Távoli operációs rendszer detektálás

Juhász Péter Károly "Stone"
<stone@elte.hu>
Modla Ferenc
<modla@inf.elte.hu>

2003. október 1.

Tartalomjegyzék

1. Miért jó ez az egész

Nyilvánvalóan hasznos dolog, ha el tudjuk dönteni, hogy egy távoli gépen milyen operációs rendszer fut. A legjobb példák a hasznosságra abból erednek, hogy a biztonsági rések erősen oprendszerfüggők. Vegyük például a következőt: Scannelünk egy gépet, és az 53-as port nyitva. Ezen keresztül egy jó TCP/IP fingerprinterrel könnyű kitalálni, hogy a gép pl. 'Linux 2.4.9'-et futtat, és ehhez igazíthatjuk a shellscriptet.

2. A triviális metódus

A legegyszerűbb eljárás természetesen nem az IP stack átnézése vagy különféle flagek előre eltárolt mintának való megfeleltetése. Sajnos még mindig a legjobban működő eljárás a következő:

```
evil@hell:/# telnet gep.egy.hu

                        Welcome all the Hitch Hikers
                                  to the
                               VAX-Galaxy !
                                                        Node Gep 1, VMS 7.2

Username:
```

Voi là! Rögtön láthatjuk, hogy *VMS 7.2* fut rajta, de ha mi látjuk, akkor más is láthatja és akkor célzott támadást tud indítani a gép ellen. Miért is bocsátkoznánk bele bonyolultabb technikák alkalmazásába, ha a gép rögtön elárulja magáról, hogy mit futtat? Sajnos sok gyártó marketing okokból ahová csak tudja, beleteszi a termék nevét, és a felelőtlen rendszergazdák ezeket nem távolítják el. Azért, mert több módszer is van a távoli oprendszer detektálásra, meg kell azt mindenkinek mondani, aki kapcsolódik hozzánk?

Ezzel a technikával az a baj, hogy azok, akik kicsit is adnak a biztonságra, kikapcsolják a bannereket, vagy az agyafúrtabbak akár hazudnak is. Némelyik program a bannereket file-ból veszi, itt csak a file-t kell átírni, néhol a forráskódot is módosítani kell, de őszintén szólva ez sem nagy feladat. De még ha ki is vannak kapcsolva a bannerek, sok olyan program van, ami készségesen elárulja magáról, hogy mi a neve és a verziója. Példának vegyünk egy FTP szervert:

```
evil@hell:/# telnet gep.ketto.hu 21
Trying 207.200.74.26...
Connected to gep.ketto.hu.
Escape character is '^]'.
220 ftp29 FTP server (UNIX(r) System V Release 4.0) ready.
SYST
215 UNIX Type: L8 Version: SUNOS
```

Amit nem árult el a banner, azt szívesen elmondta a szerver. De vannak más módszerek is. Ha hozzáférünk valahogy a file-okhoz, például egy publikus ftp szerver segítségével, és lementünk bármilyen bináris file-t, akkor arról meg lehet állapítani, hogy milyen arhitektúrára fordították. Sok más program is létezik, ami elárulja magáról, hogy kicsoda, például a web-szerverek. Lássunk egy példát:

```
evil@hell:/# telnet gep.harom.hu 80
Connected to gep.harom.hu.
Escape character is '^]'.
HEAD / HTTP/1.0

HTTP/1.0 302 Moved Temporarily
Location: http://gep.harom.hu/
```

```
Server: HTTPd-WASD/7.2.1 OpenVMS/AXP SSL
Date: Wed, 29 May 2002 16:41:13 GMT
Content-Type: text/html
```

Rögtön megtudtuk az operációs rendszer valamint a web-szerver típusát. Vagy nézzük meg az IRC programokat:

```
[ctcp(userb)] VERSION
>>> usera [~usera@gep.negy.hu] requested VERSION from userb
--- CTCP VERSION relpy from [userb]: BitchX-75p1+ by panasync - AIX 3 : Keep
          it to yourself!
```

De ezen kívül még nagyon sok olyan alkalmazás van, ami szívesen közöl információkat a gépről.

3. Módszerek

Sokféle technika létezik IP-stack-ek fingerprintelésére. Alapvetően olyan dolgokat keresünk, amik különböznek az operációs rendszerek között, és írunk egy programot a különbségek felderítésére. Ha eleget kombinálunk ezekből, viszonylag pontosan meghatározhatjuk az oprendszer típusát. Általában megbízhatóan meg tudjuk különböztetni a Solaris 2.5-2.51-et a 2.6-tól, a Linux 2.4.0-2.4.18, 2.4.3, 2.4.7 és a 2.4.9 verziókat vagy éppen a Windows 3.1-et a Win 9x-től és az NT 5-től.

