5. Bash Scripting

勞動部 產業人才投資計畫
中國文化大學 推廣教育部
張耀鴻 副教授
2022年 暑期班

綱要

- Bash script簡介
- 變數
- If, Else, Elif
- 布林邏輯運算
- 變數
- 函式
- 實例演練

Bash script簡介

- Bash script的副檔名為.sh
- 檔案第1行為#!/bin/bash
- 註解行以#開頭

```
┌──(kali㉿kali)-[~]
└─$ cat ./hello-world.sh
#!/bin/bash
# Hello World Bash Script
echo "Hello World!"
```

- 第 1 行：#!稱為 shebang, 通常被 Bash 解釋器忽略。 /bin/bash 是解釋器的絕對路徑，用於執行“Bash script”
- 第 2 行：# 為註解，後面的所有文字會被忽略
- 第 3 行：回顯“Hello World！”使用 echo Linux 命令將給定的字串列印到終端機

製作可執行script

- chmod 命令連同+x 選項用於使script可執行
- ./hellowortd.sh 用於執行script
- ./ 符號只是一個路徑符號，表示此script位於當前目錄中
- 每當我們鍵入命令時，Bash 都會嘗試在PATH變數的一系列目錄中找到可執行檔的位置

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

┌──(kali㉿kali)-[~]
└─$ ./hello-world.sh
Hello World!
```

變數

- 我們可以透過多種方式宣告變數值
- 最簡單的方法是用 name=value 宣告直接設置變數值
- “=” 號之前或之後不能有空格：

```
┌──(kali㉿kali)-[~]
└─$ first_name=Good
```

- 使用變數時在前面加上 “$”
- 變數名稱有區分大小寫

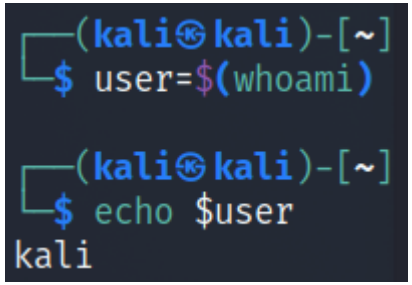
```
┌──(kali㉿kali)-[~]
└─$ first_name=Good

┌──(kali㉿kali)-[~]
└─$ last_name=Hacker

┌──(kali㉿kali)-[~]
└─$ echo $first_name $last_name
Good Hacker
```

命令替換

- 我們還可以將變數的值設置為命令或程序的結果，這稱為命令替換
- 可讓我們取得命令或程序的輸出（通常會列印到屏幕上）並將其儲存為變數的值



引數

- 並非所有 Bash script都需要引數
- 了解 Bash 如何解讀和使用引數非常重要
- 例如，命令 ls -l /var/log ，-l 和 /var/log 都是 ls 命令的引數
- 我們可以提供命令列引數並在script中使用它們：

```
┌──(kali㉿kali)-[~]
└─$ cat ./arg.sh
#!/bin/bash
echo "The first two arguments are $1 and $2"

┌──(kali㉿kali)-[~]
└─$ chmod +x ./arg.sh

┌──(kali㉿kali)-[~]
└─$ ./arg.sh hello there
The first two arguments are hello and there
```

特殊Bash變數

變數名稱	說明
$0	Bash script檔名
$1 - $9	命令列引數
$#	傳入Bash script的引數個數
$@	所有傳入Bash script的引數
$?	上一個程序的結束狀態
$$	目前script的PID
$USER	執行script的使用者
$HOSTNAME	主機名稱
$RANDOM	隨機數字
#LINENO	在script中的第幾行

讀取使用者輸入

```
┌──(kali㉿kali)-[~]
└─$ cat ./input.sh
#!/bin/bash
echo "Hello there, would you like to learn how to hack: Y/N?"
read answer
echo "Your answer was $answer"

