Global Information Assurance Certification Paper

Network Reconnaissance – Detection and Prevention

GSEC v1.4b
Andy Millican
January 23, 2003

Abstract

The focus of this paper is basic network reconnaissance. I will attempt to define some basic port scanning and OS fingerprinting techniques, a few common tools used to accomplish these scans, and basic methods to detect and defend against these recon attempts. I have included examples of port scan traffic to illustrate my points.

"… virtually every network attack is preceded by network reconnaissance. Hackers scan and probe networks before they attack in order to get information about the target" - Lenny Liebmann [1]

From my experience, both on my personal network and working as an IDS analyst, this statement sums up the importance of regular recon auditing, understanding basic scanning mechanisms, and implementing defense and detection of recon events. An attacker who is able to successfully conduct recon on your network has a much higher likelihood of attempting to compromise your network than an attacker whose recon attempts are thwarted. Furthermore, the attacker with recon information has a higher probability of successfully compromising your network. A network admin who understands network scanning, and can implement both defense and detection from this knowledge, will be able to significantly increase the security level of his or her network while, at the same time, significantly reducing the response time if a network incursion occurs.

The Setup: Tools and Test Network

I used two machines from my personal network to research scanning techniques and to gather the packet captures that I have included in this paper. The host machine (10.0.0.7) is running Linux 2.4.18, with NMAP 3.10 alpha4 and Ring 0.0.1-linux installed. The target machine (10.0.0.5) is a FreeBSD 4.7-STABLE box with both SSHd and Telnetd listening on ports 22 and 23 respectively. All IP addresses are non-routable so no packet sanitation was needed. For each of the scans I performed I limited the scan range to ports 22-24 – hitting one closed and two open TCP ports. I used Nmap to perform all of my port scanning. The Nmap utility contains a suite of network scans that extend even beyond the scope of this paper.

In order to gather packet data, I used a program called Tcpdump on the target machine. Tcpdump is a network utility that can be configured to monitor specific network traffic, including all traffic headers and packet payload information. I also configured Snort 1.9.0 (I used the default install settings) on the target machine as my IDS tool for detecting my port scans and OS fingerprint attempts. I disabled the firewall that was active on the target machine, so that all scan data could be captured.

Port Scanning

Port scanning is defined as "The act of systematically scanning a computer's ports. Since a port is a place where information goes into and out of a computer, port scanning identifies open doors to a computer." - Jupiter Media Corporation [5] When performed regularly, port scanning will provide a network admin with a view of his network as an attacker would see it before attacking. Forearmed with this knowledge, if a network compromise does occur, the network admin will know what services are visible to the outside world and thus where the attack entry was likely to have occurred. In depth knowledge of recon attempts is important to a network admin as it will allow him or her to set up their network defense to properly block many different scans.

The principal behind port scanning is simple: Illicit data from any port you are interested in from the target machine then analyze the data for patterns that differ for closed and open ports. There are several methods to perform port scanning. Each method has its own strengths and weaknesses. I will outline a few port scanning techniques in this paper, specifically: TCP Null , TCP Xmas, TCP FIN, TCP SYN, and UDP port scans.

TCP NULL Scan

"As a general rule, reset (RST) must be sent whenever a segment arrives which apparently is not intended for the current connection. A reset must not be sent if it is not clear that this is the case." RFC 793 [6]

In other words if I send any TCP packet without the RST flag set to a closed port on a remote network, I should receive an RST response. If the port is open (or listening) then I should receive no response. The TCP-NULL scan takes advantage of this RFC standard by utilizing TCP packets with no flags set. For each RST packet received a closed port is logged; when no response is received a remote open TCP port is assumed. This scan does not work on certain operating systems that do not adhere strictly to the RFC standard (Windows 95/NT, Cisco, BSDI, HP/UX, MVS, and IRIX)

Below is the data captured from an Nmap scan I ran using the –sN switch across ports 22-24. As expected the only closed port is 24, which returned an RST packet.

17:28:13.891604 10.0.0.7.36813 > 10.0.0.5.24: . win 3072
17:28:13.891685 10.0.0.5.24 > 10.0.0.7.36813: R 0:0(0) ack 0 win 0
17:28:13.892104 10.0.0.7.36813 > 10.0.0.5.23: . win 3072
17:28:13.892229 10.0.0.7.36813 > 10.0.0.5.22: . win 3072
17:28:14.199533 10.0.0.7.36814 > 10.0.0.5.23: . win 3072
17:28:14.199648 10.0.0.7.36814 > 10.0.0.5.22: . win 3072

IDS (SNORT) Detected: YES
Snort Rule:
[**] spp_stream4: STEALTH ACTIVITY (NULL scan) detection [**]

The default installation of snort was able to detect this scan. The SPP plugin performs stateful packet monitoring for Snort. I will explain stateful devices later in this paper.

