Linux netfilter Hacking HOWTO

Rusty Russell, mailing list `netfilter@lists.samba.org`
Fordította Kis-Szabó András*kisza@sch.bme.hu*
v1.11 2001/10/31, fordítás ideje: 2001. december 20.

Ez a dokumentum bemutatja a netfilter felépítését, hogyan kell hozzá kiegészítéseket írni, és néhány nagyobb, erre az architektúrára épülõ rendszer is bemutatásra kerül, mint pl. a csomagszûrés (packet filtering), kapcsolatkövetés (connecion tracking) és címfordítás (Network Address Translation).

Contents

1 Bevezetés

Szervusztok!

Ez a leírás olyan, mint egy utazás; egyes részei jól ki vannak dolgozva, míg más részeken esetleg egyedül érezhetitek magatokat. A legjobb tanács, amit adhatok Nektek, hogy szerezzetek egy nagy bögre kávét vagy meleg kakaót, üljetek le egy kényelmes székbe, és figyeljétek a tartalomjegyzéket mielõtt belevágtok a hálózat- programozás néha veszélyes világába.

A netfilter felületén elhelyezkedõ szerkezet jobb megismeréséhez ajánlom a Packet Filtering HOWTO és a NAT HOWTO átolvasását. A kernel programozásához szükséges információkhoz ajánlom a következõ leírásokat: Rusty's Unreliable Guide to Kernel Hacking és Rusty's Unreliable Guide to Kernel Locking.

1.1 Mi az a netfilter?

A netfilter egy váz a csomagok megváltoztatására, ami a hagyományos Berkeley socket felületen kívül helyezkedik el. Négy része van. Elsõ: minden protokoll definiál hook-okat (IPv4-nél 5 darab van), amik jól definiált pontokat határoznak meg a csomagok protokoll-stack-beli útjuk során. Mindegyik ilyen ponton a protokoll képes meghívni a netfilter vázat a csomaggal és a hook sorszámával.

Második: a kernel részei regisztrálni tudják magukat a különbözõ hook-okhoz minden protokoll esetében. Amikor a csomag megérkezik a netfilter vázhoz, az ellenõrzi, hogy valaki regisztrálta-e magát a megadott protokollhoz és hookhoz. Amennyiben van ilyen rész, akkor mindegyik lehetõséget kap a csomag megvizsgálására (esetleg megváltoztatására), ezután figyelmen kívül hagyhatja a csomagot (`NF_DROP`), átengedheti (`NF_ACCEPT`), kiveheti a netfilterbõl a csomagot (`NF_STOLEN`), vagy kérheti a netfiltert, hogy állítsa sorba a csomagot az userspace programok számára (`NF_QUEUE`).

Harmadik: a sorba állított csomagok (az ip_queue driverrel) a felhasználói programokhoz szabályozottan kerülnek elküldésre. A csomagok kezelése aszinkron.

Az utolsó rész a jó forráskód kommentezésbõl és a csodálatos dokumentációból áll. Ez szükséges bármilyen kísérleti projecthez. A netfilter mottója (szemtelenül lopva Cort Dougan-tól):

```
``So... how is this better than KDE?''
```

Ehhez az egyszerû vázhoz számos kiegészítés készült, amik az elõzõ kernelekhez hasonló funkcionalitást adnak a rendszerhez, ide értve a bõvíthetõ NAT rendszert, valamint a bõvíthetõ csomagszûrõ rendszert (iptables).

1.2 Mi a baj azzal, amit a 2.0 és 2.2-es verziókban találhatunk?

1. Nincs kialakított metódusa a csomagok felhasználói térbe való továbbításának
   - Kernel programozás nehéz
   - Kernel programozást C/C++ -ban kell végezni
   - Dinamikus szûrési szabályok nem tartozhatnak a kernelhez
   - 2.2 bevezette a csomagok userspace-be küldését netlinken keresztül, de a csomagok ismételt elküldése lassú. Például nm lehetséges a csomag olyan újraküldése, mintha valós interface-rõl érkezne.
2. Transzparens proxyzás olyan, mint egy cserépedény:
   - **Minden** csomagot meg kell néznünk, hogy van-e olyan socket, ami az adott címre van bind-olva
   - A root idegen címekre is bind-olhat
   - Nem lehet a helyileg készült csomagokat átirányítani
   - REDIRECT nem kezeli le az UDP válaszokat: az UDP névszerver csomagok átirányítása a 1153-as portra nem mûködik, mert egyes kliensek nem szeretik azokat a válaszokat, amik nem az 53-as portról érkeznek.
   - REDIRECT nincs összhangban a tcp/udp port-hozzárendeléssel: a felhasználó kaphat olyan portot, ami egy REDIRECT szabállyal el van fedve.

- Legalább kétszer hibás volt a 2.1-es sorozat alatt.
- A program-kód nagyon csúnya, tolakodó. Az #ifdef CONFIG_IP_TRANSPARENT_PROXY a 2.2.1-es kernelben 34-szer fordul elõ 11 fileban, szemben a CONFIG_IP_FIREWALL-al, ami 10-szer található meg 5 fileban.

3. Interface-tõl független csomagszûrõ szabályok készítése nem lehetséges:
   - Muszáj tudni a helyi interface címét, hogy a helyileg induló és végzõdõ csomagokat meg tudjuk különböztetni az áthaladóktól.
   - S mi több: ez nem elég a redirection vagy masquerading esetén.
   - Forward láncnak csak a kimenõ interface-en van információja a csomagról, ami azt jelenti, hogy a hálózati felépítés ismeretében magadnak kell kitalálnod a csomag lehetséges eredetét.
4. Masquerading össze van fûzve a csomagszûréssel:A masquerading és a szûrés közti kölcsönhatások bonyolulttá teszik a tûzfal kialakítását:
   - Bemeneti szûrésnél a válaszcsomag mintha magának az eszköznek szólna
   - Átmenû szûrésnél a visszaalakított (demasquerad) csomagok nem kerültek ellenõrzésre soha többé
   - Kimeneti szûrésnél: mintha maga az eszköz küldte volna a csomagot
5. TOS módosítás, átirányítás, ICMP unreachable és mart (ami a port átirányításra, routolásra és a QoS-re van hatással) szintén a csomagszûrési programkódban kerültek megvalósításra.
6. ipchains kód se nem moduláris, se nem bõvíthetõ (pl. MAC cím szûrés, opciók szûrése, stb.).
7. A szükséges alapok hiánya a különbözõ megoldások bõséges választékához vezetett:
   - Masquerading, és protokollonkénti modulok
   - Gyors állandó NAT routolási kóddal (nincs protokoll támogatás)
   - Port forwarding, redirect, automatikus forwarding
   - The Linux NAT és Virtual Server Project.
8. A CONFIG_NET_FASTROUTE és a csomagszûrés összeférhetetlensége:
   - Továbbított csomagok három láncon is áthaladnak
   - Nincs lehetõség ezeknek a láncoknak a kihagyására
9. Routing védelem (pl. forráscím-ellenõrzés) nem lehetséges a csomagok ellenõrzésének hiánya miatt
10. Nincs lehetõség a csomagszûrõ szabályok számlálóinak automatikus olvasására.
11. CONFIG_IP_ALWAYS_DEFRAG egy fordítási opció, nehezebbé téve az életet azok számára, akik egységes felépítést szeretnének.

1.3 Ki vagy?

Az egyetlen, aki olyan bolond, hogy ezt csinálja! Mint az ipchains társszerzõje és az aktuális Linux Kernel IP Firewall karbantartója látom azokat a problémákat, amelyek az alkalmazás során elõjönnek, valamint azokat a feladatokat, amiket egyes emberek megpróbálnak megoldani.

1.4 Miért omlott össze?

Woah! Bizonyára az **elmúlt** héten láttad!

Azért, mert nem vagyok egy nagy programozó - legalábbis mint szeretném, és természetesen nem tudtam minden esetet kitesztelni idõhiány, felszereltség és/vagy inspiráció miatt. Van egy tesztállományunk, aminek a kibõvítését bátran felajánlom.

2 Hol érhetem el az utolsó változatot?

Van egy CVS szerver a samba.org-on, ami tartalmazza a legutolsó HOWTO-kat, userspace eszközöket és tesztbázist. Alkalmi tallózásra használhatod a

Webes felületet http://cvs.samba.org/cgi-bin/cvsweb/netfilter/ .

Az utolsó változat megszerzéséhez a következõt kell tenned:

1. Jelentkezz be a SAMBA CVS szerverbe anonymous-ként:

   ```
   cvs -d :pserver:cvs@cvs.samba.org:/cvsroot login
   ```

2. Ajelszóhoz gépeld be, hogy ‘cvs’.
3. A kód kikéréséhez használd a következõt:

   ```
   cvs -d :pserver:cvs@cvs.samba.org:/cvsroot co netfilter
   ```

4. Az utolsó verzióhoz a következõt használd:

   ```
   cvs update -d -P
   ```

3 Netfilter Felépítés

Netfilter csupán hook-ok sorozata a protokoll-stack különbözõ pontjain (jelen állapotban IPv4, IPv6 és DECnetben). Az (idealizált) IPv4 diagramm a következõképpen néz ki:

```
Csomag útja a netfilter rendszerben:

   --->[1]--->[ROUTE]--->[3]--->[4]--->
                 |            ^
                 |            |
                 |         [ROUTE]
                 v            |
                [2]          [5]
                 |            ^
                 |            |
                 v            |
```

A csomag a bal oldalon érkezik be: átesik az alapvetõ ésszerûségi ellenõrzéseken (pl. nem csonkolt, IP checksum rendben, nem promiscous csomag), átadásra kerül a netfilter keretrendszernek az NF_IP_PRE_ROUTING [1] hookon.

