Alkalmazásszintű proxyzás a Zorp segítségével (2. rész)

A Zorp proxykiszolgáló a rendszermag Netfilterével együttműködve alkalmazásszintű, az ügyfelek számára átlátszó proxyzást végez.

Előző írásomban ódákat zengtem az alkalmazásszintű proxytűzfalakról, valamint ***Scheidler Balázs*** kereskedelmi és ingyenes változatban egyaránt elérhető Zorp tűzfalcsomagját mutattam be. Ez alkalommal ott folytatom, ahol abbahagytam, vagyis áttekintem a Zorpnak azokat az alapszintű beállításait, amelyekre egy belső hálózat – demilitarizált zóna (DMZ) – külső hálózat környezetben szükség lehet. Csak néhány szolgáltatás beállításáról lesz szó, de bízom abban, hogy ezek alapján a leendő Zorp-felhasználók meg tudják kezdeni saját intelligens tűzfalrendszerük kiépítését. Idézzük fel néhány szóban az alkalmazásszintű proxyk feladatát. A proxyk nem pusztán továbbadják, sokkal inkább közvetítik a rajtuk keresztül haladó forgalmat. Például, ha a belső hálózat valamelyik felhasználója HTTP-kapcsolatot kezdeményez egy a proxytűzfal túloldalán lévő kiszolgálóval, a tűzfal elfogja és megtöri a kapcsolatot; így kiszolgáló (az ügyfél felől nézve) és ügyfél (a célkiszolgáló oldaláról szemlélve) szerepet egyaránt játszik.

A Zorp átlátszó proxykat használ, a Zorp tűzfal mögött dolgozó felhasználóknak tehát nem kell tudniuk róla, hogy tűzfal védi őket – olyan módon érhetik el a külső címeket és állomásneveket, hogy saját alkalmazásaikban nem kell a proxyval való kapcsolattartáshoz szükséges beállításokat megadniuk. Ezt fontos kiemelni, mert lényeges ellenérvként szolgál, ha azzal az idejétmúlt nézettel találkozunk, hogy a proxyk szükségszerűen sokkal bonyolultabbak, mint a másfajta tűzfalak. A Zorp használatakor a bonyolultság a háttérrendszernél jelentkezik, a felhasználók boldogságát így semmi sem árnyékolja be.

Ugyanakkor a Zorp használata a rendszergazda számára sem feltétlenül jelenti a kínok kínját. Ha engem kérdeznének, azt mondanám, a beállítása bonyolultabb, mint az `iptables` beüzemelése, de könnyebb, mint a ***sendmail.cf*** megírása. Ne is várjunk tovább, húzzuk fel saját Zorp tűzfalunkat!

Előfeltételek

A továbbiakban feltételezem, hogy előző írásom alapján sikeresen foltozott 2.4-es rendszermaggal rendelkezel, `iptables`-példányod pedig támogatja a `Tproxy` modult (lásd ➲ http://www.balabit.hu/products/oss/tproxy). Feltesszem azt is, hogy lefordítottad és telepítetted a `libzorpll`, a `zorp` és a `zorp-modules` csomagokat – forráskód vagy deb-csomag formájában ezek a ➲ http://www.balabit.com/products/zorp_gpl címről érhetők el. Példáim ugyan a Zorp GPL 2.0-s változata alapján készültek, de Zorp Pro 2.0-s alatt is érvényesek. A Zorp Pro rendelkezik néhány olyan proxymodullal, amelyeket a Zorp GPL nem tartalmaz, de a közös modulok azonosan működnek.

A környezet

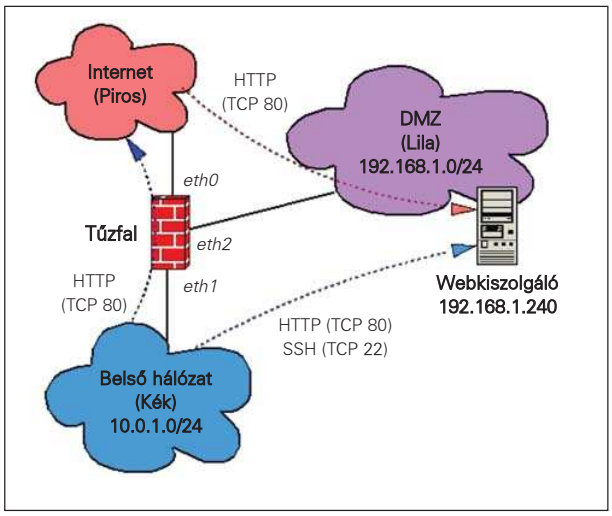
A Zorp tűzfalanként háromnál jóval több csatoló támogatására is képes, ám a leggyakrabban az *1. ábrán* láthatóhoz hasonló, háromlaki rendszerekkel találkozhatunk. A továbbiakban ilyen hálózat meglétét feltételezem.

