Passive System Fingerprinting using Network Client Applications

Jose Nazario
crimelabs research
jose@crimelabs.net

November 27, 2000

Contents

1 Abstract

Passive target fingerprinting involves the utilization of network traffic between two hosts by a third system to identify the types of systems being used. Because no data is sent to either system by the monitoring party, detection approaches the impossible. Methods which rely solely on the IP options present in normal traffic are limited in the accuracy about the targets. Further inspection is also needed to determine avenues of vulnerability, as well. We describe a method to rapidly identify target operating systems and version, as well as vectors of attack, based on data sent by client applications. While simplistic, it is robust. The accuracy of this method is also quite high in most cases. Four methods of fingerprinting a system are presented, with sample data provided.

2 Introduction

Passive OS mapping has become a new area of research in both white hat and black hat arenas. For the white hat, it becomes a new method to map their network and monitor traffic for security. For example, a new and possibly subversive host can be identified quickly, often with great accuracy. For the black hat, this method provides a nearly undetectable method to map a network, finding vulnerable hosts.

To be sure, passive mapping can be a time consuming process. Even with automated tools like Siphon[1] a sufficient quantity packets to arrive to build up a statistically significant reading of the subjects' operating systems. Compare this to active OS fingerprinting methods, using tools like nmap[2] and queso[3], which can operate in under a minute usually, and only more determined attackers, or curious types, will be attracted to this method.

2.1 Current Methods and Research

Two major methods of operating system fingerprinting exist in varying degrees of use, active and passive. Active scanning involves the use of IP packets sent to the host and the scanner then monitoring the replies to guess the operating systems. Passive scanning, in contrast, allows the scanning party to obtain information in the absence of any packets sent from

[1]Available from http://www.subterrain.net/projects/siphon/ .

[2]Available from http://www.insecure.org/nmap/ .

[3]Available from http://www.apostols.org/.

the listening system to the targets. Each method has their advantages, and their limitations.

2.1.1 Active Scanning

By now nearly everyone is familiar with active scanning methods. The premier port scanning tool, nmap, has been equipped for some time now with accurate active scanning measures. This code is based off of an earlier tool, queso, from the group The Apostols. Nmap's author, Fyodor, has written an excellent paper on this topic in the e-zine Phrack (issue 54 article 9)[4]. Ofir Arkin has been using ICMP bit handling to differentiate between certain types of operating systems[5]. Because ICMP usually slips below the threshold of analysis, and most of the ICMP messages used are legitimate, the detection of this scanning can be more difficult than, say, queso or nmap fingerprinting.

The problems with active scanning are mainly twofold: first, we can readily firewall the packets used to fingerprint our system, obfuscating the information; secondly, we can detect it quite easily. Because of this, it is less attractive for a truly stealthy adversary.

2.1.2 Passive Scanning

In a message dated June 30, 1999, Photon posted to the nmap-hackers list with some ideas of passive operating system fingerprinting[6]. He set up a webpage with some of his thoughts, which has since been taken down. In short, by using default IP packet construction behavior, including default TTL values, the presence of the DF bit, and the like, one can gain a confident level of the system's OS.

These ideas were quickly picked up by others and several lines of research have been active since then. Lance Spitzer's paper[7] dated May 24, 2000, on passive fingerprinting included many of the data needed to build such a tool. In fact, two quickly appeared, one from Craig Smith[8], and another tool called p0f from Michael Zalewski[9].

[4]This is definitely a must read to understanding how active, and hence passive, scanning occurs. Obtain this article from http://phrack.infonexus.org/ .

[5]These papers can be found online at http://www.sys-security.com/ .

[6]This note is available from the MARC archives of the nmap-hackers list, at http://marc.theaimsgroup.com/ .

[7]Please see http://www.enteract.com/~lspitz/pubs.html for this paper.

[8]This tool can be found at http://www.enteract.com/~lspitz/passfing.tar.gz .