Ezután belépnek a rouolási kódba, itt dõl el, hogy a csomag egy másik interface felé tart, vagy helyi feldolgozásra kerül. A routolási kód eldobhatja a nem továbbítható csomagokat.

Amennyiben az eszköznek érkezett a csomag, a netfilter váz újra meghívódik a NF_IP_LOCAL_IN [2] hookkal, még mielõtt megkapná a program (ha van ilyen).

Ha egy másik interface-nek kell továbbítani, akkor a netfilter rendszer meghívja a NF_IP_FORWARD [3] hookot.

A csomag ezután az utolsó hookhoz érkezik, a NF_IP_POST_ROUTING [4] hookhoz, mielõtt ismét kikerülne a hálózatra.

Az NF_IP_LOCAL_OUT [5] hook a helyileg keletkezett csomagokra hívódik meg. Mint látható, az útvonalválasztás csak a hook után történik meg: abban az esetben, ha a routing kód elõbb kerülne meghívásra (a forráscím és pár opció kiszámítására) - ha meg akarod változtatni a routolást, magát az 'skb->dst' mezõt kell módosítanod, ahogy az a NAT kódban is történik.

3.1 Netfilter Alapok

Van egy példánk az IPv4-es netfilterhez, ahol láthatod, hogy minden hook meghívódik. Ez a lényege a netfilternek.

Kernel modulok bármelyik hookra regisztrálhatják magukat. A modulnak, ami regisztrálja magát, prioritást kell rendelnie a funkciójához; ezután amikor a netfilter hook aktivizálódik a belsõ hálózati részekbõl, minden bejegyzett modul meghívásra kerül a prioritási sorrend alapján, s szabadon módosíthatja a csomagot. A modul öt dologra kérheti a netfiltert:

1. NF_ACCEPT: folytassa az útját.
2. NF_DROP: dobja el a csomagot, ne folytassa az útját.
3. NF_STOLEN: átvettem a csomagot, ne folytassa az útját.
4. NF_QUEUE: rakja be a sorba a csomagot (rendszerint userspace-beli feldolgozásra).
5. NF_REPEAT: ismételje meg a hook-ot.

A netfilter egyéb részei (sorba-állított csomagok kezelése, kiegészítések) a kernel részben kerülnek bemutatásra.

Ezen az alapon nagyon összetett csomagmanipulációt tudunk kiépíteni, mind azt a következõ két szakasz is bemutatja.

3.2 Csomag kiválasztás: IP Tables

A csomag-kiválasztási rendszert IP Tablesnek hívják, s a netfilter vázra épült. Közvetlen leszármazottja az ipchains (ami az ipfwadmnak, s ami a BSD ipfw-jének) - csak bõvíthetõséggel. Kernel modulok tudnak új táblákat bejegyezni, s arra ítélni egy csomagot, hogy az adott táblán haladjon végig. Ez a kiválasztás használatba kerül a csomagszûrésnél ('filter' tábla), hálózati címfordításnál (NAT) ('nat' tábla) valamint az általános routolás elõtti csomagmódosításnál ('mangle' tábla).

A következõ hookok vannak regisztrálva a netfilterben (abban a sorrendben felsorolva, ahogyan meghívásra kerülnek):

```
--->PRE------>[ROUTE]--->FWD---------->POST------>
    Conntrack    |       Filter   ^    NAT (Src)
    Mangle       |                |    Conntrack
    NAT (Dst)    |             [ROUTE]
```

```
        (QDisc)      v                |
                     IN Filter       OUT Conntrack
                     |  Conntrack     ^  Mangle
                     |                |  NAT (Dst)
                     v                |  Filter
```

3.2.1 Csomagszûrés

Ez a tábla, a 'filter' sose változtatja meg a csomagot: csak megszûri.

Az iptables elõnye az ipchains-szel szemben, hogy kicsi és gyors, valamint a netfilterbe az NF_IP_LOCAL_IN, NF_IP_FORWARD és NF_IP_LOCAL_OUT pontokon kapcsolódik. Ez azt jelenti, hogy minden egyes csomag egy (és csak is egy) helyen kerülhet megszûrésre. Ez sokkal könnyebbé teszi a használatát az ipchains-szel szemben. Szintén elõny, hogy a netfilter váz biztosítani képes a be és kimenõ interface-t is az NF_IP_FORWARD hooknak, ami számos szûrést egyszerûbbé tesz.

Megjegyzés: az ipchains és az ipfwadm is portolásra került a netfilter vázra, lehetõvé téve a meglevõ rendszerek használatát.

3.2.2 NAT

Ez a 'nat' tábla birodalma, ami két helyrõl eszi a csomagokat: a nem helyi csomagokat a NF_IP_PRE_ROUTING és a NF_IP_POST_ROUTING hookokról, amik lehetõséget adnak a cél és forrás megváltoztatására. Ha a CONFIG_IP_NF_NAT_LOCAL definiált, a NF_IP_LOCAL_OUT és NF_IP_LOCAL_IN hookok kerülnek használatba a helyi eredetû csomagok esetében.

Ez a tábla kevésben tér el a 'filter' táblától: a kapcsolatnak csak az elsõ csomagja halad keresztül a láncon, s az eredmény ezután minden csomagra alkalmazva lesz a kapcsolat alatt.

Masquerade, Port Forward, Transparent Proxy A NAT-ot két részre bontottam: Source NAT (ahol a csomag forrása változhat) és Destination NAT (ahol az elsõ csomag célja változhat).

Masquerading az SNAT egy speciális formája; a port forwarding és a transzparens proxy-zás a DNAT esetei. Ezek mindegyike megvalósítható a NAT keretrendszerrel, ahelyett, hogy különálló rendszerek lennének.

3.2.3 Csomag megváltoztatás

A csomagváltoztató tábla ('mangle' tábla) használható a csomag tartalmának megváltoztatására. Ez a NF_IP_PRE_ROUTING és a NF_IP_LOCAL_OUT pontokon kapcsolódik a netfilterhez.

3.3 Kapcsolatkövetés

Kapcsolatkövetés sarkalatos pontja a NAT-nak, de ennek ellenére külön modulként került megvalósításra; ez megengedi, hogy a csomagszûrõ egy kiegészítése egyszerûen és tisztán használja a kapcsolatkövetést (a 'state' modul).

3.4 Egyéb kiegészítések

Az új flexibilitás lehetõséget ad igazán vad dolgokra, de azok számára is nyitott a lehetõség, akik bõvítéseket, vagy teljesen helyettesítõ részeket szeretnének belevenni a rendszerbe.

4 Programozói információk

Elmondok egy titkot: az én kicsi hörcsögöm végezte a teljes kódolást. Én csak egy csatorna, egy arcvonal vagyok a kis kedvencem nagy tervében. Nos, ne engem hibáztassatok, ha hiba van benne, hanem az aranyos, kis szõröst.

4.1 ip_tables megértése

iptables egyszerûen a szabályok egy nevesített tömbjét szolgáltatja (innen a név: ip-táblák), valamint informálnak a beérkezõ csomagok útjáról. Miután egy tábla bejegyzésre került, a felhasználói programok képesek olvasni és kicserélni a tartalmát a getsockopt() és setsockopt() függvényekkel.

iptables nincs bejegyezve egyik netfilter hookhoz sem: arra számít, hogy más modulok megteszik ezt, s a csomagokat helyes sorrendben továbbítják felé; egy modult be kell jegyezni a netfilter hookokra, valamint az ip_tables-be is, valamint lehetõséget kell adni az ip_tables meghívására, ha a hook elérésre került.

4.1.1 ip_tables adatstruktúra

A kényelmesség miatt azonos adatstruktúra társul egy szabályhoz az userspace-ben és a kernelben is, annak ellenére, hogy egyes mezõk csak a kernelben kerülnek alkalmazásra.

Minden szabály a következõ részekbõl áll:

1. 'struct ipt_entry'
2. Nulla vagy több 'struct ipt_entry_match' struktúra, mindegyik változó hosszúságú (0 vagy nagyobb) adatterülettel.
3. 'struct ipt_entry_target' struktúra, változó hosszúságú (0 vagy nagyobb) adatterülettel.

A szabály változó természete nagy szabadságot kölcsönöz a bõvítményeknek, mint láthatjuk is akár minden match vagy target különbözõ mennyiségû adatot hordozhat. Ez azonban néhány csapdát hordoz magában: figyelni kell az igazításra, kerekítésre. Ennek során ügyelünk, hogy a 'ipt_entry', 'ipt_entry_match' és a 'ipt_entry_target' struktúrák megfelelõ méretûek legyenek, és a rendszeren elérhetõ legnagyobb igazítási méretre legyenek felkerekítve (IPT_ALIGN() macro).

'struct ipt_entry' mezõi:

1. 'struct ipt_ip' rész: specifikációkat tartalmaz az IP fejlécre vonatkozóan, amire egyezni fog.
2. 'nf_cache' bitmezõ, ami azt mutatja meg, hogy a csomag melyik mezõit vizsgálja a szabály.
3. 'target_offset' mezõ, ami a szabály az ipt_entry_target struktúra elejétõl mért távolságát mutatja. Ez mindit igazított érték (IPT_ALIGN macro).
4. 'next_offset' adja meg a szabály teljes méretét, beleértve az egyezéseket és a célokat. (match és target). Ez szintén igazított érték (IPT_ALIGN macro).
5. 'comefrom' a kernel által használt változó, a csomag útjának követésére.
6. 'struct ipt_counters' mezõ tartalmazza a csomag és byte számlálókat, amik az adott szabállyal való egyezésre mutatnak.

