Analysis of Remote Active Operating System Fingerprinting Tools

Ryan Spangler
University of Wisconsin - Whitewater
Department of Computer Science
ryan@packetwatch.net

May 2003

Abstract
There are many tools today that are used for remote active operating system fingerprinting. They all have their own fingerprinting techniques. This paper gives an in-depth analysis of three such tools: Nmap, RINGv2, and Xprobe2. The purpose of the paper is to show how these tools work, and to understand the advantages and disadvantages they each offer.

Table of Contents

1.0 Introduction

Remote active operating system fingerprinting is the process of determining the identity of a remote host's operating system. This is done by actively sending packets to the remote host and analyzing the responses. Tools like `Nmap` and `Xprobe2` take the responses and form a fingerprint that can be queried against a signature database of known operating systems.

Learning which operating system is running on a remote host can be very valuable for both pen-testers and black-hats. It's valuable because when vulnerabilities are found they are normally dependent on the OS version.

Originally, determining the OS on the remote host was done by a technique known as "banner grabbing". Banner grabbing consists of either looking at the banner displayed when trying to connect to a service like ftp or by downloading a binary file like /bin/ls to determine what architecture it was built for.

Eventually, more advanced techniques based on stack querying came about. Stack querying means to actively send packets to the network stack on the remote host and analyze the responses. This idea takes advantage of each OS vendor's network stack implementation. The first method to use stack querying was aimed at the TCP stack. It involves sending standard and non-standard TCP packets to the remote host and analyzing the responses. The next method was known as ISN (Initial Sequence Number) analysis[1]. This identifies the differences in the random number generators found in the TCP stack. Up until that point all of the stack querying methods were found by looking at the TCP protocol. Later the same year, researchers found a new method that used the ICMP protocol. The method is known as ICMP response analysis. It involves sending ICMP messages to the remote host and analyzing the responses. The newest method is called temporal response analysis. Like others this method uses the TCP protocol. Temporal response analysis looks at the retransmission timeout (RTO) responses from a remote host.

This paper presents an in-depth analysis of three remote active OS fingerprinting tools. I will be explaining how each of the different OS detection methods work in order to identify the OS running on the remote host. The goal of the paper is to show how the tools work, and to understand the advantages and disadvantages they each offer.

[1] A more in depth explanation of ISN analysis can be found at http://lcamtuf.coredump.cx/newtcp/.

2.0 Nmap Tool

`Nmap` is a network exploration tool and security scanner. It is designed to allow users to scan networks to determine which hosts are up and what services they offer. `Nmap` supports a number of scanning techniques that use the following protocols: TCP, ICMP, UDP, and IP. `Nmap` also includes features like remote OS detection, parallel scanning, port filtering detection, timing options, and flexible target and port specification.

For the purpose of this paper I'll be focusing on the remote OS detection method that `Nmap` uses. I'll be using `Nmap` version 3.27 since it was the current version at the time of this writing.

2.1 Technique

Before `Nmap` runs it's OS detection method it runs a port scan against the target machine. It performs a port scan so it can find some open and closed ports on the target machine. `Nmap` works best when it finds at least one open TCP port, one closed TCP port, and one closed UDP port. `Nmap` works by conducting a set of tests against the target machine to try to determine what OS it is running.

The first test is named T1 for test 1. In this test a TCP packet with the SYN, and ECN-Echo flags enabled is sent to an open TCP port.

The second test is named T2 for test 2. It involves sending a TCP packet with no flags enabled to an open TCP port. This type of packet is known as a NULL packet.

The third test is named T3 for test 3. It involves sending a TCP packet with the URG, PSH, SYN, and FIN flags enabled to an open TCP port.

The fourth test is named T4 for test 4. It involves sending a TCP packet with the ACK flag enabled to an open TCP port.

The fifth test is named T5 for test 5. It involves sending a TCP packet with the SYN flag enabled to a closed TCP port.

The sixth test is named T6 for test 6. It involves sending a TCP packet with the ACK flag enabled to a closed TCP port.

The seventh test is named T7 for test 7. It involves sending a TCP packet with the URG, PSH, and FIN flags enabled to a closed TCP port.

The eighth test is named PU for port unreachable test. It involves sending a UDP packet to a closed UDP port. The goal is to elicit an ICMP packet with an ICMP port unreachable message back from the target machine.

The last test that `Nmap` performs is named TSeq for TCP sequenceability test. The test tries to determine the sequence generation patterns of the TCP initial sequence numbers also known as TCP ISN sampling, the IP identification numbers also known as IPID sampling, and the TCP timestamp numbers. The test is performed by sending six TCP packets with the SYN flag enabled to an open TCP port.

After Nmap has received the results from all of the tests it builds an OS fingerprint signature, then tries to find a match in the fingerprint database. If the signature is found in the database, Nmap will list it as the Remote OS guess. If the signature isn't found in the database, Nmap will display a message saying "No exact matches for host", and tells you that if you know what the OS is running on it to go to http://www.insecure.org/cgi-bin/nmap-submit.cgi.

2.2 OS Signature

Based on the results (information in the packets sent back to us) of the tests that Nmap performs, an OS signature is built. I will now show an example signature from the database, and then explain what it means.

```
Fingerprint Solaris 9 Beta through Release on SPARC
TSeq(Class=RI%gcd=<6%SI=<A927C&>116A%IPID=I%TS=100HZ)
T1(DF=Y%W=C0B7|807A%ACK=S++%Flags=AS%Ops=NNTMNW)
T2(Resp=N)
T3(Resp=N)
T4(DF=Y%W=0%ACK=O%Flags=R%Ops=)
T5(DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=Y%W=0%ACK=O%Flags=R%Ops=)
T7(DF=Y%W=0%ACK=S%Flags=AR%Ops=)
PU(DF=Y%TOS=0%IPLEN=70%RIPTL=148%RID=E%RIPCK=E%UCK=E|F%ULEN=134%DAT=E)
```

I will go through the signature line by line.

```
Fingerprint Solaris 9 Beta through Release on SPARC
```

This signature covers Solaris 9 for the SPARC platform from the Beta through the Release version.

Figure 1 is included to show the TCP header which will be referred to in the results of some of the tests.

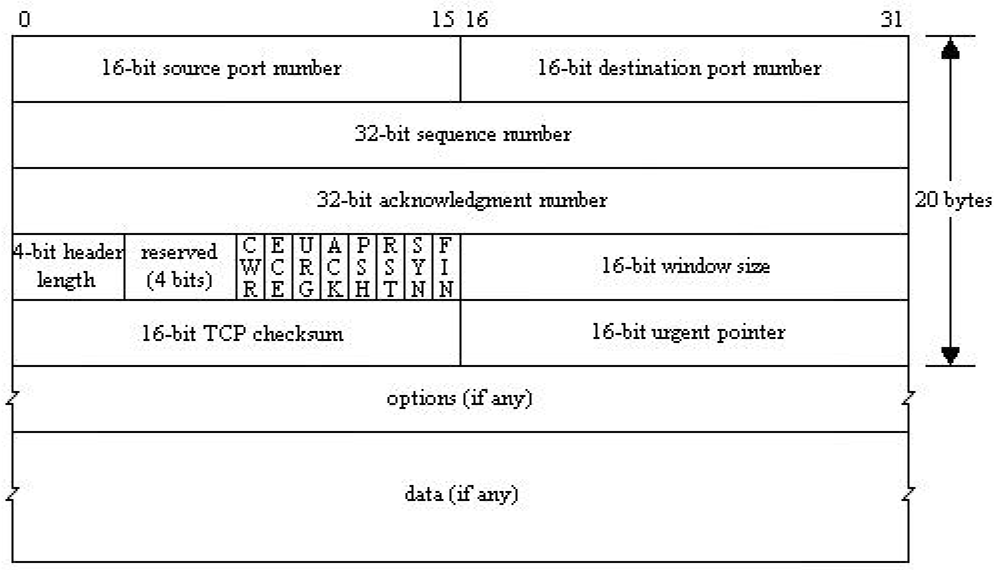

Figure 1: TCP header

```
TSeq(Class=RI%gcd=<6%SI=<A927C&>116A%IPID=I%TS=100HZ)
```

This line represents the results of the TCP sequenceability test. Remember that this test tries to determine the generation patterns of the TCP initial sequence numbers, the IP identification numbers, and the TCP timestamp numbers. Class, gcd, and SI all come from TCP ISN sampling. Class=RI means that it falls in the "Random Increments" class which means that each new TCP sequence number increases in such a way that it is difficult to guess precisely from the last one. gcd=<6 means that the greatest common divisor is less than six, which means that the sequence numbers don't have any large factors in common. SI=<A927C&>116A means that the sequence increments are less than 0xA927C in hexadecimal or 692860, and greater than 0x116A in hexadecimal or 4458. The sequence increments value is a statistical measure of variance meaning that the higher the number the more random looking the sequence numbers tend to be. IPID comes from IPID sampling. IPID=I means that the IP identification numbers fall in the "Incremental" class meaning they increase by one for each packet sent. TS comes from the TCP timestamp sampling. TS=100HZ means that the TCP timestamp option seems to increment by about 100 every second.

```
T1(DF=Y%W=C0B7|807A%ACK=S++%Flags=AS%Ops=NNTMNW)
```

This line represents the results of test 1. DF=Y means that the Don't fragment flag in the IP header was enabled. W=C0B7|807A means that the window size in the TCP header is either 0xC0B7 (49335) or 0x807A (32890) in hexadecimal. ACK=S++ means that the acknowledgment number in the TCP header is our initial sequence number plus 1. Flags=AS means that the ACK and SYN flags in the TCP header were enabled. Ops=NNTMNW means that the following options were in this order: <NOP><NOP><Timestamp><Maximum segment size (MSS)><NOP><Window scale>.

```
T2(Resp=N)
```

This line represents the results of test 2. Resp=N means that we did not get a response back.

```
T3(Resp=N)
```

This line represents the results of test 3. Resp=N means that we did not get a response back.

```
T4(DF=Y%W=0%ACK=O%Flags=R%Ops=)
```

This line represents the results of test 4. DF=Y means that the Don't fragment flag in the IP header was enabled. W=0 means that the window size in the TCP header is 0. ACK=O means that the acknowledgment number in the TCP header is 0. Flags=R means that the RST flag in the TCP header was enabled. Ops= means that there weren't any options sent back.

```
T5(DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
```

This line represents the results of test 5. DF=Y means that the Don't fragment flag in the IP header was enabled. W=0 means that the window size in the TCP header is 0. ACK=S++ means that the acknowledgment number in the TCP header is our initial sequence number plus 1. Flags=AR means that the ACK and RST flags in the TCP header were enabled. Ops= means that there weren't any options sent back.

```
T6(DF=Y%W=0%ACK=O%Flags=R%Ops=)
```

This line represents the results of test 6. DF=Y means that the Don't fragment flag in the IP header was enabled. W=0 means that the window size in the TCP header is 0. ACK=O means that the acknowledgment number in the TCP header is 0. Flags=R means that the RST flag in the TCP header was enabled. Ops= means that there weren't any options sent back.

```
T7(DF=Y%W=0%ACK=S%Flags=AR%Ops=)
```

This line represents the results of test 7. DF=Y means that the Don't fragment flag in the IP header was enabled. W=0 means that the window size in the TCP header is 0. ACK=S means that the acknowledgment number in the TCP header is our initial sequence number. Flags=AR means that the ACK and RST flags in the TCP header were enabled. Ops= means that there weren't any options sent back.

```
PU(DF=Y%TOS=0%IPLEN=70%RIPTL=148%RID=E%RIPCK=E%UCK=E|F%ULEN=134%DAT=E)
```

This line represents the results of the port unreachable test. DF=Y means that the Don't fragment flag in the IP header was enabled. TOS=0 means that the type of service (TOS) in the IP header was 0. IPLEN=70 means that the total length in the IP header is 0x0070 in hexadecimal or 112. RIPTL=148 means that the total length given in the IP header sent back to us is 0x0148 in hexadecimal or 328. RID=E means that the identification number in the IP header we got back in the copy of our original UDP packet was the same as what we sent. RIPCK=E means that the header checksum in the IP header was the same. UCK=E|F means that the UDP checksum in the UDP header sometimes is found to be the same and sometimes not. ULEN=134 means that the UDP length in the UDP header is 0x0134 in hexadecimal or 308. DAT=E means that the UDP data was echoed back correctly. Since most implementations don't send any UDP data back, they get DAT=E by default.

