HRVATSKA AKADEMSKA I ISTRAŽIVAČKA MREŽA
CROATIAN ACADEMIC AND RESEARCH NETWORK

Detektiranje pregledavanja portova na udaljenim računalima

CCERT-PUBDOC-2000-08-02

CARNet CERT u suradnji s LS&S

Sigurnosni problemi u računalnim programima i operativnim sustavima područje je na kojem CARNet CERT kontinuirano radi.

Rezultat toga rada ovaj je dokument koji je nastao suradnjom CARNet CERT-a i LS&S-a, a za koji se nadamo se da će Vam koristiti u poboljšanju sigurnosti Vašeg sustava.

CARNet CERT, www.cert.hr - nacionalno središte za **sigurnost računalnih mreža** i sustava.

LS&S, www.lss.hr - laboratorij za sustave i signale pri Zavodu za elektroničke sustave i obradbu informacija Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu.

Sadržaj

1. Uvod

Ovaj dokument pokazuje potencijalne probleme sa sustavima za detekciju neovlaštenih aktivnosti korisnika (eng. *Intrusion Detection Systems* – IDS) prilikom napada skeniranjem portova.
Nadalje dokument nije ograničen na mrežno orijentirane alate, ustvari alat za detekciju skeniranja portova (`scanlogd`) priložen na kraju dokumenta je računalno orijentiran.

2. Koji napadi se mogu detektirati

Skeniranje portova podrazumijeva napadača koji se pokušava spojiti na mnoge odredišne portove, obično uključujući i one koji nisu namijenjeni za primanje. Jedna od značajki, odnosno "potpis" koji se može koristiti za detekciju skeniranja portova je slanje više paketa na različite odredišne portove sa istom izvorišnom adresom unutar kratkog vremenskog perioda. Drugi takav potpis može biti SYN na port koji nije namijenjen primanju. Očigledno je da postoje mnoge metode za detekciju skeniranja portova sve do spremanja zaglavlja svih paketa u datoteku i manualne analize.
Sve gore navedene metode imaju određene prednosti, ali i nedostatke, što rezultira različitim brojevima neispravnih pozitivnih i neispravnih negativnih detekcija. Također postoje mnogi načini maskiranja napada za koje je vjerojatnost detekcije, ili pronalaženja stvarnog izvora vrlo mala, dok se pri tome ipak mogu dobiti informacije o brojevima portova.
Za maskiranje napada, napadač može napraviti skeniranje vrlo polako. Ukoliko napadnuti sustav ne radi u normalnom režimu (u tom slučaju jedan paket na port koji nije namijenjen primanju je dovoljan za slanje administrativnog upozorenja; što je prilično nevjerojatno u stvarnom svijetu), moguće je učiniti razmake između pojedinih skeniranja dovoljno velikim da napad ne bude prepoznat kao skeniranje portova.
Isto tako, izvorište napada također se može maskirati, a istovremeno to izvorište prima informacije. Moguće je poslati velik broj skenirajućih paketa sa lažnim zaglavljima (adresom izvora) npr. 999, dok samo jedan od njih predstavlja stvarnu adresu. Čak i ako se svi skenovi detektiraju i zabilježe, nema načina otkriti koja je stvarna adresa izvora napada. Jedino se može odrediti da su portovi skenirani.
Ovdje valja uočiti da, iako su gore opisane modifikacije napada moguće, one očito zahtijevaju više resursa. Neki napadači neće ni pokušati ovakve spore ili komplicirane metode napada, dok će oni drugi morati to platiti svojim vremenom. To je razlog zbog kojeg ima smisla detektirati barem neke napade skeniranjem portova (onih koje je moguće detektirati).
Mogućnost takvih napada znači da nije cilj detektirati sve pokušaje skeniranja portova (što je nemoguće), već detektirati što više vrsta napada, a istovremeno osigurati razuman stupanj pouzdanosti.

3. Koje informacije su pouzdane

Očito je da adresa izvora može biti lažna, tako da se tome ne može vjerovati ukoliko nema drugih dokaza. Isto tako neki skeneri portova ostavljaju dodatne informacije koje se mogu koristiti prilikom određivanja pravog izvora napada.
Na primjer, ukoliko paketi koji se primaju imaju IP TTL vrijednost 255, sigurno je da su poslani s lokalne mreže, bez obzira na što piše u polju izvorne adrese. Isto tako ukoliko je TTL vrijednost 250 može se zaključiti da je napadač udaljen manje od 5 skokova, ali ne i točno koliko ima skokova do njega.
Početni TTL i broj(evi) izvorišnih portova mogu također dati neke informacije od vrsti skenera (za "*stealth*" skenere) ili operacijskom sustavu (za potpuna skeniranja TCP spojeva) koji se koristi za napad. Npr., `nmap` podešava TTL na vrijednost 255, a port na 49724, dok Linux podešava TTL na 64.