A módszerek illusztrálására a *Perl* `RawIP` csomagját használjuk. A teszteredmények vizsgálatára bármilyen *sniffer* program használható, mi az *Ethereal*-t valamint egy saját, szintén perl-ben írt programot használunk, ami a következő (természetesen a különféle teszteknél néha változtatjuk a programot, attól függően, hogy melyik mezőjére vagyunk kíváncsiak a csomagnak):

```
#!/usr/bin/perl

use Net::RawIP qw(:pcap);

$dev = "lo";

$packet_tcp = new Net::RawIP({tcp=>{}});
$packet_udp = new Net::RawIP({udp=>{}});
$packet_icmp = new Net::RawIP({icmp=>{}});
$filter_tcp = "ip proto \\tcp";
$filter_udp = "ip proto \\udp";
$filter_icmp = "ip proto \\icmp";
$pcap_tcp = $packet_tcp->pcapinit($dev,$filter_tcp,1500,60);
$pcap_udp = $packet_udp->pcapinit($dev,$filter_udp,1500,60);
$pcap_icmp = $packet_icmp->pcapinit($dev,$filter_icmp,1500,60);
$offset_tcp = linkoffset($pcap_tcp);
$offset_udp = linkoffset($pcap_udp);
$offset_icmp = linkoffset($pcap_icmp);
if (fork){ loop $pcap_tcp,-1,\&sniff_tcp,\@s;}
if (fork){ loop $pcap_udp,-1,\&sniff_udp,\@s;}
if (fork){ loop $pcap_icmp,-1,\&sniff_icmp,\@s;}

sub sniff_tcp {
  $packet_tcp->bset($_[2],$offset_tcp);
  my ($version,$ihl,$tos,$id,$frag_off,$ttl,$protocol,$check1,$saddr,$daddr)=
        $packet_tcp->get({ip=>['version','ihl','tos','id','frag_off',
        'ttl','protocol','check','saddr','daddr']}
     );
  my ($source,$dest,$seq,$ack_seq,$doff,$res1,$res2,$urg,$ack,$psh,$rst,$syn,
```

```
      $fin,$window,$check2,$urg_ptr,$data)=
        $packet_tcp->get({tcp=>['source','dest','seq','ack_seq','doff',
        'res1','res2','urg','ack','psh','rst','syn','fin','window','check',
        'urg_ptr','data']}
      );
  print "TCP ";
  if ($urg) { print "U"};
  if ($ack) { print "A"};
  if ($psh) { print "P"};
  if ($rst) { print "R"};
  if ($syn) { print "S"};
  if ($fin) { print "F"};
  print " " . ip2name($saddr) . ":$source -> ". ip2name($daddr) . ":$dest ";
  print "TOS: $tos ";
  printf "SEQ: %x " , $seq;
  printf "ACK_SEQ: %x ", $ack_seq;
#  printf "WINDOW: %x ", $window;
  print "\n";
}

sub sniff_udp {
  $packet_udp->bset($_[2],$offset_udp);
  my ($version,$ihl,$tos,$id,$frag_off,$ttl,$protocol,$check1,$saddr,$daddr)=
        $packet_udp->get({ip=>['version','ihl','tos','id','frag_off',
        'ttl','protocol','check','saddr','daddr']}
      );
  my ($source,$dest,$len,$check2,$data)=
      $packet_udp->get({udp=>['source','dest','len','check','data']}
      );
  print "UDP " . ip2name($saddr) . ":$source -> ". ip2name($daddr) . ":$dest\n";
}

sub sniff_icmp {
  $packet_icmp->bset($_[2],$offset_icmp);
  my ($version,$ihl,$tos,$id,$frag_off,$ttl,$protocol,$check1,$saddr,$daddr)=
        $packet_icmp->get({ip=>['version','ihl','tos','id','frag_off',
        'ttl','protocol','check','saddr','daddr']}
      );
  my ($type,$code,$check2,$gateway,$id,$sequence,$unused,$mtu,$data)=
        $packet_icmp->get({icmp=>['type','code','check','gateway',
        'id','sequence','unused','mtu','data']}
      );
  print "ICMP " . ip2name($saddr) . " -> ". ip2name($daddr) . " ";
  print "TOS: $tos ";
  printf "TYPE: %x " , $type;
  printf "CODE: %x " , $code;
  printf "ID: %x " , $id;
  printf "SEQ: %x " , $sequence;
  print "\n";
}

sub ip2name {
  my $addr = shift;
  (gethostbyaddr(pack("N",$addr),AF_INET))[0] || ip2dot($addr);
```

```
}

sub ip2dot {
  sprintf("%u.%u.%u.%u",unpack "C4", pack "N1", shift);
}
```

3.1. A FIN próba

Küldünk egy FIN csomagot (vagy bármilyen másikat ACK vagy SYN flag nélkül) egy nyitott portra, és várjuk a reakciót. A helyes RFC793 viselkedés: nem reagálni, de sok szabványtól eltérő megvalósítás, mint pl.: a MS Windows, a BSDI, a CISCO, a HP/UX, az VMS, és az IRIX egy RESET-et küld vissza. A legtöbb alkalmazás használja ezt a módszert.

Íme a tesztprogram:

```
#!/usr/bin/perl
use Net::RawIP qw(:pcap);

$daddr = "172.16.1.4"; $saddr = "172.16.1.1"; $sport = "10000"; $dport = "6346";

$s = new Net::RawIP;
$s->set({ip => {daddr => $daddr, saddr => $saddr},
         tcp => {dest => $dport, source => $sport, fin => 1}});
         $s->send;
```

A fenti program úgy van paraméterezve, hogy a mi gépünk (172.16.1.1) 10000-es portjáról küldjön egy csomagot a 172.16.1.4-es című gép 6346-os portjára. A csomagban csak a FIN flag van beállítva.

És az eredmények, Windows 2000-es gép esetén:

```
TCP F 172.16.1.1:10000 -> 172.16.1.4:6346 TOS: 16 SEQ: 0 ACK_SEQ: 0
TCP AR 172.16.1.4:6346 -> 172.16.1.1:10000 TOS: 0 SEQ: 0 ACK_SEQ: 1
```

láthatjuk, hogy a gép egy ACK RST csomagot küldött vissza, de 2.4 esetén:

```
TCP F 172.16.1.1:10000 -> 172.16.1.1:6346 TOS: 16 SEQ: 0 ACK_SEQ: 0
```

nem kaptunk választ.

3.2. A hibás flag próba

Ez egy ötletes teszt, aminek a lelke egy definiálatlan TCP 'flag' (64 vagy 128) beállítása egy SYN csomag TCP fejlécében. A Linuxok 2.0.35-ig megtartják ezt a beállított flag-et a válaszukban. Nem tudunk másik rendszerről, ami rendelkezik ezzel a hibával. Néhányan viszont megszakítják a kapcsolatot, ha ilyen csomagot kapnak. Ez a viselkedás segítségünkre lehet az azonosításukban.

3.3. TCP ISN mintavétel

Az ötlet az, hogy találjunk szabályosságokat a TCP által választott ISN (kezdeti azonosítószám)-ekben, amikor a kapcsolatkérésre kapunk választ. Ezek a számok sok csoportba oszthatók, mint például a tradícionális 64K (régi UNIX rendszerek), véletlen növekedésű (IRIX, FreeBSD, Digital UNIX, Cray, Solaris újabb verziói, és még sokan mások), igazi véletlen (Linux 2.*, OpenVMS, újabb AIX). Windows-os gépek (és néhány másik) egy időfüggő módszert alkalmaznak, ahol az ISN minden időegység alatt meghatározott mértékben növekszik. Nem mondunk sokat, ha azt állítjuk, hogy ezt sem nehezebb kihasználni a régi 64K-s viselkedésnél. Egy érdekes megoldás a konstans szám: A gép mindig ugyanazt az ISN-t használja. Ilyenek például egyes 3Com hub-ok (0x803) és Apple LaserWriter nyomtatók (0xC7001). Csoportosíthatjuk még pl. a véletlen növekményű számokat számítási változatok, legnagyobb közös osztók, és az azonosítókon, vagy különbségeiken vett egyéb függvények szerint is.

3.4. A don't fragment bit

Az IP csomag fejlécében ez a bit azt mondja meg a gépnek, hogy ha találkozik a csomaggal, ne darabolja fel kisebb csomagokra. Ez akkor lenne szükséges, ha a csomag túl nagy lenne, és az aktuális oprendszer nem tud kezelni akkora csomagot. Ilyenkor feldarabolja azt kisebb csomagokra. Ha ez a bit be van állítva, akkor eldobja, és hibaüzenetet küld vissza. Ezt a bitet egyes oprendszerek beállítják, mások csak esetenként, hatékonysági és egyéb okokat is figyelembe véve. Kitartó jegyzeteléssel – ki mikor állítja be –, és figyeléssel információkat nyerhetünk az adott rendszer típusáról.