2.3 Advantages

The `Nmap` tool offers many advantages over other remote active OS fingerprinting tools. Here are some reasons users might want to utilize this tool for its OS detection method:

- "Half-open" scanning[2] is supported. The TCP SYN scan is one of these scan types. The advantage to this scan type is that it doesn't complete the TCP three-way handshake which means that it will often not be logged by basic intrusion detection systems.
- The tests that `Nmap` performs include multiple flags and protocols so that it can still make an OS guess even if filtering devices block some of the tests. Some tests might not work because filtering devices might block protocols like ICMP from entering their network.
- There is a large OS signature database. For example, the latest `Nmap` (3.27) recognizes 867 fingerprints while the latest `Xprobe2` (0.1) recognizes 53.
- There is a lot of support for network devices like routers, firewalls, switches, and printers.
- Strict signature matching is utilized by default, while fuzzy matching can be enabled by specifying the undocumented --fuzzy option.
- It has built-in learning functions.

[2] "Half-open" scanning is a form of port scanning where the client terminates the connection before the TCP three-way handshake can be completed.

2.4 Usage Example

The sample run was produced utilizing an OpenBSD 3.2 machine running `Nmap` targeting a FreeBSD 4.7-RELEASE machine on the same local LAN. Starting here and throughout the rest of the paper `Tcpdump` [3] will be used to display the network traffic.

The following is a sample run that `Nmap` produced:

```
# nmap –sS –O –v –p 22,23 10.10.1.8

Starting nmap 3.27 ( www.insecure.org/nmap/ ) at 2003-05-05 17:09 CDT
Host mars.packetwatch.net (10.10.1.8) appears to be up ... good.
Initiating SYN Stealth Scan against mars.packetwatch.net (10.10.1.8) at 17:09
Adding open port 22/tcp
The SYN Stealth Scan took 0 seconds to scan 2 ports.
For OSScan assuming that port 22 is open and port 23 is closed and neither are
firewalled
Interesting ports on mars.packetwatch.net (10.10.1.8):
(The 1 port scanned but not shown below is in state: closed)
Port       State       Service
22/tcp     open        ssh
Remote operating system guess: FreeBSD 4.6.2-RELEASE - 4.7-STABLE
Uptime 0.045 days (since Mon May  5 16:04:16 2003)
TCP Sequence Prediction: Class=truly random
                         Difficulty=9999999 (Good luck!)
IPID Sequence Generation: Incremental

Nmap run completed -- 1 IP address (1 host up) scanned in 7.961 seconds
```

The `Tcpdump` output for the T1 test:

```
17:09:01.367448 10.10.1.3.37782 > 10.10.1.8.22: SE [tcp sum ok]
345889334:345889334(0) win 3072 <wscale 10,nop,mss 265,timestamp 1061109567
0,eol> (ttl 58, id 32560, len 60)
                     4500 003c 7f30 0000 3a06 eb6d 0a0a 0103
                     0a0a 0108 9396 0016 149d da36 0000 0000
                     a042 0c00 2456 0000 0303 0a01 0204 0109
                     080a 3f3f 3f3f 0000 0000 0000
17:09:01.368027 10.10.1.8.22 > 10.10.1.3.37782: S [tcp sum ok]
1496601119:1496601119(0) ack 345889335 win 57344 <mss 1460,nop,wscale
0,nop,nop,timestamp 388530 1061109567> (DF) (ttl 64, id 3348, len 60)
                     4500 003c 0d14 4000 4006 178a 0a0a 0108
                     0a0a 0103 0016 9396 5934 521f 149d da37
                     a012 e000 bacd 0000 0204 05b4 0103 0300
                     0101 080a 0005 edb2 3f3f 3f3f
```

The `Tcpdump` output for the T2 test:

```
17:09:01.367578 10.10.1.3.37783 > 10.10.1.8.22: . [tcp sum ok]
345889334:345889334(0) win 4096 <wscale 10,nop,mss 265,timestamp 1061109567
0,eol> (ttl 59, id 25998, len 60)
                     4500 003c 658e 0000 3b06 0410 0a0a 0103
                     0a0a 0108 9397 0016 149d da36 0000 0000
                     a000 1000 2097 0000 0303 0a01 0204 0109
                     080a 3f3f 3f3f 0000 0000 0000
```

[3] More information about Tcpdump can be found at http://www.tcpdump.org/.

The `Tcpdump` output for the T3 test:

```
17:09:01.367689 10.10.1.3.37784 > 10.10.1.8.22: SFP [tcp sum ok]
345889334:345889334(0) win 4096 urg 0 <wscale 10,nop,mss 265,timestamp
1061109567 0,eol> (ttl 55, id 26761, len 60)
                     4500 003c 6889 0000 3706 0515 0a0a 0103
                     0a0a 0108 9398 0016 149d da36 0000 0000
                     a02b 1000 206b 0000 0303 0a01 0204 0109
                     080a 3f3f 3f3f 0000 0000 0000
17:09:01.368171 10.10.1.8.22 > 10.10.1.3.37784: S [tcp sum ok]
1146976671:1146976671(0) ack 345889335 win 57344 <mss 1460,nop,wscale
0,nop,nop,timestamp 388530 1061109567> (DF) (ttl 64, id 3349, len 60)
                     4500 003c 0d15 4000 4006 1789 0a0a 0108
                     0a0a 0103 0016 9398 445d 799f 149d da37
                     a012 e000 a822 0000 0204 05b4 0103 0300
                     0101 080a 0005 edb2 3f3f 3f3f
```

The `Tcpdump` output for the T4 test:

```
17:09:01.367798 10.10.1.3.37785 > 10.10.1.8.22: . [tcp sum ok]
345889334:345889334(0) ack 0 win 4096 <wscale 10,nop,mss 265,timestamp
1061109567 0,eol> (ttl 59, id 32437, len 60)
                     4500 003c 7eb5 0000 3b06 eae8 0a0a 0103
                     0a0a 0108 9399 0016 149d da36 0000 0000
                     a010 1000 2085 0000 0303 0a01 0204 0109
                     080a 3f3f 3f3f 0000 0000 0000
17:09:01.368637 10.10.1.8.22 > 10.10.1.3.37785: R [tcp sum ok] 0:0(0) win 0
(ttl 64, id 3350, len 40)
                     4500 0028 0d16 0000 4006 579c 0a0a 0108
                     0a0a 0103 0016 9399 0000 0000 0000 0000
                     5004 0000 0613 0000 7e7e 7e7e 7e7e
```

The `Tcpdump` output for the T5 test:

```
17:09:01.367907 10.10.1.3.37786 > 10.10.1.8.23: S [tcp sum ok]
345889334:345889334(0) win 1024 <wscale 10,nop,mss 265,timestamp 1061109567
0,eol> (ttl 48, id 33179, len 60)
                     4500 003c 819b 0000 3006 f302 0a0a 0103
                     0a0a 0108 939a 0017 149d da36 0000 0000
                     a002 0400 2c91 0000 0303 0a01 0204 0109
                     080a 3f3f 3f3f 0000 0000 0000
17:09:01.369716 10.10.1.8.23 > 10.10.1.3.37786: R [tcp sum ok] 0:0(0) ack
345889335 win 0 (ttl 64, id 3351, len 40)
                     4500 0028 0d17 0000 4006 579b 0a0a 0108
                     0a0a 0103 0017 939a 0000 0000 149d da37
                     5014 0000 172c 0000 7e7e 7e7e 7e7e
```

The `Tcpdump` output for the T6 test:

```
17:09:01.368064 10.10.1.3.37787 > 10.10.1.8.23: . [tcp sum ok]
345889334:345889334(0) ack 0 win 3072 <wscale 10,nop,mss 265,timestamp
1061109567 0,eol> (ttl 42, id 21753, len 60)
                     4500 003c 54f9 0000 2a06 25a5 0a0a 0103
                     0a0a 0108 939b 0017 149d da36 0000 0000
                     a010 0c00 2482 0000 0303 0a01 0204 0109
                     080a 3f3f 3f3f 0000 0000 0000
17:09:01.369784 10.10.1.8.23 > 10.10.1.3.37787: R [tcp sum ok] 0:0(0) win 0
(ttl 64, id 3352, len 40)
                     4500 0028 0d18 0000 4006 579a 0a0a 0108
                     0a0a 0103 0017 939b 0000 0000 0000 0000
                     5004 0000 0610 0000 7e7e 7e7e 7e7e
```

The `Tcpdump` output for the T7 test:

```
17:09:01.368392 10.10.1.3.37788 > 10.10.1.8.23: FP [tcp sum ok]
345889334:345889334(0) win 1024 urg 0 <wscale 10,nop,mss 265,timestamp
1061109567 0,eol> (ttl 40, id 20620, len 60)
                     4500 003c 508c 0000 2806 2c12 0a0a 0103
                     0a0a 0108 939c 0017 149d da36 0000 0000
                     a029 0400 2c68 0000 0303 0a01 0204 0109
                     080a 3f3f 3f3f 0000 0000 0000
17:09:01.369843 10.10.1.8.23 > 10.10.1.3.37788: R [tcp sum ok] 0:0(0) ack
345889334 win 0 (ttl 64, id 3353, len 40)
                     4500 0028 0d19 0000 4006 5799 0a0a 0108
                     0a0a 0103 0017 939c 0000 0000 149d da36
                     5014 0000 172b 0000 7e7e 7e7e 7e7e
```