Amint az 1. ábráról is kitűnik, összesen három adatáramlási irányt különböztetünk meg: HTTP-forgalom az internet és a DMZ-ben lévő webkiszolgáló között; HTTP-forgalom a belső hálózat és az internet között; HTTP és SSH-forgalom a belső hálózat és a DMZ között. Nem térek ki olyan, még az egyszerűbb hálózatokban is általánosan használt szolgáltatásokra, mint az IMAP, az NNTP vagy az FTP. Ha megérted, hogyan tudod beállítani a Zorpot a fenti alapszolgáltatások támogatására, akkor a többivel is boldogulni fogsz. A DNS és az SMTP viszont szóba fog kerülni, függetlenül attól, hogy az 1. ábráról lemaradtak.

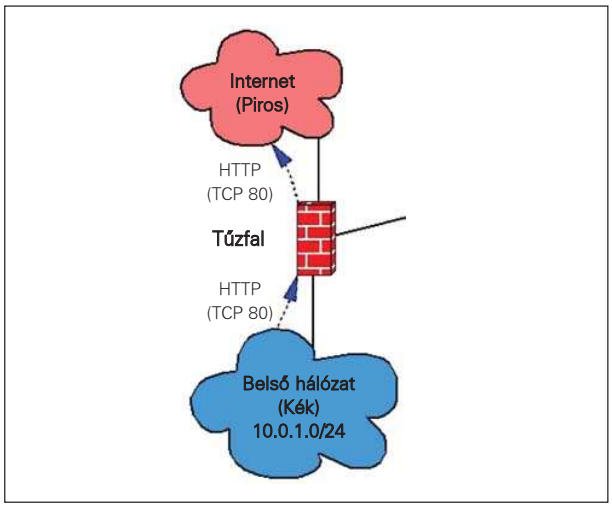
Dummy csatoló megadása

Az első dolog, amivel foglalkoznunk kell, nem annyira a Zorp, inkább a Tproxy rendszermagmodul működésével függ össze. Átlátszó proxyzásnál a Tproxynak szüksége van egy dummy (néma, ál-) hálózati csatolóra, amelyhez az adatfolyamok kettéválasztásakor kötődni tud. Ennek a csatolónak olyan IP-címet kell adni, amely az interneten nem irányítható, és a tűzfalhoz csatlakozó hálózatok egyikébe sem tartozik.

A 2.4-es rendszermagok alapértelmezett fordítás után is támogatják a dummy hálózati csatolók használatát. Valószínűleg neked is van ilyened, hacsak nem a dummy illesztőprogram támogatása nélkül fordítottad a rendszermagot. Ha így tettél, most új, dummy-támogatással rendelkező rendszermagra lesz szükséged. A Tproxy működéséhez mindössze annyit kell tenned, hogy megadsz egy `dummy0` csatolót, és nem irányítható, használaton kívüli címet rendelsz hozzá. Debian alatt a következő sorokat kell hozzáadnod a */etc/networking/interfaces* fájlhoz:



1. ábra Példahálózat



2. ábra HTTP-kapcsolat, amely a kék számára kimenő, a piros számára bejövő

```
auto dummy0
iface dummy0 inet static
       address 1.2.3.4
       netmask 255.255.255.255
```

Más terjesztések alatt ettől eltérhet a hálózati beállítások kezelése. A Red Hat- és a SuSE-terjesztések például ***ifcfg*** fájlokat helyeznek a ***/etc/sysconfig/network*** könyvtárba. Remélem, azért könnyen meg tudod fejteni a jelentésüket. Talán feltűnt, hogy az imént 32 bites alhálózati maszkot adtam meg. Miért? Ismétlem, a dummy csatoló címének egyik hálózatba sem kell tartoznia.