TCP FIN Stealth Scan

The TCP FIN scan is identical to the NULL scan except that the FIN flag is set on each outbound TCP packet. The result is the same, closed ports respond with an RST packet.

Once again, I used this Nmap scanning utility, this time with the –sF switch. The closed port 24 responded with an RST packet as expected.

17:26:41.683512 10.0.0.7.50203 > 10.0.0.5.24: F 0:0(0) win 1024
17:26:41.683595 10.0.0.5.24 > 10.0.0.7.50203: R 0:0(0) ack 0 win 0
17:26:41.684078 10.0.0.7.50203 > 10.0.0.5.23: F 0:0(0) win 1024
17:26:41.684202 10.0.0.7.50203 > 10.0.0.5.22: F 0:0(0) win 1024
17:26:41.991955 10.0.0.7.50204 > 10.0.0.5.23: F 0:0(0) win 1024
17:26:41.992070 10.0.0.7.50204 > 10.0.0.5.22: F 0:0(0) win 1024

IDS (SNORT) Detected: YES
SNORT Alert:
[**] spp_stream4: STEALTH ACTIVITY (FIN scan) detection [**]

Again the stateful operations of Snort detected this scan.

TCP XMAS Scan

The TCP XMAS scan is also identical to the NULL and FIN scans except that all flags are set on each outbound TCP packet. Again, only closed ports

respond with an RST packet, and this type of scan will not work on systems that do not adhere to the RFC standard.

Nmap –sX executes this scan. Once again my FreeBSD box dutifully replied on port 24 with an RST packet.

17:27:48.764160 10.0.0.7.47742 > 10.0.0.5.22: FP 0:0(0) win 1024 urg 0
17:27:48.764483 10.0.0.7.47742 > 10.0.0.5.23: FP 0:0(0) win 1024 urg 0
17:27:48.764662 10.0.0.7.47742 > 10.0.0.5.24: FP 0:0(0) win 1024 urg 0
17:27:48.764720 10.0.0.5.24 > 10.0.0.7.47742: R 0:0(0) ack 0 win 0
17:27:49.072919 10.0.0.7.47743 > 10.0.0.5.22: FP 0:0(0) win 1024 urg 0

IDS (SNORT) Detected: YES
SNORT Alert:
[**] spp_stream4: STEALTH ACTIVITY (nmap XMAS scan) detection [**]

Stateful monitoring by Snort also picked up this scan.

TCP SYN Stealth Scan

The TCP SYN scan functions on the premise that a listening port will respond to a TCP connection attempt, while a closed port will reject the connection with an RST packet. For each SYN packet sent, the scanning tool will receive either a SYN-ACK or a RST packet. When a SYN-ACK response is received, the port is noted as open and an RST packet is sent to close the connection.

The data captured below is traffic generated by Nmap using the –sS switch. The two open ports (22 and 23) responded to the SYN requests with SYN-ACK packets as expected.

17:22:21.919848 10.0.0.7.33941 > 10.0.0.5.22: S 1776161780:1776161780(0) win 4096
17:22:21.919886 10.0.0.7.33941 > 10.0.0.5.24: S 1776161780:1776161780(0) win 4096
17:22:21.919907 10.0.0.7.33941 > 10.0.0.5.23: S 1776161780:1776161780(0) win 4096
17:22:21.920048 10.0.0.5.22 > 10.0.0.7.33941: S 1206968201:1206968201(0) ack 1776161781 win 57344 <mss 1460> (DF)
17:22:21.920163 10.0.0.7.33941 > 10.0.0.5.22: R 1776161781:1776161781(0) win 0 (DF)
17:22:21.920220 10.0.0.5.23 > 10.0.0.7.33941: S 2295585801:2295585801(0) ack 1776161781 win 57344 <mss 1460> (DF)
17:22:21.920315 10.0.0.7.33941 > 10.0.0.5.23: R 1776161781:1776161781(0) win 0 (DF)

IDS (SNORT) Detected: NO

The reason Snort did not detect this scan is outlined in a this paper under the section on “Detecting and Blocking” port scans in the subsection “Connection Logging and Correlation.”

