第一章

指令列基礎

透過電腦的指令列介面，你可以更貼近其作業系統（OS）。作業系統中含有為數驚人的功能，它們都已歷經數十年的使用和部署，堪稱完備。可惜的是，藉由指令列與作業系統互動的能力幾乎已成絕響。它已逐漸被圖形使用者介面（GUI）所取代，後者以犧牲速度與彈性、同時讓使用者疏遠底層功能的方式，藉此提升易用度。

有效地運用指令列的能力，依舊是安全從業人員與管理員的必備技巧。像是 Metasploit、Nmap 和 Snort 等工具，都需要精通指令列才能操縱自如。在滲透測試時，你唯一能與遠端目標系統互動的方式就是指令列介面，尤其是在剛展開入侵的早期階段。

為了打好基礎，我們會先略述指令列及其元件；然後會探討如何運用它來加強你的網路安全功力。

何謂指令列？

綜觀本書，**指令列**（*command line*）一詞代表一套作業系統中所安裝的所有各種非 GUI 的可執行檔，以及 shell 中具備的內建功能、關鍵字、和指令稿功能，而 shell 就是指令列介面。

為了有效地運用指令列，你需要具備兩樣條件：了解既有指令的功能及選項、以及如何藉由指令稿語言將它們串連起來。

本書會介紹範圍橫跨 Linux 和 Windows 作業系統的 40 種以上指令、以及各種 shell 的內建功能和關鍵字。其中大部分的指令都源於 Linux 環境，但各位也會發現，在 Windows 平台上也有好幾種方式可以執行它們。

為何是 bash ？

基於要撰寫指令稿，我們選擇了 bash shell 和其指令語言。bash shell 已問世數十年，幾乎所有的 Linux 版本都有它的蹤影，甚至也進入到了 Windows 作業系統裡。由於其技術及指令稿可以跨平台運作，因而使得 bash 成為安全運作的理想技術。bash 的普及也替攻擊演練人員及滲透測試人員帶來了獨特的優勢，因為在大部分的情況下，它都不需要額外的基礎服務支援或是解譯，就能安裝在目標系統上。

指令列展示

本書以大量的範例來運用指令列。單獨一行指令會像這樣示範：

```
ls -l
```

如果單行指令示範還要加上輸出，就會像這樣：

```
$ ls -l

-rw-rw-r-- 1 dave dave  15 Jun 29 13:49 hashfilea.txt
-rwxrw-r-- 1 dave dave 627 Jun 29 13:50 hashsearch.sh
```

注意示範中如何以 $ 字元來區分輸入的指令和輸出。開頭的 $ 字元並非指令的一部分，而是代表 shell 指令列的簡易輸入提示。如此各位就能分辨指令（你會輸入的內容）和執行後在終端機畫面輸出的效果。範例中會以空白列把指令和其輸出分開來，但你在實際執行指令時不會有這種情形。這只是方便讀者而採取的編排。

除非特別說明，否則 Windows 指令的範例均以 Git Bash 執行，而非 Windows 自己的命令提示字元。

在 Windows 上運行 Linux 和 bash[譯註 1]

我們會探討的 bash shell 和指令，基本上所有版本的 Linux 都會預先安裝。但 Windows 環境則不盡然。還好，要在 Windows 系統上執行 Linux 指令和 bash 指令稿的方法有很多種。我們會介紹四種選項：Git Bash、Cygwin、Windows Subsystem for Linux、以及 Windows 命令提示字元與 PowerShell。

譯註 1　基於目前虛擬機架設十分容易，只要你的記憶體充足，用虛擬機架設一套 Linux 來實驗也不錯（除非要實驗 Windows 的指令）。

第二章

基本的 Bash

Bash 並不僅僅是一種可以執行程式的簡易指令列介面。它也是自成一格的程式語言。其預設的運作方式就是要啟動其他的程式。先前我們曾談過，如果指令列中出現連續幾個字眼；bash 就會假設第一個字是啟動的程式名稱，其後的字眼都是要傳遞給程式的引數。

但作為一種程式語言，它同時也具備了對輸入及輸出的支援功能，以及像是 `if`、`while`、`for`、`case` 等等的控制結構。其基本資料類型為字串（例如檔案名稱及路徑名稱等），但也支援整數作為資料類型。由於它主要專注在指令稿及啟動程式上，數字計算並非其所長，因此它並不直接支援浮點運算，而是要靠其他指令支援。本章要介紹若干讓 bash 之所以能成為強大程式語言的特性，尤其是在撰寫指令稿方面。

