GIAC
CERTIFICATIONS

Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Network Monitoring and Threat Detection In-Depth (Security 503)"
at http://www.giac.org/registration/gcia

Wireshark: A Guide to Color My Packets

Detecting Network Reconnaissance to Host Exploitation

GIAC GCIA Gold Certification

Author: Roy Cheok
Advisor: Robert VandenBrink

Accepted: 1st July 2014

Abstract

When was the last time you faced a packet trace file, and hoped to remember all the different filters used to detect anomalous network activities? Were you typing in the filters as you progress, and hoping for an alternate solution? This paper discusses some basic features in Wireshark, and the advanced techniques for creating simple to complex Display filters for Coloring rules, using it to identify network reconnaissance, attacks and recovering evidence from within the packet trace files.

© 2014 SANS Institute Author retains full rights.

© 2014 The SANS Institute Author retains full rights

1. Introduction

Incident Responders investigating technology-facilitated crime in an unfamiliar or even non-homogenous network environment can be given access to raw packet trace files. These files can provide a plethora of information to determine the source of compromised, and related nefarious activities. However, in the field environment with access to limited tools, the challenge is to analyze the packet trace files, identify any anomalous network activities, and provide an informed assessment in a time-critical environment.

Wireshark is an open source network protocol analyzer (Combs, n.d.). It can be customized to create Display filters and Coloring rules to highlight the obvious packets when reviewing complex network interaction; establishing the general flow of network activities, or identify anomalous traffic that relates to a network attack.

It is evident, the use and application of Wireshark Coloring rules can provide a faster analysis of result. By using colors to attract attention to the packet details, it can be a useful presentation tool to demonstrate the flow of complex data to non-technical staff, or jury in a court of law. Incident Responders can also harness the flexibility and portability of Wireshark Coloring rules by sharing customized rules that are helpful and assist in getting results that would otherwise be difficult or missed entirely.

This paper will guide the readers through examples, exploring Wireshark features used in creating Display filters and Coloring rules to detect network reconnaissance, attacks, and recovering data from packet trace files. With due diligence, experience and research, an Incident Responder should build a collection of Wireshark Coloring rules and adopt them for network forensics or incident response efforts.

2. Wireshark

The Wireshark project (*formerly known as Ethereal*) is created by Gerald Combs (Orebaugh, et al., 2007), it comes with an extensive library of supported network protocols and runs on many platforms, including Windows, Linux and OS-X. By default, the Wireshark installation package installs the feature-rich Wireshark Graphical User

© 2014 The SANS Institute Author retains full rights.

Interface (GUI)[1]. It also provides the option to install various Wireshark command line utilities, including a command line version of Wireshark called Tshark. This section introduces the Wireshark GUI using the labels depicted in Figure 1, which sets out some of the references used in this paper.

2.1. Wireshark Graphical User Interface

To view a packet trace file, launch Wireshark, and open a packet trace file via the File Menu. The Wireshark GUI view of an opened packet trace file is illustrated in Figure 1 below:

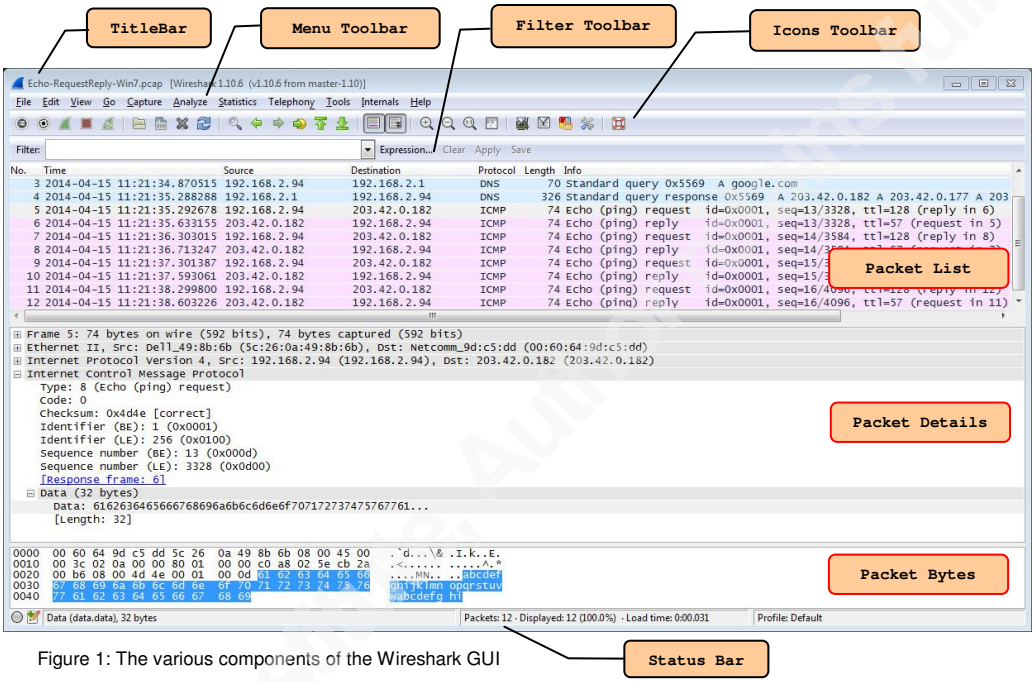


Figure 1: The various components of the Wireshark GUI

2.2. Wireshark: Profile

It is useful to create custom Wireshark Profiles for specific tasks relating to network protocol or packet analysis and troubleshooting (i.e. network scanning detection, unauthorized or anomalous network traffic identification).

To create or manage a new or existing Wireshark Profile, the user can navigate to the Configuration Profiles via the Edit Menu Toolbar. This option is also available via right-click on the Profile column in the Status Bar. A Bluetooth[2] and Classic[3] profile is

[1] For the purpose of this paper, Wireshark version: 1.10.6 (Windows 64-bit) was installed on a Microsoft Windows 7 (64-bit) workstation. Wireshark GUI for Linux and OSX Operating Systems utilized similar Wireshark GUI layout.
[2] The Wireshark Bluetooth profile contains additional Bluetooth related Coloring rules.
[3] The Wireshark Classic profile uses the default rules but displays the Packet List using a brighter color palette.

© 2014 The SANS Institute Author retains full rights

available as part of the default Wireshark installation and can be used as a template for creating a new profile.

When creating a new Profile, a folder with the given profile name is saved in the *profiles* folder within the Personal configuration directory. It is used for storing profile settings, such as user preferences, and files containing custom filters, i.e. Capture filters (cfilters), Display filters (dfilters) or Color filters (colorfilters).

The 'About Wireshark' dialog box via the Help Menu (in Figure 2) provides the essential Wireshark folders location (i.e. the Global and Personal configuration directories) installed on the system.

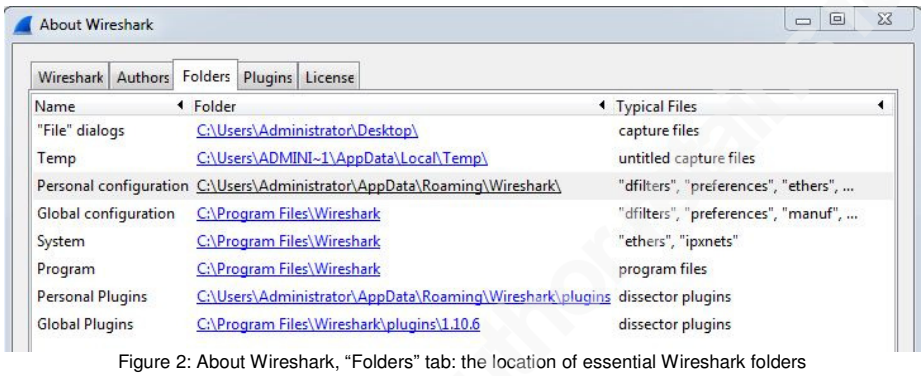


Figure 2: About Wireshark, "Folders" tab: the location of essential Wireshark folders

2.3. Wireshark: Display Filter

The Wireshark Display filter is temporarily applied to locate and display specific packets based on defined protocol field name(s). We will walk through some of the options to determine the protocol field names, using it to create simple or compound Display filters.

Option 1: Filter Toolbar – use Wireshark Filter Input Box

In Wireshark, network protocols and its fields used for Display filters are defined in lowercase (i.e. arp, ip, icmp, tcp, udp, dns, bootp, http). To create a Display filter to show all TCP packets, enter tcp within the Wireshark Display Filter Input Box as shown in Figure 3:

Figure 3: By itself, tcp is a valid Display filter to show all TCP packets
(auto-filter checking: Green color background indicates a valid display filter
Red color background indicates an invalid display filter)

© 2014 The SANS Institute Author retains full rights

Wireshark Display Filter Input Box also comes with an auto-lookup feature. It lists the available protocol fields as you type in the Filter Input Box toolbar. Enter tcp**.** (followed by a period), as shown below in Figure 4:

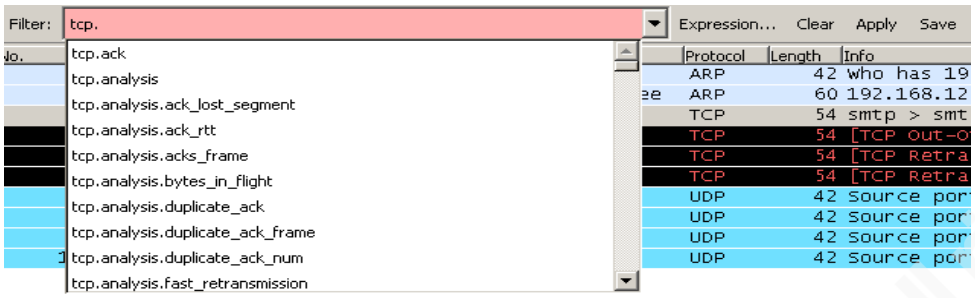


Figure 4: Wireshark auto-lookup feature, listing valid protocol field names.

Option 2: Filter Toolbar – use Wireshark Filter Expression

Apply the following steps shown in Figure 5 to create Display Filter via the 'Expression…' button[4] or to view a list of supported protocols and related field names:

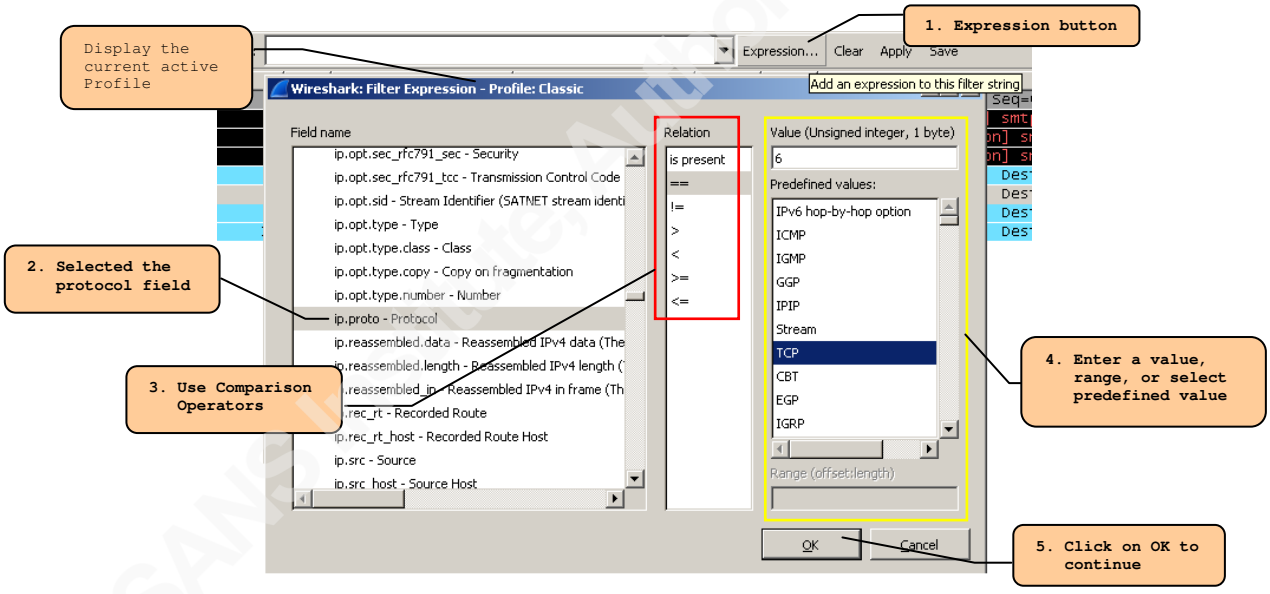


Figure 5: Creating Display Filter via the Expression…button

Option 3: Packet Detail Pane and Status Bar

To determine the correct Display filter protocol field name, select the required field in the Packet Details pane and lookup the status bar for the corresponding protocol field name (as shown by the yellow arrow in Figure 6).