A 'struct ipt_entry_match' és 'struct ipt_entry_target' struktúrák nagyon hasonlóak: tartalmazzák a teljes (IPT_ALIGN-olt) hossz mezejét ('match_size' és 'target_size'), valamint egy unionban a match vagy target nevét (userspace) és mutatóját (kernel).

A szabályszerkezet trükkös természete miatt pár segédfunkció is elérhetõ:

ipt_get_target()
: Ez a beépített függvény visszaad egy pointert a szabály targetjére.

IPT_MATCH_ITERATE()
: Ez a makró meghívja az adott funkciót minden egyes match-ra az adott szabályban. A függvények elsõ argumentuma egy 'struct ipt_match_entry', míg a többi (ha létezik) az, amit az IPT_MATCH_ITERATE() makró kapott. A funkció nullát ad vissza az iteráció folytatásához, nem nulla értéket a megszakításához.

IPT_ENTRY_ITERATE()
: Ez a funkció mutatókat vár egy bejegyzésre, a tábla teljes bejegyzéseinek méretére, valamint a meghívandó funkcióra. A funkció elsõ argumentuma egy 'struct ipt_entry', és a további argumentumai (ha vannak) megegyeznek az IPT_ENTRY_ITERATE()-ben megadottakkal. A funkció nullát ad vissza az iteráció folytatásához, nem nulla értéket a megszakításához.

4.1.2 ip_tables userspace formája

Userspace-nek négy mûvelete van: olvasni tudja az aktuális táblát, információhoz juthat (hook helye, tábla mérete), kicserélheti a táblát (és megtarthatja a régi számlálókat), és új számlálókat adhat hozzá.

Ezzel bármilyen elemi mûvelet szimulálható userspace-bõl: a libiptc könyvtáron keresztül, ami kényelmes "add/delete/replace" szemantikát ad a programokhoz.

Amiért ezek a táblák továbbításra kerülnek a kernelbe, az igazítás komoly kérdés olyan rendszerekben, ahol eltérõ a méret (pl. Sparc64 kernel 32bites userspace-el). Ezek az esetek az IPT_ALIGN makró felüldefiniálásával vannak megoldva a 'libiptc.h'-ban.

4.1.3 ip_tables használat

A kernel azon a helyen kezdi el az értelmezést, ahol az adott hook kívánja. Az a szabály kerül vizsgálatra, aminek a 'struct ipt_ip' elemei megegyeznek, minden 'struct ipt_entry_match' sorban ellenõrzésre kerül (a match-al összerendelt függvényen keresztül). Ha a match függvény 0-t ad eredményül, akkor megáll az értelmezés. Ha a 'hotdrop' paramétert 1-be állítja, a csomag azonnal eldobásra kerül.

Ha az iteráció végigér a számlálók növelésre kerülnek, s a 'struct ipt_entry_target' kerül megvizsgálásra: ha ez egy alap target, akkor a 'verdict' mezõ kerül olvasásra (negatív jelenti azt, hogy már van döntés, a pozitív pedig egy ugrási eltolást ad meg.) Ha pozitív a válasz, s az offset nem a következõ szabályra mutat, a 'back' változó beállításra kerül és az elõzõ 'back' értéke a szabály 'comefrom' mezõjébe kerül.

A nem standard targeteknél a target függvény hívódik meg: ez egy döntést ad vissza (nem alap targetek nem tudnak ugrani, ugyanis megsérthetnék a hurokdetektálást). A döntés lehet IPT_CONTINUE a következõ szabályon való továbbhaladáshoz.

4.2 Iptables bõvítése

Amiért ilyen lusta vagyok, az `iptables` meglehetõsen jól bõvíthetõ. Ez alapjában egy csalás a munka másra való áthárításával, ami kb. az, amirõl az Open Source szól (Free Software - ahogy RMS mondaná - a szabadságról szól, és én egy beszédén ülve írtam ezt.)

`iptables`i kibõvítése potenciálisan két részbõl áll: a kernel kibõvítése egy új modul írásával, és lehetõség szerint az userspace rész `iptables` programjának bõvítése egy új shared könyvtár írásával.

4.2.1 A Kernel

Egy kernel modult írni magában egy egyszerû feladat, ahogy azt a példában is láthatod. Amire oda kell figyelned, hogy a kódodnak újra-belépõnek kell lennie: elõfordulhat, hogy egy csomag érkezik az userspace-bõl, míg egy másik egy megszakításon keresztül. SMP esetében minden CPU esetében lehet csomag a megszakításokon (2.3.4 és fölötte).

Azok a funkciók, amikrõl tudnod kell:

init_module()

Ez a modul belépési pontja. Ez egy negatív hibaszámot ad vissza, vagy 0-t, ha sikeresen regisztrálta magát a netfilterben.

cleanup_module()

A modul kilépési pontja; itt veheti ki magát a modul a netfilterbõl.

ipt_register_match()

Ez egy új match bejegyzésére használható. Egy 'struct ipt_match'-al kezelhetõ, amit rendszerint static-ként deklarálnak.

ipt_register_target()

Ez agy új target bejegyzésére használható. Egy 'struct ipt_target'-al kezelhetõ, amit rendszerint static-ként deklarálnak.

ipt_unregister_target()

A tergetem visszavonására használható.

ipt_unregister_match()

A match-em visszavonására használható.

Egy figyelmeztetés a trükkös dolgokkal kapcsolatban (mint pl. számlálók nyújtása) az extra helyekben az új match-ben vagy target-ban. SMP eszközön a teljes tábla megduplázódik egy memcpy()-val minden CPU-ra: ha valóban központi információt akarsz tárolni, akkor nézd meg azt, ahogy ez a 'limit' match-ben megvalósításra került.

Új match funkciók Új match funkciók rendszerint különálló modulokként kerülnek megírásra. Ez lehetõvé teszi ezeknek a moduloknak a felváltott bõvítését, bár ez rendszerint nem szükséges. Egyik lehetõség a netfilter váz 'nf_register_sockopt' funkciója a felhasználói kapcsolatteremtésre. Másik lehetõség szimbólumok kiexportálása más modulok felé, ahol regisztrálhatják magukat, azonos módon, mint ahogy a netfilter és az ip_tables csinálja.

Az új match funkciód központi része az ipt_match struktúra, ami az 'ipt_register_match()'-nak kerül átadásra. A struktúra szerkezete:

list

Tetszõleges junk lehet, állítsd '{ NULL, NULL }'-ra.

name

Ez a mezõ tartalmazza a match funkció nevét, ahogyan az userspace-bõl hivatkozunk rá. A név lehetõleg egyezzen meg a modul nevével (pl. ha a név 'mac', akkor a modul neve legyen 'ipt_mac.o'), hogy az automatikus betöltés mûködhessen.

match

Ez egy mutató a match funkcióra, ami megkapja az 'skb', az 'in' és 'out' device mutatókat (ami lehet NULL, a hook-tól függõen), egy mutatót a match adatra az éppen feldolgozott szabályban (az a struktúra, ami az userspace-ben készült), IP offsetet (nem nulla jelenti hogy nem-fejléc csomag), egy pointert a protokollfejlécre, az adat hosszát (pl. a csomag hossza az IP fejléc méretével csökkentve) és végül egy mutatót a 'hotdrop' változóra. Ez nem-nulla értékkel jelzi, ha a csomag egyezett, és a 'hotdrop' 1-be állításával ill. 0 visszaadásával dobathatja el azonnal a csomagot.

checkentry

Ez egy mutató, ami egy olyan függvényre mutat, ami ellenõrzi a szabály specifikációját; ha 0-t ad vissza, akkor nem lett elfogadva a szabály. Például: a 'tcp' match típus csak TCP csomagokat fog elfogadni, s ha a 'struct ipt_ip' része a szabálynak nem tartalmazza, hogy a protokollnak TCP-nek kell lennie, nullát ad vissza. A táblanév argument segít megtalálni, hogy hol van a szabály, míg a 'hook_mask' bitmask megadja, hogy melyik hookokból kerülhet meghívásra a szabály. Ha a szabály nem függ a hookoktól, akkor figyelmen kívül lehet hagyni.

destroy

Ez egy mutató egy olyan függvényre, ami akkor hívódik meg, amikor a match-ot tartalmazó szabály törlésre kerül. Ez lehetõvé teszi a dinamikus területfoglalást a chechkentry-ben, s a felszabadítást.

me

Ez a mezõ 'THIS_MODULE'-ra van beállítva, egy pointert ad erre a modulra. Egy használatszámlálóhoz van kötve, ami fel- le változik amikor szabály születik vagy törlésre kerül. Ez meggátolja a felhasználót a modul eltávolításában (cleanup_module()) ha szabály tartalmazza.

Új target-ek Az új target-ek rendszerint különálló modulként kerülnek megvalósításra. A tárgyalás módja megegyezik az 'Új match funkciók' fejezetben találhatókkal.