輸出

就像任何程式語言一樣，bash 可以把資訊輸出到螢幕上。使用 `echo` 指令就能達到輸出的目的：

```
$ echo "Hello World"

Hello World
```

你也可以改用內建的 `printf` 指令，它還支援更多的格式：

```
$ printf "Hello World\n"
```
譯註 1

```
Hello World
```

譯註 1　echo 會在它顯示的字串後面自動加上換行字符。這個差異後面還會提到。

Bash 甚至還可以用類似方式處理一連串管線指令：

```
if ls | grep pdf
then
    echo "found one or more pdf files here"
else
    echo "no pdf files found"
fi
```

若管線存在，則以管線中最後一道指令的成敗為判斷的標準，決定是否要走「為真」的途徑。以下就是一個要緊的例子：

```
ls | grep pdf | wc
```

就算 grep 指令沒有找到任何 pdf 檔案，這一串指令的結果仍是「為真」。因為最後一道 wc 指令的執行結果（計算輸入字數）一定是成功的，而且輸出如下：

```
0       0       0
```

以上輸出的意思是，因為 grep 指令的搜尋比對未果，因此 wc 計算自述的結果就成了零行、零字、零位元組（亦即零字元）。但這對於 wc 來說仍然算是執行成功（為真）的結果，沒有任何錯誤或失敗訊息。它只是要計算先前輸出的結果有幾行字而已，就算給它的是空無一物的輸入也一樣。

更典型的 if 比較方式，是利用複合式指令 [[或是 shell 的內建指令 [或 test。利用這些就能測試檔案的屬性、或是進行值的比較。

如要測試檔案系統中是否存在某個檔案：

```
if [[ -e $FILENAME ]]
then
    echo $FILENAME exists
fi
```

表 2-1 列出了 if 可以比對的其他檔案測試選項。

表 2-1　檔案測試運算子

檔案測試運算子	用途
-d	測試目錄是否存在
-e	測試檔案是否存在
-r	測試檔案是否存在且可讀取
-w	測試檔案是否存在且可寫入
-x	測試檔案是否存在且可執行

讓 / 從原本的含義「跳脫」出來，這樣 bash 才不會把它當成是替換語法的一部分。我們在這裡用了 \/ 指明它是一個字面上的斜線字元[譯註 7]。

❾ 接著又是一個用雙重小括號包覆的算術表示式。在算術表示式中，bash 就不需要在大多數的變數名稱前面加上 $ 了。但如果牽涉到例如 $1 之類的位置性參數，則 $ 就不能省，因為省掉就無法分辨它到底是整數 1、還是第一個位置性參數。

❿ 最後把所有的 .evtx 用 tar 打包成一個封存檔。參數 -z 代表要壓縮資料，如果不使用參數 -v，tar 就會悄悄地作業（因為指令稿已經在以上的說明項目 7 取出檔案名稱時顯示過一次了）。

這個指令稿會在一個 subshell 裡執行，因而儘管我們在指令稿裡切換了現行目錄，一旦離開指令稿，還是會回到我們當初開始的地方。如果想要回到指令稿裡切換過的位置，就必須改用 cd - 指令返回前一個所在目錄[譯註 8]。

蒐集系統資訊

倘若你可以在系統上任意執行指令，就可以透過標準的 OS 指令來蒐集系統的各種相關資訊。要用哪種指令端看你面對的作業系統而定。表 5-3 所列的就是能夠從系統取得大量資訊的常見指令。注意，指令會因你執行的是 Linux 或是 Windows 環境而有所不同。

表 5-3　本機端資料蒐集指令

Linux 指令	Windows Git Bash 等效指令	目的
uname -a	uname –a	作業系統版本資訊
cat /proc/cpuinfo	systeminfo	系統硬體相關資訊
ifconfig	ipconfig	網卡資訊
route	route print	路由表資訊
arp –a	arp –a	顯示位址解析表（ARP table）
netstat –a	netstat –a	顯示網路連線
mount	net share	顯示檔案系統
ps –e	tasklist	顯示執行中的程序

譯註 7　所以，第一組 // 代表要被替換的舊字串從這裡開始，而且跟上一項一樣，凡是有發現就要替換；然後 \/ 代表要替換的字元就是 / 本身，而且特地用 \ 做逸出處理；最後的 / 代表要替換的新字串從這裡開始，也就是 -。

