GUEST FEATURE

Low-Level Enumeration With TCP/IP

by Randy Williams <dolph -at- woh.rr.com>
Dec 19 2003 10:41PM GMT

We've all used most of the popular stealth scanning techniques out there right now. Tools such as nmap are excellent for enumerating remote hosts with increasingly complex techniques. The only problem being most of the nmap users out there do not take the time to find out exactly what is going on behind the scenes to make these scans work. In the following paragraphs I will attempt to explain the theory and concept behind many of today's advanced scanning techniques, and try to show you what is going on behind the scenes with them.

Syn Scanning

I know. Most of you are thinking syn scans are just about on the bottom of scans in a technical sense, and indeed you are correct. This section is being offered in this text as a step for beginners to be able to better comprehend some of the later subjects of the article.

Syn scanning, a technique that emerged into the hacking community several years ago, is still widely in use across the internet today. While it has been replaced by more stealthy and firewall negating scans, it is still the best balance of stealth and speed. The syn scan, also called the "half open" scan, is the ability to determine a ports state without making a full connection to the host. As such, the majority of systems do not log the attempt, and discard it as a communications error. To understand how this vulnerability is exploited by scanning software, you must first understand the concept of the "three way handshake" technique that is used to establish all tcp connections between systems.

Standard tcp communications are controlled by flags in the tcp packet header. These flags control what happens to the connection between the hosts, and helps give order to the system. The flags are as follows:

Synchronize - also called "SYN", used to initiate a connection between hosts.
Acknowledgement - also called "ACK", used in establishing a connection between hosts
Push - "PSH", Instructs receiving system to send all buffered data immediately
Urgent - "URG", states that the data contained in the packet should be processed asap
Finish - also called "FIN" tells remote system that there will be no more transmissions
Reset - also called "RST", also used to reset a connection.

As far as syn scanning is concerned, we are only worried about three of these flags, syn, ack, and rst. Tcp is a connection based protocol, which means before any application later data can be exchanged, a connection must first be established. This connection is established with what is called a "three way handshake". The three way handshake is illustrated below:

```
192.168.1.2:2342 ------------------syn----------------------> 192.168.1.3:80
192.168.1.2:2342 <-----------------syn/ack--------------------192.168.1.3:80
192.168.1.2:2342------------------ack------------------->192.168.1.3:80
                            Connection Established
```

The above diagram shows what a connection initiation between a host and a http server would look like. The initiating host (192.168.1.2) initiates a connection to the server (192.168.1.3) via a packet with only the syn flag set. The server, then replies with a packet with both the syn and the ack flag set. For the final step, the client responds back the server with a single ack packet. If these three steps are completed without complication, then a tcp connection has been established between the client and server.

So how is the three way handshake exploited by stealth scans? Simple, you simply don't complete the third step.

What is this?

12/19/2003
Low-Level Enumeration With TCP/IP

12/04/2003
Simulating and optimising worm propagation algorithms (PDF)

12/04/2003
The Rise of the Spammers

10/16/2003
The Flaw of Security Through Diversification

10/10/2003
Counterpoint: Linux vs. Windows Viruses

05/30/2003
Untrustworthy Passport

01/27/2003
DNS Cache Poisoning - The Next Generation

10/18/2002
Ten Things to Do With IIS

10/10/2002
An Introduction to Computer Forensic Tools

09/20/2002
Hackback or the High Road? The question goes beyond Nimda

[more...]

By examining each packet as it enters the interface, you can guess the remote port's state, and terminate the connection before it is even established. Take the example below:

```
192.168.1.2:2342 -------------------------syn-------------------->192.168.1.3:80
192.168.1.2:2342<---------------------syn/ack-----------------192.168.1.3:80
192.168.1.2:2342-------------------------rst-------------------->192.168.1.3:80
```

Those of you with little tcp/ip experience have no clue what the point of that was. Just hang with me and you'll begin to see. The initiation begins the same as in the previous example, the client sends a single syn packet to the server on the appropriate port. It is in the server's response that shows the idea behind stealth scanning. If the server responds with a syn/ack packet, we can very safely assume that the port is in state "open". If the server responds with an rst packet, then the remote port is in state "closed" As seen in the example, the server did indeed respond with the syn/ack packet, knowing that this means the port is open, we then respond with a rst packet, to close the initiation before a connection can ever be established.