Beállítások: iptables

Jogos a kérdés, most a Zorp vagy az `iptables` a téma? A helyzet az, hogy a Zorp nem az `iptables` helyett, hanem vele együttműködve végzi feladatát. Maga a Tproxy is egy Netfilter-folt. A Tproxyt az `iptables` paranccsal tudjuk beállítani, ahogy a Netfilter egyéb részeit is. (A Netfilter a Linux 2.4-es tűzfalkódjának neve, az `iptables` pedig az a parancs, amellyel a rá vonatkozó beállításokat megadhatjuk.)

Emellett bizonyos szolgáltatásokat, különösen a DNS-t és az SMTP-t ajánlott öntartalmazó proxyként (self-contained proxies) futtatni a tűzfalon. Ha így teszel, akkor az `iptables` segítségével közvetlenül be kell állítanod a tűzfalat e kapcsolatok elfogadására. Például a BIND v9 támogatja a láthatár-megosztásos (split-horizon) DNS-szolgáltatást, amely a külső és a belső ügyfeleket más-más zónafájlokból szolgálja ki. Hasonlóan a Postfixet is könnyen be lehet állítani oly módon, hogy közvetítőként szolgáljon a belső állomások számára, de kizárólag helyi szállítást végezzen, ha külső állomásokkal kerül kapcsolatba. Ilyen proxyjellegű szolgáltatásokat sokszor érdemes a tűzfalon futtatni, feltéve, hogy beállításuk során rendkívüli gondossággal járunk el.

Ha nem vagy jártas a Netfilter/iptables kezelésében, akkor az alábbiakat nem nagyon fogod érteni, és sajnos helyhiány miatt nekem sincs módom arra, hogy részletesebb magyarázatokkal szolgáljak. Sajnálom, de a Zorp nem kezdőknek való eszköz. Dióhéjban csak annyit, hogy az `iptables` segítségével minden csomagot egyszerű ellenőrzésnek vetünk alá, amely során megpróbáljuk kiszűrni a hamisított IP-címeket. Ezután elfogjuk az átlátszó proxyzásra kiszemelt csomagokat, majd a normál `FORWARD` lánc helyett egyedi láncok segítségével dolgozzuk fel őket. Továbbítani lényegében semmit nem fogunk. Végül gondoskodunk néhány, magának a tűzfalnak küldött csomag célba juttatásáról.

A Zorp Próhoz egy ***iptables-utils*** nevű parancsfájlgyűjtemény is tartozik, ezek segítségével lényegesen leegyszerűsödik az `iptables` Zorp-vonatkozású kezelése. Az ***iptables-utils*** ingyenes kiadása a Zorp GPL 2.0-s változathoz a ➲ http://www.balabit.com/downloads/zorp/zorp-os/pool/i/iptables-utils címről tölthető le. Mindenkinek javaslom, hogy próbálkozzon meg az ***iptables-utils*** használatával, segítségével ugyanis sokkal könnyebb kipróbálni egy `iptables`-beállítást, mielőtt érvénybe léptetnénk.

Írásmódját helyhiány miatt nem ismertethetem, ezért az alábbi példa egy hagyományos `iptables` indítóparancsfájl lesz. Tekintsük át a parancsfájl legfontosabb részeit! Elöl találhatók a különleges `tproxy` tábla szabályai, ezeket a `Tproxy` modul adja hozzá a Netfilterhez (*1. kódrészlet*). Itt saját hálózataink mindegyikéhez egy-egy egyedi proxyláncot adunk meg: `PRkék` a belső hálózat felől indított, proxyzott kapcsolatok számára; `PRlila` a DMZ felől indított és proxyzott kapcsolatokhoz (ebben az esetben nincs ilyen); `PRpiros` az internetről indított, proxyzott kapcsolatok számára.

Az *1. kódrészletben* több érdekes dolgot is találunk. Először is a `tproxy` tábla saját `PREROUTING` és `OUTPUT` kimeneti láncokat tartalmaz. Zorp alatt a `tproxy/PREROUTING` lánccal továbbítjuk a csomagokat a megfelelő egyedi proxylánc felé (`PRkék`), annak alapján, hogy az egyes csomagok mely csatolón érkeznek be. Mint minden egyedi `iptables` láncnál, ha egy csomag úgy halad keresztül ezek valamelyikén, hogy egyik szabállyal sem mutat egyezést, akkor ahhoz a sorhoz kerül vissza, amely a csomagot az egyedi láncba küldő szabályt közvetlenül követi. Ez az oka annak, hogy az egyedi láncok nem tartalmaznak alapértelmezett célokat. A `PRkék` láncban két szabály található, mind a kettő a belső