[4] Invoking the Expression button, displays a list of all Wireshark supported protocols and its associated field names.

© 2014 The SANS Institute Author retains full rights.

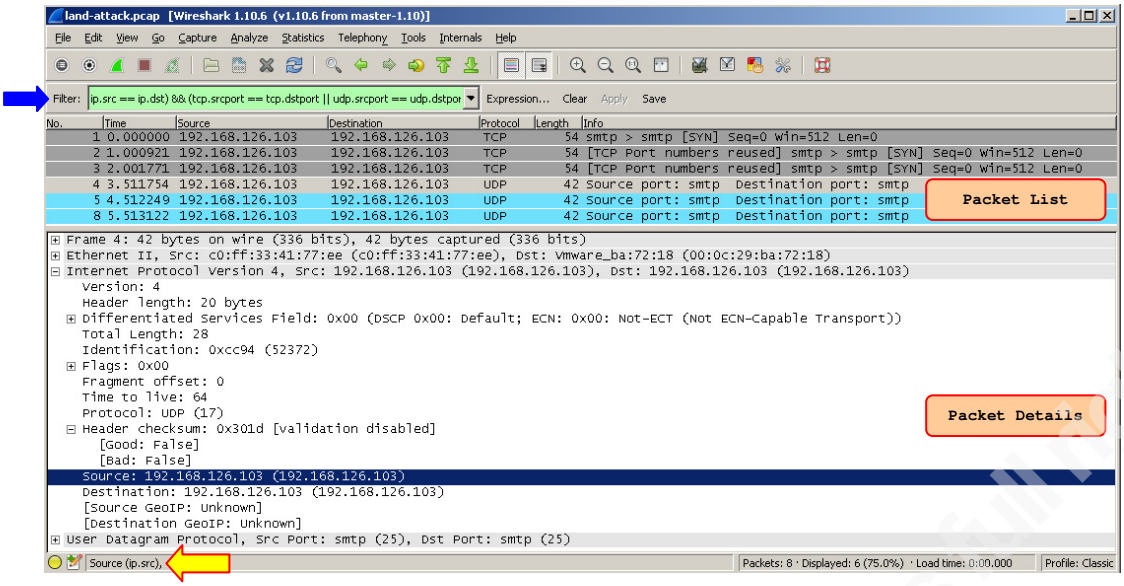


Figure 6: The selected protocol field name displayed in the Status Bar

Wireshark Display filter can be created by entering the appropriate network protocol field name in the Display Filter Input Box. Alternatively, right-click on a selected packet in the Packet List, or protocol field name in the Packet Details to bring up the list of available options, as shown in Figure 7 below:

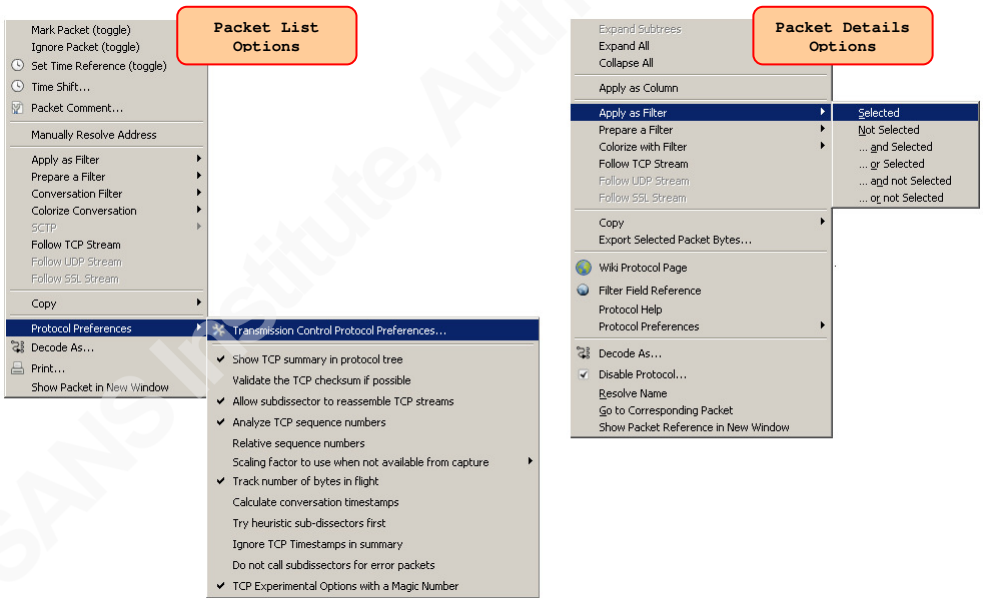


Figure 7: A myriad of options are available by right-clicking on a selected packet/protocol field name

The 'Apply as' or 'Prepare a Filter' option is particularly useful. It can be used to apply a simple Display filter, or create compound Display filter using comparison and logical operators. A common list of comparison and logical operators applicable to Wireshark is provided in the Table 1, with the exception of the 'contains' and 'matches' operator; either the short-form expression or the symbol can be used.

© 2014 The SANS Institute Author retains full rights

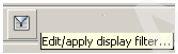
eq	==
ne	!=
gt	>
lt	<
ge	>=
le	<=
contains	case-sensitive string comparison or byte sequences matching
matches	search string fields and byte sequences using Perl Compatible Regular Expression (PCRE)
and	&&
or	||
not	!

Table 1: A list of comparison and logical operators for comparing values or combining Wireshark Display filters

In the example shown in Figure 6, the packet trace file shows an attempt to cause a Denial of Service (DoS) attack, where specially crafted packets were sent using the same source IP address and port number as the listening host's destination IP address and port number; this is known as the LAND attack (Imperva, n.d.). A vulnerable host will crash or freeze from processing such packets.

Using the options described in this section, the following Wireshark compound Display filter created below identifies TCP or UDP based land attack attempts:

```
(ip.src == ip.dst) && (tcp.srcport == tcp.dstport ||
udp.srcport == udp.dstport)
```

To save this Display filter to the current active Wireshark Profile for future analysis, select the Display filter icon 

located on the Icon Toolbar, or click on the 'Filter' button (shown by the blue arrow in Figure 6) on the Filter Toolbar to bring up the Display Filter Manager dialog box shown in Figure 8. Select New, enter a filter name (e.g. Land Attack), the Display filter string, and click Apply to save the Display filter.

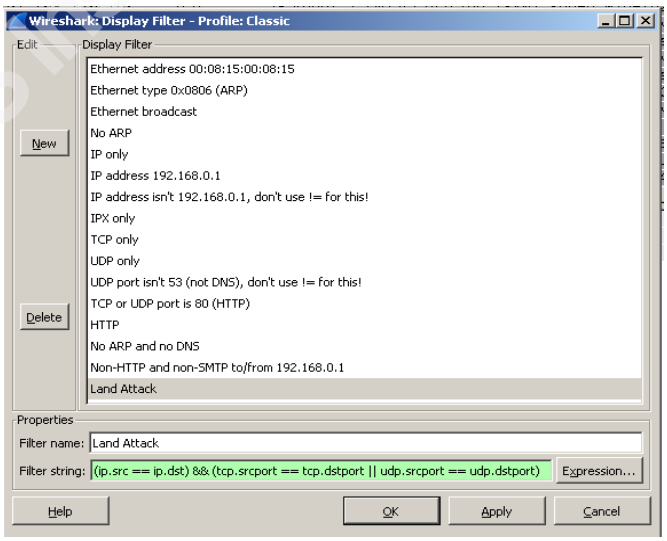


Figure 8: Wireshark Display Filter Manager Dialog Box

© 2014 The SANS Institute Author retains full rights

Newly created Display filters are saved to the "dfilters" file in the Personal configuration directory or within its *profiles* folder, it can be edited using a text editor or via the Display Filter Manager. Each Display filter in the "dfilter" file uses the following syntax that ends with a newline: *"Display filter name" Display filter string*.

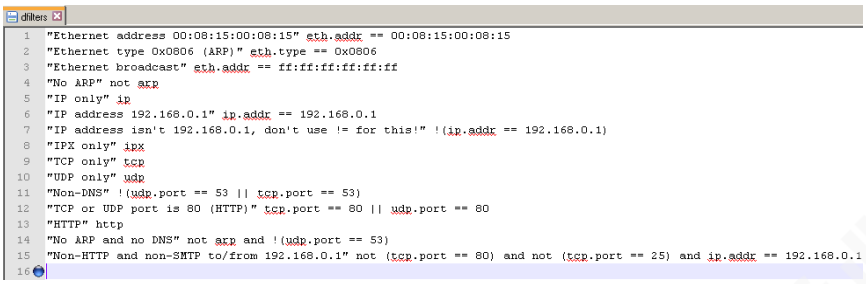
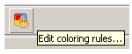


Figure 9: Wireshark *dfilter* file of an active profile, showing the described syntax ending with a newline

To remove the display filter shown in the Filter Input Box, select the 'Clear' button on the Filter Toolbar. To manage or apply previously stored Display filters, click on the Display filter icon or 'Filter' button and select the required Display filter listed in the Display Filter Manager dialog box

2.4. Wireshark: Coloring Rules

One of the features of Wireshark is the ability to apply custom Coloring rules to highlight the packets of interest. A Coloring rule is created using the Display filter syntax, and saving it to the "colorfilters" file using the Wireshark Color Filter rule editor. The Coloring rules are applied automatically to the packets in the background, integrating seamlessly with the packet display.

To create and manage a new or existing Coloring rule, navigate to the Wireshark Coloring rules manager (in Figure 10) via the Coloring Rules icon 

located on the Icon Toolbar, or via the sub-menu item: 'Coloring Rules...' under the View Menu.

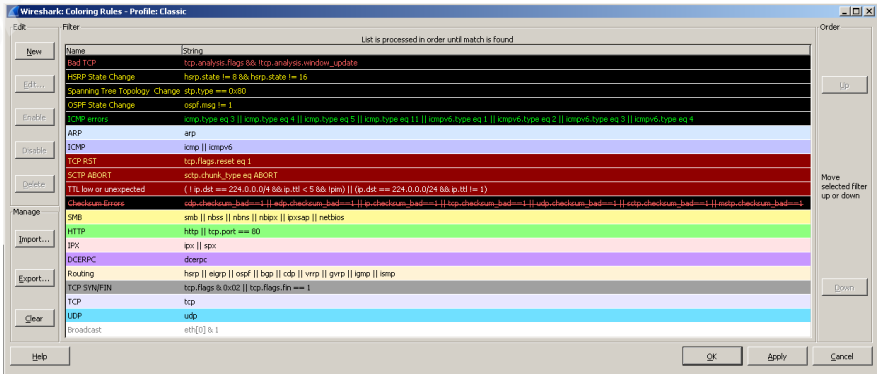


Figure 10: Wireshark: Coloring rules manager, (The title bar indicates the applicable coloring rules for the current active Profile and the "Checksum Error" rule is shown as being disabled)

© 2014 The SANS Institute Author retains full rights.

