

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

## Abstract

This paper details many of the techniques used to determine what ports (or similar protocol abstraction) of a host are listening for connections. These ports represent potential communication channels. Mapping their existence facilitates the exchange of information with the host, and thus it is quite useful for anyone wishing to explore their networked environment, including hackers. Despite what you have heard from the media, the Internet is NOT all about TCP port 80. Anyone who relies exclusively on the WWW for information gathering is likely to gain the same level of proficiency as your average AOLer, who does the same. This paper is also meant to serve as an introduction to and ancillary documentation for a coding project I have been working on. It is a full featured, robust port scanner which (I hope) solves some of the problems I have encountered when dealing with other scanners and when working to scan massive networks. The tool, nmap, supports the following:

- vanilla TCP connect() scanning,
- TCP SYN (half open) scanning,
- TCP FIN (stealth) scanning,
- TCP ftp proxy (bounce attack) scanning
- SYN/FIN scanning using IP fragments (bypasses packet filters),
- UDP recvfrom() scanning,
- UDP raw ICMP port unreachable scanning,
- ICMP scanning (ping-sweep), and
- reverse-ident scanning.

The freely distributable source code is appended to this paper.

## Introduction

Scanning, as a method for discovering exploitable communication channels, has been around for ages. The idea is to probe as many listeners as possible, and keep track of the ones that are receptive or useful to your particular need. Much of the field of advertising is based on this paradigm, and the "to current resident" brute force style of bulk mail is an almost perfect parallel to what we will discuss. Just stick a message in every mailbox and wait for the responses to trickle back.

Scanning entered the h/p world along with the phone systems. Here we have this tremendous global telecommunications network, all reachable through codes on our telephone. Millions of numbers are reachable locally, yet we may only be interested in 0.5% of these numbers, perhaps those that answer with a carrier.

The logical solution to finding those numbers that interest us is to try them all. Thus the field of "wardialing" arose. Excellent programs like Toneloc were developed to facilitate the probing of entire exchanges and more. The basic idea is simple. If you dial a number and your modem gives you a CONNECT, you record it. Otherwise the computer hangs up and tirelessly dials the next one.

While wardialing is still useful, we are now finding that many of the computers we wish to communicate with are connected through networks such as the Internet rather than analog phone dialups. Scanning these machines involves the same brute force technique. We send a blizzard of packets for various protocols, and we deduce which services are listening from the responses we receive (or don't receive).

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

## Techniques

Over time, a number of techniques have been developed for surveying the protocols and ports on which a target machine is listening. They all offer different benefits and problems. Here is a line up of the most common:

- TCP connect() scanning : This is the most basic form of TCP scanning. The connect() system call provided by your operating system is used to open a connection to every interesting port on the machine. If the port is listening, connect() will succeed, otherwise the port isn't reachable. One strong advantage to this technique is that you don't need any special privileges. Any user on most UNIX boxes is free to use this call. Another advantage is speed. While making a separate connect() call for every targeted port in a linear fashion would take ages over a slow connection, you can hasten the scan by using many sockets in parallel. Using non-blocking I/O allows you to set a low time-out period and watch all the sockets at once. This is the fastest scanning method supported by nmap, and is available with the -t (TCP) option. The big downside is that this sort of scan is easily detectable and filterable. The target hosts logs will show a bunch of connection and error messages for the services which take the connection and then have it immediately shutdown.

- TCP SYN scanning: This technique is often referred to as "half-open" scanning, because you don't open a full TCP connection. You send a SYN packet, as if you are going to open a real connection and wait for a response. A SYN|ACK indicates the port is listening. A RST is indicative of a non-listener. If a SYN|ACK is received, you immediately send a RST to tear down the connection (actually the kernel does this for us). The primary advantage to this scanning technique is that fewer sites will log it. Unfortunately you need root privileges to build these custom SYN packets. SYN scanning is the -s option of nmap.
- TCP FIN scanning: There are times when even SYN scanning isn't clandestine enough. Some firewalls and packet filters watch for SYNs to an unallowed port, and programs like synlogger and Courtney are available to detect these scans. FIN packets, on the other hand, may be able to pass through unmolested. This scanning technique was featured in detail by Uriel Maimon in Phrack 49, article 15. The idea is that closed ports tend to reply to your FIN packet with the proper RST. Open ports, on the other hand, tend to ignore the packet in question. This is a bug in TCP implementations and so it isn't 100% reliable (some systems, notably Micro\$oft boxes, seem to be immune). It works well on most other systems I've tried. FIN scanning is the -U (Uriel) option of nmap.
- Fragmentation scanning: This is not a new scanning method in and of itself, but a modification of other techniques. Instead of just sending the probe packet, you break it into a couple of small IP fragments. You are splitting up the TCP header over several packets to make it harder for packet filters and so forth to detect what you are doing. Be careful with this! Some programs have trouble handling these tiny packets. My favorite sniffer segmentation faulted immediately upon receiving the first 36-byte fragment. After that comes a 24 byte one! While this method won't get by packet filters and firewalls that queue all IP fragments (like the CONFIG\_IP\_ALWAYS\_DEFRAG option in Linux), a lot of networks can't afford the performance hit this causes. This feature is rather unique to scanners (at least I haven't seen any others that do this). Thanks to daemon9 for suggesting it. The -f instructs the specified SYN or FIN scan to use tiny fragmented packets.
- TCP reverse ident scanning: As noted by Dave Goldsmith in a 1996 Bugtraq post, the ident protocol (rfc1413) allows for the disclosure of the username of the owner of any process connected via TCP, even if that process didn't initiate the connection. So you can, for example,

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

connect to the http port and then use identd to find out whether the server is running as root. This can only be done with a full TCP connection to the target port (i.e. the -t option). nmap's -i option queries identd for the owner of all listen()ing ports.

- FTP bounce attack: An interesting "feature" of the ftp protocol (RFC 959) is support for "proxy" ftp connections. In other words, I should be able to connect from evil.com to the FTP server-PI (protocol interpreter) of target.com to establish the control communication connection. Then I should be able to request that the server-PI initiate an active server-DTP (data transfer process) to send a file ANYWHERE on the internet! Presumably to a User-DTP, although the RFC specifically states that asking one server to send a file to another is OK. Now this may have worked well in 1985 when the RFC was just written. But nowadays, we can't have people hijacking ftp servers and requesting that data be spit out to arbitrary points on the internet. As \*Hobbit\* wrote back in 1995, this protocol flaw "can be used to post virtually untraceable mail and news, hammer on servers at various sites, fill up disks, try to hop firewalls, and generally be annoying and hard to track down at the same time." What we will exploit this for is to (surprise, surprise) scan TCP ports from a "proxy" ftp server. Thus you could connect to an ftp server behind a firewall, and then scan ports that are more likely to be blocked (139 is a good one). If the ftp server allows reading from and writing to a directory (such as /incoming), you can send arbitrary data to ports that you do find open.

For port scanning, our technique is to use the PORT command to declare that our passive "User-DTP" is listening on the target box at a certain port number. Then we try to LIST the current directory, and the result is sent over the Server-DTP channel. If our target host is listening on the specified port, the transfer will be successful (generating a 150 and a 226 response). Otherwise we will get "425 Can't build data connection: Connection refused." Then we issue another PORT command to try the next port on the target host. The advantages to this approach are obvious (harder to trace, potential to bypass firewalls). The main disadvantages are that it is slow, and that some FTP servers have finally got a clue and disabled the proxy "feature". For what it is worth, here is a list of banners from sites where it does/doesn't work:

Bounce attacks worked:

```
220 xxxxxxx.com FTP server (Version wu-2.4(3) Wed Dec 14 ...) ready.
220 xxx.xxx.xxx.edu FTP server ready.
220 xx.Telcom.xxxx.EDU FTP server (Version wu-2.4(3) Tue Jun 11 ...) ready.
220 lem FTP server (SunOS 4.1) ready.
220 xxx.xxx.es FTP server (Version wu-2.4(11) Sat Apr 27 ...) ready.
220 elios FTP server (SunOS 4.1) ready
```

Bounce attack failed:

```
220 wcarchive.cdrom.com FTP server (Version DG-2.0.39 Sun May 4 ...) ready.
220 xxx.xx.xxxxx.EDU Version wu-2.4.2-academ[BETA-12](1) Fri Feb 7
220 ftp Microsoft FTP Service (Version 3.0).
220 xxx FTP server (Version wu-2.4.2-academ[BETA-11](1) Tue Sep 3 ...) ready.
220 xxx.unc.edu FTP server (Version wu-2.4.2-academ[BETA-13](6) ...) ready.
```

The 'x's are partly there to protect those guilty of running a flawed server, but mostly just to make the lines fit in 80 columns. Same thing with the ellipse points. The bounce attack is available with the -b <proxy\_server> option of nmap. proxy\_server can be specified in standard URL format, username:password@server:port, with everything but server being optional.

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

- UDP ICMP port unreachable scanning: This scanning method varies from the above in that we are using the UDP protocol instead of TCP. While this protocol is simpler, scanning it is actually significantly more difficult. This is because open ports don't have to send an acknowledgement in response to our probe, and closed ports aren't even required to send an error packet. Fortunately, most hosts do send an ICMP\_PORT\_UNREACH error when you send a packet to a closed UDP port. Thus you can find out if a port is NOT open, and by exclusion determine which ports which are. Neither UDP packets, nor the ICMP errors are guaranteed to arrive, so UDP scanners of this sort must also implement retransmission of packets that appear to be lost (or you will get a bunch of false positives). Also, this scanning technique is slow because of compensation for machines that took RFC 1812 section 4.3.2.8 to heart and limit ICMP error message rate. For example, the Linux kernel (in net/ipv4/icmp.h) limits destination unreachable message generation to 80 per 4 seconds, with a 1/4 second penalty if that is exceeded. At some point I will add a better algorithm to nmap for detecting this. Also, you will need to be root for access to the raw ICMP socket necessary for reading the port unreachable. The -u (UDP) option of nmap implements this scanning method for root users.

Some people think UDP scanning is lame and pointless. I usually remind them of the recent Solaris rcpbind hole. Rcpbind can be found hiding on an undocumented UDP port somewhere above 32770. So it doesn't matter that 111 is blocked by the firewall. But can you find which of the more than 30,000 high ports it is listening on? With a UDP scanner you can!

- UDP recvfrom() and write() scanning : While non-root users can't read port unreachable errors directly, Linux is cool enough to inform the user indirectly when they have been received. For example a second write() call to a closed port will usually fail. A lot of scanners such as netcat and Pluvius' pscan.c does this. I have also noticed that recvfrom() on non-blocking UDP sockets usually return EAGAIN ("Try Again", errno 13) if the ICMP error hasn't been received, and ECONNREFUSED ("Connection refused", errno 111) if it has. This is the technique used for determining open ports when non-root users use -u (UDP). Root users can also use the -l (lamer UDP scan) options to force this, but it is a really dumb idea.
- ICMP echo scanning : This isn't really port scanning, since ICMP doesn't have a port abstraction. But it is sometimes useful to determine what hosts in a network are up by pinging them all. the -P option does this. Also you might want to adjust the PING\_TIMEOUT #define if you are scanning a large network. nmap supports a host/bitmask notation to make this sort of thing easier. For example 'nmap -P cert.org/24 152.148.0.0/16' would scan CERT's class C network and whatever class B entity 152.148.\* represents. Host/26 is useful for 6-bit subnets within an organization.

