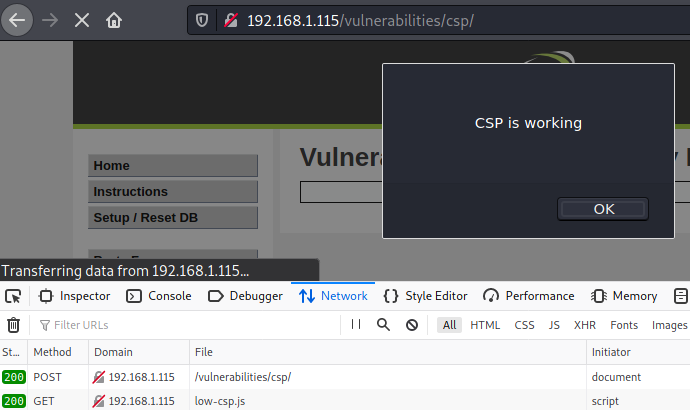
Allowed Resources for Low DVWA CSP Vulnerability

Even though Pastebin is one of the domains allowed by the policy, it is [not possible to use Pastebin for low DVWA CSP level because of the changes in Pastebin website](https://github.com/digininja/DVWA/issues/382). Even though we are able to load content from the page, it will be **text, not script** format.

As the **‘self’**is allowed in the CSP, we can host our own script from the DVWA server. Create a new file in the **/var/www/html** directory of your DVWA installation and put a single line with an alert: **alert(‘CSP is working’)**.



After this, try to load the script by submitting URL with the script – **http://YOUR-INSTALLATION-IP/low-csp.js**.



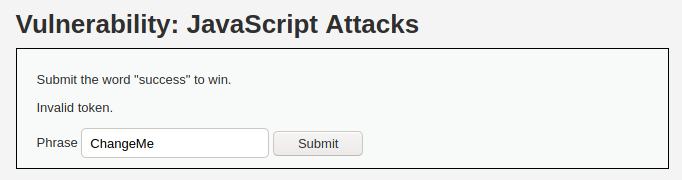
As a result, script will be executed.

### JavaScript

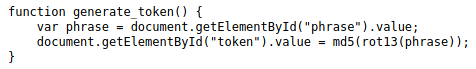
While JavaScript scripts can’t be called as the vulnerabilities itself, it can definitely become an attack vector. And also, in some cases JavaScript script can give an attacker useful information that might lead to further system exploitation. DVWA JavaScript vulnerability focuses on showing what can be the potential consequences if a script contains sensitive information.

**Low**

There is an easy task (at least it looks easy) – enter word “success” and hit Submit. However, the problem you will face is the token. Token will be invalid.



However, the purpose of the DVWA JavaScript vulnerability is to reverse engineer JavaScript code to get the needed information. Such information will help to exploit vulnerability. First step would be to locate JavaScript code. This can be done by inspecting page HTML. You will that there is a JS code included inside the <script> element. One function is really interesting for us. It consists of the logic for creating token, which we need.

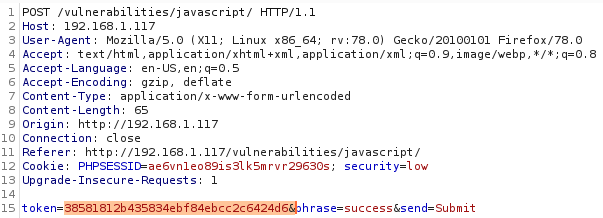


Now we know that we can construct token by using ROT13 function and generating a MD5 hash. If you are on Kali Linux, this can be done easily with the terminal only:

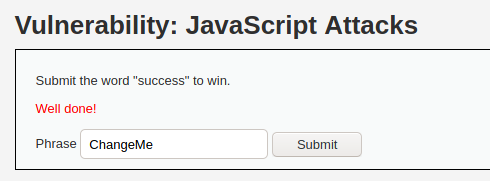
**echo -n ‘success’ | tr ‘A-Za-z’ ‘N-ZA-Mn-za-m’ | md5sum**

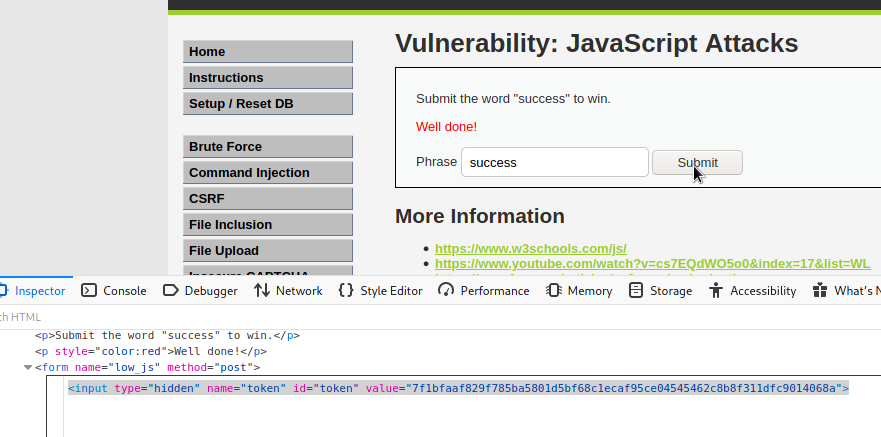
As a result, token will be generated – **38581812b435834ebf84ebcc2c6424d6**.

Let’s try to send a request, this time with the correct token. Intercept request with Burp Suite and edit the token.



And BINGO. We managed to get the phrase ‘**Well done!’**instead of the **‘Invalid token’**.





**Conclusion :**

DVWA serves as an indispensable tool for cybersecurity enthusiasts, ethical hackers, and security professionals aiming to enhance their understanding of web application security. This intentionally vulnerable web application provides a controlled environment for users to explore and practice exploiting various security vulnerabilities commonly found in real-world web applications.

**Educational Value:** DVWA is primarily designed as an educational resource, allowing users to gain hands-on experience in identifying, exploiting, and mitigating security vulnerabilities.

**Realistic Challenges:** The application offers a diverse set of security challenges, including SQL injection, cross-site scripting (XSS), command injection, and more. Users can choose from different security levels to match their skill proficiency.

**Safe Testing Ground:** DVWA ensures a safe testing environment where users can experiment with security vulnerabilities without posing any risks to actual production systems.

**Gradual Skill Development:** With varying security levels, DVWA accommodates users at different skill levels. Beginners can start with easier challenges and gradually progress to more complex scenarios.

**Open Source Collaboration:** Being an open-source project, DVWA encourages collaboration and contributions from the cybersecurity community. The source code is freely accessible, promoting transparency and community-driven improvement.

**Continuous Updates:** DVWA receives regular updates to introduce new challenges, address vulnerabilities, and align with evolving security standards. This ensures that users have access to up-to-date and relevant security scenarios.

**Responsible Usage:** While DVWA provides a valuable learning platform, users must approach it responsibly. It is crucial to use DVWA in controlled environments, respecting legal and ethical boundaries. Practicing responsible disclosure and sharing knowledge gained from DVWA contribute to the overall improvement of web application security awareness.

In conclusion, Damn Vulnerable Web Application stands as an instrumental tool in the realm of cybersecurity education, providing a practical and immersive learning experience for individuals aspiring to master the art of securing web applications. Always use DVWA with a commitment to responsible and ethical hacking practices.