To create a new Wireshark Coloring rule:

1. Click on the New button to launch the Wireshark Color Filter rule editor (shown in Figure 10 below).
2. Provide a Coloring rule name
3. Enter the Display Filter (if required, use the Expression option to assist)

   e.g. using the example shown in Figure 1, we will create a simple filter to identify Windows ICMP Echo Request (ICMP type 8, code 0) packets, containing about 32 bytes of data resembling a string of English alphabets in lowercase.
4. Click on the "Foreground Color" button to select a foreground color
5. Click on the "Background Color" button to select a background color
6. Check to ensure that the Coloring rule is valid via the displayed color shown in the String Display Filter Input Box (in Figure 11 below):

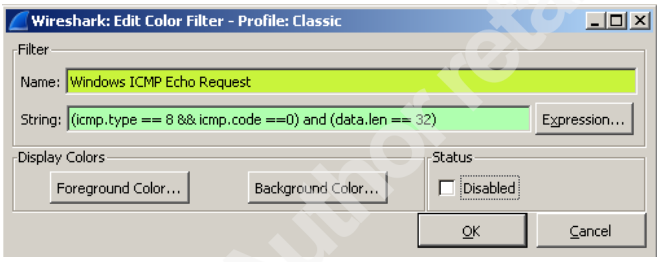


Figure 11: Wireshark Color Rule Editor with a valid Color Filter
(String Input box: a Green color background indicates a valid Display filter;
a Red color background indicates an invalid Display filter)

7. Click the "OK" button to create the Coloring rule.

By default, the new Coloring rule is placed at the top of the list in the Coloring rules manager, the rules are processed and the result is illustrated below:

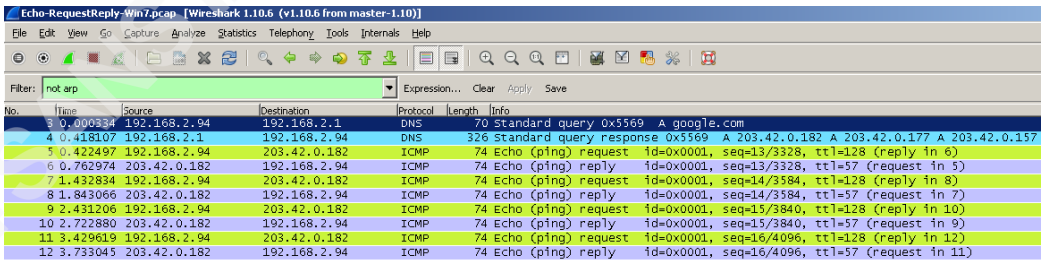


Figure 12: The "Windows ICMP Echo Request" Coloring Rule was successfully applied, highlighting the ICMP Echo Request (Ping) packets: 5, 7, 9, and 11 using the configured color scheme

To identify the Coloring rule that was applied to a specific packet, select one of the packets (i.e. ICMP Echo Request) from the Packet List, and expand the frame section in the Packet Details pane to reveal the details of the applied Coloring Rule (as shown in

© 2014 The SANS Institute Author retains full rights

Figure 13). It is also possible to display a list of packets with the same applied rule; right-click on the Coloring rule in the Packet Details pane, and select the option to apply it as a simple Display filter.

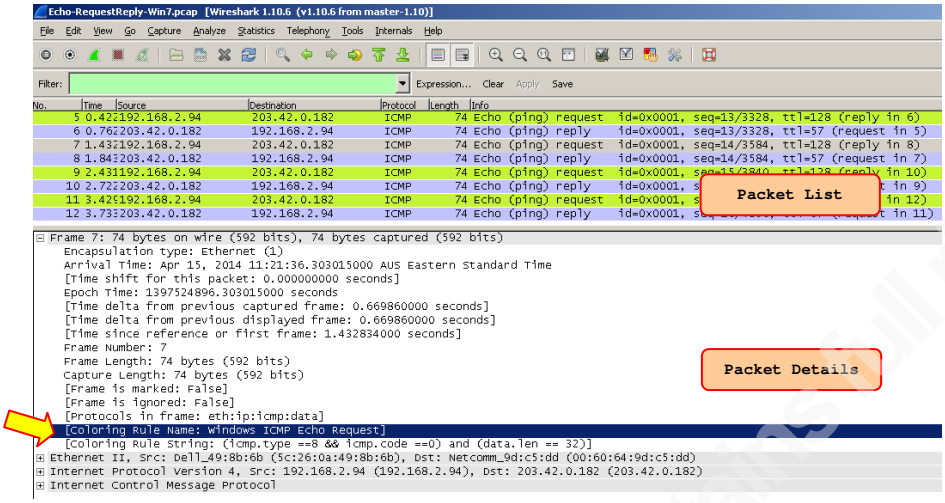


Figure 13: Indentifying the triggered Coloring Rule from the Frame section in the Packet Details pane

To manage an existing Coloring rule listed in the Coloring rules manager, select the required Coloring rule and click on the appropriate option button. A temporary disabled Coloring rule is shown with its Coloring rule name and filter string being strikethrough in the Coloring rules manager. The example in Figure 10 shows the disabled 'Checksum Error' Coloring rule.

To share a copy of the Coloring rules used in the current Wireshark Profile, use the Export option within the Coloring rules manager to save it to a file. The Import option takes in a file containing the imported Coloring rules, and appends it to the end of any existing Coloring rules. This may include duplicated Coloring rules that can be removed using the Delete option.

2.4.1. Wireshark: colorfilters file

Wireshark saves pre-defined (default), imported or custom created Coloring rules of an active profile to a file named: "colorfilters". It is located within the Personal configuration directory or *profiles* folder, and can be edited or reviewed using a text editor. The syntax used by Wireshark "colorfilters" file is explained below:

#	: denotes a single line comment, not processed as a coloring rule
@	: denotes a delimiter used to identify and separate the required fields used to describe the Coloring Rule:

© 2014 The SANS Institute Author retains full rights

e.g. @Coloring Rule Name@Display Filter String@[Background Color] [Foreground Color]

note: the Background and Foreground color co-ordinates are each separated by a comma

! : denotes a disabled Coloring Rule

```
colorfilters
1  # DO NOT EDIT THIS FILE!  It was created by Wireshark
2  @Windows ICMP Echo Request@(icmp.type ==8 && icmp.code == 0) and (data.len == 32)@[51693,62630,15267][0,0,0]
3  @Bad TCP@tcp.analysis.flags && !tcp.analysis.window_update@[4718,10030,11796][63479,34695,34695]
4  @HSRP State Change@hsrp.state != 8 && hsrp.state != 16@[4718,10030,11796][65535,64764,40092]
5  @Spanning Tree Topology  Change@stp.type == 0x80@[4718,10030,11796][65535,64764,40092]
6  @OSPF State Change@ospf.msg != 1@[4718,10030,11796][65535,64764,40092]
7  @ICMP errors@icmp.type eq 3 || icmp.type eq 4 || icmp.type eq 5 || icmp.type eq 11 || icmpv6.type eq 1 || icmpv6.type eq 2 || icmpv6.type eq 3 || icmpv6.type eq
   4@[4718,10030,11796][47031,63479,29812]
8  @ARP@arp@[64250,61680,55255][4718,10030,11796]
9  @ICMP@icmp || icmpv6@[64764,57568,65535][4718,10030,11796]
10 @TCP RST@tcp.flags.reset eq 1@[42148,0,0][65535,64764,40092]
11 @SCTP ABORT@sctp.chunk_type eq ABORT@[42148,0,0][65535,64764,40092]
12 @TTL low or unexpected@( ! ip.dst == 224.0.0.0/4 && ip.ttl < 5 && !pim) || (ip.dst == 224.0.0.0/24 && ip.ttl != 1)@[42148,0,0][60652,61680,60395]
13 !@Checksum Errors@eth.fcs_bad==1 || ip.checksum_bad==1 || tcp.checksum_bad==1 || udp.checksum_bad==1 || sctp.checksum_bad==1 || mstp.checksum_bad==1 ||
   cdp.checksum_bad==1 || edp.checksum_bad==1 || wlan.fcs_bad==1@[4718,10030,11796][63479,34695,34695]
14 @SMB@smb || nbss || nbns || nbipx || ipxsap || netbios@[65278,65535,53456][4718,10030,11796]
15 @HTTP@http || tcp.port == 80@[58596,65535,51143][4718,10030,11796]
16 @IPX@ipx || spx@[65534,58325,58808][4718,10030,11796]
17 @DCERPC@dcerpc@[51199,38706,65533][4718,10030,11796]
18 @Routing@hsrp || eigrp || ospf || bgp || cdp || vrrp || gvrp || igmp || ismp@[65534,62325,54808][4718,10030,11796]
19 @TCP SYN/FIN@tcp.flags & 0x02 || tcp.flags.fin == 1@[41026,41026,41026][4718,10030,11796]
20 @TCP@tcp@[59345,58980,65535][4718,10030,11796]
21 @UDP@udp@[56026,61166,65535][4718,10030,11796]
22 @Broadcast@eth[0] & 1@[65535,65535,65535][47802,48573,46774]
```

Figure 14: The Wireshark *colorfilters* file showing the syntax used for Wireshark pre-defined and custom Coloring rules

2.4.2. Wireshark: colorfilters file

To determine or verify the processing order of the Wireshark Coloring rules, we will re-order the "Windows ICMP Echo Request" Coloring rule created earlier, and moving it below the Wireshark pre-defined "ICMP" Coloring rule as shown in the figure below:

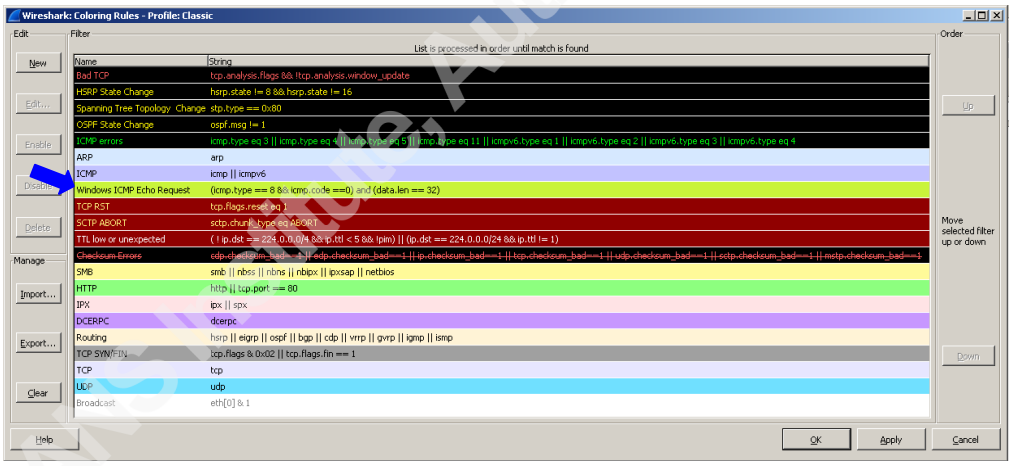


Figure 15: The re-ordering (up/down move buttons) of the Windows ICMP Echo Request Coloring Rule

The results illustrated in Figure 16, shows that Wireshark processed the Coloring rules sequentially from top to bottom in the order they are listed, until a match is found for each packet. If there is no match, the Wireshark default Coloring rule is applied. Essentially, a more specific rule should be placed before a general rule to ensure that the Coloring rule is triggered during processing.

© 2014 The SANS Institute Author retains full rights

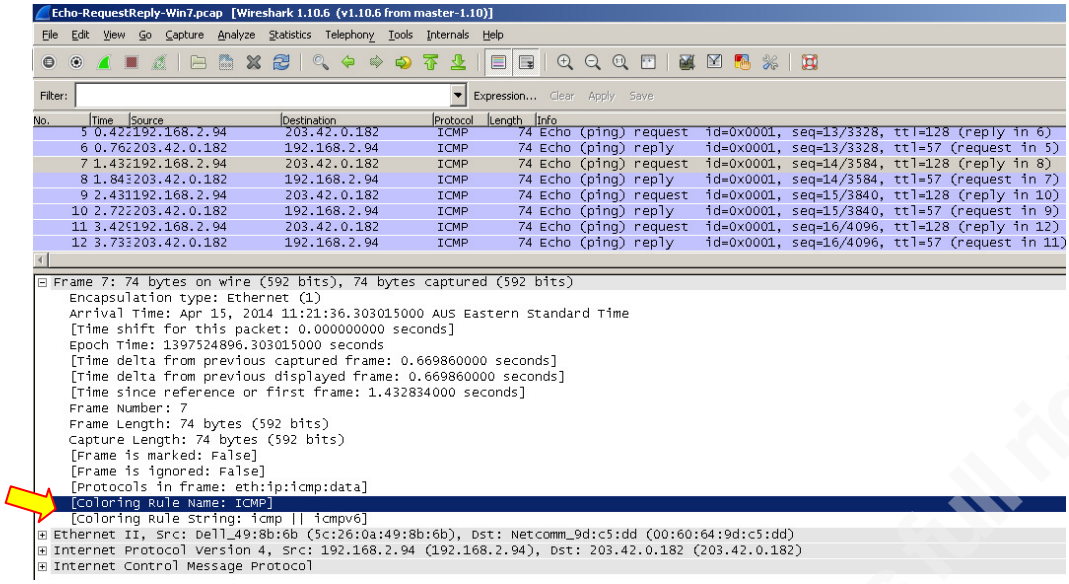


Figure 16: The re-ordering of the Coloring ruleset, invoked the Wireshark pre-defined ICMP Coloring Rule and removing the colors from the ICMP Echo Request (Ping) packets: 5, 7, 9, and 11

3. Network Reconnaissance

Network reconnaissance is one of the most important stages of any attack. It aims to identify active hosts listening on the targeted network. It is during this information gathering and preparatory phase that the attacker(s) maps the network topology, identify network devices, host IP addresses, open ports, network services and operating system information that can be associated to known or potential vulnerabilities; exploiting it to gain unauthorized access and control its targets. The more complete is the information gathered about the targeted network and its hosts, the higher chance of a successful attack.

