Práctica de laboratorio - Análisis de código de automatización

Objetivos

En esta práctica de laboratorio se cumplirán los siguientes objetivos:

- Parte 1: Escribir un script de Bash para automatizar un escaneo de Nmap y almacenar los resultados.
- Parte 2: Diferenciar entre scripts escritos en Bash, Python, Ruby y PowerShell.

Trasfondo / Escenario

Las pruebas de penetración a menudo requieren tareas repetitivas que utilizan varias herramientas para realizar el reconocimiento, el análisis y el aprovechamiento de los sistemas vulnerables. La creación de guiones para automatizar estas tareas reduce el tiempo necesario para completar el proyecto de pruebas de penetración.

Recursos necesarios

- Kali VM personalizada para el curso Ethical Hacker

Instrucciones

Parte 1: Escriba un script de Bash para automatizar un escaneo de Nmap y almacenar los resultados.

Paso 1: Cree un script básico de Bash.

El shell de Bash tiene un intérprete de scripts incorporado. Los scripts de Bash se pueden escribir en cualquier editor de texto y requieren una experiencia mínima en programación. Los scripts se pueden ejecutar desde el indicador de shell de Bash. La sintaxis y la estructura de los scripts de Bash son similares a las que escribiría en la línea de comandos si realizara la tarea manualmente. En este paso, escribirá un breve script denominado `recon.sh` para realizar un escaneo simple de Nmap.

1. Para comenzar su primer guion, inicie sesión en Kali con el nombre de usuario **kali** y la contraseña **kali**. Abra una ventana de terminal y haga ping al host de destino en 10.6.6.23 para asegurarse de que esté disponible en la red.

```
┌──(kali㉿Kali)-[~/Desktop]
└─$ 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=0.046 ms
64 bytes from 10.6.6.23: icmp_seq=2 ttl=64 time=0.100 ms
64 bytes from 10.6.6.23: icmp_seq=3 ttl=64 time=0.038 ms
64 bytes from 10.6.6.23: icmp_seq=4 ttl=64 time=0.048 ms
^C
── 10.6.6.23 ping statistics ──
4 packets transmitted, 4 received, 0% packet loss, time 3051ms
rtt min/avg/max/mdev = 0.038/0.058/0.100/0.024 ms
```

2. Abra el editor de texto **Mousepad** desde el menú **Applicacions**. (Se puede utilizar cualquier editor de texto para crear el archivo). La primera línea del script de Bash es un tipo especial de línea de comentario que indica la ubicación del intérprete que se utilizará para ejecutar el código. Esta línea se denomina "shebang" y es común a la mayoría de los scripts de Linux. Ingrese shebang **#!/bin/bash** en la línea 1, esto identifica el lenguaje del guion para el intérprete de comandos.

Los números de línea se agregan automáticamente en Mousepad, no los escriba. No deben aparecer en el script.

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

3. En este script, el usuario ingresará la dirección IP del destino como una opción de línea de comando. Si no se ingresa ninguna opción, el usuario recibirá un mensaje de error con la sintaxis correcta del comando. Ingrese la secuencia **if/then** como se muestra.

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

Analizar el código del script es una habilidad importante para un pentester. No solo escribirá código de automatización, a menudo deberá determinar qué hace un script existente. El significado de cada línea de es el siguiente:

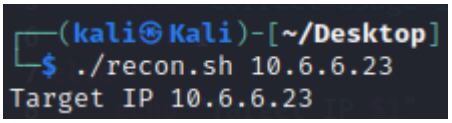
- La línea 2 es una línea de comentario que comienza con un número de etiqueta hash. Las líneas que comienzan con # se utilizan para documentar el script. El intérprete de comandos ignora las líneas de comentario.
- La línea 3 inicia una prueba para determinar si existe la variable de opción de entrada **$1**. De manera predeterminada, los scripts de Bash aceptan opciones de la línea de comandos en variables numeradas por su posición en el comando. **-Z** devuelve

"verdadero" si el valor de **$1** es nulo. Bash requiere un espacio después del primer corchete **[y un espacio antes del último corchete]**.