## Features

Prior to writing nmap, I spent a lot of time with other scanners exploring the Internet and various private networks (note the avoidance of the "intranet" buzzword). I have used many of the top scanners available today, including strobe by Julian Assange, netcat by \*Hobbit\*, stcp by Uriel Maimon, pscan by Pluvius, ident-scan by Dave Goldsmith, and the SATAN tcp/udp scanners by Wietse Venema. These are all excellent scanners! In fact, I ended up hacking most of them to support the best features of the others. Finally I decided to write a whole new scanner, rather than rely on hacked versions of a dozen different scanners in my /usr/local/sbin. While I wrote all the code, nmap uses a lot of good ideas from its predecessors. I also incorporated some new stuff like fragmentation scanning and options that were on my "wish list" for other scanners. Here are some of the (IMHO) useful features of nmap:

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

- dynamic delay time calculations: Some scanners require that you supply a delay time between sending packets. Well how should I know what to use? Sure, I can ping them, but that is a pain, and plus the response time of many hosts changes dramatically when they are being flooded with requests. Nmap tries to determine the best delay time for you. It also tries to keep track of packet retransmissions, etc. so that it can modify this delay time during the course of the scan. For root users, the primary technique for finding an initial delay is to time the internal "ping" function. For non-root users, it times an attempted connect() to a closed port on the target. It can also pick a reasonable default value. Again, people who want to specify a delay themselves can do so with -w (wait), but you shouldn't have to.
- retransmission: Some scanners just send out all the query packets, and collect the responses. But this can lead to false positives or negatives in the case where packets are dropped. This is especially important for "negative" style scans like UDP and FIN, where what you are looking for is a port that does NOT respond. In most cases, nmap implements a configurable number of retransmissions for ports that don't respond.
- parallel port scanning: Some scanners simply scan ports linearly, one at a time, until they do all 65535. This actually works for TCP on a very fast local network, but the speed of this is not at all acceptable on a wide area network like the Internet. nmap uses non-blocking i/o and parallel scanning in all TCP and UDP modes. The number of scans in parallel is configurable with the -M (Max sockets) option. On a very fast network you will actually decrease performance if you do more than 18 or so. On slow networks, high values increase performance dramatically.
- Flexible port specification: I don't always want to just scan all 65535 ports. Also, the scanners which only allow you to scan ports 1 - N sometimes fall short of my need. The -p option allows you to specify an arbitrary number of ports and ranges for scanning. For example, '-p 21-25,80,113, 60000-' does what you would expect (a trailing hyphen means up to 65536, a leading hyphen means 1 through). You can also use the -F (fast) option, which scans all the ports registered in your /etc/services (a la strobe).
- Flexible target specification: I often want to scan more than one host, and I certainly don't want to list every single host on a large network to scan. Everything that isn't an option (or option argument) in nmap is treated as a target host. As mentioned before, you can optionally append /mask to a hostname or IP address in order to scan all hosts with the same initial <mask> bits of the 32 bit IP address.
- detection of down hosts: Some scanners allow you to scan large networks, but they waste a huge amount of time scanning 65535 ports of a dead host! By default, nmap pings each host to make sure it is up before wasting time on it. It is also capable of bailing on hosts that seem down based on strange port scanning errors. It is also meant to be tolerant of people who accidentally scan network addresses, broadcast addresses, etc.
- detection of your IP address: For some reason, a lot of scanners ask you to type in your IP address as one of the parameters. Jeez, I don't want to have to 'ifconfig' and figure out my current address every time I scan. Of course, this is better than the scanners I've seen which require recompilation every time you change your address! nmap first tries to detect your address during the ping stage. It uses the address that the echo response is received on, as that is the interface it should almost always be routed through. If it can't do this (like if you don't have host pinging enabled), nmap tries to detect your primary interface and uses that address. You can also use -S to specify it directly, but you shouldn't have to (unless you want to make it look like someone ELSE is SYN or FIN scanning a host).

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

Some other, more minor options:

- `v` (verbose): This is highly recommended for interactive use. Among other useful messages, you will see ports come up as they are found, rather than having to wait for the sorted summary list.
- `-r` (randomize): This will randomize the order in which the target host's ports are scanned.
- `-q` (quash argv): This changes `argv[0]` to `FAKE_ARGV` ("pine" by default). It also eliminates all other arguments, so you won't look too suspicious in 'w' or 'ps' listings.
- `-h` for an options summary.

Also look for <http://www.dhp.com/~fyodor/nmap/>, which is the web site I plan to put future versions and more information on. In fact, you would be well advised to check there right now.

## Greets

Of course this paper would not be complete without a shout out to all the people who made it possible.

- Congratulations to the people at Phrack for getting this thing going again!
- Greets to the whole dc-stuff crew.
- Greets to the STUPH, Turntec, L0pht, TACD, the Guild, cDc, and all the other groups who help keep the scene alive.
- Shout out to `_eci` for disclosing the coolest Windows bug in recent history.
- Thanks to the Data Haven Project ([dhp.com](http://dhp.com)) admins for providing such great service for \$10/month.
- And a special shout out goes to all my friends. You know who you are and some of you (wisely) stay out of the spotlight, so I'll keep you anonymous ... except of course for Ken and Jay, and Avenger, Grog, Cash Monies, Ethernet Kid, Zos, JulCe, Mother Prednisone, and Karen.

And finally, we get to ...

## The Code

This should compile fine on any Linux box with `'gcc -O6 -o nmap nmap.c -lm'`.

It is distributed under the terms of the GNU GENERAL PUBLIC LICENSE. If you have problems or comments, feel free to mail me ([fyodor@dhp.com](mailto:fyodor@dhp.com)).

```
<+> nmap/Makefile
# A trivial makefile for Network Mapper
nmap: nmap.c nmap.h
    gcc -Wall -O6 -o nmap nmap.c -lm
<-->
```

```
<+> nmap/nmap.h
#ifndef NMAP_H
#define NMAP_H
```

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

```
/******INCLUDES******/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <rpc/types.h>
#include <sys/socket.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <netinet/in.h>
#include <unistd.h>
#include <netdb.h>
#include <time.h>
#include <fcntl.h>
#include <signal.h>
#include <signal.h>
#include <linux/ip.h> /*<netinet/ip.h>*/
#include <linux/icmp.h> /*<netinet/ip_icmp.h>*/
#include <arpa/inet.h>
#include <math.h>
#include <time.h>
#include <sys/time.h>
#include <asm/byteorder.h>
#include <netinet/ip_tcp.h>

/******DEFINES******/

/* #define to zero if you don't want to ignore hosts of the form
   xxx.xxx.xxx.{0,255} (usually network and broadcast addresses) */
#define IGNORE_ZERO_AND_255_HOSTS 1

#define DEBUGGING 0

/* Default number of ports in paralell. Doesn't always involve actual
   sockets. Can also adjust with the -M command line option. */
#define MAX_SOCKETS 36
/* If reads of a UDP port keep returning EAGAIN (errno 13), do we want to
   count the port as valid? */
#define RISKY_UDP_SCAN 0
/* This ideally should be a port that isn't in use for any protocol on our machine or on
   the target */
#define MAGIC_PORT 49724
/* How many udp sends without a ICMP port unreachable error does it take before we consider
   the port open? */
#define UDP_MAX_PORT_RETRIES 4
/*How many seconds before we give up on a host being alive? */
#define PING_TIMEOUT 2
#define FAKE_ARGV "pine" /* What ps and w should show if you use -q */
/* How do we want to log into ftp sites for */
#define FTPUSER "anonymous"
#define FTPPASS "-wwwuser@"
#define FTP_RETRIES 2 /* How many times should we relogin if we lose control
   connection? */

#define UC(b) (((int)b)&0xff)
#define MORE_FRAGMENTS 8192 /*NOT a user serviceable parameter*/
#define fatal(x) { fprintf(stderr, "%s\n", x); exit(-1); }
#define error(x) fprintf(stderr, "%s\n", x);

/******STRUCTURES******/

typedef struct port {
    unsigned short portno;
```

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

```
unsigned char proto;
char *owner;
struct port *next;
} port;

struct ftpinfo {
    char user[64];
    char pass[256]; /* methinks you're paranoid if you need this much space */
    char server_name[MAXHOSTNAMELEN + 1];
    struct in_addr server;
    unsigned short port;
    int sd; /* socket descriptor */
};

typedef port *portlist;

/*****PROTOTYPES*****/

/* print usage information */
void printusage(char *name);

/* our scanning functions */
portlist tcp_scan(struct in_addr target, unsigned short *portarray,
                 portlist *ports);
portlist syn_scan(struct in_addr target, unsigned short *portarray,
                 struct in_addr *source, int fragment, portlist *ports);
portlist fin_scan(struct in_addr target, unsigned short *portarray,
                 struct in_addr *source, int fragment, portlist *ports);
portlist udp_scan(struct in_addr target, unsigned short *portarray,
                 portlist *ports);
portlist lamer_udp_scan(struct in_addr target, unsigned short *portarray,
                       portlist *ports);
portlist bounce_scan(struct in_addr target, unsigned short *portarray,
                    struct ftpinfo *ftp, portlist *ports);

/* Scan helper functions */
unsigned long calculate_sleep(struct in_addr target);
int check_ident_port(struct in_addr target);
int getidentinfoz(struct in_addr target, int localport, int remoteport,
                 char *owner);
int parse_bounce(struct ftpinfo *ftp, char *url);
int ftp_anon_connect(struct ftpinfo *ftp);

/* port manipulators */
unsigned short *getpts(char *expr); /* someone stole the name getports()! */
unsigned short *getfastports(int tcpscan, int udpscan);
int addport(portlist *ports, unsigned short portno, unsigned short protocol,
           char *owner);
int deleteport(portlist *ports, unsigned short portno, unsigned short protocol);
void printandfreeports(portlist ports);
int shortfry(unsigned short *ports);

/* socket manipulation functions */
void init_socket(int sd);
int unblock_socket(int sd);
int block_socket(int sd);
int recvtime(int sd, char *buf, int len, int seconds);

/* RAW packet building/dissassembling stuff */
int send_tcp_raw( int sd, struct in_addr *source,
                 struct in_addr *victim, unsigned short sport,
                 unsigned short dport, unsigned long seq,
                 unsigned long ack, unsigned char flags,
                 unsigned short window, char *data,
```

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

```
    unsigned short datalen);
int isup(struct in_addr target);
unsigned short in_cksum(unsigned short *ptr,int nbytes);
int send_small_fragz(int sd, struct in_addr *source, struct in_addr *victim,
    int sport, int dport, int flags);
int readtcppacket(char *packet, int readdata);
int listen_icmp(int icmpsock, unsigned short outports[],
    unsigned short numtries[], int *num_out,
    struct in_addr target, portlist *ports);
```

```
/* general helper functions */
void hdump(unsigned char *packet, int len);
void *safe_malloc(int size);
#endif /* NMAP_H */
<-->
```

```
<+> nmap/nmap.c
```

```
#include "nmap.h"
```

```
/* global options */
short debugging = DEBUGGING;
short verbose = 0;
int number_of_ports = 0; /* How many ports do we scan per machine? */
int max_parallel_sockets = MAX_SOCKETS;
extern char *optarg;
extern int optind;
short isr00t = 0;
short identscan = 0;
char current_name[MAXHOSTNAMELEN + 1];
unsigned long global_delay = 0;
unsigned long global_rtt = 0;
struct in_addr ouraddr = { 0 };
```

```
int main(int argc, char *argv[]) {
int i, j, arg, argvlen;
short fastscan=0, tcpscan=0, udpscan=0, synscan=0, randomize=0;
short fragscan = 0, finscan = 0, quashargv = 0, pingscan = 0, lamerscan = 0;
short bouncescan = 0;
short *ports = NULL, mask;
struct ftpinfo ftp = { FTPUSER, FTPPASS, "", { 0 }, 21, 0};
portlist openports = NULL;
struct hostent *target = 0;
unsigned long int lastip, currentip, longtmp;
char *target_net, *p;
struct in_addr current_in, *source=NULL;
int hostup = 0;
char *fakeargv[argc + 1];
```

```
/* argv faking silliness */
for(i=0; i < argc; i++) {
    fakeargv[i] = safe_malloc(strlen(argv[i]) + 1);
    strncpy(fakeargv[i], argv[i], strlen(argv[i]) + 1);
}
fakeargv[argc] = NULL;
```

```
if (argc < 2 ) printusage(argv[0]);
```

```
/* OK, lets parse these args! */
while((arg = getopt(argc,fakeargv,"b:dFfhiLM:Pp:qrS:stUuw:v")) != EOF) {
    switch(arg) {
        case 'b':
            bouncescan++;
            if (parse_bounce(&ftp, optarg) < 0 ) {
```

