

Könnyű álm (8. rész)

Hálózati forgalom vizsgálata.

A mikor a rendszer nem úgy viselkedik, ahogy elváránk, vagy egyszerűen nem tudjuk, hogy mi történik a hálózaton, hasznos segédeszköz lehet a `tcpdump` program. A gépünket érintő vagy a hálózati területen átmenő forgalom lehallgatásával és megjelenítésével nélkülözhetetlen segédeszközzé válik a hálózati hibák felderítésében. Írásunkban a program használatának fortélyait ismertetjük, valamint a cikksorozatunk korábbi részében [1.] már bemutatott TCP/IP protokoll működését követjük nyomon.

Hogyan működik a `tcpdump`?

A `tcpdump` működéséhez elengedhetlenül szükséges, hogy a program a rendszer által vett vagy a hálózati csatoló által érzékelhető összes keretet megkapja. A keretek vételét a rendszermag biztosítja számunkra. Ethernethálózat esetén a kártyák általában csak a nekik szóló, valamint a csoportcímezett (`multicast`) és az üzenetszórt (`broadcast`) csomagokat veszik. A kártyák az összes keret vételére is



lehetőséget adnak, ezt hívjuk *lehallgató* (`promiscuous`) módnak. Fontos azonban megjegyezni, hogy a lehallgató mód egy terhelt hálózaton észrevehető további terhelést jelenthet a számítógépnek. A `tcpdump` több Unix-alapú rendszeren is működőképes, a rendszermag-támogatás viszont eltérő. A System V-alapú rendszereknél a keretek beolvasására a DLPI (Data Link Provider Interface) használatos, míg a BSD-alapú rendszerek a BPF-et (BSD Packet Filter) támogatják. A Linux egyik szabványának sem felel meg, e célra saját alrendszert vesz igénybe. Hordozhatóság céljából a programból a szorosan a rendszermaghoz kapcsolódó keretbeolvasó szolgáltatásokat leválasztották és a `libpcap` csomagban helyezték el. Mint az előzőekben említettük, a lehallgató mód komoly terhelést jelenthet a számítógépnek. A helyzetet súlyosbítja, hogy a vett kereteknek a rendszermagból a felhasználói programba is át kell jutniuk. Ha csupán jól meghatározott csomagokat szeretnénk látni, akkor a szűrést a felhasználói programban végezni erőforrás-pazarlás. Emiatt

A `tcpdump` szűrő alapelemei

Alapelem	Jelentés
<code>dst host <gép></code>	Igaz, ha az IP-csomag célja a megadott gép.
<code>src host <gép></code>	Igaz, ha az IP-csomag forrása a megadott gép.
<code>host <gép></code>	Igaz, ha az IP-csomag forrása vagy célja a megadott gép.
<code>ether dst <cím></code>	Igaz, ha a csomag ethernet-célcíme egyezik.
<code>ether src <cím></code>	Igaz, ha a csomag ethernet-forráscíme egyezik.
<code>ether host <cím></code>	Igaz, ha a csomag ethernet-forrása vagy célja a megadott cím.
<code>gateway <gép></code>	Igaz, ha az IP-csomag a gépet mint átjárót használja. Megegyezik az „ether host <cím> and not host <gép>” kifejezéssel, ahol is a cím a gép ethernet címe míg a gép az IP-címe.
<code>dst net <háló></code>	Igaz, ha az IP-csomag célja a megadott hálózaton helyezkedik el.
<code>src net <háló></code>	Igaz, ha az IP-csomag forrása a megadott hálózaton helyezkedik el.
<code>net <háló></code>	Igaz, ha az IP-csomag forrása vagy célja a megadott hálózaton helyezkedik el.
<code>net <háló> mask <maszk></code>	Igaz, ha az IP-csomag forrása vagy célja a megadott net <háló>/<hossz> hálózaton helyezkedik el. Itt a hálózat maszkját kifejtve adjuk meg.
<code>dst port <kapu></code>	Igaz, ha az IP-csomag célkapuja a megadott.
<code>src port <kapu></code>	Igaz, ha az IP-csomag forráskapuja a megadott.
<code>port <kapu></code>	Igaz, ha az IP-csomag forrás vagy célkapujával megegyezik.
<code>less <hossz></code>	Igaz, ha a csomag hossza kisebb vagy egyenlő a megadott értéknél.
<code>greater <hossz></code>	Igaz, ha a csomag hossza nagyobb vagy egyenlő a megadott értéknél.
<code>ip proto <proto></code>	Igaz, ha a keret egy proto-protokoll azonosítójú IP-csomagot hordoz.
<code>ether broadcast</code>	Igaz, ha a keret ethernet-címe üzenetszórt.
<code>ip broadcast</code>	Igaz, ha a csomag IP-címe valamilyen üzenetszórt cím.
<code>ether multicast</code>	Igaz, ha a keret ethernet-címe csoportcímezett.
<code>ip multicast</code>	Igaz, ha a csomag IP-címe valamilyen csoportcímezett cím.
<code>ether proto <proto></code>	Igaz, ha az ethernet-keret proto-protokolllal hordoz.
<code>ip, arp, rarp</code>	Igaz, ha az ethernet-keret protokollja a megadott. Rövidítések az „ether proto <proto>” formulára.
<code>tcp, udp, icmp</code>	Igaz, ha az IP-protokoll azonosítója a megadott. Rövidítések az „ip proto <proto>” formulára.