UDP Port Scan

So far we have discussed only TCP scanning techniques. However, there are other protocols and methods of scanning that do not use TCP and are still useful to auditors and attackers. One such scan is the UDP port scan. Similar to the TCP SYN scan, the UDP scan will send a UDP packet to each port on a target. Any open UDP port will accept the packet, sending no reply, any closed port should respond with an ICMP unreachable packet. Though certainly not foolproof, this scan may be effective in finding unknown open UDP ports.

Below is the traffic capture from nmap using the –sU switch. There are no UDP ports running on ports 22-24 so I received an ICMP unreachable packet from each port scanned.

17:25:46.945054 10.0.0.7.33049 > 10.0.0.5.22: udp 0
17:25:46.945129 10.0.0.5 > 10.0.0.7: icmp: 10.0.0.5 udp port 22 unreachable
17:25:46.945458 10.0.0.7.33049 > 10.0.0.5.24: udp 0
17:25:46.945499 10.0.0.5 > 10.0.0.7: icmp: 10.0.0.5 udp port 24 unreachable
17:25:46.945669 10.0.0.7.33049 > 10.0.0.5.23: udp 0
17:25:46.945710 10.0.0.5 > 10.0.0.7: icmp: 10.0.0.5 udp port 23 unreachable

IDS (SNORT) Detected: NO

The reason Snort did not detect this scan is outlined in a this paper under the section on “Detecting and Blocking” port scans in the subsection “Connection Logging and Correlation.”

Detecting and Blocking Port Scans

Now that I have determined how port scanning works and what a port scan will look like I will attempt to answer the question: How do I detect and block port scans? There are several methods for detecting or blocking port scans. I will give examples of ways to detect and block the scans that have been outlined in this paper. These techniques can be used for many scans not listed in my examples.

Signatures

An effective method for blocking scans is to have a device or service at your network perimeter that can recognize scan packets and drop them. The difficulty arises when you try to sort the malicious scan packets from the legitimate traffic. Some of the scans listed above have a unique signature or pattern that can identify them. Of the scans listed in this paper, the XMAS scan is most easily recognizable as all TCP flags in the TCP header are turned on. This makes the scan easier to detect and block. The same method cannot be applied to other scans. Take the TCP SYN (or half open) scan for example. If you write a signature to block all incoming TCP SYN packets, you will certainly stop any scans, but also any legitimate TCP connections will cease to function. Signatures work best on unusual or malformed packets.

Stateful Firewalls/IDS

Port scans such as the TCP NULL, XMAS, and FIN scan were designed to circumvent stateless firewalls and stateless IDS devices. A stateless device is not able to correlate packets that are part of the same connection - such a device can only process one packet at a time. These devices are generally limited to signature based detection and prevention methods. Stateless devices work fine against basic TCP SYN scans but could often be confused by abnormal packets such as the NULL, XMAS, and FIN scan. The introduction of stateful devices remedied the problem. These devices could keep track of what connection a packet was part of and how it related to other packets. A NULL, XMAS, or FIN scan would be detected and blocked by a stateful device because the device would realize that the packets were not part of an already established TCP stream (which should start with the SYN/ACK handshake. A stateless device could not block the FIN scan without blocking legitimate traffic, and also a stateless device would require manual signatures to be written to catch and block XMAS and NULL scans.

By default, Snort installs with stateful IDS active. As shown by my examples, snort was able to detect the FIN, NULL, and XMAS scans. These types of packets should never precede a TCP handshake. Snort was aware that no SYN-SYN/ACK handshake had occurred and logged the suspicious FIN, NULL, and XMAS TCP packets.

Connection attempt Logging and Correlation

The installation of Snort on my target machine was unable to detect the TCP SYN scan and the UDP scan. The reason for this was that the connection-logging feature was not turned on in Snort. TCP SYN and UDP scans appear to be legitimate connection attempts to the target, their only distinguishing trait is that they hit multiple ports. The SYN packet is sent but the handshake is not completed. This will occur naturally due to network disruptions, but should not be a common occurrence. This should certainly not occur five times in less than a

second from the same source IP. Snort does have the ability to log multiple connection attempts from the same source IP address (although it does not by default.) A SYN port scan or UDP port scan would be detected by this method, as the SYN/UDP scan appears to be multiple failed legitimate connections. By correlating connection attempts, it becomes apparent that the source IP was performing a scan on my target machine, as the source made five connection attempts to different ports in less than a second. Depending on your network, a setup that logged five or more connection attempts from the same source IP in ten second time window would correctly detect most SYN/UDP scan activity.