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

fprintf(stderr, "Your argument to -b is fucked up. Use the normal url style:

user:pass@server:port or just use server and use default anon login\n Use -h for help\n");

```
    }
    break;
case 'd': debugging++; break;
case 'F': fastscan++; break;
case 'f': fragscan++; break;
case 'h':
case '?': printusage(argv[0]);
case 'i': identscan++; break;
case 'l': lamerscan++; udpscan++; break;
case 'M': max_parallel_sockets = atoi(optarg); break;
case 'P': pingscan++; break;
case 'p':
    if (ports)
        fatal("Only 1 -p option allowed, seperate multiple ranges with commas.");
    ports = getpts(optarg); break;
case 'r': randomize++; break;
case 's': synscan++; break;
case 'S':
    if (source)
        fatal("You can only use the source option once!\n");
    source = safe_malloc(sizeof(struct in_addr));
    if (!inet_aton(optarg, source))
        fatal("You must give the source address in dotted deciman, currently.\n");
    break;
case 't': tcpscan++; break;
case 'U': finscan++; break;
case 'u': udpscan++; break;
case 'q': quashargv++; break;
case 'w': global_delay = atoi(optarg); break;
case 'v': verbose++;
}
}

/* Take care of user wierdness */
isr00t = !(geteuid()|geteuid());
if (tcpscan && synscan)
    fatal("The -t and -s options can't be used together.\
If you are trying to do TCP SYN scanning, just use -s.\
For normal connect() style scanning, use -t");
if ((synscan || finscan || fragscan || pingscan) && !isr00t)
    fatal("Options specified require r00t privileges. You don't have them!");
if (!tcpscan && !udpscan && !synscan && !finscan && !bouncescan && !pingscan) {
    tcpscan++;
    if (verbose) error("No scantype specified, assuming vanilla tcp connect()\
scan. Use -P if you really don't want to portscan.");
if (fastscan && ports)
    fatal("You can use -F (fastscan) OR -p for explicit port specification.\
Not both!\n");
}
/* If he wants to bounce of an ftp site, that site better damn well be reachable! */
if (bouncescan) {
    if (!inet_aton(ftp.server_name, &ftp.server)) {
        if ((target = gethostbyname(ftp.server_name)))
            memcpy(&ftp.server, target->h_addr_list[0], 4);
        else {
            fprintf(stderr, "Failed to resolve ftp bounce proxy hostname/IP: %s\n",
                ftp.server_name);
            exit(1);
        }
    }
} else if (verbose)
    printf("Resolved ftp bounce attack proxy to %s (%s).\n",
        target->h_name, inet_ntoa(ftp.server));
```

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

```
}
printf("\nStarting nmap V 1.21 by Fyodor (fyodor@dhp.com, www.dhp.com/~fyodor/nmap/\n");
if (!verbose)
    error("Hint: The -v option notifies you of open ports as they are found.\n");
if (fastscan)
    ports = getfastports(synscan|tcpscan|fragscan|finscan|bouncescan,
                        udpscan|lamerscan);
if (!ports) ports = getpts("1-1024");

/* more fakeargv junk, BTW malloc'ing extra space in argv[0] doesn't work */
if (quashargv) {
    argvlen = strlen(argv[0]);
    if (argvlen < strlen(FAKE_ARGV))
        fatal("If you want me to fake your argv, you need to call the program with a longer
name. Try the full pathname, or rename it fyodorssuperdedouperportscanner");
    strncpy(argv[0], FAKE_ARGV, strlen(FAKE_ARGV));
    for(i = strlen(FAKE_ARGV); i < argvlen; i++) argv[0][i] = '\0';
    for(i=1; i < argc; i++) {
        argvlen = strlen(argv[i]);
        for(j=0; j <= argvlen; j++)
            argv[i][j] = '\0';
    }
}

srand(time(NULL));

while(optind < argc) {

    /* Time to parse the allowed mask */
    target = NULL;
    target_net = strtok(strdup(fakeargv[optind]), "/");
    mask = (p = strtok(NULL, "")) ? atoi(p) : 32;
    if (debugging)
        printf("Target network is %s, scanmask is %d\n", target_net, mask);

    if (!inet_aton(target_net, &current_in)) {
        if ((target = gethostbyname(target_net)))
            memcpy(&currentip, target->h_addr_list[0], 4);
        else {
            fprintf(stderr, "Failed to resolve given hostname/IP: %s\n", target_net);
        }
    } else currentip = current_in.s_addr;

    longtmp = ntohl(currentip);
    currentip = longtmp & (unsigned long) (0 - pow(2,32 - mask));
    lastip = longtmp | (unsigned long) (pow(2,32 - mask) - 1);
    while (currentip <= lastip) {
        openports = NULL;
        longtmp = htonl(currentip);
        target = gethostbyaddr((char *) &longtmp, 4, AF_INET);
        current_in.s_addr = longtmp;
        if (target)
            strncpy(current_name, target->h_name, MAXHOSTNAMELEN);
        else current_name[0] = '\0';
        current_name[MAXHOSTNAMELEN + 1] = '\0';
        if (randomize)
            shortfry(ports);
#ifdef IGNORE_ZERO_AND_255_HOSTS
        if (IGNORE_ZERO_AND_255_HOSTS
            && (!(currentip % 256) || currentip % 256 == 255))
            {
                printf("Skipping host %s because IGNORE_ZERO_AND_255_HOSTS is set in the source.\n",
inet_ntoa(current_in));
                hostup = 0;
            }
#endif
    }
}
```

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

```
    }
    else{
#endif
    if (isr00t) {
    if (!(hostup = isup(current_in))) {
    if (!pingscan)
    printf("Host %s (%s) appears to be down, skipping scan.\n",
    current_name, inet_ntoa(current_in));
    else
    printf("Host %s (%s) appears to be down\n",
    current_name, inet_ntoa(current_in));
    } else if (debugging || pingscan)
    printf("Host %s (%s) appears to be up ... good.\n",
    current_name, inet_ntoa(current_in));
    }
    else hostup = 1; /* We don't really check because the lamer isn't root.*/
    }

    /* Time for some actual scanning! */
    if (hostup) {
    if (tcpscan) tcp_scan(current_in, ports, &openports);

    if (synscan) syn_scan(current_in, ports, source, fragscan, &openports);

    if (finscan) fin_scan(current_in, ports, source, fragscan, &openports);

    if (bouncescan) {
    if (ftp.sd <= 0) ftp_anon_connect(&ftp);
    if (ftp.sd > 0) bounce_scan(current_in, ports, &ftp, &openports);
    }
    if (udpscan) {
    if (!isr00t || lamerscan)
    lamer_udp_scan(current_in, ports, &openports);

    else udp_scan(current_in, ports, &openports);
    }

    if (!openports && !pingscan)
    printf("No ports open for host %s (%s)\n", current_name,
    inet_ntoa(current_in));
    if (openports) {
    printf("Open ports on %s (%s):\n", current_name,
    inet_ntoa(current_in));
    printandfreeports(openports);
    }
    }
    currentip++;
    }
    optind++;
}

return 0;
}

__inline__ int unblock_socket(int sd) {
int options;
/*Unblock our socket to prevent recvfrom from blocking forever
on certain target ports. */
options = O_NONBLOCK | fcntl(sd, F_GETFL);
fcntl(sd, F_SETFL, options);
return 1;
}

__inline__ int block_socket(int sd) {
```

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

```
int options;
options = (~O_NONBLOCK) & fcntl(sd, F_GETFL);
fcntl(sd, F_SETFL, options);
return 1;
}

/* Currently only sets SO_LINGER, I haven't seen any evidence that this
   helps. I'll do more testing before dumping it. */
__inline__ void init_socket(int sd) {
struct linger l;

l.l_onoff = 1;
l.l_linger = 0;

if (setsockopt(sd, SOL_SOCKET, SO_LINGER, &l, sizeof(struct linger))
    {
    fprintf(stderr, "Problem setting socket SO_LINGER, errno: %d\n", errno);
    perror("setsockopt");
    }
}

/* Convert a string like "-100,200-1024,3000-4000,60000-" into an array
   of port numbers*/
unsigned short *getpts(char *origexpr) {
int exlen = strlen(origexpr);
char *p,*q;
unsigned short *tmp, *ports;
int i=0, j=0,start,end;
char *expr = strdup(origexpr);
ports = safe_malloc(65536 * sizeof(short));
i++;
i--;
for(;j < exlen; j++)
    if (expr[j] != ' ') expr[i++] = expr[j];
expr[i] = '\0';
exlen = i + 1;
i=0;
while((p = strchr(expr',')) {
    *p = '\0';
    if (*expr == '-') {start = 1; end = atoi(expr+ 1);}
    else {
        start = end = atoi(expr);
        if ((q = strchr(expr,'-')) && *(q+1) ) end = atoi(q + 1);
        else if (q && !(q+1)) end = 65535;
    }
    if (debugging)
        printf("The first port is %d, and the last one is %d\n", start, end);
    if (start < 1 || start > end) fatal("Your port specifications are illegal!");
    for(j=start; j <= end; j++)
        ports[i++] = j;
    expr = p + 1;
}
if (*expr == '-') {
    start = 1;
    end = atoi(expr+ 1);
}
else {
    start = end = atoi(expr);
    if ((q = strchr(expr,'-')) && *(q+1) ) end = atoi(q+1);
    else if (q && !(q+1)) end = 65535;
}
if (debugging)
    printf("The first port is %d, and the last one is %d\n", start, end);
if (start < 1 || start > end) fatal("Your port specifications are illegal!");
```