This scan type also comes with a risk. If a host is scanned to long, it is possible to "syn flood" the remote host. This is a result of the scan filling up the servers connection table, causing it to stop responding to legitimate traffic. However, this side effect has been remedied in modern version of the majority of operating systems in use. The developers simply coded the kernel to limit the amount of connection attempts that could be made from one ip address. While this eliminates the danger of inadvertently disabling the host, is does hold another hazard for the scanner. If you were to scan to many ports on a given system, you will eventually reach the threshold of half open connections allowed on the host, consequently ending your scan. To deal with this problems, the hacking community simply began limiting the scan to only ports that will be relevant for their attack, instead of the large range of ports that would trigger the os's ip filtering methods.

Xmas, FIN, and NULL scanning

While Xmas, FIN, and NULL scans are among the most undetected by intrusion detection systems, they are also among the most unreliable. Unreliable in the fact that a multitude of different conditions could result in the scanning software reading the port as open, when it is in fact not. Illustrations for the Xmas scan are shown below:

Xmas scan directed at open port:

```
192.5.5.92:4031 ------------------------FIN/URG/PSH---------------->192.5.5.110:23
192.5.5.92:4031 <---------------------NO RESPONSE---------------192.5.5.110:23
```

Xmas scan directed at closed port:

```
192.5.5.92:4031 ----------------------FIN/URG/PSH---------------->192.5.5.110:23
192.5.5.92:4031<--------------------------RST/ACK-----------------192.5.5.110:23
```

As you can see, the xmas tree scan simply sets the initial tcp packets control flags to FIN (Finish), URG (Urgent), and PSH (Push). If a system's tcp/ip implementation is developed according to RFC 793, then the above packet sent to an open port will not elicit a response from the host. Where as if the port is in state closed, the remote host will reply with a RST/ACK. Given this, we can assume any port scanned on an active host that does not return a response to the client system is in state open.

One disadvantage to this technique was partially discussed in the above paragraph, the system's tcp/ip implementation MUST correspond with RFC standard 793. As such, this method will not work against any current version of Microsoft Windows. Xmas scans directed at any Microsoft system will show all ports on the host as being closed. This is because Microsoft followed their usual habit of negating inter-compatibility standards and go about their merry way with their coding. When a Microsoft system receives an xmas packet, it will respond with a RST/ACK, regardless of whether or not the port is open or closed. Since an xmas scan deems that a response from the remote host indicates a closed port, a Microsoft system is invulnerable to the xmas tree scan, as it will show all ports closed regardless of their state.

FIN and NULL scans

Both the FIN and NULL scans work on exactly the same principle as the Xmas scan, with the exception of the initiating packet structure. As you may have guessed, the initiation of the FIN scan is simply a tcp packet with only the FIN flag set, and a NULL scan begins with a packet that has no flags set. The actual process of determining the ports state is identical to that of the Xmas scan. For those still interested, examples of both the FIN and the NULL scans are below:

FIN scan directed at open port:

```
192.5.5.92:4031--------------------------------FIN-------------------------->192.5.5.110:23
192.5.5.92:4031<---------------------NO RESPONSE-----------------------192.5.5.110:23
```

FIN scan directed at closed port:

```
192.5.5.92:4031----------------------------FIN--------------------------->192.5.5.110:23
192.5.5.92:4031<---------------------------RST/ACK----------------------192.5.5.110:23
```

NULL scan directed at open port:

```
192.5.5.92:4031----------------------NO FLAGS SET--------------------->192.5.5.110:23
192.5.5.92:4031<---------------------NO RESPONSE----------------------192.5.5.110:23
```

NULL scan directed at closed port:

```
192.5.5.92:4031----------------------NO FLAGS SET-------------------->192.5.5.110:23
192.5.5.92:4031<---------------------------RST/ACK----------------------192.5.5.110:23
```

As you can see, the process of determining closed and open ports is identical to that of the Xmas scan. So in the light, it also falls prey to the same limitations that its predecessor does, it cannot be used against non RFC compliant systems, such as Microsoft Windows. The positive aspect of this limitation is that even though it cannot be used to determine a remote operating system, and can be used to determine what the remote operating system is not. For example if you were to scan a host that you know to be active, and the result is that of all remote ports being closed, it can safely be assumed that the remote system is non RFC compliant, so more than likely, is Windows.

Another problem that lies within these types of scans is that the majority of firewalls in use on the internet today will not send back the standard RST/ACK when an Xmas, FIN, or NULL packet hits a filtered port. It simply drops the packet and continues on with its business. The problem being that these scans take the absence of a response to mean that the port is open and listening. So any of these scans directed at a firewalled system will result in the scan showing all attempted ports as being open. Obviously not very effective as far as enumeration goes. Never the less, these three scans remain a powerful, and extremely stealth way to enumerate a remote system's ports.