a) ICMP-példa

```
(1) 16:13:26.681407 P dolphin > cheetah: icmp: dolphin tcp port 1234
    unreachable [tos 0xc0] (ttl 255, id 25596)
(2) 16:13:29.191709 P cheetah > dolphin: icmp: echo request (ttl 64, id 115)
(3) 16:13:29.191709 P dolphin > cheetah: icmp: echo reply (ttl 255, id 25595)
```

b) ARP-példa

```
(1) 15:53:50.474273 B 0:c0:c:3:9:4d Broadcast arp 60: arp who-has dolphin tell cheetah
(2) 15:53:50.474273 P 0:4f:4c:3:35:ea 0:c0:c:3:9:4d arp 60: arp reply dolphin is-at 0:4f:4c:3:35:ea
    (0:c0:c:3:9:4d)
```

c) UDP-példa

```
( 1) 16:10:31.076878 B 0.0.0.0.bootpc > 255.255.255.255.bootps: xid:0xa1bcf3fc vend-rfc1048
    DHCP:DISCOVER
( 2) 16:10:31.076878 < dolphin.1313 > hawk.domain: 23418+ A? cheetah.home. (30)
( 3) 16:10:31.076878 > hawk.domain > dolphin.1313: 23418* 1/1/1 A cheetah (81)
( 4) 16:10:31.086877 < dolphin.1313 > hawk.domain: 23419+ A? hawk.home. (27)
( 5) 16:10:31.086877 > hawk.domain > dolphin.1313: 23419* 1/1/1 A hawk (73)
( 6) 16:10:31.086877 P dolphin.bootps > cheetah.bootpc: xid:0xa1bcf3fc Y:cheetah S:dolphin ether
    0:c0:c:3:9:4d vend-rfc1048 DHCP:OFFER SID:dolphin LT:3232235520 RN:1616117760 RB:2828206080
    SM:255.255.255.0 DG:hawk NS:hawk HN:"cheetah" DN:"home" BR:10.1.2.255 [tos 0x10]
( 7) 16:10:31.136873 B 0.0.0.0.bootpc > 255.255.255.255.bootps: xid:0xa2bcf3fc vend-rfc1048
    MSZ:9218 PR:SM+DG+NS+DN+BR+HN+LOG+LPR+NTP+XFS+XDM HN:"cheetah.andrews^@" LT:3232235520
    DHCP:DISCOVER
( 8) 16:10:31.136873 P dolphin.bootps > cheetah.bootpc: xid:0xa2bcf3fc
    Y:cheetah S:dolphin ether 0:c0:c:3:9:4d vend-rfc1048 DHCP:OFFER SID:dolphin LT:3232235520
    SM:255.255.255.0 DG:hawk NS:hawk DN:"home^@" BR:10.1.2.255 HN:"cheetah" [tos 0x10]
( 9) 16:10:31.186868 B 0.0.0.0.bootpc > 255.255.255.255.bootps: xid:0xa2bcf3fc vend-rfc1048
    MSZ:9218 PR:SM+DG+NS+DN+BR+HN+LOG+LPR+NTP+XFS+XDM HN:"cheetah.andrews^@" LT:3232235520
    DHCP:REQUEST SID:dolphin RQ:cheetah
(10) 16:10:31.186868 P dolphin.bootps > cheetah.bootpc: xid:0xa2bcf3fc
    Y:cheetah S:dolphin ether 0:c0:c:3:9:4d vend-rfc1048 DHCP:ACK SID:dolphin LT:3232235520
    SM:255.255.255.0 DG:hawk NS:hawk DN:"home^@" BR:10.1.2.255 HN:"cheetah" [tos 0x10]
```

