# A Review of Port Scanning Techniques

*Marco de Vivo* mdevivo@reacciun.ve
*Eddy Carrasco* ecarrasco@ciens.ucv.ve
*Germinal Isern* gisern@ciens.ucv.ve
*Gabriela O. de Vivo* gdevivo@reacciun.ve
Apartado Postal 68274
Caracas, Venezuela.

**LACORE U.C.V.**

## Abstract *

This paper reports the most important techniques used by **TCP *port scanners***. TCP port scanners are specialized programs used to determine what TCP ports of a host have processes *listening on* them for possible connections. Since these ports characterize, in part, the amount of exposure of the hosts to potential external attacks, knowing their existence is a fundamental matter for network and/or security administrators. Moreover, as scanners are also used by hackers, administrators need to know how they work and what possible weakness they exploit to be able to prevent unwanted scanning or at least to record each scanning attempt.

**Keywords:** TCP/IP, UDP, Three-way Handshake, SYN Scanning, FIN Scanning, Stealth Scanning, Indirect Scanning, Decoy Scanning, Fingerprinting, and Coordinated Scanning.

## Introduction.

In the first section we review the TCP connection establishment process (often called the *three-way handshake*), and discuss some implementation details that are relevant to the design of scanner programs.
Next, classical scanners (*full TCP connection*) are reviewed as well as the so-called ***SYN*** (or "*half open*") scanners.
A third section is devoted to study ***indirect*** and ***stealth*** scanning; techniques developed to conceal scanning attacks and/or their origin.
Stealth scanners are based on the use of ***FIN*** segments. The idea is that in most implementations, closed TCP ports reply to a FIN segment with a ***RST***, while open ports usually discard the segment with no reply at all. Indirect scanning, as will be explained, is realized with the (usually involuntary) help of a spoofed host.
In the next section several scanning techniques developed to bypass *firewalls* analysis and/or filtering are discussed. We first explain the use of IP fragmentation to split up the TCP header of a SYN or FIN probe over several packets. Then ***decoy*** scanning is presented and finally several forms of ***coordinated*** scanning are analyzed.
The fifth section is dedicated to examine the basics of UDP scanning.
The sixth section presents several forms of scanning related to specific *application level protocols*. Usually these scanners exploit weak and/or faulty implementations as well as permissive features. The ***ident*** (or *reverse ident*) scanning is explained as well as the ***proxy*** based scanning approach.
In the last section, additional scanning techniques are reviewed. Scanning tools not based only on TCP *port* abstraction or designed mainly for ***security scanning*** are considered (e.g., **SATAN**). Also the use of scanners for ***stack fingerprinting*** is analyzed. Stack fingerprinting solves the problem of operating systems identification in a unique way, by probing a host's TCP and comparing the answer (or the lack of answer) against the results expected from known operating systems TCP/IP stacks.

## 1. TCP/IP Relevant Issues.

**End Points and Flags**:
The pair (IP address, port number) is called a *socket* and represents an end point of a TCP connection. To obtain TCP service, a connection must be explicitly established between a socket on the sending machine and a socket on the receiving machine. TCP connections are thus identified by its two end points, that is (*socket1*, *socket2*). End points send each other ***segments***.

* This work was partially supported by C.D.C.H. (U.C.V.) under Grant No. 03-13-4051-97

A TCP segment consists of a *TCP header* optionally followed by data. A TCP header includes six flag bits. One or more of them can be turned on at the same time **[1,2]**:

- **SYN** Synchronize sequence numbers to initiate a connection.
- **FIN** The sender is finished sending data.
- **RST** Reset the connection.
- **URG** The *urgent pointer* is valid.
- **ACK** The acknowledgment number is valid.
- **PSH** The receiver should pass this data to the application as soon as possible.