1. kódrészlet Tproxy-szabályok

```
iptables -t tproxy -P PREROUTING ACCEPT
iptables -t tproxy -A PREROUTING -i eth1 -j
➥PRkék
iptables -t tproxy -A PREROUTING -i eth2 -j
➥PRlila
iptables -t tproxy -A PREROUTING -i eth0 -j
➥PRpiros

iptables -t tproxy -P OUTPUT ACCEPT

iptables -t tproxy -N PRkék
iptables -t tproxy -A PRkék -p tcp --dport 80 \
  -j TPROXY --on-port 50080
iptables -t tproxy -A PRkék -p tcp --dport 22 \
  ! -d tuzfal.pelda.net -j TPROXY --on-port 50022

iptables -t tproxy -N PRlila

iptables -t tproxy -N PRpiros
iptables -t tproxy -A PRpiros -p tcp --dport
➥80 \
  -j TPROXY --on-port 50080
```

hálózat felől kiindulni engedélyezett forgalomtípushoz egy-egy. Minden kimenő HTTP-forgalom proxyzásra kerül, vagyis átadásra egy az 50080-as kapun csücsülő proxyfolyamat felé. Az SSH-szabály annyiban más, hogy itt a Netfilternek csak addig kell proxyznia a kimenő SSH-forgalmat, amíg az nem maga a tűzfal felé irányul. Ugyan az 1. ábrán ezt a forgalomtípust nem tüntettem fel (Kék®SSH®Tűzfal), azonban a tűzfal felügyeletéhez szükség van rá. Ehhez az adatforgalomhoz a normál szűrőtábla `INPUT` láncában is szükséges egy szabály. A példahálózatban a DMZ-be helyezett webkiszolgáló nem kezdeményezhet semmilyen kapcsolatot, ezért a `PRlila` lánc üres maradt.
Lépjünk tovább a normál szűrőtáblára. Ez az a Netfilter tábla, amit valószínűleg sokan ismernek, az `iptables` ezt tekinti alapértelmezettként, ha a `-t` kapcsolót elhagyjuk. A *2. kódrészletben* a példatűzfal szűrőtáblájának `INPUT` szabályai láthatók.
Az első néhány sor segítségével – egyedi láncok felhasználásával – megpróbáljuk kiszűrni a hamisított IP-címeket. Ha egy csomag túljut ezen az ellenőrzésen, akkor lejjebb lép az `INPUT` láncban. A `Tproxy` modul által létrehozott csomagokat elfogadjuk, nemkülönben a meglévő és engedélyezett kapcsolatokhoz tartozókat és a hurokcsomagokat (loopback packets; 4–6. sor). Következőként, ahogy a `tproxy` tábla `PREROUTING` láncával is, a bejövő csatolójuk alapján a csomagokat egyedi láncokhoz irányítjuk. Ezeket az egyedi láncokat helyi célcímmel ellátott csomagokhoz készítettük, nem proxyozott csomagokhoz, ezért `LOkék` és hasonló nevekkel láttam el őket. Ezután lássuk a szűrőtábla egyedi láncait (*3. kódrészlet*).
Az egyedi láncok közül az első három a legfontosabb: `HEkék`, `HElila` és `HEpiros`, ezek adják meg a Netfilternek, hogyan kezelje a tűzfalnak küldött csomagokat, attól függően, hogy melyik csatolón keresztül érkeztek be. A `HEkék` lánccal elfogadjuk a DNS-lekérdezéseket, az SSH- és az SMTP-kapcsolatokat. A `HElila` csak a DNS-lekérdezéseket engedélyezi. Végül a `HEpiros` lánccal az internetszolgáltató DNS-kiszolgálójának (***upstream.dns.server***) válaszait és az SMTP-kapcsolatokat engedjük tovább. Az egyedi láncok közül az utolsó három a legegyszerűbb: a ***zaj*** a linuxos tűzfalak naplóit hagyományosan zagyvasággá változtató NETBIOS-csomagokat tartja távol; a ***hamis*** az egyértelműen hamisított, vagyis érvénytelen forrás-IP-címmel érkező csomagokat szűri ki; a ***hamis_eldob*** pedig a ***hamis*** által elfogott csomagokat naplózza és dobja el.