3.1. DNS Reconnaissance

Domain Name Server (DNS) is used to translate a domain name to an IP address. An organization's public accessible domain name server generally stores its own DNS records and can provide a variety of information, including its function described in RFC1035 (Mocakapetris, 1987).

DNS information is publicly available and can be queried using built-in operating system command line tools such as nslookup, host, or dig. This information is extremely useful to an attacker, since DNS reconnaissance can easily obtain a list of known host IP addresses from the targeted network.

© 2014 The SANS Institute Author retains full rights

Unlike normal DNS query and response that occurs over UDP Port 53, DNS zone transfer requests and responses occur over TCP Port 53. DNS zone transfer is the replication process to provide a copy of the DNS zone file resource records from a primary DNS server to a set of secondary DNS server(s). DNS zone transfer request/response should be restricted to known secondary DNS server(s); otherwise, an entire copy of the DNS zone file can be requested.

The following Wireshark compound Display filters retrieve any DNS Zone Transfer requests and responses from within a packet trace file:

```
DNS Zone Transfer request:
(tcp.dstport == 53) && (dns.flags.response == 0) && (dns.qry.type == 0x00fc)

DNS Zone Transfer response:
(tcp.srcport == 53) && (dns.flags.response == 1) && (dns.qry.type == 0x00fc)
```

3.1.1. DNS Information

To retrieve a list of DNS queries and responses from within a packet trace file, use the following Wireshark Display filters:

```
DNS queries and responses  : dns

DNS queries                : dns.flags.response == 0

DNS response               : dns.flags.response == 1
```

To display a list of DNS queries or responses containing a specific record type, e.g. DNS A record; a common list of DNS record types is also provided in Table 2:

```
DNS query    : (dns.flags.response == 0) and (dns.qry.type == 0x0001)

DNS response : (dns.flags.response == 1) && (dns.resp.type == 0x0001)
```

A	IPv4 host address	0x0001
AAAA	IPv6 host address	0x001c
NS	authoritative name server	0x0002
CNAME	alias canonical name	0x0005
SOA	start of zone authority	0x0006
PTR	Domain name pointer	0x000c
HINFO	host info	0x000d
MINFO	mailbox/mail list info	0x000e
MX	mail exchange	0x000f
AXFR	zone transfer	0x00fc

Table 2: A list of common DNS records types

© 2014 SANS Institute, Author retains full rights.
© 2014 The SANS Institute Author retains full rights

Alternatively, Wireshark can display a list of all name resolution from within a packet trace file via "Show Address Resolution" option found under the Statistics Menu:

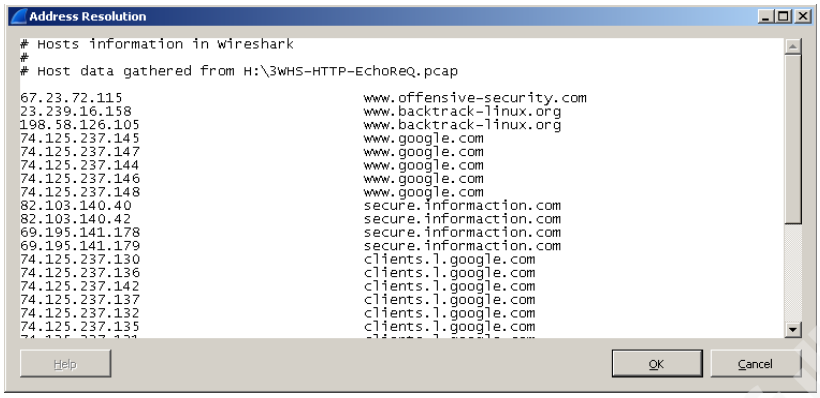


Figure 17: A list of DNS name resolution obtained from within a trace file

3.2. Network Mapping

Network mapping attempts to elicit response from active hosts, mapping its topology, identifying its IP address, listening ports and services provided over the network.

3.2.1. NMAP Host Discovery & Port Scanning

Network Mapper or commonly known as NMAP was released by Gordon "Fyodor" Lyon as an open source utility in 1997 for network exploration and security auditing, and runs on Linux, OS-X, and Windows (Lyon, 2008). While an exhaustive explanation of every NMAP option and scan technique is not possible, the focus is on identifying the type of network scans. It is essential to mention that packet-crafting tools, such as Hping[5] or Scapy[6] can create similar packets to elicit response. The following are some of the observation made regarding NMAP[7] host discovery and port scanning options:

- **NMAP Host Discovery: Ping Scan**

  NMAP Ping scan is used only for host discovery, it does not conduct a port scan if it identifies an active host. This is the default scan uses by NMAP before starting its port or service scanning and remote OS fingerprinting (Lyon, 2014), and can be disabled.

[5] Hping, an open source software available via http://www.hping.org/download.html
[6] Scapy, an open source software available via http://www.secdev.org/projects/scapy
[7] NMAP version 6.01 was used in this paper and can be found as part of Backtrack 5-R3 Linux Penetration Testing distribution which has been discontinued, but available via: http://ftp.uio.no/pub/security/backtrack/

© 2014 The SANS Institute Author retains full rights

According to the NMAP man page, running NMAP Ping scan in privilege mode sends the following four packets to identify an active host (shown with the corresponding Wireshark Display filters to identify NMAP Ping scan):

1. NMAP sends an ICMP Echo Request with no data

```
(ip.flags == 0x00) && (icmp.type == 8 && icmp.code == 0) &&
(icmp.seq == 0) && (not data)
```

2. NMAP sends a TCP SYN to port 443

```
(ip.flags == 0x00) && (tcp.flags == 0x0002) &&
(tcp.dstport == 443)
```

3. NMAP sends a TCP ACK to port 80

```
(ip.flags == 0x00) && (tcp.flags == 0x0010) &&
(tcp.dstport == 80)
```

4. NMAP sends an ICMP timestamp request without originate time

```
(ip.flags == 0x00) && (icmp.type == 13 && icmp.code == 0) &&
(icmp.seq == 0) && (icmp[8:4] == 00:00:00:00) && (not data)
```

Before adding the above Wireshark Display Filters to the Coloring rules, it is pertinent to remember the order of precedence. By re-ordering the fourth ICMP rule before the first ICMP rule ensures the Coloring rule is correctly triggered. The figure below illustrates the successful application of the Coloring rules to identify NMAP Ping scan packets:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.2.10	192.168.2.1	DNS	75	Standard query 0x5f15 A scanme.nmap.org
2	1.597149	192.168.2.1	192.168.2.10	DNS	91	Standard query response 0x5f15 A 74.207.244.221
3	1.606320	192.168.2.10	74.207.244.221	ICMP	42	Echo (ping) request id=0xe4f1, seq=0/0, ttl=38
4	1.606758	192.168.2.10	74.207.244.221	TCP	58	62070 > https [SYN] Seq=0 Win=1024 Len=0 MSS=1460
5	1.606927	192.168.2.10	74.207.244.221	TCP	54	62070 > http [ACK] Seq=1 Ack=1 Win=1024 Len=0
6	1.607092	192.168.2.10	74.207.244.221	ICMP	54	Timestamp request id=0x7e71, seq=0/0, ttl=46
7	2.495778	74.207.244.221	192.168.2.10	TCP	60	https > 62070 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

Figure 18: The Coloring Rules detect NMAP Ping scan successfully in packets: 3-6; illustrating the characteristics documented in the man page for NMAP Ping scan

3.2.2. NMAP Port Scanning

- **TCP SYN Stealth scan:**

The NMAP Stealth scan is a half-open TCP SYN scan. It does not complete the TCP 3-way handshake. NMAP sends a SYN packet to elicit a SYN/ACK packet response from a listening port. If it receives a response, NMAP sends a RST packet to tear down the connection. The following Wireshark Display filter identifies the TCP SYN packet:

```
(ip.flags == 0x00) && (tcp.flags == 0x0002) &&
(tcp.option_kind == 2) && not (tcp.option_kind == 3 ||
tcp.option_kind ==4)
```


© 2014 The SANS Institute Author retains full rights

- **TCP Connect scan:**

The NMAP TCP Connect scan completes the TCP 3-way handshake with the target listening port(s), and tears down the connection with a RST/ACK packet. The following Wireshark Display filter identifies the TCP SYN packet:

```
(ip.flags == 0x02) && (tcp.flags == 0x0002) &&
(tcp.option_kind == 2 && tcp.option_kind ==3 &&
tcp.option_kind ==4 && tcp.option_kind ==8)
```

While both NMAP TCP SYN Stealth and TCP Connect scans initiate destination port connection using TCP SYN packet, a notable difference in the IP header was observed:

- NMAP SYN Stealth scan sends SYN packet with IP Don't Fragment flag set to 0 and a single TCP Option: Maximum Segment Size (MSS).
- NMAP Connect scan sends SYN packet with IP Don't Fragment flag set to 1 with multiple TCP Options.

The Display filters described above were added to the Wireshark Coloring rules, with the results in Figure 19 illustrating the differences:

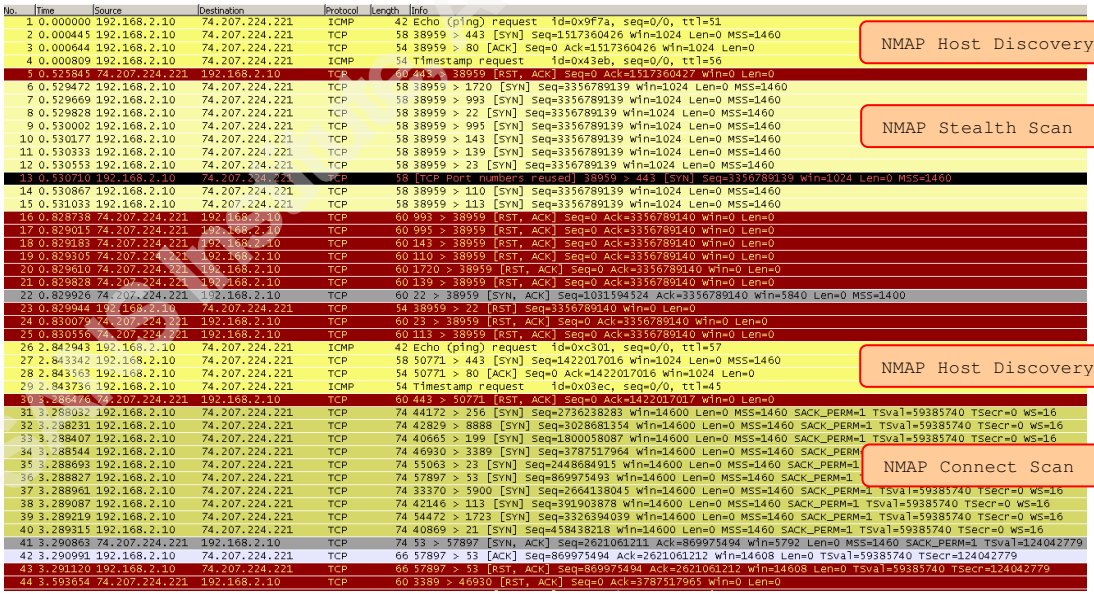


Figure 19: Coloring rules applied to identify packets relating to NMAP Stealth Scan and TCP Connect Scan

© 2014 The SANS Institute Author retains full rights

4. Detecting Web Server Scanning

Apart from mapping the topology of a private network or port scanning for listening hosts connected to Internet, attackers are also looking for vulnerable web servers; seemingly looking for any opportunity to compromise the integrity of the web server via a vulnerability, and gaining access to its data store.

Nikto is a web server scanner, an open-source project that is publicly available (Sullo & Lodge, 2012). It can be used by a diligent system administrator with explicit given permission or by an attacker to discover well-known vulnerabilities, mis-configurations, and unpatched software installed on the web server(s). Nikto performs a number of tests on web servers, including directory traversal attempts to access restricted directories on the web server, trying to gain access to the underlying operating system or known data files, and attempts to execute commands outside of the web server root directory.