4. Izvor informacija (E-box)

Za detekciju napada skeniranjem TCP portova, uključujući i "*stealth*" napade, mora se moći pristupiti čistim zaglavljima IP i TCP paketa.

U mrežno orijentiranim sustavima za detekciju neovlaštenih aktivnosti za to je potrebna analiza svih paketa u mreži (eng. *promiscous mode*). Kod toga su moguće i pogreške, pa se tako mogu dogoditi lažne pozitivne ili lažne negativne detekcije, no i to može biti prihvatljivo u nekim slučajevima.
Za računalno orijentirane sustave za detekciju neovlaštenih aktivnosti korisnika postoje dvije mogućnosti dohvaćanja paketa; čitanja iz čistih TCP ili IP *socketa* ili dohvaćanja podataka direktno iz jezgre operacijskog sustava.
Korištenjem čistog TCP *socketa* dobivaju se svi paketi koje jezgra operacijskog sustava prepoznaje, no to je pasivna analiza (moguće je propustiti neke pakete). To može biti prihvatljivo za detekciju napada skeniranjem portova, ali nije dobro rješenje za detekciju drugih napada. Ukoliko se koristi čisti IP *socket* (neki sustavi ne koriste IP sockete), ponovo se pojavljuje mogućnost lažnih pozitivnih i negativnih detekcija. Programski kôd priložen na kraju dokumenta koristi se TCP *socketima*.
Najpouzdaniji alati za detekciju neovlaštenih aktivnosti korisnika koriste podršku jezgre operacijskog sustava, što im omogućava pristup svim važnim informacijama. Nedostatak tih kernel modula ili zakrpa jest što nisu dovoljno portabilni.

5. Napadački potpis (A-box)

Već je spomenuto da se različiti "potpisi" koji predstavljaju značajke pojedinog napada mogu koristiti za detekciju napada. Potpisi također uvjetuju broj lažnih pozitivnih odnosno lažnih negativnih detekcija. Odabir potpisa napada trebao bi osigurati što manji broj lažnih pozitivnih uzbuna, istodobno držeći broj lažnih negativnih prihvatljivim. Ponekad se postavlja pitanje što je prihvatljivo; smatra se da to ovisi i o ozbiljnosti napada (odnosno "cijeni" lažne negativne detekcije) i o postupcima provedenim u slučaju detekcije napada ("cijena" lažne pozitivne detekcije). Obje "cijene" ovise o sustavu na kojem se vrši detekcija neovlaštenih aktivnosti korisnika, pa je poželjna mogućnost korisničkog ugađanja.
U programu priloženom na kraju dokumenta koristi se sljedeći potpis napada: "barem COUNT portova mora biti skeniran sa iste izvorišne adrese, sa razmakom manjim od DELAY otkucaja između portova". Varijable COUNT i DELAY mogu se proizvoljno konfigurirati. Skeniranjem TCP porta se smatra primanje paketa kojima ACK bit nije postavljen.

6. Bilježenje rezultata (D-box)

Bez obzira da mjesto gdje se nalaze log datoteke (disk, udaljeni sustav ili čak pisač), prostor rezerviran za njih je ograničen. Kada je prostor pun podaci se gube; ili se zapisivanje prekida ili se stari podaci brišu.
Najočitiji napad je ispunjavanje log datoteka sa nevažnim informacijama, a zatim stvarno napasti dok je sustav za detekciju efektivno onesposobljen. Za skeniranje portova, lažna skeniranja mogu poslužiti za ispunjavanje logova, a zatim se može napraviti stvarno skeniranje, sa mogućom kasnijom kompromitacijom napadnutog sustava. Očito je da loše projektirani alati za detekciju skeniranja portova mogu biti iskorišteni za izbjegavanje bilježenja pokušaja provale, što bi eventualno bilo zabilježeno da alat nije bio pokrenut.
Jedno od rješenja ovog problema je postavljanje ograničenja na frekvenciju bilježenja (npr. ne više od 5 poruka u 20 sekundi) svake vrste napada posebno, a kada se ta granica dostigne u log zabilježiti i tu činjenicu i privremeno prestati bilježiti napade te vrste. Za napade kod kojih se izvorišna adresa ne može lažirati gore opisana ograničenja mogu se postaviti prema adresi izvora. Pošto se prilikom skeniranja portova izvorišna adresa može lažirati, napadač može sakriti svoju stvarnu adresu, ali ukoliko se koristi ograničenje frekvencije bilježenja, ne može sakriti vrstu napada koju koristi.
Drugo rješenje, koje ima slične prednosti i nedostatke jest alociranje zasebnog prostora za poruke za svaku pojedinu vrst napada. Oba opisana rješenja mogu se i kombinirati.