[9]p0f can be found at http://lcamtuf.hack.pl/p0f.tgz .

One very interesting tool that is under active development, extending the earlier work, is Siphon. By utilizing not only IP stack behavior, but also routing information and spanning tree updates, a complete network map can be built over time. Passive port scans also take place, adding to the data. This tool promises to be truly useful for the white hat, and a patient black hat.

One limitation of these methods, though, is that they only provide a measure of the operating system. Vulnerabilities may or may not exist, and further investigations must be undertaken to evaluate if this is the case. While suitable for the white hat for most purposes (like accounting), this is not suitable to a would-be attacker. Simply put, more information is needed.

2.2 An Alternative Approach

An alternative method to merely fingerprinting the operating system is to perform an identification by using client applications. Quite a number of network clients send revealing information about their host systems, either directly or indirectly. We use application level information to map back to the operating system, either directly or indirectly.

One very large advantage to the method described here is that in some situations, much more accurate information can be gained about the client. Because of stack similarities, most Windows systems, including 95, 98 and NT 4.0, look too similar to differentiate. The client application, however, is willing to reveal this information.

This provides not only a measure of the target's likely operating system, but also a likely vector for entrance. Most of these client applications have numerous security holes, to which one can point malicious data. In some cases, this can provide the key information needed to begin infiltrating a network, and one can proceed more rapidly. In most cases it provides a starting point for the analysis of vulnerabilities of a network.

One major limitation of this method, however, comes when a system is emulating another to provide access to client software. This includes OpenBSD using the BSDI version of the Netscape browser, and both Solaris and SCO's support for Linux binaries. As such, under these circumstances, the data should be taken with some caution and evaluated in the presence of other information. This limitation, however, is similar to the limitation that IP stack tweaking can place on passive fingerprinting at the IP level, or the effect on active scanning from these adjustments or firewalling.

Four different type of network clients are discussed here which provide suitable fingerprinting information. Email clients, which leave telltale information in most cases on their messages; Usenet clients, which, like mail applications, litter their posts with client system information; web browsers, which send client information with each request; and even the ubiquitous telnet client, which sends such information more quietly, but can just as effectively fingerprint an operating system.

Knowing this, one now only needs to harvest the network for this information and map it to source addresses. Various tools, including sniffers, both generic and specialized, and even web searches will yield this information. A rapid analysis of systems can be quickly performed. This works quite well for the white hat and the black hat hacker, as well.

In this paper is described a low tech approach to fingerprinting systems for both their operating system and a likely route to gaining entry. By using application level data sent from them over the network, we can quickly gather accurate data about a system. In some cases, one doesn't even have to be on the same network as the targets, they can gather the information from afar, compile the information and use it at their discretion at a later date.

3 Mail Clients

One of the largest type of traffic the network sees is electronic mail. Nearly everyone who uses the Internet on a regular basis uses email in those transaction sessions. They not only receive mail, but also send a good amount of mail, too. Because it is ubiquitous, it makes an especially attractive avenue for system fingerprinting and ultimately penetration.

Within the headers of nearly every mail message is some form of system identification. Either through the use of crafted message identification tags, as used by Eudora and Pine, or by explicit header information, such as headers generated by OutLook clients or CDE mail clients.

The scope of this method, both in terms of information gained and the potential impact, should not be underestimated. If anything, viruses that spread by email, including ones that are used to steal passwords from systems, should illustrate the effectiveness of this method.