A simple Wireshark Display filter, using the "contains" comparison operator can be used to match a sequence of hexadecimal byte values (shown in ASCII as ../) to return a list of directory traversal attempts: `http.request.uri contains 2e:2e:2f:`

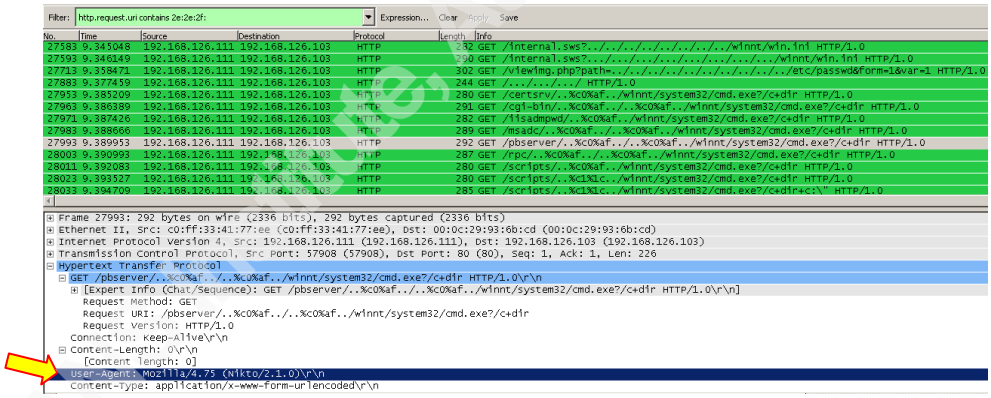


Figure 20: Identifying Nikto scan via its user-agent string

In the figure above, the user-agent string captured within the HTTP request shows that the web traffic was generated by Nikto scan. By extending the above filter, a compound Display filter is created to identify Nikto and its related directory traversal attempts:

```
http.user_agent contains "Nikto" and http.request.uri contains 2e:2e:2f:
```

To display a list of directory traversals with its corresponding response, update the Wireshark Display filter to search the packet frame: `frame contains 2e:2e:2f:`

© 2014 The SANS Institute Author retains full rights.

As shown in Figure 21, a manual review identifies a successful attack amongst the scanning activities:

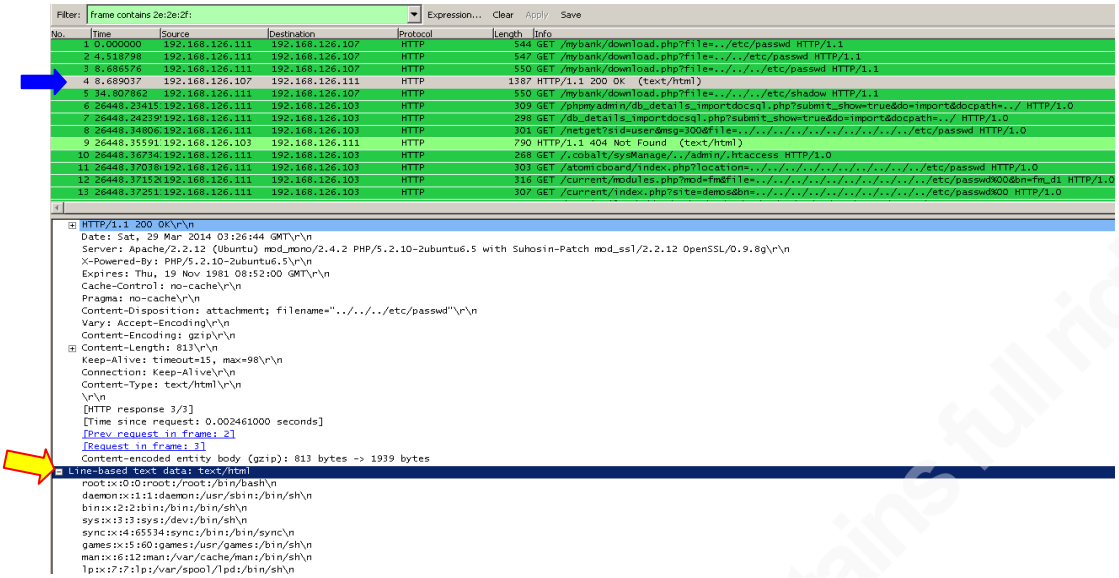


Figure 21: Detecting a successful directory traversal attack, retrieving the /etc/password file

While it is trivial to modify the user-agent[8] string, the following Wireshark compound Display filter is created to retrieve any HTTP request(s) containing uncommon user-agent strings, with its result shown in Figure 22:

```
http.request && (not http.user_agent matches "(?i)
(Chrome|Mozilla|Gecko|MSIE|Safari|AppleWebKit|Android|Blackberry|Opera)")
```

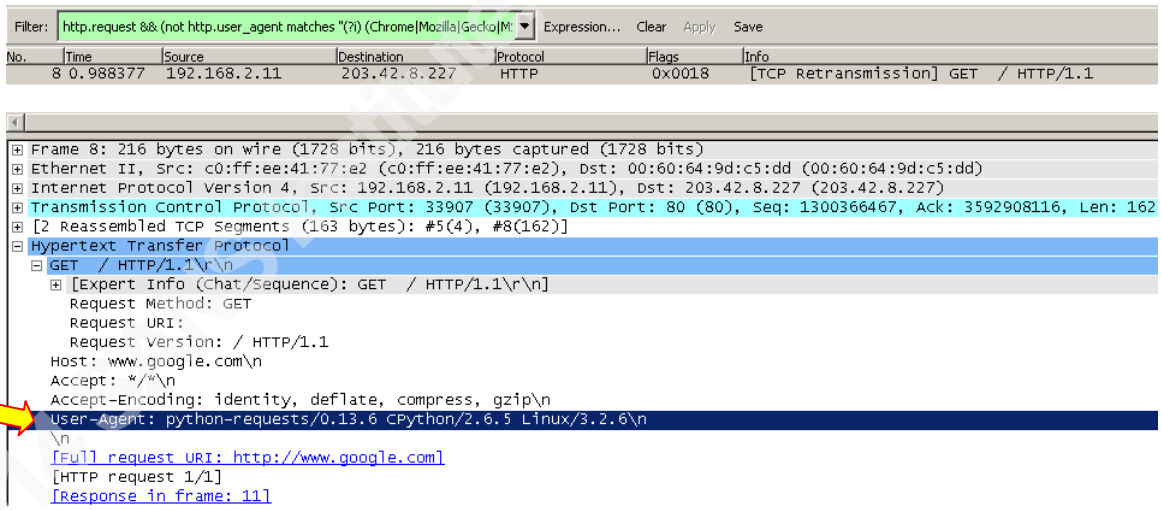


Figure 22: Identifying a non-browser based web request

[8] A comprehensive list of user-agent string can be found via http://www.useragentstring.com/pages/useragentstring.php

© 2014 The SANS Institute Author retains full rights

5. Detecting Host Exploitation

While it is not possible to discuss every attack and exploitation technique used in the wild, researching common attack vectors can help Incident Responders recognize the type of attacks used by the attacker(s). As an example, we will briefly discuss using Wireshark Coloring rules to detect an FTP brute force attack.

5.1. FTP Account/Password Guessing & Brute Force Attack

File Transfer Protocol (FTP) server can be easily setup for sharing of files across the Internet. It is described in RFC959, and utilizes a client/server model. The client connects to the FTP server on TCP Port 21 (control channel for FTP command) to initiate a service request, but utilizes a separate port for data exchange (Postel & Reynolds, 1985). In relation to data exchange, an FTP server can operate in either Active or Passive mode:

- In Active FTP mode: client issues the "PORT" command and provides the listening data port. The FTP server then uses TCP Port 20 to connect to the client specified port for data exchange.

- In Passive FTP mode: server provides the listening data port in response to client "PASV" command request. The client then connects to server specified port for data exchange.

According to RFC959, the data port is communicated using the convention: h1,h2,h3,h4,p1,p2, where h1-h4 is the host address and the listening port for data exchange is computed using the value of p1 and p2, such that the port is: p1*256+p2.

Some of the common client service commands found in RFC959 are listed below:

USER	submit username for login to FTP Server
PASS	submit password for login to FTP Server
LIST	request for a list of files in the current directory
STOR	upload data onto FTP server
RETR	download data from the server

The following Wireshark Display filter displays the list of FTP related commands issued by the FTP client in a packet trace file: `ftp.request.command`

It is also important to know that FTP transactions are in cleartext, thus susceptible to sniffing attacks. Depending on its configuration, it can also be prone to user

© 2014 The SANS Institute Author retains full rights

account/password brute force attacks. Unbeknownst to the owner, their FTP server can be attacked for nefarious use, such as hosting malicious malware, and insidious use of sharing offensive images over the Internet.

In Wireshark, the protocol name: "ftp" displays related packets that communicate on FTP command channel, TCP port 21. FTP data channel can be located using the field name: "ftp-data". Nonetheless, it is essential to review the packets to identify and verify the host IP address and data port number via a simple Wireshark Display filter:

```
(ftp.request.command == PORT) or (ftp.response.code == 227)
```

To identify brute force attack on an FTP server, create the following Display filters and add them as two separate Coloring rules:

To display a list of credential used in FTP server login attempts:

```
(ftp.request.command == "USER") or (ftp.request.command == "PASS")
```

To confirm a successful login to the FTP server:

```
(ftp.response.code == 200)
```

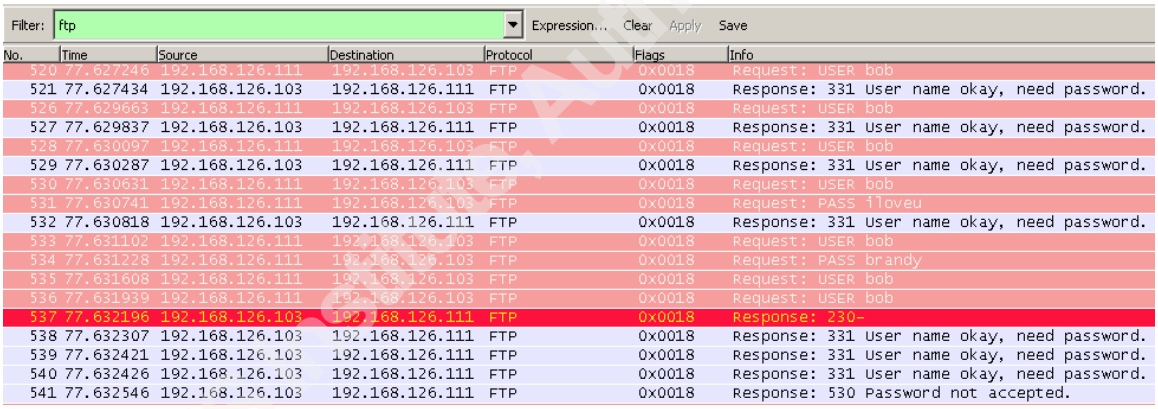


Figure 23: Wireshark Coloring rules successfully detects FTP brute Force login attack

As illustrated in the figure above, the Incident Responder can quickly identify the automated attack using the request time sequence, and reliably determine any successful login. To investigate subsequent actions, right-click on the packet: "*FTP Response: 230-*" in the Packet List, and select the Wireshark option to "Follow TCP stream". It brings up a dialog box showing the entire conversation stream, including any request that occurs after the successful login.

© 2014 The SANS Institute Author retains full rights

6. Data Recovery

Apart from compromising a system, attacker(s) will attempt to take a foothold of its victim's infrastructure. They will commonly upload toolkits, insert backdoors, and extract data from compromised machines. These may include password hash files for offline password cracking, intellectual property data, email and business financial or transaction records, etc. In this section, we will take advantage of Wireshark auto-packet reassemble feature, and use it like a forensic tool to recover data file from raw packet trace files.

