Examining port scan methods - Analysing Audible Techniques

whitepaper by dethy@synnergy.net

Abstract

I will attempt to enumerate a variety of ways to discover and map internal/external networks using signature-based packet replies and known protocol responses when scanning. Specifically, this document presents all known techniques used to determine open/closed ports on a host and ways an attacker may identify the network services running on arbitrary servers.

1.1 Introduction

This paper will provide an in-depth analysis of known port scan methods, with exhaustive information for each technique used in the wild today to map and identify open and closed ports on various network servers.

Note: This paper will not describe techniques used to fingerprint operating systems nor identify daemon versions (banner scanning).

With an epidemic of port scan instances occurring each and everyday, it should be recognized the ways an attacker could probe network hosts using a variety of techniques aimed to avoid detection whilst obscuring the sender's true source. Understanding actions to defend against these network oriented scans is first to identify and acknowledge the ways a scan can present appearing as normal inbound traffic.

Port scanning is one of the most popular techniques used in the wild to discover and map services that are listening on a specified port. Using this method an attacker can then create a list of potential weaknesses and vulnerabilities in

the proposed open port leading to exploitation and compromise of a remote host.

One of the primary stages in penetrating/auditing a remote host is to firstly compose a list of open ports, using one or more of the techniques described below. Once this has been established, the results will help an attacker identify various services that are running on that port using an RFC-compliant port list, (/etc/services in UNIX, getservbyport() function automatically obtains this) allowing further compromisation of the remote host after this initial discovery.

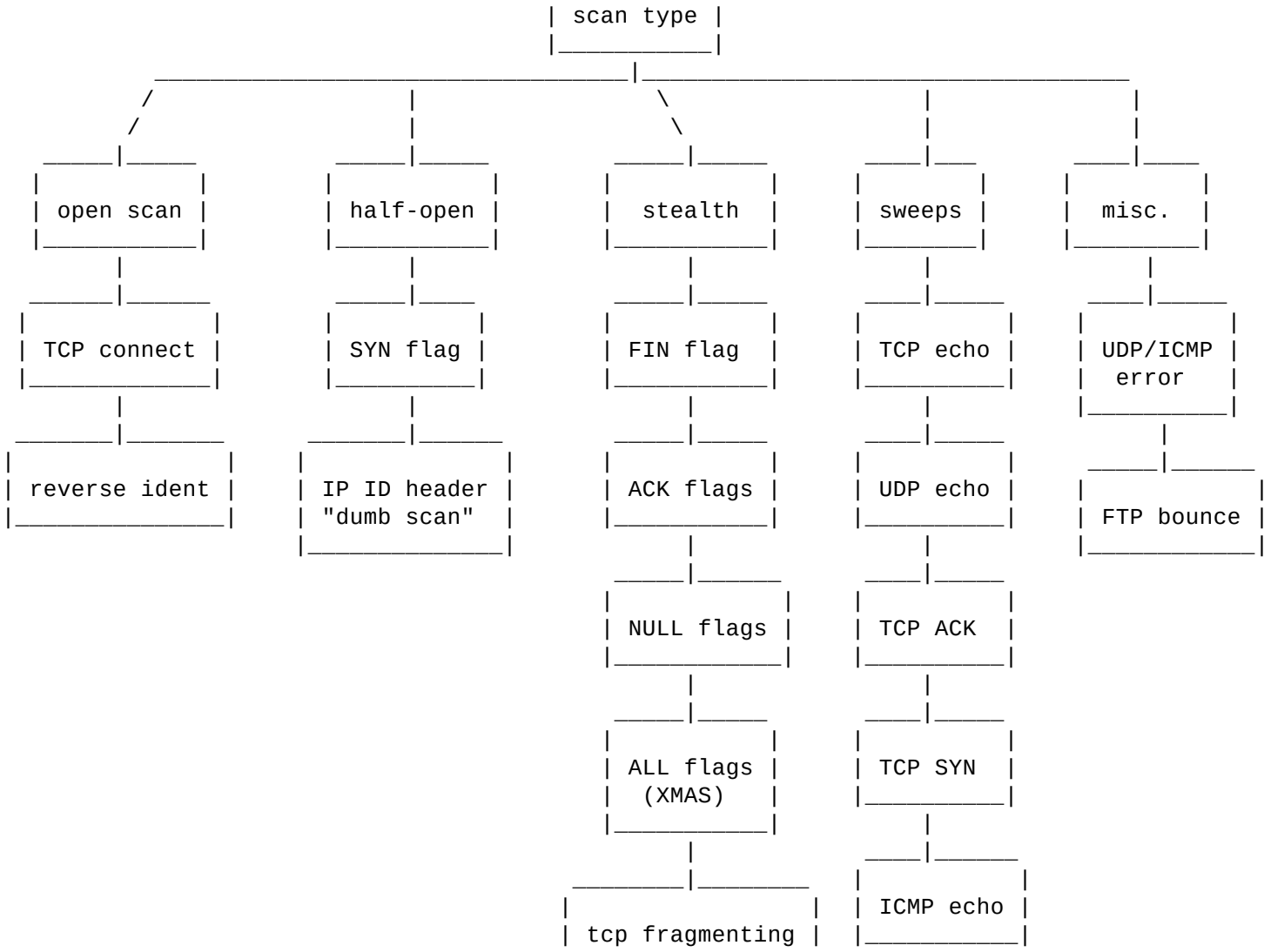
Port scanning techniques take form in three specific and differentiated ways.