- La línea 4 indica qué hacer si la variable de opción no existe (es nula). Las líneas 5 y 6 están indentadas para indicar que forman parte de la cláusula **then**.
- La línea 5 imprime un mensaje en la pantalla. Bash usa el comando **echo** para imprimir lo que está en las comillas dobles en la pantalla.
- La línea 6 hará que la ejecución del script se detenga y salga a la CLI si se cumple la condición.
- La línea 7 indica qué hacer si la condición if es falsa.
- La línea 8 imprime un mensaje con el valor de entrada que se proporcionó y almacenó en la variable **$1**. Tenga en cuenta que se requiere más trabajo para validar que la entrada sea realmente una dirección IP válida.
- La línea 9 indica que las cláusulas **if/then** están completas.

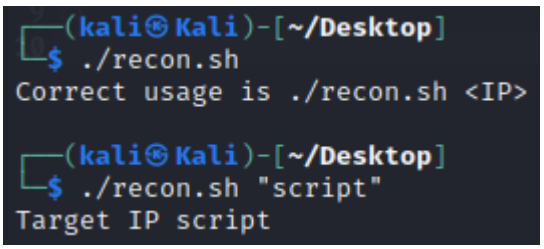
4. Guarde su archivo con el nombre **recon.sh**. En este ejemplo, el archivo se guarda en el directorio **/home/kali/Desktop**.
5. Para convertir un archivo de texto en un ejecutable, es necesario cambiar los permisos de Linux en el archivo. Abra una ventana de terminal en el escritorio de Kali. Enumere el directorio con **ls** y verifique que el archivo de script esté allí y tenga el nombre correcto. Ingrese el comando **chmod +x** para agregar el permiso ejecutable a su archivo.

6. Pruebe el script ejecutándolo primero con la dirección IP del destino (10.6.6.23) especificada.



7. Pruebe el script ejecutándolo de la siguiente manera:
   1. sin entrada proporcionada después del nombre del guion
   2. con otro texto o una dirección IP no válida después del nombre del guion. Tenga en

cuenta que si ingresa texto no numérico, debe ir entre comillas.



El propósito del guion es automatizar el escaneo de Nmap usando el valor de la dirección IP de destino que se proporciona al guion.

¿Qué cree que sucederá si el valor no es una dirección IP legal?

```
RTA => Causará un error en Nmap y el script fallará.
```

8. Ahora edite el archivo de secuencia de comandos para ingresar los comandos que ejecutarán el análisis de Nmap. Utilice la variable **$1** para indicar la dirección IP del dispositivo de destino que desea escanear. Los resultados del análisis de Nmap se escribirán en un archivo denominado **scan_results.txt** en el directorio actual.

```
#!/bin/bash
# Check if IP of target is entered
if [ -z "$1" ]
then
    echo "Correct usage is ./recon.sh <IP>"
    exit 1
else
    echo "Running Nmap…"
# Run Nmap scan on target and save results to file
sudo nmap -p- -sSV -O --open --min-rate 5000 -vvv -n -Pn $1 > scan_results.txt
echo "Scan complete – results written to scan_results.txt"
fi
```

¿Qué tipo de análisis de Nmap se ejecutará con este script?

Este script realiza un escaneo agresivo y detallado con Nmap, ideal para reconocimiento inicial en entornos de pentesting.

Opción	Descripción
`-p-`	Escanea **todos los puertos** (1-65535).
`-sSV`	Realiza **detección de servicios y versiones**.
`-O`	Intenta **identificar el sistema operativo** del objetivo.
`--open`	Muestra **solo los puertos abiertos**.
`--min-rate 5000`	Establece una **tasa mínima de 5000 paquetes/segundo** (escaneo rápido).

Opción	Descripción
`-vvv`	Modo **muy detallado** (verbo alto).
`-n`	**No realiza resolución DNS**.
`-Pn`	**No realiza ping**, asume que el host está activo.

