

# [TOOL] PScanDetect – TCP Portscan Detector

**Source:** <http://www.derkeiler.com/Mailing-Lists/Securiteam/2003-12/0086.html>

---

**From:** SecuriTeam ([support\\_at\\_securiteam.com](mailto:support_at_securiteam.com))

**Date:** 12/24/03

To: [list@securiteam.com](mailto:list@securiteam.com)

Date: 24 Dec 2003 15:11:40 +0200

The following security advisory is sent to the securiteam mailing list, and can be found at the SecuriTeam web site: <http://www.securiteam.com>

-- promotion

The SecuriTeam alerts list – Free, Accurate, Independent.

Get your security news from a reliable source.

<http://www.securiteam.com/maillinglist.html>

-----

PScanDetect – TCP Portscan Detector

---

## DETAILS

PScanDetect is a utility that will detect TCP-based portscans.

Tool source code:

```
/*
 * TCP Portscan Detector v0.8
 * os: Linux, FreeBSD, OpenBSD
 * author: dodo <dodo@darkwired.org>
 * date: 23-dec-2003
 *
 * description:
 * this tool can detect TCP portscans and alerts them
 * check -h for more info
 * tested on: FreeBSD 5.1, OpenBSD 3.3, Slackware linux 9.0
 *
 * license:
 * this code is licensed under the GNU GPL
 * by compiling this, you agree to this license
 *
 * compilation:
 * gcc PScanDetect.c -o pscandetect -lpcap
 *
```

## Securiteam: [TOOL] PScanDetect – TCP Portscan Detector

```
* todo:  
* SCAN_INSPECT detection  
*  
*/
```

```
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <errno.h>  
#include <time.h>  
#include <sys/ioctl.h>  
#include <sys/socket.h>  
#include <net/if.h>
```

```
#ifndef __OpenBSD__  
#include <net/ethernet.h>  
#endif
```

```
#include <netinet/in.h>  
#include <netinet/if_ether.h>  
#include <netinet/tcp.h>  
#include <arpa/inet.h>  
#include <pcap.h>
```

```
#ifdef __Win32__  
#error "you idiot, I did not code this for Win32!"  
#endif
```

```
#ifdef __Linux__  
#include <netinet/ether.h>  
#endif
```

```
/* define the scan types */  
#define SCAN_SINGLESTEP 0x1  
#define SCAN_LARGESTEP 0x2  
#define SCAN_INSPECT 0x3  
#define SCAN_MANYHIGH 0x4
```

```
/* define boundaries */  
#define MAX_BLASTERS 20  
#define MAX_SCANNERS 20  
#define MAX_PORTS 20  
#define BOUNDARY_MANYHIGH 5000
```

```
/* define misc constants */  
#define VERSION 0.8  
#define NUM_TYPES 4  
#define NUM_DROP_SECS 5  
#define NAME_SCAN_SINGLESTEP "SCAN_SINGLESTEP"  
#define NAME_SCAN_LARGESTEP "SCAN_LARGESTEP"  
#define NAME_SCAN_INSPECT "SCAN_INSPECT"
```

## Securiteam: [TOOL] PScanDetect – TCP Portscan Detector

```
#define NAME_SCAN_MANYHIGH "SCAN_MANYHIGH"

struct my_ip {
    u_int8_t ip_vhl; /* header length, version */
#define IP_V(ip) (((ip)->ip_vhl & 0xf0) >> 4)
#define IP_HL(ip) ((ip)->ip_vhl & 0x0f)
    u_int8_t ip_tos; /* type of service */
    u_int16_t ip_len; /* total length */
    u_int16_t ip_id; /* identification */
    u_int16_t ip_off; /* fragment offset field */
#define IP_DF 0x4000 /* dont fragment flag */
#define IP_MF 0x2000 /* more fragments flag */
#define IP_OFFMASK 0x1fff /* mask for fragmenting bits */
    u_int8_t ip_ttl; /* time to live */
    u_int8_t ip_p; /* protocol */
    u_int16_t ip_sum; /* checksum */
    struct in_addr ip_src,ip_dst; /* source and dest address */
};

struct BlastRadius {
    unsigned short id; /* identifier */
    unsigned short ports[MAX_PORTS]; /* ports accessed */
    unsigned short portsn; /* port pointer */
    unsigned long birth; /* since when? */
    unsigned long lporttime; /* time when last added port */
    unsigned short active; /* recycling? */
    struct in_addr host; /* the bastard */
};

struct Scanner {
    unsigned short id; /* identifier */
    unsigned short portsn; /* num ports accessed */
    unsigned char type; /* scan type */
    unsigned long lporttime; /* time when last added port */
    struct in_addr host; /* the bastard */
};

/* interface attributes */
char *dev;
char *logfile = NULL;
FILE *logfd;
unsigned short int level = 3;
unsigned char promiscuous = 0;
unsigned char verbose = 0;

/* internal attributes */
char selfhost[20];
char errbuf[PCAP_ERRBUF_SIZE];
pcap_t *handle;
    struct pcap_pkthdr header;
    const u_char *packet;
```