* open scanning
* half-open scanning
* stealth scanning

Each of these techniques allow an attack to locate open/closed ports on a server, but knowing to use the correct scan in a given environment depends completely on the type of network topology, IDS, logging features a remote host has in place. Although open scans log heavily and are easily detectable they produce fairly positive results on open/closed ports.

Alternatively, using a stealth scan, may avoid certain IDS and bypass firewall rulesets but the scanning mechanism, such as packet flags, used in identifying these open/closed ports maybe offset by dropped packets over a network, leading to false positives. Further discussion of this concept takes place in the FIN scan section of this document.

Focusing more directly at each of the above techniques, these methods can be further categorised into individual scan types. Let's look at a basic scan model which includes PING sweeping:


scan type
open scan
half-open
stealth
sweeps
misc.
TCP connect
SYN flag
FIN flag
TCP echo
UDP/ICMP error
reverse ident
IP ID header "dumb scan"
ACK flags
UDP echo
FTP bounce
NULL flags
TCP ACK
ALL flags (XMAS)
TCP SYN
tcp fragmenting
ICMP echo

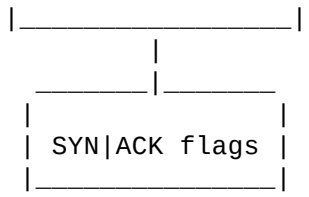


Diagram: known scan methods

The first nodes indicate the scan category which then traverses downward to list the individual scans for that class.

1.2 open scan methods

Open scanning techniques are blatantly easy to detect and to filter. This type of scan method involves opening a full connection to a remote host using a typical three-way TCP/IP handshake. A standard transaction involves issuing the following flags to create an accepted connection:

```
client -> SYN
server -> SYN|ACK
client -> ACK
```

The above example shows a port answering our initial connection request with a SYN|ACK. This response means the port the packet was targeted to is in the LISTENING (open) state. Once this full handshake has taken effect, the connection will be terminated by the client allowing a new socket to be created/called allowing the next port to be checked, until the maximum port threshold has been reached.

Reversely, taking a look at a response from a closed port would reveal the

following:

```
client -> SYN
server -> RST|ACK
client -> RST
```

The RST|ACK flags returned by the server is telling the client to tear down the connection attempt since the port is not in LISTENING state thus is closed.

This method is created through connect() system call, allowing almost instantaneous identification of an open or closed port. If the connect() call returns true, the port is open, else the port is closed.

Since this technique issues a three-way handshake to connect to an arbitrary host, a spoofed connection is impossible, that is to say a client can not manipulate the true source IP, as a spoofed connection attempt involves sending a correct sequence number as well as setting the correct return flags to setup for data transaction.