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

```
for(j=start; j <= end; j++)
    ports[i++] = j;
number_of_ports = i;
ports[i++] = 0;
tmp = realloc(ports, i * sizeof(short));
    free(expr);
    return tmp;
}

unsigned short *getfastports(int tcpscan, int udpscan) {
    int portindex = 0, res, lastport = 0;
    unsigned int portno = 0;
    unsigned short *ports;
    char proto[10];
    char line[81];
    FILE *fp;
    ports = safe_malloc(65535 * sizeof(unsigned short));
    proto[0] = '\0';
    if (!(fp = fopen("/etc/services", "r"))) {
        printf("We can't open /etc/services for reading! Fix your system or don't use -f\n");
        perror("fopen");
        exit(1);
    }

    while(fgets(line, 80, fp)) {
        res = sscanf(line, "%*s %u/%s", &portno, proto);
        if (res == 2 && portno != 0 && portno != lastport) {
            lastport = portno;
            if (tcpscan && proto[0] == 't')
                ports[portindex++] = portno;
            else if (udpscan && proto[0] == 'u')
                ports[portindex++] = portno;
        }
    }

    number_of_ports = portindex;
    ports[portindex++] = 0;
    return realloc(ports, portindex * sizeof(unsigned short));
}

void printusage(char *name) {
    printf("%s [options] [hostname[/mask] . . .]
options (none are required, most can be combined):
    -t tcp connect() port scan
    -s tcp SYN stealth port scan (must be root)
    -u UDP port scan, will use MUCH better version if you are root
    -U Uriel Maimon (P49-15) style FIN stealth scan.
    -l Do the lamer UDP scan even if root. Less accurate.
    -P ping \"scan\". Find which hosts on specified network(s) are up.
    -b <ftp_relay_host> ftp \"bounce attack\" port scan
    -f use tiny fragmented packets for SYN or FIN scan.
    -i Get identd (rfc 1413) info on listening TCP processes.
    -p <range> ports: ex: \"-p 23\" will only try port 23 of the host(s)
        \"-p 20-30,63000-\" scans 20-30 and 63000-65535 default: 1-1024
    -F fast scan. Only scans ports in /etc/services, a la strobe(1).
    -r randomize target port scanning order.
    -h help, print this junk. Also see http://www.dhp.com/~fyodor/nmap/
    -S If you want to specify the source address of SYN or FYN scan.
    -v Verbose. Its use is recommended. Use twice for greater effect.
    -w <n> delay. n microsecond delay. Not recommended unless needed.
    -M <n> maximum number of parallel sockets. Larger isn't always better.
    -q quash argv to something benign, currently set to \"%s\".
```

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

Hostnames specified as internet hostname or IP address. Optional '/mask' specifies subnet. cert.org/24 or 192.88.209.5/24 scan CERT's Class C.\n",

```
    name, FAKE_ARGV);
exit(1);
}

portlist tcp_scan(struct in_addr target, unsigned short *portarray, portlist *ports) {

int starttime, current_out = 0, res, deadindex = 0, i=0, j=0, k=0, max=0;
struct sockaddr_in sock, stranger, mysock;
int sockaddr_in_len = sizeof(struct sockaddr_in);
int sockets[max_parallel_sockets], deadstack[max_parallel_sockets];
unsigned short portno[max_parallel_sockets];
char owner[513], buf[65536];
int tryident = identscan, current_socket /*actually it is a socket INDEX*/;
fd_set fds_read, fds_write;
struct timeval nowait = {0,0}, longwait = {7,0};

signal(SIGPIPE, SIG_IGN); /* ignore SIGPIPE so our 'write 0 bytes' test
                           doesn't crash our program!*/

owner[0] = '\0';
starttime = time(NULL);
bzero((char *)&sock, sizeof(struct sockaddr_in));
sock.sin_addr.s_addr = target.s_addr;
if (verbose || debugging)
    printf("Initiating TCP connect() scan against %s (%s)\n",
           current_name, inet_ntoa(sock.sin_addr));
sock.sin_family=AF_INET;
FD_ZERO(&fds_read);
FD_ZERO(&fds_write);

if (tryident)
    tryident = check_ident_port(target);

/* Initially, all of our sockets are "dead" */
for(i = 0 ; i < max_parallel_sockets; i++) {
    deadstack[deadindex++] = i;
    portno[i] = 0;
}

deadindex--;
/* deadindex always points to the most recently added dead socket index */

while(portarray[j]) {
    longwait.tv_sec = 7;
    longwait.tv_usec = nowait.tv_sec = nowait.tv_usec = 0;

    for(i=current_out; i < max_parallel_sockets && portarray[j]; i++, j++) {
        current_socket = deadstack[deadindex--];
        if ((sockets[current_socket] = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) == -1)
            {perror("Socket troubles"); exit(1);}
        if (sockets[current_socket] > max) max = sockets[current_socket];
        current_out++;
        unblock_socket(sockets[current_socket]);
        init_socket(sockets[current_socket]);
        portno[current_socket] = portarray[j];
        sock.sin_port = htons(portarray[j]);
        if ((res = connect(sockets[current_socket],(struct sockaddr *)&sock,sizeof(struct
sockaddr)))!=-1)
            printf("WTF???? I think we got a successful connection in non-blocking!!@#$\n");
        else {
            switch(errno) {
                case EINPROGRESS: /* The one I always see */
                case EAGAIN:
```

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

```
block_socket(sockets[current_socket]);
FD_SET(sockets[current_socket], &fds_write);
FD_SET(sockets[current_socket], &fds_read);
break;
default:
    printf("Strange error from connect: (%d)", errno);
    perror(""); /*falling through intentionally*/
case ECONNREFUSED:
    if (max == sockets[current_socket]) max--;
    deadstack[++deadindex] = current_socket;
    current_out--;
    portno[current_socket] = 0;
    close(sockets[current_socket]);
    break;
}
}
}
if (!portarray[j]) sleep(1); /*wait a second for any last packets*/
while((res = select(max + 1, &fds_read, &fds_write, NULL,
    (current_out < max_parallel_sockets)?
    &nowait : &longwait)) > 0) {
for(k=0; k < max_parallel_sockets; k++)
    if (portno[k]) {
        if (FD_ISSET(sockets[k], &fds_write)
            && FD_ISSET(sockets[k], &fds_read)) {
            /*printf("Socket at port %hi is selectable for r & w.", portno[k]);*/
            res = recvfrom(sockets[k], buf, 65536, 0, (struct sockaddr *)
                &stranger, &sockaddr_in_len);
            if (res >= 0) {
                if (debugging || verbose)
                    printf("Adding TCP port %hi due to successful read.\n",
                        portno[k]);
                if (tryident) {
                    if ( getsockname(sockets[k], (struct sockaddr *) &mysock,
                        &sockaddr_in_len ) ) {
                        perror("getsockname");
                        exit(1);
                    }
                    tryident = getidentinfoz(target, ntohs(mysock.sin_port),
                        portno[k], owner);
                }
                addport(ports, portno[k], IPPROTO_TCP, owner);
            }
        }
        if (max == sockets[k])
            max--;
        FD_CLR(sockets[k], &fds_read);
        FD_CLR(sockets[k], &fds_write);
        deadstack[++deadindex] = k;
        current_out--;
        portno[k] = 0;
        close(sockets[k]);
    }
    else if(FD_ISSET(sockets[k], &fds_write)) {
        /*printf("Socket at port %hi is selectable for w only.VERIFYING\n",
            portno[k]);*/
        res = send(sockets[k], buf, 0, 0);
        if (res < 0 ) {
            signal(SIGPIPE, SIG_IGN);
            if (debugging > 1)
                printf("Bad port %hi caught by 0-byte write!\n", portno[k]);
        }
        else {
            if (debugging || verbose)
                printf("Adding TCP port %hi due to successful 0-byte write!\n",
```

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

```
    portno[k]);
    if (tryident) {
        if ( getsockname(sockets[k], (struct sockaddr *) &mysock ,
            &sockaddr_in_len ) ) {
            perror("getsockname");
            exit(1);
        }
        tryident = getidentinfoz(target, ntohs(mysock.sin_port),
            portno[k], owner);
    }
    addport(ports, portno[k], IPPROTO_TCP, owner);
}
if (max == sockets[k]) max--;
FD_CLR(sockets[k], &fds_write);
deadstack[++deadindex] = k;
current_out--;
portno[k] = 0;
close(sockets[k]);
}
else if ( FD_ISSET(sockets[k], &fds_read) ) {
    printf("Socket at port %hi is selectable for r only.  This is very wierd.\n",
portno[k]);
    if (max == sockets[k]) max--;
    FD_CLR(sockets[k], &fds_read);
    deadstack[++deadindex] = k;
    current_out--;
    portno[k] = 0;
    close(sockets[k]);
}
else {
    /*printf("Socket at port %hi not selecting, reading.\n",portno[k]);*/
    FD_SET(sockets[k], &fds_write);
    FD_SET(sockets[k], &fds_read);
}
}
}
}

if (debugging || verbose)
    printf("Scanned %d ports in %ld seconds with %d parallel sockets.\n",
        number_of_ports, time(NULL) - starttime, max_parallel_sockets);
return *ports;
}

/* gawd, my next project will be in c++ so I don't have to deal with
   this crap ... simple linked list implementation */
int addport(portlist *ports, unsigned short portno, unsigned short protocol,
    char *owner) {
    struct port *current, *tmp;
    int len;

    if (*ports) {
        current = *ports;
        /* case 1: we add to the front of the list */
        if (portno <= current->portno) {
            if (current->portno == portno && current->proto == protocol) {
                if (debugging || verbose)
                    printf("Duplicate port (%hi/%s)\n", portno ,
                        (protocol == IPPROTO_TCP)? "tcp": "udp");
                return -1;
            }
            tmp = current;
            *ports = safe_malloc(sizeof(struct port));
            (*ports)->next = tmp;
        }
    }
}
```

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

```
current = *ports;
current->portno = portno;
current->proto = protocol;
if (owner && *owner) {
    len = strlen(owner);
    current->owner = malloc(sizeof(char) * (len + 1));
    strncpy(current->owner, owner, len + 1);
}
else current->owner = NULL;
}
else { /* case 2: we add somewhere in the middle or end of the list */
while( current->next && current->next->portno < portno)
    current = current->next;
if (current->next && current->next->portno == portno
    && current->next->proto == protocol) {
    if (debugging || verbose)
        printf("Duplicate port (%hi/%s)\n", portno ,
            (protocol == IPPROTO_TCP)? "tcp": "udp");
    return -1;
}
tmp = current->next;
current->next = safe_malloc(sizeof(struct port));
current->next->next = tmp;
tmp = current->next;
tmp->portno = portno;
tmp->proto = protocol;
if (owner && *owner) {
    len = strlen(owner);
    tmp->owner = malloc(sizeof(char) * (len + 1));
    strncpy(tmp->owner, owner, len + 1);
}
else tmp->owner = NULL;
}
}
}

else { /* Case 3, list is null */
*ports = safe_malloc(sizeof(struct port));
tmp = *ports;
tmp->portno = portno;
tmp->proto = protocol;
if (owner && *owner) {
    len = strlen(owner);
    tmp->owner = safe_malloc(sizeof(char) * (len + 1));
    strncpy(tmp->owner, owner, len + 1);
}
else tmp->owner = NULL;
tmp->next = NULL;
}
return 0; /*success */
}

int deleteport(portlist *ports, unsigned short portno,
    unsigned short protocol) {
portlist current, tmp;

if (!*ports) {
    if (debugging > 1) error("Tried to delete from empty port list!");
    return -1;
}
/* Case 1, deletion from front of list*/
if ((*ports)->portno == portno && (*ports)->proto == protocol) {
    tmp = (*ports)->next;
    if ((*ports)->owner) free((*ports)->owner);
    free(*ports);
}
```