a BPF-megfelelő alrendszeret egy egyszerű szűrőrésszel is ellátták. A szűrő használata esetén a felhasználói programba már csak a kért csomagok jutnak el. A nem BPF-megfelelő rendszereknél a szűrés a pcap eljáráskönyvtáron belül valósul meg. Ahhoz, hogy a program használható legyen, a rendszermagot a `CONFIG_PACKET` lehetőséggel kell fordítani, és a támogatás modulba helyezésekor az `af_packet.o` állományba fog kerülni. Ha azonban a rendszernaplóban az alábbi üzenetet látjuk, akkor a libpcap számára szükséges támogatás hiányzik a rendszermagból:

```
modprobe: can't locate module net-pf-17 socket:
Address family not supported by protocol
```

A 2.2 rendszermag-sorozat óta a Linux is tartalmaz a BPF-nek megfelelő szűrőalrendszert, amit Linux Socket Filternek neveznek. Ezt a `CONFIG_FILTER` lehetőség kiválasztásával fordíthatjuk a rendszermagba.

A tcpdump használata

Először is nézzünk egy egyszerű példát a tcpdump használatára: kapcsolódjunk a `telnet` paranccsal egy számítógépre (a 2. *c listán* a *cheetah* gépről kapcsolódunk a *dolphin* SSH-kapujára), de előtte indítsuk el a tcpdump programot (ha a program a hálózatról vesz csomagokat, akkor elindításához rendszergazdai jogosultságok szük-

ségesek). A tcpdump minden sorba egy vett keretet ír (ez azonban a képernyő szélessége miatt törlik). A program kimenete függ a vizsgált protokolltól. Írásunkban TCP-, UDP- és ICMP-protokollokat vizsgálunk ethernethálózaton. A legelső oszlopban található az időbélyeg, utána a csomag forráscíme és -kapuja, majd célcíme és -kapuja szerepel. A címet a kaputól a pont (.) karakter választja el. A program a címetet és kaput megpróbálja feloldani. Siker esetén a cím vagy kapu helyett a neve fog szerepelni. Néhány esetben ez a feloldás lassú lehet vagy olyan forgalmat hozhat létre, ami a vizsgálatot zavarja. Ez esetben a tcpdump programnak a `-n` kapcsolót megadva a címfeloldás elmarad, a `-nn` hatására pedig kapufeloldás sem történik. Ebben az esetben mind a címek, mind a kapuk eredeti formában láthatók. Ha az adott keret egy IP-csomag darabja (erről már az IP-darabolás bemutatásakor az [1.] és [5.] részben bővebben szót ejtetünk), akkor a sor végén a *(frag azonosító:méret@kezdőpozíció)* jelölés található. Az azonosító az IP-fejléc azonosító mezője, a méret az adott darab mérete (bájtokban számítva az IP-fejléc hosszát), míg a kezdőpozíció a darab kezdete az eredeti IP-csomagban. Ezek után még a pluszjel (+) is előfordulhat, amely további darabokra utal. Ez nem más, mint az IP-csomag MF (More Fragments) bitje. Amennyiben a csomag nem darabolható, a sor végén egy (DF)-jelzés található. Ez a csomag fejlécében rejlő DF (Don't Fragment) bitre utal. Ha a programnak a `-v` vagy a `-vv` kapcsolót is megadjuk, további adatok birtokába jutunk. Ez esetben a sor végén mind az IP-fejléc azonosító

a) UDP-kapu zárva

```
(1) 18:27:26.019610 P cheetah.1024 > dolphin.1233: udp 6
(2) 18:27:26.019610 P dolphin > cheetah: icmp: dolphin udp port 1233
    unreachable [tos 0xc0]
```

b) UDP-kapu tiltva

```
(1) 18:27:49.667538 P cheetah.1024 > dolphin.1234: udp 6
(2) 18:27:49.667538 P dolphin > cheetah: icmp: dolphin udp port 1234
    unreachable [tos 0xc0]
```