Obviously this technique is easily identifiable on any inbound traffic because it opens a full connection, thus all IDS and firewalls are able to detect and block against this scan. However, because the connect() method uses the three way handshake, results of this scan are about as accurate as you could get to determine open/closed ports.

Advantages : fast, accurate, requires no additional user privileges
Disadvantages: easily detectable and logged

1.2.1 - reverse ident scanning

This technique involves issuing a response to the ident/auth daemon, usually port 113 to query the service for the owner of the running process. The main reason behind this is to find daemons running as root, obviously this result

would entice an intruder to find a vulnerable overflow and instigate other suspicious activities involving this port. Alternatively, a daemon running as user nobody (httpd) may not be as attractive to a user because of limited access privileges. Unknowing to most users is that identd could release miscellaneous private information such as:

 * user info
 * entities
 * objects
 * processes

Although the identification protocol would appear as an authentication mechanism, it was not designed or intended for this purpose. As the RFC states, "At best, it provides some additional auditing information with respect to TCP connections". Needless to say, it should not be used as an access control service nor relied upon added host/username authenticity.

The formal syntax taken from RFC 1413 reveals the following EBNF:

FORMAL SYNTAX

```
   <request> ::= <port-pair> <EOL>

   <port-pair> ::= <integer> "," <integer>

   <EOL> ::= "015 012"  ; CR-LF End of Line Indicator, octal \r\n equivalents

   <integer> ::= 1*5<digit> ; 1-5 digits.
```

Using this grammar applied to the data we send to an arbitrary host piped to the ident/auth port will reveal the process owner running on a given port, even though we initiated the connection.

Advantages   : fast, requires no additional privileges, return vital service information
Disadvantages: easily detactable

1.3 - half open scan methods

The term 'half-open' applies to the way the client terminates the connection before the three-way handshake is completed. As such, this scan method will often go unlogged by connection based IDS', and will return fairly positive results (reliability of open/closed port recognition).

1.3.1 - SYN scanning

The implementation of this scan method is similar to a full TCP connect() three way handshake except instead of sending ACK responses we immediately tear down the connection. A demonstration of this technique is necessary to show a half open transaction:

```
client -> SYN
server -> SYN|ACK
client -> RST
```

This example has shown the target port was open, since the server responded with SYN|ACK flags. The RST bit is kernel oriented, that is, the client need not send another packet with this bit, since the kernel's TCP/IP stack code automates this.

Inversely, a closed port will respond with RST|ACK.

```
client -> SYN
server -> RST|ACK
```

As is displayed, this combination of flags is indicative of a non- listening port.

Although, this technique has become rather easy to detect by many IDS, owing to the fact that a paramount of Denial of Service (DoS) utilities base their attacks by sending excess SYN packets. Fairly standard intrusion detection systems are no doubt capable of logging these half-open scans: TCP wrappers, SNORT, Courtney, iplog, to a name a few, thus the effectiveness has dithered over recent years.

Advantages : fast, reliable, avoids basic IDS, avoids TCP three-way handshake
Disadvantages: require root privileges, rulesets block many SYN scan attempts

1.3.2 - IP ID header aka "dumb" scanning

ID header scanning is a rather obscure scan method involving implementation peculiarities in the TCP/IP stack of most operating systems. Originally this technique was discovered by antirez, who described it's technical details in a post to bugtraq. Evidently the basis of this scans implementation is reflective on the SYN scan method, although involves a third party host to use as a dummy source.

Before explaining any further it's important to recognize what a so- called "dumb" host is. Contrasting to a bastion host, a silent or dumb host is a server that sends and receives little to no traffic at all, hence the characteristic name endowed upon it. Locating one of these hosts requires much effort and host sweeping itself, and is probably more trouble than what it is worth. Nevertheless, it is a genuine and creative scan, that brings a thirdhost into play adding to it's obscurity.

