Tűzfalszabályok előállítása Perl program segítségével

Az internet gonosztevőinek köszönhetően már a legtöbb kezdő felhasználó is tisztában van egy biztonságos tűzfal szükségességével.

Mint a legtöbb ***Linux***-felhasználó, kezdetben magam is egyszerű ***Bash*** parancsfájlokkal készítettem el a gépem tűzfalszabályait. Lassan azonban belefáradtam, hogy ugyanazokat a kódsorokat írom újra és újra, így hát elkezdtem ciklusokat használni az ismétlések elkerülésére. El akartam különíteni magukat a szabályokat is a program többi részétől – erre a megoldás az, hogy a program külső beállításfájlokkal dolgozik. Mivel a ***Perlhez*** lényegesen jobban konyítok, mint a ***Bash***-hez, úgy döntöttem, Perlben fogom megírni a tűzfalszabályaimat. A cikkben bemutatott programot meg lehet írni ***Bash***-ben, vagy bármilyen parancsnyelven, de akár ***C++***-ban is. Nem a nyelv a fontos. A lényeg, hogy megírjuk a programot, teszteljük, és ezután már nincs más dolgunk, csak módosítani a beállításfájl biztonsági szabályait és újra lefuttatni a programot. A beállításfájlok szerkezete magáért beszél, így könnyen olvashatók, érthetők, módosíthatók.

Az *1. Listában* látható a ***Perl*** szkript. A fentről lefelé elvet követtem (***top-down programming***), úgyhogy az első néhány sorból már látszik, hogy mit csinál a program. Remélhetőleg nem csak ***Perl*** programozók számára lesz érthető.

Látható, egészen rövid program és nem is bonyolult. Elég rugalmas azonban ahhoz, hogy fekete- és fehérlistákat hozzunk létre egyes gépekből, vagy akár egész hálózatokból. Mint majd látni fogjuk, a `build_chains()` és az `add_rules()` függvényekben alkalmazott szabálykurtító

1. Lista firewall.pl

```
#!/usr/bin/perl
$default_policy = "DROP";
$iptables = "/sbin/iptables";
$work_dir = "/root/fw";
set_ip_forwarding(0);
load_interfaces();
$protocols{tcp}++; $protocols{udp}++; $protocols{icmp}++;
init();
set_default_policy();
add_good_hosts();
add_bad_hosts();
build_chains();
add_rules();
set_default_action();
set_ip_forwarding(1);
exit;
################################################
sub    load_interfaces {
    my($int, $name);
    local(*FILE);
    open FILE, "$work_dir/interfaces.conf";
    while (<FILE>) {
        chomp($_);
        if ($_ eq "") { next; }
        ($name, $int) = split(/\s*=\s*/, $_);
        $interface{$name} = $int;
    }
}
sub    init {
    iptables("-F");  # flush rules
    iptables("-t nat -F");
    iptables("-X");  # delete chains
    iptables("-Z");  # zero counters
    iptables("-t nat -A POSTROUTING -j MASQUERADE");
    iptables("-A INPUT -m conntrack -ctstate ESTABLISHED
        -j ACCEPT");
}
sub    set_default_policy {
    iptables("-P INPUT $default_policy");
    iptables("-P OUTPUT ACCEPT");
    iptables("-P FORWARD ACCEPT");
    return;
```