c) TCP-példa

```
(1) 15:59:36.933715 P cheetah.1031 > dolphin.ssh: S 4011370143:4011370143(0)
    win 32120 <mss 1460,sackOK,timestamp 1205813 0,nop,wscale 0> (DF)
(2) 15:59:36.933715 P dolphin.ssh > cheetah.1031: S 3980097340:3980097340(0)
    ack 4011370144 win 32120 <mss 1460,sackOK,timestamp 1237732
    1205813,nop,wscale 0> (DF)
(3) 15:59:36.933715 P cheetah.1031 > dolphin.ssh: . 1:1(0) ack 1 win 32120
    <nop,nop,timestamp 1205813 1237732> (DF)
(4) 15:59:36.933715 P dolphin.ssh > cheetah.1031: P 1:26(25) ack 1 win 32120
    <nop,nop,timestamp 1237732 1205813> (DF)
(5) 15:59:36.933715 P cheetah.1031 > dolphin.ssh: . 1:1(0) ack 26 win 32120
    <nop,nop,timestamp 1205813 1237732> (DF)
(6) 15:59:47.122817 P cheetah.1031 > dolphin.ssh: F 1:1(0) ack 26 win 32120
    <nop,nop,timestamp 1206832 1237732> (DF)
(7) 15:59:47.122817 P dolphin.ssh > cheetah.1031: . 26:26(0) ack 2 win 32120
    <nop,nop,timestamp 1238752 1206832> (DF)
(8) 15:59:47.122817 P dolphin.ssh > cheetah.1031: F 26:26(0) ack 2 win 32120
    <nop,nop,timestamp 1238752 1206832> (DF)
(9) 15:59:47.122817 P cheetah.1031 > dolphin.ssh: . 2:2(0) ack 27 win 32120
    <nop,nop,timestamp 1206832 1238752> (DF)
```

mezőjét, mind a csomag élettartam-számlálóját kiíratjuk. A tcpdump az adatkapcsolati réteg adatait alapesetben nem mutatja. Ha ezekre is kíváncsiak vagyunk, használjuk a `-e` kapcsolót. Ethernethálózat esetén a keret ethernetszintű forrás- és célcíme, a hordozott protokoll, valamint a keret hossza is megjelenítésre kerülnek. Alapesetben a tcpdump csak a gép által vett vagy küldött forgalmat mutatja. Amikor a `-p` kapcsolót megadjuk, akkor a kártyát nem kapcsolja lehallgató módba. A `-p` kapcsoló értelmezése sajnos nem teljesen egyértelmű. Az eredeti forrásban és a Debian-változatban a fenti értelmezés az elfogadott, de a RedHat-csomagban a `-p` kapcsoló értelmezése ennek épp az ellenkezője. Érdemes a kapcsolók értelmezésének rendszerünk tcpdump súgóoldalán utánanézni. Abban az esetben, ha a gép több hálózati csatlót is tartalmaz, a csomagok beolvasását kényelmesebb csak az egyikre szűkíteni. Amennyiben a lehallgató módot választjuk, ezt kötelező megtenni. Ezt a `-i` kapcsolóval ejthetjük meg, a kapcsoló után pedig a csatlókártya nevét kell beírni. A terhelés csökkentése céljából a tcpdump nem olvassa fel a teljes keretet, pusztán csak az első 68 bajtot. Ha a teljes keret érdekel (például a csomag adatairól vagyunk kíváncsiak), a `-s` kapcsolóval megmondhatjuk, milyen hosszan tárolja. A kapcsoló után a legnagyobb beolvasandó hossz jelöli – célszerű a csatlókártya MTU-értékét megadni, ez ethernet esetén általában 1500 bajt. Ha a csomag tartalmát is meg szeretnénk jeleníteni, akkor a `-a` kapcsolót (ASCII) vagy a `-x` kapcsolót (hexadecimális) is megadhatjuk. Sok esetben hasznos lehet, ha a beolvasott adatokat a későbbiekben is

meg tudjuk vizsgálni, ezért a tcpdump lehetőséget nyújt arra, hogy a beolvasott kereteket állományba menthessük. Természetesen az ilyen állományok beolvasására is képes. Ez sokkal hasznosabb lehet, mint ha a program kimenetét mentenénk el, hiszen ilyenkor lehetőség nyílik a későbbi szűrésekre vagy más programokkal történő elemzésre.