┌──(kali㉿kali)-[~]
└─$ chmod +x ./input.sh

┌──(kali㉿kali)-[~]
└─$ ./input.sh
Hello there, would you like to learn how to hack: Y/N?
Y
Your answer was Y
```

read的命令列選項

- -p 允許我們指定提示
- -s 允許無聲輸入，是取得使用者密碼的理想選擇：

```
┌──(kali㉿kali)-[~]
└─$ cat ./input2.sh
#!/bin/bash
# Prompt the user for credentials
read -p ' Username: ' username
read -sp 'Password: ' password
echo "Thanks, your creds are as follows: "$username" and" $pas
sword

┌──(kali㉿kali)-[~]
└─$ chmod +x ./input2.sh

┌──(kali㉿kali)-[~]
└─$ ./input2.sh
 Username: kali
Password: Thanks, your creds are as follows: kali and hello
```

If, Else, Elif

- 條件敘述允許我們根據不同的條件執行不同的操作。最常見的條件 Bash 敘述包括 if、else 和 elif
- if 敘述檢查條件是否為真，請特別注意所需空格的用法：

```
┌──(kali㉿kali)-[~]
└─$ cat ./if.sh
#!/bin/bash
# if statement example
read -p "What is your age: " age
if [ $age -lt 16 ]
then
  echo "You might need parental permission to take this course
!"
fi

┌──(kali㉿kali)-[~]
└─$ chmod +x ./if.sh

┌──(kali㉿kali)-[~]
└─$ ./if.sh
What is your age: 15
You might need parental permission to take this course!
```

常見的test指令運算子 1/2

運算子	說明：若成立則運算式為真
!EXPRESSION	EXPRESSION為false
-n STRING	STRING長度大於0
-z STRING	STRING長度為0
STRING1 != STRING2	STRING1不等於STRING2
STRING1 == STRING2	STRING1等於STRING2
INT1 –eq INT2	INT1等於INT2
INT1 –ne INT2	INT1不等於INT2
INT1 –gt INT2	INT1 > INT2
INT1 –lt INT2	INT1 < INT2
INT1 –ge INT2	INT1 >= INT2
INT1 –le INT2	INT1 <= INT2

常見的test指令運算子 2/2

運算子	說明: 若成立則運算式為真
-d FILE	FILE為目錄
-e FILE	FILE存在
-r FILE	FILE為可讀
-s FILE	FILE內容不為空
-w FILE	FILE為可寫
-x FILE	FILE為可執行

在if敘述中使用test命令

```
┌──(kali㉿kali)-[~]
└─$ cat ./if2.sh
#!/bin/bash
# if statement example 2
read -p "What is your age: " age
if test $age -lt 16
then
  echo "You might need parental permission to take this course
!"
fi

┌──(kali㉿kali)-[~]
└─$ chmod +x ./if2.sh

┌──(kali㉿kali)-[~]
└─$ ./if2.sh
What is your age: 15
You might need parental permission to take this course!
```

if-then-else敘述

```
┌──(kali㉿kali)-[~]
└─$ cat ./else.sh
#!/bin/bash
# else statement example
read -p "What is your age: " age
if [ $age -lt 16 ]
then
  echo "You might need parental permission to take this course
!"
else
  echo "Welcome to the course!"
fi

┌──(kali㉿kali)-[~]
└─$ chmod +x ./else.sh

┌──(kali㉿kali)-[~]
└─$ ./else.sh
What is your age: 21
Welcome to the course!
```

if-then-elif敘述

```
┌──(kali㉿kali)-[~]
└─$ cat ./elif.sh
# elif example
read -p "What is your age: " age
if [ $age -lt 16 ]
then
  echo "You might need parental permission to take this course
!"
elif [ $age -gt 60 ]
then
  echo "Hats off to you, respect!"
else
  echo "Welcome to the course!"
fi