Port Scan Stealthing and Obfuscation

Knowing techniques that are employed to stealth or obfuscate a scan are important to any network admin. With the proper understanding of port scan stealthing, a network admin will be able to determine which detection method will work most efficiently while detecting as many port scans as feasible.

Several methods exists that will stealth a scan so that it will not show up on an IDS device. As we have seen from the examples, some scans are not picked up by stateless firewalls. This is a basic form of stealthing and was demonstrated by the XMAS, NULL, and FIN scans. A further method of port scan stealthing is something called TCP fragmenting. By breaking a scan into tiny TCP packets (packets so small that the headers span many packets) a scan will bypass many basic IDS and firewall devices.

An attacker may choose to obfuscate his scan instead of trying to stealth it. A common obfuscation method is to spoof many source IP addresses along with the legitimate source IP for a port scan. The target machine will likely log the scan, but it will be extremely difficult for the network admin to determine from which IP address the port scan actually originated. The downside to this method is that it does set off any port alarms on the target network and will likely put the network admin in a state of alert.

OS Fingerprinting

The term OS fingerprinting defines any method used to determine what operating system is running on a remote computer. OS fingerprinting is a key element in network reconnaissance as most exploitable vulnerabilities are operating system specific. An attempt to exploit a Microsoft IIS vulnerability on a Linux 2.4 machine is doomed to fail. If an attacker is able to determine what remote operating system a target is running then he or she will likely be able to cross off a large number of exploits from their known exploit list and instead concentrate on exploits that may work. This will both decrease the likelihood of an attack being detected and greatly increase the chances of an attack being successful.

Much like port scanning, OS fingerprinting has evolved over time and there are multiple methods to successfully fingerprint an OS. The general methods used for OS fingerprinting are to find distinguishing traits for each operating system, traits that can be observed from a remote network. More specifically these methods can range from examining the default TCP window size in a packet, to measuring the amount of data in ICMP packets, and even gauging TCP initial sequence numbers. In the next section of this paper I will demonstrate one simple method (TCP banners) and two tools (Nmap, and Ring) that can be used in an attempt to gain an accurate OS fingerprint.

TCP Banners

Querying the services running on a target machine is often the simplest and quickest method for OS fingerprinting. Many servers actively announce their operating system to any computer that attempts to make a connection. While this may seem like a nice feature to some users, this is very disconcerting to a security conscious network administrator.

In the example below, SSH is listening on the target machine so I simply telnetted to SSH port 22 to establish a TCP connection and waited for the service to tell me something important.

```
etc# telnet 10.0.0.5 22
Trying 10.0.0.5...
Connected to 10.0.0.5.
Escape character is '^]'.
SSH-1.99-OpenSSH_3.4p1 FreeBSD-20020702
```

As you can see I have already managed to fingerprint my FreeBSD 4.7 as a FreeBSD box running SSH-1.99-OpenSSH_3.4p1. I also have a date 2002-07-02. I can now find out which FreeBSD operating systems are likely to have this version of SSH installed.

Although examining TCP banners appears to be the perfect way to fingerprint any OS, there are downsides to this method. Most banners are easily modified or disabled, so it is possible that this method will return either no information or incorrect information.

Nmap OS Fingerprint

If insufficient data is available from a target's TCP banners, then Nmap may be a popular next step. Not only does Nmap provide multiple methods of port scanning, it also combines most modern OS fingerprinting techniques into

one tool. The Nmap utility cross-references several methods of fingerprinting a remote OS, a comprehensive list of methods used by Nmap can be found at the Nmap web page http://www.insecure.org/nmap/nmap-fingerprinting-article.html .