3.1 An Example: Pine

Pine itself is one of the worst offenders of any application for the system it is on. It gives away a whole host of information useful to an attacker in one fell swoop. To wit[10]:

```
Message-ID: <Pine.LNX.4.10.9907191137080.14866-100000@somehost.example.ca>
```

It is clear it's Pine, we know the version (4.10), and we know the system type. Too much about it, in fact. This is a list of the main ports of Pine as of 4.30:

```
a41 IBM RS/6000 running AIX 4.1 or 4.2
a32 IBM RS/6000 running AIX 3.2 or earlier
aix IBM S/370 AIX
aos AOS for IBM RT (untested)
mnt FreeMint
aux Macintosh A/UX
bsd BSD 4.3
bs3 BSDi BSD/386 Version 3 and Version 4
bs2 BSDi BSD/386 Version 2
bsi BSDi BSD/386 Version 1
dpx Bull DPX/2 B.O.S.
cvx Convex
d54 Data General DG/UX 5.4
d41 Data General DG/UX 4.11 or earlier
d-g Data General DG/UX (even earlier)
ult DECstation Ultrix 4.1 or 4.2
gul DECstation Ultrix using gcc compiler
vul VAX Ultrix
os4 Digital Unix v4.0
osf DEC OSF/1 v2.0 and Digital Unix (OSF/1) 3.n
sos DEC OSF/1 v2.0 with SecureWare
epx EP/IX System V
bsf FreeBSD
gen Generic port
hpx Hewlett Packard HP-UX 10.x
hxd Hewlett Packard HP-UX 10.x with DCE security
ghp Hewlett Packard HP-UX 10.x using gcc compiler
hpp Hewlett Packard HP-UX 8.x and 9.x
shp Hewlett Packard HP-UX 8.x and 9.x with Trusted Computer Base
gh9 Hewlett Packard HP-UX 8.x and 9.x using gcc compiler
```

[10] I have tried to sanitize all network addresses or hostnames. If I missed a few, it was inadvertant, and I apologize. You should be running more secure software, anyhow.

isc	Interactive Systems Unix
lnx	Linux using crypt from the C library
lnp	Linux using Pluggable Authentication Modules (PAM)
slx	Linux using -lcrypt to get the crypt function
sl4	Linux using -lshadow to get the crypt() function
sl5	Linux using shadow passwords, no extra libraries
lyn	Lynx Real-Time System (Lynxos)
mct	Tenon MachTen (Mac)
osx	Macintosh OS X
neb	NetBSD
nxt	NeXT 68030's and 68040's Mach 2.0
bso	OpenBSD with shared-lib
sc5	SCO Open Server 5.x
sco	SCO Unix
pt1	Sequent Dynix/ptx v1.4
ptx	Sequent Dynix/ptx
dyn	Sequent Dynix (not ptx)
sgi	Silicon Graphics Irix
sg6	Silicon Graphics Irix >= 6.5
so5	Sun Solaris >= 2.5
gs5	Sun Solaris >= 2.5 using gcc compiler
so4	Sun Solaris <= 2.4
gs4	Sun Solaris <= 2.4 using gcc compiler
sun	Sun SunOS 4.1
ssn	Sun SunOS 4.1 with shadow password security
gsu	SunOS 4.1 using gcc compiler
s40	Sun SunOS 4.0
sv4	System V Release 4
uw2	UnixWare 2.x and 7.x
wnt	Windows NT 3.51

Pine system types used in Message-ID tags as of Pine 4.30. This table was gathered from the supported systems listed in the Pine source code documentation, in the file pine4.30/doc/pine-ports, and was edited for brevity.

Hence, with the above message ID, one knows the target's hostname, an account on that machine that reads mail using Pine, and that it's Linux without shadowed passwords (the LNX host type). Hang out on a mailing list, maybe something platform agnostic, and collect targets. In this case, one could use a well known exploit within the mail message, grab the system password file and send it back to ourselves for analysis. This can easily scaled to as many clients as has been fingerprinted; one mass mailing, and sit back and wait for the password files to come in.

3.2 Other Mail Clients

This is not to say that other mail clients are not vulnerable to such information leaks. Most mail clients give out similar information, either directly or indirectly. Direct information would be an entry in the message headers, such as an X-Mailer tag. Indirect information would be similar to that seen for Pine, a distinctive message ID tag. When this information is coupled to the information about the originating host, a fingerprint can occur rapidly.