Megfigyeléseim szerint általában mindenki beállítja, de a Windows 2000-es gépek a RST csomagokban nem állítják be.

3.5. TCP kezdő ablak méret

Mi is ez az "ablak"? Az IP csomag fejlécében ez a szám mondja meg, hogy az adott gép mennyi fogadott csomag után küld vissza, egy "A csomagokat vettem" ACK csomagot. Egyszerű ezt a számot ellenőrizni a csomagban. Mivel ez egy az operációs rendszerekre jellemző konstans, sok oprendszert be lehet azonosítani egyedül ennek a számnak az ismerete alapján. Pl.: az AIX ezt 0x4000-re állítja, a Microsoft NT5 0x402E-re (ezt használja egyébként az OpenBSD és a FreeBSD is), a VMS 0x0bb8-ra valamint a Linux 2.4 0x7fff-re.

Ezt egyszerűen lehet tesztelni, én például a cél gépről betelneteltem egy véletlenszerű portjára a gépemnek, ahol futtattam a snifferemet. Az eredmény ez lett:

Linuxról a következő lett az eredmény:

```
TCP S 172.16.1.9:32920 -> 172.16.1.1:543 WINDOW: 7fff
```

VMS-röl:

```
TCP S 172.16.1.6:32920 -> 172.16.1.1:543 WINDOW: 0bb8
```

valamint AIX-röl:

```
TCP S 172.16.1.8:32920 -> 172.16.1.1:543 WINDOW: 4000
```

3.6. ICMP Hibaüzenetek korlátozása

Az 1812-es RFC-ben vannak ajánlások arra vonatkozóan, hogy az oprendszer mennyi ICMP hibaüzenetet küldjön vissza. Ezt néhány oprendszer meg is fogadja. Például a Linux a *cél nem elérhető* (destination unreachable) üzeneteket 4 másodpercenként 80 darabra korlátozza, 1/4 másodperc büntetéssel ha átlépné a határt. Egy jó módszer az oprendszer megállapításához, ha egy nem nyitott UDP portra küldünk egy nagyobb mennyiségű csomagot, és számoljuk, hogy mennyi hibaüzenet jön vissza. Sajnos ez egy nagyon időigényes dolog, mert sok csomagot kell kiküldeni, és várni, hogy visszaérkezzenek.

3.7. Az ACK értéke

Ez a 32 bites szám azt jelöli (ha az ACK bit be van állítva), hogy milyen sorszámú csomagot vár a küldő. Azt gondolhatnánk, hogy ez minden implementációban ugyanaz, de nem. Például: ha küldünk egy olyan csomagot egy zárt TCP portra, amiben a következő flag-ek vannak beállítva: FIN, PSH és URG. A legtöbb implementáció az ACK-t a mi SEQ-ünkre állítja, de a Windows, Solaris 2.8 és egy-két hülye printer ISN+1-et küld vissza. Ha egy olyan csomagot küldünk egy nyitott portra, aminek a következő flag-jei vannak beállítva: SYN, FIN, URG és PSH, akkor a Windows nagyon kiszámíthatatlanul viselkedik. Néha visszaküldi az ISN-ünket, néha ISN+1-et, és van úgy, hogy egy teljesen véletlen számot.

Az eredmények FIN, PSH, URG csomaggal, Solaris 2.8-at futtató gép esetén:

```
TCP FPU 172.16.1.1:10000 -> 172.16.1.4:81 SEQ: 666 ACK_SEQ: 0 WINDOW: 65535
TCP AR 172.16.1.4:81 -> 172.16.1.1:10000 SEQ: 0 ACK_SEQ: 667 WINDOW: 65535
```

VMS 7.2 esetén:

```
TCP FPU 172.16.1.1:10000 -> 172.16.1.4:81 SEQ: 666 ACK_SEQ: 0 WINDOW: 65535
TCP AR 172.16.1.4:81 -> 172.16.1.1:10000 SEQ: 0 ACK_SEQ: 666 WINDOW: 0
```

valamint Linux 2.2 esetén:

```
TCP FPU 172.16.1.1:10000 -> 172.16.1.5:81 SEQ: 666 ACK_SEQ: 0 WINDOW: 65535
TCP AR 172.16.1.5:81 -> 172.16.1.1:10000 SEQ: 0 ACK_SEQ: 666 WINDOW: 0
```

Látható, hogy a Solaris SEQ + 1-et küldött a válaszban amíg a Linux és a VMS SEQ-et.

Most lássuk a teszteredményeket SYN, FIN, PSH, URG csomaggal, Solaris 2.8-at futtató gép esetén:

```
TCP SFPU 172.16.1.1:10000 -> 172.16.1.4:81 SEQ: 666 ACK_SEQ: 0 WINDOW: 65535
TCP AR 172.16.1.4:81 -> 172.16.1.1:10000 SEQ: 0 ACK_SEQ: 668 WINDOW: 65535
```

VMS 7.2 esetén:

```
TCP SFPU 172.16.1.1:10000 -> 172.16.1.4:81 SEQ: 666 ACK_SEQ: 0 WINDOW: 65535
TCP AR 172.16.1.4:81 -> 172.16.1.1:10000 SEQ: 0 ACK_SEQ: 667 WINDOW: 0
```

valamint Linux 2.2 esetén:

```
TCP SFPU 172.16.1.1:10000 -> 172.16.1.5:81 SEQ: 666 ACK_SEQ: 0 WINDOW: 65535
TCP AR 172.16.1.5:81 -> 172.16.1.1:10000 SEQ: 0 ACK_SEQ: 667 WINDOW: 0
```

Itt is látható, hogy a Solaris SEQ + 2-t küldött vissza, a többiek SEQ + 1-jével szemben.

Érdekes megfigyelni a kezdő ablak méretét a válaszokban, a Solaris beállítja őket, míg mások nem.

3.8. Az ICMP üzenet idézete

Az RFC-kben le van írva, hogy az ICMP üzenetekben egy kicsit kell idézni abból a csomagból, amire küldjük válaszul. Egy *port nem elérhető* (port unreachable) hibaüzenetben majdnem az összes implementáció csak az IP fejlécét és 8 byte-ot küld vissza. De a Solaris egy kicsivel több bitet, a Linux pedig még egy kicsivel többet. Ebben az a gyönyörű, hogy akkor is meg lehet mondani egy gépről, hogy Linuxot vagy Solarist futtat, ha egyetlen port sincsen nyitva rajta.

3.9. ICMP hibaüzenetben a „visszhang" sértetlensége

Mint az előző pontban írtam, az ICMP *port nem elérhető* hibaüzenetekbe bele kell foglalni az eredeti üzenet egy részét. Egyes oprendszerek a mi üzenetünk fejlécét használják átmeneti tárnak a válasz készítése során, ezért időnként egy kicsit kuszán kapjuk vissza. Például az AIX és a BSDI az IP - *Teljes hossz* (total length) mezőt hússzal többre állítják mint kellene. Néhány BSDI, FreeBSD, OpenBSD, ULTRIX, és VAX elrontja az általunk küldött IP ID-jét. Az ellenőrző összegnek (checksum) is változnia kell a megváltozott TTL (Time To Live) miatt, de van egy-két gép, például az AIX, FreeBSD és még mások, akik vagy rosszat küldenek, vagy egyszerűen 0-át. Ugyanezt teszik az UDP ellenőrző összeggel is.

