Lab - Analyze Automation Code

Objectives

In this lab, you will complete the following objectives:

- Part 1: Write a Bash Script to Automate an Nmap Scan and Store the Results.
- Part 2: Differentiate between Scripts Written in Bash, Python, Ruby, and PowerShell.

Background / Scenario

Penetration testing often requires repetitive tasks that use various tools to perform reconnaissance, analyze, and exploit vulnerable systems. Creating scripts to automate these tasks reduces the time necessary to complete the penetration testing project.

Required Resources

- Kali VM customized for the Ethical Hacker course

Instructions

Part 1: Write a Bash Script to Automate an Nmap Scan and Store the Results.

Step 1: Create a basic Bash script.

The Bash shell has a built-in script interpreter. Bash scripts can be written in any text editor and require minimal programming experience. The scripts can then be run from the Bash shell prompt. The syntax and structure of Bash scripts is similar to what you would type at the command prompt if you were doing the task manually. In this step, you will write a short script named recon.sh to perform a simple Nmap scan.

a. To begin your first script, log in to Kali, with the username **kali** and password **kali**. Open a terminal window and ping the target host at 10.6.6.23 to ensure that it is available on the network.

```
┌──(kali㉿Kali)-[~]
└─$ ping -c5 10.6.6.23
PING 10.6.6.23 (10.6.6.23) 56(84) bytes of data.
64 bytes from 10.6.6.23: icmp_seq=1 ttl=64 time=0.229 ms
64 bytes from 10.6.6.23: icmp_seq=2 ttl=64 time=0.048 ms
64 bytes from 10.6.6.23: icmp_seq=3 ttl=64 time=0.060 ms
64 bytes from 10.6.6.23: icmp_seq=4 ttl=64 time=0.054 ms
64 bytes from 10.6.6.23: icmp_seq=5 ttl=64 time=0.038 ms


--- 10.6.6.23 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 5451ms
rtt min/avg/max/mdev = 0.038/0.085/0.229/0.071 ms
```

b. Open the **Mousepad** text editor from the **Applications** menu. (Any text editor can be used to create the file.) The first line in the Bash script is a special kind of comment line that indicates the location of the interpreter to be used to run the code. This line is called a "shebang" and is common to most Linux scripts. Enter the shebang **#!/bin/bash** on line 1 this identifies the language of the script to the command interpreter.

Line numbers are added automatically in Mousepad, do not type them. They should not appear in the script itself.

```
1 #!/bin/bash
```

c. In this script, the user will enter the IP address of the target as a command line option. If no option is entered, the user will receive an error message showing the proper command syntax. Enter the **if/then** sequence as shown.

```
2 # Check if IP of target is entered
3 if [ -z "$1" ]
4 then
5     echo "Correct usage is ./recon.sh <IP>"
6     exit 1
7 else
8     echo "Target IP $1"
9 fi
```

Analyzing script code is an important skill for a penetration tester. Not only will you write automation code, you will often need to determine what an existing script does. The meaning of each line of is as follows:

- Line 2 is a comment line that begins with a hash tag #. Lines starting with # are used to document the script. Comment lines are ignored by the command interpreter.
- Line 3 starts a test to determine if the input option variable **$1** exists. By default, Bash scripts accept command line options into variables numbered by their position in the command. The **-z** returns "true" if the value of **$1** is null. Bash requires a space after the first bracket **[** and a space before the last bracket **]**.
- Line 4 indicates what to do if the option variable does not exist (is null). Lines 5 and 6 are indented to indicate that they are part of the **then** clause.
- Line 5 prints a message to the screen. Bash uses the **echo** command to print what is in the double quotes to the screen.
- Line 6 will cause the script execution to stop and exit to the CLI if the condition is met.
- Line 7 indicates what to do if the if condition is false.
- Line 8 prints a message with the input value that was supplied and stored in the **$1** variable. Note that further work is required to validate that the input was actually a valid IP address. This is beyond the scope of this lab.
- Line 9 indicates that the **if/then** clauses are complete.

d. Save your file with the name **recon.sh**. In this example, the file is saved in the **/home/kali** directory.

