Yapscan User Documentation

pentestmonkey@pentestmonkey.net

22 January 2007

Contents

1 Overview **2**

2 License **2**

3 Installation **2**
3.1 Quick Start 2
3.2 Trouble Shooting 3
3.3 Workaround For Missing OpenSSL Libraries 3

4 Features **4**
4.1 TCP SYN Scanning 4
4.2 (Limited) UDP Port Scanning 5
4.3 ICMP Scanning 5
4.4 Scanning Speed 5
4.5 Retries 6

5 Yapscan Output **6**
5.1 Scan Information 6
5.2 Scan Start and End Times 7
5.3 TCP Scanning 8
5.4 ICMP Scanning 9
5.4.1 Generic ICMP Fields 9
5.4.2 ICMP Address Mask Fields 9
5.4.3 ICMP Timestamp Fields 10
5.4.4 ICMP Echo Fields 10
5.4.5 ICMP Information Fields 10

6 Limitations **11**
6.1 Memory-hungry 11
6.2 Replies from different IPs 11

7 Why write Yet Another Port Scanner? **11**

8 Credit **12**

1 Overview

Yapscan is primarily a half-open TCP port scanner and ICMP scanner. It has a few other uses too. These are explained more fully in the "Features" section below.

2 License

This tool may be used for legal purposes only. Users take full responsibility for any actions performed using this tool. The author accepts no liability for damage caused by this tool. If these terms are not acceptable to you, then do not use this tool.

In all other respects the GPL version 2 applies:

```
This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License version 2 as
published by the Free Software Foundation.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License along
with this program; if not, write to the Free Software Foundation, Inc.,
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
```

3 Installation

3.1 Quick Start

If you have a Linux system, with any luck the installation will just work. Other systems are not supported at this time.

Firstly, go and download the latest tarball of yapscan from http://pentestmonkey.net if you haven't already.

```
su -
cd /usr/local/src
tar xfz yapscan-X.Y.tar.gz
cd yapscan-X.Y
make
make install
```

3.2 Trouble Shooting

Yapscan should install on most flavours of Linux. I've tried it on Gentoo using gcc v3.3.x, 3.4.x and 4.1.x (both x86 and AMD64) and Debian "Testing" using gcc-4.1.2. I think I've ironed out any bugs which stop compilation. That said, you're reading this section, so I guess something went wrong...

Any installation problems will be results of either missing header files, missing libraries or my dodgy code. Whatever the cause I'd like to make sure that this section covers the problem (or that I fix any dodgy code). If you run into problems and the notes below don't help, please email me at the address on the first page.

Yapscan depends on libpcap to capture reply packets, so firstly make sure you've installed it. On Gentoo the package is called "libpcap" on Debian it's "libpcap-dev"(the version that includes the header files).

Linking against OpenSSL is also recommended because it speeds up scanning. Installation is still possible if you can't install OpenSSL - there's a workaround below. The OpenSSL package on Gentoo is called "openssl", on Debian it's "openssl-dev".

If installing these dependencies doesn't solve your problem, mail me and I'll try to help.

3.3 Workaround For Missing OpenSSL Libraries

Open up "Makefile" in you favourite text editor and change this bit:

```
# OpenSSL's MD5 library speeds up scanning.  If you have openssl installed, do this:
DEFINES=-DHAVE_LIBCRYPTO ${DEBUGDEFINES}
LDLIBS=-lpcap -lcrypto

# Otherwise do this:
# DEFINES=${DEBUGDEFINES}
# LDLIBS=-lpcap
```

to:

```
# OpenSSL's MD5 library speeds up scanning.  If you have openssl installed, do this:
# DEFINES=-DHAVE_LIBCRYPTO ${DEBUGDEFINES}
# LDLIBS=-lpcap -lcrypto

# Otherwise do this:
DEFINES=${DEBUGDEFINES}
LDLIBS=-lpcap
```

The try to compile again:

```
make clean
make
make install
```

4 Features

4.1 TCP SYN Scanning

Also known as half-open scanning.

On an internal network you probably just want to see the open ports. Here are some example of how to specify hosts and ports:

```
# yapscan -sS 192.168.0.1-254 -p 1-1024
# yapscan -sS 10.0.0.1-10.0.255.255 -p 21,22,23,53,80,139,445,3389
# yapscan -sS 172.16.0.0/16 -p 80
# yapscan -sS -f targets-ips.txt -p 4444
```

From an external network, you might also want to see the closed ports (-c):

```
# yapscan -sS www.example.com -p 1-65535 -c
```

You can scan just the common ports (using a portlist derived from nmap):

```
# yapscan -sS 127.0.0.1 -i lo -P common
```

(supported keywords are based on filenames what ship with yapscan. As of v0.5.5-beta there is: all, known, common, database)

Note that we needed to specify the different interface to listen for replies on (default is eth0).