Some examples:

```
User-Agent: Mutt/1.2.4i

X-Mailer: Microsoft Outlook Express 5.00.3018.1300
X-MimeOLE: Produced By Microsoft MimeOLE V5.00.3018.1300

X-Mailer: dtmail 1.2.1 CDE Version 1.2.1 SunOS 5.6 sun4u sparc

X-Mailer: PMMail 2000 Professional (2.10.2010) For Windows 2000 (5.0.2195)

X-Mailer: QUALCOMM Windows Eudora Version 4.3.2
Message-ID:  <4.3.2.7.2.20001117142518.043ad100@mailserver3.somewhere.gov>
```

While not all clients give out their host system or processors, such as Mutt or Outlook Express, this information can be used by itself to get a larger vulnerability assessment. For example, if we know what version strings appear only on Windows, as opposed to a MacOS system, we can determine the processor type. The dtmail application is entirely too friendly to someone determining vulnerabilities, giving up the processor and OS revision. Given the problems that have appeared in the CDE suite, and in older versions of Solaris, an attack would be all too easy to construct.

3.3 Finding such information

There are two main avenues for finding this information for lots of clients quickly. First, we can sniff the network for this information. Using a tool like mailsnarf, ngrep or any sniffer with some basic filtering, a modest collection of host to client application data can be gathered. The speed of collection and the ultimate size of this database depends chiefly on the amount of traffic your network segment sees. This is the main drawback to this method, a limited amount of data.

A much more efficient method, and one that can make use of this above information, is in offline (for the target with respect to the potential attacker) system fingerprinting, with an exploit path included. How do we do this? We search the web, with it's repleat mailing list archives, and we turn up some boxes.

```
Altavista: 2,033 pages found (for pine.ult)
Google results 1-10 of about 141,000 for pine.lnx
Altavista: 16,870 pages found (for pine.osf)
```

You get the idea. Tens of thousands of hits, thousands of potentially exploitable boxes ready to be picked. Simply evaluate the source host information and map it to the client data and a large database of vulnerable hosts is rapidly built.

The exploits are easy. Every week, new exploits are found in client software, either mail applications like Pine, or methods to deliver exploits using mail software. Examples of this include the various buffer overflows that have appeared (and persist) in Pine and OutLook, the delivery of malicious DLL files using Eudora attachments, and such. We know from viruses like ILOVEYOU and Melissa that more people than not will open almost any mail message, and we know from spammers that it's trivial to bulk send messages with forged headers, making traceback difficult. These two items combine to make for a very readily available exploit.

4 Usenet Clients

In a manner similar to electronic mail, Usenet clients leave significant information in the headers of their posts which reveal information about their host operating systems. One great advantage to Usenet, as opposed to email or even web traffic, is that posts are distributed. As such, we can be remote and collect data on hosts without their knowledge or ever having to gain entry into their network.

Among the various newsreaders commonly used, copious host info is included in the headers. The popular UNIX newsreader 'tin' is among the worst offenders of revealing host information. Operating system versions, processors and applications are all listed in the 'User-Agent' field, and when coupled to the NNTP-Posting-Host information, a remote host fingerprint has been performed:

```
User-Agent: tin/1.5.2-20000206 ("Black Planet") (UNIX) (SunOS/5.6(sun4u))
User-Agent: tin/pre-1.4-980226 (UNIX) (FreeBSD/2.2.7-RELEASE (i386))
User-Agent: tin/1.4.2-20000205 ("Possession") (UNIX) (Linux/2.2.13(i686))
NNTP-Posting-Host: host.university.edu
```

The standard web browsers also leave copious information about themselves and their host systems, as they do with HTTP requests and mail. We will elaborate on web clients in the next section, but they are also a problem as Usenet clients:

```
X-Http-User-Agent: Mozilla/4.75 [en] (Windows NT 5.0; U)
X-Mailer: Mozilla 4.75 [en] (X11; U; Linux 2.2.16-3smpi686)
```