e. To make a text file into an executable, it is necessary to change the Linux permissions on the file. Open a terminal window on the Kali desktop. List the directory using **ls** and verify that your script file is there and has the correct name. Enter the **chmod +x** command to add the executable permission to your file.

```
┌──(kali㉿Kali)-[~]
└─$ chmod +x recon.sh
```

f. Test your script by running it first with the IP address of the target (10.6.6.23) specified.

```
┌──(kali㉿Kali)-[~]
└─$ ./recon.sh 10.6.6.23
Target IP 10.6.6.23
```

g. Further test your script by running it as follows:
   1. with no input supplied after the script name
   2. with other text or an invalid IP address supplied after the script name. Note that if you are entering non-numeric text, it must be surrounded by quotation marks.

The purpose of the script is to automate Nmap scanning using the target IP address value that is supplied to the script. What do you think will happen if the value is not a legal IP address?

Answer Area

<!----><!---->

h. Now edit the script file to enter the commands that will run the Nmap scan. Use the variable **$1** to indicate the IP address of the target device you want to scan. The results of the Nmap scan will be written to a file named **scan_results.txt** in the current directory.

```
8  echo "Running Nmap…"
9  # Run Nmap scan on target and save results to file
10 nmap -sV $1 > scan_results.txt
11 echo "Scan complete – results written to scan_results.txt"
```

What type of Nmap scan will be run with this script?

Answer Area

<!----><!---->

i. Save the script. Below is the **recon.sh** script so far.

```
#!/bin/bash
# Check if IP of target is entered
if [ -z "$1" ]
  then
    echo "Correct usage is ./recon.sh <IP>"
    exit
  else
    echo "Target IP $1"
    echo "Running Nmap…"
# Run Nmap scan on target and save results to file
    nmap -sV $1 > scan_results.txt
    echo "Scan complete – results written to scan_results.txt"
```

```
fi
```

j. Run it again with the target IP address supplied.

```
┌──(kali㉿kali)-[~]
└─$ ./recon.sh 10.6.6.23
Target IP 10.6.6.23
Running Nmap....
Scan complete -- results written to scan_results.txt
```

k. Use the **cat** command to view the contents of the **scan_results.txt** file that you created with the script.
What ports are open on the target?

Answer Area

<!----><!---->

Step 2: Modify the script to enumerate shares on the target.

As seen in the previous step, the target at 10.6.6.23 has open ports that could indicate a Samba server. In this step, you will edit your script to run **enum4linux** if a Samba drive share port is open to determine any available drive shares or user accounts.

What indicates that a Samba server is running on the hosts?

Answer Area

<!----><!---->

a. Open the **recon.sh** file in the text editor. Add the following commands.

```
13 # If the Samba port 445 is found and open, run enum4linux.
14 if grep 445 scan_results.txt | grep -iq open
15 then
16    enum4linux -U -S $1 >> scan_results.txt
17    echo "Samba found. Enumeration complete."
18    echo "Results added to scan_results.txt."
19    echo "To view the results, cat the file."
20 else
21    echo "Open SMB share ports not found."
22 fi
```

b. Analyze the additional code.

- Line 13 is a comment.
- Line 14 indicates the start of an **if/then** statement that will search in the Nmap results for open port 445. The **grep** command searches lines in the file that match the pattern “445 open.” The **grep** command searches for lines that match the pattern “445” first. Then the output is piped into a second **grep** command to search again for lines that match the pattern **open**. With the option **-i**, the **grep** command ignores the case distinctions in the search patterns. The option **-q** suppresses standard outputs.
- Line 15 is the "**then**" clause. This contains the command that will be executed should the **if** test return "true".
- Lines 16 - 19 are executed if the SMB file sharing port (445) is found. Line 16 runs **enum4linux** with the **-U** and **-S** options on the target host specified in **$1** and appends the results to the end of the **scan_results.txt** file. Lines 17, 18 and 19 display messages when emun4linux finished the scan and provides directions to view the results.
- Line 20 indicates that the action to take if the logical if the condition fails.
- Line 21 displays a message if the SMB file sharing port (445) is not open.
- Line 22 the **fi** signifies the end of the **if/then** clause.

What do the **-U** and **-S** options do in enum4linux?