Specify ports using names as well as numbers (from /etc/services):

```
# yapscan -sS router -i eth1 -p telnet,80,443,ssh,6000-6063
```

Or specify your own port list (1 port per line):

```
# yapscan -sS -f mytargets.txt -P myports.txt
```

You can do "-p -" like in nmap too if you want to scan 1-65535:

```
# yapscan -sS -f mytargets.txt -p -
```

I've also implemented the exotic type of scans like Xmas tree, null, etc. These aren't particularly well tested as of v0.5.5-beta. See the help message for more info: yapscan -h

Also see "Scanning Speed" and "Retries" near the end of this section. In particular, make sure you specify a low speed for remote testing (e.g. -b 32k).

4.2 (Limited) UDP Port Scanning

This scan mode is designed to answer the question "Does host X have any closed UDP ports?" i.e. does it reply with an ICMP Port Unreachable for probes sent to one or more of its UDP ports.

This type of scan will (unfortunately) not tell you all the open UDP ports.

I use this mainly for scanning Firewalled hosts which I'm pretty sure won't have any closed UDP ports.

Yapscan sends empty UDP packets to a range of ports at a steady (usually quite fast) rate. It will report any ICMP port unreachable messages it receives.

If you receive no replies then you know there are no closed UDP ports.

```
# yapscan -su router -i eth1 -p 1-65535
```

IMPORTANT NOTE: If you receive 1 or more ICMP port unreachable error messages, you cannot infer that these are the only close ports. Yapscan does not back-off intelligently like nmap, so a host which limits that rate at which it sends ICMP errors, will (falsely) appear to have less ports open.

4.3 ICMP Scanning

Yapscan can perform the following type of ICMP sweeps:

- Echo Request
- Timestamp Request
- Addressmask Request
- Information Request

You can perform 1 or more types of scan at once:

```
# yapscan -sI 10.0.0.0/16 -t echo
# yapscan -sI 10.0.0.0/16 -t echo -t addr
# yapscan -sI 10.0.0.0/16 -t info
# yapscan -sI 10.0.0.0/16 -t time
# yapscan -sI 10.0.0.0/16 -t -
```

The last example will scan all supported ICMP types.

As of v0.5.5-beta yapscan is also able to send Router Solicitations, but it won't report replies, so this 5th type isn't much use at present.

4.4 Scanning Speed

Yapscan scans at a steady (and configurable) speed. You can get an ETA on you scan by pressing Enter during the scan.

As of v0.4.9-beta yapscan will never underestimate the remaining scan time, though it can over estimate it under certain conditions.

By default yapscan scans at 1000000 Bits / Second. Unless you have a fast link / understanding clients or both I suggest you only use the default for LAN testing. I wouldn't recommend going much about 2Mb/s for reliability / DoS reasons, but you can try it if you like:

```
# yapscan -sS -p - 192.168.0.1-14 -b 4M
```

WAN testing's probably better done at a more sociable speed like 64Kb/s:

```
# yapscan -sS -p - www.example.com -b 64k
```

Obviously, if the scan rate is set higher than either your upstream bandwidth or the client's downstream bandwidth, packets will be dropped and the reliability of the scan reduced.

4.5 Retries

Reliability is obviously paramount during pentests, so the use of retries is encouraged. ICMP scans do 2 retries by default (a total of 3 tries in all). TCP and UDP only do 1.

For an even more reliable ICMP scan you could do:

```
# yapscan -sI -r 5 myhost -t -
```

A TCP scan would be made more reliable by:

```
# yapscan -sS -r 2 myhost -p -
```

5 Yapscan Output

5.1 Scan Information

The first thing you see when you run yapscan is the "Scan Information" section. This section summarises the parameters for the scan. I included this basically so that when I looked back over my scan results I had some idea of what I'd scanned.

```
 ------------------------------------------------------
|                   Scan Information                   |
 ------------------------------------------------------
Target count: ...... 1
Interface: ......... lo
Bandwidth limit: ... 1000000 bits/sec
Source address: .... 127.0.0.1
RTT: ............... 0.950000 secs
Tries: ............. 3
ICMP Probe Types: .. 8 (ECHO_REQUEST)
```

The output looks slightly different for the various scan types.