And several other clients also leave verbose information about their hosts to varying degrees. Again, when combined with the NNTP-Posting-Host or other identifying header, one can begin to amass information about hosts without too much work:

```
Message-ID: <Pine.LNX.4.21.0010261126210.32652-100000@host.example.co.nz>

User-Agent: MT-NewsWatcher/3.0 (PPC)

X-Operating-System: GNU/Linux 2.2.16
User-Agent: Gnus/5.0807 (Gnus v5.8.7) XEmacs/21.1 (Bryce Canyon)

X-Newsreader: Microsoft Outlook Express 5.50.4133.2400

X-Newsreader: Forte Free Agent 1.21/32.243

X-Newsreader: WinVN 0.99.9 (Released Version) (x86 32bit)
```

Either directly or indirectly, we can fingerprint the operating system over the source host. Other programs are not so forthcoming, but still leak information about a host that can be used to determine vulnerability analysis.

```
X-Newsreader: KNode 0.1.13
```

```
User-Agent: Pan/0.9.1 (Unix)

User-Agent: Xnews/03.02.04

X-Newsreader: trn 4.0-test74 (May 26, 2000)

X-Newsreader: knews 1.0b.0 (mrsam/980423)

User-Agent: slrn/0.9.5.7 (UNIX)

X-Newsreader: InterChange (Hydra) News v3.61.08
```

None of these header fields are required by the specifications for NNTP, as noted in RFC 2980. They provide only some additional information about the host which was the source of the data. However, given that more transactions that concern the servers are between servers, this data is entirely extraneous. It is, it appears, absent from RFC 977, the original specification for NNTP.

On interesting possibility to exploiting a user agent like Mozilla is to examine the accepted languages. In the below example, we see not only English is supported, but that the browser is linked to Acrobat. Given potential holes, and past problems[11], with malicious PDF files, this could be another avenue to gaining entry to a host.

```
X-Mailer: Mozilla 4.75 [en] (Win98; U)
X-Accept-Language: en,pdf
```

While this may seem that we're limited to fingerprinting hosts, or out of luck if they are using a proxy, this is not the case. We can also retrieve proxy info from the headers. Recall recent problems with Squid[12]:

```
X-Http-Proxy: 1.0 x72.deja.com:80 (Squid/1.1.22) for client 10.32.34.18
```

While in this case the proxy is disconnected from the client's network, if this were a border proxy, we could use this to gain information about a possible entry point to the network and, over time and with enough sample data, information about the network behind the protected border.

[11] See BUGTRAQ vulnerabilities with ID's 666 and 1509 at http://www.securityfocus.com/ for more information

[12] See BUGTRAQ ID's 471 and 89 at http://www.securityfocus.com/.

5 Using Web Traffic

A remarkably simple and highly effective means of fingerprinting a target is to follow the web browsing that gets done from it. Most every system in use is a workstation, and nearly everyone uses their web browsers to spend part of their day. And just about every browser sends too much information in it's 'User-Agent' field.

RFC 1945[13] notes that the 'User-Agent' field is not required in an HTTP 1.0 request, but can be used. The authors state, "user agents should include this field with requests." They cite statistics as well as on the fly tailoring of data to meet features or limitations of browsers. The draft standard for HTTP version 1.1 requests, RFC 2616, also notes similar usage of the 'User-Agent' field.

We can gather this information in two ways. First, we could run a website and turn on logging of the User-Agent field from the client (if it's not already on). Simply generate a lot of hits and watch the data come in. Get on Slashdot, advertise some pornographic material, or mirror some popular software (like warez) and you're ready to go. Secondly, we can sniff web traffic on our visible segment. While almost any sniffer will work, one of the easiest for this type of work is urlsnarf from the dsniff package from Dug Song[14].

Examples of browsers that send not only their application information, such as the browser and the version, but also the operating system which the host runs include:

```
Netscape (UNIX, MacOS, and Windows)
Internet Explorer
```

One shining example of a browser that doesn't send extraneous information is Lynx. On both 2.7 and 2.8 versions, only the browser information is sent, no information about the host.