7. Odgovor na napad (R-box)

Neki sustavi za detekciju neovlaštenih aktivnosti mogu odgovarati na napade koje detektiraju. Akcije koje poduzimaju obično su usmjerene prema prevenciji daljnjih napada i/ili dobivanju dodatnih informacija o napadaču. Nažalost, te karakteristike mudar napadač može i zlorabiti.

Tipična akcija koja se poduzima je blokiranje napadačkog računala (rekonfiguracija pristupnih listi vatrozida ili slično). Ova akcija može očito dovesti do napada odbijanja usluge (eng. *Denial of Service* – DoS) ukoliko se napad koji je detektiran može lažirati (za skeniranje portova to je moguće). Manje je uočljivo da to može dovesti do DoS ranjivosti i kod napada koji se ne mogu lažirati. Razlog tomu je što su IP adrese ponekad dijeljene između više osoba (dinamičko dodjeljivanje adresa).
Uz ovakav pristup postoje i neki implementacijski problemi, kao npr. ograničena veličina pristupnih lista na vatrozidu i sl. Također tu je problem korištenja CPU-a. Ukoliko sustav za detekciju neovlaštenih aktivnosti ne obraća pažnju na ta pitanja, to može dovesti do DoS napada na cijelu mrežu (ukoliko se npr. vatrozid sruši).
Očito je da blokiranje napadačkog računala nije nužno najsretnije rješenje za slučaj napada skeniranjem portova.
Druga uobičajena akcija je uspostavljanje veze sa napadačkim računalom u cilju dobivanja dodatnih informacija. Za napade koji se mogu lažirati to može dovesti do toga da se treća strana smatra napadnutom, što nikako nije poželjno.
Za napade koji se ne mogu lažirati ovakva akcija se može implementirati, ali vrlo oprezno. Treba biti pažljiv sa trošenjem resursa (propusnost mreže, CPU vrijeme, memorija) odnosno ograničiti broj zahtijeva koji se šalju i podesiti određeno vrijeme neaktivnosti.
Kôd na kraju dokumenta, zbog svih gore navedenih razloga ne koristi nikakve akcije kao odgovor na detekciju napada.

8. Strukture podataka i izbor algoritama

Prilikom nalaženja odluke o sortiranju ili korištenju algoritma za pronalaženje podataka, obično se vodi računa o specifičnom problemu. U sustavima za detekciju neovlaštenih aktivnosti uvijek se mora uzeti u obzir najgora mogućnost; napadač može opskrbiti sustav gotovo proizvoljnim podacima. Ukoliko sustav ostaje otvoren prilikom pada, napadač ga može obići, a ukoliko se zatvara napadač može izazvati uskraćivanje usluge na čitavom zaštićenom segmentu.
To se može ilustrirati na primjeru priloženom na kraju dokumenta. `Scanlogd` program koristi se *hash* tabelom za pronalaženje izvornih adresa; to radi dobro za slučaj kada je ta tabela dovoljno velika (broj adresa koje se čuvaju je ipak ograničen). Prosječno vrijeme pronalaženja je bolje od binarnog pretraživanja. No napadač može odabrati adrese (obično lažirane) da uzrokuje kolizije u tabeli, efektivno zamjenjujući pretraživanje *hash* tabele linearnim pretraživanjem. Ovisno o broju elemenata koji se čuvaju u tabeli, može se dogoditi da program ne uspijeva prihvatiti nove pakete na vrijeme. To će na računalno orijentiranim sustavima uzrokovati oduzimanjem procesorskog vremena drugim procesima.
U priloženom programu taj problem se rješava ograničavanjem maksimalnog broja kolizija u hash tabeli i odbacivanjem najstarijih podataka sa istim *hash* vrijednostima kada je maksimalni broj kolizija dostignut. To je prihvatljivo za skeniranje portova, ali ne mora biti prihvatljivo i za detekciju drugih vrsta napada.
Ukoliko se u programu koriste instrukcije poput `malloc` ili `free` iz `libc` biblioteke , napadač može pokušati iskoristiti slabosti na sličan način. To može uključivati pitanje iskorištenja procesora i "curenja" memorije. Pouzdani sustav za detekciju neovlaštenih aktivnosti mora imati vlastito upravljanje memorijom i biti posebno obazriv s njenom alokacijom.
Također valja spomenuti da se slična pitanja nameću i kod jezgri operacijskih sustava. Na primjer, *hash* tabele se široko koriste sa pronalaženje aktivnih veza, portova za primanje i slično. Uglavnom postoje druga ograničenja koja ne čine ova pitanja toliko opasnima, ali oprez nikad nije suvišan.