## Securiteam: [TOOL] PScanDetect – TCP Portscan Detector

```
struct ether_header *data_ether;
const struct my_ip* data_ip;
const struct tcphdr* data_tcp;
struct BlastRadius *blasters[MAX_BLASTERS];
struct Scanner *scanners[MAX_SCANNERS];
unsigned long int blastsize = 0;
unsigned long int scansize = 0;
int h;

/* methods */
void initPcap();
char *getDevice();
void getDeviceIP();
void setFilter();
void openLive();
void closeLive();
void doLoop();
void nibblePacket();
void inspectBlast(struct in_addr blasthost, unsigned int port);
char addBR(struct in_addr blasthost, unsigned short int port);
void replaceBR(struct in_addr blasthost, unsigned int port);
void updateBR(struct in_addr blasthost, unsigned short int port);
char checkBR(struct in_addr blasthost);
char checkBRPort(struct BlastRadius *br, unsigned short int port);
char checkBRPortMax(struct BlastRadius *br);
void outputAlert(struct BlastRadius *br, char type);
void dumpBR();
void cleanBR();
char checkSelf(struct in_addr blasthost);
char checkScanner(struct in_addr blasthost);
char addScanner(struct in_addr blasthost, unsigned short portsn,
unsigned char type);
char updateScanner(struct in_addr blasthost);
void dumpScanners();
void usage(char *self);
char parseOptions(int argc, char *argv[]);

int main(int argc, char *argv[])
{
int i;
dev = getDevice();
if(parseOptions(argc, argv)==0) usage(argv[0]);
getDeviceIP();
openLive();
setFilter();
doLoop();
closeLive();
dumpBR();
}
```

## Securiteam: [TOOL] PScanDetect – TCP Portscan Detector

```
void usage(char *self)
{
fprintf(stdout,
"\nTCP Portscan Detector v%2.1f\n"
"http://www.darkwired.org/\n\n"
"usage: %s [options]\n"
"options:\n"
"-l <file> output to logfile\n"
"-a <level> alert when scan alert type <= then <level> (default 3)\n"
"-d <device> use this device to detect scans on\n"
"-p enable promiscuous mode (detect scans in all traffic on line)\n"
"-v enable verbose mode\n"
"-h show this\n"
"scan alert types:\n"
" 0x1 SCAN_SINGLESTEP\n"
" 0x2 SCAN_LARGESTEP\n"
" 0x3 SCAN_INSPECT\n"
" 0x4 SCAN_MANYHIGH\n",
VERSION, self);
exit(-1);
}
```

```
char parseOptions(int argc, char *argv[])
/* this will be messy :) */
{
int i;
for(i=0; argc > i; i++) {

if(argv[i][0]=='-') {
switch(argv[i][1]) {
case 'l': {
if((i+1)>=argc) return 0; /* option requires param */
logfile = argv[i+1]; i++;
break;
}

case 'd': {
if((i+1)>=argc) return 0; /* option requires param */
dev = argv[i+1]; i++;
break;
}

case 'a': {
if((i+1)>=argc) return 0; /* option requires param */
level = atoi(argv[i+1]); i++;
if(level<1 || level>NUM_TYPES) {
fprintf(stderr, "bad alert level: %d\n", level);
return 0;
}
break;
}
}
}
```

```

    case 'h': {
        return 0;
        break;
    }

    case 'p': {
        promiscuous = 1;
        break;
    }

    case 'v': {
        verbose = 1;
        break;
    }
}
}
}

if(verbose==1) {
    printf("device: %s\n", dev);
    if(logfile == NULL) {
        printf("logfile: stdout\n");
    } else {
        printf("logfile: %s\n", logfile);
    }

    printf("alertlevel: %d\n", level);
    printf("promiscuous: %d\n", promiscuous);
}
return 1;
}

char *getDevice()
{
    char *dev;
    dev = pcap_lookupdev(errbuf);
    return dev;
}

void getDeviceIP()
{
    struct sockaddr_in *ssin;
    struct ifreq ifr;
    unsigned long int ip;
    int fd;

    fd = socket(PF_INET, SOCK_DGRAM, 0);

    /* ripped this piece of code somewhere using google */
    memset(&ifr, 0, sizeof(ifr));
    ssin = (struct sockaddr_in *)&ifr.ifr_addr;