2. kódrészlet A szűrőtábla INPUT lánca

```
iptables -P INPUT DROP
iptables -A INPUT -j zaj
iptables -A INPUT -j hamis
iptables -A INPUT -m tproxy -j ACCEPT
iptables -A INPUT -m state \
  --state ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -i lo -j ACCEPT
iptables -A INPUT -i eth1 -j LOkék
iptables -A INPUT -i eth0 -j LOpiros
iptables -A INPUT -i eth2 -j LOlila
iptables -A INPUT -j LOG --log-prefix "INPUT
➥DROP: "
iptables -A INPUT -j DROP
```

A *4. kódrészlet* `iptables` példaparancsfájlunk maradékát tartalmazza, egy lényegében üres `FORWARD` láncot egy alapértelmezett `DROP` házirenddel, valamint egy üres `OUTPUT` láncot alapértelmezett `ACCEPT` lánccal. Ismétlem, ez egy proxytűzfal, így semmit sem fog továbbítani. A tűzfalról származó csomagokra vonatkozó alapértelmezett `ACCEPT` házirend miatt lehet, hogy gombóc marad a torkodban, de semmi gond, Zorp alapú tűzfal esetében az ilyesmi szükséges és biztonságos is.

Zorp-példányainak beállítása

Végre elérkeztünk a Zorp beállításait megadó fájlokhoz. Ezeket a */etc/zorp* könyvtárban találjuk; mindjárt vessük is rá magunkat az ***instances.conf*** fájlra, amely megadja a Zorp-példányokat, illetve szabályozza a működésüket. Ökölszabályként elfogadható, hogy hálózati zónánként egy példány szükséges, ezért példakörnyezetünkben, mint már nyilván kitaláltad, a piros, a kék és a lila zónához egyaránt egy-egy példány tartozik. Az *5. kódrészlet* szemlélteti, hogyan kell kinéznie egy ***instances.conf*** fájlnak.
Minden sorban az első mező a példány neve. A nevet szabadon választhatjuk meg, de ügyeljünk arra, hogy a Zorp ***policy.py*** beállításfájljában hibátlanul kell hivatkoznunk rá. Ha már itt tartunk, amennyiben úgy gondoljuk, az egyes példányokhoz külön beállítófájlokat is megadhatunk, de egyetlen fájlban felsorolhatjuk több zóna beállításait is. Teljesen mindegy, melyik megoldást választjuk, az `instances.conf -p` kapcsolója adja meg a Zorpnak, hogy melyik példányhoz melyik fájl társul.
A `-v` kapcsolóval a naplóüzenetek részletességét szabályoz-

3. kódrészlet Egyedi láncok a szűrőtáblában

```
iptables -N LOkék
iptables -A LOkék -p tcp --dport 22 --syn -j
➥ACCEPT
iptables -A LOkék -p udp --dport 53 -j ACCEPT
iptables -A LOkék -p tcp --dport 25 --syn -j
➥ACCEPT
iptables -A LOkék -j LOG --log-prefix "LOkék
➥DROP: "
iptables -A LOkék -j DROP

iptables -N LOlila
iptables -A LOlila -p udp --dport 53 -j ACCEPT
iptables -A LOlila -j LOG \
  --log-prefix "LOlila DROP: "
iptables -A LOlila -j DROP

iptables -N LOpiros
iptables -A LOpiros -p udp -s
➥upstream.dns.server \
  -sport 53 -j ACCEPT
iptables -A LOpiros -p tcp --dport 25 --syn -j
➥ACCEPT
iptables -A LOpiros -j LOG --log-prefix "LOpiros
➥DROP: "
iptables -A LOpiros -j DROP

iptables -N zaj
iptables -A zaj -p udp --dport 137:139 -j DROP
iptables -A zaj -j RETURN

iptables -N hamis
iptables -A hamis -i lo -j RETURN
iptables -A hamis! -i lo -s 127.0.0.0/8 -j
➥hamis_eldob
iptables -A hamis -i eth1 ! -s 10.0.1.0/24 \
  -j hamis_eldob
iptables -A hamis! -i eth1 -s 10.0.1.0/24 \
  -j hamis_eldob
iptables -A hamis -i eth2 ! -s 192.168.1.0/24 \
  -j hamis_eldob
iptables -A hamis! -i eth2 -s 192.168.1.0/24 \
  -j hamis_eldob
iptables -A hamis -j RETURN

iptables -N hamis_eldob
iptables -A hamis_eldob -j LOG \
  --log-prefix "Hamisított csomag: "
iptables -A hamis_eldob -j DROP
```