**TCP Connection Establishment**:
TCP is a connection-oriented, reliable transport protocol. Connection-oriented means the two applications using TCP must establish a TCP connection with each other before they can exchange data. Reliability is provided in TCP by the use of checksums, timers, data sequencing and acknowledgments. By assigning a sequence number to **every** byte transferred, and requiring an acknowledgment from the other end upon receipt, TCP can guarantee reliable delivery. Sequence numbers are used to ensure proper ordering of the data and to eliminate duplicate data bytes. Note that in a TCP session there are usually two streams of data (every end point is receiving from one of them and sending through the other). So an ***ISN*** (initial sequence number) must be assigned to each stream when the connection is being established. To see how it is done, let's suppose that **C** is a client wishing to connect to the server **S**, and analyze the connection establishment process (often called the *three-way handshake*):

| | | | |
|---|---|---|---|
| 1 | **C** | ---SYN XX---> | **S** |
| 2 | **C** | <---SYN YY/ ACK XX+1--- | **S** |
| 3 | **C** | ---ACK YY+1---> | **S** |

**1. C** sends a TCP message (known as SYN request) to **S** with the special flag SYN (SYNchronize sequence numbers) set to ON. A SYN request specifies the port number of the server that the client wants to connect to, and the client's ISN (XX in this example).
**2.** The server responds with its own SYN message containing **S**'s ISN (YY) and acknowledging **C**'s SYN by specifying that the **next** byte expected from **C** is the byte numbered XX+1.
**3. C** acknowledges the SYN message from **S**, and data transfer may take place.

**Some Implementation details**:
Most TCP/IP implementations (with important exceptions, though) follow these rules **[3,4]**:

- When a **SYN** (or **FIN**) segment arrives for a *closed port* (i.e., for which no TCB exists), TCP drops the segment and sends a **RST**.
- When a **RST** segment arrives for a *listening port*, it is simply dropped.
- When a **RST** segment arrives for a *closed port*, it is simply dropped.
- When a segment containing an **ACK** arrives for a *listening port*, it is dropped and a **RST** is sent.
- When a segment with the **SYN** bit **off** arrives for a *listening port*, it is simply dropped.
- When a **SYN** segment arrives for a *listening port*, the normal *three-way handshake* continues by replying **SYN**|**ACK**.
- When a **FIN** segment arrives for a *listening port*, it is simply dropped. "**FIN** behavior" (closed port = **RST**, listening port = dropped) can also be seen with the **PSH** and **URG** flags, with a TCP segment with no flags, and with all the combinations of **FIN**|**PSH**|**URG [5]**.

## 2. *Full TCP Connection* and *SYN* Scanners.

**Full TCP Connection**:
This has been for long the fundamental form of TCP port scanning. The scanning host tries to establish regular connections (i.e., using the full three-way handshake) with chosen ports on the target machine. Connections are started using the system call ***connect* () [6]**. For every listening port, *connect* () will succeed, otherwise a **–1** is returned indicating that the port is unreachable. This technique is easily implemented because connect () can be accessed by any user (even under multi-user systems) since, usually, no special privileges are needed.
This scanning method is easily detectable (logs will report a rapid sequence of connections and error). Monitoring programs like ***Courtney***, ***Gabriel***, and ***TCP Wrappers* [7]** are usually employed for detection. Besides, since wrappers also provide some control over incoming requests, they can be used to (try to) prevent full connection scanning originating from known hosts.

**TCP SYN Scanning**:
In this technique, the scanning host performs the equivalent of an *active open* by sending a SYN segment to a selected port on the target machine. If the

answer is a RST then the port is closed and, if scheduled, a different port is probed; else, if a SYN|ACK segment is received, it means that the target port is indeed listening. Since this information is all the scanning host needs to know, a RST is sent back to the target to break the connection establishment. Observe that since under SYN scanning not full connections are ever established, this technique is frequently known as *half-open* scanning. The main advantage of SYN scanning resides on the fact that even if this method can still be detected by certain loggers (e.g., ***tcplogger***), SYN connection attempts are logged less frequently than full connections. The trade-off is that the sender, under most operating systems, needs to custom build the entire IP packet for this kind of scanning, and usually *super-user* or privileged group access to special system calls is needed to build these custom SYN packets **[8]**.

Both options (SYN and full connection scanning) as well as almost all the techniques discussed in this paper are implemented in **nmap [9]**, a very powerful and comprehensive TCP port scanner.