┌──(kali㉿kali)-[~]
└─$ chmod +x ./elif.sh

┌──(kali㉿kali)-[~]
└─$ ./elif.sh
What is your age: 65
Hats off to you, respect!
```

布林邏輯運算

- Bash 以多種方式使用像 AND (&&) 和 OR (||)的布林運算子
- 一種常見的用法是在command list中，由運算子控制流程的命令鏈
- "|"（管線）符號是command list中常用的運算子，它將一個命令的輸出傳給另一個命令的輸入
- 布林邏輯運算子則根據前一個命令是成功（傳回 True 或 0）還是失敗（傳回 False 或非零）來執行命令

&&執行例

- 首先將要搜尋的使用者名稱指派給 user2 變數
- 接下來用 grep 命令檢查 /etc/passwd 檔中是否有這個使用者
- 如果有的話，則 grep 傳回 True 並執行 echo 命令
- 但是如果使用者不存在，則 echo 命令不會執行

```
┌──(kali㉿kali)-[~]
└─$ export user2=kali

┌──(kali㉿kali)-[~]
└─$ grep $user2 /etc/passwd && echo "$user2 found\!"
kali:x:1000:1000:Kali,,,:/home/kali:/usr/bin/zsh
kali found!

┌──(kali㉿kali)-[~]
└─$ export user2=bob

┌──(kali㉿kali)-[~]
└─$ grep $user2 /etc/passwd && echo "$user2 found\!"
```

||執行例

- 當在command list使用OR (||) 運算子時，只有在上一個命令失敗（傳回 false 或非零）時，才會執行下一個命令：

```
┌──(kali㉿kali)-[~]
└─$ echo $user2                                                    1 ×
bob

┌──(kali㉿kali)-[~]
└─$ grep $user2 /etc/passwd && echo "$user2 found\!" || echo "$user2
 not found\!"
bob not found!
```

- 我們在之前的命令後面，再添加了 OR (||) 運算子，然後是第二個 echo 命令
- 現在，當 grep 找不到匹配的使用者名稱並傳回 False 時，才會執行 OR (||) 運算子之後的第二個 echo 命令

用&&測試多個條件

```
┌──(kali㉿kali)-[~]
└─$ cat ./and.sh
#!/bin/bash
# and example
if [ $USER == 'kali' ] && [ $(hostname) == 'kali' ]
then
  echo "Multiple statements are true!"
else
  echo "Not much to see here ... "
fi

┌──(kali㉿kali)-[~]
└─$ chmod +x ./and.sh

┌──(kali㉿kali)-[~]
└─$ ./and.sh
Multiple statements are true!

┌──(kali㉿kali)-[~]
└─$ echo $USER && echo $(hostname)
kali
kali
```

用||測試多個條件

```
┌──(kali㉿kali)-[~]
└─$ cat ./or.sh
#!/bin/bash
# or example
if [ $USER == 'kali' ] || [ $(hostname) == 'pwn' ]
then
  echo "One condition is true, this line is printed"
else
  echo "You are out of luck!"
fi

┌──(kali㉿kali)-[~]
└─$ chmod +x ./or.sh

┌──(kali㉿kali)-[~]
└─$ ./or.sh
One condition is true, this line is printed

┌──(kali㉿kali)-[~]
└─$ echo $USER && echo $(hostname)
kali
kali
```

迴圈

- 迴圈可幫助我們完成重複性任務，對於滲透測試特別有用
- 在 Bash 中，兩個最主要的迴圈命令是 for 和 while
- For 迴圈非常實用，並且很適合用在 Bash one-liners 中，為列表中的每個項目執行一組給定的命令
- While 迴圈當邏輯運算式為真時會執行程式碼，While 迴圈的格式，和 if 一樣，使用方括號 ([]) 進行測試

for迴圈

- 快速列印 10.0.2.0/24 子網域中的前 10 個 IP 位址，兩種寫法

```
┌──(kali㉿kali)-[~]
└─$ for ip in $(seq 1 10); do echo 10.0.2.$ip; done
10.0.2.1
10.0.2.2
10.0.2.3
10.0.2.4
10.0.2.5
10.0.2.6
10.0.2.7
10.0.2.8
10.0.2.9
10.0.2.10
```