Az új targeted központi része az ipt_target struktúra, ami az ipt_register_target()-nek kerül átadásra. A struktúra a következõ mezõket tartalmazza:

list

A mezõ értéke tetszõleges junk lehet, legyen '{ NULL, NULL }'.

name

A target neve, ahogyan az userspace-bõl hivatkoznak rá. A név lehetõleg egyezzen meg a modul nevével (pl. ha a név 'REJECT', a modult hívjad 'ipt_REJECT.o'-nak), hogy az automatikus betöltés mûködjön.

target

Pointer a target funkcióra, ami megkapja az skbuff-ok, a hook számok, a be- és kimenõ eszközöket (bármelyik lehet NULL), egy pointert a target adataira és a szabály helyét a táblában. A visszatérési érték lehet IPT_CONTINUE(-1), ha a vizsgálat folytatódhat, vagy egy netfilter döntés (NF_DROP, NF_ACCEPT, NF_STOLEN stb.).

checkentry

Egy mutató arra a funkcióra, amely a szabály szerkezetét ellenõrzi. Nullával jelzi, ha a megadott szabály nem elfogadható.

destroy

A target törlésekor meghívandó függvényre egy mutató. Lehetõség van a checkentry-ben lefoglalt területek felszabadítására.

me

A mezõ értéke 'THIS_MODULE', ami egy pointert ad a modulra. Tartalmaz egy számlálót, aminek az értéke nõ vagy csökken amikor a targetre hivatkoznak, vagy megszüntetik a hivatkozást. Ez meggátolja a felhasználót a modul eltávolításában (cleanup_module()), ha szabály hivatkozik rá.

Új táblák Tetszõleges célra létrehozhatsz egy táblát, amikor csak akarod. Ehhez a 'ipt_register_table()'-t kell meghívnod egy 'struct ipt_table' struktúrával, aminek a következõ mezõi vannak:

list

A mezõ értéke tetszõleges junk, legyen '{ NULL, NULL }'.

name

A táblának a nevét atrtalmazza, ahogyan az userspace-bõl hivatkozunk rá. A név lehetõleg egyezzen meg a modul nevével (pl. ha 'nat' a tábla neve, akkor a modul legyen 'iptable_nat.o'), hogy az autómatikus betöltés mûködjön.

table

Ez egy teljesen kitöltött 'struct ipt_replace', ahogy az userspace-bõl a tábla kicserélésére használják. A 'counters' mutatót NULL-ra kell állítani. Az adatterültet '__initdata'-nak lehet deklarálni, s így betöltés után eldobható.

valid_hooks

Ez egy bitmaszk az IPv4 netfilter hook-okról, ahol a csomag belelép: a bejegyzés helyességének ellenõrzésére használható, valamint az ipt_match és ipt_target 'checkentry()' funkciójának lehetséges hookjainak származtatásához.

lock

Ez egy írható/olvasható zár(lock) az egész táblára; RW_LOCK_UNLOCKED-re kell beállítani.

private

Az ip_tables kód belsõ használatára fenntartott.

4.2.2 Userspace eszközök

Nos, megírtad a szép, csillogó kernelmodulodat, s most használni szeretnéd a funkcióit userspace-bõl. Ahelyett, hogy magát az `iptables`-t kellene módosítani minden bõvítéshez, egy késõ 90-es évek beli technológiát használok: furbikat. Bocsánat, shared library-kre gondoltam.

Új táblák rendszerint nem igényelnek bõvítést az `iptables`-ben: a felhasználó használhatja a '-t' opciót az új tábla használatához.

A könyvtárban jó, ha van egy '_init()' funkció, ami automatikusan meghívódik betöltéskor: egy megfelelõje a kernel modulok 'init_module()' funkciójának. Ez meghívhatja a 'register_match()' vagy a 'register_target()' függvényeket, attól függõen, hogy a könyvtár match-et vagy target-et tartalmaz.

A könyvtárat el kell készítened: ez használható a struktúrák beállítására, vagy további opciók nyújtására. Jelenleg ragaszkodunk a shared library-hoz, még akkor is, ha nem csinál semmit, az olyan problémák csökkentésére, amik a könyvtár hiányára hivatkoznak.

Van pár hasznos funkció az 'iptables.h' fejlécfileban:

check_inverse()

azt ellenõrzi, hogy egy argument '!'-e, ha az, akkor beállítja az 'invert' flaget, ha még nem volt beállítva. Ha igazat ad vissza, az optind-t növelned kell, ahogy a példában is látszik.

string_to_number()

egy karaktersort számmá konvertál az adott tartományban, -1-et ad vissza ha hibás, vagy a határon túli a karaktersor.

exit_error()

lehetõleg ezt hívd meg, ha hibát találtál. Rendszerint az elsõ paramétere 'PARAMETER_PROBLEM', ami azt jelenti, hogy a felhasználó hibás parancssort adott be.

Új match funkciók A könyvtár _init() funkciója egy 'register_match()' hívást tartalmaz egy statikus 'struct iptables_match' struktúra-pointerrel, aminek a következõ mezõi vannak:

next

A match-ek láncolt listájának kezelésére használják (pl. a szabályok listája). alapértelmezésben NULL értékre kell állítani.

name

A match funkció neve. Meg kell egyeznie a könyvtár nevével (pl. 'tcp' - 'libipt_tcp.so').

version

Rendszerint a IPTABLES_VERSION makróra van állítva: azt biztosítja, hogy az `iptables` nem olvas be hibás könyvtárat.

size

A match adat mérete ehhez a match-hez; lehetõleg használd az IPT_ALIGN() makrót a helyes igazításhoz.

userspacesize

Néhány matchben a kernel módosít pár mezõt. Ez azt jelenti, hogy egy egyszerû 'memcmp()' nem elég két szabály összehasonlítására (a delete-matching-rule funkcióhoz elengedhetetlen). Ha ez a helyzet, akkor az állandó mezõket a struktúra elején kell elhelyezni, s a nem módosuló rész méretét kell itt megadni. Rendszerint ez megegyezik a 'size' mezõvel.

help

Az a funkció, ami az opciók használatát írja ki.

init

Az extra helyek beállítására használható (ha van) az ipt_entry_match struktúrában, s állíthatja az nfcache biteket. Ha valami olyant vizsgálsz, ami nem kifejezhetõ a 'linux/include/netfilter_ipv4.h'-val, egyszerûen OR-old meg az NFC_UNKNOWN bitet. A 'parse()' elõtt fog meghívódni.

parse

Ez akkor kerül meghívásra, ha egy nem ismert funkciót talál a parancssorban: nem nullát ad vissza, ha valóban a könyvtáradhoz tartozik. 'invert' értéke igaz, ha már talált '!'-t. A 'flags' kizárólag a match könyvtár által használt, rendszerint bitmaszk tárolására használják, ami a beállított kapcsolókat reprezentálja. Meg kell gyõzõdnöd arról, hogy az nfcache mezõt állítod. Szükséged lehet az 'ipt_entry_match' méretének növelésére áthelyezéssel, de a méretet az IPT_ALIGN makróval kell megadnod!

final_check

A parancssor értelmezése után hívódik meg, és a 'flags' értékét vizsgálja. Lehetõséget ad összeférhetetlenség-vizsgálatra, s az 'exit_error()' hívással jelezheted a problémát.

print

A lánclistázó kód használja a (standard kimenetre) való funkciókiíráskor. A numeric flag be van állítva, ha a felhasználó megadta a '-n' kapcsolót.

save

A parse ellentettje: az 'iptables-save' használja a szabályt létrehozó opciók visszaállításához.

extra_opts

Ez egy NULL-lezárt listája az extra funkcióknak, amiket a könyvtárad nyújt. Az eddigi opciókkal összedolgozásra kerül, s úgy kerül a getopt_long-hoz (nézd meg a mauálját). A getopt_long visszatérési kódja az elsõ argument lesz ('c') a 'parse()' funkcióhoz.

Van még pár extra funkció a struktúra végén, de azokat az `iptables` használja: nem kell beállítanod õket!

Új target-ek A könyvtárak _init() funkciója kezeli a 'register_target()'-t, s a statikus 'struct iptables_target' struktúráját, aminek a felépítése hasonló az iptables_match struktúrájához.

4.2.3 'libiptc' használata

`libiptc` a táblakezelõ könyvtár, az iptables szabályok listázására és módosítására tervezett könyvtár. Jelenleg csak az iptables program használja, könnyû egyéb programok implementálása. Root jogokkal kell rendelkezned a használatához.

A kernel táblák magukban csak egyszerû szabálytáblázatok, valamint belépési pontokat tartalmazó halmazok. A láncok elnevezése ("INPUT", stb.) csak egy, a könyvtárak által szolgáltatott leképezés. Felhasználó által definiált láncok neveit a chain fejléce elé beillesztett hiba-bejegyzés tartalmazza a target extra adat-területén (a beépített chain pozíciók a három tábla belépési pontjainál vannak definiálva.

A következõ standart target-ek támogatottak: ACCEPT, DROP, QUEUE (amik NF_ACCEPT, NF_DROP és NF_QUEUE -ra vannak fordítva), RETURN (ami a speciális IPT_RETURN-nek felel meg, s az ip_tables kezeli), valamint a JUMP (ami egy eltolási értékre (offset) fordul le).

Az 'iptc_init()' meghívásakor a tábla - beleértve a számlálókat - kerül beolvasásra. A tábla az 'iptc_insert_entry()', 'iptc_replace_entry()', 'iptc_append_entry()', 'iptc_delete_entry()', 'iptc_delete_num_entry()', 'iptc_flush_entries()', 'iptc_zero_entries()', 'iptc_create_chain()', 'iptc_delete_chain()' és 'iptc_set_policy()' függvényekkel módosítható.

A változtatások nem kerülnek visszaírásra, csak az 'iptc_commit()' meghívása után. Ez azt jelenti, hogy két felhasználó is módosíthatja ugyanazt a táblát, s így versenyhelyzetet kialakítva; szükség lenne lock-olásra, de még nem készült el.

A számlálókra nem él a versenyhelyzet: a visszaíráskor az érték korrigálódik az eltelt idõ alatti változással.