The `Tcpdump` output for the port unreachable test:

```
17:09:01.368674 10.10.1.3.37775 > 10.10.1.8.23: udp 300 (ttl 58, id 28293, len
328)
                     4500 0148 6e85 0000 3a11 fb01 0a0a 0103
                     0a0a 0108 938f 0017 0134 2694 5757 5757
                     5757 5757 5757 5757 5757 5757 5757 5757
                     5757 5757 5757 5757 5757 5757 5757 5757
                     5757 5757 5757 5757 5757 5757 5757 5757
                     5757
17:09:01.369930 10.10.1.8 > 10.10.1.3: icmp: 10.10.1.8 udp port 23 unreachable
for 10.10.1.3.37775 > 10.10.1.8.23: [no cksum] udp 300 (ttl 58, id 28293, len
328) (ttl 64, id 3354, len 56)
                     4500 0038 0d1a 0000 4001 578d 0a0a 0108
                     0a0a 0103 0303 6822 0000 0000 4500 0148
                     6e85 0000 3a11 fb01 0a0a 0103 0a0a 0108
                     938f 0017 0134 0000
```

The `Tcpdump` output for the TCP sequenceability test:

```
17:09:05.260147 10.10.1.3.37776 > 10.10.1.8.22: S [tcp sum ok]
345889335:345889335(0) win 4096 <wscale 10,nop,mss 265,timestamp 1061109567
0,eol> (ttl 59, id 25222, len 60)
                     4500 003c 6286 0000 3b06 0718 0a0a 0103
                     0a0a 0108 9390 0016 149d da37 0000 0000
                     a002 1000 209b 0000 0303 0a01 0204 0109
                     080a 3f3f 3f3f 0000 0000 0000
17:09:05.260640 10.10.1.8.22 > 10.10.1.3.37776: S [tcp sum ok]
3836437649:3836437649(0) ack 345889336 win 57344 <mss 1460,nop,wscale
0,nop,nop,timestamp 388919 1061109567> (DF) (ttl 64, id 3355, len 60)
                     4500 003c 0d1b 4000 4006 1783 0a0a 0108
                     0a0a 0103 0016 9390 e4ab 6491 149d da38
                     a012 e000 1b64 0000 0204 05b4 0103 0300
                     0101 080a 0005 ef37 3f3f 3f3f
```

```
17:09:05.260758 10.10.1.3.37776 > 10.10.1.8.22: R [tcp sum ok]
345889336:345889336(0) win 0 (DF) (ttl 64, id 25086, len 40)
                         4500 0028 61fe 4000 4006 c2b3 0a0a 0103
                         0a0a 0108 9390 0016 149d da38 0000 0000
                         5004 0000 1746 0000
17:09:05.684526 10.10.1.3.37777 > 10.10.1.8.22: S [tcp sum ok]
345889336:345889336(0) win 4096 <wscale 10,nop,mss 265,timestamp 1061109567
0,eol> (ttl 51, id 7781, len 60)
                         4500 003c 1e65 0000 3306 5339 0a0a 0103
                         0a0a 0108 9391 0016 149d da38 0000 0000
                         a002 1000 2099 0000 0303 0a01 0204 0109
                         080a 3f3f 3f3f 0000 0000 0000
17:09:05.684983 10.10.1.8.22 > 10.10.1.3.37777: S [tcp sum ok]
1270573067:1270573067(0) ack 345889337 win 57344 <mss 1460,nop,wscale
0,nop,nop,timestamp 388961 1061109567> (DF) (ttl 64, id 3356, len 60)
                         4500 003c 0d1c 4000 4006 1782 0a0a 0108
                         0a0a 0103 0016 9391 4bbb 680b 149d da39
                         a012 e000 b0ae 0000 0204 05b4 0103 0300
                         0101 080a 0005 ef61 3f3f 3f3f
17:09:05.685083 10.10.1.3.37777 > 10.10.1.8.22: R [tcp sum ok]
345889337:345889337(0) win 0 (DF) (ttl 64, id 10628, len 40)
                         4500 0028 2984 4000 4006 fb2d 0a0a 0103
                         0a0a 0108 9391 0016 149d da39 0000 0000
                         5004 0000 1744 0000
17:09:06.106325 10.10.1.3.37778 > 10.10.1.8.22: S [tcp sum ok]
345889337:345889337(0) win 3072 <wscale 10,nop,mss 265,timestamp 1061109567
0,eol> (ttl 50, id 16305, len 60)
                         4500 003c 3fb1 0000 3206 32ed 0a0a 0103
                         0a0a 0108 9392 0016 149d da39 0000 0000
                         a002 0c00 2497 0000 0303 0a01 0204 0109
                         080a 3f3f 3f3f 0000 0000 0000
17:09:06.106795 10.10.1.8.22 > 10.10.1.3.37778: S [tcp sum ok]
4221982072:4221982072(0) ack 345889338 win 57344 <mss 1460,nop,wscale
0,nop,nop,timestamp 389004 1061109567> (DF) (ttl 64, id 3357, len 60)
                         4500 003c 0d1d 4000 4006 1781 0a0a 0108
                         0a0a 0103 0016 9392 fba6 5578 149d da3a
                         a012 e000 1329 0000 0204 05b4 0103 0300
                         0101 080a 0005 ef8c 3f3f 3f3f
17:09:06.106891 10.10.1.3.37778 > 10.10.1.8.22: R [tcp sum ok]
345889338:345889338(0) win 0 (DF) (ttl 64, id 19726, len 40)
                         4500 0028 4d0e 4000 4006 d7a3 0a0a 0103
                         0a0a 0108 9392 0016 149d da3a 0000 0000
                         5004 0000 1742 0000
17:09:06.520218 10.10.1.3.37779 > 10.10.1.8.22: S [tcp sum ok]
345889338:345889338(0) win 2048 <wscale 10,nop,mss 265,timestamp 1061109567
0,eol> (ttl 53, id 9665, len 60)
                         4500 003c 25c1 0000 3506 49dd 0a0a 0103
                         0a0a 0108 9393 0016 149d da3a 0000 0000
                         a002 0800 2895 0000 0303 0a01 0204 0109
                         080a 3f3f 3f3f 0000 0000 0000
17:09:06.520682 10.10.1.8.22 > 10.10.1.3.37779: S [tcp sum ok]
2973548148:2973548148(0) ack 345889339 win 57344 <mss 1460,nop,wscale
0,nop,nop,timestamp 389045 1061109567> (DF) (ttl 64, id 3358, len 60)
                         4500 003c 0d1e 4000 4006 1780 0a0a 0108
                         0a0a 0103 0016 9393 b13c be74 149d da3b
                         a012 e000 f46b 0000 0204 05b4 0103 0300
                         0101 080a 0005 efb5 3f3f 3f3f
17:09:06.520780 10.10.1.3.37779 > 10.10.1.8.22: R [tcp sum ok]
345889339:345889339(0) win 0 (DF) (ttl 64, id 27369, len 40)
                         4500 0028 6ae9 4000 4006 b9c8 0a0a 0103
                         0a0a 0108 9393 0016 149d da3b 0000 0000
                         5004 0000 1740 0000
```

```
17:09:06.942130 10.10.1.3.37780 > 10.10.1.8.22: S [tcp sum ok]
345889339:345889339(0) win 2048 <wscale 10,nop,mss 265,timestamp 1061109567
0,eol> (ttl 41, id 44087, len 60)
                     4500 003c ac37 0000 2906 cf66 0a0a 0103
                     0a0a 0108 9394 0016 149d da3b 0000 0000
                     a002 0800 2893 0000 0303 0a01 0204 0109
                     080a 3f3f 3f3f 0000 0000 0000
17:09:06.942585 10.10.1.8.22 > 10.10.1.3.37780: S [tcp sum ok]
2330511456:2330511456(0) ack 345889340 win 57344 <mss 1460,nop,wscale
0,nop,nop,timestamp 389087 1061109567> (DF) (ttl 64, id 3359, len 60)
                     4500 003c 0d1f 4000 4006 177f 0a0a 0108
                     0a0a 0103 0016 9394 8ae8 c860 149d da3c
                     a012 e000 10a8 0000 0204 05b4 0103 0300
                     0101 080a 0005 efdf 3f3f 3f3f
17:09:06.942672 10.10.1.3.37780 > 10.10.1.8.22: R [tcp sum ok]
345889340:345889340(0) win 0 (DF) (ttl 64, id 17270, len 40)
                     4500 0028 4376 4000 4006 e13b 0a0a 0103
                     0a0a 0108 9394 0016 149d da3c 0000 0000
                     5004 0000 173e 0000
17:09:07.067128 10.10.1.3.37781 > 10.10.1.8.22: S [tcp sum ok]
345889340:345889340(0) win 4096 <wscale 10,nop,mss 265,timestamp 1061109567
0,eol> (ttl 43, id 60721, len 60)
                     4500 003c ed31 0000 2b06 8c6c 0a0a 0103
                     0a0a 0108 9395 0016 149d da3c 0000 0000
                     a002 1000 2091 0000 0303 0a01 0204 0109
                     080a 3f3f 3f3f 0000 0000 0000
17:09:07.067616 10.10.1.8.22 > 10.10.1.3.37781: S [tcp sum ok]
2371689692:2371689692(0) ack 345889341 win 57344 <mss 1460,nop,wscale
0,nop,nop,timestamp 389100 1061109567> (DF) (ttl 64, id 3360, len 60)
                     4500 003c 0d20 4000 4006 177e 0a0a 0108
                     0a0a 0103 0016 9395 8d5d 1cdc 149d da3d
                     a012 e000 b9a8 0000 0204 05b4 0103 0300
                     0101 080a 0005 efec 3f3f 3f3f
17:09:07.067715 10.10.1.3.37781 > 10.10.1.8.22: R [tcp sum ok]
345889341:345889341(0) win 0 (DF) (ttl 64, id 17039, len 40)
                     4500 0028 428f 4000 4006 e222 0a0a 0103
                     0a0a 0108 9395 0016 149d da3d 0000 0000
                     5004 0000 173c 0000
```

The results are then examined and compared with the OS signature database. If `Nmap` returns a guess like it has in our example it means that an exact signature match was found.

```
Remote operating system guess: FreeBSD 4.6.2-RELEASE - 4.7-STABLE
```

In our example, FreeBSD 4.6.2-RELEASE to 4.7-STABLE was found to be the best match. From FreeBSD 4.6.2 – 4.7 there are few differences. This is due to the fact that they share a very similar TCP/IP stack.

2.5 Defenses

There are some defensive measures to protect against the OS detection method used by `Nmap`. Among the defenses are:

- In a normal network environment systems should sit behind some type of firewall[4]. Machines that are viewable from the Internet should only keep the needed ports open while the rest of the ports should be filtered by the firewall. This is a good defense since `Nmap` works best when it finds at least one open TCP port, one closed TCP port, and one closed UDP port. If all the needed ports aren't found then Nmap's accuracy drops off.
- Users can change characteristics of the machines TCP/IP stack. This can also be accomplished via kernel patches[5].
- Network intrusion detection systems[6] (IDS) can be used, if configured correctly, to detect the OS detection method used by `Nmap` because of the utilization of malformed packets.

[4] Check Point makes a firewall product named FireWall-1 (http://www.checkpoint.com/).
[5] An example for Linux is IP Personality (http://ippersonality.sourceforge.net/). An example for BSD is the Blackhole option.
[6] Snort is an example of a network intrusion detection system (http://www.snort.org/).

3.0 RINGv2 Tool

RINGv2 is a remote OS detection tool. It is designed to determine the OS running on the remote machine with minimal target disturbance. The RINGv2 OS detection methods have been included as a patched Nmap version 3.00 now called `Nmap-ringv2`.