Involved in this scenario are three hosts:

```
        * A -> attackers host
        * B -> dumb host
        * C -> target host

Let's examine this cycle.

 * Host A sends a series of PING's analysing the ID field, encapsulated within
   the IP header to Host B. A dumb host will have the ID increment the reply by
   1 each time during the PING sequence.


   60 bytes from BBB.BBB.BBB.BBB: seq=1 ttl=64 id=+1 win=0 time=96 ms
   60 bytes from BBB.BBB.BBB.BBB: seq=2 ttl=64 id=+1 win=0 time=88 ms
   60 bytes from BBB.BBB.BBB.BBB: seq=3 ttl=64 id=+1 win=0 time=92 ms


 * Host A sends a spoofed SYN packet to Host C using the source address of Host B.
   The remote port is any arbitrary port (1-65535) that the attacker wishes to test
   for open/closed responses. Host C will reply to Host B with one of two standard
   responses:

   -> SYN|ACK response indicates an open LISTENING port. Host B will then reply with
      an RST bit flagged in the packet (automated by kernel).
   -> RST|ACK will indicate a NON-LISTENING port, (a standard SYN scan method reply),
      and Host B will ignore that packet and send nothing in reply.


Now, how could Host A know what flags were sent to Host B ?

Well, assuming the port  was open on the  target server, our series  of parallel
PING's that Host A  had been sending whilst  the spoofed SYN packets  were being
sent will hold our answers.
Analyzing the ID field  in these PING responses,   one would notice a  higher ID
increment.
```

```
60 bytes from BBB.BBB.BBB.BBB: seq=25 ttl=64 id=+1 win=0 time=92 ms
60 bytes from BBB.BBB.BBB.BBB: seq=26 ttl=64 id=+3 win=0 time=80 ms
60 bytes from BBB.BBB.BBB.BBB: seq=27 ttl=64 id=+2 win=0 time=83 ms
```

Notice the second and third packets ID responses contain values greater than 1, hence an open port was located. Any further increment of more than 1 is indicative of an open port in Host B's responses, during this period.

Originally, the increment was 1, but because Host A sent a spoofed SYN to an open port, Host B had to reply to Host C with the SYN|ACK bit packet, thus incrementing the ID field. Following this the PING response to Host A would then in turn have a higher ID field, as suspected.

On the other hand, a closed port state on Host C would not require Host B to send anything, so the ID field in the PING response would not be incremented at all.

```
60 bytes from BBB.BBB.BBB.BBB: seq=30 ttl=64 id=+1 win=0 time=90 ms
60 bytes from BBB.BBB.BBB.BBB: seq=31 ttl=64 id=+1 win=0 time=88 ms
60 bytes from BBB.BBB.BBB.BBB: seq=32 ttl=64 id=+1 win=0 time=87 ms
```

As is shown, the ID field is still bounded by a constant of 1.

Once again this is why a "dumb" host is required, so incoming and outgoing traffic is kept at a bare minimum in order to decrease false- positive results.

In fact, a variety of scan methods could be used involving a dumb host. This scan is not limited to the SYN scan technique. Any method involving Host B to respond to Host A's port reply could be practiced (hint: inverse mapping techniques).

1.4 - stealth scanning

The definition of a "stealth" scan has varied over recent years from what Chris Klaus, author of a paper titled "Stealth Scanning: Bypassing Firewalls/SATAN Detectors" delineated. Originally the term was used to describe a technique that avoided IDS and logging, now know as "half-open" scanning. However, nowadays stealth is considered to be any scan that is concerned with a few of the following:

* setting individual flags (ACK, FIN, RST, ..)
* NULL flags set
* All flags set
* bypassing filters, firewalls, routers
* appearing as casual network traffic
* varied packet dispersal rates

All scan the scans described below use the inverse mapping technique for open port assumptions.

1.4.1 - SYN|ACK scanning

This technique has been disregarded in most, if not all, port scanners to date. Ironically, the theory behind this method is not unlike the SYN method, we cut out the first step in our half-open TCP/IP setup. A standard response would act as follows:

```
client -> SYN|ACK
server -> RST
```

The above flags have denoted to the client that the port is in a non- listening

state. Since the transmission control protocol realizes that no initial SYN was sent, an immediate termination response was sent out. In other words, the protocol thinks there has been an error in the connection transaction to that port when a SYN|ACK has been received without a SYN, as a result the reset flag is sent back.