9. Sustavi za detekciju neovlaštenih aktivnosti i drugi procesi

Za mrežno orijentirane sustave za detekciju neovlaštenih aktivnosti, isto kao i za računalno orijentirane sustave, uvijek postoji neka razina interakcije sa ostatkom sustava, uključujući pri tome druge procese i jezgru operacijskog sustava.
Neki oblici DoS napada na operacijske sustave mogu onemogućiti i neke sustave za detekciju neovlaštenih aktivnosti (samo tzv. *fail-open* sustave), čak i bez upozorenja. Isto tako loše kôdirani

alati za detekciju neovlaštenih aktivnosti mogu biti iskorišteni za DoS napade na druge procese sustava.

10. Primjer programskog kôda

Program `scanlogd` namijenjen je prvenstveno Linux operacijskim sustavima. Prevođenje je moguće i u nekim drugim okolinama, ali vjerojatno neće raditi zbog nedostatka čistih TCP *socketa*.

```
<++> Scanlogd/scanlogd.c
/*
 * Linux scanlogd v1.0 by Solar Designer. You're allowed to do
whatever you
 * like with this software (including re-distribution in any form,
with or
 * without modification), provided that credit is given where it is
due, and
 * any modified versions are marked as such. There's absolutely no
warranty.
 */

#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>
#include <ctype.h>
#include <time.h>
#include <syslog.h>
#include <sys/times.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in_systm.h>
#include <netinet/in.h>
#if (linux)
#define __BSD_SOURCE
#endif
#include <netinet/ip.h>
#include <netinet/tcp.h>
#include <arpa/inet.h>

/*
 * Port scan detection thresholds: at least COUNT ports need to be
scanned
 * from the same source, with no longer than DELAY ticks between
ports.
 */
#define SCAN_COUNT_THRESHOLD          10
#define SCAN_DELAY_THRESHOLD          (CLK_TCK * 5)

/*
 * Log flood detection thresholds: temporarily stop logging if more
than
 * COUNT port scans are detected with no longer than DELAY between
them.
 */
#define LOG_COUNT_THRESHOLD           5
#define LOG_DELAY_THRESHOLD           (CLK_TCK * 20)

/*
```

```
 * You might want to adjust these for using your tiny append-only
log file.
 */
#define SYSLOG_IDENT                    "scanlogd"
#define SYSLOG_FACILITY                 LOG_DAEMON
#define SYSLOG_LEVEL                    LOG_ALERT

/*
 * Keep track of up to LIST_SIZE source addresses, using a hash
table of
 * HASH_SIZE entries for faster lookups, but limiting hash
collisions to
 * HASH_MAX source addresses per the same hash value.
 */
#define LIST_SIZE               0x400
#define HASH_LOG                11
#define HASH_SIZE               (1 << HASH_LOG)
#define HASH_MAX                0x10

/*
 * Packet header as read from a raw TCP socket. In reality, the TCP
header
 * can be at a different offset; this is just to get the total size
right.
 */
struct header {
  struct ip ip;
  struct tcphdr tcp;
  char space[60 - sizeof(struct ip)];
};

/*
 * Information we keep per each source address.
 */
struct host {
  struct host *next;            /* Next entry with the same hash */
  clock_t timestamp;            /* Last update time */
  time_t start;                 /* Entry creation time */
  struct in_addr saddr, daddr;  /* Source and destination addresses
*/
  unsigned short sport;         /* Source port, if fixed */
  int count;              /* Number of ports in the list */
  unsigned short ports[SCAN_COUNT_THRESHOLD - 1]; /* List of ports */
  unsigned char flags_or;       /* TCP flags OR mask */
  unsigned char flags_and;      /* TCP flags AND mask */
  unsigned char ttl;            /* TTL, if fixed */
};

/*
 * State information.
 */
struct {
  struct host list[LIST_SIZE];  /* List of source addresses */
  struct host *hash[HASH_SIZE]; /* Hash: pointers into the list */
  int index;              /* Oldest entry to be replaced */
} state;

/*
```

```
 * Convert an IP address into a hash table index.
 */
int hashfunc(struct in_addr addr)
{
 unsigned int value;