Answer Area

<!----><!---->

c. Save the **recon.sh** file in the text editor and exit to the command prompt. Below is the complete **recon.sh** script.

```
#!/bin/bash
# Check if IP of target is entered
if [ -z "$1" ]
 then
```

```
    echo "Correct usage is ./recon.sh "
   exit
  else
    echo "Target IP $1"
    echo "Running Nmap…"
# Run Nmap scan on target and save results to file
    nmap -sV $1 > scan_results.txt
    echo "Scan complete – results written to scan_results.txt"
fi
# If the Samba port 445 is found and open, run enum4linux.
if grep 445 scan_results.txt | grep -iq open
  then
    enum4linux -U -S $1 >> scan_results.txt
    echo "Samba found. Enumeration complete."
    echo "Results added to scan_results.txt."
    echo "To view the results, cat the file."
  else
   echo "Open SMB share ports not found."
fi
```

d. Run the script again on the target system (10.6.6.23).

```
┌──(kali㉿kali)-[~]
└─$ ./recon.sh 10.6.6.23
Running Nmap....
Scan complete -- results written to scan_results.txt
Samba found. Enumeration complete.
Results added to scan_results.txt.
To view the results, cat the file.
```

e. Use the **cat** command to view the results contained in the **scan_results.txt** file.
What file shares were found on the target?

Answer Area


```
<!----><!---->
```


Step 3: Automate Nmap from the command line.

Another way to automate Nmap is to scan a group of specific targets that are specified in an external file.

a. Create a new file in Mousepad and type in the IP addresses of the existing hosts on the 10.6.6.0/24 network. To list all the available hosts with their IP addresses, enter the command **containers** at a terminal.

Be sure the IP addresses are separated with a space or list each IP address on a separate line.

b. Save the file with the name **to_scan.txt**.
c. At the prompt, enter the command to run Nmap with the targets from the file. For the purposes of this lab, will just run a simple ping scan, but any type of scan that takes an IP address as a target can be run in this way.

```
┌──(kali㉿Kali)-[~]
└─$ nmap -sn -iL to_scan.txt
```

d. After a brief delay, you should see Nmap output the scan reports for each host that was specified in the **to_scan.txt** file.

Note: The to_scan.txt file does not require executable permissions because it is serving as a data file, not as a script file.

Part 2: Differentiate between scripts written in Bash, Python, Ruby, and PowerShell

In this part, you will use what you learned in the previous part about writing and analyzing a Bash script to analyze pre-written scripts. Knowing what scripting language is being used in scripts that you discover while penetration testing enables you to understand the purpose of the script, and potentially be able to modify it to obtain additional information.

Use this chart that illustrates the different syntax characteristics of the scripting languages.

Function	Python	Bash	Ruby	PowerShell
Shebang Example – special comment line at the top of script that identifies the path to the interpreter	`#!/usr/bin/python3`	`#!/bin/bash`	`#!/usr/local/bin/ruby`	`#!/usr/bin/env pwsh` Only needed if running PS in Linux, not required in Windows
Loading Modules	`import libraryName as alias` `from libraryName import subModule`	n/a; Bash does not require loading modules	Require *'libraryName'* Self contained libraries are called 'gems'	n/a; PowerShell does not require loading modules
Defining Variables	`variableName = variableValue`	`variableName = variableValue` Variables read in as options from CLI are assigned $1, $2, ..., $n	`variableName = variableValue` Cannot begin with number or capital letter.	`$varvariableName = varvariableValue`
Calling Variables	*variableName* Example*:* **print***(varibleName)*	*$variableName* Example*:* **echo** *$variableName*	*#variableName* Example*:* **puts** *variableName*	*$varibleName* Example*:* `PS C:\> $variableName`
Comparison	Uses Arithmetic symbols Equal is **==** Not Equal is **!=** Greater Than is **>** Equal or Greater Than is **>=** Less Than is **<** Equal or Less Than is **<=**	Uses alpha: Equal to **-eq** Not equal to **-ne** Greater than **-gt** Greater than or equal to **-ge** Less than **-lt** Less than or equal to **-le** Example: **$x -gt 8** If using arithmetic symbols, enclose in double parenthesis. **(($a > $b))**	Uses Arithmetic symbols Equal is **==** Not Equal is **!=** Greater Than is **>** Equal or Greater Than is **>=** Less Than is **<** Equal or Less Than is **<=**	Uses a variety of operators: Equal to: **-eq**, **-ieq**, **-ceq** Not equal to: **-ne**, **-ine**, **-cne** Greater than: **-gt**, **-igt**, **-cgt** Greater than or equal to: **-ge**, **-ige**, **-cge** Less than: **-lt**, **-ilt**, **-clt** Less than or equal to: **-le**, **-ile**, **-cle**