Szűrés

Hasznos segédeszköz a tcpdumpba épített szűrési lehetőség, amivel a hálózatról felolvasott keretek körét befolyásolhatjuk. A szűrő logikai kifejezés, csak azokat a kereteket olvassa fel, amelyekben a kifejezés értéke igaz. Ha nem adunk meg szűrőkifejezést, az összes keretet felolvassa. A kifejezés alapelemek (primitíves) összekapcsolásából áll. A lehetséges alapelemeket az 56. oldalon található táblázat mutatja. A szűrőfeltételben – a protokoll nevét tömbként használva – a csomag tartalma is vizsgálható. A formai követelmény az alábbi: *protokoll [eltolás : méret]* ahol a protokoll az ether, fddi, ip, arp, rarp, tcp, udp vagy icmp

valamelyike. Az eltolás a protokollréteg kezdetétől számított táv (offset), a méret az adat hossza. Mindkét értéket bajtokban adjuk meg. A méret lehetséges értékei 1, 2 vagy 4, az alapértelmezett az 1. A `len` a tcpdump beépített változója, amely a keret hosszát tartalmazza. Ezek között az adatok között a C nyelvben megszokott aritmetikai és relációs műveletjeleket is használhatjuk. A lehetséges aritmetikai műveletjelek: `+`, `-`, `*`, `/`, `&`, `|`, míg a relációs műveletjelek: `>`, `<`, `>=`, `<=`, `=`, `!=`. Az adatvizsgálat relációs műveletjelek segítségével kapcsolható a logikai kifejezésbe. A logikai kifejezés tagjai a már megszokott módon kapcsolhatók össze. Lehetőség nyílik *ellentétképzésre* (negálásra: `not` vagy `!`), az *és* (`and` vagy `&&`), valamint a *vagy* (`or` vagy `||`) műveletre is. Az ellentétképzés sorrendisége a legmagasabb, az *és*, valamint a *vagy* műveletek sorrendisége megegyezik, és balról jobbra értékelődnek ki. A műveletek sorrendje zárójelek segítségével módosítható.

Az ICMP-csomagok

Az ICMP-csomagokat a forrás- és célcíme után következő `icmp:` rész jelöli. Ezután következik maga az üzenet, például a `port unreachable` (*dest unreachable/port unreachable* – célkapu nem érhető el) vagy a `ping` parancs által küldött `echo request`, valamint a válaszul kapott `echo reply` üzenetek. Egy `port unreachable` üzenet látható az 1. a lista (1.) sorában és a `ping` parancs eredménye a (2–3.) sorokban.

a) TCP-kapu zárva

```
(1) 18:04:16.998792 P cheetah.1036 > dolphin.1233: S 3321364585:3321364585(0)
    win 32120 <mss 1460,sackOK,timestamp 1953835 0,nop,wscale 0> (DF)
(2) 18:04:16.998792 P dolphin.1233 > cheetah.1036: R 0:0(0) ack 3321364586 win 0
```

b) TCP-kapu tiltva

```
(1) 18:04:33.597399 P cheetah.1037 > dolphin.1234: S 3328752345:3328752345(0)
    win 32120 <mss 1460,sackOK,timestamp 1955496 0,nop,wscale 0> (DF)
(2) 18:04:33.607398 P dolphin > cheetah: icmp: dolphin tcp port 1234 unreachable [tos 0xc0]
```

c) tcpdump-példák

```
(1) tcpdump -i eth0 'host cheetah'
(2) tcpdump -i eth0 'src host cheetah'
(3) tcpdump -i eth0 'host cheetah and tcp and port ssh'
(4) tcpdump -i eth0 'not ( host cheetah and tcp and port ssh )'
(5) tcpdump -i eth0 'not ( tcp and ( src host cheetah and dst port ssh ) or
    ( dst host cheetah and src port ssh ) )'
(6) tcpdump -i eth0 'tcp[13] & 7 != 0'
(7) tcpdump -i eth0 'tcp[13] & 5 != 0 or tcp[13] & 18 = 2'
```

Az ARP-csomagok

Az ARP (Address Resolution Protocol) feladatait cikksorozatunk 5. részében [2.] már bemutattuk. Most nézzük meg, hogyan is működik ez a valóságban. Ahogyan az *1. b listán* is látható, a gép ARP-kérést küld ki a hálózatra (1.). A címzett (vagy az erre feljogosított egyéb) gép válaszol (2.), az ARP-kérés üzenetszört módon kerül továbbításra (miként a példában is látható), míg a válasz közvetlen címzéssel érkezik. Ahhoz, hogy az ethernetcímekeket is láthassuk, a `-e` kapcsolót is megadtuk a programnak.