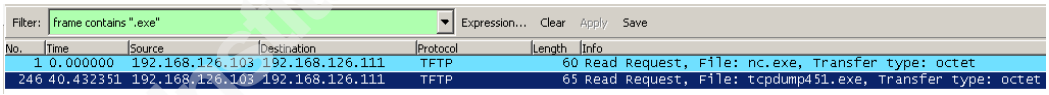
6.1. Identification of common data files in packet trace file:

Prior to any data extraction or recovery, Wireshark Display filter or Coloring rules can be used for searching common data files contained within the packet trace file. The following described some of the options:

1. Search by common file extension (e.g *.doc, *.pdf, *.exe, *.zip, *.gz., *.tar): It is possible to search an entire packet using "frame" or search for packets relating to specified protocol, defined in lowercase (e.g. ip, tcp, icmp, udp, tftp, ftp, http, smtp, dns, bootp):

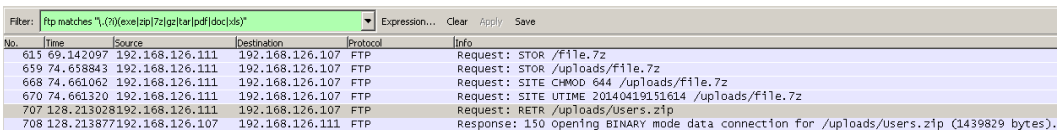
Option A: Use Wireshark comparison operator "contains" for case sensitive string matching of a single specific file extension:

```
frame contains ".exe"
```



Options B: Use Wireshark PCRE comparison operator "matches" to search for a set of strings (keywords are separate by pipe |, group by brackets and enclosed by inverted commas, where \. denotes a dot and (?i) for case-insensitive search):

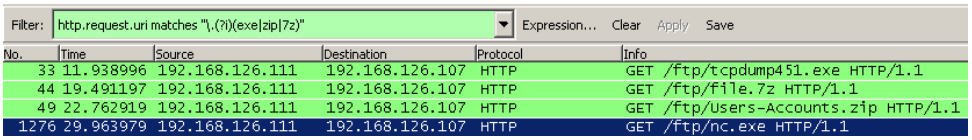
```
ftp matches "\.(?i)(exe|zip|7z|gz|tar|pdf|doc|xls)"
```



It is also possible to search via specific fields name within a protocol, e.g. HTTP web request, with the result shown below:

```
http.request.uri matches "\.(?i)(exe|zip|7z)"
```

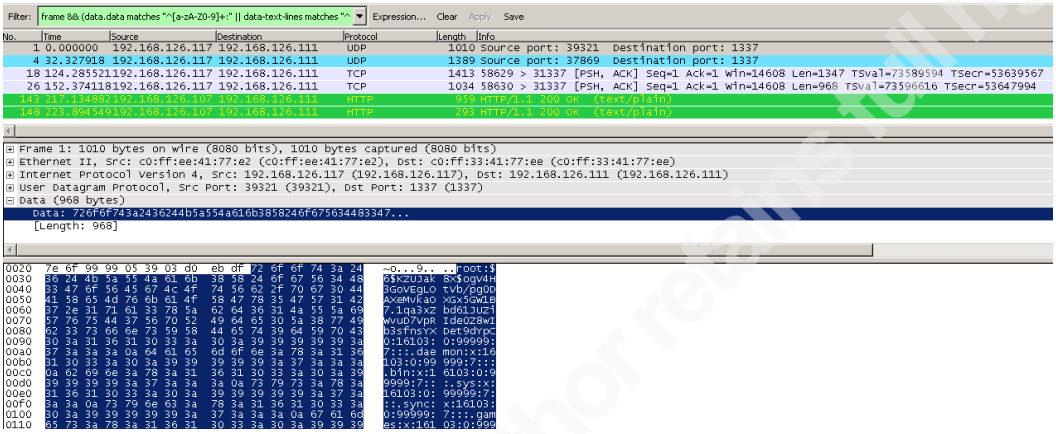

© 2014 The SANS Institute Author retains full rights



2. Search by known content or case insensitive string matching:

Wireshark comparison operator: "matches" enables PCRE string matching. As an example, the following Display filter identifies the data stored in the /etc/passwd and /etc/shadow files:

```
frame and (data.data matches "^[a-zA-Z0-9]+:" ||
data-text-lines matches "^[a-zA-Z0-9]+:")
```

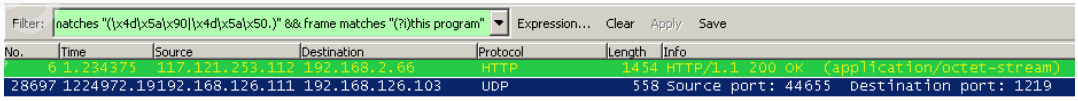


3. Search by file header signature (aka magic number):

Wireshark can also search for specific file types, using known file header signature. As an example, the Windows binary executable file contains the file header value: "MZ" (hex value: 0x4d5a), accompanied by a string of values: "This Program must be run under Win32" or "This program cannot be run in DOS".

The following Wireshark Display filters search for Windows executable file header values (Bruneau, n.d.), and case insensitive text string matching for "This program":

```
frame matches "(\x4d\x5a\x90|\x4d\x5a\x50)"
&& frame matches "(?i)this program"
```



To recover the data file, select the packet in display (from either Packet List or the Data field in the Packet Details pane). Right-click for the option to follow the protocol[9] stream, then filter the conversation stream (uni-direction) to show the data content/binary file, and export the file as a raw file using the "Save-As" option.

[9] For TCP Data Recovery, ensures that the TCP protocol preference is set to "Allow subdissector to reassemble TCP streams". In some instances, the combination use of hex editor and file carving using known file header signature may be required after exporting the data as a raw file.

© 2014 The SANS Institute Author retains full rights.

6.2. Manual Data Recovery of SMTP Data Attachments

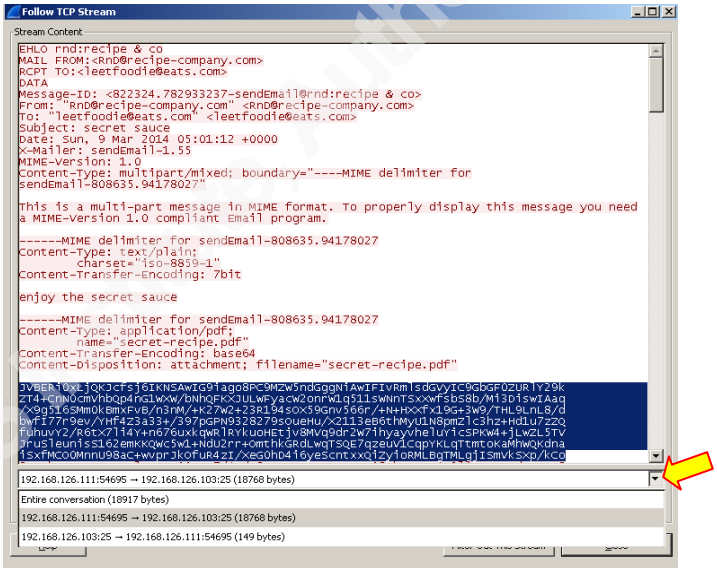
The SMTP email standard is described in RFC822 (Postel, 1982) and its attachments are encoded using Base64 multipurpose content transfer encoding described in RFC2045 (Freed & Borenstein, 1996). Emails can be used as a transport mechanism, sending malicious content via a spear-phishing attack directed at targeted employee(s), or it can be used as a means to exfiltrate data.

Manual recovery of SMTP data attachment requires the following steps:

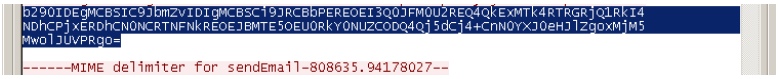
1. Use the following Wireshark filter to locate SMTP packets with attachment(s):

```
smtp matches "(?i)(Content-Transfer-Encoding: base64|attachment)"
```

2. Select the packet in the Packet List pane, and right-click to select the Wireshark option: "Follow TCP stream". In the Wireshark Stream Content window, select the appropriate data stream containing the file attachment (shown in figure below; in Wireshark, Red color text denotes data sent by client, and Blue colortext denotes data sent by the server):



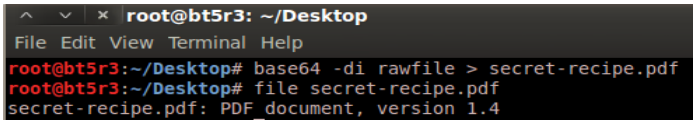
3. The MIME delimiter tag delimits each attachment. It starts after the line "Content-Disposition: …" and terminates near the end of the MIME delimiter tag (as shown in the figure below):



Highlight (left-click and drag) to select the data containing the attachment, then right-click to copy the data. Insert the data into a plaintext file and save the file.

© 2014 The SANS Institute Author retains full rights.

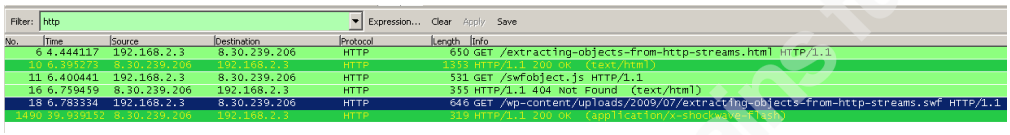
4. It is possible to decode the Base64 encoded data attachment using the base64 utility[10] installed in a Linux workstation (example shown using Backtrack 5-R3[11]) using the command shown in the diagram below:



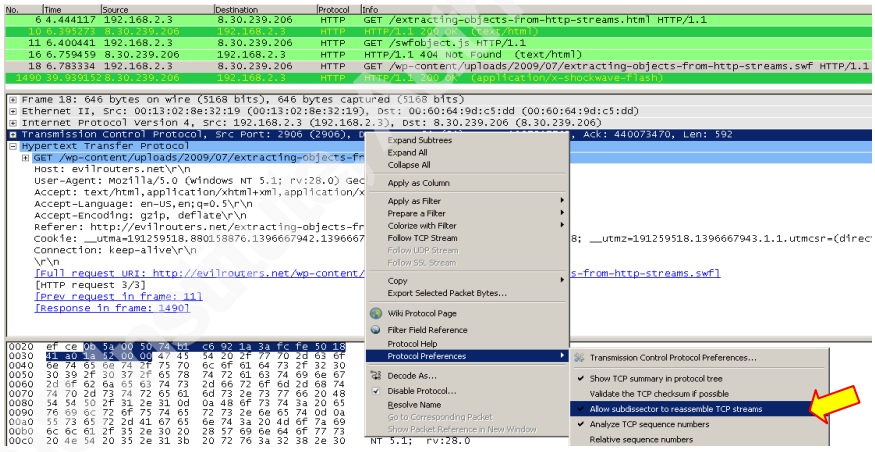
6.3. Data Recovery using Wireshark

For HTTP and SMB related file recovery, use the "Export Object" feature in Wireshark:

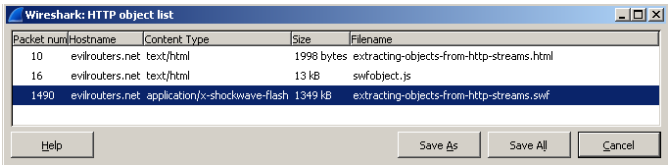
1. Use Wireshark Display filter, http to display web related packets:



2. Right-click on one of the TCP packets (in the Packet List or Packet Details pane) and ensure the TCP "Protocol Preferences" option to "Allow subdissector to reassemble TCP streams" is enabled (as shown in the diagram below):



3. Click on the File Menu, and select "HTTP" within the "Export Objects". Depending on the size of the packet trace file, it may take a while to reassemble the packets and load the HTTP object list shown below. Once it is loaded, select the file and use the save-as option to export the file.



[10] Linux Base64 utility is part of GNU core utilities available via http://ftp.gnu.org/gnu/coreutils/ or standalone Base64 utility for windows available via http://www.fourmilab.ch/webtools/base64/base64.zip

[11] Backtrack 5-R3 Linux Pentesting distribution has been discontinued, but available via: http://ftp.uio.no/pub/security/backtrack/

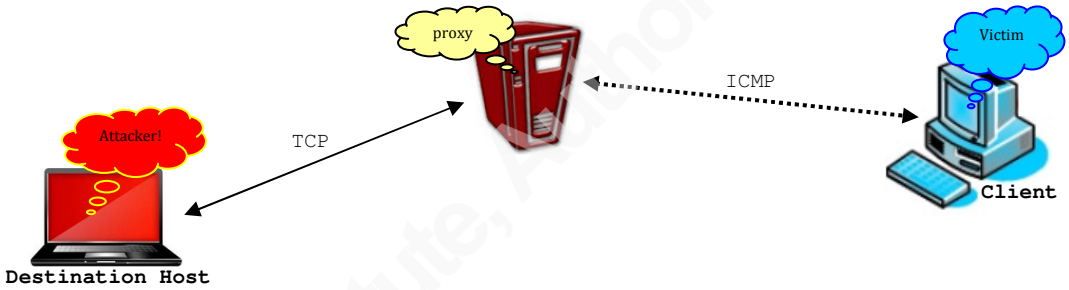
© 2014 The SANS Institute Author retains full rights.

7. Detecting Covert Channels

In an environment that has implemented tight security controls, users may be looking at creative means to access the Internet for their favorite websites. In general, protocol tunneling carries non-legitimate traffic inside allowed protocol traffic to bypass restrictions or network security monitoring. Similarly, attacker(s) can be using tunneling techniques as a backdoor to maintain access to the network, or as a pivot to compromise other systems; allowing them to move within the environment without raising too much attention via such convert channel, and exfiltrate the data that are important to the organization.

7.1. Ping Tunnel (ptunnel): tunneling using ICMP packets

ptunnel is a free and open source software that can be used for tunneling TCP packets over ICMP Echo Request and Reply packets, and can be executed on a Windows or Linux based machines (Stodle, 2011).