1. Lista folytatás

```
}
sub    build_chains {
    my($interface, $protocol, $chain);
    foreach $interface (keys %interface) {
        foreach $protocol (keys %protocols) {
            $chain = "$interface-$protocol";
            iptables("-N $chain");
            iptables("-A INPUT -i
            ➥$interface{$interface}
                -p $protocol -j $chain");
        }
    }
}
sub    add_rules {
    local(*FILE);
    open FILE, "$work_dir/ports.conf";
    while (<FILE>) {
        chomp($_);
        $_ =~ s/#.?//;
        if ($_ eq "") { next; }
        ($int, $proto, $port) = split(/\t/, $_);
        $i = $interface{$int};
        $chain = "$int-$proto";
        if ($proto eq "all") {
            foreach $proto (keys %protocols) {
                $chain = "$int-$proto";
                iptables("-A $chain -i $i -p
                ➥$proto -j ACCEPT");
            }
            next;
        }
        if ($proto eq "udp") {
            iptables("-A $chain -i $i -p udp
            ➥-dport $port
                -j ACCEPT");
            iptables("-A $chain -i $i -p udp
            ➥-sport $port
                -j ACCEPT");
        }
        if ($proto eq "tcp") {
            iptables("-A $chain -i $i -p tcp
            ➥-dport $port -syn
                -j ACCEPT");
            iptables("-A $chain -i $i -p tcp
            ➥-dport $port
                -j ACCEPT");
        }
    }
}
sub    set_default_action {
    my($interface, $protocol, $chain);
    foreach $interface (keys %interface) {
        foreach $protocol (keys %protocols) {
            $chain = "$interface-$protocol";
            iptables("-A $chain -j LOG
                -log-prefix
DEFAULT_$default_policy-$chain-");
            iptables("-A $chain -j
            ➥$default_policy");
        }
    }
}
sub    iptables {
    my($line) = @_;
    print "$iptables $line > /dev/null\n" if
    ➥($debug);
    $result = system("$iptables $line >
    ➥/dev/null");
    if ($result != 0) {
        print "X: ($result) iptables $line\n";
    }
}
sub    set_ip_forwarding {
    my($value) = @_;
    local(*FILE);
    print "Setting IP forwarding to $value.\n";
    open FILE, ">/proc/sys/net/ipv4/ip_forward";
    print FILE $value;
    close FILE;
}
sub    add_good_hosts {
    my($host, $comment);
    local(*FILE);
    open FILE, "$work_dir/good_hosts.conf";
    while (<FILE>) {
        ($host, $comment) = split(/\t/, $_);
        iptables("-A INPUT -s $host -j ACCEPT");
        iptables("-A OUTPUT -d $host -j
        ➥ACCEPT");
    }
}
sub    add_bad_hosts {
    my($host, $comment);
    local(*FILE);
    open FILE, "$work_dir/bad_hosts.conf";
    while (<FILE>) {
        chomp($_);
        ($host, $comment) = split(/\t/, $_);
        iptables("-A INPUT -s $host -j LOG
             -log-prefix $comment");
        iptables("-A OUTPUT -d $host -j LOG
             -log-prefix $comment");
        iptables("-A INPUT -s $host -j DROP");
        iptables("-A OUTPUT -d $host -j DROP");
    }
}
```

(*rule-pruning*) algoritmusnak köszönhetően a *Linux* rendszermagnak nem kell a lényegtelen szabályokkal foglalkoznia.

A `set_ip_forwarding()` függvény azt csinálja, ami a neve alapján elvárható, megmondja a rendszermagnak, hogy továbbítson, vagy éppen ne továbbítson *IP* csomagokat. A függvény egyetlen paramétert fogad, 0 vagy 1 értékkel attól függően, hogy akarunk-e *IP* továbbítást.

A szkript a tűzfalszabályok betöltésekor letiltja a továbbítást, majd futásának befejezte előtt újra engedélyezi azt. E plusz lépések célja, hogy az útválasztónk biztonságos állapotban legyen a szabályok

betöltésekor. Jobb minden forgalmat blokkolni, mint egyetlen támadást beengedni.

A `load_interfaces()` függvény beolvassa a hálózati eszközök neveit, majd könnyen megjegyezhető címkéket kapcsol hozzájuk. A további beállítások során már ezekkel a címkékkel hivatkozunk az eszközökre. Az olyan elnevezések, mint a `lan` (helyi hálózat), vagy a `vpn_to_work` (munkahelyi ***VPN***) csökkentik a hibalehetőségek számát, de így könnyebben is módosíthatjuk a tűzfalat, hogy azt máshol mások is használhassák. Gyakran csak az ***interfaces.conf*** fájlt kell módosítanom az adott hálózatnak megfelelően és máris egy jól használható tűzfalat nyújthatok át egy barátomnak.

A szkript négy beállításfájlt használ: ***interfaces.conf***, ***good_hosts.conf***, ***bad_hosts.conf*** és ***ports.conf***.

A *2. Listában* látható az ***interfaces.conf*** fájlom. Látható, hogy az útválasztómban hat hálózati eszköz van.