5.2 Scan Start and End Times

The start and end times are included to provide a record of when the scan was done.

```
######## Scan started at 2006-10-22 20:37:26 +0000 #########
...
####### Scan completed at 2006-10-22 20:37:27 +0000 #########
```

5.3 TCP Scanning

Below is the output of a fictional TCP SYN scan run with the -c option to show closed ports as well as open ones (so I can illustrate some fields not always shown in the output).

```
10.0.1.1:25   smtp  Len=46 TTL=19 IPID=0 FLAGS=_AR_____ SEQ=0x00000000 ACK=0x4e5d5003 WIN=0 DATA="rctcpoy"
10.0.2.2:80    http  Len=46 TTL=21 IPID=0 FLAGS=_AR_____ SEQ=0x00000000 ACK=0x0b712500 WIN=0 DATA="rctcpo"
10.0.3.3:53    domain  Len=44 TTL=64 IPID=0 FLAGS=SA______ SEQ=0xd1183250 ACK=0x17f67482 WIN=32792
10.0.4.4:1999     tcp-id-port     Len=40 TTL=59 IPID=43634 FLAGS=_AR____C SEQ=0x00000000
ACK=0x28f45290 WIN=0
10.0.5.5:1999      tcp-id-port     Len=45 TTL=244 IPID=22340 FLAGS=_AR_____ SEQ=0x00000000
ACK=0xab7e84c2 WIN=0 DATA="cisco"
```

We'll use the first reply to quickly cover most of the generic fields, then we'll look at some of the interesting ones.

Field	**Meaning**
10.0.3.3	IP Address reply came from
25	TCP Port number reply came from
smtp	Name of port (from /etc/services)
Len=46	Length of packet (IP header + TCP header + data)
TTL=19	Time-to-Live of reply packet
IPID=0	IP ID on reply
FLAGS=_AR_____	TCP Flags: S=SYN, A=ACK, R=RST, F=FIN, P=PSH, U=URG, E=ECN, C=CWR
SEQ=0x00000000	TCP Sequence number
ACK=0x4e5d5003	TCP Acknowledgement number
WIN=0	TCP Windows size
DATA="rctcpoy"	Data carried by TCP packet (if any)

We see that IP address 10.0.3.3 has TCP port 53 open (note the SYN and ACK flags are set on the response). The rest of the packets indicate closed ports (the RST flag is set).

Some of the RST packets have data attached. This is much more rare than I've made it seem in this example, but it's an interesting feaure of yapscan, so worth some discussion.

According to http://www.sonicwall.com/support/pdfs/technotes/SonicOS_TCP_RST.pdf, the string "rctcpo" for 10.0.2.2 seems to indicate that the host is protected by a Sonicwall Firewall. Presumably 10.0.3.3 is similarly protected, but I haven't tracked down any documentation as to the exact meaning of "rctcpoy".

The data "cisco" is return on the response from 10.0.5.5. This is a feature of certain older version of Cisco IOS. The feature is documented as a fingerprinting vulnerability at http://cve.mitre.org/cgi-bin/cvename.cgi?name=1999-0453.

Finally, note that the packet from 10.0.4.4 has the TCP C flag set. This is the Congestion Window Reduced flag, a relatively new TCP option.

NB: There is currently no support for displaying IP options or TCP options.

5.4 ICMP Scanning

The output below shows pretty much all of the various fields you're likely to come across when running ICMP sweeps with yapscan:

```
10.0.1.1:18/0 [ADDRESS_REPLY] Len=32 TTL=56 IPID=10224 ID=10406 SEQ=13643 MASK=255.255.254.0
10.0.2.2:18/0 [ADDRESS_REPLY] Len=32 TTL=248 IPID=15111 ID=4233 SEQ=39939 MASK=255.255.255.252
10.0.3.3:14/0 [TIMESTAMP_REPLY] Len=40 TTL=249 IPID=31174 ID=2463 SEQ=52706 orig=0x040f417e
recv=0x83ecf893 xmit=0x83ecf893 std=0 end=1
10.0.4.4:14/0 [TIMESTAMP_REPLY] Len=40 TTL=128 IPID=1617 ID=2182 SEQ=92 orig=0x041ae462
recv=0x9cdc1a04 xmit=0x9cdc1a04 std=1 end=0 xmit-time=19:07:49.276 delta=-1.984
10.0.5.5:14/0 [TIMESTAMP_REPLY] Len=40 TTL=240 IPID=10035 ID=6689 SEQ=42152 orig=0x040f40aa
recv=0x04131907 xmit=0x04131907 std=1 end=1 xmit-time=18:59:20.455 delta=252.000
10.0.6.6:16/0 [INFO_REPLY] Len=28 TTL=247 IPID=48924 ID=3706 SEQ=64144
10.0.7.7:16/0 [INFO_REPLY] Len=28 TTL=243 IPID=64889 ID=10674 SEQ=17306
10.0.8.8:0/0 [ECHO_REPLY] Len=28 TTL=128 IPID=1616 ID=21821 SEQ=92
```