Számos segítõ funkciót implementáltak:

iptc_first_chain()
: Visszaadja az elsõ lánc nevét a táblában.

iptc_next_chain()
: A következõ chain nevét adja, NULL-al jelzi a lista végét.

iptc_builtin()
: Igazat ad vissza, ha az adott láncnév egy beépített chain neve.

iptc_first_rule()
: Egy mutatót ad vissza az elsõ szabályra az adott láncon belül. NULL jelzi az üres láncot.

iptc_next_rule()
: A következõ szabályt adja az adott chain-ben. NULL jelenti a lánc végét.

iptc_get_target()
: Az adott szabály target-jét adja vissza. Ha ez egy kiterjesztett target, akkor a nevét adja vissza. Ha egy másik chain-re ugrás, akkor az új chain nevét. Ha egy döntés (pl. DROP), akkor azt tartalmazza, s ha nincs target (accounting szabály), akkor üres sort.

  Figyelem! Ezt a funkciót célszerû használni az ipt_entry struktúra 'verdict' mezeje helyett, mert bõvebb információ szerezhetõ belõle.

iptc_get_policy()
: A beépített lánc policy-ét kérdezi le, valamint a 'counters' argumentumát kitölti a szabály találati paramétereivel.

iptc_strerror()
: Az iptc könyvtár hibajelzéseinek jelentéssel való kibõvítését adja. A hibával visszatérõ függvény beállítja az errno értékét, s ez a funkció kiírja a hibakódhoz tartozó üzenetet.

4.3 A NAT megértése

Üdvözöllek a kernel címfordítási részében! Felhívnám arra a figyelmedet, hogy az itt nyújtott infrastruktúra inkább a teljességre, mint a nyers hatásfokra helyezte a hangsúlyt, és a jövõ trükkjei jelentõsen növelhetik a teljesítõképességet. Jelenleg boldog vagyok, hogy mûködik.

A NAT fel van bontva kapcsolat-követési (connection tracking) (ez nem módosítja a csomagokat) és magára a fordítási kódra. Connection tracking az iptables modulokban való felhasználhatóságra lett tervezve, így olyan állapotok szövevényes rendszer alapján dönthet, amelyek a NAT-ot nem érdeklik.

4.3.1 Connection Tracking

A Connection tracking hookjai nagy prioritási szinttel az NF_IP_LOCAL_OUT és az NF_IP_PRE_ROUTING hookokban találhatók, így a rendszerbe való megérkezésük elõtt vizsgálja a csomagokat.

Az nfct mezõ az skb struktúrában egy pointer az ip_conntrack struktúrába, az infos[] tömb egy elemére. Ennélfogva meg tudjuk mondani az skb állapotát az általa mutatott elemen keresztül: ez a mutató tárolja az állapot-struktúrát és az skb - állapot közötti kapcsolatot is.

A legjobb eljárás az ‘nfct’ mezõ kicsomagolására az ‘ip_conntrack_get()’ használata, ami NULL-al jelzi, ha nincs beállítva, vagy visszaadja a kapcsolat-mutatót, valamint kitölti a ctinfo-t, ami leírja a csomag és a kapcsolat viszonyát. Ennek a változónak számos értéke lehet:

IP_CT_ESTABLISHED
: A csomag egy már létrejött kapcsolathoz tartozik, az eredei irányban.

IP_CT_RELATED
: A csomag kapcsolatban van egy connection-nel, és az eredi irányba halad.

IP_CT_NEW
: A csomag egy új kapcsolatot próbál kialakítani (természetesen az eredeti irányban).

IP_CT_ESTABLISHED + IP_CT_IS_REPLY
: A csomag egy már létrejött kapcsolathoz tartozik, az ellenkezõ irányban. (Válasz)

IP_CT_RELATED + IP_CT_IS_REPLY
: A csomag kapcsolatban van egy connection-nel, és az ellenkezõ irányba halad. (Válasz)

Így a válasz csomagra egyszerûen ellenõrizhetünk egy >=IP_CT_IS_REPLY teszttel.

4.4 Connection Tracking/NAT kibõvítése

Ezek a programvázak tetszõleges protokollhoz és leképezési módhoz való illesztésre lettek tervezve. Néhány ilyen leképezés nagyon speciális is lehet, pl. terheléselosztás vagy tartalékolás.

Belsõleg a connection tracking párokba alakítja a csomagot, ami az érdekes részét mutatja a csomagnak, mielõtt függõségekre vagy egyezõ szabályokra keresne. Ennek a leírónak van egy módosítható és egy nem módosítható része is; "src" és "dst" névvel, ahogy ez a Source NAT-ban az elsõ csomagnál látható (lehetõleg kell lennie egy válasz csomagnak is a Destination NAt világában). Ez a leíró minden csomagra az adott folyamaton belül az adott irány mellett azonos.

Például a TCP csomag leírójában a módosítható rész tartalma: forráscím é sport, a nem módosíthatóé: célcím és port. A két résznek nem feltétlenül kell azonos szerkezetûnek lennie: pl. egy ICMP csomagnál a forráscím és az ICMP id a módosítható; míg a célcím és az ICMP típus és kód a nem módosítható rész.

Minden leírónak (tuple) van egy inverze, ami a válsz csomagnak a leírója. Például egy ICMP ping csomagnak (icmp id 12345, from 192.168.1.1 to 1.2.3.4) az ellentettje a ping-reply csomag (icmp id 12345, from 1.2.3.4 to 192.168.1.1).

Ezek a párok, amiket a ‘struct ip_conntrack_tuple’ testesít meg, széles körben használtak. Valójában azzal a hook-kal, ahonnan a csomag jött (aminek a várható módosításra van hatása) és a beérkezési device adataival a csomaggal kapcsolatos összes információnkat tartalmazza.

A legtöbb leyrót a ‘struct ip_conntrack_tuple_hash’ struktúrában találjuk meg, ami egy két-irányba láncolt lista kezeléséhez szükséges kiegészítést és egy segédmutatót (a leíró melyik kapcsolathoz tartozik) rendel még mellé.

A kapcsolatot a ‘struct ip_conntrack’-al reprezentáljuk: két ‘struct ip_conntrack_tuple_hash’ mezõje van: egyik az eredeti irányba mutat (tuplehash[IP_CT_DIR_ORIGINAL]), a másik pedig a válasz-csomagokra (tuplehash[IP_CT_DIR_REPLY]).

Az elsõ dolog, amit a NAT kód végrehajt az, hogy megnézi, hogy a connection tracking kód kicsomagolta-e a leíróját, s egy meglevõ kapcsolathoz tartozónak találta-e az skbuff nfct mezõjének vizsgálatával; ez megmondja, hogy új kapcsolathoz tartozik-e vagy nem, melyik irányba halad. Az utóbbi esetben a szükséges módosítás már meghatározásra került a kapcsolathoz.

Ha egy új kapcsolat kezdete, akkor megpróbál szabályt keresni a leíróhoz, az alap iptables keresési rendszerrel a 'nat' táblában. Ha egy szabály egyezik rá, akkor felhasználja a mindkét irányba szükséges módosítások meghatározásához; a connection-tracking kód jelezheti, hogy várhatóan a másik irányba is módosítani kell. Ezután a fentiek alapján módosításra kerül.

Ha nem talál szabályt, akkor egy 'null' kötést készít: ez rendszerint nem kezeli a csomagot, de létezik, hogy biztosítsuk, hogy nem lapolunk be egy új kapcsolatot a régi fölé. Néha azonban nem sikerül elkészíteni a null-kötést, mert egy már meglevõ kapcsolattal felülírtuk. Ebben az esetben a protokollonkénti módosítás megpróbálhatja újra felvenni, annak ellenére, hogy ez egy null-kötés.

4.4.1 Standard NAT Target-ek

NAT targetek hasonlítanak a hagyományos iptales kiegészítésekre, kivéve, hogy a 'nat' táblában kerülnek csak felhasználásra. Az SNAT és DAT targetek mindegyike kap egy 'struct ip_nat_multi_range'-t az extra adataihoz; a kötések elkészítéséhez használható címterületet adja meg. Egy tartományelem, 'struct ip_nat_range' tartalmaz egy minimum és egy maximum IP címet és egy protokoll-függõ maximum és minimum értéket (pl. TCP portok). Szintén található hely a flageknek, amik megmondhatják, hogy legyen az IP cím beírva (néha csak a protokoll-specifikus részre van szükségünk a leíróból), vagy azt, hogy a protokoll-specifikus része a tartománynak értelmezetõ.

Egy több elembõl álló tartomány egy tömb ezekbõl a 'struct ip_nat_range' elemekbõl. Ez azt jelenti, hogy a tartomány lehet: "1.1.1.1-1.1.1.2 ports 50-55 AND 1.1.1.3 port 80". Minden tartományelem hozzáadásra kerül a tartományhoz (egy union, azoknak, akik ezt szeretik).

4.4.2 Új protokollok

A Kernelen belül Új protokoll implementálása a leíró(tuple) a változtatható és a nem változtatható részeinek meghatározásával kezdõdik. A leíróban mindenre megvan a lehetõséged, hogy egy folyamot egyértelmûen azonosíthass. A változtatható része a leírónak az a rész, amivel a NAT-ot elvégezheted: a TCP-hez ez a forrásport, az ICMP-nek az icmp ID; valami, amit a folyam azonosítására használhatsz. A nem módosítható rész a csomag maradéka, ami egyértelmûen meghatározza a hálózati folyamot, de nem szeretnél játszani vele (pl. a TCP célport, ICMP típus).