3.1 SYN_RCVD Method

This is one of the OS detection methods that `Nmap-ringv2` can perform. The SYN_RCVD method works by measuring the retransmission timeout (RTO) values of the SYN_ACK responses from the target machine.

3.1.1 Technique

Before `Nmap-ringv2` runs this OS detection method it runs a port scan against the target machine. It performs a port scan so it can find an open port on the target machine. `Nmap-ringv2` needs at least one open TCP port to work. The SYN_RCVD method works by measuring the retransmission timeout values of the SYN_ACK responses from the target machine. See Diagram 1 for a general idea of the way it works.

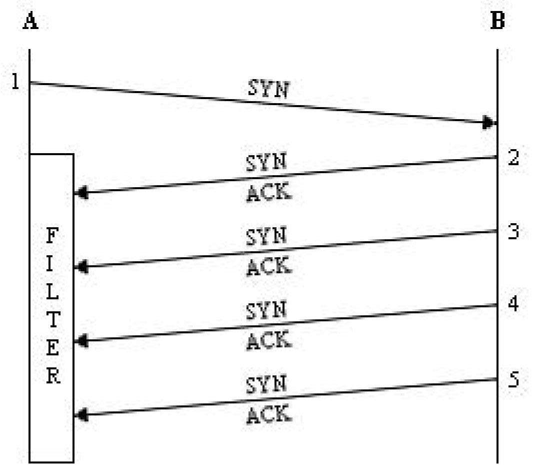

Diagram 1: SYN_RCVD method

First, a firewall rule is set up locally to block incoming TCP packets with the SYN and ACK flags enabled from the target machine specified by the user. Next, a TCP packet with the SYN flag enabled is sent to an open TCP port to attempt a connection establishment. The target machine tries to send an acknowledgement packet but the firewall rule that was set up blocks it. `Nmap-ringv2` keeps track of the delay between each retransmission.

After `Nmap-ringv2` has received the retransmission timeout responses it builds an OS fingerprint signature, then tries to find a match in the fingerprint database. If the signature is found in the database, `Nmap-ringv2` will list it as the Remote OS guess. If the signature isn't found in the database, `Nmap-ringv2` will display a message saying "No exact matches for host (test conditions non-ideal).", and shows the TCP/IP fingerprint information found.

3.1.2 OS Signature

Based on the results (information in the packets sent back to us) of the tests that `Nmap-ringv2` performs, an OS signature is built. I will now show an example signature from the database, and then explain what it means. Since we are dealing with the SYN_RCVD method I will only explain the lines that apply.

```
FingerPrint NetBSD 1.6 i386
Ring_Syn(nbPkt=4%Time=60%p=2900000%p=6000000%p=12000000%p=24000000)
Ring_LastAck(nbPkt=12%Time=480%Connect=4040%p=960000%p=2000000%p=4000000%p=8000
000%p=16000000%p=32000000%p=64000000%p=64000000%p=64000000%p=64000000%p=6400000
0%p=64000000)
```

I will go through the signature line by line.

```
FingerPrint NetBSD 1.6 i386 RINGv2
```

This signature covers NetBSD 1.6 for the i386 platform.

```
Ring_Syn(nbPkt=4%Time=60%p=2900000%p=6000000%p=12000000%p=24000000)
```

This line represents the results of the SYN_RCVD test. Remember that this test documents the retransmission timeout values of the SYN_ACK responses. NbPkt=4 means that the number of packets sent was 4. Time=60 means that `Nmap-ringv2` timeout value is 60 seconds. p=2900000 means that the SYN_ACK was resent 2.90~ seconds after the first SYN_ACK, then 6.00~, 12.00~, and finally 24.00~ seconds after that.

3.1.3 Advantages

The `Nmap-ringv2` tool offers many advantages over other remote active OS fingerprinting tools. Here are some reasons users might want to utilize this tool for this OS detection method:

- "Half-open" scanning is supported like in `Nmap`.
- The main advantage is that it only requires one open TCP port. If the target system is behind some sort of filtering device, normally there will only be one open port, with the rest being filtered.
- This technique uses a standard TCP packet. It tries to make a standard connection to the target machine; the connection establishment never gets completed. This type of method won't disturb the target machine.
- Strict signature matching is utilized.
- It has built-in learning functions.

3.1.4 Usage Example

The sample run was produced utilizing a Linux Kernel 2.4.18 machine running `Nmap-ringv2` targeting a Microsoft Windows 2000 Server machine on the same local LAN. It should be noted that the `Tcpdump` log timestamps are slightly off of what `Nmap-ringv2` records.

The following is a sample run that `Nmap-ringv2` produced using the SYN_RCVD method:

```
# nmap-ringv2 –sS –-ring –-ring_timeout 30 –-ring_method s –v –p 445 10.10.1.7

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
```

```
Host pluto.packetwatch.net (10.10.1.7) appears to be up ... good.
Initiating SYN Stealth Scan against pluto.packetwatch.net (10.10.1.7)
Adding open port 445/tcp
The SYN Stealth Scan took 0 seconds to scan 1 ports.
Warning:  OS detection will be MUCH less reliable because we did not find at
least 1 open and 1 closed TCP port
Try Time: 2847773 6008463 Interesting ports on pluto.packetwatch.net
(10.10.1.7):
Port       State       Service
445/tcp    open        microsoft-ds
Remote operating system guess: WinXP Home Base/SP1a, WinXP Pro Base/SP1a RINGv2

Nmap run completed -- 1 IP address (1 host up) scanned in 33 seconds
```

The `Tcpdump` output for the SYN packet that we sent to establish the connection:

```
17:15:46.862626 10.10.1.5.26423 > 10.10.1.7.445: S 25644:25644(0) win 1024 <mss
1460,sackOK,timestamp 79877292[|tcp]> (ttl 64, id 21012, len 60)
                     4500 003c 5214 0000 4006 1289 0a0a 0105
                     0a0a 0107 6737 01bd 0000 642c 0000 0000
                     a002 0400 8758 0000 0204 05b4 0402 080a
                     04c2 d4ac 0000
```

The `Tcpdump` output for the first SYN_ACK response:

```
17:15:46.863370 10.10.1.7.445 > 10.10.1.5.26423: S 3497663947:3497663947(0) ack
25645 win 17520 <mss 1460,nop,wscale 0,nop,nop,timestamp[|tcp]> (DF) (ttl 128,
id 76, len 64)
                     4500 0040 004c 4000 8006 e44c 0a0a 0107
                     0a0a 0105 01bd 6737 d07a 1dcb 0000 642d
                     b012 4470 1ffa 0000 0204 05b4 0103 0300
                     0101 080a 0000
```

The `Tcpdump` output for the next SYN_ACK response:

```
17:15:49.849402 10.10.1.7.445 > 10.10.1.5.26423: S 3497663947:3497663947(0) ack
25645 win 17520 <mss 1460,nop,wscale 0,nop,nop,timestamp[|tcp]> (DF) (ttl 128,
id 77, len 64)
                     4500 0040 004d 4000 8006 e44b 0a0a 0107
                     0a0a 0105 01bd 6737 d07a 1dcb 0000 642d
                     b012 4470 1ffa 0000 0204 05b4 0103 0300
                     0101 080a 0000
```

The `Tcpdump` output for the final SYN_ACK response:

```
17:15:55.857912 10.10.1.7.445 > 10.10.1.5.26423: S 3497663947:3497663947(0) ack
25645 win 17520 <mss 1460,nop,wscale 0,nop,nop,timestamp[|tcp]> (DF) (ttl 128,
id 78, len 64)
                     4500 0040 004e 4000 8006 e44a 0a0a 0107
                     0a0a 0105 01bd 6737 d07a 1dcb 0000 642d
                     b012 4470 1ffa 0000 0204 05b4 0103 0300
                     0101 080a 0000
```

The results are then examined and compared with the OS signature database. If `Nmap-ringv2` returns a guess like it has in our example it means that an exact signature match was found.

```
Remote operating system guess: WinXP Home Base/SP1a, WinXP Pro Base/SP1a RINGv2
```

In our example, Microsoft Windows XP Home Edition base through SP1a, and Microsoft Windows XP Professional base through SP1a were found to be the best match. From Microsoft Windows 2000 base through SP1a there are few differences. This is due to the fact that they share a very similar TCP/IP stack.

3.1.5 Defenses

There are some defensive measures to protect against this OS detection method used by `Nmap-ringv2`. Among the defenses are:

- First, machines should be behind some sort of firewall or filtering device. Only needed ports should be left open, the rest should be filtered. This means that there should only be a few ports left open. Closing all TCP ports would protect against this method.
- A packet mangler like netfilter/iptables[7] could be used to mangle packets to enforce misleading RTO values on all outgoing packets. This would cause pen-testers or black-hats to think that were fingerprinting a different OS.
- A proxy or firewall could be used to hide machines behind. You would need to use proxy or firewall that supported SYN Relay or SYN Gateway techniques[8]. Check Point has had support for these techniques in their FireWall-1 product. A chart of FireWall-1's support for these techniques can be found at http://www.fw-1.de/aerasec/ng/syndefender-01.html. Some diagrams of how they work are included.

[7] More information on netfilter/iptables can be found at http://www.netfilter.org/.
[8] A guide of how these techniques work can be found at http://www.phoneboy.com/fom-serve/cache/153.html.

3.2 LAST_ACK Method

This is one of the OS detection methods that `Nmap-ringv2` can perform. The LAST_ACK method works by measuring the retransmission timeout (RTO) values of the FIN_ACK responses from the target machine.

3.2.1 Technique

Before `Nmap-ringv2` runs this OS detection method it runs a port scan against the target machine. It performs a port scan so it can find an open port on the target machine. `Nmap-ringv2` needs at least one open TCP port to work. The LAST_ACK method works by measuring the retransmission timeout values of the FIN_ACK responses from the target machine. See Diagram 2 for a general idea of the way it works.

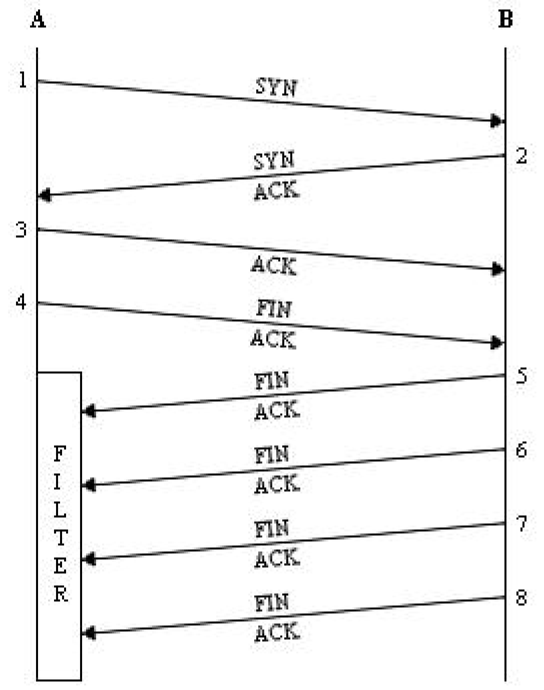

Diagram 2: LAST_ACK method

First, a firewall rule is set up locally to block incoming TCP packets with the FIN and ACK flags enabled from the target machine specified by the user. Next, a TCP packet with the SYN flag enabled is sent to an open TCP port to attempt a connection establishment. The target machine responds with an acknowledgement packet. Next, a TCP packet with the ACK flag enabled is sent to the complete the connection. Now we have a connection established between the two hosts. Next, a TCP packet with the FIN and ACK flags enabled is sent to the target machine to attempt a connection termination. The target machine tries to send an acknowledgement packet but the firewall rule that was set up blocks it. `Nmap-ringv2` keeps track of the delay between each retransmission.