Idle or "Zombie" Scans

Idle scanning is the newest and most stealth of all the port scanning techniques today. Also called "zombie" scanning, this method is unique in the fact that it offers completely blind scanning of a remote host. No packets will with the attackers ip will ever reach the victim system. This is accomplished by using a flaw that exists in the majority of operating system's IP ID generation for ip communications on the internet. The problem being that most system's IP ID's are incremented by one after every transmission made. An attacker can easily use this predictability to gain a surprisingly accurate idea of what is going on between the remote host and any other systems it comes in contact with. Below is the output of an hping command:

Hping2 output:

```
[root@illiterate /]# hping2 -1 192.5.5.254
HPING 192.5.5.254 (eth0 192.5.5.254): icmp mode set, 28 headers + 0 data bytes
len=28 ip=192.5.5.254 ttl=255 id=5819 icmp_seq=0 rtt=0.7 ms
len=28 ip=192.5.5.254 ttl=255 id=5820 icmp_seq=1 rtt=0.3 ms
len=28 ip=192.5.5.254 ttl=255 id=5821 icmp_seq=2 rtt=0.3 ms
len=28 ip=192.5.5.254 ttl=255 id=5822 icmp_seq=3 rtt=0.2 ms
```

The hping2 output shows that the system in question's IP ID is in fact incremented by one after every transmission made. Knowing that the system has a predictable IP ID. We can then use hping again to gain an idea about the remote machines traffic that is not our own. Examine the following hping output:

Hping2 output:

```
[root@illiterate /]# hping2 -1 192.5.5.254
HPING 192.5.5.254 (eth0 192.5.5.254): icmp mode set, 28 headers + 0 data bytes
len=28 ip=192.5.5.254 ttl=255 id=6241 icmp_seq=0 rtt=0.5 ms
len=28 ip=192.5.5.254 ttl=255 id=6242 icmp_seq=1 rtt=0.3 ms
len=28 ip=192.5.5.254 ttl=255 id=6425 icmp_seq=2 rtt=0.4 ms
len=28 ip=192.5.5.254 ttl=255 id=6427 icmp_seq=3 rtt=0.2 ms
len=28 ip=192.5.5.254 ttl=255 id=6428 icmp_seq=4 rtt=0.3 ms
len=28 ip=192.5.5.254 ttl=255 id=6429 icmp_seq=5 rtt=0.1 ms
len=28 ip=192.5.5.254 ttl=255 id=6433 icmp_seq=6 rtt=0.2 ms
```

Looking at the output above, we can see that the machine in question was interacting with hosts other than our own. After the second ping, the IP ID incremented by three, knowing that this host uses predictable IP ID's, we can see that sometime between our second and third pings, the remote host sent two other packets to an unknown location. This anomaly is seen again between the third and fourth packets, and also between the sixth and seventh.

So how do we use IP ID predictability to blindly scan another host? We make sure that we know what our zombie is sending, and where it is sending it to. The following is an example of how to check the status of a single port on

a remote system using a zombie host to hide our scan. Using the nmap -sI option will indeed give you the same effect and faster, but I believe using hping will better help you understand what is happening.

Use Hping2 to check the current IP ID of the zombie:

```
 [root@illiterate /]# hping2 -1 192.168.0.15
HPING 192.168.0.15 (eth0 192.168.0.15): icmp mode set, 28 headers + 0 data bytes
len=28 ip=192.168.0.15 ttl=255 id=3723 icmp_seq=0 rtt=0.3 ms
```

From the hping output above, we can see that the current ID is 3723. Now we spoof a SYN packet from the machine we wish to scan, lets say it's IP is 192.168.0.100 and we wish to check port 25 (smtp):

```
[root@illiterate /]# hping2 -S 192.168.0.100 -p 25 -a 192.168.0.15 -s 139 -c 1
HPING 192.168.0.100 (eth0 192.168.0.100): S set, 40 headers + 0 data bytes

--- 192.168.0.15 hping statistic ---
1 packets transmitted, 0 packets received, 100% packet loss
```

So we've used hping2 to check the initial IP ID of the zombie, and then used hping2 again to send a spoofed packet from our zombie's ip and known port to our target on the port in question. Don't let the fact that the syn packet did not return us a response, its not supposed to. If our targets port is open, it will respond to the zombie with a single SYN/ACK packet.

```
[root@illiterate /]# hping2 -1 192.168.0.15
HPING 192.168.0.15 (eth0 192.168.0.15): icmp mode set, 28 headers + 0 data bytes
len=28 ip=192.168.0.15 ttl=255 id=3725 icmp_seq=0 rtt=0.4 ms
```