3.10. A szolgáltatás típusa

A *Type of Service* (TOS) értéke az ICMP *port elérhetetlen* hibaüzenetekben általában 0, de a régi Linux-ok 0xC0-át használtak, most 0xD0-át használnak. Nem lehet tudni, hogy miért csinálja ezt a Linux, de így legalább különbséget lehet tenni a régi és az új között.

3.11. TCP opciók

Ez egy nagyon jó információ-lelőhely. A szépsége az, hogy nem mindenki implmentálta ezt eddig (legalábbis sok régebbi oprendszer van, melyek még nem támogatták teljesen), valamint ha valakinek küldünk egy csomagot, amiben benne van, hogy támogatjuk ezeket az opciókat, a célpont válaszában benne lesz, hogy ő támogatja-e őket, és ha igen, akkor pontosan melyiket.

El kell küldeni egy csomagot, melyben minden be van állítva, és megnézni, hogy mit válaszol, melyeket támogatja. Pl.: A FreeBSD mindet támogatja, a régebbi, 2.0.x Linuxok viszont csak nagyon keveset, de a 2.1.x óta a Linux is támogatja mindet.

Lehet, hogy több oprendszer is támogatja ugyanazon opciókat, de akkor is megkülönböztethetők az opciók értékei alapján. Pl.: ha küldünk egy csomagot kicsi MSS (Maximum Segment Size)-al, egy Linuxos gépnek, akkor ugyanazt az MSS-t küldi vissza a válaszban. Más oprendszerek mást fognak visszaküldeni.

Ha ugyanazon szolgáltatások vannak is implementálva, és ugyanolyanok az értékek, még akkor is találhatunk különbséget abban, hogy milyen sorrendben küldik őket.

3.12. Darabok kezelése

Ez a technika arra épül, hogy a különböző oprendszerek különféle módon kezelik az egymást átfedő IP darabkákat. Néhányan felülírják a régebben érkezőt, és van, ahol az újból dobják el az átfedő részt.

Következzen egy példaprogram ennek a technikának a kihasználására. Teszteléshez olyan portot válasszunk, ami nyitva van a célponti gépen. A program két darabból álló csomagot készít, aminek a darabkái átfedik egymást, konkrétan a TCP flag-eknél, az elöbb küldött darabkába a SYN flag van beállítva, a másodikban a FIN. Ha a gép úgy rakja össze, hogy az átfedő és később érkező darabot eldobja, akkor SYN csomagot kap és erre SYN-ACK-val válaszol. Ha rosszul rakja össze, azaz az újonnan érkező átfedő résszel felülírja az előzőt, akkor egy FIN csomagot kap, és erre – ha betartja az RFC-t – nem küld vissza semmit, de ha nem tartja be, akkor is csak egy RST-et küld.

Sajnos nem sikerült találnom olyan gépet ami rosszul rakja össze a csomgot, de a technika biztosan működik, főleg a régebbi gépeknél, mivel sok DOS támadás (pl.: Teardrop) is azon alapult, hogy az átfedő csoamgokat ki hogyan rakja össze.

A program ha paraméterek nélkül indítjuk kiírja a használati utasításait.

```
#include <string.h>
#include <sys/types.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
#include <sys/select.h>
#include <stdio.h>

struct {
  unsigned int src;
  unsigned int dst;
  unsigned char dummy;
  unsigned char proto;
  unsigned short len;
  struct tcphdr tcp;
} pseudohdr;
```

```
struct {
  struct iphdr ip;
  struct tcphdr tcp;
} packet;

unsigned short cksum(unsigned short *addr, int len) {
  register int nleft = len;
  register unsigned short *w = addr;
  register int sum = 0;
  unsigned short answer = 0;
  while (nleft > 1) {
    sum += *w++;
    nleft -= 2;
  }
  if (nleft == 1) {
    *(u_char *)(&answer) = *(u_char *)w ;
    sum += answer;
  }
  sum = (sum >> 16) + (sum & 0xffff);
  sum += (sum >> 16);
  answer = ~sum;
  return(answer);
}

int main(int argc, char* argv[]) {
  char *outpacket;
  char *p;
  int one = 1, i;
  const int *val = &one;
  int fd;
  struct sockaddr_in sin;
  struct timeval time;
  fd_set fds;

  if (argc < 4) {
    printf("Hasznalat: %s <sajat ip> <cel ip> <cel port>\n", argv[0]);
    exit(0);
  }

  outpacket = malloc(40);
  p = malloc(40);

  packet.ip.saddr = inet_addr(argv[1]);
  packet.ip.daddr = inet_addr(argv[2]);
  packet.ip.id = rand(); packet.ip.check = 0;
  packet.ip.protocol = 6; packet.ip.version = 4;
  packet.ip.ihl = 5; packet.ip.tos = 0x10;
  packet.ip.ttl = 64; packet.ip.tot_len = htons(20 + 16);
  packet.ip.frag_off = htons(IP_MF);

  packet.tcp.source = htons(57043); packet.tcp.dest = htons(argv[3]);
  packet.tcp.seq = rand(); packet.tcp.ack_seq = 0;
  packet.tcp.res1 = 0; packet.tcp.doff = 5; packet.tcp.res2 = 0;
  packet.tcp.syn = 1; packet.tcp.fin = 0; packet.tcp.rst = 0;
```

```
packet.tcp.psh = 0; packet.tcp.ack = 0; packet.tcp.urg = 0;
packet.tcp.window = htons(6000); packet.tcp.urg_ptr = 0;
packet.tcp.check = 0;

pseudohdr.len = htons(sizeof(struct tcphdr));
pseudohdr.src = inet_addr(argv[1]);
pseudohdr.dst = inet_addr(argv[2]);
pseudohdr.dummy = 0;
pseudohdr.proto = IPPROTO_TCP;
memcpy(&pseudohdr.tcp,&packet.tcp,sizeof(struct tcphdr));
packet.tcp.check = cksum((unsigned short *)&pseudohdr, sizeof(pseudohdr));

fd = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
sin.sin_family = AF_INET;
sin.sin_port = htons(argv[3]);
sin.sin_addr.s_addr = inet_addr(argv[2]);
setsockopt(fd, IPPROTO_IP, IP_HDRINCL, val, sizeof (one));

memset(outpacket, '\0', 40);
memcpy(p, &packet, 40);
memcpy(outpacket, p ,20 + 16);
sendto(fd, outpacket, 20 + 16, 0, (struct sockaddr*)&(sin), sizeof(sin));
printf("Elso darab elkuldve...\n");

memset(outpacket, '\0', 40);
packet.tcp.syn = 0; packet.tcp.fin = 1;
packet.ip.check = 0;
packet.ip.tot_len = htons(20 + 12);
packet.ip.frag_off = htons(8/8);
memcpy(p, &packet, 40);
memcpy(outpacket, p, 20);
memcpy(outpacket + 20, p + 20 + 8, 12);
sendto(fd, outpacket, 20 + 12, 0, (struct sockaddr*)&(sin), sizeof(sin));
printf("Masodik darab elkuldve...\n");
close(fd);

memset(&packet, '\0', sizeof(packet));
fd = socket(AF_INET, SOCK_RAW, IPPROTO_TCP);
FD_ZERO(&fds); FD_SET(fd, &fds);
time.tv_sec = 5; time.tv_usec = 0;
if (select(fd + 1, &fds, NULL, NULL, &time)) {
  i = sizeof(sin);
  recvfrom(fd, &packet, sizeof(packet), 0, (struct sockaddr *)&(sin), &i);
  if(packet.tcp.syn == 1 && packet.tcp.ack == 1) {
    printf("SYN-ACK jott valaszuk.\n");
  } else {
    printf("Jott valasz, de nem SYN-ACK.\n");
  }
} else {
  printf("Nem jott valasz 5 masodpercen belul.\n");
}
close(fd);

return 0;
```

```
}
```