On the other hand a LISTENING port will not respond to these flags.

```
client -> SYN|ACK
server -> -
```

As is seen, the server ignores the SYN|ACK packet sent to an open port. Needless to say the absence of the server's response packet to ours, will produce false positives. Imagine sending a SYN|ACK packet and receiving no response due to stately packet filters, firewalls or even timeout limits blocking transmission, thus the scanner would then produce false positives for that port. Naturally this scan is not considered as reliable as TCP connect() scans because of this very reason. This type of assumption falls under what is known as "inverse mapping".

Advantages : fast, avoids basic IDS/firewalls, avoids TCP three-way handshake
Disadvantages: less reliable (false positives)

1.4.2 - FIN scanning

The FIN scan method uses inverse mapping to discover closed ports. Unfortunately, this techniques relies on bad BSD network code which most operating systems have based their TCP/IP stacks on (all the better for scanning). Ideally, once a FIN flagged packet is sent, a closed port will resend with an RST bit. Open ports, alternatively will not send a packet back, therefore what precisely is not answered with the FIN bit, is assumed to be open

through this process of inverse mapping.

Take a look at the negotiation for open/closed port recognition displayed below.

```
client -> FIN
server -> -
```

No reply signaled by the server is iconic of an open port. The server's operating system silently dropped the incoming FIN packet to the service running on that port. Opposing this is the RST reply by the server upon a closed port reached. Since, no service is bound on that port, issuing a FIN invokes a reset (RST) response from the server.

```
client -> FIN
server -> RST
```

Arguably there are two ways to test for an open port. The first is receiving a list of closed port responses and subtracting these port replies from a list of the port probes originally sent. For example, sending 3 packets to ports 1, 2, 3 on a remote host.

If the response back is an RST for ports 1 and 3, we then compare the original port list: 1, 2, 3 to the received ports: 1, 3 and deduce that 2 is the open port via comparison.

The second test involves using a timeout for the packet response. If the timeout limit is reached to receive the packet in question then we assume it to be open. Obviously, this method is test for false positives and should be avoided where possible. The responses for the packet could be obscured because of firewalls, filters, routers, slow links, and heavy traffic, thus is not a solid test to be used as a rule of thumb for stealth FIN scanning.

Advantages   : avoids many IDS, avoids TCP three-way handshake
Disadvantages: slow false positives

1.4.3 - ACK scanning

Uriel Maimon first described this technique in Phrack 49 article 15. Needless to say this technique revolves around a bug in the IP layer of a few operating systems.

In order to test for an open port using this method an initial ACK packet is sent to the target host. There are actually two ways to classify the response packet. The first involves an assessment of the TTL field, the second is analyzing the WINDOW field. Both of these fields should be obtained with the response packet that has the RST bit set.

The reply should be a reset connection, that is, a packet with the RST bit set. Accompanying the RST flag, an analysis of the IP header, for some operating systems, will provide a TTL that is lower than the other packets received from a closed port. Evidently any TTL sent to an open port would reveal a TTL less than or equal to 64, if the upper/lower ports have a higher TTL.

```
client -> ACK
server -> RST -> (TTL <= 64)
```

A real life response is show below:

```
packet 1: host XXX.XXX.XXX.XXX port 20: F:RST -> ttl: 70 win: 0 => closed
packet 2: host XXX.XXX.XXX.XXX port 21: F:RST -> ttl: 70 win: 0 => closed
packet 3: host XXX.XXX.XXX.XXX port 22: F:RST -> ttl: 40 win: 0 => open
```

```
 packet 4: host XXX.XXX.XXX.XXX port 23: F:RST -> ttl: 70 win: 0 => closed
```

Notice the TTL of the sequential packets before and after packet 3 is higher than 64. As packet 3 is received it is observed that the TTL for port 22 is less than the boundary 64, indicating an open port.