If Conditions	**if** *condition1*:
	  action1
	elif *condition2*:
	  action2
	else:
	  action3
		if [*condition1*]
		  then *action1*
		elif [*condition2*]
		  then *action2*
		else
		  action3
		fi
			if *condition1*
			  action1
			elsif *condition2*
			  action2
			else
			  action3
			end
			
			unless can be used if just checking If a condition is "not true"
				if (*condition1*) {
				  action1
				}
				elseif (*condition2*) {
				  action2
				}
				else {
				  action3
				}

Do while loops	Example:
	i= 1
	while i < 6**:**
	  print(i)
		Example:
		x=1
		while [$x -le 5 **]**
		do
			Example:
			while x >= 1
			  puts #@x
			  x = x - 1
				Example :
				do {
				  Write-Host $x
				  $x =$ x--

	`i = I + 1`
	`print ("All done")`
		`echo "count " $x`
		`x=$(( $x + 1 ))`
		`done`
			`end`	`} while ($x -ge 1)`

a. Review the code sample shown. Use the syntax characteristics to determine which scripting language is used to interpret the code.

```
import nmap
# take the range of ports to
# be scanned
begin = 21
end = 80
target = '10.6.6.23'
# scan the port range
for i in range(begin,end+1):
 results = nmap.PortScanner(target,str(i))
 results = results['scan'][target]['tcp'][i]['state']
 print('Port {i} is {results}.')
```

Using the information from the chart, what shebang should be the first line of code?

Answer Area

<!----><!---->

What port range will be scanned?

Answer Area

<!----><!---->

b. Review the code sample shown. Use the syntax characteristics to determine which scripting language is used to interpret the code.

```
require 'nmap/command'
Nmap::Command.sudo do |nmap|
  nmap.syn_scan       = true
  nmap.os_fingerprint = true
  nmap.service_scan   = true
  nmap.output_xml     = 'scan.xml'
  nmap.verbose        = true
  nmap.ports   = [20, 21, 22, 23, 25, 80, 110, 443, 512, 522, 8080, 1080]
  nmap.targets = '10.6.6.*'
end
#Parse Nmap XML scan files:
require 'nmap/xml'
Nmap::XML.open('scan.xml') do |xml|
  xml.each_host do |host|
    puts "[#{host.ip}]"
    host.each_port do |port|
      puts "  #{port.number}/#{port.protocol}        #{port.state}   #{port.service}"
    end
  end
end
```

Using the information from the chart, what scripting language interpreter will be used to run this code?

Answer Area

<!----><!---->

What is the target of this Nmap scan?

Answer Area

<!----><!---->

c. Review the code sample shown. Use the syntax characteristics to determine which scripting language is used to interpret the code.

```
1  $nmapExe = "Program Files (x86)Nmap
map.exe"
2  #define nmap targets
3  $target = "10.6.6.0/24", "172.17.0.0/29"
4  #run nmap scan for each target
5  foreach ($target in $target)
6  {
7      $filename = "nmap_results"
8      $nmapfile = ".   emp" + $filename + $target +".xml"
9      cmd.exe /c "$nmapExe -p 20-25,80,443,3389,8080 -oX $nmapfile -A -v $target"
10  }
```

Using the information from the chart, what scripting language interpreter will be used to run this code?

Answer Area

<!----><!---->

What options will Nmap use for the scan and what do those options mean?

Answer Area

<!----><!---->

Reflection Question

Code analysis skills are tested on penetration testing certifications. What benefit does having code analysis skills provide to penetration testers when discovering vulnerabilities?

Answer Area

<!----><!---->