3.13. Fejlődés történet[1]

A fenti tesztekkel nem lehet megkülönböztetni egymástól a Win95, Win98 valamint a WinNT gépeket, mert bár több év is eltelt a kiadások között, mégis ugyanazt a buta TCP vermet használják mind, de ne adjuk fel ilyen könnyen!

Kezdjük el szépen támadni őket DOS (Denial Of Service) támadásokkal (Ping of Death, Winnuke, Teardrop, Land, stb), és közben mindegyik után PING-eljük meg, hogy él-e még. Amikor meghal, akkor már tudni fogjuk, hogy milyen patch-ek vannak feltéve rá, és milyen verziószámú Windows fut rajta.

Ez tényleg nagyon csúnya dolog, szóval csak a saját gépünkön teszteljük!

3.14. SYN áradat védelem

Néhány rendszer nem fogad el több kapcsolatot, ha túl sok hamis SYN csomagot kap. Mivel egy kapcsolat úgy épül fel, hogy a kezdeményező gép küld egy SYN (synchronize/start) csomagot a cél gépnek, az visszaküld egy SYN ACK (synchronize acknowledge) csomagot és erre vár egy ACK (acknowledge) csomagot. Ha ez mind megtörtént akkor a kapcsolat felépül. De ha nem küldünk neki a SYN ACK csomagjára ACK csomagot, akkor sok félig kész kapcsolat kerül be a várakozási sorba. Ez általában maximum 8 hosszú (mivel egy ilyen kapcsolat felépítése pár millisecundum alatt létrejön). Ha betelik ez a várakozási sor, akkor a gép nem fogad több kapcsolatot. Némely oprendszer különböző módokon megpróbálja ezt kikerülni pl. SYN cookie-k használatával (lásd később). Szóval ha küldünk valakinek 8 db SYN csomagot, és aztán megpróbálunk hozzá kapcsolódni, az eredményből következtethetünk az oprendszerre.

3.14.1. Mi az a SYN cookie?

Egy sajátságos alternatíva a TCP sorszámra (sequence number) a szerver oldalán. A különbség a szerver és a kliens kezdő sorszáma (ISN) között a következő:

Felső 5 bit t mod 32, ahol a t egy 32 bites számláló ami 64 másodpercenként nő.

Következő 3 bit A kliens MSS-ére válaszul küldött, a szerver által választott MSS.

Alsó 24 bit A szerver által – egy titkos eljárással készített – kulcs, ami függ: a kliens IP címétől, port számától, a szerver IP címétől valamint portszámától és t-től.

Ez a választás eleget tesz annak a TCP elvárásnak, hogy az ISN lassan növekszik; a szerver ISN-je egy picit gyorsabban nő, mint a kliensé.

A szerver, aki SYN cookie-kat használ, ellenál a SYN áradatnak, és nem kell eldobnia egyetlen egy kapcsolatot sem, ha betelik a SYN sora. Mivel ha kap egy SYN csomagot, akkor rögtön küld egy SYN+ACK csomagot pontosan úgy, mintha a SYN sora nagyobb lenne. (Kivéve: a szervernek el kell utasítania az olyan TCP opciókat, mint például a nagyobb ablak (larger window).)

Amikor a server kap egy ACK csomagot, akkor a t értékéből és a kódolt MSS-ből – a titkos eljárásával – felépíti a SYN sort.

Ha a támadó ki tudja találni a sorozatszámot, amit a szerver küld egy másik kliensnek, akkor fel tud a kliens nevében építeni egy hamis kapcsolatot a szerverrel. Bár nincsen sok esélye arra, hogy kitalálja, de van rá esélye, hogy pár millió próba után eltalálja. Azért, hogy ezt megnehezítsük, a következő két módszerrel védekezhetünk:

1. Kövessük nyomon a SYN sor túlcsordulásokat (minden egyes sorra), és ne építsük újra a SYN-eket, ha éppen nincs túlcsordulás. Ezzel megakadályozhatjuk az ACK hamisítást.

2. Adjunk még egy 32 bites számot a cookie-hoz, ami a szerver és a kliens címéből jön ki egy titkos eljárással. Így a támadónak még kisebb esélye van.

[1] Ezt a módszert nem ajánljuk, mert nem etikus mások gépét lefagyasztani!

3.15. SYN-ACK időköz analízis

Ennek a technikának az előnye az, hogy gyors, nehéz észrevenni, mivel nem használ semmi különleges csomagot, valamint kicsi az általa generált adatforgalom.

A módszer azon alapul, hogy a TCP kapcsolatok megbízhatóak, azaz, ha egy csomag elvész, azt érzékeli (mivel nem kap a megérkeztéről visszajelzést, és az elveszett csomagot újraküldi). A technika ezt használja ki egy kapcsolat felépítése alatt. A kapcsolatot kezdeményező (szkennelő) gép küld egy SYN csomagot a célpontnak, az erre válaszol (mint mindig) egy SYN-ACK csomaggal, így tudatva a küldővel, hogy vette az üzenetet, és itt lépünk be mi... Elvileg a mi gépünknek erre kell válaszolnia egy ACK csomaggal, hogy vettük a célgép SYN-ACK-ját, de mi ezt nem küldjük el. Erre a célpont azt hiszi, hogy elveszett az üzenete, ezért újra elküldi, erre mi megint nem válaszolunk, ő megint elküldi és így tovább. Vannak oprendszerek, amik egy idő után küldenek egy RST (reset) csomagot, hogy tudassák a géppel, hogy a kapcsolat megszakadt. Na és amiről fel lehet ismerni az oprendszereket, az a SYN-ACK csomagok közt eltelt idő. Mivel ez minden oprendszernél állandó. Egyedül az zavarhatja meg egy kicsit, ha nagyon leterhelt a hálózat.