Using the WINDOW field method, any non-zero response packet received from the server is indicative of an open port. This is true for several early BSD (FreeBSD, OpenBSD) and UNIX (AIX, DGUX) but has been patched/fixed in more recent versions.

```
client -> ACK
server -> RST -> WINDOW (non-zero)
```

A real life response is shown below:

```
 packet 6: host XXX.XXX.XXX.XXX port 20: F:RST -> ttl: 64 win: 0 => closed
 packet 7: host XXX.XXX.XXX.XXX port 21: F:RST -> ttl: 64 win: 0 => closed
 packet 8: host XXX.XXX.XXX.XXX port 22: F:RST -> ttl: 64 win: 512 => open
 packet 9: host XXX.XXX.XXX.XXX port 23: F:RST -> ttl: 64 win: 0 => closed
```

Notice that although the TTL equals 64, the surrounding packets do also. Thus the TTL method would not work on this host, however the WINDOW offset method shows a non-zero value indicative of an open port.

Advantages : difficult to log, avoids IDS detection
Disadvantages: relies on BSD network code bug, OS incompatible

1.4.4 - NULL scanning

Clearly through it's endowed named, the NULL scan unsets ALL flags available in the TCP header. ACK, FIN, RST, SYN, URG, PSH all become unassigned. The reserved bits (RES1, RES2) actually do not effect the result of any scan, whether or not they are set clearly does not matter. On arrival of this packet to the server, BSD networking code informs the kernel to drop the incoming call if the port is open.

```
client -> NULL (no flags)
server -> -
```

Alternatively, an RST packet will be returned if a closed port has been reached (yes another inverse mapped scan).

```
client -> NULL (no flags)
server -> RST
```

Owing to the fact that the RFC does not exclaim exactly how a host should respond to these types of packets, various network code for the major operating systems will differ in the packet responses, ie Microsoft vs UNIX.

Advantages : avoids IDS, avoids TCP three-way handshake
Disadvantages: UNIX only, false positives

1.4.5 - XMAS scanning

Contrastedly, a so called XMAS scan is the inverse of the NULL scan method. All the available flags in the TCP header are set (ASK, FIN, RST, SYN, URG, PSH). XMAS or "Christmas Tree" scanning is named rightly so after the decorative effect the scan has with the flagging implementation. The reserved bits do not

effect the scan result, so setting or unsetting is ofno importance. Once again, since this method is based on BSD networking code the technique will only work against UNIX hosts.

XMAS scanning works by initializing all the flags and transmitting this packet to the remote host. The kernel will drop the packet if an open port is at the receiving end. A returned RST flag will reflect a closed, NON-LISTENING port again this is an inverse mapped scan, so false positives is all a client has to detect an open/closed port.

```
client -> XMAS (all flags)
server -> -
```

This signature tells us that the port is in LISTENING state, or the packet was filtered by a firewall/router. Alternatively a closed port will produce the following reply:

```
client -> XMAS (all flags)
server -> RST
```

The RST would be sent to the client because the server is tricked into thinking that the client has a connection on that port without negotiating with the standard three-way handshake. Since TCP is stateful the kernel sends a reset bit (RST) back to the client to end transmission immediately.

Advantages : avoids IDS, avoid TCP three-way handshake
Disadvantages: UNIX only, false positives

1.4.6 - TCP Fragmenting

TCP fragmenting is not a scan method so to speak, although it employs a method to obscure scanning implementations by splitting the TCP header into smaller fragments. IP reassembly on the server-side can often lead to unpredictable and abnormal results (IP headers carrying data can be fragmented). Many hosts are unable to parse and reassemble the tiny packets and thus may cause crashes, reboots, or even network device monitoring dumps. Alternatively, these tiny packets may be potentially blocked by IP fragmentation queues in the kernel or caught by a stately firewall ruleset.

Since many intrusion detection systems use signature-based mechanisms to signify scanning attempts based on IP and/or the TCP header, fragmentation is often able to defeat this type of packet filtering and detection, and naturally the scan will go undiscovered.