After `Nmap-ringv2` has received the retransmission timeout responses it builds an OS fingerprint signature, then tries to find a match in the fingerprint database. If the signature is found in the database, `Nmap-ringv2` will list it as the Remote OS guess. If the signature isn't found in the database, `Nmap-ringv2` will display a message saying "No exact matches for host (test conditions non-ideal).", and shows the TCP/IP fingerprint information found.

3.2.2 OS Signature

Based on the results (information in the packets sent back to us) of the tests that `Nmap-ringv2` performs, an OS signature is built. I will now show an example signature from the database, and then explain what it means. Since we are dealing with the LAST_ACK method I will only explain the lines that apply.

```
FingerPrint NetBSD 1.6 i386
Ring_Syn(nbPkt=4%Time=60%p=2900000%p=6000000%p=12000000%p=24000000)
Ring_LastAck(nbPkt=12%Time=480%Connect=4040%p=960000%p=2000000%p=4000000%p=8000
000%p=16000000%p=32000000%p=64000000%p=64000000%p=64000000%p=64000000%p=6400000
0%p=64000000)
```

I will go through the signature line by line.

```
FingerPrint NetBSD 1.6 i386 RINGv2
```

This signature covers NetBSD 1.6 for the i386 platform.

```
Ring_LastAck(nbPkt=12%Time=480%Connect=4040%p=960000%p=2000000%p=4000000%p=8000
000%p=16000000%p=32000000%p=64000000%p=64000000%p=64000000%p=64000000%p=6400000
0%p=64000000)
```

This line represents the results of the LAST_ACK test. Remember that this test documents the retransmission times of the FIN_ACK responses. NbPkt=12 means that the number of packets sent was 12. Time=480 means that `Nmap-ringv2` timeout value is 480 seconds. Connect=4040 means that the connect time was 4040. p=960000 means that the FIN_ACK was resent 4.00~ seconds after the first FIN_ACK, then 8.00~, 16.00~, 32.00~, 64.00~, 64.00~, 64.00~, 64.00~, 64.00, and finally 64.00~ seconds after that.

3.2.3 Advantages

The `Nmap-ringv2` tool offers many advantages over other OS fingerprinting tools. Here are some reasons users might want to utilize this tool for this OS detection method:

- "Half-open" scanning is supported like in `Nmap`.
- The main advantage is that it only requires one open TCP port. If the target system is behind some sort of filtering device, normally there will only be one open port, with the rest being filtered.
- This technique uses a standard TCP packet. It establishes a standard connection, then tries to terminate it, but never gets completed. This type of method won't disturb the target machine.
- Strict signature matching is utilized.
- It has built-in learning functions.

3.2.4 Usage Example

The sample run was produced utilizing a Linux Kernel 2.4.18 machine running `Nmap-ringv2` targeting a Microsoft Windows 2000 Server machine on the same local LAN. It should be noted that the `Tcpdump` log timestamps are slightly off of what `Nmap-ringv2` records.

The following is a sample run that `Nmap-ringv2` produced using the LAST_ACK method:

```
# nmap-ringv2 –sS ––ring ––ring_timeout 120 ––ring_method l –v –p 445 10.10.1.7

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Host pluto.packetwatch.net (10.10.1.7) appears to be up ... good.
Initiating SYN Stealth Scan against pluto.packetwatch.net (10.10.1.7)
Adding open port 445/tcp
The SYN Stealth Scan took 0 seconds to scan 1 ports.
Warning:  OS detection will be MUCH less reliable because we did not find at
least 1 open and 1 closed TCP port
Try Time: 2922934 6008497 12017073 24034151 48068190 Interesting ports on
pluto.packetwatch.net (10.10.1.7):
Port       State       Service
445/tcp    open        microsoft-ds
Remote operating system guess: WinXP Home Base/SP1a, WinXP Pro Base/SP1a RINGv2

Nmap run completed -- 1 IP address (1 host up) scanned in 124 seconds
```

The `Tcpdump` output for the FIN packet that we sent to terminate the connection:

```
17:18:18.210573 10.10.1.5.1050 > 10.10.1.7.445: F [tcp sum ok]
1320277821:1320277821(0) ack 3535179022 win 5840 <nop,nop,timestamp 641619 0>
(DF) (ttl 64, id 7816, len 52)
                     4500 0034 1e88 4000 4006 061d 0a0a 0105
                     0a0a 0107 041a 01bd 4eb1 d73d d2b6 8d0e
                     8011 16d0 f3e4 0000 0101 080a 0009 ca53
                     0000 0000
```

The `Tcpdump` output for the first FIN-ACK response:

```
17:18:18.211232 10.10.1.7.445 > 10.10.1.5.1050: F [tcp sum ok]
3535179022:3535179022(0) ack 1320277822 win 17520 <nop,nop,timestamp 5623
641619> (DF) (ttl 128, id 84, len 52)
                     4500 0034 0054 4000 8006 e450 0a0a 0107
                     0a0a 0105 01bd 041a d2b6 8d0e 4eb1 d73e
                     8011 4470 b04c 0000 0101 080a 0000 15f7
                     0009 ca53
```

The `Tcpdump` output for the next FIN-ACK response:

```
17:18:21.164037 10.10.1.7.445 > 10.10.1.5.1050: F [tcp sum ok]
3535179022:3535179022(0) ack 1320277822 win 17520 <nop,nop,timestamp 5653
641619> (DF) (ttl 128, id 85, len 52)
                     4500 0034 0055 4000 8006 e44f 0a0a 0107
                     0a0a 0105 01bd 041a d2b6 8d0e 4eb1 d73e
                     8011 4470 b02e 0000 0101 080a 0000 1615
                     0009 ca53
```

The `Tcpdump` output for the next FIN-ACK response:

```
17:18:27.172561 10.10.1.7.445 > 10.10.1.5.1050: F [tcp sum ok]
3535179022:3535179022(0) ack 1320277822 win 17520 <nop,nop,timestamp 5713
641619> (DF) (ttl 128, id 86, len 52)
                     4500 0034 0056 4000 8006 e44e 0a0a 0107
                     0a0a 0105 01bd 041a d2b6 8d0e 4eb1 d73e
                     8011 4470 aff2 0000 0101 080a 0000 1651
                     0009 ca53
```

The `Tcpdump` output for the next FIN-ACK response:

```
17:18:39.189615 10.10.1.7.445 > 10.10.1.5.1050: F [tcp sum ok]
3535179022:3535179022(0) ack 1320277822 win 17520 <nop,nop,timestamp 5833
641619> (DF) (ttl 128, id 87, len 52)
                     4500 0034 0057 4000 8006 e44d 0a0a 0107
                     0a0a 0105 01bd 041a d2b6 8d0e 4eb1 d73e
                     8011 4470 af7a 0000 0101 080a 0000 16c9
                     0009 ca53
```

The `Tcpdump` output for the next FIN-ACK response:

```
17:19:03.223727 10.10.1.7.445 > 10.10.1.5.1050: F [tcp sum ok]
3535179022:3535179022(0) ack 1320277822 win 17520 <nop,nop,timestamp 6073
641619> (DF) (ttl 128, id 88, len 52)
                     4500 0034 0058 4000 8006 e44c 0a0a 0107
                     0a0a 0105 01bd 041a d2b6 8d0e 4eb1 d73e
                     8011 4470 ae8a 0000 0101 080a 0000 17b9
                     0009 ca53
```

The `Tcpdump` output for the next FIN-ACK response:

```
17:19:51.291956 10.10.1.7.445 > 10.10.1.5.1050: F [tcp sum ok]
3535179022:3535179022(0) ack 1320277822 win 17520 <nop,nop,timestamp 6554
641619> (DF) (ttl 128, id 89, len 52)
                     4500 0034 0059 4000 8006 e44b 0a0a 0107
                     0a0a 0105 01bd 041a d2b6 8d0e 4eb1 d73e
                     8011 4470 aca9 0000 0101 080a 0000 199a
                     0009 ca53
```

The results are then examined and compared with the OS signature database. If `Nmap-ringv2` returns a guess like it has in our example it means that an exact signature match was found.

```
Remote operating system guess: WinXP Home Base/SP1a, WinXP Pro Base/SP1a
```

In our example, Microsoft Windows XP Home Edition base through SP1a, and Microsoft Windows XP Professional base through SP1a were found to be the best match. From Microsoft Windows 2000 base through SP1a there are few differences. This is due to the fact that they share a very similar TCP/IP stack.

3.2.5 Defenses

There are some defensive measures to protect against this OS detection method used by `Nmap-ringv2`. Among the defenses are:

- First, machines should be behind some sort of firewall or filtering device. Only needed ports should be left open, the rest should be filtered. This means that there should only be a few ports left open. Closing all TCP ports would protect against this method.
- A packet mangler can be used to modify outgoing RTO values.
- A proxy or firewall that supports SYN Relay or SYN Gateway techniques can be used to hide machines. Check Point's FireWall-1 product supports these techniques. More information can be found at the links mentioned in section 3.1.5.

3.3 FIN_WAIT_1 Method

This is the last of the OS detection methods that `Nmap-ringv2` can perform. The FIN_WAIT_1 method works by measuring the retransmission timeout (RTO) values of the FIN_ACK responses from the target machine after a normal exchange of data. This method is experimental therefore I've only included information on how the technique works and the advantages it offers.

3.3.1 Technique

Before `Nmap-ringv2` runs this OS detection method it runs a port scan against the target machine. It performs a port scan so it can find an open port on the target machine. `Nmap-ringv2` needs at least one open TCP port to work. The FIN_WAIT_1 method works by measuring the retransmission timeout values of the FIN_ACK responses from the target machine after a normal exchange of data. See Diagram 3 for a general idea of the way it works.