Egyszerű a mért eredményeket összevetni az ujjlenyomatokkal, és új ujjlenyomatokat is könnyű felvenni.

3.15.1. Hogyan is valósítsuk ezt meg

Erre nem térnék ki túl részletesen, mivel amit itt le fogok írni, az a *kézi* módszer, vannak már jól megírt programok is, amik ezt valósítják meg.

Először is kell találni egy nyitott kaput a célgépen, ami lehet pl. egy webserver. Ha ez megvan, akkor a saját gépünk tűzfalába fel kell venni egy olyan szabályt, hogy minden arról a gépről érkező csomagot el kell dobnunk (hogy ne tudjunk rá válaszolni egy ACK-al) aztán figyelni kell minden, a célgépről jövő csomagot (még a tűzfalunk előtt), és nézni bennük az ACK-SYN-eket.

Itt használhatjuk a pontosabb eredmény érdekében a TCP *Időbélyeg*et (timestamp), ami nem az aktuális idő, hanem egy, az oprendszertől függő változó (általában 1 ms - 1 s pontosságú).

Mérjük az eredményeket, és a későbbiekben majd látható táblázatokban megtalálható értékekkel összehasonlítva megállapítható az oprendszer, ami valószínüleg az a rendszer, amire a kapott értékek a legjobban hasonlítanak.

$$\sum |\lambda_i - \alpha_i| = \text{Eltérés}$$

A λ_i az általunk mért i-edik idő, az α_i a táblázat (ujjlenyomat) i-edik sora. Ahol ez az *Eltérés* a legkisebb, az a vizsgált gépen futó oprendszer. Most következzen egy-két példa a teljesség igénye nélkül:

Próba	Win 98	Win 2000	Linux 2.2	Linux 2.4	Minolta printer	Cisco Router
1.	3	3	3.5	4.26	4.5	2
2.	6	6	6.5	6	4.5	3.9
3.	12	NTK	12.5	12	9	5.9
4.	NTK		24.5	24	18	NTK
5.			48.5	48.2	36	
6.			96.5	NTK	72	
7.			120.5		144	
8.			NTK		285	
9.					576	
10.					169	
11.					169	
12.					169	
Reset	Nincs	Nincs	Nincs	Nincs	Van	Nincs

NTK = Nincs Több Kísérlet

3.15.2. Egy megvalósítás

A *Perl* `Net::RawIP` csomagjával mutatom be ennek a módszernek egy megvalósítását. A programnak szüksége van a saját gépünk IP-jére, egy szabad portunkra valamint a cél gép IP-jére és egy olyan portra, ahol várakozik valaki. Valamint a programot *root* jogokkal kell futtatni.

A programnak a következő paramétereket kell átadni: melyik interfészen érjük el a cél gépet: *i*, mi az IP címünk: *s*, mi a célpont IP címe: *d*, melyik portról küldjük a csomagokat: *p*[2] valamint egy nyitott portra a célponti gépen: *f*. Az paraméterek magyarázatát a program is kiírja, ha paraméterek nélkül indítjuk.

Most lássunk először egy példa futtatást majd a program forrását!

```
[root@hell ring]# ./ring.pl -i ppp0 -s 127.7.43.1 -p 10000 -d 127.5.64.1 -f 6346
1023603633.35892
4.19984701633453
6.00001001358032
12.2000069618225
23.9999990463257
48.1999959945679
^C
[root@hell ring]# /sbin/iptables -D INPUT 1[3]
[root@hell ring]#
```

A program kimenetét összevetve a táblázatunkkal, rögtön látszik, hogy ez bizony egy Linux-2.4-es gép!

Most következzen a forrás:

```
#!/usr/bin/perl

use Net::RawIP qw(:pcap);
require 'getopts.pl';

Getopts('i:s:p:d:f:');

$dev = $opt_i;
$daddr = $opt_d;
$dport = $opt_f;
$saddr = $opt_s;
$sport = $opt_p;

if (!$opt_i && !$opt_d && !$opt_p && !$opt_s && !$opt_f) {
  print "Usage: $0 -i interface -s myhost -p myport -d destaddress -f destport\n";
  exit;
}

$oldtime = 0;

$s = new Net::RawIP;
$filter = "ip proto \\tcp and src host $daddr and src port $dport";
$pcap = $s->pcapinit($dev,$filter,1500,60);
$offset = linkoffset($pcap);
if (fork){ loop $pcap,-1,\&check,\@s;}

system "/sbin/iptables -I INPUT -p tcp -s $daddr --source-port $dport -j DROP";
```

[2] Itt figyeljünk rá, hogy a tűzfalunk engedje be az erre a portra érkező csomagokat!

[3] A program betesz egy új szabályt az `INPUT` láncba, ezt manuálisan kell utána kiszedni.

```
$s->set({ip => {daddr => $daddr, saddr => $saddr},
         tcp => {dest => $dport, source => $sport, syn => 1}});
$s->send;

sub check {
  $t = timem();
  $time = $t - $oldtime;
  $oldtime = $t;
  $s->bset($_[2],$offset_tcp);
  my ($ack,$syn) = $s->get({tcp=>['ack','syn']});
  print "$time\n";
}
```

4. Tűzfal azonosítás

Az operációs rendszer azonosítását megnehezítheti, ha a célpont *tűzfal* mögött van. Ilyenkor meg kell találnunk a tűzfalat futtató gépet, majd fel kell térképezni azt – milyen szabályok vannak beállítva, milyen csomagokat enged át, és hova engedi őket –, majd fel kell térképezni a mögötte lévő hálózatot.

Természetesen ezt mind nem úgy akarjuk kivitelezni, hogy felhívjuk magunkra a figyelmet – szóval nem port szkenneljük a gépet, nem építünk fel teljes 3 lépcsős (SYN - SYNACK - ACK) kapcsolatot –, mivel ha jók a behatolás érzékelő rendszerük (IDS), akkor esetleg teljesen letiltják az IP-nket, és figyelmesebbek lesznek a biztonságra.

4.1. A tűzfal beazonosítása

Jó módszer arra, hogy megtaláljunk egy tűzfalat a `traceroute` program használata.

Ez egy olyan program, ami küld a kiszemelt gépnek egy UDP csomagot vagy egy ICMP ping-et úgy, hogy a TTL (time to live)-t 1-re állítja, majd a későbbiekben mindig eggyel növelni fogja. A gép, amelyet elér egy ilyen csomag, útközben az csökkenti eggyel a TTL-t, és küldi tovább. Ha a TTL 0-ra csökken, akkor eldobja, és visszaküld nekünk egy ICMP Time Exceeded csomagot, ezzel tudatva ezt velünk. Ezekből a csomagokból fel tudjuk építeni a célponthoz vezető utat.