```

## Securiteam: [TOOL] PScanDetect – TCP Portscan Detector

```
strncpy(ifr.ifr_name, dev, sizeof(ifr.ifr_name));
ifr.ifr_addr.sa_family = AF_INET;

if (ioctl(fd, SIOCGIFADDR, (char*) &ifr) < 0)
{
    fprintf(stderr, "unable to get IP address of device: %s\n", dev);
    close(fd);
    exit(-1);
}

*ssin = * ( (struct sockaddr_in *)&ifr.ifr_addr);
if(verbose==1)
printf("device ip address: %s\n", inet_ntoa(ssin->sin_addr));
strncpy(selfhost, inet_ntoa(ssin->sin_addr), sizeof(selfhost));
}

void setFilter()
{
    struct bpf_program filter;
    char filter_app[] = "";
    bpf_u_int32 mask;
    bpf_u_int32 net;
    pcap_lookupnet(dev, &net, &mask, errbuf);
    pcap_compile(handle, &filter, filter_app, 0, net);
    pcap_setfilter(handle, &filter);
}

void openLive()
{
    if(dev == NULL) {
        fprintf(stderr, "could not get network device (check permissions)\n");
        exit(-1);
    }
    handle = pcap_open_live(dev, BUFSIZ, promiscuous, 0, errbuf);
}

void closeLive()
{
    pcap_close(handle);
}

void nibblePacket
(
    u_char *user,
    struct pcap_pkthdr *phrd,
    u_char *packet
)
{
    data_ether = (struct ether_header *)packet;
```

## Securiteam: [TOOL] PScanDetect – TCP Portscan Detector

```
if(ntohs(data_ether->ether_type)==ETHERTYPE_IP)
{
    data_ip = (struct my_ip *)(packet+sizeof(struct ether_header));
    data_tcp = (struct tcphdr *)(packet+sizeof(struct
ether_header)+sizeof(struct my_ip));

#ifdef __FreeBSD__
inspectBlast(data_ip->ip_src, ntohs(data_tcp->th_dport));
#elif defined __OpenBSD__
inspectBlast(data_ip->ip_src, ntohs(data_tcp->th_dport));
#elif defined __linux__
inspectBlast(data_ip->ip_src, ntohs(data_tcp->dest));
#endif

/*printf("%s:%d -> %s:%d\n", //debugging
    inet_ntoa(data_ip->ip_src),
    ntohs(data_tcp->source),
    inet_ntoa(data_ip->ip_dst),
    ntohs(data_tcp->dest)
    );
*/

}
}

void doLoop()
{
pcap_loop(handle, -1, (pcap_handler)nibblePacket, 0);
}

void inspectBlast(struct in_addr blasthost, unsigned int port)
{
    if(checkSelf(blasthost)) {
        return;
    }

    if(checkScanner(blasthost)==1) {
        updateScanner(blasthost);
        return;
    }

    if(checkBR(blasthost)==0) {
        if(addBR(blasthost, port)==0) {
            replaceBR(blasthost, port);
        }
    } else {
        updateBR(blasthost, port);
    }

    /* now we check wether someone scanned us */
    cleanBR();
}
```

```

}

void dumpBR()
{
int i=0, a=0;
  for(i=0; blastersize>i; i++) {
printf("%d BlastRadius->host: %s (%d ports in %d secs) [%d]\n",
blasters[i]->id, inet_ntoa(blasters[i]->host), blasters[i]->portsn,
(int)(time(NULL) - blasters[i]->lporttime), blasters[i]->active);
  for(a=0; blasters[i]->portsn>a; a++) {
    printf("%d,", blasters[i]->ports[a]);
  }
  printf("\n");
}
}