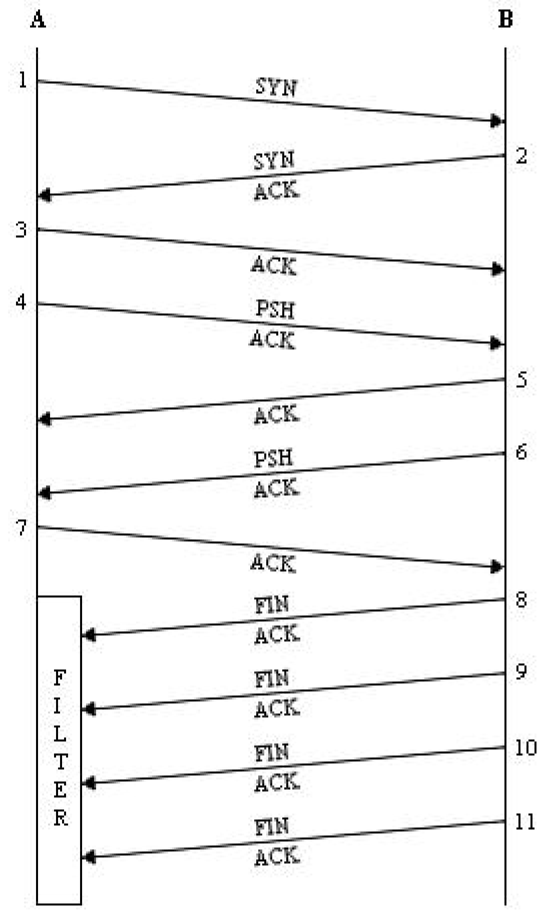

Diagram 3: FIN_WAIT_1 method

First, a firewall rule is set up locally to block incoming TCP packets with the FIN and ACK flags enabled from the target machine specified by the user. Next, a TCP packet with the SYN flag enabled is sent to an open TCP port to attempt a connection establishment. The target machine responds with an acknowledgement packet. Next, a TCP packet with the ACK flag enabled is sent to the complete the connection. Now we have a connection established between the two hosts. Next, a

TCP packet with the PSH and ACK flags enabled is sent to the target machine to start a normal exchange of data. The target machine sends an acknowledgement packet back. Next, the target machine sends a TCP packet with the PSH and ACK flags enabled. Now, we reply with an acknowledgement packet back. This concludes the exchange of data. The target machine then sends a TCP packet with the FIN and ACK flags enabled to attempt a connection termination but the firewall rule that was set up blocks it. `Nmap-ringv2` keeps track of the delay between each retransmission.

After `Nmap-ringv2` has received the retransmission timeout responses it builds an OS fingerprint signature, then tries to find a match in the fingerprint database. If the signature is found in the database, `Nmap-ringv2` will list it as the Remote OS guess. If the signature isn't found in the database, `Nmap-ringv2` will display a message saying "No exact matches for host (test conditions non-ideal).", and shows the TCP/IP fingerprint information found.

3.3.2 Advantages

The `Nmap-ringv2` tool offers many advantages over other OS fingerprinting tools. Here are some reasons users might want to utilize this tool for this OS detection method:

- "Half-open" scanning is supported like in `Nmap`.
- The main advantage is that it only requires one open TCP port. If the target system is behind some sort of filtering device, normally there will only be one open port, with the rest being filtered.
- It's able to bypass SYN Relay and SYN Gateway techniques to perform OS fingerprinting tests on the real target.
- Strict signature matching is utilized.
- It has built-in learning functions.

4.0 Xprobe2 Tool

`Xprobe2` is a remote active OS fingerprinting tool. It is designed with a different approach to OS fingerprinting. The `Xprobe2` OS detection method identifies the type of the remote OS with a matrix based fingerprinting approach. This approach is also known as ''fuzzy'' matching. I'll be using `Xprobe2` version 0.1 since it was the current version at the time of this writing.

4.1 Technique

Unlike the other tools, `Xprobe2` doesn't run a port scan against the target machine. `Xprobe2` needs at least one closed UDP port to work. `Xprobe2` heavily uses the results found in the "ICMP Usage in Scanning" research project[9] by Ofir Arkin. It relies primarily on the use of the ICMP protocol.

`Xprobe2` is modular in design so it has the capability to accept new modules, or other fingerprinting tests. Another idea to note is that `Xprobe2` comes with an API so that users can write their own modules.

`Xprobe2` works by running different modules or tests against the target machine. There are two types of modules. There are the reachability modules, and the fingerprinting modules. The first two modules run are reachability tests. They try to determine if the target machine is alive. The rest of the modules are fingerprinting tests. They try to determine the OS the target is running.

The first reachability module is the ICMP echo (ping) test. In this test an ICMP packet with an ICMP echo request message is sent. The goal is to elicit an ICMP packet with an ICMP echo reply message back from the target machine.

The second reachability module is the TTL distance test. In this test a TCP packet with the SYN flag enabled is sent to a TCP port. The goal is to elicit, a TCP packet with the SYN and ACK flags enabled meaning the TCP port is opened or a TCP packet with the RST flag enabled meaning the TCP port is closed back from the target machine. If no response is received another TCP packet with the same options is sent to a different TCP port, with the same goal.

The first fingerprinting module, known as Module A, is the ICMP echo request test. In this test an ICMP packet with an ICMP echo request message is sent.

The second module, known as Module B, is the ICMP timestamp test. In this test an ICMP packet with an ICMP timestamp request message is sent.

The third module, known as Module C, is the ICMP address mask test. In this test an ICMP packet with an ICMP address mask request message is sent.

The fourth module, known as Module D, is the ICMP information request test. In this test an ICMP packet with an ICMP information request message is sent.

The last module that `Xprobe2` performs, known as Module E, is the ICMP port unreachable test. In this test a UDP packet acting as a DNS query result is sent. The goal is to elicit an ICMP packet with an ICMP port unreachable message back from the target machine.

---

[9] Information about the project can be found at http://www.sys-security.com/html/projects/icmp.html.

After Xprobe2 has received all of the responses from the modules, the scores are calculated and compared to the fingerprint database. Next, Xprobe2 returns with a primary guess which is the most probable match, according to the sum of all the scores. It also returns other possibilities.

4.2 OS Signature

Based on the results (information in the packets sent back to us) of the tests that Xprobe2 perfoms, an OS signature is built. I will now show an example signature from the database, and then explain what it means.

```
fingerprint {
	OS_ID = "OpenBSD 3.2"
	#Entry inserted to the database by: Ofir Arkin (ofir@sys-security.com)
	#Entry contributed by: Ofir Arkin (ofir@sys-security.com)
	#Date: 6 April 2003
	#Modified: 6 April 2003

	#Module A
	icmp_echo_code = !0
	icmp_echo_ip_id = !0
	icmp_echo_tos_bits = !0
	icmp_echo_df_bit = 1
	icmp_echo_reply_ttl = <255

	#Module B
	icmp_timestamp_reply = y
	icmp_timestamp_reply_ttl = <255
	icmp_timestamp_reply_ip_id = !0

	#Module C
	icmp_addrmask_reply = n
	icmp_addrmask_reply_ttl = <255
	icmp_addrmask_reply_ip_id = !0

	#Module D
	icmp_info_reply = n
	icmp_info_reply_ttl = <255
	icmp_info_reply_ip_id = !0

	#Module E
	#IP_Header_of_the_UDP_Port_Unreachable_error_message
	icmp_unreach_echoed_dtsize = 8
	icmp_unreach_reply_ttl = <255
	icmp_unreach_precedence_bits = 0
	icmp_unreach_df_bit = 0
	icmp_unreach_ip_id = !0

	#Original_data_echoed_with_the_UDP_Port_Unreachable_error_message
	icmp_unreach_echoed_udp_cksum = OK
	icmp_unreach_echoed_ip_cksum  = BAD
	icmp_unreach_echoed_ip_id = OK
	icmp_unreach_echoed_total_len = <20
	icmp_unreach_echoed_3bit_flags = OK
}
```

I will go through the signature line by line.

```
OS_ID = "OpenBSD 3.2"
```

This signature covers OpenBSD 3.2.

Figure 2 is included to show the IP header which will be referred to in the results of some of the tests.

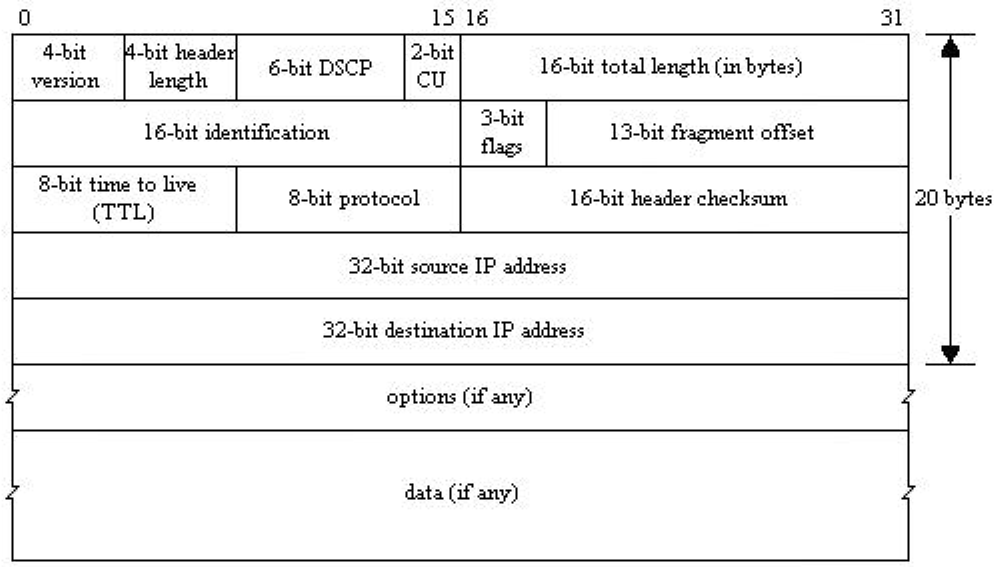

Figure 2: IP header

```
#Module A
icmp_echo_code = !0
icmp_echo_ip_id = !0
icmp_echo_tos_bits = !0
icmp_echo_df_bit = 1
icmp_echo_reply_ttl = <255
```

These lines represent the results from the ICMP echo request test. icmp_echo_code = !0 means that the code in the ICMP header does not equal 0. icmp_echo_ip_id = !0 means that the identification number in the IP header does not equal 0. icmp_echo_tos_bits = !0 means that the TOS or Differentiated Services Field in the IP header does not equal 0. icmp_echo_df_bit = 1 means that the Don't fragment flag in the IP header was enabled so is therefore equal to 1. icmp_echo_reply_ttl = <255 means that the time to live in the IP header is less than or equal to 255.

```
#Module B
icmp_timestamp_reply = y
icmp_timestamp_reply_ttl = <255
icmp_timestamp_reply_ip_id = !0
```

These lines represent the results from the ICMP timestamp test. icmp_timestamp_reply = y means that we received an ICMP timestamp reply message from the target machine. icmp_timestamp_reply_ttl = <255 means that time to live in the IP header is less than or equal to 255. icmp_timestamp_reply_ip_id = !0 means that the identification number in the IP header does not equal 0.

```
#Module C
icmp_addrmask_reply = n
icmp_addrmask_reply_ttl = <255
```

```
icmp_addrmask_reply_ip_id = !0
```

These lines represent the results from the ICMP address mask test. icmp_addrmask_reply = n means that we didn't receive an ICMP address mask reply message from the target machine. The rest of the variables are sent back to their default values.

```
#Module D
icmp_info_reply = n
icmp_info_reply_ttl = <255
icmp_info_reply_ip_id = !0
```

These lines represent the results from the ICMP information request test. icmp_info_reply = n means that we didn't receive an ICMP information request reply message from the target machine. The rest of the variables are sent back to their default values.

Figure 3 is included to show the UDP header which will be referred to in the results of Module E.