```
┌──(kali㉿kali)-[~]
└─$ for ip in {1..10}; do echo 10.0.2.$ip; done
10.0.2.1
10.0.2.2
10.0.2.3
10.0.2.4
10.0.2.5
10.0.2.6
10.0.2.7
10.0.2.8
10.0.2.9
10.0.2.10
```

for迴圈的應用

- 在螢幕上顯示 IP 位址看起來可能不是很有用，但我們可以使用相同的迴圈以 nmap進行開放埠掃描
- 還可以嘗試用 ping 命令查看是否有任何 IP 位址回應 ICMP 回顯請求等

while迴圈

- 前述例子改用while迴圈列印 10.0.2.0/24 子網域中的前 10 個 IP 位址

```
┌──(kali㉿kali)-[~]
└─$ cat ./while.sh
#!/bin/bash
# while loop example
counter=1
while [ $counter -le 10 ]
do
  echo "10.0.2.$counter"
((counter++))
done

┌──(kali㉿kali)-[~]
└─$ chmod +x ./while.sh

┌──(kali㉿kali)-[~]
└─$ ./while.sh
10.0.2.1
10.0.2.2
10.0.2.3
10.0.2.4
10.0.2.5
10.0.2.6
10.0.2.7
10.0.2.8
10.0.2.9
10.0.2.10
```

函式

- Bash script的函式可視為script中的script，這在我們需要在script中多次執行相同的程式碼時很有用
- 換句話說，函式是一個子程序，或者是一個實現一組操作的程式區塊，一個執行特定任務的“黑盒子”
- 函式可以用兩種不同的格式編寫。第一種格式在Bash script中較常見，第2種格式對C語言程式設計師而言會比較熟悉：

```
function function_name {
  commands…
}
```

```
function_name () {
  commands…
}
```

函式參數傳遞

```
┌──(kali㉿kali)-[~]
└─$ cat ./func.sh
#!/bin/bash
# function example
print_me () {
  echo "You have been printed!"
}
print_me

┌──(kali㉿kali)-[~]
└─$ chmod +x ./func.sh

┌──(kali㉿kali)-[~]
└─$ ./func.sh
You have been printed!

┌──(kali㉿kali)-[~]
└─$
```

```
┌──(kali㉿kali)-[~]
└─$ cat ./funcarg.sh
#!/bin/bash
# passing arguments to functions
pass_arg() {
  echo "Today's random number is: $1"
}
pass_arg $RANDOM

┌──(kali㉿kali)-[~]
└─$ chmod +x ./funcarg.sh

┌──(kali㉿kali)-[~]
└─$ ./funcarg.sh
Today's random number is: 10137
```

函式傳回值

```
┌──(kali㉿kali)-[~]
└─$ cat ./funcvalue.sh
#!/bin/bash
# function return value example
return_me() {
  echo "Oh hello there, I'm returning a random value!"
  return $RANDOM
}

return_me

echo "The previous function returned a value of $?"

┌──(kali㉿kali)-[~]
└─$ chmod +x ./funcvalue.sh

┌──(kali㉿kali)-[~]
└─$ ./funcvalue.sh
Oh hello there, I'm returning a random value!
The previous function returned a value of 237
```

變數範圍

- 預設的變數範圍具有全域範圍，也就是可以在整個script中存取它
- 區域變數只能在定義它的函式、程式區塊或子shell中看到
- 我們可以在全域變數前加上 local 關鍵字，改成區域變數

變數範圍範例

```
┌──(kali㉿kali)-[~]
└─$ cat ./varscope.sh
#!/bin/bash
# var scope example

name1="John"
name2="Jason"

name_change() {
  local name1="Edward"
  echo "Inside of this function, name1 is $name1 and name2 is $name2"
  name2="Lucas"
}

echo "Before the function call, name1 is $name1 and name2 is $name2"

name_change

echo "After the function call, name1 is $name1 and name2 is $name2"