Az internetkapcsolat az `eth5`. A házon belüli hálózatot ***10/100TX Ethernet*** köti össze. Egy ***MythTV PVR Gigabites Ethernettel*** kapcsolódik az útválasztóhoz fájltárolás céljából. Van még egy csatoló ***Wi-Fihez*** és ***VoIP***-hoz, továbbá egy ***VPN*** kapcsolat néhány barátom számítógépéhez. Könnyebb megjegyezni, hogy a `lan` nevű eszköz a 10/100-as rézszálas hálózat, mint azt, hogy eth3 a ***VoIP***, vagy a ***Wi-Fi*** csatoló. Azt pedig a legkevésbé sem akarjuk, hogy valamelyik csatoló nem a neki készített szabályokat kapja.

Az `init()` függvény végzi az ***iptables*** beállításnak első lépéseit. Először kiürítjük (`flush`) vagy töröljük az összes szabályt és felhasználói láncot. Aztán lenullázzuk a számlálókat. Később ezekből tudhatjuk meg, hogy az egyes tűzfalszabályaink hány csomagot kaptak el. Ezután beállítjuk a címálcázást (***IP masquerading***). Az olyan forgalmat, amely egy létező kapcsolathoz tartozik, további ellenőrzés nélkül átengedjük a tűzfalon. Így nem kell minden csomagnak végigjárnia az összes szabályt, elég, ha csak az új kapcsolatokat vizsgáljuk.

A `set_default_policy()` megadja, hogy mi legyen az egyetlen szabályra sem illeszkedő csomagok sorsa. Ez esetben csak a bejövő forgalommal foglalkozunk, a házirend elfogad minden kimenő és továbbítandó csomagot, de kisebb módosítással minden irányhoz megadhatunk házirendet (***policy***). Az alapértelmezett házirend alapján a szkriptünk eldobja a csomagokat, tehát az adminisztrátornak külön ki kell jelölnie az engedélyezett forgalmat. Ez a legbiztonságosabb megoldás, és lényegesen jobb annál, mintha mindent átengednénk és csak az ártó szándékú forgalmat blokkolnánk. Előre úgysem tudhatjuk, mi az ártó, így a legjobb mindent tiltani és csak azt engedélyezni, ami tényleg szükséges.

Az `add_good_hosts()` a ***good_hosts.conf*** fájlban megadott gépekből és hálózatokból hoz létre szabályokat, melyek minden forgalmat engedélyeznek. Ezek a szabályok nem hálózati eszközhöz kötődnek. A fehérlistán levő gépekről és hálózatokból minden csatolón minden adatforgalmat beengedünk. Ezen a listán általában szerepel az otthoni gépem és a munkahelyi hálózat, így ha valami valami hibát vétek a beállítás közben, akkor sem zárom ki magam az útválasztó gépről, tehát otthonról és a munkahelyemről is javíthatom a hibát. Persze alapfeltétel, hogy mind az otthoni gép, mind a munkahelyi hálózat biztonságos legyen, és lehetőleg ez a lista se legyen hosszú.

Az `add_bad_host()` által létrehozott szabályok blokkolnak minden forgalmat a ***bad_hosts.conf*** fájlban megadott gépek és hálózatok felől. Ez a függvény majdnem ugyanúgy működik, mint az `add_good_hosts()`, egyetlen fontos kivétellel. A feketelistán levő helyekről érkező csomagokat az útválasztó naplózza, a bejegyzéshez pedig egy megjegyzést is fűz úgy, ahogy azt a ***bad_hosts.conf*** fájlban előre megadtuk. Így elég csak belepillantani a naplóba és rögtön látjuk, melyik csomag miért lett eldobva. Hasznos fejlesztés lenne, ha az `add_bad_hosts()` függvény külön láncba tenné a feketelista szabályokat, a láncot pedig egy korai szakaszban hívná meg. Így akár egy külső programból is kényelmesen törölhetnénk, vagy adhatnánk címeket ehhez a lánchoz, esetleg éppen a naplóbejegyzések függvényében.