 int hash;

 value = addr.s_addr;
 hash = 0;
 do {
       hash ^= value;
 } while ((value >>= HASH_LOG));

 return hash & (HASH_SIZE - 1);
}

/*
 * Log this port scan.
 */
void do_log(struct host *info)
{
 char s_saddr[32];
 char s_daddr[32 + 8 * SCAN_COUNT_THRESHOLD];
 char s_flags[8];
 char s_ttl[16];
 char s_time[32];
 int index, size;
 unsigned char mask;

/* Source address and port number, if fixed */
 snprintf(s_saddr, sizeof(s_saddr),
       info->sport ? "%s:%u" : "%s",
       inet_ntoa(info->saddr),
       (unsigned int)ntohs(info->sport));

/* Destination address, if fixed */
 size = snprintf(s_daddr, sizeof(s_daddr),
       info->daddr.s_addr ? "%s ports " : "ports ",
       inet_ntoa(info->daddr));

/* Scanned port numbers */
 for (index = 0; index < info->count; index++)
       size += snprintf(s_daddr + size, sizeof(s_daddr) - size,
             "%u, ", (unsigned int)ntohs(info->ports[index]));

/* TCP flags: lowercase letters for "always clear", uppercase for
"always
 * set", and question marks for "sometimes set". */
 for (index = 0; index < 6; index++) {
       mask = 1 << index;
       if ((info->flags_or & mask) == (info->flags_and & mask)) {
             s_flags[index] = "fsrpau"[index];
             if (info->flags_or & mask)
                   s_flags[index] = toupper(s_flags[index]);
       } else
             s_flags[index] = '?';
 }
 s_flags[index] = 0;
```

```
/* TTL, if fixed */
  snprintf(s_ttl, sizeof(s_ttl), info->ttl ? ", TTL %u" : "",
        (unsigned int)info->ttl);

/* Scan start time */
  strftime(s_time, sizeof(s_time), "%X", localtime(&info->start));

/* Log it all */
  syslog(SYSLOG_LEVEL,
        "From %s to %s..., flags %s%s, started at %s",
        s_saddr, s_daddr, s_flags, s_ttl, s_time);
}

/*
 * Log this port scan unless we're being flooded.
 */
void safe_log(struct host *info)
{
  static clock_t last = 0;
  static int count = 0;
  clock_t now;

  now = info->timestamp;
  if (now - last > LOG_DELAY_THRESHOLD || now < last) count = 0;
  if (++count <= LOG_COUNT_THRESHOLD + 1) last = now;

  if (count <= LOG_COUNT_THRESHOLD) {
        do_log(info);
  } else if (count == LOG_COUNT_THRESHOLD + 1) {
        syslog(SYSLOG_LEVEL, "More possible port scans follow.\n");
  }
}

/*
 * Process a TCP packet.
 */
void process_packet(struct header *packet, int size)
{
  struct ip *ip;
  struct tcphdr *tcp;
  struct in_addr addr;
  unsigned short port;
  unsigned char flags;
  struct tms buf;
  clock_t now;
  struct host *current, *last, **head;
  int hash, index, count;

/* Get the IP and TCP headers */
  ip = &packet->ip;
  tcp = (struct tcphdr *)((char *)packet + ((int)ip->ip_hl << 2));

/* Sanity check */
  if ((char *)tcp + sizeof(struct tcphdr) > (char *)packet + size)
        return;

/* Get the source address, destination port, and TCP flags */
```

```
 addr = ip->ip_src;
 port = tcp->th_dport;
 flags = tcp->th_flags;

/* We're using IP address 0.0.0.0 for a special purpose here, so
don't let
 * them spoof us. */
 if (!addr.s_addr) return;

/* Use times(2) here not to depend on someone setting the time while
we're
 * running; we need to be careful with possible return value
overflows. */
 now = times(&buf);

/* Do we know this source address already? */
 count = 0;
 last = NULL;
 if ((current = *(head = &state.hash[hash = hashfunc(addr)])))
 do {
       if (current->saddr.s_addr == addr.s_addr) break;
       count++;
       if (current->next) last = current;
 } while ((current = current->next));

/* We know this address, and the entry isn't too old. Update it. */
 if (current)
 if (now - current->timestamp <= SCAN_DELAY_THRESHOLD &&
   now >= current->timestamp) {
/* Just update the TCP flags if we've seen this port already */
       for (index = 0; index < current->count; index++)
       if (current->ports[index] == port) {
             current->flags_or |= flags;