```

```

char addBR(struct in_addr blasthost, unsigned short int port)
{
  if(blastersize==MAX_BLASTERS) {
return 0;
  }
  blasters[blastersize] = malloc(sizeof(struct BlastRadius));
  blasters[blastersize]->id = blastersize;
  blasters[blastersize]->portsn = 0;
  blasters[blastersize]->active = 1;
  blasters[blastersize]->birth = time(NULL);
  blasters[blastersize]->lporttime = time(NULL);
  blasters[blastersize]->host = blasthost;
  blastersize++;
return 1;
}

```

```

void replaceBR(struct in_addr blasthost, unsigned int port)
{
int i;
  for(i=0; blastersize>i; i++) {
  if(blasters[i]->active==0) {
    blasters[i]->portsn = 0;
    blasters[i]->active = 1;
    blasters[i]->birth = time(NULL);
    blasters[i]->lporttime = time(NULL);
    blasters[i]->host = blasthost;

    break;
  }
}
}

```

```

void updateBR(struct in_addr blasthost, unsigned short int port)
{

```

```

char bh[20];
int i;
strncpy(bh, inet_ntoa(blasthost), (sizeof(bh)-1));
for(i=0; blastersize>i; i++) {
    if(
        strcmp(inet_ntoa(blasters[i]->host), bh) == 0
        &&
        blasters[i]->active == 1
    ) {
        if(blasters[i]->portsn < MAX_PORTS) {
            if(checkBRPort(blasters[i], port)==0) {
                /* update last_portadd field */
                blasters[i]->lporttime = time(NULL);
                blasters[i]->ports[(blasters[i]->portsn)] = port;
                blasters[i]->portsn++;
            }
        }
    }
}

char checkBRPort(struct BlastRadius *br, unsigned short int port)
{
int i;
for(i=0; br->portsn > i; i++) {
    if(br->ports[i] == port) {
        return 1;
    }
}
return 0;
}

char checkBRPortMax(struct BlastRadius *br)
{
if(br->portsn==MAX_PORTS) return 1;
return 0;
}

char checkBRPortScan(struct BlastRadius *br, char type)
{
if(type==SCAN_LARGESTEP) {

    if(checkBRPortScan(br, SCAN_MANYHIGH)==1)
        return 0;

    if(br->portsn > 8 && checkBRPortScan(br, SCAN_MANYHIGH)==0)
        return 1;
}

if(type==SCAN_SINGLESTEP) {
int i;

```

## Securiteam: [TOOL] PScanDetect – TCP Portscan Detector

```
int cport = br->ports[0];

if(br->portsn < 8) {
return 0;
}

if(checkBRPortScan(br, SCAN_MANYHIGH)==1) {
return 0;
}

for(i=1; br->portsn > i; i++) {
if( (cport+1) == br->ports[i] ) {
cport++;
} else {
return 0;
}
}
return 1;
}

if(type==SCAN_MANYHIGH) {
int i;
if(br->portsn < 8)
return 0;

for(i=1; br->portsn > i; i++) {
if(br->ports[i] < BOUNDARY_MANYHIGH)
return 0;
}

return 1;
}

return 0;
}

char checkBR(struct in_addr blasthost)
{
char bh[20];
int i;
strncpy(bh, inet_ntoa(blasthost), (sizeof(bh)-1));
for(i=0; blastersize>i; i++) {
if(
strcmp(inet_ntoa(blasters[i]->host), bh) == 0
&&
blasters[i]->active == 1
)
return 1;
}
return 0;
}
```