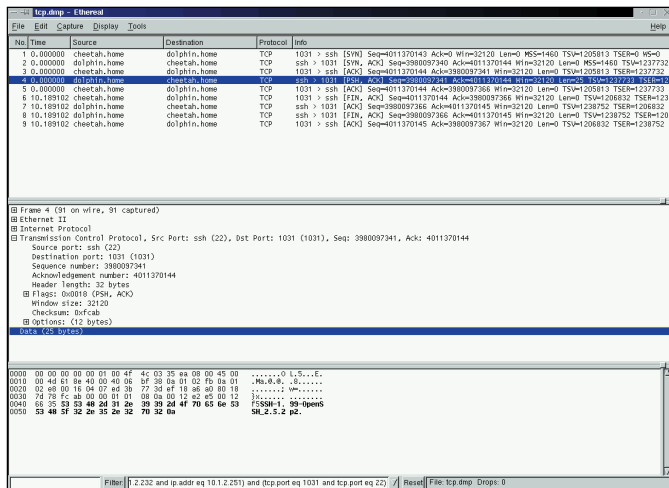
Az UDP-csomagok

Az UDP szemléltetéséhez egy gép indulását vizsgáltuk az *1. c listában*: itt a BOOTP- illetve a DHCP-, valamint a DOMAIN-protokollok használatára is láthatunk példát. A `tcpdump` mindkét protokollt ismeri, így ezek értelmezésére is képes. A *cheetah* induláskor üzenetszört csomagot küld ki, amellyel DHCP-kiszolgálót keres 1. sor. A kiszolgáló-bejegyzésben címek helyett nevek szerepelnek, így a DHCP-kiszolgálónak ezeket először fel kell oldania. A két tartománykérés a 2. és 4. sorban, a DNS válasza pedig a 3. és 5. sorokban látható. A két névfeloldás egymás után következik be. A DHCP a választ a csomagban (6. sor) küldi el az ügyfélnek. Az ügyfél újabb adatokat kér (7. sor), amelyeket a kiszolgáló vissza is küld (8.). A beérkezett adatok alapján a gép DHCP-kiszolgálót választ (9.), amelyet a kiszolgáló nyugtáz (10.). Mi történik azonban, ha a kiszolgáló adott kapuja zárva marad, miközben megkísérelünk csomagot küldeni neki? Ezt szemlélteti a *2. a lista*. A csomagot a *cheetah* (1.) küldi a *dolphin* gép 1233-as kapujára. Az adott kaput azonban zárva találja, így a *dolphin ICMP port unreachable* üzenettel válaszol (2.). Ugyanígy ICMP-üzenetet bocsát ki a kiszolgáló akkor is, ha az adott kaput csomagszűrővel védjük. Az üzenet típusa *destination unreachable*, a kódja viszont változhat. A Linux 2.2.x magorozat ilyenkor *port unreachable* kóddal válaszol, mintha a kapu zárva lenne. Ezt mutatja meg a *2. b lista*.

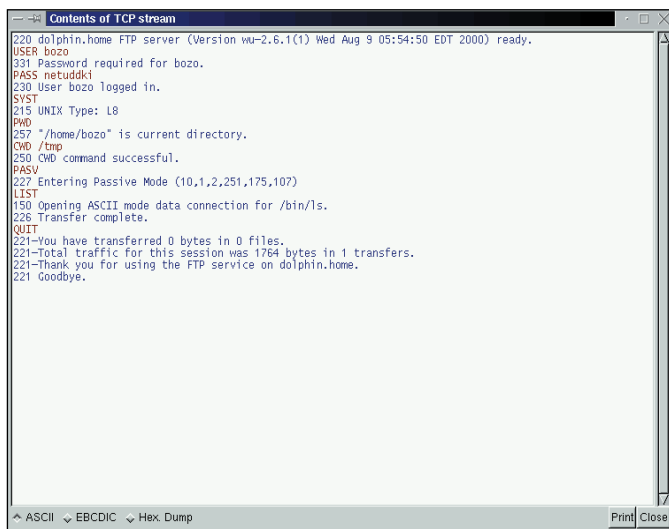
A TCP-csomagok

A *2. c listán* a TCP-kapcsolat felépítésére, az adattovábbításra és a lebontásra láthatunk példát. Mint már a protokollok ismertetésénél [1.] említettük, a kapcsolat felépítése három lépésből áll. A példán a *cheetah* gép kapcsolódik a *dolphin* SSH-kapujára. A kezdeményező gép egy