The User-Agent field can be important to the web server for legitimate reasons. Due to implementations, both Netscape and Explorer are not equivalent on many items, including how they handle tables, scripting and style sheets. However, host information is not needed and is sent gratuitously.

A typical request from a popular browser looks like this:

[13]This and all other listed RFC's are available from the IETF website at http://www.ietf.org/ .

[14]This package is available at http://www.monkey.org/~dugsong/dsniff/

```
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.08 [en] (X11; I; SunOS 5.7 sun4u)
Host: 10.10.32.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
```

The User-Agent field is littered with extra information that we don't need to know: the operating system type, version and even the hardware being used.

Instantly we know everything there is to know about compromising this host: the operating system, the host's architecture, and even a route we could use to gain entry. For example a recent problems in Netscape's JPEG handling[15].

Using urlsnarf to log these transactions is the easiest method to sniff this information from the network. A typical line of output is below:

```
10.10.1.232 - - [25/Oct/2000:12:55:51 -0400] "GET http://www.latino.com/
HTTP/1.0" - - "http://www.latino.com/" "Mozilla/4.07 [en] (Win95; I ;Nav)"
```

We can also use the tool ngrep[16] to listen to this information on the wire. A simple filter to listen only to packets that contain the information 'User-Agent' can be set up and used to log information about hosts on the network.

A simple regular expression filter can do the trick:

```
ngrep -qid ep1 'User-Agent' tcp port 80
```

This will print out all TCP packets which contain the case insensitive string User-Agent in them. And, within this field, for too many browsers, is too much information about the host. With the above options to ngrep, typical output will look like this:

```
T 10.10.11.43:1860 -> 130.14.22.107:80 [AP]
  GET /entrez/query/query.js HTTP/1.1..Accept: */*..Referer: http://www.
  ncbi.nlm.nih.gov/entrez/query.fcgi?CMD=search&DB=PubMed..Accept-Langua
```

[15] See BUGTRAQ ID 1503 for more information.

[16] ngrep can be obtained from the PacketFactory website, http://www.packetfactory.net/Projects/Ngrep/ .

```
  ge: en-us..Accept-Encoding: gzip, deflate..If-Modified-Since: Thu, 29
  Jun 2000 18:38:45 GMT; length=4558..User-Agent: Mozilla/4.0 (compatibl
  e; MSIE 5.5; Windows 98)..Host: www.ncbi.nlm.nih.gov..Connection: Keep
  -Alive..Cookie: WebEnv=FpEB]AfeA>>Hh^‘Ba@<]^d]bCJfdADh\j=^a=T=EjIE=b<F
  bg<....
```

Even more information is contained within the request than urlsnarf showed us, information including cookies.

5.1 Web Server Fingerprinting

In much the same way as one can use the strings sent during requests by the clients to determine what system type is in use, one can follow the replies sent back by the server to determine what type it is. Again we will use ngrep, this time matching the expression 'server:' to gather the web server type:

```
T 192.168.0.5:80 -> 192.168.0.1:1033 [AP]
  HTTP/1.0 200 OK..Server: Netscape-FastTrack/2.01..Date: Mon, 30 Oct 20
  00 00:15:31 GMT..Content-type: text/html....
```

While specifics about the operating system information are lost, this works to passively gather vulnerability information about the target server. This can be coupled to other information to decide how best to proceed with an attack.

This information will not be covered as this paper is limited to client applications and systems being fingerprinted.

6 Telnet Clients

While telnet is no longer in widespread use due to the fact that all of its data is sent in plain text, including authentication data, it is still used widely enough to be of use in fingerprinting target systems. What is interesting is that it not only gives us a mechanism to gather operating system data, it gives us the particular application in use, which can be of value in determining a mechanism of entry.