As illustrated in the above diagram, the proxy is the bridge between the client and the destination host; communication between the client machine and the proxy host is conducted using ICMP Echo Request/Reply, the proxy host will forward the packets via TCP to the destination host. The ptunnel client software can be executed from the victim or the attacker's machine. However, ptunnel requires the attacker to have control of a machine that is used as a proxy host.

7.2. Wireshark: Detecting SSH over ICMP

The Internet Control Message Protocol (ICMP) is described in RFC777 and is used to communicate transient error condition (Postel, 1981). To determine if a host is online, an ICMP Echo Request (Type 8) is sent to elicit an ICMP Echo Reply (Type 0). These packets are commonly allowed to traverse within and across network perimeter.

© 2014 The SANS Institute Author retains full rights

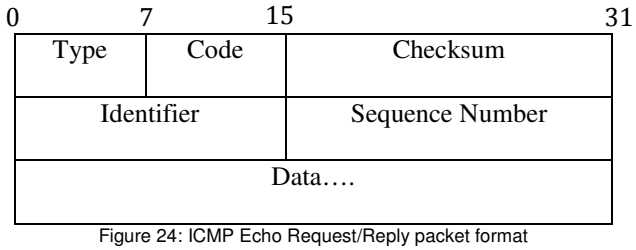


Figure 24: ICMP Echo Request/Reply packet format

To provide an understanding of the fields contained within an ICMP Echo Request/Reply packet, the packet layout is shown in Figure 24. Normally, an ICMP Echo Request sent by a Windows operating system (in Figure 1), and Linux operating system (in Figure 25) contains 32 bytes and 48 bytes of data respectively. Excessive ICMP data size is abnormal and should be checked. As a defense against network mapping, external facing network devices should not be configured to respond to external ICMP request.

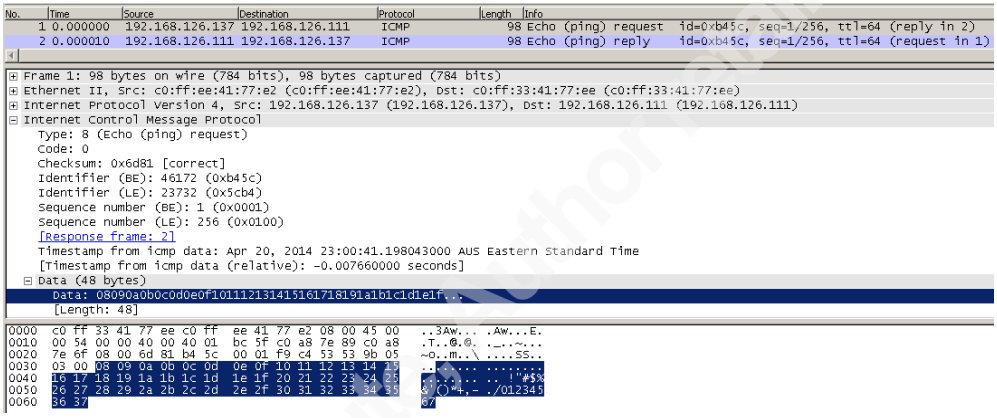


Figure 25: Linux ICMP Echo Request/Reply

Secure Shell (SSH) encrypts data packets in transit and its identification ensures that any unauthorized outbound encrypted traffic running from internal host(s) can be investigated, and prevented from leaving the network. To detect SSH encrypted session over ICMP packet in Wireshark, create a Display filter to list all ICMP packets matching case insensitive string "SSH-": `icmp && data.data matches "(?i)SSH-"`

One of the ICMP Echo Request packets, identified by the above Wireshark Display filter shows a ptunnel packet within its ICMP data section. A ptunnel packet can be identified by its magic hex-value: 0xD5200880 (highlighted in red box), and contained the corresponding ICMP header session identifier number (highlighted by the yellow boxes), shown in Figure 26 below:

© 2014 The SANS Institute Author retains full rights

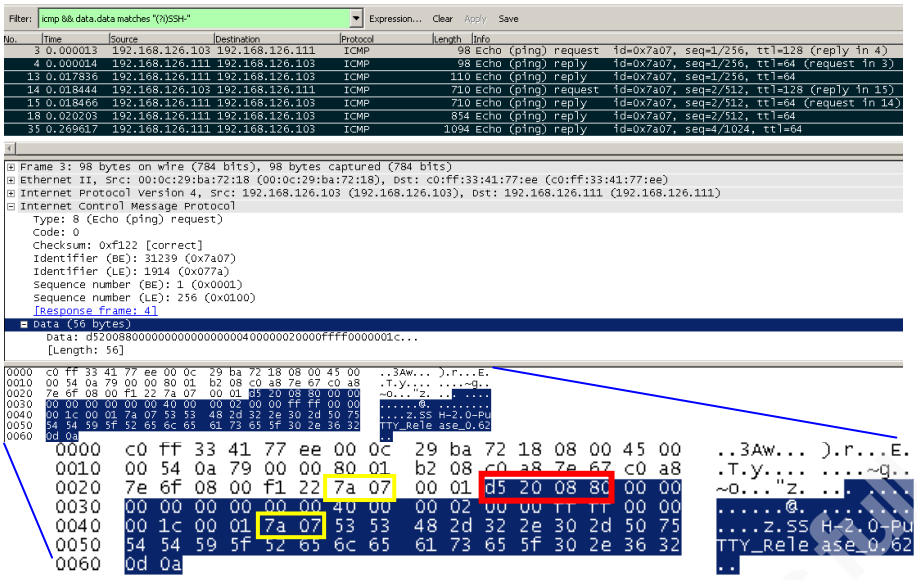


Figure 26: Wireshark Display filter successfully displaying packets related to SSH tunneling over ICMP packets

A review of the packet that initiates the SSH session over ICMP (in Figure 27 below), shows the ptunnel packet within the ICMP data section (magic hex value: 0xD5200880), its final SSH destination host IP, and listening port in hexadecimal value.

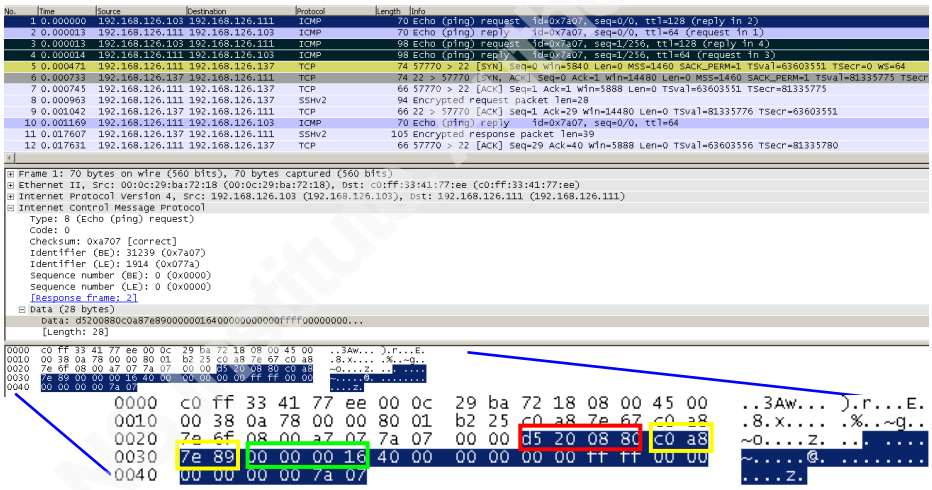


Figure 27: The packet captured via the proxy shows the host that initiated the SSH session in packet 1 (the red box highlights the ptunnel magic value, the yellow box highlights the SSH destination host IP and the green box highlights the SSH destination host port)

Finally as a short-term containment measure, use Wireshark to create an Access Control List (ACL) rule for CISCO network device or supported firewalls to drop incoming or outgoing packets from a specific host by selecting the offending packet from the Packet List pane, and use the 'Firewall ACL rules' option located under the Tools Menu (as shown in Figure 28).

© 2014 The SANS Institute Author retains full rights

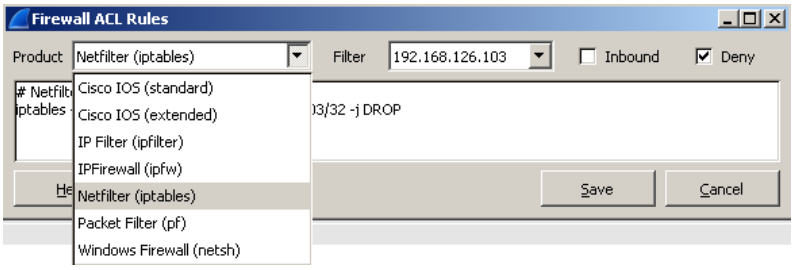


Figure 28: A list of supported firewalls shown in the Wireshark Firewall ACL rules option

8. Conclusion

When it comes to the number of options available for a packet and protocol analyzer, few come close to the number of options for the price offered by Wireshark. While it is not possible to describe every Wireshark option or create filters for every known or unknown attack, we can always seize the opportunity to think outside the box, and re-purpose the features that exist in Wireshark and apply it to our own situation. This paper described the basic features of Wireshark for packet analysis, and the advanced techniques using Coloring rules to deal with simple to complex network interaction and network attacks.

The Wireshark Coloring rules present another option in our detection and defense efforts, limited only by an Incident Responder's knowledge and creativity in developing meaningful Display filters and Coloring rules. It is portable, and can be used systematically to identify unauthorized activities, showing cause in the event of an attack to detect network reconnaissance and vulnerability scans, establishing source and type of attack, identify covert channels used for circumventing controls, and aid in the discovery and recovery of evidence from raw packet trace files. The flexibility of Wireshark Coloring rules is a useful tool to have in the Incident Responder's toolkit for packet analysis, and network forensics.

© 2014 The SANS Institute Author retains full rights

9. References

Bambenek, J. (2008). *Double Flux Defense in the DNS Protocol.* Retrieved May 4, 2014, from http://tools.ietf.org/html/draft-bambenek-doubleflux-01

Bruneau, G. (n.d.). *SANS Institute Hex File Headers and Regex for Forensics Cheat Sheet v1.0.* Retrieved May 4, 2014 from: http://digital-forensics.sans.org/media/hex_file_and_regex_cheat_sheet.pdf

Combs, G. (n.d.). *About Wireshark.* Retrieved May 4, 2014, from http://www.wireshark.org/about.html

Freed, N., & Borenstein, N. (1996). *RFC2045 Multipurpose Internet Mail Extension.* Retrieved May 4, 2014, from http://www.ietf.org/rfc/rfc2045.txt

Imperva. (n.d.). *LAND Attacks.* Retrieved May 4, 2014, from http://www.imperva.com/resources/glossary/land_attacks.html

Lyon, G. F. (2014). *Nmap Changelog.* Retrieved May 4, 2014 from http://nmap.org/changelog.html

Lyon, G. F. (2008). *Nmap network scanning: official Nmap project guide to network discovery and security scanning.* Sunnyvale, CA: Insecure.Com, LLC.

Orebaugh, A., Ramirez, G., Burke, J., Morris, G., Pesce, L., Wright, J. (2007). *Wireshark & Ethereal network protocol analyzer toolkit* (p. 53). Rockland, MA: Syngress

Postel, J. (1981). *RFC777 Internet Control Message Protocol.* Retrieved May 4, 2014, from https://tools.ietf.org/html/rfc777

Postel, J. (1982). *RFC821 Simple Mail Transfer Protocol.* Retrieved May 4, 2014, from http://tools.ietf.org/html/rfc821

Postel, J., & Reynolds, J. (1995). *RFC959 File Transfer Protocol (FTP).* Retrieved May 4, 2014, from http://www.ietf.org/rfc/rfc959.txt

Stodle, D. (2011). *Ping Tunnel For these times when everything else is blocked.* Retrieved May 4, 2014, from http://www.cs.uit.no/~daniels/PingTunnel/

Sullo, C., & Lodge, D. (2012). *Nikto.* Retrieved May 4, 2014, from https://www.cirt.net/node/89

The Sys-Security Group. (n.d.). *Xprobe.* Retrieved May 4, 2014, from http://ofirarkin.wordpress.com/xprobe/

University of Southern California. (1981). *RFC793 Transmission Control Protocol.* Retrieved May 4, 2014, from http://tools.ietf.org/html/rfc793

© 2014 The SANS Institute Author retains full rights.