Ha egyszer eldöntöttük, meg lehet írni a bõvítést a connection-tracking kódhoz a megfelelõ könyvtárban, és elkezdheted az 'ip_conntrack_protocol' struktúra kitöltését, ami az 'ip_conntrack_register_protocol()' híváshoz szükséges.

A 'struct ip_conntrack_protocol' szerkezete:

list

Legyen '{ NULL, NULL }'; a listakezeléshez használjuk.

proto

A protokoll-sorszáma ('/etc/protocols').

name

A protokoll neve. Ezt a nevet fogja látni a felhasználó; rendszerint az a legjobb, ha megegyezik az '/etc/protocols'-ban találhatóval.

pkt_to_tuple

A funkció, ami kitölti a protokoll-specifikus részét a leírónak az adott csomagra vonatkozóan. A ‘datah’ pointer a fejléc kezdetére mutat, és a datalen a csomag méretét tartalmazza. Ha a csomag nem elég hosszú ahhoz, hogy benne legyen a teljes fejléc, 0-t ad vissza, noha a datalen mindig legalább 8 byte lesz (a keretrendszer garantálja).

invert_tuple

Ez a funkció egyszerûen megváltoztatja a protokoll-függõ részét a leírónak a válasz irányból érkezõ csomagnak megfelelõen.

print_tuple

Ez a funkció jeleníti meg a protokoll-függõ részét a leírónak; rendszerint a megadott bufferbe kerül beírásra. A használt karakterek száma a visszatérési érték. A /proc-ban az állapotok megjelenítésére használjuk.

print_conntrack

A conntrack struktúra privát részének megjelenítésére használjuk (ha van ilyen), valamint szintén megjelenik a /proc-bejegyzésben.

packet

Akkor kerül meghívásra, ha egy csomag a kapcsolathoz tartozónak tûnik. Mutatókat kapcs a conntrack struktúrára, az IP fejlécre, a méretre és a ctinfo-ra. Egy döntést kell visszaadnod a csomagra (rendszerint NF_ACCEPT), vagy -1-et, ha a csomag nem tartozik a kapcsolathoz. Törölni is tudod a kapcsolatot, de használnod kell a következõ beszúrást a versenyhelyzet elkerülésére (ip_conntrack_proto_icmp.c):

```
if (del_timer(&ct->timeout))
        ct->timeout.function((unsigned long)ct);
```

new

Ha a csomag egy új kapcsolat indít; nincs ctinfo paramétere, mert az elsõ csomag ctinfo-ja definíció szerint IP_CT_NEW. 0-t ad vissza, ha nem sikerült elkészítenie a kapcsolatot, vagy timeout volt.

Miután megírtuk és leteszteltük, hogy remekül tudjuk követni a protokollunk, itt az idõ, hogy megtanítsuk a NAT-ot, hogy hogyan kell átfordítania azt. Ez egy új modul írását jelenti; egy kiterjesztést a NAT kódhoz, valamint az ‘ip_nat_protocol’ struktúra feltöltését az ‘ip_nat_protocol_register()’ funkcióhoz.

list

‘{ NULL, NULL }’

name

A protokoll neve. Ezt a nevet fogja látni a felhasználó; rendszerint az a legjobb, ha megegyezik az ‘/etc/protocols’-ban találhatóval. (Fõleg az userspace-beli autómatikus betöltés miatt.)

protonum

A protokoll-sorszáma (‘/etc/protocols’).

manip_pkt

Ez a másik fele a connection tracking pkt_to_tuple funkciójának: gondold azt, hogy egy "tuple_to_pkt". Azért van néhány különbség: kapsz egy mutatót az IP fejléc kezdetére és a teljes csomagméretre. Ez azért van, mert néhány protokollhoz (UDP, TCP) szükség van a fejlécre. Továbbá az ip_nat_tuple_manip mezejét a leírónak (pl. az "src" mezõ) a teljes leíró helyett, és a módosítás típusát.

in_range

Megmondja, hogy a módosítható része az adott leírónak a megadott tartományon belül van-e. A funkció egy kicsit trükkös: megadjuk a leíróra alkalmazott módosítás típusát, ami megmondja nekünk, hogyan kell értelmezni a tartományt (a forrás vagy a céltartományra céloztunk...).

Ezzel a funkcióval ellenõrizhetjük, hogy egy meglevõ megfeleltetés helyes tartományba rakott-e minket, valamint azt is hogy szükséges-e a módosítás.

unique_tuple

Ez a funkció a NAT központja(magja): egy leírót és egy tartományt kap, s itt kerül a protokoll-függõ része a leírónak a tartományon belülivé, s válik egyedivé. Ha nem találsz nemhasznált leírót a tartományban, akkor 0-t kell visszaadnod. Szintén kapunk egy mutatót a conntrack struktúrára, ami az ip_nat_used_tuple()-hoz szükséges.

A szokásos hozzáállás az, hogy egyszerûen folyamatosan léptetjük végig a protokoll-függõ részét a leírónak a tartományon, elvégezve az 'ip_nat_used_tuple()' ellenõrzést rajta, amíg hamisat ad vissza.

A null-megfeleltetés már tesztelve volt: vagy a tartományon kívül van, vagy már foglalt.

Ha az IP_NAT_RANGE_PROTO_SPECIFIED nem lett beállítva, az azt jelenti, hogy a felhasználó NAT-ot csinál, s nem NAPT-ot: valami érzékeny dolgot csinál a tartománnyal. Ha nincs szükség megfeleltetésre (például a TCP-n belül a cél-megfeleltetésnek nem kell megváltoztatnia a TCP portot, kivéve, ha utasítják rá), 0-t adjon vissza.

print

Egy karakter-buffert, egy megegyezõ leírót és egy maszkot vár, s kiírja a protokoll-függõ részeket, s visszaadja a felhasznált bufferméretet.

print_range

Egy karakter-buffert és egy tartományt vár, s kiírja a protokoll-függõ részét a tartománynak, s visszaadja a felhasznált bufferméretet. Nem kerül meghívásra, ha az IP_NAT_RANGE_PROTO_SPECIFIED jelzõbit nem volt beállítva a tartományhoz.

4.4.3 Új NAT Target-ek

Ez egy valóban érdekes rész. Lehetõséged van új NAT targetek írására, amelyek új leképezést valósítanak meg. Két extra targetet alapból biztosít a csomag: MASQUERADE és REDIRECT. Ezek egyszerû minták, s remekül bemutatják az új NAT targetek életképességét és erejét.

Ezek a többi iptables atrgethez hasonlóan vannak megírva, de belül megszakítják a kapcsolatot és meghívják az 'ip_nat_setup_info()'-t.

4.4.4 Protokoll segítõk (Helperek)

A kapcsolatkövetés protokoll helperei lehetõvé teszik a követõ kódnak a több kapcsolatot tartalmazó protokollok megértését (pl. FTP), és megjelölik a leszármazott kapcsolatokat, a szülõ alá rendelik azokat, rendszerint az adatkapcsolatból kiolvasott cím alapján.

A NAT protokoll helperei két dolgot végeznek: lehetõvé teszik, hogy a NAT kód módosítsa az adatfolyamot a benne található címek átírásával, valamint elvégezhesse a NAT-ot a kapcsolódó folyamon amikor az beérkezik (az eredeti kapcsolat alapján).

4.4.5 Connection Tracking Helper Modules

Leírás A connection tracking module kötelessége, hogy meghatározza, melyik csomagok tartoznak egy már felépült kapcsolathoz. A modul a következõket teszi:

- Jelzi a netfilternek, melyik csomagok fontosak a modulunknak (a legtöbb helper egy megadott porton üzemel).
- Bejegyzi a funkcióját a netfilterbe. A funkció meghívásra kerül minden egyes csomagra, ami illeszkedik a feltételre.
- Egy 'ip_conntrack_expect_related()' függvényt meghívhat, hogy jelezze a netfilternek, hogy egy kapcsolatra vár.

Elérhetõ struktúrák és függvények A kernel modulod init funkciójának meg kell hívnia a 'ip_conntrack_helper_register()' függvényt egy pointerrel a 'struct ip_conntrack_helper'-ra. A struktúra a következõ mezõkkel rendelkezik:

list
: A láncolt lista feje. A netfiletr belsõleg kezeli. Alapbeállításban legyen '{ NULL, NULL }'.

tuple
: Egy 'struct ip_conntrack_tuple', ami megadja, hogy milyen csomagok érdeklik a helperünket.

mask
: Mégegy 'struct ip_conntrack_tuple'. A mask megadja, hogy a `tuple`-nek melyik bitjei valósak.

help
: A függvény, amit a netfilter meghívhat minden csomagra, ami illeszkedik a tuple+mask-ra.

Egy minta conntrack helper modul

```
#define FOO_PORT        111

static int foo_help(const struct iphdr *iph, size_t len,
                struct ip_conntrack *ct,
                enum ip_conntrack_info ctinfo)
{
        /* analyze the data passed on this connection and
           decide how related packets will look like */

        if (there_will_be_new_packets_related_to_this_connection)
        {
                t = new_tuple_specifying_related_packets;
                ip_conntrack_expect_related(ct, &t);

                /* save information important for NAT in
                        ct->help.ct_foo_info;   */

        }
        return NF_ACCEPT;
}
```

```
static struct ip_conntrack_helper foo;

static int __init init(void)
{
        memset(&foo, 0, sizeof(struct ip_conntrack_helper);

        /* we are interested in all TCP packets with destport 111 */
        foo.tuple.dst.protonum = IPPROTO_TCP;
        foo.tuple.dst.u.tcp.port = htons(FOO_PORT);
        foo.mask.dst.protonum = 0xFFFF;
        foo.mask.dst.u.tcp.port = 0xFFFF;
        foo.help = foo_help;

        return ip_conntrack_helper_register(&foo);
}

static void __exit fini(void)
{
        ip_conntrack_helper_unregister(&foo);
}
```

4.4.6 NAT helper modulok

Leírás A NAT-helper modulok alkalmazásfüggõ NAT-kezelést tesznek lehetõvé. Rendszerint az adatok röptében történõ elemzését jelentik: gondolj csak a PORT parancsra az FTP-ben, ahol a kliens megmondja a szervernek, hogy melyik IP/port-párhoz kell kapcsolódnia. Így az FTP-helper modulnak ki kell cserélnie az IP/port-ot a PORT parancs után az FTP parancs-csatornában.