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

```
*ports = tmp;
}
else {
    current = *ports;
    for(;current->next && (current->next->portno != portno || current->next->proto !=
protocol); current = current->next);
    if (!current->next)
        return -1;
    tmp = current->next;
    current->next = tmp->next;
    if (tmp->owner) free(tmp->owner);
    free(tmp);
}
return 0; /* success */
}

void *safe_malloc(int size)
{
    void *mymem;
    if (size < 0)
        fatal("Tried to malloc negative amount of memmory!!!");
    if ((mymem = malloc(size)) == NULL)
        fatal("Malloc Failed! Probably out of space.");
    return mymem;
}

void printandfreeports(portlist ports) {
    char protocol[4];
    struct servent *service;
    port *current = ports, *tmp;

    printf("Port Number Protocol Service");
    printf("%s", (identscan)? "          Owner\n":"\n");
    while(current != NULL) {
        strcpy(protocol,(current->proto == IPPROTO_TCP)? "tcp": "udp");
        service = getservbyport(htons(current->portno), protocol);
        printf("%-13d%-11s%-16s%\n", current->portno, protocol,
            (service)? service->s_name: "unknown",
            (current->owner)? current->owner : "");
        tmp = current;
        current = current->next;
        if (tmp->owner) free(tmp->owner);
        free(tmp);
    }
    printf("\n");
}

/* This is the version of udp_scan that uses raw ICMP sockets and requires
root priviliges.*/
portlist udp_scan(struct in_addr target, unsigned short *portarray,
    portlist *ports) {
    int icmpsock, udpsock, tmp, done=0, retries, bytes = 0, res, num_out = 0;
    int i=0,j=0, k=0, icmperrlimittime, max_tries = UDP_MAX_PORT_RETRIES;
    unsigned short outports[max_parallel_sockets], numtries[max_parallel_sockets];
    struct sockaddr_in her;
    char senddata[] = "blah\n";
    unsigned long starttime, sleeptime;
    struct timeval shortwait = {1, 0 };
    fd_set  fds_read, fds_write;

    bzero(outports, max_parallel_sockets * sizeof(unsigned short));
    bzero(numtries, max_parallel_sockets * sizeof(unsigned short));
```

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

```
/* Some systems (like linux) follow the advice of rfc1812 and limit
 * the rate at which they will respond with icmp error messages
 * (like port unreachable). icmperrlimittime is to compensate for that.
 */
icmperrlimittime = 60000;

sleeptime = (global_delay)? global_delay : (global_rtt)? (1.2 * global_rtt) + 30000 :
1e5;
if (global_delay) icmperrlimittime = global_delay;

starttime = time(NULL);

FD_ZERO(&fds_read);
FD_ZERO(&fds_write);

if (verbose || debugging)
    printf("Initiating UDP (raw ICMP version) scan against %s (%s) using wait delay of %li
usecs.\n", current_name, inet_ntoa(target), sleeptime);

if ((icmpsock = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP)) < 0)
    perror("Opening ICMP RAW socket");
if ((udpsock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)
    perror("Opening datagram socket");

unblock_socket(icmpsock);
her.sin_addr = target;
her.sin_family = AF_INET;

while(!done) {
    tmp = num_out;
    for(i=0; (i < max_parallel_sockets && portarray[j]) || i < tmp; i++) {
        close(udpsock);
        if ((udpsock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)
            perror("Opening datagram socket");
        if ((i > tmp && portarray[j]) || numtries[i] > 1) {
            if (i > tmp) her.sin_port = htons(portarray[j++]);
            else her.sin_port = htons(outports[i]);
            FD_SET(udpsock, &fds_write);
            FD_SET(icmpsock, &fds_read);
            shortwait.tv_sec = 1; shortwait.tv_usec = 0;
            usleep(icmperrlimittime);
            res = select(udpsock + 1, NULL, &fds_write, NULL, &shortwait);
            if (FD_ISSET(udpsock, &fds_write))
                bytes = sendto(udpsock, senddata, sizeof(senddata), 0,
                    (struct sockaddr *) &her, sizeof(struct sockaddr_in));
            else {
                printf("udpsock not set for writing port %d!", ntohs(her.sin_port));
                return *ports;
            }
        }
        if (bytes <= 0) {
            if (errno == ECONNREFUSED) {
                retries = 10;
                do {
                    /* This is from when I was using the same socket and would
                     * (rather often) get strange connection refused errors, it
                     * shouldn't happen now that I create a new udp socket for each
                     * port. At some point I will probably go back to 1 socket again.
                     */
                    printf("sendto said connection refused on port %d but trying again anyway.\n",
                        ntohs(her.sin_port));
                    usleep(icmperrlimittime);
                    bytes = sendto(udpsock, senddata, sizeof(senddata), 0,
                        (struct sockaddr *) &her, sizeof(struct sockaddr_in));
                    printf("This time it returned %d\n", bytes);
                } while (retries-- > 0);
            }
        }
    }
}
```

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

```
    } while(bytes <= 0 && retries-- > 0);
  }
  if (bytes <= 0) {
    printf("sendto returned %d.", bytes);
    fflush(stdout);
    perror("sendto");
  }
}
if (bytes > 0 && i > tmp) {
  num_out++;
  outports[i] = portarray[j-1];
}
}
}
usleep(sleeptime);
tmp = listen_icmp(icmpsock, outports, numtries, &num_out, target, ports);
if (debugging) printf("listen_icmp caught %d bad ports.\n", tmp);
done = !portarray[j];
for (i=0,k=0; i < max_parallel_sockets; i++)
  if (outports[i]) {
    if (++numtries[i] > max_tries - 1) {
      if (debugging || verbose)
        printf("Adding port %d for 0 unreachable port generations\n",
              outports[i]);
      addport(ports, outports[i], IPPROTO_UDP, NULL);
      num_out--;
      outports[i] = numtries[i] = 0;
    }
    else {
      done = 0;
      outports[k] = outports[i];
      numtries[k] = numtries[i];
      if (k != i)
        outports[i] = numtries[i] = 0;
      k++;
    }
  }
}
if (num_out == max_parallel_sockets) {
  printf("Numout is max sockets, that is a problem!\n");
  sleep(1); /* Give some time for responses to trickle back,
            and possibly to reset the hosts ICMP error limit */
}
}
}

if (debugging || verbose)
  printf("The UDP raw ICMP scanned %d ports in %ld seconds with %d parallel sockets.\n",
        number_of_ports, time(NULL) - starttime, max_parallel_sockets);
close(icmpsock);
close(udpsock);
return *ports;
}

int listen_icmp(int icmpsock, unsigned short outports[],
               unsigned short numtries[], int *num_out, struct in_addr target,
               portlist *ports) {
  char response[1024];
  struct sockaddr_in stranger;
  int sockaddr_in_size = sizeof(struct sockaddr_in);
  struct in_addr bs;
  struct iphdr *ip = (struct iphdr *) response;
  struct icmp_hdr *icmp = (struct icmp_hdr *) (response + sizeof(struct iphdr));
  struct iphdr *ip2;
  unsigned short *data;
```

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

```
int badport, numcaught=0, bytes, i, tmptry=0, found=0;

while ((bytes = recvfrom(icmpsock, response, 1024, 0,
                        (struct sockaddr *) &stranger,
                        &sockaddr_in_size)) > 0) {
    numcaught++;
    bs.s_addr = ip->saddr;
    if (ip->saddr == target.s_addr && ip->protocol == IPPROTO_ICMP
        && icmp->type == 3 && icmp->code == 3) {
        ip2 = (struct iphdr *) (response + 4 * ip->ihl + sizeof(struct icmp_hdr));
        data = (unsigned short *) ((char *)ip2 + 4 * ip2->ihl);
        badport = ntohs(data[1]);
        /*delete it from our outports array */
        found = 0;
        for(i=0; i < max_parallel_sockets; i++)
            if (outports[i] == badport) {
                found = 1;
                tmptry = numtries[i];
                outports[i] = numtries[i] = 0;
                (*num_out)--;
                break;
            }
        if (debugging && found && tmptry > 0)
            printf("Badport: %d on try number %d\n", badport, tmptry);
        if (!found) {
            if (debugging)
                printf("Badport %d came in late, deleting from portlist.\n", badport);
            if (deleteport(ports, badport, IPPROTO_UDP) < 0)
                if (debugging) printf("Port deletion failed.\n");
        }
    }
    else {
        printf("Fucked up packet!\n");
    }
}
return numcaught;
}

/* This fuction is nonsens. I wrote it all, really optimized etc. Then
found out that many hosts limit the rate at which they send icmp errors :(
I will probably totally rewrite it to be much simpler at some point. For
now I won't worry about it since it isn't a very important functions (UDP
is lame, plus there is already a much better function for people who
are r00t */
portlist lamer_udp_scan(struct in_addr target, unsigned short *portarray,
                       portlist *ports) {
    int sockaddr_in_size = sizeof(struct sockaddr_in), i=0, j=0, k=0, bytes;
    int sockets[max_parallel_sockets], trynum[max_parallel_sockets];
    unsigned short portno[max_parallel_sockets];
    int last_open = 0;
    char response[1024];
    struct sockaddr_in her, stranger;
    char data[] = "\nhelp\nquit\n";
    unsigned long sleeptime;
    unsigned int starttime;

    /* Initialize our target sockaddr_in */
    bzero((char *) &her, sizeof(struct sockaddr_in));
    her.sin_family = AF_INET;
    her.sin_addr = target;

    if (global_delay) sleeptime = global_delay;
    else sleeptime = calculate_sleep(target) + 60000; /*large to be on the
                                                    safe side */
}
```

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

```
if (verbose || debugging)
    printf("Initiating UDP scan against %s (%s), sleeptime: %li\n", current_name,
        inet_ntoa(target), sleeptime);

starttime = time(NULL);

for(i = 0 ; i < max_parallel_sockets; i++)
    trynum[i] = portno[i] = 0;

while(portarray[j]) {
    for(i=0; i < max_parallel_sockets && portarray[j]; i++, j++) {
        if (i >= last_open) {
            if ((sockets[i] = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1)
                {perror("datagram socket troubles"); exit(1);}
            block_socket(sockets[i]);
            portno[i] = portarray[j];
        }
        her.sin_port = htons(portarray[j]);
        bytes = sendto(sockets[i], data, sizeof(data), 0, (struct sockaddr *) &her,
            sizeof(struct sockaddr_in));
        usleep(5000);
        if (debugging > 1)
            printf("Sent %d bytes on socket %d to port %hi, try number %d.\n",
                bytes, sockets[i], portno[i], trynum[i]);
        if (bytes < 0 ) {
            printf("Sendto returned %d the FIRST TIME!@#$, errno %d\n", bytes,
                errno);
            perror("");
            trynum[i] = portno[i] = 0;
            close(sockets[i]);
        }
    }
    last_open = i;
    /* Might need to change this to 1e6 if you are having problems*/
    usleep(sleeptime + 5e5);
    for(i=0; i < last_open ; i++) {
        if (portno[i]) {
            unblock_socket(sockets[i]);
            if ((bytes = recvfrom(sockets[i], response, 1024, 0,
                (struct sockaddr *) &stranger,
                &sockaddr_in_size)) == -1)
            {
                if (debugging > 1)
                    printf("2nd recvfrom on port %d returned %d with errno %d.\n",
                        portno[i], bytes, errno);
                if (errno == EAGAIN /*11*/)
                {
                    if (trynum[i] < 2) trynum[i]++;
                    else {
                        if (RISKY_UDP_SCAN) {
                            printf("Adding port %d after 3 EAGAIN errors.\n", portno[i]);
                            addport(ports, portno[i], IPPROTO_UDP, NULL);
                        }
                        else if (debugging)
                            printf("Skipping possible false positive, port %d\n",
                                portno[i]);
                        trynum[i] = portno[i] = 0;
                        close(sockets[i]);
                    }
                }
            }
            else if (errno == ECONNREFUSED /*111*/) {
                if (debugging > 1)
                    printf("Closing socket for port %d, ECONNREFUSED received.\n",
```