Tételezzük fel, hogy van egy hálózat, amit egy tűzfal véd. Tegyük fel továbbá, hogy az 53-as (domain) portra érkező UDP csomagokon kívül nem enged át semmit. Próbáljunk meg traceroutolni egy gépet a tűzfal mögött, ezt kapjuk eredményül:

```
evil@hell:/# traceroute 1.0.0.10
traceroute to 1.0.0.10 (1.0.0.10) from 12.34.56.78 (12.34.56.78), 30 hops max
outgoing MTU = 1500
1  1.0.0.1 (1.0.0.1)  5 ms  2 ms  1 ms
2  1.0.0.1 (1.0.0.1)  3 ms  3 ms  3 ms
3  1.0.0.2 (1.0.0.2)  3 ms  3 ms  3 ms
4  1.0.0.5 (1.0.0.5)  4 ms  5 ms  4 ms
5  1.0.0.7 (1.0.0.7)  5 ms  5 ms  5 ms
6  * * *
7  * * *
```

Láthatjuk, hogy nem tudunk átjutni az 1.0.0.7-es gépen. Szóval feltehetőleg az a tűzfal.

A traceroute úgy működik, hogy az UDP csomagjaiban a cél port-ot mindig eggyel növeli minden egyes csomagnál. Namármost tudjuk, hogy a tűzfal átengedi az 53-as portra menő csomagokat. Állítsuk be a kezdeti portot a program -p paraméterével, a következő képlet szerint:

$$(\text{Cél port} - (\text{Tűzfal távolsága} * \text{Próbák száma})) - 1$$

és próbáljuk újra!

```
evil@hell:/# traceroute -p 34 1.0.0.10
traceroute to 1.0.0.10 (1.0.0.10) from 12.34.56.78 (12.34.56.78), 30 hops max
outgoing MTU = 1500
1  1.0.0.1 (1.0.0.1)  5 ms  2 ms  1 ms
2  1.0.0.1 (1.0.0.1)  3 ms  3 ms  3 ms
3  1.0.0.2 (1.0.0.2)  3 ms  3 ms  3 ms
4  1.0.0.5 (1.0.0.5)  4 ms  5 ms  4 ms
5  1.0.0.7 (1.0.0.7)  5 ms  5 ms  5 ms
6  1.0.0.9 (1.0.0.9)  5 ms  6 ms  5 ms
7  1.0.0.10 (1.0.0.10) 6 ms  6ms   6ms
```

Sikerült! Átjutottunk a tűzfalon. Ha esetleg a tűzfal mögött is egy olyan gép van, amely szűri a kapcsolatot, akkor már nem biztos, hogy a normális traceroute megfelel nekünk, mivel nagyon nagy kötöttség az, hogy mindig eggyel növeli a portszámot. Erre az esetre írhatunk egy egyszerű kis progit, aminek be lehet állítani a portot, amit használjon, vagy akár találhatunk a neten is megfelelőt.

4.2. Stateless tűzfalak

Ezek olyan tűzfalak, amik nem jegyzik meg a kapcsolatokról, hogy ki építette fel őket, vagy hogy egyáltalán fel van-e építve egy kapcsolat. Egyszerű szabályaik vannak, mint például minden a 20-as portról jövő csomagot engedjünk be (ezt használja az FTP az adatküldésre).

Itt jövünk a képbe mi. A csomagokat, amikkel feltérképezzük a tűzfal mögött lévő rendszert 20-as forrás porttal küldjük, amit egyszerűen megtehetünk. Tegyük fel továbbá, hogy a tűzfal mindent kiereszt. Ekkor a 3.1.-es szakaszban lévő programmal ha forrás portnak 20-ast állítunk be a cél gép készségesen válaszolni fog bármilyen csomagunkra.

4.3. ICMP csomagok kiszűrése

A legtöbb rendszergazda úgy állítja be a gépét, hogy az eldobja a pingeket (ICMP ECHO REQUEST). De megpróbálkozhatunk más ICMP csomagokkal is, úgymint az ICMP TIME STAMP REQUEST vagy ICMP INFO. A figyelmetlenebb adminisztrátorok ezeket nem tiltják le, így ezeket nyugodtan használhatjuk a felderítésekhez.

Az ICMP csomagok még nagyon sok lehetőséget kínálnak, de ezeknek a bemutatása túlnyúlik ezen irás keretein, viszont *Ofir Arkin* írt egy elég vaskos (218 oldalas) dokumentumot e témában, ami fellelhető a következő címen:

`www.sys-security.com/archive/papers/ICMP_Scanning_c3.0.zip`.

Ha már feltérképeztük a tűzfalat, akkor elkezdhetjük felderíteni a mögötte lévő területet, azokkal az eszközökkel, amiket a tűzfal megenged.

5. A hadicsel, avagy "Hogyan tévesszük meg a kutakodókat?"

> "Így a hadseregnek akkor adjuk a lehető legtökéletesebb formát, amikor már nincs is érzékelhető formája. Mert ha nincs érzékelhető formája, akkor a legtapasztaltabb kém sem tud kifürkészni semmit, és a legbölcsebb ember sem tud eligazodni rajtuk."
> – Szun-Ce, A hadviselés törvényei

Először is: csak akkor kezdjünk el az ujjlenyomat hamisítással foglalkozni, ha a gépünk biztonságában már tökéletesen meg vagyunk győződve, mert hiába látszunk egy printernek, ha bármelyik script, ami egy bizonyos tartományon minden gépet megtámad, rögtön talál biztonsági rést

rajtunk. Néha a hamis információk kibocsátása által a gépünk biztonsága gyengül, valamint a nagy forgalmat kiszolgáló gépeknél a hatékonyságra is negatívan hat a sok csomag meghamisítása.

Na jó, tegyük fel, hogy már teljesen biztonságos a gépünk, minden patch-et feltettünk, és egy elég jól működő tűzfalat állítottunk fel, most már hozzákezdhetünk a rejtőzködéshez.

Először is tervezzük meg, hogy minek akarunk látszani, milyen szolgáltatások illenek az álcánkhoz (elég hülyén nézne ki, ha egy printeren smtp daemon futna, vagy esetleg egy Cisco routeren az ircd). Szóval, miután összeszedtük, hogy hogyan akarunk kinézni, elkezdhetjük megvalósítani a tervünket.

Kezdjük a bannerek eltüntetésével vagy meghamisításával, ehhez általában újra kell majd fordítanunk a programjainkat. Itt figyeljünk arra, hogy egyes programok frissítései a program bannerében megjelenő verziószámokra támaszkodnak, így ebből még gondunk adódhat a későbbiekben. Itt mindenre gondoljuk, ne csak a telnetd üdvözlő szövegére, pl. árulkodó jel lehet, a kimenő leveleink fejléce is vagy a bevezetőben már említett IRC kliensek.