譯註 8　這個範例引用了很多變數替代的用法，作者的說明僅點到為止。如果你看得一頭霧水，可以參閱作者之一 Carl Albing 所著的 *Bash Cookbook* 第二版（*http://shop.oreilly.com/product/0636920058304.do*），同樣為 O'Reilly 出版。書中第 5 章對於本例中 shell 變數的操縱方式，不論是上述說明項目 2 的位置變數遞補、項目 3 的預設值指定、項目 6 的切削清除、還是項目 8 的新舊字串替換，都有極為精闢的講解。請務必研讀到理解為止，不然後面示範的指令稿讀起來會倍感吃力。

jq

jq 是一種輕巧的語言和 JSON 剖析器，適合 Linux 指令列使用。它很厲害，但大部分的 Linux 版本並未事先預裝它。

如果要用 jq 取出 *book.json* 裡的 title 鍵：

```
$ jq '.title' book.json

"Cybersecurity Ops with bash"
```

要列出所有作者的名字：

```
$ jq '.authors[].firstName' book.json

"Paul"
"Carl"
```

由於作者欄 authors 是一個 JSON 的陣列，你必須在取用時加上 []。如果要取得陣列的特定元素，就要以索引為之，從位置 0（[0] 代表陣列的第一個元素）。如果要取得陣列中所有元素，只要用不含索引的 [] 即可。

關於 jq 的詳情，請參閱 jq 網站（*http://bit.ly/2HJ2SzA*）。

匯總資料

蒐集而來的資料常常來源各異，檔案和格式也都五花八門。在資料可供分析之前，你必須要先將資料集中一處，並轉換成便於分析的格式。

設想你要在一套名為 ProductionWebServer 的系統中搜尋檔案資料庫。而先前我們已用指令稿將蒐集而來的資料用 XML 標籤包裝過，其格式為 <systeminfo host="">。在蒐集資料時，我們也為檔案名稱加上了相關的主機名。現在你就可以利用這些屬性把資料找出來、並將其匯總在單一場所：

```
find /data -type f -exec grep '{}' -e 'ProductionWebServer' \;
-exec cat '{}' >> ProductionWebServerAgg.txt \;
```

指令 find /data -type f 會列出 *data* 目錄及子目錄下的所有檔案。然後針對每一個找出來的檔案執行 grep，挑出內文帶有 ProductionWebServer 字串的檔案。凡是挑出來的檔案，一律用 cat 將內容附加到（>>）*ProductionWebServerAgg.txt* 檔案裡。如果你只想把

以長條圖顯示資料

你可以讓計算再進一步，將結果用更為視覺化的方式呈現。不妨把來自 *countem.sh* 或 *summer.sh* 的輸出以管線轉給另一個指令稿，一個可以把結果呈現為產生長條圖型態的指令稿。

列印用的指令稿會把第一個欄位當成關聯式陣列的索引，再把第二個欄位當成陣列元素值。接著它會迭代陣列，並印出一定數量的井字號來代表加總數量，清單中最大的數量就會以 50 個 # 字符號來呈現。

範例 *7-6　histogram.sh*

```
#!/bin/bash -
#
# Cybersecurity Ops with bash
# histogram.sh
#
# 說明：
# 針對資料產生水平長條圖
#
# 用法：./histogram.sh
#   輸入格式：標籤 數值[譯註 10]
#

function pr_bar ()                                    ❶
{
    local -i i raw maxraw scaled                      ❷
    raw=$1
    maxraw=$2
    ((scaled=(MAXBAR*raw)/maxraw))                    ❸
    # 以下為確保 scaled 至少為 1 而設計
    ((raw > 0 && scaled == 0)) && scaled=1            ❹

    for((i=0; i<scaled; i++)) ; do printf '#' ; done
    printf '\n'

} # 函式 pr_bar 到此告一段落

#
# " 以下為指令稿主體 "
#
declare -A RA                                                ❺
```

譯註 10　如果用手動輸入標籤和數值資料的方式，而非事後以管線餵入資料，則結束輸入後記得要按下 Ctrl-D 以便開始繪圖計算。

❻ 從日誌檔讀入的內容是一長串混合了複雜字眼和引號的字串。若要捕捉使用者代理端的字串，我們必須改以「雙引號」來做為區隔字符，以便取出 `$1` 和 `$6` 兩欄資料。但是這樣做就代表第一個欄位 `$1` 會含有 IP 位址以外的資訊（參閱範例 7-2，其實是從 IP 位址開始一直到代表時區的 `-0500` 為止）。利用 bash 的 `read`，可以再把空格去掉以取得 IP 位址。`read` 的最後一個引數會把所有尾隨的字眼都納入，這樣就能取得內含使用者代理端字串的內容[譯註 22]。