This method of system fingerprinting is not unique to this paper. At Hope2k in New York City in the summer of 2000, I saw this demonstrated by a security analyst from Bell Labs. He had a honey-pot system set up that one would telnet to. An application would fingerprint the client and hence the operating system. While I do not recall his name, his research is acknowledged here as my introduction of this method of system fingerprinting.

The specification for the telnet protocol describes a negotiation between the client and the host for information such as line speed, terminal type and echoing[17]. What is interesting to note is that each client behaves in a unique way, even different client applications on the same host type. Similarly, the telnet server, running a telnet daemon, can be fingerprinted by following the negotiations with the client. This information can be viewed from the telnet command line application on a UNIX host by issuing the 'toggle options' command at the telnet⟩ prompt.

This information can be gather directly, using a wedge application, or a honey-pot as demonstrated on the network at Hope2k, or it can be sniffed off the network in a truly passive fashion. We discuss below gathering data about both the client system and the server being connected to. The same principles apply to both host identification methods.

6.1 Fingerprinting Telnet Clients

The negotiations described above, and in the references listed, can be used to fingerprint the client based upon the options set and the order in which they are negotiated. Table 1 describes the behavior of several telnet clients in these respects. Their differences are immediately obvious, even for different clients on the same operating system, such as Tera Term Pro and Windows Telnet on a Windows 95 host.

In this table, all server commands and negotiation options are ignored and only data originating from the client is shown.

Some operating systems, such as IRIX, use a specific and particular terminal type. However, this is usually not a good metric of the operating system, as it can be spoofed or ambiguous, with a value such as vt100 or xterm. Instead, the value and order of various commands sent by the client can be used to distinguish hosts and applications. For example, Windows doesn't set the terminal speed, Linemode options or accept new environmental options. To differentiate between the normal Windows telnet client or Tera Term Pro, one can look for the option to negotiate a window size, for example.

[17]For descriptive information on these options and their negotiations, please see RFCs 857, 858, 859, 860, 1091, 1073, 1079, 1184, 1372, and 1408. Also, see TCP Illustrated, Volume 1: The Protocols by W. Richard Stevens

Table 1: Shown here are the options and the order in which they are sent from three telnet clients. The IRIX 6.5 system was the stock telnet client with no special arguments. The Windows system used was Windows 95B, using either Tera Term Pro 2.3 or the built in telnet client. In all cases an OpenBSD telnet server, with all Kerberos options turned off, was used as the connection target. Data was captured using the tool Ethereal and decoded by the application.

Packet	**IRIX 6.5**	**Windows 95 Tera Term Pro**	**Windows 95 Telnet**
1	Do Supress Go Ahead Will Terminal Type Will Negotiate About Window Size Will Terminal Speed Will Remote Flow Control Will Linemode Do Status Will X Display Location	Will Terminal Type Do Suppress Go Ahead Will Suppress Go Ahead Do Echo Will Negotiate Window Size	Will Terminal Type
2	Suboption: Window Size R\000(Suboption: Linemode Send Your Linemode Do Suppress Go Ahead Suboption: Linemode Send Your Linemode	Won't Terminal Speed	Won't Terminal Speed Won't X Display Location
3	Suboption: Terminal Speed Value: 19200,19200 Suboption: X Display Location workstation:0.0 Suboption: Environment Option \001DISPLAY\000workstation:0.0 Suboption: Terminal Type IRIS-ANSI-NET	Won't X Display Location Won't New Environment Option Suboption: Window Size P\000\030	Won't New Environment Option
4	Won't Echo	Suboption: Terminal Type vt100	Suboption: Terminal Type vt100
5	Do Echo Don't Echo	Won't Echo	Do Suppress Go Ahead
6		Don't Status Won't Remote Flow Control	Will Echo Won't Negotiate Window Size Don't Status Won't Remote Flow Control
7			Won't Echo
8			Do Echo

Space only permits the above three clients to be shown. However, as one can imagine, differences both striking and subtle exist between the various clients.