## 3. *Stealth* and Indirect Scanning.

**Stealth** (FIN, *Xmas Tree*, *Null*) **scan techniques**:
*Stealth scanning* was first proposed in **[8]**. Since this technique does not consist of any part of the standard TCP three-way handshake, it is very difficult to log and thus more clandestine that SYN scanning. Besides, FIN segments may be able to pass through packet filters watching only for SYNs.
The stealth scan technique uses FIN segments to probe ports. As explained in **section 1**, when a FIN segment arrives for a closed port, the receiving TCP drops the segment and sends back a RST. Otherwise, when a FIN segment arrives for a listening port, it is simply dropped (no RST sent).
*Xmas Tree*, and *Null* scan modes **[9]** are just variations of the preceding one based on the fact that "FIN behavior" (closed port = RST, listening port = dropped) can also be seen with the PSH and URG flags, with a TCP segment with no flags, and with all the combinations of FIN|PSH|URG. The Xmas Tree scan turns on the FIN, URG, and PUSH flags. The Null scan turns off all flags. These combinations are supposed to eventually bypass certain filters like the so-called *FIN FLAG detectors*.
Stealth scanning usually works with UNIX targets except for those few systems (including Cisco, BSDI, HP/UX, MVS, and IRIX) that send resets when they should just drop the packet. The method however fails with systems running Windows95/NT since they always send back a RST whether the probed port is open or closed.
As in the case of SYN scanners, the IP packets must still be custom built.

**Indirect Scanning**:
A very interesting technique for anonymous scanning was recently proposed in **[10]**. The idea is to use the (spoofed) IP address of a third host to disguise the real scanning system. Since the scanned host will react by sending or not certain segments to the spoofed host, all that is needed is to monitor the IP activity of the spoofed host to know the results of the original scan. This form of indirect scanning is called by the author "dumb host scan" and works as follows:
Suppose that the three participating hosts are **Scanner**, **Silent**, and **Target**. The roles of **Scanner** and **Target** are obvious. **Silent** is a very particular host that must not send any packets while **Scanner** is scanning **Target** (except for the ones related to the scan). The author suggests as candidates for **Silent**, any of the reachable "zero traffic" hosts that can be found at "night" in the Internet. Finally, all the involved TCP implementations must be well behaved (i.e., following the general rules listed in section **1**).

**Scanner** starts the process by sending probes to **Silent** (e.g., ICMP echo requests) in order to monitor the identification values contained in the IP headers of the replies (from now on we will call this value **IP-id**). Since **Silent** is not exchanging packets with any other host, each of its successive replies to **Scanner** will show an IP-id increment of 1.

Next, **Scanner** sends some SYN segments to port X on **Target** using packets with the spoofed IP address of **Silent**.

- If X is a closed port, then **Target** sends the same amount of RSTs to **Silent**, which in turn simply drops them. In this case, since no additional packets are sent by **Silent**, each IP-id of its successive replies to **Scanner** will still show an increment of 1 which will be interpreted as X being a closed port on **Target**.
- If X is a listening port, then **Target** will answer by sending SYN|ACK segments to **Silent**, which, of course, will send back the corresponding RST segments. Since every RST sent will cause an increment of its IP-id counter, the series of replies from **Silent** to **Scanner** will not show anymore a constant IP-id increment of 1, which will be interpreted by **Scanner** as X being a listening port on **Target**.

Note that since this technique assumes the correct delivery of IP datagrams, eventual implementations might be not so straightforward owing to the unreliable nature of IP.

## 4. *Fragmented, Decoy,* and *Coordinated* Scanning.

**Fragmented Packets Scanning**:
The anonymity of SYN, FIN, XMAS, or NULL scans can be improved by using tiny fragmented IP packets. In this way, the TCP headers themselves are split up into several packets, making harder for packet filters and intrusion detection systems to detect or prevent the scan.

***Decoy* Scanning**:
SYN, FIN, XMAS, and NULL scans, as well as several other forms not yet discussed, can be further *anonimyzed* by the use of *decoy* probes. In this technique several probes are sent to the same target. All packets but one (the real probe) are sent with spoofed addresses, so that even if the port scans are detected, there is no way to know which was the real scanning host and which were mere decoys.
Besides being a serious security threat, decoy scanning presents some unpleasant side effects:

- The hosts used as decoys must be up and reachable, otherwise the target might be accidentally ***SYN flooded* [11]**. Of course, the more concerned with this matter should be the intruder himself, because if the decoys were all unreachable it would be quite obvious which host is the real scanner.
- Since some "scan detectors" react by denying access to hosts that (supposedly) attempt port scans, involuntary ***DoS* [11]** attacks against the target machine (causing it to lose connectivity with the decoys) might result.