9. Vuelva a ejecutarlo con la dirección IP de destino proporcionada.

```
┌──(kali㉿Kali)-[~/Desktop]
└─$ ./recon.sh 10.6.6.23
Running Nmap…
[sudo] password for kali:
Host discovery disabled (-Pn). All addresses will be marked 'up' and scan times may be slower.
Scan complete – results written to scan_results.txt
```

10. Utilice el comando **cat** para ver el contenido del archivo **scan_results.txt** que creó con el script.

```
PORT    STATE SERVICE     REASON         VERSION
21/tcp  open  ftp         syn-ack ttl 64 vsftpd 3.0.3
22/tcp  open  ssh         syn-ack ttl 64 OpenSSH 7.9p1 Debian 10+deb10u2 (protocol 2.0)
53/tcp  open  domain      syn-ack ttl 64 ISC BIND 9.11.5-P4-5.1+deb10u5 (Debian Linux)
80/tcp  open  http        syn-ack ttl 64 nginx 1.14.2
139/tcp open  netbios-ssn syn-ack ttl 64 Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
445/tcp open  netbios-ssn syn-ack ttl 64 Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
MAC Address: 02:42:0A:06:06:17 (Unknown)
Device type: general purpose
Running: Linux 4.X|5.X
OS CPE: cpe:/o:linux:linux_kernel:4 cpe:/o:linux:linux_kernel:5
OS details: Linux 4.15 - 5.8
```

¿Qué puertos están abiertos en el destino?

```
RTA =>

 21/tcp  open  ftp
 22/tcp  open  ssh
 53/tcp  open  domain
 80/tcp  open  http
 139/tcp open  netbios-ssn
 445/tcp open  microsoft-ds
```

Paso 2: Modifique el script para enumerar los recursos compartidos en el destino.

Como se vio en el paso anterior, el destino en 10.6.6.23 tiene puertos abiertos que podrían indicar un servidor Samba. En este paso, editará el script para ejecutar **enum4linux** si un puerto de uso compartido de unidad Samba está abierto para determinar los recursos compartidos de unidad o las cuentas de usuario disponibles.

¿Qué indica que se está ejecutando un servidor Samba en los hosts?

```
RTA => Los puertos 139 y 445 abiertos.
```

1. Abra el archivo **recon.sh** en el editor de texto. Agregue los siguientes comandos:

```
1 #!/bin/bash
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 "Running Nmap…"
9 # Run Nmap scan on target and save results to file
10 sudo nmap -p- -sSV -O --open --min-rate 5000 -vvv -n -Pn $1 > scan_results.txt
11 echo "Scan complete – results written to scan_results.txt"
12 fi
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
23
```

2. Analice el código adicional.

- La línea 13 es un comentario.
- La línea 14 indica el inicio de una instrucción **if/then** que buscará en los resultados de Nmap el puerto abierto 445. El comando [**grep** busca líneas en el archivo que coincidan con el patrón “445 open”. El comando **grep** busca primero las líneas que coincidan con el patrón “445”. Luego, la salida se canaliza a un segundo comando **grep** para buscar nuevamente las líneas que coinciden con el patrón **open** (abierto). Con la opción **-i**, el comando **grep** ignora las distinciones entre mayúsculas y minúsculas en los patrones de búsqueda. La opción **-q** suprime las salidas estándar.
- La línea 15 es la cláusula "**then**". Contiene el comando que se ejecutará si la prueba **if** devuelve "true" (verdadero).
- Las líneas 16 a 19 se ejecutan si se encuentra el puerto de uso compartido de archivos SMB (445). La línea 16 ejecuta **enum4linux** con las opciones **-U** y **-S** en el host de destino especificado en **$1** y agrega los resultados al final de **scan_results.txt**. Las líneas 17, 18 y 19 muestran mensajes cuando emun4linux finalizó el análisis y proporciona instrucciones para ver los resultados.
- La línea 20 indica la acción a realizar si la condición falla.

- La línea 21 muestra un mensaje si el puerto de uso compartido de archivos SMB (445) no está abierto.
- En la línea 22 **, fi** significa el final de la cláusula **if/then**.