A `build_chains()` tűzfalszabály láncokat hoz létre. Minden hálózati eszköz és protokoll-kombinációhoz külön láncot építünk. Például egy linuxos útválasztón négy eszközzel – `eth0`, `eth1`, `eth2`, `eth3` – a következő láncaink lennének: `eth0-tcp`, `eth0-udp`, `eth1-tcp`, `eth1-udp` és így tovább. Aztán létrehozzuk a szabályokat, amelyek a megfelelő lánchoz küldik a csomagokat. Eredményül egy faszerkezetet kapunk amely meghatározza az útválasztóra érkező hálózati csomagok útvonalát. Ellentétben egy lineáris tűzfalszabály listával, a rendszermagnak itt nem kell nyilvánvalóan lényegtelen szabályokkal foglalkoznia. Nem fogunk tehát ellenőrizni a ***WAN*** csatolón bejövő ***TCP*** csomagokat a ***Wi-Fi*** csatoló ***UDP*** csomagjaira vonatkozó szabályok alapján.

Nem vizsgáltam még meg, hogy ez a szabály-visszametsző eljárás eredményez-e számottevő teljesítménynövekedést. Nem kerül azonban semmibe ennek a döntési fának a létrehozása, tehát ha a teljesítményt nem is növeli jelentősen, nem is igényel több programozói munkát, tehát összességében mégiscsak megéri.

A munka oroszlánrésze az `add_rules()` függvényben történik, amely beolvassa az *5. Listában* látható ***ports.conf*** fájl tartalmát.

2. Lista interfaces.conf

```
lo = lo
gig = eth0
lan = eth1
wifi = eth2
voip = eth3
wan = eth5
tun = tun0
```

3. Lista good_hosts.conf

```
127.0.0.1     Loopback
224.0.0.0/8     Multicast
10.4.0.0/16     VPN
10.0.1.1/32     Home office
```

4. Lista bad_hosts.conf

```
216.250.128.12  My_comment
www.microsoft.com       Micro
                        ➥soft
```

5. Lista ports.conf

```
wan   tcp   22   # ssh
wan   tcp   25   # smtp
wan   tcp   80   # http
wan   udp   53   # dns
wan   udp   1194   # openvpn
wan   udp   5060   # sip
wan   udp   4569   # iax2
wan   udp   10000:20000   # rtp
lo   all
lan   tcp   22   # ssh
lan   tcp   25   # smtp
lan   udp   53   # dns
lan   tcp   53   # dns
lan   udp   67   # dhcp
lan   udp   68   # dhcp
lan   tcp   80   # http
lan   tcp   111   # portmapper
lan   udp   111   # portmapper
lan   tcp   143   # imap
lan   tcp   443   # https
lan   tcp   2049   # nfs
lan   udp   2049   # nfs
lan   tcp   3306   # mysql
lan   udp   4569   # iax2
lan   udp   5060   # sip
lan   tcp   5432   # postgresql
lan   tcp   10000   # webmin
lan   all
gig   all
tun   all
wifi   udp   1194   # openvpn
voip   udp   5060   # sip
voip   udp   4569   # iax2
voip   udp   53   # dns
voip   tcp   22   # ssh
voip   udp   10000:20000   # rtp
voip   tcp   80   # http
```