執行 *useragents.sh* 時，它會輸出任何無法在 *useragents.txt* 檔案裡找到的使用者代理端字串：

```
$ bash useragents.sh < access.log

anomaly: 192.168.0.36 Mozilla/4.5 (compatible; HTTrack 3.0x; Windows 98)
anomaly: 192.168.0.36 Mozilla/4.5 (compatible; HTTrack 3.0x; Windows 98)
anomaly: 192.168.0.36 Mozilla/4.5 (compatible; HTTrack 3.0x; Windows 98)
anomaly: 192.168.0.36 Mozilla/4.5 (compatible; HTTrack 3.0x; Windows 98)
.
.
.
anomaly: 192.168.0.36 Mozilla/4.5 (compatible; HTTrack 3.0x; Windows 98)
```

總結

在這一章裡，我們研究了各種統計分析的技術，藉以找出日誌檔中不尋常的行為。這類分析方式可以幫大家深入了解曾經發生過的事情。在下一章當中，我們要探討如何分析日誌檔和其他資料，以便即時掌握系統中正在發生的事。

研討

1. 以下範例使用了 `cut` 來印出 *access.log* 檔案的第一和第十個欄位：

   ```
   $ cut -d' ' -f1,10 access.log | bash summer.sh | sort -k 2.1 –rn
   ```

   請用 `awk` 指令代替 `cut` 指令進行改寫。結果是否相同？這兩種方式會有何不同？

譯註 22 注意，此處 `read` 的資料來源已經是 `awk` 切割出來的 `$1` 和 `$6`，也就是第一個雙引號之前的字串、加上第五和第六這一對雙引號之間的字串。如果把 `read` 後面賦值的變數一一對照範例 7-2，很容易就可以看出每個變數字面上的用意，譬如代表兩個破折號 `dash1` 和 `dash2`、代表時序的 `dtstamp`、代表時差的 `delta`、以及最重要的 `useragent` 等等。這樣你就知道 `read` 已把使用者代理端的字串讀進變數 `useragent` 了。

若要進入指令（Command）模式時，只需按下 Esc 鍵即可。各位可以輸入表 11-1 所列的任何指令，並按下 Enter 鍵令其生效執行。

表 11-1　常見的 vi 指令

指令	目的
b	後退一整個字
cc	取代現在這一整行
cw	取代現在這一個字
dw	刪除現在這一個字
dd	刪除現在這一整行
:w	寫入 / 儲存檔案
:w *filename*	寫入 / 儲存至指定的檔名 *filename*
:q!	退出不存檔
ZZ	儲存並退出
:set number	顯示行號
/	向前找
?	往回找
n	找出下一個發生處

vi 的完整介紹已經超出本書範疇。詳情可參閱 Vim 編輯器網頁（*https://www.vim.org/*）。

xxd

xxd 指令會以二進位或十六進位格式將檔案顯示在螢幕上。

常用的指令選項

-b

以二進位格式顯示檔案，而非十六進位輸出。

-l

印出 n 個位元組。

-s

從第 n 個位元組的位置開始。

```
        "result": "Trojan.Ransom.WannaCryptor.H",
        "update": "20180712"}
         .
         .
         .
```

雖然 JSON 十分適合用來呈現結構化資料，用肉眼判讀仍嫌吃力。你可以用 grep 取出一些重要資訊，像是檔案是否被判定為惡意之類：

```
$ grep -Po '{"detected": true.*?"result":.*?,' Calc_VirusTotal.txt

{"detected": true, "version": "1.3.0.9466", "result": "W32.WannaCrypLTE.Trojan",
{"detected": true, "version": "14.0.297.0", "result": "Trojan.Ransom.WannaCryptor.H",
{"detected": true, "version": "14.00", "result": "Trojan.Mauvaise.SL1",
```

grep 的選項 -P 會啟用 Perl 引擎，這樣就可以引用 .*? 這種樣式的簡易量詞（lazy quantifier）。這個簡易量詞只會比對出符合整個正規表示式的最少量字元，於是你就可以擷取到每個防毒引擎的回應，而不必一次抓一大串。