Ha elbántunk a TCP-vel, akkor a dolgok kissé összetettebbé válnak. Az ok a lehetséges csomagméret-változás (FTP példa: a PORT utáni IP/port-párt reprezentáló string hossza megváltozik). Ha megváltoztatjuk a csomagméretet, akkor egy syn/ack eltéréshez jutunk a NAT két oldala között. (Ez azt jelenti, hogy ha kiegészítettük a csomagot 4 oktettel, akkor ezután minden csomag TCP sorszámához hozzá kell adnunk).

A kapcsolódó csomagok speciális kezelésére is szükség van. Az FTP példához visszatérve: az adatkapcsolat minden egyes csomagját NAT-olni kell a kliens által a parancscsatornán belül a PORT parancsban megadott IP/port-párra, a sima táblázatos keresés helyett.

- callback a csomagra, ami az adott csatornát elindította (foo_help)
- callback az összes kapcsolódó csomagra (foo_nat_expected)

Elérhetõ struktúrák és függvények A nat helper modulod 'init()' funkciójának meg kell hívnia a 'ip_nat_helper_register()' függvényt egy pointerrel a 'struct ip_nat_helper'-ra. A struktúra a következõ mezõkkel rendelkezik:

list

A láncolt lista feje. A netfiletr belsõleg kezeli. Alapbeállításban legyen '{ NULL, NULL }'.

tuple

Egy 'struct ip_conntrack_tuple', ami megadja, hogy milyen csomagok érdeklik a NAT helperünket.

mask

Mégegy 'struct ip_conntrack_tuple'. A mask megadja, hogy a `tuple`-nek melyik bitjei valósak.

help

A függvény, amit a netfilter meghívhat minden csomagra, ami illeszkedik a tuple+mask-ra.

name

Egy egyedi név, ami a modulunkat azonosítja.

Ez pontosan megegyezik a connection tracking helper modul írásával.

Jelezni tudod, hogy a modulod képes minden várható kapcsolat NAT-olását elvégezni (valószínûleg egy connection tracking modullal került beillesztésre). Ezt a 'ip_nat_expect_register()' függvénnyel teheted meg, ami egy 'struct ip_nat_expect' struktúrát vár. A struktúra a következõ mezõkkel rendelkezik:

list

A láncolt lista feje. A netfiletr belsõleg kezeli. Alapbeállításban legyen '{ NULL, NULL }'.

expect

a funkció, ami elvégzi a NAT-olást a várt csomagokra. TRUE-val jelzi, hogy lekezelte a csomagot, különben a következõ bejegyzett funkció kerül meghívásra. Ha TRUE-t ad vissza, akkor ki kell töltenie a döntés mezõt!

Egy minta NAT helper modul

```
#define FOO_PORT        111

static int foo_nat_expected(struct sk_buff **pksb,
                        unsigned int hooknum,
                        struct ip_conntrack *ct,
                        struct ip_nat_info *info,
                        struct ip_conntrack *master,
                        struct ip_nat_info *masterinfo,
                        unsigned int *verdict)

/* called whenever a related packet (as specified in the connection tracking
   module) arrives
   params:      pksb    packet buffer
                hooknum HOOK the call comes from (POST_ROUTING, PRE_ROUTING)
                ct      information about this (the related) connection
                info    &ct->nat.info
                master  information about the master connection
                masterinfo &master->nat.info
                verdict what to do with the packet if we return 1.
{
        /* Check that this was from foo_expect, not ftp_expect, etc */
        /* Then just change ip/port of the packet to the masqueraded
           values (read from master->tuplehash), to map it the same way,
           call ip_nat_setup_info, set *verdict, return 1. */

}

static int foo_help(struct ip_conntrack *ct,
                struct ip_nat_info *info,
                enum ip_conntrack_info ctinfo,
                unsigned int hooknum,
```

```
                struct sk_buff  **pksb)
/* called for the packet causing related packets
   params:      ct      information about tracked connection
                info    (STATE: related, new, established, ... )
                hooknum HOOK the call comes from (POST_ROUTING, PRE_ROUTING)
                pksb    packet buffer
*/
{

        /* extract information about future related packets (you can
           share information with the connection tracking's foo_help).
           Exchange address/port with masqueraded values, insert tuple
           about related packets */
}

static struct ip_nat_expect foo_expect = {
        { NULL, NULL },
        foo_nat_expected };

static struct ip_nat_helper hlpr;

static int __init(void)
{
        int ret;

        if ((ret = ip_nat_expect_register(&foo_expect)) == 0) {
                memset(&hlpr, 0, sizeof(struct ip_nat_helper));
                hlpr.list = { NULL, NULL };
                hlpr.tuple.dst.protonum = IPPROTO_TCP;
                hlpr.tuple.dst.u.tcp.port = htons(FOO_PORT);
                hlpr.mask.dst.protonum = 0xFFFF;
                hlpr.mask.dst.u.tcp.port = 0xFFFF;
                hlpr.help = foo_help;

                ret = ip_nat_helper_register(hlpr);
                if (ret != 0)
                        ip_nat_expect_unregister(&foo_expect);
        }
        return ret;
}

static void __exit(void)
{
        ip_nat_expect_unregister(&foo_expect);
        ip_nat_helper_unregister(&hlpr);
}
```

4.5 Értsük meg a Netfilter-t!

Netfilter nagyon egyszerû, és elég pontosan le van írva az elõzõ fejezetekben. Néha azonban szükséges a NAT és az ip_tables által nyújtott szolgáltatások alá menni, vagy esetleg teljesen ki is cserélhetõek.

Egy fontos kitétel a netfilterhez (nos, a jövõben) az elrejtés. Minden skb-nek van egy 'nfcache' mezõje: egy bitmask, ami megmutatja, hogy milyen mezõket kell a fejlécbõl megvizsgálni, valamint, hogy megváltozott-e a csomag, vagy nem. A terv az, hogy minden netfilter hook VAGY-al állítja be a fontos bitjeit, így lehetõvé válik a későbbiekben egy olyan rendszer kialakítása, ami okos annyira, hogy el tudja dönteni, hogy a csomagot el kell-e küldeni a netfilterben, vagy teljesen kihagyható.

A legfontosabb bitek az NFC_ALTERED, ami azt jelenti, hogy a csomag megváltozott-e (ez már használva van az IPv4-es NF_IP_LOCAL_OUT-ban a megváltoztatott csomagok routolására), és NFC_UNKNOWN, ami azt mutatja, hogy a caching-et nem lehet elvégezni, mert néhány olyan jellemzõ került megvizsgálásra, amit nem lehet kifejezni. Ha kétségeid vannak, akkor egyszerûen állítsd be a NFC_UNKNOWN flaget az skb nfcache mezõjében, a hook-odon belül.

4.6 Új netfilter modulok írása

4.6.1 Csatlakoztatás a Netfilter hook-okra

Kernelen belül a csomagok fogadásához egyszerûen tudsz írni egy modult, ami bejegyez egy "netfilter hook"-ot. Ez alapjában az érdeklõdés kifejezésének a módja; az aktuális pontok protokoll-specifikusak lehetnek, és külön headerekben találhatók, mint pl. a "netfilter_ipv4.h".

Netfilter hook bejegyzéséhez és eltávolításához használd az 'nf_register_hook' és 'nf_unregister_hook' függvényeket. Ezek mindegyike vár egy pointert a 'struct nf_hook_ops'-ra, ami a következõképpen épül fel:

list

'{ NULL, NULL }', láncolt listába illesztéshez használt.

hook

A funkció akkor kerül meghívásra, ha a csomag eléri a hook-ot. A lehetséges visszatérési értékek: NF_ACCEPT, NF_DROP vagy NF_QUEUE. NF_ACCEPT: a következõ, erre a pontra csatlakozó hook kerül meghívásra. NF_DROP: a csomag eldobásra kerül. NF_QUEUE: sorbaállításra kerül. Egy mutatót kapsz egy skb pointerre, szóval teljesen le tudod cserélni az skb-t, ha akarod.

flush

Aktuálisan nem hsznált: a cache ürítésekor a találatok kezeléséhez készült. Talán sose lesz implementálva: állítsd NULL-ra!

pf

A protokollcsalád, pl. 'PF_INET' IPv4-hez.

hooknum

A hook száma, amiben érdekelt vagy. Pl: 'NF_IP_LOCAL_OUT'.

4.6.2 Queued csomagok feldolgozása

Ezt a felületet az ip_queue használja; be tudod jegyezni, hogy egy adott protokollhoz tartozó csomagokat lekezelje. Hasonló a felépítése, mintha egy hook-hoz regisztrálnád magad, kivéve, hogy lehetõséged van a scomag feldolgozásának megállítására, valamint csak azokat a csomagokat látod, amelyekre a hook 'NF_QUEUE'-t válaszolt.