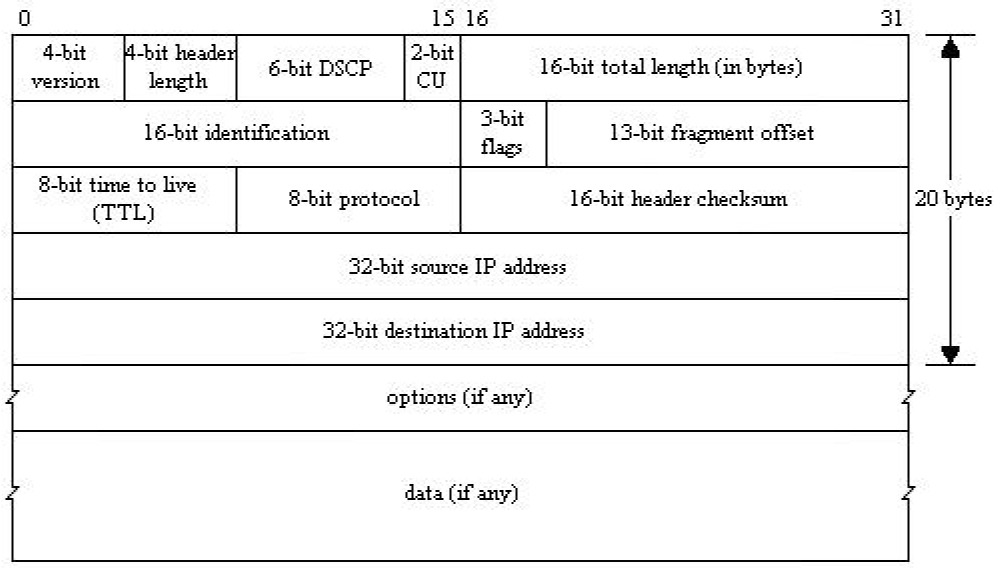

Figure 3: UDP header

```
#Module E
#IP_Header_of_the_UDP_Port_Unreachable_error_message
icmp_unreach_echoed_dtsize = 8
icmp_unreach_reply_ttl = <255
icmp_unreach_precedence_bits = 0
icmp_unreach_df_bit = 0
icmp_unreach_ip_id = !0

#Original_data_echoed_with_the_UDP_Port_Unreachable_error_message
icmp_unreach_echoed_udp_cksum = OK
icmp_unreach_echoed_ip_cksum  = BAD
icmp_unreach_echoed_ip_id = OK
icmp_unreach_echoed_total_len = <20
icmp_unreach_echoed_3bit_flags = OK
```

These lines represent the results from the ICMP port unreachable test. The first set of variables are from the IP header of the UDP port unreachable error message. icmp_unreach_echoed_dtsize = 8 means that the amount of data echoed back to us with the ICMP port unreachable error message from the original UDP packet sent to the target machine is equal to 8 bytes. icmp_unreach_reply_ttl = <255 means that the time to live in the IP header is less than or equal to 255. icmp_unreach_precedence_bits = 0 means that the precedence flag in the IP header isn't enabled so is therefore equal to 0. icmp_unreach_df_bit = 0 means that the Don't fragment flag in the IP header wasn't enabled and is therefore equal to 0. icmp_unreach_ip_id = !0 means that identification number in the IP header does not equal 0.

The second set of variables are from the original data echoed back with the UDP port unreachable error message. icmp_unreach_echoed_udp_cksum = OK means that the checksum in the UDP header echoed back to us with the ICMP port unreachable error message from the original UDP packet is correct. icmp_unreach_echoed_ip_cksum = BAD means that the checksum in the IP header echoed back to us with the ICMP port unreachable error message from the original UDP packet is incorrect. icmp_unreach_echoed_ip_id = OK means that the identification number in the IP header echoed back to us with the ICMP port unreachable error message from the original UDP packet is correct. icmp_unreach_echoed_total_len = <20 means that the total length in the IP header echoed back to us with the ICMP port unreachable error message from the original UDP packet is less than or equal to 20. icmp_unreach_echoed_3bit_flags = OK means that the 3-bit flags in the IP header echoed back to us with the ICMP port unreachable error message from the original UDP packet is correct.

4.3 Advantages

The `Xprobe2` tool offers many advantages over other OS fingerprinting tools. Here are some reasons users might want to utilize this tool for its OS detection method:

- It is very quick because not many packets are sent to the remote machine.
- There is little target host disturbance.
- There is a small but growing OS signature database.
- There is support for network devices like routers and switches.
- Matrix based fingerprint matching approach is utilized. This approach is also known as ''fuzzy'' matching.
- Xprobe2 comes with an API that allows users to write their own modules.

4.4 Usage Example

The sample run was produced utilizing an OpenBSD 3.2 machine running `Xprobe2` targeting a Linux Kernel 2.4.18 machine on the same local LAN.

The following is a sample run that `Xprobe2` produced:

```
# xprobe2 –v –p tcp:22:open 10.10.1.5

XProbe2 v.0.1 Copyright (c) 2002-2003 fygrave@tigerteam.net, ofir@sys-
security.com

[+] Target is 10.10.1.5
[+] Loading modules.
[+] Following modules are loaded:
       [x]ICMP echo (ping)
       [x]TTL distance
       [x]ICMP echo
       [x]ICMP Timestamp
       [x]ICMP Address
       [x]ICMP Info Request
       [x]ICMP port unreach
[+] 7 modules registered
[+] Initializing scan engine
[+] Running scan engine
[+] Host: 10.10.1.5 is up (Guess probability: 100%)
[+] Target: 10.10.1.5 is alive
[+] Primary guess:
[+] Host 10.10.1.5 Running OS: "Linux Kernel 2.4.5 and above" (Guess
probability: 95%)
[+] Other guesses:
[+] Host 10.10.1.5 Running OS: "Linux Kernel 2.2.x" (Guess probability: 95%)
[+] Host 10.10.1.5 Running OS: "NetBSD 1.6" (Guess probability: 87%)
[+] Host 10.10.1.5 Running OS: "Linux Kernel 2.4.0 - 2.4.4" (Guess probability:
83%)
[+] Host 10.10.1.5 Running OS: "SCO OpenServer Release 5" (Guess probability:
83%)
[+] Host 10.10.1.5 Running OS: "OpenBSD 2.5" (Guess probability: 83%)
[+] Host 10.10.1.5 Running OS: "NetBSD 1.5.0" (Guess probability: 83%)
[+] Host 10.10.1.5 Running OS: "NetBSD 1.5.1" (Guess probability: 83%)
[+] Host 10.10.1.5 Running OS: "NetBSD 1.5.2" (Guess probability: 83%)
[+] Host 10.10.1.5 Running OS: "Mac OS X 10.2.3" (Guess probability: 79%)
[+] Host 10.10.1.5 Running OS: "Mac OS X 10.2.4" (Guess probability: 79%)
[+] Host 10.10.1.5 Running OS: "Microsoft Windows ME" (Guess probability: 79%)
[+] Host 10.10.1.5 Running OS: "OpenBSD 2.6" (Guess probability: 79%)
[+] Host 10.10.1.5 Running OS: "OpenBSD 2.7" (Guess probability: 79%)
[+] Host 10.10.1.5 Running OS: "OpenBSD 2.8" (Guess probability: 79%)
[+] Host 10.10.1.5 Running OS: "OpenBSD 2.9" (Guess probability: 79%)
[+] Host 10.10.1.5 Running OS: "FreeBSD 5.0" (Guess probability: 79%)
[+] Host 10.10.1.5 Running OS: "FreeBSD 4.8" (Guess probability: 79%)
[+] Host 10.10.1.5 Running OS: "FreeBSD 4.7" (Guess probability: 79%)
[+] Host 10.10.1.5 Running OS: "FreeBSD 4.6" (Guess probability: 79%)
[+] Host 10.10.1.5 Running OS: "FreeBSD 4.4" (Guess probability: 79%)
[+] Host 10.10.1.5 Running OS: "FreeBSD 4.5" (Guess probability: 79%)
[+] Host 10.10.1.5 Running OS: "Cisco IOS 12.0" (Guess probability: 75%)
[+] Host 10.10.1.5 Running OS: "Cisco IOS 11.3" (Guess probability: 75%)
[+] Host 10.10.1.5 Running OS: "Mac OS X 10.1.5" (Guess probability: 75%)
[+] Host 10.10.1.5 Running OS: "Microsoft Windows 98/98SE" (Guess probability:
75%)
[+] Host 10.10.1.5 Running OS: "Microsoft Windows NT 4 Service Pack 4 and
Above" (Guess probability: 75%)
[+] Host 10.10.1.5 Running OS: "Microsoft Windows 2000/2000SP1/2000SP2/2000SP3"
(Guess probability: 75%)
[+] Host 10.10.1.5 Running OS: "Microsoft Windows XP Professional / XP
Professional SP1" (Guess probability: 75%)
[+] Host 10.10.1.5 Running OS: "Sun Solaris 5 (SunOS 2.5)" (Guess probability:
75%)
[+] Host 10.10.1.5 Running OS: "Sun Solaris 6 (SunOS 2.6)" (Guess probability:
75%)
[+] Host 10.10.1.5 Running OS: "Sun Solaris 7 (SunOS 2.7)" (Guess probability:
75%)
[+] Host 10.10.1.5 Running OS: "Sun Solaris 8 (SunOS 2.8)" (Guess probability:
75%)
[+] Host 10.10.1.5 Running OS: "Sun Solaris 9 (SunOS 2.9)" (Guess probability:
75%)
[+] Host 10.10.1.5 Running OS: "FreeBSD 4.3" (Guess probability: 75%)
[+] Host 10.10.1.5 Running OS: "FreeBSD 4.2" (Guess probability: 75%)
[+] Host 10.10.1.5 Running OS: "FreeBSD 4.1.1" (Guess probability: 75%)
[+] Host 10.10.1.5 Running OS: "OpenBSD 3.0" (Guess probability: 75%)
[+] Host 10.10.1.5 Running OS: "OpenBSD 3.1" (Guess probability: 75%)
[+] Host 10.10.1.5 Running OS: "OpenBSD 3.2" (Guess probability: 75%)
[+] Host 10.10.1.5 Running OS: "Digital UNIX 5.6" (Guess probability: 70%)
[+] Host 10.10.1.5 Running OS: "HPUX B.11.0 x" (Guess probability: 70%)
[+] Host 10.10.1.5 Running OS: "Microsoft Windows NT 4 Service Pack 3 and
Below" (Guess probability: 70%)
[+] Host 10.10.1.5 Running OS: "Cisco IOS 11.2" (Guess probability: 70%)
[+] Host 10.10.1.5 Running OS: "Cisco IOS 11.1" (Guess probability: 70%)
[+] Host 10.10.1.5 Running OS: "Mac OS 9.2.x" (Guess probability: 70%)
[+] Host 10.10.1.5 Running OS: "FreeBSD 3.1" (Guess probability: 62%)
[+] Host 10.10.1.5 Running OS: "FreeBSD 2.2.8" (Guess probability: 62%)
[+] Host 10.10.1.5 Running OS: "FreeBSD 2.2.7" (Guess probability: 62%)
[+] Host 10.10.1.5 Running OS: "FreeBSD 3.3" (Guess probability: 62%)
[+] Host 10.10.1.5 Running OS: "FreeBSD 3.4" (Guess probability: 62%)
[+] Host 10.10.1.5 Running OS: "FreeBSD 4.0" (Guess probability: 62%)
[+] Host 10.10.1.5 Running OS: "FreeBSD 3.2" (Guess probability: 58%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.
```