SYN-es csomagot küld (1.), amelyben az általa választott ISN (Initial Sequence Number – kezdő sorszám) rejtőzik. A SYN bit meglétét a címek utáni S betű jelzi. Ha a címzett gép adott kapuja fogadja a kérést, akkor SYN+ACK csomagot küld vissza (2.). A SYN bit itt szintén látható, de később az `ack` kulcsszó is szerepel, mögötte a *cheetah* által választott kezdősorszámnál eggyel nagyobb értékkel. Figyeljük meg, hogy a távoli gép maga is előállít egy kezdősorszámot. A *cheetah* gép a csomag vétele után egy ACK-csomagot küld (3.), ezzel a kapcsolat felépült – a csomagon nincsenek jelzőbitek, ezért itt most pont (.) szerepel. A két szám a kezdő és a végső sorszám, mögötte zárójelben a csomagban levő adatbájtok száma található. Látható, hogy a csomag csak nyugta, adatot nem hordoz. A sorszámok viszont furcsa módon megváltoztak. Ezt a változtatást a `tcpdump` követte el, így az eredmény könnyebben olvasható lett. Ha a csomag nem tartalmaz SYN bitet, a kezdősorszámot mind a pillanatnyi, mind a nyugta sorszámából kivonja. A `-s` kapcsoló megadásával az eredeti abszolút sorszámokat látjuk, amelyeket a csomagok is tartalmaznak, kapcsoló nélkül pedig a relatív TCP-sorszámokat fogja megjeleníteni. Aki kíváncsi rá, próbálja ki – pillanatokon belül azt is észreveszi, hogy miért tesz így a program. A (4.) csomagban a *dolphin* elküldi az SSH-protokoll azonosító üzenetét az ügyfélgépnek. Láthatjuk, hogy ez a 1–26. relatív sorszámokat foglalja el, így az üzenet 25 bájttal hosszú. A csomagon a P (PUSH) jelzőbit szerepel, amely az ügyfél TCP-, illetve IP-alrendszerét utasítja, hogy a vett adatokat továbbítsa az alkalmazásnak. Az (5.) csomagban a *cheetah* nyugtázza mind a 26 bájtot, mely az `ack` után látható. Az ügyfél nem küld adatokat. A kapcsolat lebontását a (6–9.) csomagok végzik. Mint már említettük, mindkét adatirányt függetlenül kell lebontani. A bontást a *cheetah* kezdeményezi (6.), a csomag egy F (FIN) jelzőbitet tartalmaz; a *dolphin* nyugtázza a bontási kérést (7.). A FIN bit egy sorszámot foglal el, ettől lesz a nyugtasorszám 2. A FIN vétele után a *dolphin* is kezdeményezi a kiszolgáló→ügyfél adatirány-bontását (8.), a *cheetah* ezt nyugtázza (9.), majd sikeresen lebontja a kapcsolatot. Mi történik, ha a kiszolgáló adott kapuja zárva marad, miközben megkísérelünk rákapcsolódni? Ezt szemlélteti a *3. a lista*. Az 1. csomagban a *cheetah* kapcsolatfelépítési kérést küld a *dolphin* gép 1233-as kapujára. Az adott kaput viszont zárva találja, így a *dolphin* egy RST-szakaszt küld válaszul (2.) – mint láthatjuk, az RST jelzőbit meglétét az R betű jelzi. Ha azonban a kaput csomagszűrővel védjük, a



1. kép Az ethereal fő képernyője



2. kép Az ethereal TCP kapcsolatkövető képernyője

visztautasítás módja megváltozik. Ilyenkor a rendszermag egy ICMP-üzenetet küld vissza. Az üzenet típusa *destination unreachable*, a kódja viszont változhat. A Linux 2.2.x magorozat ilyenkor *port unreachable* kóddal küldi vissza (ezt szemlélteti a 3. b lista).