雖然這種方式的確會奏效，但是使用 bash 指令稿的方式會更好，如範例 11-2 所示。

範例 11-2　vtjson.sh

```
#!/bin/bash -
#
# Rapid Cybersecurity Ops
# vtjson.sh
#
# 說明：
# 在 JSON 檔案中尋找 VirusTotal 命中的惡意軟體
#
# 用法：
# vtjson.sh [<json file>]
#   <json file> 含有 VirusTotal 分析結果的檔案
#               預設檔名：Calc_VirusTotal.txt
#

RE='^.(.*)...\{.*detect..(.*),..vers.*result....(.*).,..update.*$'      ❶

FN="${1:-Calc_VirusTotal.txt}"
sed -e 's/{"scans": {/&\n /' -e 's/},/&\n/g' "$FN" |                    ❷
while read ALINE
do
    if [[ $ALINE =~ $RE ]]                                              ❸
    then
        VIRUS="${BASH_REMATCH[1]}"                                      ❹
```

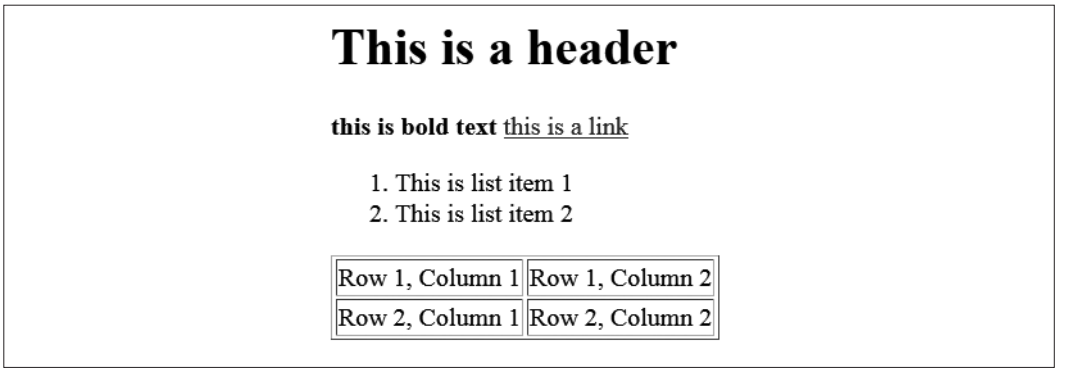

圖 12-1　呈現 HTML 網頁

為了簡化輸出為 HTML 的過程，各位可以用一支簡單的指令稿，把項目用標籤包起來。範例 12-2 就會讀入一段字串和標籤，並輸出為以標籤包覆的字串，並以換行字元結尾。

範例 *12-2*　*tagit.sh*

```
#!/bin/bash -
#
# Cybersecurity Ops with bash
# tagit.sh
#
# 說明：
# 在一段字串前後加上開頭和結尾的標籤
#
# 用法：
# tagit.sh <tag> <string>
#   <tag> 需要用到的標籤
#   <string> 需要標示的字串
#

printf '<%s>%s</%s>\n' "${1}" "${2}" "${1}"
```

這也可以改寫成一個簡單的函式，以便在其他指令稿中引用：

```
function tagit ()
{
    printf '<%s>%s</%s>\n' "${1}" "${2}" "${1}"
}
```

你可以用 HTML 標籤將幾乎任何類型的資料重新格式化，使其易於閱讀。範例 12-3 就是一段指令稿，它會讀取範例 7-2 中 Apache 的 *access.log* 檔案，再利用 `tagit` 函式將其重新格式化，讓日誌檔改以 HTML 格式輸出。

範例 *12-3　weblogfmt.sh*

```
#!/bin/bash -
#
# Cybersecurity Ops with bash
# weblogfmt.sh
#
# 說明：
# 讀取 Apache 的網頁日誌並輸出為 HTML 格式
#
# 用法：
# weblogfmt.sh input.file > output.file[譯註 1]
#

function tagit()
{
        printf '<%s>%s</%s>\n' "${1}" "${2}" "${1}"
}

# 基本的標頭標籤
echo "<html>"                                                    ❶
echo "<body>"
echo "<h1>$1</h1>"   # 標題

echo "<table border=1>"   # 有邊框的表格
echo "<tr>"   # 新資料行
echo "<th>IP Address</th>"  # 欄位標題
echo "<th>Date</th>"
echo "<th>URL Requested</th>"
echo "<th>Status Code</th>"
echo "<th>Size</th>"
echo "<th>Referrer</th>"
echo "<th>User Agent</th>"
echo "</tr>"

while read f1 f2 f3 f4 f5 f6 f7 f8 f9 f10 f11 f12plus    ❷
do
        echo "<tr>"
        tagit "td" "${f1}"
        tagit "td" "${f4} ${f5}"                                 ❸
```

譯註 1　測試證明原本的 < 是多餘的。作者在 *https://github.com/cybersecurityops/cyber-ops-with-bash/blob/master/ch12/weblogfmt.sh* 裡已修正這段註解。