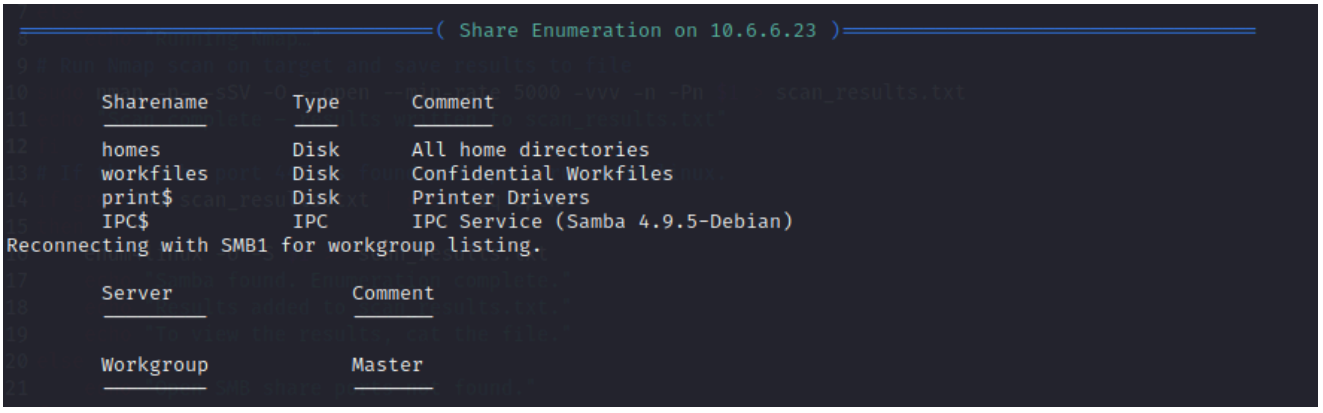
¿Qué hacen las opciones -U y -S en enum4linux?

```
RTA =>

-U enumera los usuarios encontrados
-S enumera los recursos compartidos de la unidad
```

3. Guarde el archivo **recon.sh** en el editor de texto y salga a la línea de comandos.
4. Vuelva a ejecutar el guion en el sistema de destino (10.6.6.23).

¿Qué recursos compartidos de archivos se encontraron en el destino?



Paso 3: Automatice Nmap desde la línea de comandos.

Otra forma de automatizar Nmap es escanear un grupo de objetivos específicos que se especifican en un archivo externo.

1. Cree un nuevo archivo en el Mousepad y escriba las direcciones IP de los hosts existentes en la red 10.6.6.0/24. Para enumerar todos los hosts disponibles con sus direcciones IP, ingrese el comando **containers** en una terminal.

```
Internal Hacking Network: 10.6.6.0/24
Your bridge networks:
br-internal      UP             10.6.6.1/24 fe80::42:f3ff:fe0d:b1e0/64
Your docker network: 172.17.0.0/16
docker0          UP             172.17.0.1/16 fe80::42:3ff:fef6:e1ec/64

The following are the available containers and associated IP addresses.
+-----------------+
|   IP Address    |
+-----------------+
|    10.6.6.11    |
|    10.6.6.12    |
|    10.6.6.13    |
|    10.6.6.14    |
|    10.6.6.23    |
|   172.17.0.2    |
+-----------------+
```

Asegúrese de que las direcciones IP estén separadas con un espacio o enumere cada dirección IP en una línea separada.

2. Guarde el archivo con el nombre **to_scan.txt .**

```
1 10.6.6.11
2 10.6.6.12
3 10.6.6.13
4 10.6.6.14
5 10.6.6.23
6
```

3. En el indicador, ingrese el comando para ejecutar Nmap con los destinos del archivo. Para los fines de esta práctica de laboratorio, solo se ejecutará un análisis de ping simple, pero cualquier tipo de análisis que tome una dirección IP como destino se puede ejecutar de esta manera.

```
┌──(kali㉿Kali)-[~/Desktop]
└─$ nmap -sn -iL to_scan.txt
Starting Nmap 7.94 ( https://nmap.org ) at 2025-07-22 04:20 UTC
Nmap scan report for webgoat.vm (10.6.6.11)
Host is up (0.00081s latency).
Nmap scan report for juice-shop.vm (10.6.6.12)
Host is up (0.0016s latency).
Nmap scan report for dvwa.vm (10.6.6.13)
Host is up (0.0013s latency).
Nmap scan report for mutillidae.vm (10.6.6.14)
Host is up (0.0011s latency).
Nmap scan report for gravemind.vm (10.6.6.23)
Host is up (0.00098s latency).
Nmap done: 5 IP addresses (5 hosts up) scanned in 0.00 seconds
```

4. Después de una breve demora, debería ver que Nmap genera los informes de análisis para cada host que se especificó en el archivo **to_scan.txt**.