5.4.1 Generic ICMP Fields

The following fields are common to all ICMP results:

Field	**Meaning**
10.0.1.1	IP Address reply came from
18/0	The ICMP Type of the reply was 18. The ICMP Code was 0.
[ADDRESS_REPLY]	A human readable form for the ICMP Type. Type 18 is Address Mask Reply
Len=32	Length of packet (IP header + ICMP header + data)
TTL=56	Time-to-Live of reply packet
IPID=10224	IP ID on reply
SEQ=13643	ICMP Sequence number

5.4.2 ICMP Address Mask Fields

A single field is unique to ICMP Address Mask scans:

Field	**Meaning**
MASK=255.255.254.0	The netmask of the corresponding IP is 255.255.254.0

5.4.3 ICMP Timestamp Fields

Yapscan will tell you if the remote system clock is accurate if the timestamp supplied is in a standard format. It manages this even the remote system has a broken IP stack which uses the wrong endianness for the timestamp fields.

Field	**Meaning**
orig=0x040f417e	Originator timestamp in hex
recv=0x83ecf893	Received Timestamp in hex
xmit=0x83ecf893	Transmit Timestamp in hex
std=0	Do the timestamp fileds contain the number of milliseconds since midnight UTC as suggested by RFC 792?
end=1	Does the endianness of the xmit field appear to be correct (big endian AKA network byte order)? Most versions of Windows use the wrong endianness in this field.
xmit-time=19:07:49.276	Only displayed if std=1. A human readable representation of the xmit field. If the endianness of xmit was wrong, it is reverse before being converted to human readable format, so Yapscan will display meaningful timestamps even for Windows systems.
delta=-1.984	Only displayed if std=1. The number of seconds ahead of UTC the remote system's clock is. (UTC according to your system clock).

5.4.4 ICMP Echo Fields

See Generic ICMP Fields section above. No additional fields are available for Echo replies.

5.4.5 ICMP Information Fields

See Generic ICMP Fields section above. No additional fields are available for Info replies.

6 Limitations

6.1 Memory-hungry

Yapscan implements retries by keeping a list of hosts and ports to be scanned in memory. This has the side effect of using an awful lot of memory on large scans: 770MB for 65535 ports on 256 hosts

This a pretty big problem. I really need to break the scan into chunks.

6.2 Replies from different IPs

If you send a packet to an address which elicits a reply from a different IP address (e.g. you ping 192.168.0.255 and get a reply from 192.168.0.5) the reply will not be reported by yapscan.

This is because all cookies carry a "cookie" of some description which is derived from the source and destination IP of the original probe. Yapscan will inspect the reply and ensure that cookie contained within it is derived from the source and destination IP. If an unexpected IP replies yapscan will assume that the traffic is not a response to a probe.

7 Why write Yet Another Port Scanner?

When I started writing yapscan I wanted to:

1. Learn some C++
2. Write some generic classes that handle all the mundane parts of port scanning (like hostlist traversal, retries, bandwidth usage).

The idea being that next time I needed to write a scanner (whether it be ARP, IPv6, DNS server-finder, mass DoS payload deliverer, etc.) I'd be able to quickly code up the probe, define how to parse responses and the generic classes would take care of the rest.

Alas, my eyes were too big for my belly and I've ended up with yet another IPv4 port scanner.

Maybe I'll achieve my original goal someday, but as of today I'm still some way off.

That said, I do find yapscan useful during most pentests, so I thought I'd submit it back to he community in hope that others would too.

8 Credit

Much inspiration (and even small amounts of code) as been drawn from other tools. It's only right that I pay my dues...

synscan *http://bindshell.net/tools/synscan*

The fastest half-open portscanner that I'm aware of. I borrowed the code for determining link-layer header lengths from synscan.

nmap *http://insecure.org/nmap*

The most reliable portscanner I'm aware of. I use nmap during every pentest. Most of yapscan's options are designed to be intuitive for nmap users.

ike-scan *http://www.nta-monitor.com/tools/ike-scan*

Hostlist structures and retries are heavily inspired by ike-scan.

hping2 *http://www.hping.org*

The output format is loosley based on hping2's.

scanrand *http://www.doxpara.com/paketto*

I love the Inverse SYN Cookies idea. Very simple method to determine if replies on the wire are meant for your scanner or some other app.