**Coordinated Scanning**:
In **[12]** a (possibly) new form of scanning is reported. In that work, the authors analyze attacks and probes that have been recently observed in which multiple attackers are clearly working together toward a common goal from different IP addresses. The following is a quotation from the document's summary:

**"**Often these IP addresses are also physically separated, in different countries or even different continents. There are three obvious purposes for this type of activity:

- **Stealth**. By working from multiple IP addresses the attackers achieve a smaller per-IP signature and are more difficult to detect with conventional means. In addition, stealth is enhanced by the development of new hard-to-detect probing techniques.
- **Firepower**. By coordinating multiple attacking IP addresses, the attackers will be able to deliver more exploits on targeting a smaller time window. Target in this case can be one or more sites. Further, the defense technique of blocking an attacker IP or subnet (shunning) will be less effective. We believe that the use of coordinated scans and probes from differing sites represents a new and continuing capability that merits further analysis and tracking. Some of these coordinated probes and scans we are seeing today may be practice runs for future larger scale attacks.
- **More data**. By working from different IP addresses, often entirely different subnets, against the same target it is possible to obtain data that is difficult from a single source IP scan or probe. This data may include shortest route data (i.e. packets from source A arrive faster than from source B), or even potential backdoors (i.e. packets from source A can gain access to hosts that source B can't see). This type of data can be used to optimize future scans, probes, or attacks.

**Analysis**: Multiple different attacks and probes are documented here. The commonality is that the attacker is able to launch the attack from multiple unrelated (or partially related) addresses in a coordinated fashion. Special thanks to Vicki Irwin, and Pedro Vazquez for their help in deciphering this puzzle.**"** *–end of quotation*.

Several observed cases, that supposedly back up their theory, are presented and discussed in the document:

1. Five different sources using *traceroute* coordinately possibly to have timing data for multiple paths.
2. Simultaneous *reset* probes sent from 14 different IP addresses (common *acknowledgement numbers* signature) trying possibly to help discovering a subnet map.
3. Three attackers detected (in a short frame time) searching for ***Back Orifice*** (default, UDP 31337) **[13]** on the same set of targets.
4. Coordinated probes against a firewall.
5. Low rate simultaneous scans to locate DNS servers.

A word of caution, though: even if the types of scans and the time frames observed in some of the reported cases might seem to confirm their theory, there is a good chance that most of the observations are simply the result of the obscuring techniques (e.g., decoys) used by current scanners[1].

---

[1] Shadow's document has been recently updated with the inclusion of a special note that starts with the words: "This document has largely been overtaken by new advances in hacker technology."

## 5. UDP Scanning.

UDP scans are used to determine which UDP ports are open on a host. Even if UDP scanning plays a minor role in port scanning, it must not be completely neglected since it can be very useful for certain tasks:

- The recent case of ***rpcbind*** security hole. On Solaris 2.x operating systems, rpcbind listens not only on TCP port 111, and UDP port 111, but also on a port greater than 32770. This fact, of course, renders useless a large number of packet filters. Instead of sending requests to TCP or UDP port 111, the attacker simply sends them to the UDP port greater than 32770 on which rpcbind is listening. UDP scanners are needed to check for this breach.
- The (even more recent) Back Orifice backdoor program hides on a configurable UDP port on Windows machines. Again, UDP scanners are useful for detection.
- Several important services utilize UDP (e.g., DNS, NFS, SNMP, TFTP, etc.). UDP scanning may be used to detect fake servers or unauthorized services.

UDP scanners use the fact that when a packet for a closed UDP port arrives, UDP responds with an **ICMP** *port unreachable* error message **[14]**. So, a typical UDP scanner works by sending 0 bytes UDP packets to selected ports on a target machine, and then waiting for possible replies. ICMP *port unreachable* replies imply that the associated ports are closed. No response usually indicates that the associated ports are open (some of the probes or replies could simply got lost, though).

## 6. *Ident* and *Proxy* Scanning.

***Ident* Scanning**:

Until now, we have been analyzing scanners designed with just one goal: to determine what ports of a host have processes *listening on* them. Several current port scanners **[9**,**15]**, however, are able to perform additional functions aimed to explore the characteristics and behavior of the processes (*daemons*) listening on those ports.

*Ident* scanning is an interesting example of those extra options. By exploiting the ***Identification Protocol*** **[16]**, this type of scan has the additional functionality of retrieving the username (*Userid*) that owns the daemon running on a specified port. The scanner does this by attempting to connect to a TCP port, and if it succeeds (full connection), it will send out an **ident request** to ***identd*** (the *Identification Protocol* daemon) on TCP port 113 of the target host. Ident scanning option is also known as ***reverse ident*** scanning, because even if the original *RFC* seems to suggest a protocol for helping servers authenticate clients, the actual implementations allow for this reverse use (clients identifying servers).