┌──(kali㉿kali)-[~]
└─$ chmod +x ./varscope.sh

┌──(kali㉿kali)-[~]
└─$ ./varscope.sh
Before the function call, name1 is John and name2 is Jason
Inside of this function, name1 is Edward and name2 is Jason
After the function call, name1 is John and name2 is Lucas
```

實例演練 1

- GOAL: 查找 megacorpone.com 主網頁上列出的所有子網域，並找到它們對應的 IP 位址
- 首先用wget取得index.html

```
┌──(kali㉿kali)-[~]
└─$ wget www.megacorpone.com
--2022-03-23 23:05:52--  http://www.megacorpone.com/
Resolving www.megacorpone.com (www.megacorpone.com)... 149.56.244.87
Connecting to www.megacorpone.com (www.megacorpone.com)|149.56.244.87|:80..
. connected.
HTTP request sent, awaiting response... 200 OK
Length: 14603 (14K) [text/html]
Saving to: 'index.html'

index.html          100%[===================>]  14.26K  --.-KB/s    in 0.06s

2022-03-23 23:05:54 (229 KB/s) - 'index.html' saved [14603/14603]


┌──(kali㉿kali)-[~]
└─$ ls -l index.html
-rw-r--r-- 1 kali kali 14603 Nov  6  2019 index.html
```

- 用 grep “href=” 來擷取 index.html 中所有包含 HTML 連結的行：

```
┌──(kali㉿kali)-[~]
└─$ grep "href=" index.html
    <link rel="shortcut icon" href="assets/ico/favicon.ico">
    <link href="assets/css/bootstrap.css" rel="stylesheet">
    <link href="assets/css/style.css" rel="stylesheet">
```

- 用 grep 縮小搜尋範圍

```
┌──(kali㉿kali)-[~]
└─$ grep "href=" index.html | grep "\.megacorpone" | grep -v "www\.megacorp
one\.com" | head
                        <li><a href="http://support.megacorpone.com">SUPPORT</a
></li>
                        <li><a href="http://intranet.megacorpone.com">LOG IN</a
></li>
                        <!--<li><a href="https://cp.megacorpone.net/">LOG IN</a
></li>-->
```

- 看起來每一行都包含一個超連結和一個子網域，但我們需要擺脫鏈接周圍的額外 HTML
- 我們將使用 awk 的 -F 選項設置多字元分隔符號

- 我們要把http://設為分隔符號，並告訴 awk 我們想要第二個欄位（ ‘{print $2}’ ），或該分隔符號之後的所有內容：

```
└─$ grep "href=" index.html | grep "\.megacorpone" | grep -v "www\.megacorp
one\.com" | awk -F "http://" '{print $2}'
support.megacorpone.com">SUPPORT</a></li>
intranet.megacorpone.com">LOG IN</a></li>

admin.megacorpone.com/admin/index.html">Cell Regeneration</a></li>
intranet.megacorpone.com/pear/">Immune Systems Supplements</a></li>
```

- 接著用 cut 將分隔符設為“/”（使用 -d）並列印第一個欄位（使用 -f 1），只留下完整的子網域名稱：

```
└─$ grep "href=" index.html | grep "\.megacorpone" | grep -v "www\.megacorp
one\.com" | awk -F "http://" '{print $2}' | cut -d "/" -f 1
support.megacorpone.com">SUPPORT<
intranet.megacorpone.com">LOG IN<

admin.megacorpone.com
intranet.megacorpone.com
```

- 問題：語法太長、太麻煩

- 改進：用正規表達式從檔案中分割出“.megacorpone.com”子網域：

```
┌──(kali㉿kali)-[~]
└─$ grep -o '[^/]*\.megacorpone\.com' index.html | sort -u > list.txt