Nota: El archivo to_scan.txt no requiere permisos de ejecución porque sirve como archivo de datos, no como archivo de secuencia de comandos.

Parte 2: Diferenciar entre scripts escritos en Bash, Python, Ruby y PowerShell

En esta parte, usará lo que aprendió en la parte anterior sobre la escritura y el análisis de un script de Bash para analizar scripts escritos previamente. Saber qué lenguaje de secuencias de comandos se utiliza en las secuencias de comandos que descubre durante las pruebas de penetración le permite comprender el propósito de la secuencia de comandos y, potencialmente, poder modificarla para obtener información adicional.

Utilice este gráfico que ilustra las diferentes características de sintaxis de los lenguajes de secuencias de comandos.

Función	Python	BASH	Ruby	PowerShell
Ejemplo de Shebang: línea de comentario especial en la parte superior del guion que identifica la ruta al intérprete	`#!/usr/bin/python3`	`#!/bin/bash`	`#!/usr/local/bin/ruby`	`#!/usr/bin/env pwsh` Solo es necesario si se ejecuta PS en Linux, no es necesario en Windows

Carga de módulos	`import libraryName as alias`
	
	`from libraryName import subModule`
		n/a; Bash no requiere cargar módulos	Requiere *'libraryName'*
			
			Las bibliotecas autónomas se denominan 'gemas'
				n/a; PowerShell no requiere cargar módulos

Definición de variables	`variableName = variableValue`	`variableName = variableValue` A las variables leídas como opciones desde la CLI se les asignan $1, $2, ..., $n	`variableName = variableValue` No puede comenzar con un número o una letra mayúscula.	`$varvariableName = varvariableValue`
Variables de llamada	*`variableName`* Ejemplo: `print(varibleName)`	*`$variableName`* Ejemplo: `echo $variableName`	*`#variableName`* Ejemplo: `puts variableName`	*`$varibleName`* Ejemplo: `PS C:\> $variableName`

Comparación	Utiliza símbolos aritméticos
	Igual es **==**
	No es igual es **!=**
	Mayor que es **>**
	Igual o mayor que **>=**
	Menor que es **<**
	Igual o menor que es **<=**
		Usos alfa:
		Igual a **-eq**
		No igual a **-ne**
		Mayor que **-gt**
		Mayor o igual a **-ge**
		Menor que **-lt**
		Menor o igual que **-le**
		
		Ejemplo:
		`$x -gt 8`
		
		Si utiliza símbolos aritméticos, escríbalos entre paréntesis dobles.
		`(($a > $b))`
			Utiliza símbolos aritméticos
			Igual es **==**
			No es igual es **!=**
			Mayor que es **>**
			Igual o mayor que **>=**
			Menor que es **<**
			Igual o menor que es **<=**
				Utiliza una variedad de operadores:
				Igual a:
				-eq, **-ieq**, **-ceq**
				No igual a:
				-ne, **-ine**, **-cne**
				Mayor que:
				-gt, **-igt**, **-cgt**
				Mayor o igual que
				-ge, **-ige**, **-cge**
				Menor que:
				-lt, **-ilt**, **-clt**
				Menor o igual que:
				-le, **-ile**, **-cle**

Condiciones If	`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 se pueda usar si solo se marca Si una condición es "not true"
				`if (condition1) {`
				`  action1`
				`}`
				`elseif (condition2) {`
				`  action2`
				`}`
				`else {`
				`  action3`
				`}`

Bucles Do While	Ejemplo:
	`i= 1`
	`while i < 6:`
	`  print(i)`
	`i = I + 1`
	`print ("All done")`
		Ejemplo:
		`x=1`
		`while [ $x -le 5 ]`
		`do`
		`echo "count " $x`
		`x=$(( $x + 1 ))`
		`done`
			Ejemplo:
			`while x >= 1`
			`  puts #@x`
			`x = x - 1`
			`end`
				Ejemplo:
				`do {`
				`  Write-Host $x`
				`  $x =$ x--`
				`} while ($x -ge 1)`