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

```
        portno[i]);
        trynum[i] = portno[i] = 0;
        close(sockets[i]);
    }
    else {
        printf("Curious recvfrom error (%d) on port %hi: ",
            errno, portno[i]);
        perror("");
        trynum[i] = portno[i] = 0;
        close(sockets[i]);
    }
}
else /*bytes is positive*/ {
    if (debugging || verbose)
        printf("Adding UDP port %d due to positive read!\n", portno[i]);
    addport(ports, portno[i], IPPROTO_UDP, NULL);
    trynum[i] = portno[i] = 0;
    close(sockets[i]);
}
}
}
/* Update last_open, we need to create new sockets.*/
for(i=0, k=0; i < last_open; i++)
    if (portno[i]) {
        close(sockets[i]);
        sockets[k] = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
        /* unblock_socket(sockets[k]);*/
        portno[k] = portno[i];
        trynum[k] = trynum[i];
        k++;
    }
last_open = k;
for(i=k; i < max_parallel_sockets; i++)
    trynum[i] = sockets[i] = portno[i] = 0;
}
if (debugging)
    printf("UDP scanned %d ports in %ld seconds with %d parallel sockets\n",
        number_of_ports, time(NULL) - starttime, max_parallel_sockets);
return *ports;
}

/* This attempts to calculate the round trip time (rtt) to a host by timing a
connect() to a port which isn't listening. A better approach is to time a
ping (since it is more likely to get through firewalls. This is now
implemented in isup() for users who are root. */
unsigned long calculate_sleep(struct in_addr target) {
    struct timeval begin, end;
    int sd;
    struct sockaddr_in sock;
    int res;

    if ((sd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) == -1)
        {perror("Socket troubles"); exit(1);}

    sock.sin_family = AF_INET;
    sock.sin_addr.s_addr = target.s_addr;
    sock.sin_port = htons(MAGIC_PORT);

    gettimeofday(&begin, NULL);
    if ((res = connect(sd, (struct sockaddr *) &sock,
        sizeof(struct sockaddr_in))) != -1)
        printf("You might want to change MAGIC_PORT in the include file, it seems to be listening
on the target host!\n");
    close(sd);
}
```

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

```
gettimeofday(&end, NULL);
if (end.tv_sec - begin.tv_sec > 5 ) /*uh-oh!*/
    return 0;
return (end.tv_sec - begin.tv_sec) * 1000000 + (end.tv_usec - begin.tv_usec);
}

/* Checks whether the identd port (113) is open on the target machine. No
   sense wasting time trying it for each good port if it is down! */
int check_ident_port(struct in_addr target) {
int sd;
struct sockaddr_in sock;
int res;

if ((sd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) == -1)
    {perror("Socket troubles"); exit(1);}

sock.sin_family = AF_INET;
sock.sin_addr.s_addr = target.s_addr;
sock.sin_port = htons(113); /*should use getservbyname(3), yeah, yeah */
res = connect(sd, (struct sockaddr *) &sock, sizeof(struct sockaddr_in));
close(sd);
if (res < 0 ) {
    if (debugging || verbose) printf("identd port not active\n");
    return 0;
}
if (debugging || verbose) printf("identd port is active\n");
return 1;
}

int getidentinfoz(struct in_addr target, int localport, int remoteport,
                 char *owner) {
int sd;
struct sockaddr_in sock;
int res;
char request[15];
char response[1024];
char *p,*q;
char *os;

owner[0] = '\0';
if ((sd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) == -1)
    {perror("Socket troubles"); exit(1);}

sock.sin_family = AF_INET;
sock.sin_addr.s_addr = target.s_addr;
sock.sin_port = htons(113);
usleep(50000); /* If we aren't careful, we really MIGHT take out inetd,
               some are very fragile */
res = connect(sd, (struct sockaddr *) &sock, sizeof(struct sockaddr_in));

if (res < 0 ) {
    if (debugging || verbose)
        printf("identd port not active now for some reason ... hope we didn't break it!\n");
    close(sd);
    return 0;
}
sprintf(request,"%hi,%hi\r\n", remoteport, localport);
if (debugging > 1) printf("Connected to identd, sending request: %s", request);
if (write(sd, request, strlen(request) + 1) == -1) {
    perror("identd write");
    close(sd);
    return 0;
}
else if ((res = read(sd, response, 1024)) == -1) {
```

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

```
perror("reading from identd");
close(sd);
return 0;
}
else {
close(sd);
if (debugging > 1) printf("Read %d bytes from identd: %s\n", res, response);
if ((p = strchr(response, ':')) {
p++;
if ((q = strtok(p, " :")) {
if (!strcasecmp(q, "error")) {
if (debugging || verbose) printf("ERROR returned from identd\n");
return 0;
}
if ((os = strtok(NULL, " :")) {
if ((p = strtok(NULL, " :")) {
if ((q = strchr(p, '\r')) *q = '\0';
if ((q = strchr(p, '\n')) *q = '\0';
strncpy(owner, p, 512);
owner[512] = '\0';
}
}
}
}
}
}
return 1;
}
```

```
/* A relatively fast (or at least short ;) ping function. Doesn't require a
seperate checksum function */
int isup(struct in_addr target) {
int res, retries = 3;
struct sockaddr_in sock;
/*type(8bit)=8, code(8)=0 (echo REQUEST), checksum(16)=34190, id(16)=31337 */
#ifdef __LITTLE_ENDIAN_BITFIELD
unsigned char ping[64] = { 0x8, 0x0, 0x8e, 0x85, 0x69, 0x7A };
#else
unsigned char ping[64] = { 0x8, 0x0, 0x85, 0x8e, 0x7A, 0x69 };
#endif
int sd;
struct timeval tv;
struct timeval start, end;
fd_set fd_read;
struct {
struct iphdr ip;
unsigned char type;
unsigned char code;
unsigned short checksum;
unsigned short identifier;
char crap[16536];
} response;

sd = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);

bzero((char *)&sock, sizeof(struct sockaddr_in));
sock.sin_family=AF_INET;
sock.sin_addr = target;
if (debugging > 1) printf(" Sending 3 64 byte raw pings to host.\n");
gettimeofday(&start, NULL);
while(--retries) {
if ((res = sendto(sd, (char *) ping, 64, 0, (struct sockaddr *)&sock,
sizeof(struct sockaddr))) != 64) {
fprintf(stderr, "sendto in isup returned %d! skipping host.\n", res);
return 0;
}
```

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

```
    }
    FD_ZERO(&fd_read);
    FD_SET(sd, &fd_read);
    tv.tv_sec = 0;
    tv.tv_usec = 1e6 * (PING_TIMEOUT / 3.0);
    while(1) {
        if ((res = select(sd + 1, &fd_read, NULL, NULL, &tv)) != 1)
            break;
        else {
            read(sd,&response,sizeof(response));
            if (response.ip.saddr == target.s_addr && !response.type
                && !response.code && response.identifier == 31337) {
                gettimeofday(&end, NULL);
                global_rtt = (end.tv_sec - start.tv_sec) * 1e6 + end.tv_usec - start.tv_usec;
                ouraddr.s_addr = response.ip.daddr;
                close(sd);
                return 1;
            }
        }
    }
}
close(sd);
return 0;
}
```

```
portlist syn_scan(struct in_addr target, unsigned short *portarray,
                 struct in_addr *source, int fragment, portlist *ports) {
    int i=0, j=0, received, bytes, starttime;
    struct sockaddr_in from;
    int fromsize = sizeof(struct sockaddr_in);
    int sockets[max_parallel_sockets];
    struct timeval tv;
    char packet[65535];
    struct iphdr *ip = (struct iphdr *) packet;
    struct tcphdr *tcp = (struct tcphdr *) (packet + sizeof(struct iphdr));
    fd_set fd_read, fd_write;
    int res;
    struct hostent *myhostent;
    char myname[MAXHOSTNAMELEN + 1];
    int source_malloc = 0;

    FD_ZERO(&fd_read);
    FD_ZERO(&fd_write);

    tv.tv_sec = 7;
    tv.tv_usec = 0;

    if ((received = socket(AF_INET, SOCK_RAW, IPPROTO_TCP)) < 0 )
        perror("socket troubles in syn_scan");
    unblock_socket(received);
    FD_SET(received, &fd_read);

    /* First we take what is given to us as source.  If that isn't valid, we take
       what should have swiped from the echo reply in our ping function.  If THAT
       doesn't work either, we try to determine our address with gethostname and
       gethostbyname.  Whew! */
    if (!source) {
        if (ouraddr.s_addr) {
            source = &ouraddr;
        }
        else {
            source = safe_malloc(sizeof(struct in_addr));
            source_malloc = 1;
        }
    }
}
```

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

```
if (gethostname(myname, MAXHOSTNAMELEN) ||
    !(myhostent = gethostbyname(myname)))
    fatal("Your system is fucked up.\n");
memcpy(source, myhostent->h_addr_list[0], sizeof(struct in_addr));
}
if (debugging)
    printf("We skillfully deduced that your address is %s\n",
        inet_ntoa(*source));
}

starttime = time(NULL);

do {
    for(i=0; i < max_parallel_sockets && portarray[j]; i++) {
        if ((sockets[i] = socket(AF_INET, SOCK_RAW, IPPROTO_RAW)) < 0 )
            perror("socket troubles in syn_scan");
        else {
            if (fragment)
                send_small_fragz(sockets[i], source, &target, MAGIC_PORT,
                    portarray[j++], TH_SYN);
            else send_tcp_raw(sockets[i], source, &target, MAGIC_PORT,
                portarray[j++], 0, 0, TH_SYN, 0, 0, 0);
            usleep(10000);
        }
    }
    if ((res = select(received + 1, &fd_read, NULL, NULL, &tv)) < 0)
        perror("select problems in syn_scan");
    else if (res > 0) {
        while ((bytes = recvfrom(received, packet, 65535, 0,
            (struct sockaddr *)&from, &fromsize)) > 0 ) {
            if (ip->saddr == target.s_addr) {
                if (tcp->th_flags & TH_RST) {
                    if (debugging > 1) printf("Nothing open on port %d\n",
                        ntohs(tcp->th_sport));
                }
                else /*if (tcp->th_flags & TH_SYN && tcp->th_flags & TH_ACK)*/ {
                    if (debugging || verbose) {
                        printf("Possible catch on port %d! Here it is:\n",
                            ntohs(tcp->th_sport));
                        readtcppacket(packet, 1);
                    }
                    addport(ports, ntohs(tcp->th_sport), IPPROTO_TCP, NULL);
                }
            }
        }
    }
}
for(i=0; i < max_parallel_sockets && portarray[j]; i++) close(sockets[i]);

} while (portarray[j]);
if (debugging || verbose)
    printf("The TCP SYN scan took %ld seconds to scan %d ports.\n",
        time(NULL) - starttime, number_of_ports);
if (source_malloc) free(source); /* Gotta save those 4 bytes! ;) */
close(received);
return *ports;
}

int send_tcp_raw( int sd, struct in_addr *source,
    struct in_addr *victim, unsigned short sport,
    unsigned short dport, unsigned long seq,
    unsigned long ack, unsigned char flags,
    unsigned short window, char *data,
    unsigned short datalen)
```