┌──(kali㉿kali)-[~]
└─$ cat list.txt
admin.megacorpone.com
beta.megacorpone.com
intranet.megacorpone.com
```

- 我們正在搜尋的字串（‘[^ /]* \. megacorpone \.com’）被包裹在單引號中，所以不允許變數擴展，並將按字面意思處理所有包含的符號
- 表達式中的第一個區塊([^/] *)是一個否定 (^) 集合 ([ ])，它搜尋不包括斜線的任意個數的符號 (*)
- 句點用倒斜線 (\.) 轉義，以強調我們正在尋找句點
- 接下來，字串必須以“.megacorpone.com” 結尾，當 grep 找到匹配的字串時，將從行中分割出來並將結果傳回

- 有了來自 megacorpone.com 首頁的網域名稱列表連結之後，接下來用host在文字檔中找出每個域名對應的IP位址
- 用 Bash 單行迴圈的寫法如下：

```
┌──(kali㉿kali)-[~]
└─$ for url in $(cat list.txt); do host $url; done
admin.megacorpone.com has address 51.222.169.208
beta.megacorpone.com has address 51.222.169.209
intranet.megacorpone.com has address 51.222.169.211
mail2.megacorpone.com has address 51.222.169.213
```

- 並非所有host 命令的輸出都是相關的，我們將通過piping將輸出傳到 “has address” 的 grep 中來擷取 IP 位址，然後剪切結果並進行排序：

```
└─$ for url in $(cat list.txt); do host $url; done | grep "has address" | c
ut -d " " -f 4 | sort -u
149.56.244.87
51.222.169.208
51.222.169.209
51.222.169.211
51.222.169.212
```

實例演練 2

- [GOAL]：假設我們正在進行滲透測試並且對 Windows 機器具有非特權存取權限。隨著我們繼續收集資訊，我們意識到它可能容易受到我們讀到的以字母 a、f 和 d 開頭的漏洞利用的攻擊，但我們不記得漏洞利用的全名。為了提升我們的權限，我們想要搜尋那個特定的漏洞
- 為此，我們需要在 https://www.exploit-db.com 中搜尋 “afd windows” ，下載符合我們搜尋條件的漏洞，並檢查它們直到找到合適的漏洞
- 我們要寫一個 Bash script，用來搜尋並在以後自動下載漏洞利用程式

- SearchSploit 是一個用於 Exploit-DB 的命令列搜尋工具，允許我們隨時地取得 Exploit Database的離線副本
- 我們將傳遞 “afd windows” 作為搜尋字串，使用 -w 以傳回https://www.exploitdb .com 的URL，使用 -t 搜尋漏洞標題：

```
└─$ searchsploit afd windows -w -t
 Exploit Title                                  |  URL
-------------------------------------------------- ---------------------------------------------
Microsoft Windows (x86) - 'af                   | https://www.exploit-db.com/exploits/40564
Microsoft Windows - 'afd.sys'                   | https://www.exploit-db.com/exploits/18755
Microsoft Windows - 'AfdJoinL                   | https://www.exploit-db.com/exploits/21844
Microsoft Windows 7 (x64) - '                   | https://www.exploit-db.com/exploits/39525
Microsoft Windows 7 (x86) - '                   | https://www.exploit-db.com/exploits/39446
Microsoft Windows 7 Kernel -                    | https://www.exploit-db.com/exploits/42009
Microsoft Windows XP - 'afd.s                   | https://www.exploit-db.com/exploits/17133
Microsoft Windows XP/2003 - '                   | https://www.exploit-db.com/exploits/18176
Microsoft Windows XP/2003 - '                   | https://www.exploit-db.com/exploits/6757
-------------------------------------------------- ---------------------------------------------
Shellcodes: No Results
```

- 目前我們只對exploit的 URL 感興趣，所以我們用 grep 查找 “http” ，然後用cut剪切所需的內容
- 我們將用 “|” 欄位分隔符號提取第二個欄位：

