STOPPING SCANNING - ENABLING FOG-OF-WAR

We have previously looked at the practice of port scanning[1], well this time we will be looking at how to mount a defense against port scanning. This type of defense is a good idea because it makes it that much more difficult for an attacker to determine what type of systems you are running, which means that they they will be less able to customize the attacks or target specific vulnerabilities. Basically, anything that makes the attackers job more difficult is a good idea.

First lets look at the test network used. There is a linux iptables[2] firewall functioning as a gateway and router, on one segment is a 10.0.0.0/24 network and on the other segment is a 192.168.10.0/24 network. The countermeasures I will be discussing will be related to the iptables firewall and other linux utilities, but these countermeasures should be able to be implemented on most decent firewalls and security devices

Full TCP Connect Scan
Lets first take a look at the nmap[3] command used (I am only scanning a single port and have switched off the initial check to determine if the host is up so that any packet captures only show the scan attempts)...

```
[root@localhost root]# nmap -sT -P0 -p 21 10.0.0.50

Starting nmap 3.48 ( http://www.insecure.org/nmap/ )
Interesting ports on 10.0.0.50:
PORT   STATE SERVICE
21/tcp open  ftp

Nmap run completed -- 1 IP address (1 host up) scanned in 0.013 seconds
```

To capture the scan packets I ran tethereal (the command line version of the ethereal[4] package), so what did the scanned machine see...

```
192.168.10.80 -> 10.0.0.50    TCP 32875 > ftp [SYN] Seq=838535498 Ack=0 Win=5840 Len=0
   10.0.0.50 -> 192.168.10.80 TCP ftp > 32875 [SYN, ACK] Seq=4073725542 Ack=838535499 Win=32120 Len=0
192.168.10.80 -> 10.0.0.50    TCP 32875 > ftp [ACK] Seq=838535499 Ack=4073725543 Win=5840 Len=0
192.168.10.80 -> 10.0.0.50    TCP 32875 > ftp [RST, ACK] Seq=838535499 Ack=4073725543 Win=5840 Len=0
```

SYN Scan
The nmap command used was...

```
[root@localhost root]# nmap -sS -P0 -p 21 10.0.0.50

Starting nmap 3.48 ( http://www.insecure.org/nmap/ )
Interesting ports on 10.0.0.50:
PORT   STATE SERVICE
21/tcp open  ftp

Nmap run completed -- 1 IP address (1 host up) scanned in 0.013 seconds
```

The target machine saw these packets..

```
192.168.10.80 -> 10.0.0.50    TCP 60605 > ftp [SYN] Seq=2118392454 Ack=0 Win=1024 Len=0
   10.0.0.50 -> 192.168.10.80 TCP ftp > 60605 [SYN, ACK] Seq=4164665868 Ack=2118392455 Win=32696 Len=0
192.168.10.80 -> 10.0.0.50    TCP 60605 > ftp [RST] Seq=2118392455 Ack=0 Win=0 Len=0
```

FIN Scan
The nmap command used was...

```
[root@localhost root]# nmap -sF -P0 -p 21 10.0.0.50

Starting nmap 3.48 ( http://www.insecure.org/nmap/ )
Interesting ports on 10.0.0.50:
PORT   STATE SERVICE
21/tcp open  ftp

Nmap run completed -- 1 IP address (1 host up) scanned in 12.027 seconds
```

The target machine saw these packets...

```
192.168.10.80 -> 10.0.0.50    TCP 54384 > ftp [FIN] Seq=0 Ack=0 Win=1024 Len=0
192.168.10.80 -> 10.0.0.50    TCP 54385 > ftp [FIN] Seq=0 Ack=0 Win=4096 Len=0
```

XMAS Scan
The nmap command used was...

```
[root@localhost root]# nmap -sX -P0 -p 21 10.0.0.50

Starting nmap 3.48 ( http://www.insecure.org/nmap/ )
Interesting ports on 10.0.0.50:
PORT   STATE SERVICE
21/tcp open  ftp

Nmap run completed -- 1 IP address (1 host up) scanned in 12.029 seconds
```

The target machine saw these packets...

```
192.168.10.80 -> 10.0.0.50    TCP 44307 > ftp [FIN, PSH, URG] Seq=0 Ack=0 Win=4096 Urg=0 Len=0
192.168.10.80 -> 10.0.0.50    TCP 44308 > ftp [FIN, PSH, URG] Seq=0 Ack=0 Win=3072 Urg=0 Len=0
```

NULL Scan
The nmap command used was...

```
[root@localhost root]# nmap -sN -P0 -p 21 10.0.0.50

Starting nmap 3.48 ( http://www.insecure.org/nmap/ ) at 2005-03-08 20:18 SAST
Interesting ports on 10.0.0.50:
PORT   STATE SERVICE
21/tcp open  ftp

Nmap run completed -- 1 IP address (1 host up) scanned in 12.026 seconds
```

The target machine saw these packets...

```
192.168.10.80 -> 10.0.0.50    TCP 52277 > ftp [] Seq=0 Ack=0 Win=3072 Len=0
192.168.10.80 -> 10.0.0.50    TCP 52278 > ftp [] Seq=0 Ack=0 Win=3072 Len=0
```

Plugging the Holes
Now lets try to stop these scans. You will notice that the NULL, XMAS, and FIN scans use crafted packets which are not part of valid normal TCP handshake, so these we can stop by using the stateful packet inspection features of the iptables firewall. Before we go any further you need to understand how a forwarding iptables firewall handles packets. All packets that the firewall forwards travel the following tables in this order:

```
PREROUTING -> FORWARD -> POSTROUTING
```