tcpdump-példák

Az 3. c lista hasznos példákat mutat a tcpdump használatára. Az 1. példában látható, hogy az eth0 hálózati kártyán a cheetah gépről induló vagy oda érkező IP- és ARP-csomagokra figyelünk. A 2. példában csak a cheetah gépről kiinduló IP-csomagokat nézzük. A 3. példa már összetettebb szűrőt használ: csak azon TCP-csomagokat mutatja, amelyekben a cheetah gép, valamint az ssh port (22/tcp) érintett (a cheetah SSH-kapujára érkező és induló vagy a cheetah gép és tetszőleges más gép 22-es TCP-kapuja közötti csomagok). Sokszor előfordul, hogy SSH-protokollon jelentkeznünk be egy gépre, és úgy futtatjuk a tcpdumpot. Ilyenkor hasznos lehet a 4. példa. Itt a cheetah gépet és az SSH-kaput érintő forgalom figyelmen kívül marad (ez a feltétel ugyan más csomagokat is eldob, de a célnak sokszor így is megfelel). A kifogástalan feltétel az 5. példában látszik. Ilyen bonyolultabb feltételek esetén lehet hasznos a -F kapcsoló, aminek hatására a szűrőkifejezést a megadott állományból olvassa; ilyenkor a parancsorbán megadott szűrőfeltételt figyelmen kívül hagyja. Ha csak a SYN, a FIN és a RST jelzőbitekkel rendelkező csomagok érdekelnek minket, akkor a 6. példa mutatja a megoldást. Mint cikk-sorozatunk korábbi részeiben említettük [1., 5., 6.], ezek a jelzőbitek

a TCP-fejléc 14. bájtyán helyezkednek el. Ha pedig azt szeretnénk, hogy a SYN+ACK bitkombinációval rendelkező csomagok ne látszanak, akkor a (7.) példa szerinti szűrőfeltételt kell használni.

ethereal

Az ethereal hálózati elemzőcsomag Unixokra. Könnyen kezelhető, számos kényelmi lehetőséggel: képes a csomagok közvetlen beolvasására, de akár a tcpdump segítségével létrehozott állományok is beolvashatók vele. A csomag két programot tartalmaz: a tethereal karakteres felülettel rendelkezik, míg az ethereal GTK+ toolkit segítségével grafikus felületen fut. Az ethereal-csomag szinte minden Linux-terjesztéshez elérhető, de forrásban is letölthető [4.]. Az ethereal-csomag több keretformátumot és protokollt ismer, mint a tcpdump. Roppant hasznos segédeszköz, ha valaki hálózaton kapcsolatot tartó programot készít vagy valamely protokoll működésére kíváncsi. Az ethereal főképernyőjét mutatja a 1. kép. Itt jól látható, hogy miként jeleníthető meg a beolvasott csomagok összes jellemzője. A program nagyon hasznos további lehetősége, hogy a TCP-kapcsolatok végigkövethetők. Ebben az esetben hexadecimálisan vagy szöveges formában láthatjuk a kapcsolatban átáramló adatokat. A két gép által küldött karaktereket eltérő színnel jeleníti meg. A 2. képen FTP-példa látható: a kiszolgáló által küldött üzenetek (kék színnel) jól elkülönülnek az ügyfél által küldött üzenetektől (piros). Az ethereal használatát rendkívül könnyű elsajátítani. Szűrőfeltétel felépítésére itt is lehetőség nyílik, bár a szűrő formai követelményei eltérnek a tcpdumpnál megismertektől. A pontos formátumot az eszköz kézikönyvében jól leírták.

Biztonsági kockázatok

Talán kissé furcsán hangzik, de a hálózati elemzőknek is komoly biztonsági kockázatai akadnak. Ennek oka, hogy a program magas jogosultsággal fut és bemenete a támadó által befolyásolható. Egy rendszergazdai jogosultsággal futó tcpdump-programra ugyanúgy kell tekinteni, mintha hálózati démon lenne. Bármilyen biztonsági hiba (amire már volt is példa a tcpdump esetén is) a figyelő állomáson kód-végrehajtást tehet lehetővé.

Irodalomjegyzék

- [1.] Könnyű álmok: Az Interneten használt fontosabb protokollok – Linuxvilág, 2001. január
- [2.] Könnyű álmok: Behálózva – Linuxvilág, 2001. április
- [3.] A tcpdump honlapja ➔ <http://www.tcpdump.org>
- [4.] Az ethereal honlapja ➔ <http://www.ethereal.com>
- [5.] W. Richard Stevens TCP/IP Illustrated, Volume 1; ISBN 0201633469
- [6.] RFC793: Transmission Control Protocol



Mátó Péter (atya@andrews.hu), informatikus mérnök és tanár. Biztonsági rendszerek ellenőrzésével és telepítésével, valamint oktatással foglalkozik. 1995-ben találkozott először linuxos rendszerrel. Ha teheti, kirándul vagy olvas.



Borbély Zoltán (bozo@andrews.hu), okleveles mérnök-informatikus. Főként Linuxon futó számítógépes biztonsági rendszerek tervezésével és fejlesztésével foglalkozik. A 1.0.9-es rendszermag ideje óta linuxozik. Szabadidejét barátaival tölti.