A minimally allowable fragmented TCP header must contain a destination and source port for the first packet (8 octect, 64 bit), typically the initialized flags in the next, allowing the remote host to reassemble the packet upon arrival. The actual reassembly is established through an IPM (internet protocol module) that identifies the fragmented packets by the field equivalent values of:

* source
* destination
* protocol
* identification

Advantages : avoids IDS, stealth
Disadvantages: may cause network problems on remote host

1.5 Miscellaneous

This category represents scans that can not be entirely classified into the broader open/half-open/stealth classes. The scans here are dissimilar in nature

but are techniques still used in the wild today.

1.5.1 - UDP ICMP_PORT_UNREACHABLE scanning

Unlike the above scanning methods, the User Datagram Protocol (UDP) is used to determining open/closed ports on a remote host rather than TCP.

UDP is a connectionless stream protocol that sends datagrams as a means of packet transmission. Similarly to the inverse mapping system, sending a UDP packet to an open port will receive no response from a server. However, a closed port will respond with an Internet Control Message Protocol (ICMP) error reply. Using a process of extrapolation it is simple to identify the open from closed ports. The message type, ICMP_PORT_UNREACH (type 3 code 3), does not technically need to be sent when a closed port received a UDP packet, hence the difficulty with this scanning method. Additionally, UDP is known to be an unreliable protocol since packets are easily dropped during transmission, hence retransmission needs to take place, otherwise even more false positives are assumed in the scan result. Linux kernels limit ICMP error message rates, destination unreachable are set to 80 per 4 seconds with 1/4 second penalty if that is exceeded, adding to the scanning technicality, as Fyodor pointed out.

An open port signature should send no reply, also a retransmitted packet is sent to reduce false positives:

```
client -> udp packet
server -> -
client -> udp packet
server -> -
```

Closed ports will response with the ICMP error.

```
                        client -> udp packet
                        server -> UDP (ICMP_PORT_UNREACH)


Advantages   : scans non-TCP ports, avoids TCP IDS
Disadvantages: requires root, packets easily dropped, easily detected
```

1.5.2 FTP server bounce attack

This ingenious method was described in a paper by the hobbit. Using, the FTP PORT command to set a clients passive mode, a host is able to determine the status of a port by issuing an IP and port as arbitrary parameters to connect to. If a connection is established as a means of active data transfer processing (DTP), the client knows a port is open, with a 150 and 226 response issued by the server. If the transfer fails a 425 error will be generated with a refused build data message.

Early versions of WU-FTPD (less than 16) were vulnerable to this type of attack, nowadays the presence of this bug has been patched in most FTPD's. Other vulnerable versions include:

Sun FTP server in SunOS 4.1.x/5.x, SCO OpenServer 5.0.4, SCO UnixWare 2.1, AIX 3.2/4.2/4.2./4.3, Caldera 1.2, RedHat 4.X, Slackware 3.1 - 3.3.

An easy way to disallow this kind of attack is to prevent third party transfers through modification of the PORT command and/or disallowing specification of reserved ports, except port 20 the standard default data port.

```
Advantages   : bypass firewalls, allows access to local nets, hard to trace
Disadvantages: slow, most FTPD's have been patched
```

1.6 Blocking packet anomalies

Isolating and filtering the packets used in all the above scans is one step forward into securing any inter-network connected node. Any application of the following rulesets will yield many port scanning techniques with false positive information, highlighting the well known "security through obscurity" objective.

* block unassigned port traffic (traffic to ports with unassigned services)
* application-layer monitoring
* deny pass-through traffic
* monitor transport-layer connections (control of TCP, SYN, RST, ACK)
* monitor source address matching well known addresses
* filter ICMP type 3 and 8
* active network monitoring

Many audible scanning techniques exist to gather information about the services that exist on a host. However, none of these techniques will evade a well configured proxy along with an active systems analyst to spot potential traffic abnormalities.

References

Art of portscanning by Fyodor - http://www.phrack.com
Networking Scanning by Ofir Afkin - http://www.sys-security.com
FTP bounce attack by hobbit - http://www.insecure.org/nmap/hobbit.ftpbounce.txt

---