What we are going to do is add the following options to the first rule used by an allowed protocol in the prerouting table (in my example I am adding it to the FTP rule)...

```
-p tcp --syn -m state --state NEW -j ACCEPT
```

so that the full first rule looks like...

```
iptables -t nat -A PREROUTING -p tcp --syn -p tcp --dport 21 -m state --state NEW -j ACCEPT
```

What the additions do is tell the firewall to also check the traffic so that it only allows SYN packets which are the start of a NEW connection attempt. We are also going to check for other data packets with bad flags in the FORWARD table, just to further make sure that only proper packets are allowed through the firewall.

```
iptables -A FORWARD -p icmp -j DROP
iptables -A FORWARD -p tcp --tcp-flags ALL NONE -j DROP
iptables -A FORWARD -p tcp --tcp-flags ACK,FIN FIN -j DROP
iptables -A FORWARD -p tcp --tcp-flags ACK,PSH PSH -j DROP
iptables -A FORWARD -p tcp --tcp-flags ACK,URG URG -j DROP
iptables -A FORWARD -p tcp --tcp-flags FIN,RST FIN,RST -j DROP
iptables -A FORWARD -p tcp --tcp-flags SYN,FIN SYN,FIN -j DROP
iptables -A FORWARD -p tcp --tcp-flags SYN,RST SYN,RST -j DROP
iptables -A FORWARD -p tcp --tcp-flags ALL ALL -j DROP
iptables -A FORWARD -p tcp --tcp-flags ALL FIN,PSH,URG -j DROP
iptables -A FORWARD -p tcp --tcp-flags ALL SYN,FIN,PSH,URG -j DROP
iptables -A FORWARD -p tcp --tcp-flags ALL SYN,RST,ACK,FIN,URG -j DROP
```

Once this has been done, all XMAS, FIN and NULL scans will no longer work. But what about SYN and Full TCP connection scans? Well those are a trifle more difficult to deal with because of the way they work. Remeber that a proper TCP connection uses a three-way handshake, and that is exactly how a Full TCP connection scan works, each port is scanned using a full three-way handshake connection, so to the firewall -even while using stateful packet inspection- each port scan is a legal connection. This is also why a SYN scan is difficult to counter, because the first two packets also look exactly like the start of a normal three-way handshake connection, it just never completes the process. But despair not, there are one or two tricks we can use to make an attackers life more difficult, as well as to keep an eye on things.

Slowing It Down

You see, a port scan generally has to scan a large number of ports in a short time so what we can do is to tell our firewall to limit the possible number of connections which it can deal with in a given period of time. We will add the following switches to our allowed traffic rules in the PREOUTING section of the iptables firewall as the first rule...

```
-m limit --limit 10/s --limit-burst 15
```

so that the full ruleset looks like...

```
iptables -t nat -N TCP_CHK
iptables -t nat -A TCP_CHK -p tcp --syn -m state --state NEW -m limit --limit 10/s --limit-burst 15 -j ACCEPT
iptables -t nat -A TCP_CHK -j DROP
iptables -t nat -A PREROUTING -p tcp -j TCP_CHK
```

What this ruleset does is tell the firewall to create a new table called TCP_CHK, then the PREROUTING table rule passes all tcp traffic to this new table. In TCP_CHK, all tcp traffic is checked to ensure that it is a new connection, and then it ensures that a maximum of 15 new connections are allowed a second and if that limit is exceeded, then only 10 new connections a second are allowed until the rule recharges. All excess traffic is dropped. This ruleset also has the added benefit of protecting against SYN FLOOD DOS attacks. Bear in mind that the two figures I use as limits (10 and 15) work for me, but may need to be tweaked for your particular environment. My advice if you're implementing this, is to keep a eye on things after you start using the ruleset and see if you should push the limits up or down. The observant reader will also note that I am using the DROP rather than REJECT, and this is because when you use REJECT the firewall sends out a reject packet to the scanner which nmap will use for information gathering. So just use DROP to really make an attackers life difficult.

Monitoring Things
Well, we've slowed down SYN and Full TCP connection scans, but we still want to track them. That's where an IDS comes in. If you have an Intrusion Detection System, then you can use that to check for suspicious traffic. On the test network's firewall I ran Snort[5] (an opensource IDS which runs on linux and windows), and this is what it saw...

Scan Type	*Snort Log File*	*Monitored Event*
Full TCP Connection Scan	alert	portscan status from 192.168.10.80
SYN Scan	portscan.log	192.168.10.80:36615 -> 10.0.0.50:21 SYN ******S*
FIN Scan	portscan.log	192.168.10.80:59047 -> 10.0.0.50:21 FIN *******F
XMAS Scan	portscan.log	192.168.10.80:60003 -> 10.0.0.50:21 XMAS **U*P**F
NULL Scan	portscan.log	192.168.10.80:61837 -> 10.0.0.50:1600 NULL ********

Using Snort will allow your firewall to log suspicious traffic for you to investigate, no matter whether the traffic is using valid packet flags and packet types or not.

Well, that brings us to the end of this topic. I hope you have seen how with a few simple rules to help ensure proper network traffic, you can make any attackers port scanning activities a painful exercise ... for them. Also how you can utilise other tools to further add security to your firewall. Anyway, as always have fun, play around with it and work at being secure.

---

[1] Port Scanning - A General Primer
[2] Iptables homepage
[3] Nmap homepage
[4] Ethereal homepage
[5] Snort homepage