I ran two examples of the Nmap fingerprint function. The first example was run against my standard target – the FreeBSD 4.7 box at 10.0.0.5.

```
:/etc# nmap -sS -p 20-30 -P0 -O 10.0.0.5

Starting nmap V. 3.10ALPHA4 ( www.insecure.org/nmap/ )
Interesting ports on 10.0.0.5:
(The 9 ports scanned but not shown below are in state: closed)
Port       State       Service
22/tcp     open        ssh
23/tcp     open        telnet
No exact OS matches for host (If you know what OS is running on it, see
http://www.insecure.org/cgi-bin/nmap-submit.cgi).
TCP/IP fingerprint:
SInfo(V=3.10ALPHA4%P=i586-pc-linux-
gnu%D=1/22%Time=3E2F5980%O=22%C=20)
TSeq(Class=TR%IPID=I%TS=100HZ)
T1(Resp=Y%DF=Y%W=E000%ACK=S++%Flags=AS%Ops=MNWNNT)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=E000%ACK=S++%Flags=AS%Ops=MNWNNT)
T4(Resp=Y%DF=N%W=0%ACK=O%Flags=R%Ops=)
T5(Resp=Y%DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6(Resp=Y%DF=N%W=0%ACK=O%Flags=R%Ops=)
T7(Resp=Y%DF=N%W=0%ACK=S%Flags=AR%Ops=)
PU(Resp=Y%DF=N%TOS=0%IPLEN=38%RIPTL=148%RID=E%RIPCK=
E%UCK=0%ULEN=134%DAT=E)


Uptime 3.019 days (since Sun Jan 19 21:27:04 2003)

Nmap run completed -- 1 IP address (1 host up) scanned in 37.284
seconds
```

Nmap OS fingerprinting functions by comparing multiple values to a fingerprint file. In the example above no match was found for the data collected. Nmap remains up to date because it allows end users to submit new fingerprint data via the Nmap web page.

I ran a second scan against a Windows XP box at 10.0.0.3 to get a better example of the accuracy of Nmap's OS fingerprinting.

```
root@windows:/etc# nmap -sS -p 1-1024 -P0 -O 10.0.0.3

Starting nmap V. 3.10ALPHA4 ( www.insecure.org/nmap/ )
Interesting ports on 10.0.0.3:
(The 1021 ports scanned but not shown below are in state: closed)
Port       State       Service
135/tcp    open        loc-srv
139/tcp    open        netbios-ssn
445/tcp    open        microsoft-ds
Remote operating system guess: Windows 2000/XP/ME

Nmap run completed -- 1 IP address (1 host up) scanned in 20.962
seconds
```

This time the results were better. A match was found in the fingerprint database for Windows 2000/XP/ME. This is not as precise as I would have hoped (I am running Windows XP not 2000 or ME) but it does give me a general OS fingerprint.

Ring OS Fingerprint

If the information gathered from TCP banners and from Nmap is not sufficient then there are always new tools and techniques being researched that allow better OS fingerprinting. Franck Veysett, Olivier Courtay, Olivier Heen, and the Intranode Research Team released one such tool as proof of concept. The program, called Ring, was released along with a white paper in April of 2002. Ring functions by measuring the delay between an initial TCP packet and the corresponding Retransmission Time Out (RTO) packet. During a normal TCP session it is likely that one or two packets will not reach their destination. When this happens the sender will not receive an ACK packet from the target. After a certain time with no ACK response the sender will retransmit the original packet. Ring functions by measuring RTOs, compensating for network distance, and comparing the results to a fingerprint file. The theory is that each operating system uses slightly different wait times before resending a TCP packet.

I installed the proof of concept version of Ring on my Linux machine to give it a test run. I was a bit skeptical considering the fingerprint file only contained twelve entries but the results were quite accurate.

```
usage : ring -d destination_address -s source_address -p port -f
signature_file -i interface
~# ./ring -d 10.0.0.5 -s 10.0.0.7 -p 22 -f ./fingerprint -i eth0
2999767 6000910 12001833
OS:FreeBSD4.5
distance:3498
```

Ring was able to match my FreeBSD 4.7 machine against the only FreeBSD signature it had – FreeBSD 4.5. I ran the same test five times with the same results, so Ring has proven, at least in this instance, both accurate and consistant.

I ran Ring again against my Windows XP machine. This time the fingerprint returned as Windows 2000. Windows XP is based on much of the same code as Windows 2000, so it is a close OS fingerprint match.

```
~# ./ring -d 10.0.0.3 -s 10.0.0.7 -p 139 -f fingerprint -i eth0
2932357 6009000
OS:Windows_2000
distance:21442
```

The aspect of Ring that I found most helpful is the fingerprint file. The file is stored in plain text, and it is easy to add your own entries. I took the data I collected from the Windows XP fingerprint and added it to the fingerprint file. I ran Ring again and here are the results:

```
:~# ./ring -d 10.0.0.3 -s 10.0.0.7 -p 139 -f fingerprint -i eth0
3002399 6008946
OS:Windows_XP
distance:70096
```

As you can see Ring was much more accurate than Nmap as far as fingerprinting the two machines on my local network.

