Ejemplo: BASH

Tuesday, September 22, 2020 11:06 AM

Scanner de puertos abiertos

El siguiente código se encuentra guardado bajo el nombre de portScanner en un entorno Linux.

```
1  #!/bin/bash
2  #Ejemplo de scanner de puertos
3  #Obtenido de pentestlab.wordpress.com
4  #Parámetros
5  host=$1
6  firstport=$2
7  lastport=$3
8  #Funcion para escanear puertos
9  function portscan {
10 for ((counter=$firstport; counter<=$lastport; counter++))
11 do
12     (echo >/dev/tcp/$host/$counter) > /dev/null 2>&1 && echo "$counter open"
13 done
14 }
15 #llamada de la funcion
16 portscan
```

Análisis de código

- La primera línea es la shebang.
- De la línea 5 a la 7 tenemos los parámetros que recibe el script, ya que cada uno de ellos toma automáticamente los nombres de $1, $2, $3, etc.

  Ejemplo de ejecución de script:
  portScanner 192.168.1.1 10 100
  Por lo que los parámetros son:
  - $1 = 192.168.1.1
  - $2 = 10
  - $3 = 100
- De la línea 9 a la 14 tenemos la definición de la función ***portscan***.
- De la línea 10 a la 13 tenemos el bloque de código del ciclo for. El ciclo for va desde el primer puerto (indicado como el parámetro 2) hasta el último puerto (indicado como el parámetro 3), haciendo saltos de 1 en 1.
- En la línea 12 tenemos el código principal de nuestro script, el cual explicaremos más a detalle.
- La línea 16 hace la llamada a la función **portscan**, para que se ejecute todo el código contenido en la función.

Análisis de la línea 12

Sabemos que **echo** hace referencia a una impresión, por lo que esa línea desplegará en la terminal el resultado de los otros segmentos de código que la componente.
El operador **>**, hace referencia a una operación de redirección, así que los resultados se irán redirigiendo a lo largo de los comandos.
La ruta **/dev/tcp/$host/$counter** podemos verla como la acción que hará momentáneamente el SO para comunicarse a esa IP a través del puerto indicado por la variable counter.
La ruta **/dev/null** es a donde podemos redirigir los errores o algunos mensajes del sistema, sería el equivalente a la variable null de PowerShell.
En BASH, el número 1 representa la salida estándar (stdout) y el 2, la salida de errores (stderr), por lo que podemos interpretar que **2>&1** es la acción de redirigir los errores a la salida normal, por lo que la instrucción completa **/dev/null 2>&1** se interpreta como que la salida de error es dirigida a la salida estándar y, a su vez, toda la salida estándar es dirigida al archivo /dev/null, por lo que se descartan todos los mensajes de error.
Entonces, interpretando el comando completo:

- Se intente redirigir el comando echo a la ruta de la IP y puerto mencionados, en caso de que el puerto esté cerrado, se generará un mensaje de error indicando que la conexión fue rechazada (Connection refused).
- En caso de que el puerto esté cerrado, internamente ese comando (hasta antes de && echo) devolverá FALSE, si el puerto está abierto, TRUE.

- La instrucción echo "$counter open", devuelve por default TRUE, ya que es una operación simple de salida y en condiciones normales no marcará error.
- Por lo que, si el puerto está abierto, tendremos TRUE && TRUE, lo cual es TRUE, y por ende se desplegará en la terminal que el puerto analizado está "open".
- Sin embargo, si el puerto está cerrado, tendremos FALSE && TRUE, lo cual es FALSE, por lo que no se mostrará nada en pantalla.

Scanner de IP activas en la red (que responden ping)

El siguiente código se encuentra guardado bajo el nombre de netScan en un entorno Linux.

```
1  #!/bin/bash
2  function is_alive_ping() {
3    ping -c 1 $1 > /dev/null 2>&1
4    [ $? -eq 0 ] && echo "Node with IP: $i is up."
5  }
6  for i in 192.168.100.{1..255}
7  do
8  is_alive_ping $i & disown
9  done
```

Análisis de código

- La línea 1 es el shebang
- De la línea 2 a la 5 es la definición de la función.
- En la línea 3 tenemos la instrucción ping -c 1 $1, la cual hace un ping a $1, que es el primer parámetro enviado a la función que, en este caso, representa la IP.
  - Recordemos que un ping es un paquete de información de prueba que enviamos para verificar si el otro equipo responde nuestros intentos de comunicación.
  - El parámetro -c nos define la cantidad de paquetes que enviaremos, para fines de ejemplo con 1 es suficiente.
- Leyendo la instrucción completa, lo que hacemos es enviar los pings a cada IP pero todos los mensajes que usualmente despliega ping, así como los mensajes de error y advertencia, son enviados a /dev/null para que no se vean en la consola.
- En la línea 4 se hace la revisión de la condición $? -eq 0, para entender esto consideremos que $? Nos indica el estatus de salida del último comando ejecutado, es decir, si el comando anterior funcionó, tendrá un valor de 0, sino de 1.
- La condición de la línea 4 es unida con el operador && a la instrucción echo "Mensaje", por lo que si el ping fue exitoso, veremos en pantalla el mensaje de que dicha IP está arriba (up) y, sino, no veremos nada.
- De la línea 6 a la 9 es un ciclo for, el cual dará un valor a la variable i desde 1 hasta 255, lo que irá completando la dirección IP. Es importante que cambiemos el valor de la IP por la que corresponda a nuestra red.
- Dentro del for, en la línea 8, hacemos el llamado a la ejecución y usamos el comando disown para desacoplar los procesos, es decir, poder comenzar la siguiente ejecución y hacer un poco más veloz el código. El quitar el & disown del código hare nuestro script un poco más lento.