## Securiteam: [TOOL] PScanDetect – TCP Portscan Detector

```
char checkBRNoSuspect(struct BlastRadius *br)
{
    if((*br).portsn < 4) {
        /* thanks to ilja for reminding me :p */
        if((int)(time(NULL) - br->lporttime) > NUM_DROP_SECS)
            return 1;
    }

    return 0;
}

void cleanBR()
{
    int i;
    for(i=0; blastersize>i; i++) {

        if(checkBRNoSuspect(blasters[i])) {
            blasters[i]->active = 0;
        }

        if(blasters[i]->portsn > 2 && blasters[i]->active==1) {

            if(checkBRPortMax(blasters[i])==1) {
                blasters[i]->active = 0;
            }

            if(checkBRPortScan(blasters[i], SCAN_LARGESTEP)) {
                blasters[i]->active = 0;
                addScanner(blasters[i]->host, blasters[i]->portsn, SCAN_LARGESTEP);
                outputAlert(blasters[i], SCAN_LARGESTEP);
            }

            if(checkBRPortScan(blasters[i], SCAN_SINGLSTEP)) {
                blasters[i]->active = 0;
                addScanner(blasters[i]->host, blasters[i]->portsn, SCAN_SINGLSTEP);
                outputAlert(blasters[i], SCAN_SINGLSTEP);
            }

            if(checkBRPortScan(blasters[i], SCAN_MANYHIGH)) {
                blasters[i]->active = 0;
                addScanner(blasters[i]->host, blasters[i]->portsn, SCAN_MANYHIGH);
                outputAlert(blasters[i], SCAN_MANYHIGH);
            }
        }
    }
}

void outputAlert(struct BlastRadius *br, char type)
{
    char *scanname;
```

```

if(type > level) return;

if(logfile == NULL) {
    logfd = fopen("/dev/stdout", "a");
} else {
    logfd = fopen(logfile, "a");
}
if(!logfd) {
    fprintf(stderr, "unable to open logfile\n");
    exit(-1);
}

switch(type) {
case SCAN_SINGLESTEP:
    scanname = NAME_SCAN_SINGLESTEP;
    break;
case SCAN_LARGESTEP:
    scanname = NAME_SCAN_LARGESTEP;
    break;
case SCAN_MANYHIGH:
    scanname = NAME_SCAN_MANYHIGH;
    break;
case SCAN_INSPECT:
    scanname = NAME_SCAN_INSPECT;
    break;
}

fprintf(logfd, "detected scan from %s, type: %s\n", inet_ntoa(br->host),
scanname);
fclose(logfd);
/* good place to use debugging functions */
}

char checkSelf(struct in_addr blasthost)
{
    if(
        strcmp(selfhost, inet_ntoa(blasthost)) == 0
    )
        return 1;

    return 0;
}

char checkScanner(struct in_addr blasthost)
{
    char bh[20];
    int i;
    strncpy(bh, inet_ntoa(blasthost), (sizeof(bh)-1));
    for(i=0; scanrsize>i; i++) {
        if(

```

## Securiteam: [TOOL] PScanDetect – TCP Portscan Detector

```
    strcmp(inet_ntoa(scanners[i]->host), bh) == 0
    )
    return 1;
}

return 0;
}

char addScanner(struct in_addr blasthost, unsigned short portsn, unsigned
char type)
{
    if(scanrsize==MAX_SCANNERS) {
        return 0;
    }
    scanners[scanrsize] = malloc(sizeof(struct Scanner));
    scanners[scanrsize]->id = scanrsize;
    scanners[scanrsize]->type = type;
    scanners[scanrsize]->portsn = portsn;
    scanners[scanrsize]->lporttime = time(NULL);
    scanners[scanrsize]->host = blasthost;
    scanrsize++;
    return 1;
}

char updateScanner(struct in_addr blasthost)
{
    char bh[20];
    int i;
    strncpy(bh, inet_ntoa(blasthost), (sizeof(bh)-1));

    for(i=0; scanrsize>i; i++) {
        if(
            strcmp(inet_ntoa(scanners[i]->host), bh) == 0
        ) {
            scanners[i]->lporttime = time(NULL);
            return 1;
        }
    }

    return 0;
}

void dumpScanners()
{
    int i=0;
    for(i=0; scanrsize>i; i++) {
        printf("%d Scanner->host: %s (%d ports, last hit: %d secs)\n",
            scanners[i]->id, inet_ntoa(scanners[i]->host), scanners[i]->portsn,
            (int)(time(NULL) - scanners[i]->lporttime));
    }
}
```

ADDITIONAL INFORMATION

The information has been provided by <mailto:dodo@darkwired.org> dodo.

=====

This bulletin is sent to members of the SecuriTeam mailing list.

To unsubscribe from the list, send mail with an empty subject line and body to:

list-unsubscribe@securiteam.com

In order to subscribe to the mailing list, simply forward this email to: list-subscribe@securiteam.com

=====

=====

**DISCLAIMER:**

The information in this bulletin is provided "AS IS" without warranty of any kind.

In no event shall we be liable for any damages whatsoever including direct, indirect, incidental, consequential, loss of business profits or special damages.