4. kódrészlet A szűrőtábla FORWARD és OUTPUT láncai

```
iptables -P FORWARD DROP
iptables -A FORWARD -j LOG \
  --log-prefix "FORWARD DROP: "
iptables -A FORWARD -j DROP

iptables -P OUTPUT ACCEPT
```

5. kódrészlet Az instances.conf

```
kék -v3 -p /etc/zorp/policy.py \
  --autobind-ip 1.2.3.4
lila -v3 -p /etc/zorp/policy.py \
  --autobind-ip 1.2.3.4
piros -v3 -p /etc/zorp/policy.py \
  --autobind-ip 1.2.3.4
```

6. kódrészlet A policy.py; 1. rész (átfogó érvényű beállítások)

```
from Zorp.Core import *
from Zorp.Plug import *
from Zorp.Http import *

InetZone("kékzóna", "10.0.1.0/24",
   outbound_services=["kék_http", "kék_ssh"],

InetZone("lilazóna", "192.168.1.0/24",
   inbound_services=["kék_http", "kék_ssh",
                     "piros_http"])

InetZone("piroszóna", "0.0.0.0/0",
   outbound_services=["piros_http"],
   inbound_services=["*"])

InetZone("helyizóna", "127.0.0.0/8",
   inbound_services=["*"])

# átfogó érvényű szakasz vége
```

hatjuk: a 3-as a közepes szint, az 5-ös pedig hibakeresésre használható. Ezzel a kapcsolóval csak a Zorp által létrehozott naplóüzenetek tartalmát szabályozhatjuk, a Netfilter/iptables naplózására nincs hatással. Végül minden sor egy `--autobind-ip` beállítással zárul, ez határozza meg, hogy a kapcsolatok proxyzásakor a Zorpnak melyik dummy IP-hez kell kötnie a Tproxyt. Ez az IP-cím az összes példányra nézve lehet azonos is, sőt lehetőleg ezt a megoldást válasszuk. Az itt megadott címnek értelemszerűen egyeznie kell a korábban beállítottal. (Lásd a „Dummy csatoló megadása” című részt.)

A Zorp alkalmazásproxyk beállítása: policy.py

Az ***iptables*** parancsfájl határozza meg, hogy a csomagok hogyan jutnak el a proxykhoz, a ***/etc/zorp/instances.conf*** pedig a Zorp indulását szabályozza. Nyilván a Zorp proxyjainak viselkedését is vezérelni kell valahogyan, erre a célra a ***/etc/zorp/policy.py*** fájl szolgál, illetve az a fájl vagy azok a fájlok, amelyet vagy amelyeket az ***instances.conf*** fájlban

7. kódrészlet A policy.py; 2. rész (példányok megadása)

```
def kék():
    Service("kék_http", HttpProxy,
            router=TransparentRouter())
    Service("kék_ssh", PlugProxy,
            router=TransparentRouter())
    Listener(SockAddrInet(´10.0.1.254´, 50080),
             "kék_http")
    Listener(SockAddrInet(´10.0.1.254´, 50022),
             "kék_ssh")

def lila():
    pass

def piros():
    Service("piros_http", HttpProxy,
router=DirectedRouter(SockAddrInet(´192.168.1.24
➥2´, 80),
forge_addr=TRUE))
    Listener(SockAddrInet(´169.254.1.254´,
➥50080),
             "piros_http")
```

megneveztünk. A ***policy.py*** a megszokott név, de szabadon eltérhetünk tőle. A házirendfájl két részből áll: az első részben az átfogó érvényű beállítások találhatók, itt a zónák megadása a hálózati címek és a megengedett szolgáltatások szerint történik. A második rész a szolgáltatás–példány-megadásokat tartalmazza, ebben az ***instances.conf*** fájlban felsorolt példányok megadása a szolgáltatások kiinduló helye alapján történik, továbbá itt írhatjuk elő a szolgáltatások alkalmazásproxykhoz való hozzárendelését is.