Mielőtt részletesebben megvizsgálnánk az `add_rules()` függvényt, lássuk a ***ports.conf*** fájlt.
A ***ports.conf*** minden szabályhoz tartalmaz egy sort. Minden sor három oszlopból és egy választható megjegyzésből áll, amelyet # karakter előz meg. Az első annak a hálózati eszköznek a neve, azaz címkéje, amelyre a szabály vonatkozik. A második oszlop a protokoll: `tcp`, `udp`, vagy mindegyik, azaz `all`. Utóbbi esetben olyan szabályt kapunk, amely minden csomagot beenged a kérdéses csatolón. Az harmadik oszlop a kapuszám, vagyis a port. Az első sor például egy olyan szabályt hoz létre, amely beengedi az ***SSH*** forgalmat a `wan` hálózati csatolón.
Látható, hogy van egy szabály, amely engedélyezi a `lo - loopback`, azaz visszacsatoló hálózati eszközön az adatforgalmat. Enélkül sok program nehezen felderíthető hibákat produkálna. Felvetődhet a kérdés, hogy miért van szükség ennyi szabályra a ***LAN*** eszközön, hogy aztán a végén egy `all`-lal minden csomagot beengedjünk. A fő ok, hogy amíg a gyerekek fel nem nőnek és el nem kezdik használni az internetet, az otthoni hálózatot biztonságosnak tarthatom. Ha azonban minden szolgáltatáshoz külön szabályt készítek, remek statisztikát kapok arról, hogy melyik szolgáltatás mennyi forgalmat generált. No meg a biztonság biztosítása folyamatos munka. Időről időre újabb szabályokat adok a tűzfalam szigorításához, aztán előbb utóbb az az `all` is eltűnik majd a házirendből.
Térjünk vissza az `add_rules()` függvényre. Ez a program leghosszabb függvénye, de azért még könnyen érthető. A kód `tcp` és `udp` szabályokkal foglalkozó része egyszerűen csak két szabályt készít a ***ports.conf*** fájl minden szabálya alapján. Egyik a cél-, a másik a forráskapuhoz kapcsolódik. Ez elsőre furcsának tűnhet, mivel csak a bejövő forgalommal foglalkozunk. Igazából azonban így biztosítjuk, hogy mind a bejövő, mind a kimenő kapcsolat engedélyezve legyen. Például egy a ***WAN*** eszköz 80-as kapujára érkező kapcsolat a saját web kiszolgálómat célozza meg, míg ugyanezen az eszközön a 80-as kapuról érkező adat egy külső kiszolgáló válasza egy, a belső hálózatról érkező kérésre.
Az `all` szabályokat kezelő kód speciális eset. Itt minden protokollra külön szabályt alkotunk az adott hálózati eszközön. Talán túl bonyolultnak tűnket, de van egy érdekes mellékhatása. Ha az útválasztó ismeretlen protokollú csomagot kap, mint mondjuk az ***IPSec***, az alapértelmezett házirendet fogja alkalmazni, még akkor is, ha az azt jelenti, hogy engedjen át minden adatforgalmat. A „minden protokoll" tehát „minden ismert protokoll"-t jelent, és szerintem ez jó.
A szkript nagyjából olyan sorrendben adja át a szabályokat a rendszermagnak, ahogy a ports.conf fájlban szerepelnek. Mondom csak nagyjából, mivel a szabályok a hálózati eszköz és protokoll alapján meghatározott láncba kerülnek. A szabályon belül azonban nem változik a sorrend.
A `set_default_action()` függvény meghatározza, hogy mi történjen azokkal a csomagokkal, amelyekre egyetlen korábbi szabály sem illeszkedik. Ez hasonlónak tűnhet a `set_default_policy()`-hez, de van egy apró különbség.
A `set_default_policy()` az alapértelmezett tűzfal házirendet állítja be, a `set_default_action()` tűzfalszabályokat hoz létre, amelyek elkapják a korábban el nem kapott csomagokat, lezárva az egyes láncokat, azelőtt, hogy a rendszermag az alapértelmezett házirendet alkalmazná.
Ha egy megfelel egy ilyen szabálynak, létrejön egy napló bejegyzés, majd életbe lép a megadott házirend, ez esetben a `DROP`. A naplóból később megállapítható, hogy milyen csomag és miért lett eldobva.
Nem mondom, hogy ez a program tökéletes és azt sem, hogy minden elképzelhető feladatot képes ellátni. Még akár hibák is lehetnek benne. Mire ez a cikk megjelenik bizonyára további fejlesztéseket végzek a szkripten.
Látható, hogy jelenleg még egyáltalán nem kezeli az ***ICMP*** protokollt, pedig jó lenne például engedélyezni a kimenő `ping` kéréseket, de tiltani a bejövőket. Jó lenne szabályozni a kimenő és a továbbított forgalmat is. Mivel használok ***VoIP***-ot, gondoltam már arra, hogy a szkript beállíthatná a szolgáltatásminőséget (***Quality of Service - QoS***) is. Aki kiegészíti hasznos funkciókkal a programot, kérem értesítsen.
Most olyan, amilyen. Kevesebb, mint 200 sornyi ***Perl*** kód, mégis rugalmasan és hatékonyan kezeli akár tűzfalszabályok százait, a szabályok módosítása pedig olyan egyszerű, hogy még az egészen kezdő ***Linux***-felhasználóknak sem okozhat gondot.

Linux Journal 2006., 143. szám

Mike Diehl a SAIC-nak dolgozik a Sandia National Laboratories-nál Albuquerque-ben, Új-Mexikóban, hálózati felügyeletei szoftvereket fejleszt. Feleségével és két kisfiával él. E-mailben elérhető a mdiehl@diehlnet.com címen.