```
└─$ searchsploit afd windows -w -t | grep http | cut -f 2 -d "|"
https://www.exploit-db.com/exploits/40564
https://www.exploit-db.com/exploits/18755
https://www.exploit-db.com/exploits/21844
https://www.exploit-db.com/exploits/39525
https://www.exploit-db.com/exploits/39446
```

- 有了每個exploit的 URL之後，可以用迴圈來下載檔案並將它們儲存在本地
- 然而，我們注意到每個頁面都有一個下載"原始"exploit程式碼的連結，這正是我們所要的
- 每個原始頁面（如 /exploits/40564）都連結到raw exploit（如 /raw/40564）

- 有了以上這些資訊，我們執行 sed ‘s/exploits/raw/’ 來修改下載 URL 並將其插入到 Bash one-liner中，以下載exploit的原始程式碼：

```
┌──(kali㉿kali)-[~]
└─$ for e in $(searchsploit afd windows -w -t | grep http | cut -f 2 -d "|"
); do exp_name=$(echo $e | cut -d "/" -f 5) && url=$(echo $e | sed 's/explo
its/raw/') && wget -q --no-check-certificate $url -O $exp_name; done
```

- 以上script用 for 迴圈遍歷我們抓取的 SearchSploit URL
- 在迴圈內部，我們將 exp_name 設為exploit的名稱（使用 grep、cut 和命令替換），將 url 設為原始下載位置（同樣使用 sed 和命令替換）
- 如果成功（&&），我們用 wget 抓取漏洞（在沒有憑證檢查的安靜模式下）將其儲存在本地
- 並將漏洞名稱作為本地檔名

- 用ls -l驗證檔案是否存在

```
└─$ ls -l
total 78392
-rw-r--r-- 1 kali kali     1363 Mar 24 01:56 17133
-rw-r--r-- 1 kali kali    12215 Mar 24 01:56 18176
-rw-r--r-- 1 kali kali     9698 Mar 24 01:56 18755
-rw-r--r-- 1 kali kali    11590 Mar 24 01:56 21844
-rw-r--r-- 1 kali kali    10575 Mar 24 01:56 39446
-rw-r--r-- 1 kali kali    14193 Mar 24 01:56 39525
-rw-r--r-- 1 kali kali    32674 Mar 24 01:56 40564
-rw-r--r-- 1 kali kali    12636 Mar 24 01:56 42009
-rw-r--r-- 1 kali kali      612 Mar 24 01:56 6757
```

- 用file驗證檔案是否為文字檔

```
┌──(kali㉿kali)-[~]
└─$ file 17133
17133: C source, ASCII text, with CRLF line terminators
```

- 檢查每個exploit

```
└─$ cat 17133
/////////////////////////////////////////////////////////////////
/
//
// Title: Microsoft Windows xp AFD.sys Local Kernel DoS Exploit
// Author: Lufeng Li of Neusoft Corporation
// Vendor: www.microsoft.com
// Vulnerable: Windows xp sp3
//
/////////////////////////////////////////////////////////////////
```

將所有步驟放在一個 Bash script

```
┌──(kali㉿kali)-[~]
└─$ cat ./dlsploits.sh
#!/bin/bash
# Bash script to search for a given exploit and downl oad all matches.
for e in $(searchsploit afd windows -w -t | grep http | cut -f 2 -d "|")
do
  exp_name=$(echo $e | cut -d "/" -f 5)
  url=$(echo $e | sed 's/exploits/raw/')
  wget -q --no-check-certificate $url -O $exp_name
done

┌──(kali㉿kali)-[~]
└─$ chmod +x ./dlsploits.sh

┌──(kali㉿kali)-[~]
└─$ ./dlsploits.sh
```

Exercise

1. 撰寫一個簡短的Bash script來對 10.0.2.0/24 的前20個目標 IP 範圍執行 ping 掃描，並輸出有回應的IP位址
2. 用 Python語言實作上述練習
3. 使用本單元中的實際範例來幫助您建立一個從 access_log.txt 中提取 JavaScript 檔名的 Bash script。確保檔名不包含路徑、唯一而且已排序 (下載網址: http://www.offensive-security.com/pwk-files/access_log.txt.gz)
4. 用Python語言重寫前面的練習。


END