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

```
{
struct pseudo_header {
    /*for computing TCP checksum, see TCP/IP Illustrated p. 145 */
    unsigned long s_addr;
    unsigned long d_addr;
    char zer0;
    unsigned char protocol;
    unsigned short length;
};
char packet[sizeof(struct iphdr) + sizeof(struct tcphdr) + datalen];
/*With these placement we get data and some field alignment so we aren't
wasting too much in computing the checksum */
struct iphdr *ip = (struct iphdr *) packet;
struct tcphdr *tcp = (struct tcphdr *) (packet + sizeof(struct iphdr));
struct pseudo_header *pseudo = (struct pseudo_header *) (packet + sizeof(struct iphdr) -
sizeof(struct pseudo_header));
int res;
struct sockaddr_in sock;
char myname[MAXHOSTNAMELEN + 1];
struct hostent *myhostent;
int source_malloced = 0;

/* check that required fields are there and not too silly */
if ( !victim || !sport || !dport || sd < 0) {
    fprintf(stderr, "send_tcp_raw: One or more of your parameters suck!\n");
    return -1;
}

/* if they didn't give a source address, fill in our first address */
if (!source) {
    source_malloced = 1;
    source = safe_malloc(sizeof(struct in_addr));
    if (gethostname(myname, MAXHOSTNAMELEN) ||
        !(myhostent = gethostbyname(myname)))
        fatal("Your system is fucked up.\n");
    memcpy(source, myhostent->h_addr_list[0], sizeof(struct in_addr));
    if (debugging > 1)
        printf("We skillfully deduced that your address is %s\n",
            inet_ntoa(*source));
}

/*do we even have to fill out this damn thing? This is a raw packet,
after all */
sock.sin_family = AF_INET;
sock.sin_port = htons(dport);
sock.sin_addr.s_addr = victim->s_addr;

bzero(packet, sizeof(struct iphdr) + sizeof(struct tcphdr));

pseudo->s_addr = source->s_addr;
pseudo->d_addr = victim->s_addr;
pseudo->protocol = IPPROTO_TCP;
pseudo->length = htons(sizeof(struct tcphdr) + datalen);

tcp->th_sport = htons(sport);
tcp->th_dport = htons(dport);
if (seq)
    tcp->th_seq = htonl(seq);
else tcp->th_seq = rand() + rand();

if (flags & TH_ACK && ack)
    tcp->th_ack = htonl(seq);
```

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

```
else if (flags & TH_ACK)
    tcp->th_ack = rand() + rand();

tcp->th_off = 5 /*words*/;
tcp->th_flags = flags;

if (window)
    tcp->th_win = window;
else tcp->th_win = htons(2048); /* Who cares */

tcp->th_sum = in_cksum((unsigned short *)pseudo, sizeof(struct tcphdr) +
    sizeof(struct pseudo_header) + datalen);

/* Now for the ip header */
bzero(packet, sizeof(struct iphdr));
ip->version = 4;
ip->ihl = 5;
ip->tot_len = htons(sizeof(struct iphdr) + sizeof(struct tcphdr) + datalen);
ip->id = rand();
ip->ttl = 255;
ip->protocol = IPPROTO_TCP;
ip->saddr = source->s_addr;
ip->daddr = victim->s_addr;
ip->check = in_cksum((unsigned short *)ip, sizeof(struct iphdr));

if (debugging > 1) {
printf("Raw TCP packet creation completed! Here it is:\n");
readtcppacket(packet, ntohs(ip->tot_len));
}
if (debugging > 1)
    printf("\nTrying sendto(%d , packet, %d, 0 , %s , %d)\n",
        sd, ntohs(ip->tot_len), inet_ntoa(*victim),
        sizeof(struct sockaddr_in));
if ((res = sendto(sd, packet, ntohs(ip->tot_len), 0,
    (struct sockaddr *)&sock, sizeof(struct sockaddr_in))) == -1)
    {
    perror("sendto in send_tcp_raw");
    if (source_malloced) free(source);
    return -1;
    }
if (debugging > 1) printf("successfully sent %d bytes of raw_tcp!\n", res);

if (source_malloced) free(source);
return res;
}

/* A simple program I wrote to help in debugging, shows the important fields
of a TCP packet*/
int readtcppacket(char *packet, int readdata) {
struct iphdr *ip = (struct iphdr *) packet;
struct tcphdr *tcp = (struct tcphdr *) (packet + sizeof(struct iphdr));
char *data = packet + sizeof(struct iphdr) + sizeof(struct tcphdr);
int tot_len;
struct in_addr bullshit, bullshit2;
char sourcehost[16];
int i;

if (!packet) {
    fprintf(stderr, "readtcppacket: packet is NULL!\n");
    return -1;
    }
bullshit.s_addr = ip->saddr; bullshit2.s_addr = ip->daddr;
tot_len = ntohs(ip->tot_len);
strncpy(sourcehost, inet_ntoa(bullshit), 16);
```

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

```
i = 4 * (ntohs(ip->ihl) + ntohs(tcp->th_off));
if (ip->protocol == IPPROTO_TCP)
    if (ip->frag_off) printf("Packet is fragmented, offset field: %u",
        ip->frag_off);
else {
    printf("TCP packet: %s:%d -> %s:%d (total: %d bytes)\n", sourcehost,
        ntohs(tcp->th_sport), inet_ntoa(bullshit2),
        ntohs(tcp->th_dport), tot_len);
    printf("Flags: ");
    if (!tcp->th_flags) printf("(none)");
    if (tcp->th_flags & TH_RST) printf("RST ");
    if (tcp->th_flags & TH_SYN) printf("SYN ");
    if (tcp->th_flags & TH_ACK) printf("ACK ");
    if (tcp->th_flags & TH_PUSH) printf("PSH ");
    if (tcp->th_flags & TH_FIN) printf("FIN ");
    if (tcp->th_flags & TH_URG) printf("URG ");
    printf("\n");
    printf("ttl: %hi ", ip->tttl);
    if (tcp->th_flags & (TH_SYN | TH_ACK)) printf("Seq: %lu\tAck: %lu\n",
        tcp->th_seq, tcp->th_ack);
    else if (tcp->th_flags & TH_SYN) printf("Seq: %lu\n", ntohl(tcp->th_seq));
    else if (tcp->th_flags & TH_ACK) printf("Ack: %lu\n", ntohl(tcp->th_ack));
}
if (readdata && i < tot_len) {
    printf("Data portion:\n");
    while(i < tot_len) printf("%2X%c", data[i], (++i%16)? ' ' : '\n');
    printf("\n");
}
return 0;
}

/* We don't exactly need real crypto here (thank god!)\n"*/
int shortfry(unsigned short *ports) {
    int num;
    unsigned short tmp;
    int i;

    for(i=0; i < number_of_ports; i++) {
        num = rand() % (number_of_ports);
        tmp = ports[i];
        ports[i] = ports[num];
        ports[num] = tmp;
    }
    return 1;
}

/* Much of this is swiped from my send_tcp_raw function above, which
   doesn't support fragmentation */
int send_small_fragz(int sd, struct in_addr *source, struct in_addr *victim,
    int sport, int dport, int flags) {

    struct pseudo_header {
        /*for computing TCP checksum, see TCP/IP Illustrated p. 145 */
        unsigned long s_addr;
        unsigned long d_addr;
        char zer0;
        unsigned char protocol;
        unsigned short length;
    };
    /*In this placement we get data and some field alignment so we aren't wasting
       too much to compute the TCP checksum.*/
    char packet[sizeof(struct iphdr) + sizeof(struct tcphdr) + 100];
    struct iphdr *ip = (struct iphdr *) packet;
```

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

```
struct tcphdr *tcp = (struct tcphdr *) (packet + sizeof(struct iphdr));
struct pseudo_header *pseudo = (struct pseudo_header *) (packet + sizeof(struct iphdr) -
sizeof(struct pseudo_header));
char *frag2 = packet + sizeof(struct iphdr) + 16;
struct iphdr *ip2 = (struct iphdr *) (frag2 - sizeof(struct iphdr));
int res;
struct sockaddr_in sock;
int id;

/*Why do we have to fill out this damn thing? This is a raw packet, after all */
sock.sin_family = AF_INET;
sock.sin_port = htons(dport);
sock.sin_addr.s_addr = victim->s_addr;

bzero(packet, sizeof(struct iphdr) + sizeof(struct tcphdr));

pseudo->s_addr = source->s_addr;
pseudo->d_addr = victim->s_addr;
pseudo->protocol = IPPROTO_TCP;
pseudo->length = htons(sizeof(struct tcphdr));

tcp->th_sport = htons(sport);
tcp->th_dport = htons(dport);
tcp->th_seq = rand() + rand();

tcp->th_off = 5 /*words*/;
tcp->th_flags = flags;

tcp->th_win = htons(2048); /* Who cares */

tcp->th_sum = in_cksum((unsigned short *)pseudo,
sizeof(struct tcphdr) + sizeof(struct pseudo_header));

/* Now for the ip header of frag1 */
bzero(packet, sizeof(struct iphdr));
ip->version = 4;
ip->ihl = 5;
/*RFC 791 allows 8 octet frags, but I get "operation not permitted" (EPERM)
when I try that. */
ip->tot_len = htons(sizeof(struct iphdr) + 16);
id = ip->id = rand();
ip->frag_off = htons(MORE_FRAGMENTS);
ip->ttl = 255;
ip->protocol = IPPROTO_TCP;
ip->saddr = source->s_addr;
ip->daddr = victim->s_addr;
ip->check = in_cksum((unsigned short *)ip, sizeof(struct iphdr));

if (debugging > 1) {
printf("Raw TCP packet fragment #1 creation completed! Here it is:\n");
hdump(packet,20);
}
if (debugging > 1)
printf("\nTrying sendto(%d , packet, %d, 0 , %s , %d)\n",
sd, ntohs(ip->tot_len), inet_ntoa(*victim),
sizeof(struct sockaddr_in));
if ((res = sendto(sd, packet, ntohs(ip->tot_len), 0,
(struct sockaddr *)&sock, sizeof(struct sockaddr_in))) == -1)
{
perror("sendto in send_syn_fragz");
return -1;
}
if (debugging > 1) printf("successfully sent %d bytes of raw_tcp!\n", res);
```

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

```
/* Create the second fragment */
bzero(ip2, sizeof(struct iphdr));
ip2->version = 4;
ip2->ihl = 5;
ip2->tot_len = htons(sizeof(struct iphdr) + 4); /* the rest of our TCP packet */
ip2->id = id;
ip2->frag_off = htons(2);
ip2->tttl = 255;
ip2->protocol = IPPROTO_TCP;
ip2->saddr = source->s_addr;
ip2->daddr = victim->s_addr;
ip2->check = in_cksum((unsigned short *)ip2, sizeof(struct iphdr));
if (debugging > 1) {
    printf("Raw TCP packet fragment creation completed! Here it is:\n");
    hdump(packet,20);
}
if (debugging > 1)
    printf("\nTrying sendto(%d , ip2, %d, 0 , %s , %d)\n", sd,
        ntohs(ip2->tot_len), inet_ntoa(*victim), sizeof(struct sockaddr_in));
if ((res = sendto(sd, ip2, ntohs(ip2->tot_len), 0,
    (struct sockaddr *)&sock, sizeof(struct sockaddr_in))) == -1)
    {
        perror("sendto in send_tcp_raw");
        return -1;
    }
return 1;
}

/* Hex dump */
void hdump(unsigned char *packet, int len) {
    unsigned int i=0, j=0;

    printf("Here it is:\n");

    for(i=0; i < len; i++){
        j = (unsigned) (packet[i]);
        printf("%-2X ", j);
        if (!((i+1)%16))
            printf("\n");
        else if (!((i+1)%4))
            printf(" ");
    }
    printf("\n");
}

portlist fin_scan(struct in_addr target, unsigned short *portarray,
    struct in_addr *source, int fragment, portlist *ports) {

    int rawsd, tcpsd;
    int done = 0, badport, starttime, someleft, i, j=0, retries=2;
    int source_malloc = 0;
    int waiting_period = retries, sockaddr_in_size = sizeof(struct sockaddr_in);
    int bytes, dupesinarow = 0;
    unsigned long timeout;
    struct hostent *myhostent;
    char response[65535], myname[513];
    struct iphdr *ip = (struct iphdr *) response;
    struct tcphdr *tcp;
    unsigned short portno[max_parallel_sockets], trynum[max_parallel_sockets];
    struct sockaddr_in stranger;

    timeout = (global_delay)? global_delay : (global_rtt)? (1.2 * global_rtt) + 10000 : 1e5;
```