Hogyha már itt tartunk, el kell gondolkoznunk, hogy mit kezdjünk azokkal a szolgáltatásokkal, amiket futtatni akarunk, de mégsem szeretnénk, hogy látszódjanak. Pl. azt akarjuk, hogy a gépünk printernek tünjön, de mégis ssh-n keresztül akarjuk használni. Ilyenkor én a következőt tenném[4]: először is nem az alapértelmezett porton futtatnám, hanem valami jó magas porton (itt a választáshoz megnézném, hogy a fellelhető portscannerek melyik portokat nézik át alapértelmezés szerint). Tehát, ha megvan a port, ahol futtatom, a tűzfalat venném kezelésbe. Az adott portra csak azokról a gépekről engednék be forgalmat, ahonnan használni fogom, a többieknek azt mutatnám, hogy zárt. (Pl. az otthoni gépemen az ssh-t csak a pandora.inf.elte.hu gépről engedem használni, a többi gépről az erre a portra jövő csomagokat eldobom.)

Ezek után azt kell megnéznünk, hogy a hálózaton hogyan látszik a gépünk azzal szemben, amit meg akarunk valósítani. Nyugodtan támaszkodhatunk a pásztázóprogramok ujjlenyomat-állományaira. Itt nem csak a szokásos hibaüzeneteket kell figyelembe vennünk, hanem az egyedi flag-ek állapotát is. Az ujjlenyomat állományok nem tartalmaznak minden kis adatot, hanem csak a fontosabbakat.

Általában minden oprendszer igyekszik véletlenszerűvé tenni TCP ISN-jét. Ahhoz hogy a kiválasztott álcát magunkra öltsük, ezt is hozzá kell idomítani a kiválasztott rendszerhez, de ez egy nagyon rizikós dolog, mivel ezt a számot azért teszik az oprendszerek bonyolulttá, hogy meggátolják a TCP-eltérítést, de ha az álcánkban ezt le kell butítanunk, akkor ezzel a gépünket tesszük sebezhetővé. Szóval megint elérkeztünk oda, hogy el kell döntenünk, hogy a biztonság vagy az álca a fontosabb.

Ha még mindig álcázni akarjuk magunkat, akkor kell valami programot szereznünk (írnunk), ami módosítja a TCP csomagjainkat, *ráhúzza az álcát*. Erre én az *IP Personality*[5] programot javaslom, de akár saját magunk is írhatunk egy *Netfilter* modult a feladatra. Itt az a fő, hogy a program jól átgondolt, és biztonsági szempontból végletekig tesztelt legyen, mert ha valamit elrontunk, akkor sikerült egy új biztonsági rést gyártanunk.

Természetesen nem feltétlenül fontos teljesen azonosulni az álcával, mivel az is eléggé zavaró lehet a támadónak, ha a különböző módszerekkel az oprendszerünk másnak és másnak látszik. Lehet, hogy ez még jobb abból a szempontból, hogy a támadót így jól megkavarjuk.

6. Összefoglalás

Az eddigiekben bemutattuk, hogyan lehet, egy távoli számítógépről az interneten keresztül megállapítani, hogy milyen operációs rendszer fut rajta. Több eljárást is bemutattunk, példákkal illusztráltuk őket, valamint a tesztek elvégzéséhez szükséges programokat is leírtuk. Valamint egy pár szó erejéig kitértünk az itt leírt technikák elleni védekezésre is.

De ez még nem minden, sok más technika létezhet, és biztos, hogy létezik is. Valamint az operációs rendszer gyártók a jövőben bizonyosan újakat fognak tudtukon kívül lehetővé tenni,

[4] Lehet, hogy van ennél jobb módszer is, de nekem csak ez jutott eszembe.

[5] `http://ippersonality.sourceforge.net`

mivel egyesek nem nagyon tarták magukat a szabványokhoz, valamint vannak olyan dolgok amikre nincsen szabvány.

7. Rövidítések / magyarázatok

ACK Acknowledgement number – TCP fejlécben – a küldő ilyen sorszámú (SEQ) csomagot vár.

ACK / FIN / SYN-ACK / ... csomag Olyan TCP csomag amiben a a felsorolt flagek bevannak állítva

ACK flag Acknowledgement flag – Azt jelzi, hogy a csomag hordoz ACK sorszámot.

DOS támadás Szolgáltatás megtagadást előidéző támadás.

FIN flag Azt jelenti, hogy a küldőnél elfogytak a küldendő adatok.

ICMP Internet Control Message Protocol – Felhasználó számára értékes adatot nem de az oprendszer számára hasznos adatokat továbbító protokol.

IDS Intruder Detection System – Behatolásérzékelő rendszer.

IP Internet Protocol

IP ID IP Identification – Az IP csomagok egyedi azonosítószámja.

ISN Initial Sequence Number – Kezdeti Azonosító Szám – A kapcsolat első csomagjának a sorszáma (SEQ).

MSS Maximum Segment Size – Azt adja meg, hogy mekkora a legnagyobb TCP csomag amit az adott kapcsolaton át továbbítani lehet.

RST flag Reset flag – A kapcsolat bontásakor küldik el a másik félnek.

SEQ Sequence Number – A TCP csomagok egyedi azonosító száma.

SYN flag Synchronize flag – A TCP kapcsolat felépítésére vonatkozó kérelmet jelez.

TCP Transmission Control Protocol – Kapcsolatorientált, biztonságos adatküldést lehetővétevő protokol.

TOS Type Of Service – Az IP csomag típusát adja meg, pl.: TCP, UDP, ...

TTL Time To Live – A csomag maximum ennyi gépen mehet keresz"ul a feladótól a címzettig. Minden érintett gép csökkenti 1-el, ha eléri a 0-t eldobják.

UDP User Datagram Protocol – Csomagorientált, nem megbízható protokol.

Irodalomjegyzék

[1] Rob Beck - Passive-Aggressive Resistance: OS Fingerprint Evasion, Linux Journal Issue 89

[2] Fyodor - Remote OS detection via TCP/IP Stack FingerPrinting, www.insecure.org

[3] Franck Veysset, Oliver Courtay, Oliver Heen, Intranode Research Team - New Tool And Technique For Remote Operating System Fingerprinting, www.intranode.com/site/techno/ring-full-paper.pdf

[4] D. J. Bernstein - SYN cookies, ftp://koobera.math.uic.edu/syncookies.html

[5] Mount Ararat Blossom - Firewall Penestration Testing, www.wittys.com/files/mab/fwpentesting.html

[6] Transmission Control Protocol - RFC793 - http://www.rfc-editor.org/rfc/rfc793.txt

[7] Internet Protocol - RFC791 - http://www.rfc-editor.org/rfc/rfc791.txt