Now we use hping2 one last time to probe for the IP ID after the spoofed packet has been sent. If the target port on 192.168.0.100 is open, then the new IP ID should be two more than the original probe. This is a result of the target receiving the initiating SYN packet from what it believes was our zombie host, 192.168.0.15, and responding according to RFC standards with a SYN/ACK to the remote port. Upon receiving this SYN/ACK request with no prior knowledge of an initiation, the zombie then responds back with a single RST packet, incrementing it's IP ID by one, and then one more when responding to the second echo request. So looking at the previous hping echo request, we can see that the ID is in fact two larger than the original probe, so we can assume the port on 192.168.0.100 is in state open.

If the remote port on our target had been closed, it would have responded to the spoofed packet with a single RST packet, and following RFC standards, the zombie host would not render a response, thus having no increment in the IP ID.

As you may have guessed by now, time is of the essence in this scan. Should this scan be directed at a high traffic commercial web site, there would be too many transmits between our ID probes, thus, our probes and spoofed packets must be nearly simultaneous. This can be accomplished by setting the delay for hping's IP ID probes to an extremely low number, and logging the results to a text file for later review. (or using nmap -sI)

Hping2 fast probing and logging example:

```
 [root@illiterate /]# hping2 -i u15000 192.168.0.15 > hpinglog &
```

The example above will probe the zombie host fifteen times every second, and put it into the system background, if the port on the target system is open, you will find the extra increment in the log file.

While using hping as a tool for tcp idle scans is possible and effective, you are clearly better off just letting nmap handle all of the dirty work for you, or code your own implementation of this system if you are on that level.

Using Ordinary SYN Packets to Enumerate a Firewall

The past few years have seen a major spike in the use of firewalls deployed to protect both corporate networks and home users. While these firewalls are a relatively easy and cost effective defense mechanism, determined hackers will not have a great deal of trouble finding a way through.

This section of the text will once again stress the use of Antirez's hping2 utility, if you do not have a copy of this excellent packet building tool I suggest you get one right now. http://www.hping.org. Before we get into mapping out ip trust relationships and actually slipping by the firewall, first we must learn to identify one. This is not done by checking to see if a port is closed, it is done by checking to see how a port is closed. If you remember back to the SYN scan section, you can recall that an open port will respond to a SYN packet with a single SYN/ACK, and a closed port will respond with a single RST. Firewalls however, do not follow this rule. Examine the following iptables entries:

```
[root@dolphbox /]# /sbin/iptables -A INPUT -s 0/0 --proto tcp --dport 25 -j DROP
[root@dolphbox /]# /sbin/iptables -A INPUT -s 0/0 --proto tcp --dport 23 -j ACCEPT
[root@dolphbox /]# /sbin/iptables -A INPUT -s 0/0 --proto tcp --dport 20 -j ACCEPT
```

In the proceeding example we can see that we deny all access to the stmp server, port 25. The next two lines instruct our firewall to accept all traffic to both the telnet server, port 23, as well as tcp port 20, which is currently closed, and not operating a service. Now examine the following hping2 results when we attempt to initiate a connection to the denied stmp service:

```
[root@getem /]# hping2 -S 192.5.5.254 -p 25
HPING 192.5.5.254 (eth0 192.5.5.254): S set, 40 headers + 0 data bytes

--- 192.5.5.254 hping statistic ---
3 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
[root@getem /]#
```

The SYN packet from hping2 hit the firewall, was caught by the rules, and dropped accordingly by the kernel. Now the same packet but to accepted open port 23:

```
[root@getem /]# hping2 -S 192.5.5.254 -p 23
HPING 192.5.5.254 (eth0 192.5.5.254): S set, 40 headers + 0 data bytes
len=44 ip=192.5.5.254 ttl=64 DF id=0 sport=23 flags=SA seq=0 win=32767 rtt=0.8 ms

--- 192.5.5.254 hping statistic ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.8/0.8/0.8 ms
[root@getem /]#
```

This time, our SYN packet was caught by the firewall, and passed on to the listening telnet service as shown in rule two. The telnet server, responds accordingly with a SYN/ACK. Now check the final test, another SYN packet directed at firewall accepted, but locally closed port:

```
[root@getem /]# hping2 -S 192.5.5.254 -p 20
HPING 192.5.5.254 (eth0 192.5.5.254): S set, 40 headers + 0 data bytes
len=40 ip=192.5.5.254 ttl=255 DF id=0 sport=20 flags=RA seq=0 win=0 rtt=0.5 ms

--- 192.5.5.254 hping statistic ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.5/0.5/0.5 ms
[root@getem /]#
```

In the final example, the firewall examines the packet, and passes it on the local port 20, the only problem being that there is no service running on port 20, so our kernel catches this attempt to a closed port, and responds back according to RFC 793 with an RST/ACK packet.