A *6. kódrészlet* teljes egészében tartalmazza ***policy.py*** példafájlunk átfogó érvényű szakaszát. Néhány `import` paranccsal kezdődik, ekkor végezzük el a szükséges Python-függvények beemelését. A következő rész a zónamegadásokat tartalmazza. Ha az ***instances.conf*** fájlban zónánként egy Zorp-példányt adunk meg, akkor az itt használt zónanevek lehetnek hasonlók a példánynevekhez, vagy akár meg is egyezhetnek velük. A 6. kódrészletben én eltérő neveket választottam, ezzel is szemléltetni próbáltam a zónanevek és a példánynevek egymástól való függetlenségét.

Minden zónamegadásban az 1. ábrán szereplő hálózatcímek valamelyikét és az engedélyezett szolgáltatások megadását találjuk. A szolgáltatásneveket szabadon választhatjuk meg, használatukra a következő részben, a szolgáltatás-példányok megadásakor kerül sor. Az utasításokkal kapcsolatban fontos kiemelni, hogy a bejövő és a kimenő irányokat mindig a zónához vagy a hálózathoz, és nem a tűzfalhoz viszonyítva kell érteni.

A *2. ábrán* követhetjük, hogy a belső hálózat–internet irányú HTTP-kapcsolatok proxyzott kapcsolatként viselkednek. Az ábra alapján egyértelmű, hogy a teljes kapcsolat létrejöttéhez egyrészt szükség lesz egy kimenő kapcsolatra a belső zónából (kék), valamint egy bejövő kapcsolatra az internetzónába (piros). Ennek megvalósulása rendre követhető a 6. kódrészlet kékzóna és a piroszóna szakaszában. Fontos, hogy mindkét olyan zóna megadásakor, amit valamilyen adatfolyam érint, ugyanazt a szolgáltatásnevet használjuk (a 2. ábra és a 6. kódrészlet esetében ez a ***kék_http***).

A 6. kódrészlet kapcsán még annyit mondanék el, hogy a * (csillag) helyettesítő karakter minden megadott szolgáltatást helyettesít. Ez csak első ránézésre jelent sok mindent, a * ugyanis csak azokat a szolgáltatásokat helyettesíti, amelyek a ***policy.py*** fájl szolgáltatás–példány-párosításaiban szerepelnek, és nem az összes lehetséges szolgáltatást. Ne feledjük, hogy a Zorp csak a Netfilter és a `Tproxy` által hozzá eljuttatott csomagokat dolgozza fel. Ha egy zónában sem a bejövő, sem a kimenő kapcsolatokat nem akarjuk engedélyezni, akkor az `inbound_services` vagy az `outbound_services` kulcsszót elhagyhatjuk, illetve üres szögletes zárójelet írhatunk mögé.

A *7. kódrészlet* a ***policy.py*** fájl szolgáltatás–példány-megadásait tartalmazza. Minden megadás első sorának egy az ***instances.conf*** fájlban szereplő példánynévre kell hivatkoznia, a további sorokat pedig be kell húzni, a feldolgozás ugyanis a behúzásokra nézve elég finnyás Pythonnal történik. A megadás nem lehet üres. Ha adott példánytól nem indul ki szolgáltatás, akkor a `pass` kulcsszót kell használni. Erre a 7. kódrészlet `lila()` példányánál láthatunk példát. Egyéb esetben a megadásnak egy vagy több `Service` sorból kell állnia, ezek egy vagy több zónamegadásban hivatkozott szolgáltatásnevet és egy Zorp proxymodult tartalmaznak. Utóbbi beépített, az átfogó érvényű befoglaló utasításokkal beemelt vagy egyedi osztályban megadott proxy egyaránt lehet. A `Service` sorok utolsó mezője a `router`, ez adja meg, hogy a proxyzott csomagokat hova kell küldeni.

A 7. kódrészletben például a `piros_http` szolgáltatásnál a `forge_addr=TRUE` paranccsal változatlan formában adjuk át az internetről érkező webes ügyfelek forrás-IP-címét a webkiszolgálónknak. Ha ezt a parancsot elhagynánk, akkor a DMZ-be befutó webes forgalom forrása látszólag a tűzfal lenne.

Ugyan a 7. kódrészletben csak a `HttpProxy` és a `PlugProxy` (általános célú UDP- és TCP-proxy, amely az alkalmazások adatait módosítás nélkül másolja) használatára látunk példát, a Zorp GPL FTP, `whois`, SSL, `telnet` és `finger` proxyval is rendelkezik. Miként már utaltam rá, saját osztályokat is készíthetünk, ha szeretnénk módosítani vagy bővíteni a meglévő proxykat. Könnyedén létrehozhatunk például URL-szűrést végző HTTP-proxyt, vagy a HTTPS-forgalom intelligens proxyzására képes, HTTP-proxyra ültetett SSL-proxyt. Sajnos ezek már magasabb szintekre tartozó témák, amelyekkel itt nem foglalkozhatunk. Szerencsére a Zorp Python proxymoduljai bőségesen el vannak látva megjegyzésekkel.