1. Revise el ejemplo de código que se muestra. Utilice las características de sintaxis para determinar qué lenguaje de secuencias de comandos se utiliza para interpretar el código.

```
1  import nmap

2  # take the range of ports to

3  # be scanned

4  begin = 21

5  end = 80

6 target = '10.6.6.23'

7  # scan the port range

8  for i in range(begin,end+1):

9   results = nmap.PortScanner(target,str(i))

10  results = results['scan'][target]['tcp'][i]['state']

11  print('Port {i} is {results}.')
```

Usando la información del gráfico, ¿Cuál debería ser la primera línea de código?

```
RTA => #!/usr/bin/python3
```

¿Qué rango de puertos se analizará?

```
RTA => TCP 21 al 80
```

2. Revise el ejemplo de código que se muestra. Utilice las características de sintaxis para determinar qué lenguaje de secuencias de comandos se utiliza para interpretar el código.

```
1  require 'nmap/command'

2  Nmap::Command.sudo do |nmap|

3    nmap.syn_scan       = true

4    nmap.os_fingerprint = true

5    nmap.service_scan   = true

6    nmap.output_xml     = 'scan.xml'

7    nmap.verbose        = true

8    nmap.ports   = [20, 21, 22, 23, 25, 80, 110, 443, 512, 522, 8080, 1080]

9    nmap.targets = '10.6.6.*'

10 end

11 #Parse Nmap XML scan files:

12 require 'nmap/xml'

13 Nmap::XML.open('scan.xml') do |xml|

14   xml.each_host do |host|

15     puts "[#{host.ip}]"

16     host.each_port do |port|
```

```
17       puts "  #{port.number}/#{port.protocol}    #{port.state}   #
{port.service}"

18     end

19   end

20 end
```

Con la información del gráfico, ¿Qué intérprete de lenguaje de secuencias de comandos se usará para ejecutar este código?

```
RTA => Ruby
```

¿Cuál es el objetivo de este análisis de Nmap?

```
RTA => La subred 10.6.6.0/24
```

3. Revise el ejemplo de código que se muestra. Utilice las características de sintaxis para determinar qué lenguaje de secuencias de comandos se utiliza para interpretar el código.

```
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  }
```

Con la información del gráfico, ¿Qué intérprete de lenguaje de secuencias de comandos se usará para ejecutar este código?

```
RTA => PowerShell
```

¿Qué opciones utilizará Nmap para el análisis y qué significan esas opciones?

```
RTA =>

-p => lista de puertos a escanear
-oX => salida a un archivo xml
-A => escaneo agresivo
-v => salida detallada (Nivel 1/3)
```

Pregunta de reflexión

Las habilidades de análisis de código se prueban en certificaciones de pruebas de penetración. ¿Qué beneficio ofrece tener habilidades de análisis de código a los pentesters al descubrir vulnerabilidades?

```
RTA => Las habilidades de análisis de código permiten a los pentesters
identificar vulnerabilidades que no son evidentes desde el exterior, como
fallos lógicos, validaciones débiles o funciones inseguras.
Comprender el flujo interno de una aplicación les ayuda a diseñar exploits
más precisos y efectivos. También pueden detectar backdoors, credenciales
hardcodeadas o condiciones especiales que facilitan el acceso no autorizado.
Este conocimiento mejora el uso de herramientas como Burp Suite o fuzzers,
al permitir configuraciones más dirigidas.
Además, pueden evaluar la calidad del manejo de sesiones, cifrado y control
de acceso. Esto reduce la dependencia de escaneos automáticos y aumenta la
profundidad del análisis.
En entornos con acceso al código fuente, el pentester se convierte en un
auditor técnico más completo. Incluso en pruebas de caja negra, entender
cómo se codifican ciertas funciones ayuda a inferir posibles fallos.
Básicamente, el análisis de código potencia la precisión, creatividad y
efectividad del proceso de evaluación.
```