Now how does this information tell us that a firewall is in place on the remote system? If you look back to the first connection initiation attempt, you will notice that the SYN packet directed at the firewall denied port elicited absolutely no response back to the scanning system at all. This is because as a standard firewalls do not respond to restricted packets. When a host is known to be active, but makes no response what so ever to a SYN packet, then the odds are that the host is using a firewall or other packet filtering software. This can be further proven by the other two examples, where an open port responds with a SYN/ACK, and a port excepted by the firewall, but not running a service, responds with an RST/ACK .

Enumerating IP Trust Relationships

An essential element for successful deployment of a firewall is that the hosts behind the firewall be able to reach the resources that they need. As such users behind the firewall may be able to reach systems and services that people on the outside cannot. The ability of only certain hosts being able to reach crucial systems is called an "IP Trust Relationship". The ability of an attacker to map and understand these systems is often a must for successful penetration of the network.

For this example, we will be using three hosts on local network 192.168.1.0.

```
192.168.1.0 Network

192.168.1.2     Attacking Host
192.168.1.4     Legitimate Web Server
192.168.1.7     Legitimate Server: Telnet, smtp, and dns
```

Our attacker is curious about the relationship between the two known servers on the network. Obviously the best place to start would be to a simple port scan on both hosts.

```
root@hollabox[/]# nmap -sS 192.168.1.7

Starting nmap V. 3.10ALPHA4 ( www.insecure.org/nmap/ )
Skipping host 192.168.1.7 due to host timeout

Nmap run completed - 1 IP address (1 host up) scanned in 75.571 seconds
root@hollabox[/]#
```

Looking at the nmap output above, we see that the host was in fact up, but did you respond to any of our SYN

probes. If you recall back to the first section on SYN scans, you will remember that a normal system will always generate a response to a SYN packet, whether the port is in state open or closed. This system however, did not, so we can assume it is behind some sort of firewall. Now for the second system.

```
root@hollabox[/]# nmap -sS 192.168.1.4

Starting nmap V. 3.10ALPHA4 ( www.insecure.org/nmap/ )
Interesting ports on 192.168.1.4:
(The 1604 ports scanned but not shown below are in state: filtered)
Port            State           Service
80/tcp          open            http

Nmap run completed - 1 IP address (1 host up) scanned in 158.102 seconds
root@hollabox[/]#
```

We see the same firewall effect on the second system, except that this system does have a service accessible by the attacker, http.

Knowing that we have at least one port accepting traffic from systems outside the firewall, we can gain a better prospective of what is happening behind the network firewall. To accomplish this we will use the idle scan technique discussed in a previous section, only this time we will implement it using nmap.

```
Nmap -sI 192.168.1.4:80 192.168.1.7
WARNING: Many people use -P0 w/Idlescan to prevent pings from their true IP.  On the other hand,
timing info Nmap gains from pings can allow for faster, more reliable scans.

Starting nmap V. 3.10ALPHA4 ( www.insecure.org/nmap/ )
Interesting ports on 192.168.1.7:
(The 1600 ports scanned but not shown below are in state: closed)
Port            State           Service
23/tcp          open            telnet
25/tcp          open            smtp
53/tcp          open            domain

Nmap run completed - 1 IP address (1 host up) scanned in 152.305 seconds
root@hollabox[/]#
```

We can clearly see that this scan is quite different from the original scan directed at 192.168.1.7. This is because the idle scan shows us what ports are open from 192.168.1.4's prospective. The output above shows that the networks web server has complete tcp/ip trust from the ftp server.

Using this information, an attacker could then implement a number of trust based attacks against either one of these systems. However, these attacks are outside the scope of this particular paper.

The End

I hope the techniques and knowledge contained in this paper will help its readers to better understand tcp/ip, and the many ways that it can be, and is abused on the internet today. However, I cannot be held responsible for any kind of misuse that a reader does with the information contained here. This article was written to help the reader to know what kind of attacks are out there, in order to better prepare their own network for them.

And at long last____..the end.