6.2 Fingerprinting Telnet Servers

Obviously, the most direct method to fingerprint a server would be to connect to it and examine the order of options and their values as a telnet session was negotiated. However, as this study is concerned with passive scanning of clients, we will leave it to the reader to map this information and learn what to do with it.

7 Conclusions

In this paper has been illustrated the effectiveness of target system identification by using the information provided by network client applications. This provides a very efficient and precise measure of the client operating system, as well as identifying a vector for attack. This information is sent gratuitously and is not essential to the normal operation of many of these applications.

The main limitation of this information is found when a host is performing emulation of another operating system to run the client software. While this is rare, it could lead to a false system identification. This mainly falls in the open software world, however, and only for some operating systems.

The scope of this information should not be underestimated. There are some who will note that all one will likely gain on a UNIX system is an unprivilidged account. This may be, however, what we are after, the access that a particular user may have to other valuable data. We may only want their system privilidges, ie for packet generation in a DDoS network. For non-UNIX systems, the impact is well illustrated by the October, 2000, compromise of the Microsoft Corporation network, where access was gained to the source code of Windows and the Office suite. Repeating what is said often, your perimeter is only as strong as its weakest link.

Similarly, there are some that will note that some of these attacks, such as using the mail or Usenet client as a vector for entry, require a bit of social engineering. While this is true, it is by no means any less of a threat. Numerous times we have seen that people will read almost any email message that shows up in their inbox. Usenet engineering is even easier: simply reply to a message posted, such as a question, and the person is almost certain to read the reply.

As such, for the black hat, this represents a quick method of passively gathering target host information as well as a likely vector of attack. For the white hat, it suffices to map a network with respect to operating system and vulnerable application.

7.1 Recommendations for Mitigating the Risks

Would that the world were perfect, or at least software engineers were not prone to errors, this information would not be usable against a host. However, we exist in a world with operating systems littered with security problems and applications that are poorly programmed, ready to exploit. If we lived in an ideal world, but we do not.

For web browsers, which are ubiquitous and used by nearly everyone on the Internet, the host operating system should not be sent. Ideally, information about what protocols are spoken, what standards are met and what language are supported (ie English, German, French) should suffice. Lynx behaves nearly ideally in this regard, and both Netscape and Explorer should follow this lead.

With respect to Usenet and electronic mail clients, again only what features are supported should be provided. Pine is an example of how bad it can get, providing too much information about a host too quickly. There is no reason why any legitimate client should know what processor and OS is being run on the sending host.

Telnet clients are far more difficult. It is tempting to say that all telnet applications should support the same set of features, but that is simply impossible.

Proxy hosts should be used, if possible, to strip off information about the originating system, including the workstation address and operating system information. This will help obscure needed information to map a network from outside the perimeter. Coupled with strong measures to catch viruses and malicious code, such as in a web page script, the risks should be greatly reduced.

The best solution is for application authors to not send gratuitous information in their headers or requests. Furthermore, client applications should be scrutinized to the same degree as daemons that run with administrative privilidges. The lessons of RFC 1123 most certainly apply at this level.

In the intervening time, those with access to the source code of their network clients may want to consider removing gratuitous host information from their request packets or headers. This, however, doesn't apply to most users, and those that know about this method already practice this routinely.

8 Acknowledgments

This work was inspired largely by Photon's post in 1999 to the nmap-hackers list. It seems a great deal of other research, and proof of concept tools, has been initiated by this message. To them I extend my thanks, they've helped to provide me with hours of diversions and thought experiments. Also, I am thankful to the authors of the tools used in this study, especially Dug Song for his dsniff package, and Jordan Ritter for ngrep. As always, I am indebted to the people I work with at crimelabs, as well, especially Kosher Egyptian Rabbit and Jesus, with whom I have had numerous productive conversations on this topic. Rick Wash and Merlin were most helpful in their critical review of this manuscript and their helpful suggestions.