             current->flags_and &= flags;
             return;
       }

/* ACK to a new port? This could be an outgoing connection. */
       if (flags & TH_ACK) return;

/* Packet to a new port, and not ACK: update the timestamp */
       current->timestamp = now;

/* Logged this scan already? Then leave. */
       if (current->count == SCAN_COUNT_THRESHOLD) return;

/* Update the TCP flags */
       current->flags_or |= flags;
       current->flags_and &= flags;

/* Zero out the destination address, source port and TTL if not
fixed. */
       if (current->daddr.s_addr != ip->ip_dst.s_addr)
             current->daddr.s_addr = 0;
       if (current->sport != tcp->th_sport)
             current->sport = 0;
       if (current->ttl != ip->ip_ttl)
             current->ttl = 0;
```

```
/* Got enough destination ports to decide that this is a scan? Then
log it. */
        if (current->count == SCAN_COUNT_THRESHOLD - 1) {
                safe_log(current);
                current->count++;
                return;
        }

/* Remember the new port */
        current->ports[current->count++] = port;

        return;
  }

/* We know this address, but the entry is outdated. Mark it unused,
and
 * remove from the hash table. We'll allocate a new entry instead
since
 * this one might get re-used too soon. */
  if (current) {
        current->saddr.s_addr = 0;

        if (last)
                last->next = last->next->next;
        else if (*head)
                *head = (*head)->next;
        last = NULL;
  }

/* We don't need an ACK from a new source address */
  if (flags & TH_ACK) return;

/* Got too many source addresses with the same hash value? Then
remove the
 * oldest one from the hash table, so that they can't take too much
of our
 * CPU time even with carefully chosen spoofed IP addresses. */
  if (count >= HASH_MAX && last) last->next = NULL;

/* We're going to re-use the oldest list entry, so remove it from
the hash
 * table first (if it is really already in use, and isn't removed
from the
 * hash table already because of the HASH_MAX check above). */

/* First, find it */
  if (state.list[state.index].saddr.s_addr)
        head = &state.hash[hashfunc(state.list[state.index].saddr)];
  else
        head = &last;
  last = NULL;
  if ((current = *head))
  do {
        if (current == &state.list[state.index]) break;
        last = current;
  } while ((current = current->next));
```

```
/* Then, remove it */
 if (current) {
       if (last)
             last->next = last->next->next;
       else if (*head)
             *head = (*head)->next;
 }

/* Get our list entry */
 current = &state.list[state.index++];
 if (state.index >= LIST_SIZE) state.index = 0;

/* Link it into the hash table */
 head = &state.hash[hash];
 current->next = *head;
 *head = current;

/* And fill in the fields */
 current->timestamp = now;
 current->start = time(NULL);
 current->saddr = addr;
 current->daddr = ip->ip_dst;
 current->sport = tcp->th_sport;
 current->count = 1;
 current->ports[0] = port;
 current->flags_or = current->flags_and = flags;
 current->ttl = ip->ip_ttl;
}

/*
 * Hmm, what could this be?
 */
int main()
{
 int raw, size;
 struct header packet;

/* Get a raw socket. We could drop root right after that. */
 if ((raw = socket(AF_INET, SOCK_RAW, IPPROTO_TCP)) < 0) {
       perror("socket");
       return 1;
 }

/* Become a daemon */
 switch (fork()) {
 case -1:
       perror("fork");
       return 1;

 case 0:
       break;

 default:
       return 0;
 }

 signal(SIGHUP, SIG_IGN);
```

```
/* Initialize the state. All source IP addresses are set to 0.0.0.0,
which
 * means the list entries aren't in use yet. */
 memset(&state, 0, sizeof(state));

/* Huh? */
 openlog(SYSLOG_IDENT, 0, SYSLOG_FACILITY);

/* Let's start */
 while (1)
 if ((size = read(raw, &packet, sizeof(packet))) >=
sizeof(packet.ip))
       process_packet(&packet, size);
}
<-->
```