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

```
bzero(&stranger, sockaddr_in_size);
bzero(portno, max_parallel_sockets * sizeof(unsigned short));
bzero(trynum, max_parallel_sockets * sizeof(unsigned short));
starttime = time(NULL);

if (debugging || verbose)
    printf("Initiating FIN stealth scan against %s (%s), sleep delay: %ld useconds\n",
        current_name, inet_ntoa(target), timeout);

if (!source) {
    if (ouraddr.s_addr) {
        source = &ouraddr;
    }
    else {
        source = safe_malloc(sizeof(struct in_addr));
        source_malloc = 1;
        if (gethostname(myname, MAXHOSTNAMELEN) ||
            !(myhostent = gethostbyname(myname)))
            fatal("Your system is fucked up.\n");
        memcpy(source, myhostent->h_addr_list[0], sizeof(struct in_addr));
    }
    if (debugging || verbose)
        printf("We skillfully deduced that your address is %s\n",
            inet_ntoa(*source));
}

if ((rawsd = socket(AF_INET, SOCK_RAW, IPPROTO_RAW)) < 0 )
    perror("socket troubles in fin_scan");

if ((tcpsd = socket(AF_INET, SOCK_RAW, IPPROTO_TCP)) < 0 )
    perror("socket troubles in fin_scan");

unblock_socket(tcpsd);
while(!done) {
    for(i=0; i < max_parallel_sockets; i++) {
        if (!portno[i] && portarray[j]) {
            portno[i] = portarray[j++];
        }
        if (portno[i]) {
            if (fragment)
                send_small_fragz(rawsd, source, &target, MAGIC_PORT, portno[i], TH_FIN);
            else send_tcp_raw(rawsd, source, &target, MAGIC_PORT,
                portno[i], 0, 0, TH_FIN, 0, 0, 0);
            usleep(10000); /* *WE* normally do not need this, but the target
                lamer often does */
        }
    }
}

usleep(timeout);
dupesinarow = 0;
while ((bytes = recvfrom(tcpsd, response, 65535, 0, (struct sockaddr *)
    &stranger, &sockaddr_in_size)) > 0)
    if (ip->saddr == target.s_addr) {
        tcp = (struct tcphdr *) (response + 4 * ip->ihl);
        if (tcp->th_flags & TH_RST) {
            badport = ntohs(tcp->th_sport);
            if (debugging > 1) printf("Nothing open on port %d\n", badport);
            /* delete the port from active scanning */
            for(i=0; i < max_parallel_sockets; i++)
                if (portno[i] == badport) {
                    if (debugging && trynum[i] > 0)
                        printf("Bad port %d caught on fin scan, try number %d\n",
                            badport, trynum[i] + 1);
                }
        }
    }
```

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

```
        trynum[i] = 0;
        portno[i] = 0;
        break;
    }
    if (i == max_parallel_sockets) {
        if (debugging)
            printf("Late packet or dupe, deleting port %d.\n", badport);
        dupesinarow++;
        if (ports) deleteport(ports, badport, IPPROTO_TCP);
    }
}
else
    if (debugging > 1) {
        printf("Strange packet from target%d! Here it is:\n",
            ntohs(tcp->th_sport));
        if (bytes >= 40) readtcppacket(response,1);
        else hdump(response,bytes);
    }
}

/* adjust waiting time if necessary */
if (dupesinarow > 6) {
    if (debugging || verbose)
        printf("Slowing down send frequency due to multiple late packets.\n");
    if (timeout < 10 * ((global_delay)? global_delay: global_rtt + 20000)) timeout *= 1.5;
    else {
        printf("Too many late packets despite send frequency decreases, skipping scan.\n");
        if (source_malloc) free(source);
        return *ports;
    }
}

/* Ok, collect good ports (those that we haven't received responses too
after all our retries */
someleft = 0;
for(i=0; i < max_parallel_sockets; i++)
    if (portno[i]) {
        if (++trynum[i] >= retries) {
            if (verbose || debugging)
                printf("Good port %d detected by fin_scan!\n", portno[i]);
            addport(ports, portno[i], IPPROTO_TCP, NULL);
            send_tcp_raw( rawsd, source, &target, MAGIC_PORT, portno[i], 0, 0,
                TH_FIN, 0, 0, 0);
            portno[i] = trynum[i] = 0;
        }
        else someleft = 1;
    }

if (!portarray[j] && (!someleft || --waiting_period <= 0)) done++;
}

if (debugging || verbose)
    printf("The TCP stealth FIN scan took %ld seconds to scan %d ports.\n",
        time(NULL) - starttime, number_of_ports);
if (source_malloc) free(source);
close(tcpsd);
close(rawsd);
return *ports;
}

int ftp_anon_connect(struct ftpinfo *ftp) {
int sd;
struct sockaddr_in sock;
```

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

```
int res;
char recvbuf[2048];
char command[512];

if (verbose || debugging)
    printf("Attempting connection to ftp://%s:%s@%s:%i\n", ftp->user, ftp->pass,
          ftp->server_name, ftp->port);

if ((sd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0) {
    perror("Couldn't create ftp_anon_connect socket");
    return 0;
}

sock.sin_family = AF_INET;
sock.sin_addr.s_addr = ftp->server.s_addr;
sock.sin_port = htons(ftp->port);
res = connect(sd, (struct sockaddr *) &sock, sizeof(struct sockaddr_in));
if (res < 0) {
    printf("Your ftp bounce proxy server won't talk to us!\n");
    exit(1);
}
if (verbose || debugging) printf("Connected:");
while ((res = recvtime(sd, recvbuf, 2048,7)) > 0)
    if (debugging || verbose) {
        recvbuf[res] = '\0';
        printf("%s", recvbuf);
    }
if (res < 0) {
    perror("recv problem from ftp bounce server");
    exit(1);
}

snprintf(command, 511, "USER %s\r\n", ftp->user);
send(sd, command, strlen(command), 0);
res = recvtime(sd, recvbuf, 2048,12);
if (res <= 0) {
    perror("recv problem from ftp bounce server");
    exit(1);
}
recvbuf[res] = '\0';
if (debugging) printf("sent username, received: %s", recvbuf);
if (recvbuf[0] == '5') {
    printf("Your ftp bounce server doesn't like the username \"%s\"\n",
          ftp->user);
    exit(1);
}
snprintf(command, 511, "PASS %s\r\n", ftp->pass);
send(sd, command, strlen(command), 0);
res = recvtime(sd, recvbuf, 2048,12);
if (res < 0) {
    perror("recv problem from ftp bounce server\n");
    exit(1);
}
if (!res) printf("Timeout from bounce server ...");
else {
    recvbuf[res] = '\0';
    if (debugging) printf("sent password, received: %s", recvbuf);
    if (recvbuf[0] == '5') {
        fprintf(stderr, "Your ftp bounce server refused login combo (%s/%s)\n",
              ftp->user, ftp->pass);
        exit(1);
    }
}
}
while ((res = recvtime(sd, recvbuf, 2048,2)) > 0)
```

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

```
if (debugging) {
    recvbuf[res] = '\0';
    printf("%s", recvbuf);
}
if (res < 0) {
    perror("recv problem from ftp bounce server");
    exit(1);
}
if (verbose) printf("Login credentials accepted by ftp server!\n");

ftp->sd = sd;
return sd;
}

int recvtime(int sd, char *buf, int len, int seconds) {

int res;
struct timeval timeout = {seconds, 0};
fd_set readfd;

FD_ZERO(&readfd);
FD_SET(sd, &readfd);
res = select(sd + 1, &readfd, NULL, NULL, &timeout);
if (res > 0 ) {
res = recv(sd, buf, len, 0);
if (res >= 0) return res;
perror("recv in recvtime");
return 0;
}
else if (!res) return 0;
perror("select() in recvtime");
return -1;
}

portlist bounce_scan(struct in_addr target, unsigned short *portarray,
                    struct ftpinfo *ftp, portlist *ports) {
int starttime, res, sd = ftp->sd, i=0;
char *t = (char *)&target;
int retriesleft = FTP_RETRIES;
char recvbuf[2048];
char targetstr[20];
char command[512];
snprintf(targetstr, 20, "%d,%d,%d,%d,0,", UC(t[0]), UC(t[1]), UC(t[2]), UC(t[3]));
starttime = time(NULL);
if (verbose || debugging)
    printf("Initiating TCP ftp bounce scan against %s (%s)\n",
           current_name, inet_ntoa(target));
for(i=0; portarray[i]; i++) {
    snprintf(command, 512, "PORT %s%i\r\n", targetstr, portarray[i]);
    if (send(sd, command, strlen(command), 0) < 0 ) {
        perror("send in bounce_scan");
        if (retriesleft) {
            if (verbose || debugging)
                printf("Our ftp proxy server hung up on us!  retrying\n");
            retriesleft--;
            close(sd);
            ftp->sd = ftp_anon_connect(ftp);
            if (ftp->sd < 0) return *ports;
            sd = ftp->sd;
            i--;
        }
        else {
            fprintf(stderr, "Our socket descriptor is dead and we are out of retries. Giving
up.\n");
        }
    }
}
}
```



# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

```
}

/* parse a URL stype ftp string of the form user:pass@server:portno */
int parse_bounce(struct ftpinfo *ftp, char *url) {
char *p = url,*q, *s;

if ((q = strrchr(url, '@')) /*we have username and/or pass */ {
*(q++) = '\0';
if ((s = strchr(q, ':'))
{ /* has portno */
*(s++) = '\0';
strncpy(ftp->server_name, q, MAXHOSTNAMELEN);
ftp->port = atoi(s);
}
else strncpy(ftp->server_name, q, MAXHOSTNAMELEN);

if ((s = strchr(p, ':')) { /* User AND pass given */
*(s++) = '\0';
strncpy(ftp->user, p, 63);
strncpy(ftp->pass, s, 255);
}
else { /* Username ONLY given */
printf("Assuming %s is a username, and using the default password: %s\n",
p, ftp->pass);
strncpy(ftp->user, p, 63);
}
}
else /* no username or password given */
if ((s = strchr(url, ':')) { /* portno is given */
*(s++) = '\0';
strncpy(ftp->server_name, url, MAXHOSTNAMELEN);
ftp->port = atoi(s);
}
else /* default case, no username, password, or portnumber */
strncpy(ftp->server_name, url, MAXHOSTNAMELEN);

ftp->user[63] = ftp->pass[255] = ftp->server_name[MAXHOSTNAMELEN] = 0;

return 1;
}

/*
* I'll bet you've never seen this function before (yeah right)!
* standard swiped checksum routine.
*/
unsigned short in_cksum(unsigned short *ptr,int nbytes) {

register long sum; /* assumes long == 32 bits */
u_short oddbyte;
register u_short answer; /* assumes u_short == 16 bits */

/*
* Our algorithm is simple, using a 32-bit accumulator (sum),
* we add sequential 16-bit words to it, and at the end, fold back
* all the carry bits from the top 16 bits into the lower 16 bits.
*/

sum = 0;
while (nbytes > 1) {
sum += *ptr++;
nbytes -= 2;
}
}
```

# The Art of Port Scanning

Fyodor

(Originally Published in Phrack Magazine Volume 7, Issue 51 September 01, 1997)

```
/* mop up an odd byte, if necessary */
if (nbytes == 1) {
    oddbyte = 0;          /* make sure top half is zero */
    *((u_char *) &oddbyte) = *(u_char *)ptr; /* one byte only */
    sum += oddbyte;
}

/*
 * Add back carry outs from top 16 bits to low 16 bits.
 */

sum = (sum >> 16) + (sum & 0xffff); /* add high-16 to low-16 */
sum += (sum >> 16); /* add carry */
answer = ~sum; /* ones-complement, then truncate to 16 bits */
return(answer);
}
<-->
```

----[ EOF