A 7. kódrészletben szereplő `TransparentRouter` egyszerűen proxyzza a csomagokat az ügyfél által megadott cél-IP-címre és -kapura. A piros példány `piros_http` szolgáltatásánál azonnal láthatunk példát a `DirectedRouter` használatára is, ennél kötelező érvényű cél-IP-címet és -kaput is elő kell írni.

Egy szolgáltatás–példány-megadás minden `Service` sorához tartoznia kell egy `Listener` sornak. Ez a sor adja meg a Zorpnak, hogy melyik helyi (tűzfal) IP-címhez és kapuhoz kell a szolgáltatásnak kötődnie. Furcsának tűnhet, hogy a 7. kódrészlet `Listener` soraiban elég magas sorszámú kapuk szerepelnek, 80 helyett 50080 és 22 helyett 50022. Ne feledkezzünk meg azonban arról, hogy mindegyik proxy

a Netfilteren keresztül, a rendszermagtól kapja a csomagokat, nem pedig közvetlenül az ügyfelektől. Az itt megadott kapuszámoknak tehát egyezniük kell a `tproxy` tábla Netfilter-szabályaiban meghatározottakkal (1. kódrészlet). Már említettem, hogy míg a `HttpProxy` egy teljes mértékben alkalmazástudatos, a helyes HTTP-működés érdekében a vonatkozó RFC-ket mindenben követő proxy, illetve a `PlugProxy` egy általános célú proxy. A `PlugProxy` még így is jobb védelmet nyújt, mint a puszta csomagszűrés, ugyanis még az önmagában, alkalmazástudatos működés nélkül végzett proxyzás is többre képes, mint a Netfilter, hiszen utóbbival ellentétben meg tudja védeni rendszerünket az alacsony szintű támadásoktól.

Összegzés

Talán mondanom sem kell, rendkívül felszínesen tekintettem csak át a Zorp GPL szolgáltatásait. Messze ez a legbonyolultabb eszköz, mellyel ezeken az oldalakon valaha is foglalkoztam, de bízom benne, senki nem fogja elvesztegetettnek hinni azt az időt, amit a Zorp megismerésére fordított.

Linux Journal 2004. április, 120. szám

Mick Bauer (mick@visi.com)
Biztonsági szakember, a Linux Journal biztonsági témákkal foglalkozó szerkesztője, biztonsági tanácsadó a Minnesota állambeli Minneapolisban található Upstream Solutions LLC Inc.-nél.

KAPCSOLÓDÓ CÍMEK

A Zorp készítőinek (Balabit Kft.) magyar nyelvű honlapja ➲ http://www.balabit.hu címen érhető el.
A ZorpOS letöltési könyvtárának gyökerében található néhány olyan eszköz, amelyekkel a Zorp GPL használata jóval könnyebbé válik (többek közt az `iptables-utils`, Tproxy-képes Linux-rendszermag és `iptables` parancs). Ezek lényegében a Zorp Próval beszerezhető Debian-terjesztés ingyenes részei, ez az oka annak, hogy a ZorpOS-ben minden Debian csomagok formájában szerepel. Ha nem Debiant használsz, akkor minden szükséges elemet megtalálsz a *pool* könyvtár alkönyvtáraiban. Az egyes csomagok alkönyvtáraiban legfelül forráskódokat tartalmazó *tar.gz* fájlok találhatók. Ha Debiant használsz, akkor a ➲ http://www.balabit.com/downloads/zorp/zorp címet `apt-get` forrásként használhatod.
A Zorp-felhasználók levelezőlistája rendkívül gyors és könnyű módja a segítségkérésnek, akár a Pro, akár a GPL Zorp-változattal akadna gondod. A listára az alábbi oldalon lehet feliratkozni, illetve archívuma is innen érhető el. Megjegyezném, hogy a Balabit egy magyar vállalkozás, így mérnökei (valamint a leghozzáértőbb Zorp-használók egy része) közép-európai idő szerint dolgoznak:
➲ http://https://lists.balabit.hu/mailman/listinfo/zorp