***Proxy* Scanning**:

The **ftp** protocol supports an interesting option known as ***proxy*** ftp connection. The original intention of this feature (as suggested in the *RFC* **959 [17]**) was to allow a client to simultaneously establish two ftp connections with different servers and then start a data transfer directly between the servers. Most implementations, however, can be easily abused to have an ftp server send files practically anywhere on the Internet. This weakness allows for several nasty attacks, including the well-known ***ftp bounce attack*** **[18]**. Some recent port scanners **[9]** exploit the same vulnerability to implement the so-called ***ftp proxy scanning***. Ftp proxy scanners basically use "proxy" ftp servers to scan TCP ports. This is done as follows:

- Suppose that **S** is the scanning machine, **T** the target, and **F** an ftp server with the proxy option enabled and capable to establish connections with both **S** and **T**.
- **S** establishes an ftp session with **F**, and using the **PORT** command declares a selected port on **T** (let's say **p-T**) as the passive port needed for a proxy transfer.
- Then **S** tries to start a proxy data transfer to **p-T** by issuing a **LIST** command (or equivalent).
- If **p-T** is indeed listening, then the transfer will be successful (reply codes 150 and 226 sent back to **S**). Otherwise **S** will get a "425 Can't open data connection" reply.
- **S** keeps issuing PORT and LIST commands until all selected ports on **T** are scanned.

Ftp proxy scanning is not only hard to trace, but can constitute a real security threat when the participating ftp server is behind a firewall.

## 7. Additional Scanning Techniques.

***Ping* Scanning**:

When hundreds or thousands of ports are to be scanned on a machine or even on an entire subnet, it really makes sense to first try to know if the target machines are up. This is the goal of *ping* scanners. Two main techniques are used to implement ping scanning:

- Using real *pings*, i.e. sending ICMP echo re-

quests to the candidate IP addresses and marking as **up** every responding host.

- Using "TCP *pings*", i.e. sending special TCP segments to probe ports that are usually open and not filtered (e.g., port 80). For non-root users of the scanner, the standard *connect* () is employed. Otherwise, ACK segments are sent to probe every candidate address. Each RST received back indicates that the associated host is up. Additionally, a method equivalent to SYN scanning the port 80 (or analog) is also used.

**Security Scanners**:

Security scanners are programs that automatically detect security weaknesses on remote or local hosts. As in the case of regular port scanners, they query TCP (UDP) ports and record the responses. However they go a step beyond the detection of running services and their owners. They try to additionally determine the following **[19]**:

- Whether anonymous logins are supported.
- Whether certain network services require authentication.
- The presence of known security holes.

Perhaps the best-known security scanner is **SATAN** (comprehensive lists of port and security scanners can be found in **[19]** and at **[20]**). When first released (April 1995) SATAN was considered the *ultimate* scanner, not only for being able to scan for a considerable amount of known holes, but also for its capacity to provide information about any discovered vulnerability, including the associated exploitation and the corresponding fix.

Security scanners, however, have been continuously evolving since SATAN and nowadays implementations are very complex and sophisticated **[21]**.

**Stack *Fingerprinting***:

Most security holes and vulnerabilities are Operating Systems dependent. So the possibility of remote OS detection should be one of the main concerns of system and/or security administrators.

Remote OS detection is not a new issue. For years, several implementations of TCP/IP services have been offering clues and information about their host's OS. FTP, TELNET, HTTP, and DNS servers are well known examples of this behavior. However the fact that the information is usually quite incomplete or even intentionally misleading, encouraged the apparition of some basic remote OS detectors. These first scanners worked by probing for well-known differences that existed among some selected operating systems TCP/IP implementations. Because of the small number of probes used and the limited amount of differences (***fingerprints***) analyzed, they were able to identify at most a dozen of OS.

Two recent scanners, however, **QueSO [22]** and specially **NMAP [9]**, introduced a new dimension in fingerprint scanning. QueSO was the first implementation that used a separate database for fingerprints (previously the fingerprints were included in the code). Nmap additionally incorporated an important amount of OS detection techniques and defined a *template* structure to describe fingerprints. Since any new fingerprint is easily added in the form of templates, NMAP fingerprints database is constantly increasing, and the number of Operating Systems that can be flawlessly identified has become very large.