The `Tcpdump` output for the ICMP echo (ping) reachability test:

```
17:32:30.606198 10.10.1.3 > 10.10.1.5: icmp: echo request (ttl 64, id 57271,
len 84)
                         4500 0054 dfb7 0000 4001 1799 c0a8 0103
                         c0a8 0105 0800 f604 dfb7 0000 3eb6 e67e
                         0009 1202 0809 0a0b 0c0d 0e0f 1011 1213
                         1415 1617 1819 1a1b 1c1d 1e1f 2021 2223
                         2425 2627 2829 2a2b 2c2d 2e2f 3031 3233
                         3435
17:32:30.606637 10.10.1.5 > 10.10.1.3: icmp: echo reply (ttl 64, id 53573, len
84)
                         4500 0054 d145 0000 4001 260b c0a8 0105
                         c0a8 0103 0000 fe04 dfb7 0000 3eb6 e67e
                         0009 1202 0809 0a0b 0c0d 0e0f 1011 1213
                         1415 1617 1819 1a1b 1c1d 1e1f 2021 2223
                         2425 2627 2829 2a2b 2c2d 2e2f 3031 3233
                         3435
```

The `Tcpdump` output for the TTL distance reachability test:

```
17:32:32.811008 10.10.1.3.58999 > 10.10.1.5.22: S [tcp sum ok]
3441493741:3441493741(0) win 0 (DF) [ttl 1] (id 3698, len 40)
                         4500 0028 0e72 4000 0106 e805 c0a8 0103
                         c0a8 0105 e677 0016 cd21 06ed 0000 0000
                         5002 0000 71ed 0000
17:32:32.811470 10.10.1.5.22 > 10.10.1.3.58999: S [tcp sum ok]
2022406192:2022406192(0) ack 3441493742 win 5840 <mss 1460> (DF) (ttl 64, id 0,
len 44)
                         4500 002c 0000 4000 4006 b773 c0a8 0105
                         c0a8 0103 0016 e677 788b 7830 cd21 06ee
                         6012 16d0 5294 0000 0204 05b4 7e7e
```

The `Tcpdump` output for the ICMP echo request test:

```
17:32:32.821915 10.10.1.3 > 10.10.1.5: icmp: echo request (DF) [tos 0x6,ECT(0)]
(ttl 64, id 41214, len 84)
                         4506 0054 a0fe 4000 4001 164c c0a8 0103
```

```
                     c0a8 0105 087b a3f0 b954 0001 3eb6 e680
                     000c 89f8 0809 0a0b 0c0d 0e0f 1011 1213
                     1415 1617 1819 1a1b 1c1d 1e1f 2021 2223
                     2425 2627 2829 2a2b 2c2d 2e2f 3031 3233
                     3435
17:32:32.822353 10.10.1.5 > 10.10.1.3: icmp: echo reply [tos 0x6,ECT(0)]  (ttl
64, id 53574, len 84)
                     4506 0054 d146 0000 4001 2604 c0a8 0105
                     c0a8 0103 007b abf0 b954 0001 3eb6 e680
                     000c 89f8 0809 0a0b 0c0d 0e0f 1011 1213
                     1415 1617 1819 1a1b 1c1d 1e1f 2021 2223
                     2425 2627 2829 2a2b 2c2d 2e2f 3031 3233
                     3435
```

The `Tcpdump` output for the ICMP timestamp test:

```
17:32:32.840486 10.10.1.3 > 10.10.1.5: icmp: time stamp query id 47444 seq 0
(ttl 64, id 47444, len 40)
                     4500 0028 b954 0000 4001 3e28 c0a8 0103
                     c0a8 0105 0d00 670f b954 0000 000c d28f
                     0000 0000 0000 0000
17:32:32.840864 10.10.1.5 > 10.10.1.3: icmp: time stamp reply id 47444 seq 0 :
org 0xcd28f recv 0x4d36a16 xmit 0x4d36a16 (ttl 64, id 53575, len 40)
                     4500 0028 d147 0000 4001 2635 c0a8 0105
                     c0a8 0103 0e00 883c b954 0000 000c d28f
                     04d3 6a16 04d3 6a16 7e7e 7e7e 7e7e
```

The `Tcpdump` output for the ICMP address mask test:

```
17:32:32.855299 10.10.1.3 > 10.10.1.5: icmp: address mask request (ttl 64, id
47444, len 32)
                     4500 0020 b954 0000 4001 3e30 c0a8 0103
                     c0a8 0105 1100 35ab b954 0000 0000 0000
```

The `Tcpdump` output for the ICMP information request test:

```
17:32:41.836650 10.10.1.3 > 10.10.1.5: icmp: information request (ttl 64, id
36374, len 28)
                     4500 001c 8e16 0000 4001 6972 c0a8 0103
                     c0a8 0105 0f00 62e9 8e16 0000
```

The `Tcpdump` output for the ICMP port unreachable test:

```
17:32:47.771640 10.10.1.3.53 > 10.10.1.5.65535:  38585% q: A?
www.securityfocus.com. 1/0/0 www.securityfo[|domain] (DF) (ttl 255, id 21319,
len 104)
                     4500 0068 5347 4000 ff11 a4e4 c0a8 0103
                     c0a8 0105 0035 ffff 0054 834a 96b9 81b0
                     0001 0001 0000 0000 0377 7777 0d73 6563
                     7572 6974 7966 6f63 7573 0363 6f6d 0000
                     0100 0103 7777 770d 7365 6375 7269 7479
                     666f
17:32:47.772159 10.10.1.5 > 10.10.1.3: icmp: 10.10.1.5 udp port 65535
unreachable for 10.10.1.3.53 > 10.10.1.5.65535:  38585% q:[|domain] (DF) (ttl
255, id 21319, len 104) [tos 0xc0]  (ttl 64, id 53576, len 132)
                     45c0 0084 d148 0000 4001 2518 c0a8 0105
```

```
c0a8 0103 0303 80bb 0000 0000 4500 0068
5347 4000 ff11 a4e4 c0a8 0103 c0a8 0105
0035 ffff 0054 834a 96b9 81b0 0001 0001
0000 0000 0377 7777 0d73 6563 7572 6974
7966
```

The results are then examined and compared with the OS signature database. Now scores are calculated according to the results from each of the fingerprinting modules. `Xprobe2` returns a primary guess which is the most probable match. It also shows other guesses.

```
[+] Primary guess:
[+] Host 10.10.1.5 Running OS: "Linux Kernel 2.4.5 and above" (Guess
probability: 95%)
[+] Other guesses:
[+] Host 10.10.1.5 Running OS: "Linux Kernel 2.2.x" (Guess probability: 95%)
[+] Host 10.10.1.5 Running OS: "NetBSD 1.6" (Guess probability: 87%)
```

In our example, Linux Kernel 2.4.5 and above and Linux Kernel 2.2.x have received the same score. From Linux Kernel 2.2.x – 2.4.5 and above there are few differences in the implementation of the ICMP protocol. This is due to the fact that they share a very similar TCP/IP stack.

4.5 Defenses

There are some defensive measures to protect against the OS detection method used by `Xprobe2`. Among the defenses are:

- First, machines should be behind some sort of firewall or filtering device. Only needed ports should be left open, the rest should be filtered. This means that there should only be a few ports left open. Filtering all of the UDP ports, and not leaving any open will stop `Xprobe2`. This is a good defense since `Xprobe2` needs at least one closed UDP port to work.
- Block ICMP traffic at the firewall. Blocked both incoming and outgoing traffic.
- Network Intrusion Detection Systems (NIDS) can be used, if configured correctly, to detect the OS detection method used by `Xprobe2` because of the series of tests it performs.

5.0 Conclusion

After analyzing the OS detection methods in the three remote active operating system fingerprinting tools, I've come to realize that each tool works best under different circumstances. Some tools work better against certain network topologies or infrastructures. For instance, `Nmap` works best against remote hosts' that have multiple open TCP and UDP ports. The RINGv2 technology is `Nmap-ringv2` works against any remote host that has at least one open TCP port. This tool will work against almost all network topologies. Finally, there is `Xprobe2`. This tool works well against remote hosts that allow incoming and outgoing ICMP traffic. With all this said, it's clear that you can't always rely on just one tool to remotely determine which operating system is running on the host.

6.0 References

Arkin, Ofir. "ICMP Usage in Scanning – The Complete Know How." June 2001. URL: http://www.sys-security.com/archive/papers/ICMP_Scanning_v3.0.pdf (May 2, 2003).

Arkin, and Yarochkin. "Xprobe v2.0: A "Fuzzy" Approach to Remote Active Operating System Fingerprinting." August 2, 2002. URL: http://www.sys-security.com/archive/papers/Xprobe2.pdf (May 7, 2003).

Arkin, and Yarochkin. "X remote ICMP based OS fingerprinting techniques." August 14, 2001. URL: http://www.sys-security.com/archive/papers/X_v1.0.pdf (May 2, 2003).

Beardsley, Tod. "SANS Intrusion Detection & Analysis Certification." GIAC Certified Intrusion Analysts (GCIA). May 8, 2002. URL: http://www.giac.org/practical/Tod_Beardsley_GCIA.pdf (May 2, 2003).

Dethy@synnergy.net. "Examining port scan methods – Analysing Audible Techniques." URL: http://www.synnergy.net/downloads/papers/portscan.txt (May 2, 2003).

Dwelch@phoneboy.com. "How does SYNDefender Work?" November 18, 2002. URL: http://www.phoneboy.com/fom-serve/cache/153.html (May 6, 2003).

Fyodor. "Remote OS detection via TCP/IP Stack FingerPrinting." June 11, 2002. URL: http://www.insecure.org/nmap/nmap-fingerprinting-article.html (May 2, 2003).

Insecure.org. "Nmap Man Page." URL: http://www.insecure.org/nmap/data/nmap_manpage.html (May 2, 2003).

Miller, Toby. "ECN and it's Impact on Intrusion Detection." Incidents.org DETECT. May 31, 2001. URL: http://www.incidents.org/detect/ecn.html. (May 2, 2003).

RINGv2. "RINGv2." URL: http://ringv2.tuxfamily.org/ (May 2, 2003).

Stevens, W. R. TCP/IP Illustrated, Volume 1: The Protocols. Addison-Wesley, 1994.

Sys-Security Group. "Xprobe." URL: http://www.sys-security.com/html/projects/X.html (May 2, 2003).

Tcpdump.org. "Tcpdump Man Page." URL: http://www.Tcpdump.org/Tcpdump_man.html (May 5, 2003).

Veysset, Courtay, and Heen. "New Tool And Technique For Remote Operating System Fingerprinting." April 2002. URL: http://www.intranode.com/fr/doc/ring-full-paper.pdf (May 2, 2003).

7.0 Acknowledgements

I would like to thank the following people for reviewing the paper:

- Fyodor (fyodor@insecure.org)
- Ofir Arkin
- Olivier Courtay