Defense against OS Fingerprinting

Now that we have examples of how to gather OS fingerprint information an important question to answer is: How do we stop OS fingerprinting? Unfortunately, since OS fingerprinting relies on gathering data that is usually available in normal IP traffic, it is very difficult to block OS fingerprinting. One way to do so is to block direct access to sensitive computers. This can be done via Network Address Translation (NAT). In effect this leaves only one machine (the one with the routable IP address) susceptible to most external OS fingerprinting attempts. You must, of course, make sure that the live box is up to date and patched against all publicly known vulnerabilities.

Suggested Reconnaissance Countermeasures

So we have determined that OS fingerprinting and port scanning is generally a bad thing, but how do we block or detect it? A layered protection is always best; however defense strategies for every network will be different. I

would suggest layering a stateful firewall, Intrusion detection device/process, and using NAT for as much of the network as possible.

A stateful firewall placed on a network perimeter is likely one of the greatest prevention measures for any intrusion. The firewall should be configured to allow only the necessary traffic and should log multiple connection attempts from the same source IP address. This allows the firewall to detect SYN scans, and, since the firewall is stateful, it should block anomalous scans such as the FIN, NULL, and XMAS scans.

The next layer of defense should be detection. Add a device or process behind the firewall that monitors network traffic for anomalous activity. In the examples I put into this paper I used Snort as my intrusion detection. I highly recommend using it. Snort should also be configured to log multiple connection attempts from the same source in a given time range. The log files generated by Snort must be maintained and regularly reviewed; otherwise there is no point to having intrusion detection.

As a final defense step, you should configure the fewest number of machines with live IP address and use NAT and private IP space for the rest of your network. This will help block many OS fingerprint attempts. It is important to harden the machine that does have a live IP address, as it is your point of weakness in a NAT environment. Apply all known patches for whatever operating system the NAT device is running and try to insure that it is not vulnerable to any publicly known exploits.

Conclusion

From port scanning to OS fingerprinting, network reconnaissance is useful and interesting to both a would-be attacker and a network auditor. Network reconnaissance will almost always be the first step an attacker takes in compromising any network. Realizing the truth of this statement, a security conscious network admin can take preemptive action against network intrusions by using any reconnaissance blocking method that is feasible. This paper does not cover every possible reconnaissance event, but I hope that the concepts presented here will help any network admin better secure their network from outside intrusion attempts.

References:

1. Liebmann, Lenny. “The Bottom Line” URL: http://www.comnews.com/stories/articles/c0702bottom.htm (July 2002).

2. Fyodor. “Nmap network security scanner man pages” URL: http://www.insecure.org/nmap/data/nmap_manpage.html

3. “TCPDump Man Pages” URL: http://www.tcpdump.org/tcpdump_man.html (December 2002).

4. “Snort FAQ” URL: http://www.snort.org/docs/faq.html (March 2002).

5. Jupiter Media Corporation. “Webopedia” http://www.webopedia.com/TERM/P/port_scanning.html (2003).

6. Information Sciences Institute University of Southern California. “RFC 793 – Transmission Control Protocol” URL: ftp://ftp.rfc-editor.org/in-notes/rfc793.txt (September 1981).

7. Dethy@synnergy.net. “Examining port scan methods - Analysing Audible Techniques“ URL: http://packetstormsecurity.org/groups/synnergy/portscan.txt (2001).

8. iEntry, Inc. “Stateful vs. Stateless IP Filtering” URL: http://securitypronews.com/2002/0214.html (Febuary2002).

9. Fyodor. “Remote OS Detection via TCP/IP stack fingerprinting” URL: http://www.insecure.org/nmap/nmap-fingerprinting-article.html (June 2002).

10. Franck Veysett, Olivier Courtay, Olivier Heen, Intranode Research Team. “New Tool And Technique For Remote Operating System Fingerprinting” URL: http://www.intranode.com/pdf/techno/ring-full-paper.pdf (April 2002).

11. Beardsley, Tom. “Intrusion Detection and Analysis: Theory, Technique, and Tools” URL: http://www.giac.org/practical/Tod_Beardsley_GCIA.pdf (May 2002).