The technique used by scanners to remotely detect OS is known as (TCP/IP) **stack *fingerprinting***.

In **[23] Commer** and **Lin** introduced a technique called ***active probing***. Active probing treats a TCP implementation as a *black box*, and uses a set of procedures to probe the black box. By studying the way TCP responds to the probes, several characteristics of the implementation can be deduced. TCP/IP stack *fingerprinting* is a variant of active probing that applies to the **whole** TCP/IP implementation (stack) of an OS. Stack fingerprinting uses several techniques (see bellow) to detect the presence of subtle differences in the TCP/IP stack of the OS of a scanned machine. This information is then used to create a fingerprint, which is compared with a collection of known fingerprints to identify the scanned system.

The stack fingerprinting methodology includes a considerable amount of techniques. The following is a list of the most relevant ones to date:

- The FIN Probe.
- The BOGUS Flag Probe.
- TCP ISN Sampling.
- Don't Fragment Bit.
- TCP Initial Window.
- ACK Value.
- ICMP Error Message Quenching.
- ICMP Message Quoting.
- ICMP Error Message Echoing Integrity.
- Type of Service.
- Fragmentation Handling.
- TCP Options.
- SYN Flood Resistance.

A detailed explanation of each technique can be found in **[24]**.

## References.


**[1]** W.R. Stevens, **TCP/IP Illustrated, Vol. 1**. Addison-Wesley, **1994**.

**[2]** W.R. Stevens, **TCP/IP Illustrated, Vol. 2**. Addison-Wesley, **1995**.

**[3]** Daemon9, **Project Neptune**. Phrack Magazine, Issue 48, **1996**.

**[4]** RFC 793, **TRANSMISSION CONTROL PROTOCOL**, **PROTOCOL SPECIFICATION**, pp. 64.

**[5]** L. Granquist, **Port 0 Scanning**, Bugtraq mailing list archives, 8 Jul **1998**.

**[6]** D. Comer, **Internetworking with TCP/IP Vol. 3**, Second Edition. Prentice Hall, **1996**.

**[7]** D. Atkins *et al*., **Internet Security**, Second Edition. New Riders, **1997**, pp. 413.

**[8]** Uriel Maimon, **Port Scanning without the SYN flag, TCP port Stealth Scanning**. Phrack Magazine, Issue 49, **1996**.

**[9]** **NMAP**, http://www.insecure.org/nmap/index.html, **1999**.

**[10]** S. Sanfilippo, **New TCP Scan Method**. Bugtraq mailing list archives, 18 Dec **1998**.

**[11]** M. de Vivo, G. de Vivo, G. Isern, **Internet Security Attacks at the Basic Levels**. Operating Systems Review, Vol. 32, No. 2. SIGOPS, ACM, April **1998**.

**[12]** SHADOW Indications Technical Analysis, **Coordinated Attacks and Probes**. Naval Surface Warfare Center Dahlgren Division, Code CD2S, http://www.nswc.navy.mil/ISSEC/CID/ Sep **1998**, Updated Dec **98** and Mar **99**.

**[13]** **Back Orifice**, http://www.cultdeadcow.com, **1999**.

**[14]** D. Comer, **Internetworking with TCP/IP Vol. 1**. Prentice Hall, Third Edition, **1995**, pp. 123-137.

**[15]** **IdentTCPscan**, www.asmodeus.com/archive/crack-scan/.

**[16]** RFC 1413, **Identification Protocol**.

**[17]** RFC 959, **FILE TRANSFER PROTOCOL (FTP)**.

**[18]** **The FTP Bounce Attack**, ftp://avian.org/random/ftp-attack.

**[19]** Anonymus, **Maximum Security**, Second Ed., SAMS, **1998**, pp. 174-202.

**[20]** **Index of /~tattooman/scanners**, At www.genocide2600.com/~tattooman/, follow the link to *FileArchives*, and then to *scanners*, **1999**.

**[21]** **ISS**, http://www.iss.net/, **1999**.

**[22]** **QueSO**, http://www.apostols.org/projectz/queso/

**[23]** D. Commer, J.C.Lin, **Probing TCP Implementations**. Department of Computer Sciences, Purdue University, **1994**.

**[24]** Fyodor, **Remote OS detection via TCP/IP Stack FingerPrinting**. Phrack Magazine, Volume 8, Issue 54, Dec 25th, **1998**. <fyodor@dhp.com>.