10. Appendix

Here are some additional Wireshark Display filters that can be added as Coloring Rules:

- Identify IP routing options in use:

  loose source : 0x83
  strict source : 0x89
  record route : 0x07
  timestamp : 0x44

```
(ip.hdr_len > 20) && (ip.opt.type == 0x83|| ip.opt.type == 0x89 ||
ip.opt.type == 0x07 || ip.opt.type == 0x44)
```

- SMB – Identify upload of executable file(s):

```
(smb.create.action == 2) and (smb.file matches "\.(?i)exe")
```

  SMB – Identify executable file(s) being opened:

```
(smb.create.action == 1) and (smb.file matches "\.(?i)exe")
```

  SMB – Identify host operating system information:

```
smb.native_os && smb.native_lanman
```

- Identify data payload sent within a SYN packet, as permitted in section 3.4 of RFC793, (University of Southern California, 1981):

```
(tcp.flags == 0x0002) && (tcp.hdr_len >= 20) && (tcp.segment_data > 0)
```

- Possible fast flux DNS responses, consisting multiple IP addresses with low TTL:

```
(dns.flags.response == 1) && (dns.count.answers >= 5) &&
(dns.resp.ttl < 3600 || dns.resp.ttl < 86400 || dns.resp.ttl <
259200)
```

  The "dns.count.answers" should be set to an arbitrary value of choice or made optional, as the filters also checks for low TTL value that is less than 1hr, 24hrs or 72hrs (Bambenek, 2008).

- A list of some useful HTTP Wireshark Display filters:

```
HTTP requests                          : http.request
HTTP responses                         : http.response
Successful HTTP response               : http.response.code == 200
HTTP client(4xx) or server(5xx)errors  : http.response code > 399
```

  HTTP options via the Statistics menu returns related HTTP information:

  a. HTTP >> Packet counter: returns a list of HTTP request methods and status code responses

  b. HTTP >> Requests: returns a list of request to each Web server

  c. HTTP >> Load distribution: returns the number of requests & responses for each web server

© 2014 The SANS Institute Author retains full rights

- NMAP remote active OS Fingerprinting:

  NMAP sends at least 16 probes to conduct remote active OS fingerprinting. The unique probes documented in the NMAP Network Scanning book (Lyon, 2008) were used to create the corresponding Display filters for the Coloring rules:

  - **TCP Options, Sequence Generation Probes: O1-O6**

    NMAP TCP Option and Sequence Generation probes send 6 TCP SYN packets to an open port, each containing different TCP options. The options for each probe are described below, followed by the corresponding Wireshark compound Display filters:

    O1: window scale (10), NOP, MSS (1460), timestamp (TSval: 0xFFFFFFFF; TSecr: 0), SACK permitted. The window field is 1.

```
O1:
tcp.flags == 0x0002) && (frame[54:3] == 03:03:0a) &&
(tcp.options.mss_val == 1460) &&
(tcp.options.sack_perm == 1) &&
(tcp.options.timestamp.tsval == 4294967295) &&
(tcp.options.timestamp.tsecr == 0) &&
(tcp.window_size_value == 1)


alternatively O1:
(tcp.flags == 0x0002) && (tcp.window_size_value == 1) &&
(tcp.options ==
03:03:0a:01:02:04:05:b4:08:0a:ff:ff:ff:ff:00:00:00:00:04:02)
```

    O2: MSS (1400), window scale (0), SACK permitted, Timestamp (TSval: 0xFFFFFFFF; TSecr: 0), EOL. The window field is 63.

```
O2:
(tcp.flags == 0x0002) && (frame[58:3] == 03:03:00) &&
(tcp.options.mss_val == 1400) &&
(tcp.options.sack_perm == 1) &&
(tcp.options.timestamp.tsval == 4294967295) &&
(tcp.options.timestamp.tsecr == 0)  && (frame[73:1] == 00)
&& (tcp.window_size_value == 63)


Alternatively O2:
(tcp.flags == 0x0002) && (tcp.window_size_value == 63) &&
(tcp.options ==
02:04:05:78:03:03:00:04:02:08:0a:ff:ff:ff:ff:00:00:00:00:00)
```

    O3: Timestamp (TSval: 0xFFFFFFFF; TSecr: 0), NOP, NOP, window scale (5), NOP, MSS (640). The window field is 4.

```
O3:
(tcp.flags == 0x0002) && (tcp.window_size_value == 4) &&
(tcp.options ==
08:0a:ff:ff:ff:ff:00:00:00:00:01:01:03:03:05:01:02:04:02:80)
```


© 2014 SANS Institute, Author retains full rights.

© 2014 The SANS Institute Author retains full rights

O4: SACK permitted, Timestamp (TSval: 0xFFFFFFFF; TSecr: 0), window scale (10), EOL. The window field is 4.

```
O4:
(tcp.flags == 0x0002) && (tcp.window_size_value == 4) &&
(tcp.options ==
04:02:08:0a:ff:ff:ff:ff:00:00:00:00:03:03:0a:00)
```

O5: MSS (536), SACK permitted, Timestamp (TSval: 0xFFFFFFFF; TSecr: 0), window scale (10), EOL. The window field is 16.

```
O5:
(tcp.flags == 0x0002) && (tcp.window_size_value == 16) &&
(tcp.options ==
02:04:02:18:04:02:08:0a:ff:ff:ff:ff:00:00:00:00:03:03:0a:00)
```

O6: MSS (265), SACK permitted, Timestamp (TSval: 0xFFFFFFFF; TSecr: 0). The window field is 512.

```
O6:
(tcp.flags == 0x0002) && (tcp.window_size_value == 512) &&
(tcp.options ==
02:04:01:09:04:02:08:0a:ff:ff:ff:ff:00:00:00:00)
```

- **ICMP Probe: IE1 and IE2**

The ICMP probes are sent immediately after the previous TCP Sequence Generation probes. NMAP sends the first ICMP echo request packet with the IP Don't Fragment flag set, a Type-of-Service (TOS) of value 0, ICMP type 8, Code 9 (instead of 0), random IP identifier, ICMP sequence number: 295, and a data payload containing 120 bytes of 0x00. The first Wireshark filter has the data payload value hard coded 120 times. An alternative filter is provided below using the Wireshark PCRE "matches" comparison operator, nonetheless both filters provide similar result:

```
IE1:
(ip.flags == 0x02) && (ip.dsfield == 0) && (icmp.type == 8)
&& (icmp.code == 9) && (icmp.seq == 295) && (data.len ==
120) && (data.data ==
00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:
00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:
00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:
00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:
00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:
00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00)

alternatively IE1:
(ip.flags == 0x02) && (ip.dsfield == 0) && (icmp.type == 8)
&& (icmp.code == 9) && (icmp.seq == 295) && (data.data
matches "\x00+") && (data.len == 120)
```


© 2014 The SANS Institute Author retains full rights

NMAP sends the second ICMP echo request packet with IP Don't Fragment flag set 0, a Type-of-Service (TOS) of value 4, ICMP type 8 Code 0, IP ID was incremented by 1 from previous value used by IE1, ICMP sequence number: 296, and a data payload of 150 bytes of 0x00. The following Wireshark filter was updated to reflect the changes:

```
IE2:
(ip.flags == 0x00) && (ip.dsfield == 4) && (icmp.type == 8)
&& (icmp.code == 0) && (icmp.seq == 296) && (data.data
matches "\x00+") && (data.len == 150)
```

- **UDP Probe: U1**

NMAP attempts to elicit an ICMP port unreachable packet from a closed UDP port, with a UDP packet consisting of IP identification value of 0x1042, and a data payload containing 300 'C' (0x43) characters.

```
U1:
(udp && not icmp) && (ip.id == 0x1042) && (data.len == 300)
&& (data.data matches "\x43{300}")
```

- **TCP Explicit Congestion Notification Probe: ECN**

NMAP sends a SYN packet to an open TCP port with the Explict Congestion Notification (ECN), Congestion Window Reduced (CWR) and Explict Congestion Notification Echo (ECE) control flag set. An urgent pointer value of 0xF7F5, urgent flag not set, windows size of 3, and TCP options consisting: windows scale (10), No Operation (NOP), Maximum Segment Size, MSS (1460), Selective ACKnowledgement (SACK) permitted, NOP, NOP.

```
ECN:
(tcp.flags == 0x08c2) && (tcp.window_size_value == 3) &&
(tcp.urgent_pointer == 0xf7f5) &&
(tcp.options == 03:03:0a:01:02:04:05:b4:04:02:01:01)
```

- **TCP Probes: T2-T7**

NMAP sends 6 TCP probes: T2-T6 containing the TCP options: 03:03:0a:01:02:04:01:09:08:0a:ff:ff:ff:ff:00:00:00:00:04:02, each of the 6 packets are describe below, with the corresponding Wireshark Display filter:

© 2014 The SANS Institute Author retains full rights

T2: IP Don't Fragment set, TCP null packet and a window field of 128 to an open port.

```
T2:
(ip.flags == 0x02) && (tcp.flags == 0x0000) &&
(tcp.window_size_value == 128) && (tcp.options ==
03:03:0a:01:02:04:01:09:08:0a:ff:ff:ff:ff:00:00:00:00:04:02)
```

T3: IP Don't Fragment is not set, TCP packet with SYN, FIN, URG, and PSH flags set and a window field of 256 to an open port.

```
T3:
(ip.flags == 0x00) && (tcp.flags == 0x002b) &&
(tcp.window_size_value == 256) && (tcp.options ==
03:03:0a:01:02:04:01:09:08:0a:ff:ff:ff:ff:00:00:00:00:04:02)
```

T4: IP Don't Fragment set, TCP ACK packet and a window field of 1024 to an open port.

```
T4:
(ip.flags == 0x02) && (tcp.flags == 0x0010) &&
(tcp.window_size_value == 1024) && (tcp.options ==
03:03:0a:01:02:04:01:09:08:0a:ff:ff:ff:ff:00:00:00:00:04:02)
```

T5: IP Don't Fragment is not set, TCP SYN packet and a window field of 31337 to a closed port.

```
T5:
(ip.flags == 0x00) && (tcp.flags == 0x0002) &&
(tcp.window_size_value == 31337) && (tcp.options ==
03:03:0a:01:02:04:01:09:08:0a:ff:ff:ff:ff:00:00:00:00:04:02)
```

T6: IP Don't Fragment set, TCP ACK packet and a window field of 32768 to a closed port.

```
T6:
(ip.flags == 0x02) && (tcp.flags == 0x0010) &&
(tcp.window_size_value == 32768) && (tcp.options ==
03:03:0a:01:02:04:01:09:08:0a:ff:ff:ff:ff:00:00:00:00:04:02)
```

T7: IP Don't Fragment is not set, TCP packet with FIN, PSH, and URG flags set and a window field of 65535 to a closed port.

```
T7:
(ip.flags == 0x00) && (tcp.flags == 0x0029) &&
(tcp.window_size_value == 65535) && (tcp.options ==
03:03:0f:01:02:04:01:09:08:0a:ff:ff:ff:ff:00:00:00:00:04:02)
```


© 2014 SANS Institute, Author retains full rights.

© 2014 The SANS Institute Author retains full rights

- Xprobe2 is a remote active OS fingerprinting tool developed by Sys-Security Group (Sys-Security Group, n.d.). A series of Wireshark Display filters were created from observation to identify some of the packets related to Xprobe2 version 2.1 (installed as part of the Linux Backtrack 5-R3 Penetrating Testing distribution):

```
(ip.flags == 0x00) && (icmp.type == 8 ) && (icmp.code == 0) &&
(data.len == 48)

(ip.flags == 0x02) && (icmp.type == 8 ) && not (icmp.code == 0) &&
(data.len == 48)

(ip.flags == 0x00) && (icmp.type == 13 && icmp.code == 0) &&
(icmp.seq == 0) && not (icmp[8:4] == 00:00:00:00)

(ip.flags == 0x00) && (icmp.type == 15 && icmp.code == 0) &&
(icmp.seq == 0) && (not data)

(ip.flags == 0x00) && (icmp.type == 17 && icmp.code == 0) &&
(icmp.seq == 0) && (icmp[8:4] == 00:00:00:00)

(udp && not icmp)  && (ip.ttl == 255) && (dns.flags.response == 1)
&& (dns.qry.name == "www.securityfocus.com")

(udp && not icmp )&& (ip.flags == 0x02) && (udp.dstport == 161) &&
(snmp.name == 1.3.6.1.2.1.1.1.0)

(udp && not icmp) && (ip.flags == 0x00) && (udp.srcport == 5555) &&
(udp.dstport == 161)

(ip.flags == 0x02) && (ip.ttl == 1) && (tcp.flags == 0x0002) &&
(tcp.dstport == 65535) && (tcp.window_size_value == 0)

(ip.flags == 0x02) && (tcp.flags == 0x0002) && (tcp.dstport ==
65535)
```


© 2014 SANS Institute, Author retains full rights.

© 2014 The SANS Institute Author retains full rights