A két, a regisztrációhoz használható függvény: 'nf_register_queue_handler()' és 'nf_unregister_queue_handler()'. A beregisztrált függvény 'void *' pointerrel kerül lekezelésre.

Ha senki sem jelentkezett az adott protokoll lekezelésére, akkor az NF_QUEUE megegyezik az NF_DROP visszatérési értékkel.

Amennyiben bejegyezted az érdeklõdésed a sorbaállított csomagokra, elkezdõdik a sorbaállítás. Bármit megtehetsz velük, csak meg kell hívnod az 'nf_reinject()'-et miután befejezted a módosítást (ne csak egyszerûen kfree_skb()-d õket). Amikor visszaküldesz egy skb-t, te kezeled az skb-t, a 'struct nf_info'-t, ami a kezelõdet jelenti, valamint a döntést: NF_DROP eredményezi a csomag eldobását, NF_ACCEPT jelenti a hookokban való továbbküldést, NF_QUEUE jelenti az ismételt sorbaállítást, NF_REPEAT hatására pedig ismét bekerül a hook-ba (figyelt a végtelen ciklusokra!).

A 'struct nf_info'-ban információt kapsz a csomagról, mint pl. a hozzá tartozó intarface-ek, a hook, amin fennakadt, stb.

4.6.3 Parancsok értelmezése az userspace-bõl

Gyakori a netfilter részekben, hogy kommunikálni szeretnének az userspace-el. Az erre használható módszer a setsockopt mechanizmus. ehhez azonban módosítani kell minden protokollt, hogy meghívja a nf_setsockopt()-ot azokra a setsockopt számokra, amiket nem ismer (valamint a nf_getsockopt()-ot a getsockopt-hoz), s nemcsak az IPv4, IPv6 és DECnet protokollokban tegye meg ezt.

Egy nem szokványos módszerként bejegyzünk egy 'struct nf_sockopt_ops'-ot az nf_register_sockopt() hívással. A struktúra mezõi:

list

A listába való beillesztéshez használt: '{ NULL, NULL }'.

pf

A kezelt protokollcsalád, pl: PF_INET.

set_optmin

és

set_optmax

Ezek megadják a (kizárólagos) tartományát a kezelt setsockopt számoknak. Ezentúl 0-val jelezheted, hogy nincs setsockopt opciód.

set

Ez a funkció kerül meghívásra, ha a felhasználó meghívja az egyik setsockopt opciódat. Célszerû ellenõrizni, hogy megvan-e a NET_ADMIN capability-e.

get_optmin

és

get_optmax

Ezek megadják a (kizárólagos) tartományát a kezelt getsockopt számoknak. Ezentúl 0-val jelezheted, hogy nincs getsockopt opciód.

set

Ez a funkció kerül meghívásra, ha a felhasználó meghívja az egyik getsockopt opciódat. Célszerû ellenõrizni, hogy megvan-e a NET_ADMIN capability-e.

Az utolsó két mezõt belsõleg használjuk.

4.7 Csomagkezelés az userspace-ben

A libipq könyvtár és a 'ip_queue' modul segítségével majdnem mindent megtehetsz az userspace-ben, amit a kernelen belül. Ez azt jelenti - kisebb sebességcsökkenéssel - a programodat teljes egészében fejlesztheted az userspace-ben. Amíg nem akarsz nagy sávszélességet szûrni, használd inkább ezt a lehetõséget a kernelen belüli csomagkezeléssel szemben.

A netfilter nagyon korai szakaszában kipróbáltam az iptables egy nagyon korai verziójának userspace-be való portolásával. Netfilter kinyitja a kaput, hogy bárki tetszõleges, egészen hatékony modulokat írjon azon a nyelven, amelyiken csak szeretne.

5 2.0 és 2.2 csomagszûrõ moduljainak átfordítása

Nézd meg az ip_fw_compat.c file-t, ahol egy egyszerû layert találhatsz, ami jelentõsen leegyszerûsítheti a portolás folyamatát.

6 Netfilter Hookok a Tunnel-fejlesztõknek

A Tunnel (vagy encapsulation) meghajtók íróinak két egyszerû szabályt kell követniük a 2.4-es kernelnél (a kernelben megtalálható meghajtók írásaokr, mit pl. a net/ipv4/ipip.c):

- El kell engedni az skb->nfct -ot, ha a csomagot felismerhetetlenné (pl. ki/becsomagolás) teszed. Ezt nem kell megtenned, ha egy *új* skb struktúrába csomagolod át, de amennyiben ezt helyben végzet, feltétlenül meg kell tenned!

  Másképpen: a NAT kód a régi kapcsolatkövetési információt fogja használni a csomag megváltoztatásához, aminek nem a várt mûködés lesz a következménye!

- Gyõzõdj meg arról, hogy a becsomagolt csomagok áthaladnak a LOCAL_OUT hookon, a kicsomagolt csomagok pedig a PRE_ROUTING hook-on (a legtöbb tunnel az ip_rcv() függvényt használja, ami megcsinálja a beállításokat)!

  Másképpen: a felhaszáló elvárásával ellentétben nem fogja tudni szûrni a tunneleket!

Az általános eljárás az elsõ problémára a következõhöz hasonló kód beillesztése mielõtt be- vagy kicsomagolnád a csomagot:

```
        /* Tell the netfilter framework that this packet is not the
          same as the one before! */
#ifdef CONFIG_NETFILTER
        nf_conntrack_put(skb->nfct);
        skb->nfct = NULL;
#ifdef CONFIG_NETFILTER_DEBUG
        skb->nf_debug = 0;
#endif
#endif
```

Általában a második pont teljesülése érdekében meg kell találnod azt a pontot, ahol az újonnan elõállított csomag belép az "ip_send()"-be, és a következõvel lecserélni:

```
        /* Send "new" packet from local host */
        NF_HOOK(PF_INET, NF_IP_LOCAL_OUT, skb, NULL, rt->u.dst.dev, ip_send);
```

ezeket a szabályokat betartva az lesz az eredmény, hogy az a szamély, aki csomagszûrési szabályokat szeretne felvenni a tunnel-állomáson, a következõhöz hasonló csomag-útvonalat fog látni a tunnelezett csomag számára:

1. FORWARD hook: hagyományos csomag (eth0 -> tunl0)
2. LOCAL_OUT hook: becsomagolt csomag (->eth1).

És a válasz csomag számára:

1. LOCAL_IN hook: becsomagolt válasz csomag (eth1->)
2. FORWARD hook: becsomagolt csomag (eth1 -> eth0).

7 A Test Suite

A CVS fán belül lakik a tesztrendszer: amit nyújtani tud, az nagyobb biztonság arra vonatkozóan, hogy a változtatásaid nem rontottak el semmit. Triviális tesztek legalább annyira fotosak, mint a trükkösek: az egyszerû tesztek leegyszerûsítik az összetetteket (legalábbis biztos lehetsz benne, hogy az egyszerûbb dolgok mûködnek, mielõtt belekezdesz az összetettekbe).

A tesztek egyszerûek: csak shell-scriptek a testsuite/ könyvtárban, amiknek sikeresen le kell futniuk. A scriptek ABC-sorrendben futnak, így a '01test' a '02test' elõtt hajtódik végre. Jelenleg 5 könyvtár van:

00netfilter/
: Általános Netfilter tesztek

01iptables/
: iptables tesztek

02conntrack/
: connection tracking tesztek

03NAT/
: NAT tesztek

04ipchains-compat/
: ipchains/ipfwadm kompatibilitási tesztek

A testsuite/ könyvtárban található a 'test.sh'. Ez elkészít két dummy interface-t (tap0 és tap1), felkapcsolja a forwardingot, s kitölt minden netfilter modult. Ezután végighalad a fenti könyvtárakon, s elindítja a test.sh scriptjeiket, egészen addig, míg az egyik hibát nem jelez. Két argumentuma van: '-v'-re kiírja az összes tesztet végrehajtáskor, míg egy teszt nevének megadásával addig kihagy minden tesztet, míg meg nem találja a megadottat.

7.1 Tesztek írása

Készíts egy új file-t a megfelelõ könyvtárban: próbáld meg beszámozni a tesztedet, hogy a helyes idõben fusson le. Pl: az ICMP válasz teszteléséhez (02conntrack/02reply.sh) meg kell gyõzõdnünk a kérés helyes kezelésérõl (02conntrack/01simple.sh).

Jobb sok kis tesztet készíteni, ahol mindegyik lefed egy adott területet, mert segíti a probléma helyének pontos meghatározását.

Ha hibát észlelsz, egyszerûen lépj ki 'exit 1'-el, ami hibajelzést generál; ha valami, amit vizsgálsz hibát kell hogy jelezzen, akkor célszerû egy egyedi üzenet megjelenítése. A teszted 'exit 0'-val lépjen ki, ha minden sikeres volt. **Minden** parancs sikerességét ellenõrizni kell, akár a 'set -e' használatával, akár egy '|| exit 1' kiegészítéssel minden parancs után.

A segédfunkciók ('load_module' és 'remove_module') használhatók a modulok kezelésére: sose támaszkodj az automatikus betöltésre, kivéve, ha valójában azt szeretnéd tesztelni.

7.2 Változók és a környezet

Két játék interface-ed van: tap0 és tap1. A címük a `$TAP0` és `$TAP1` válozóban található. Mindkettõ netmaskja 255.255.255.0; a hálózati címük a $TAP0NET-ben és a $TAP1NET-ben található.

Van egy üres ideiglenes file-od a $TMPFILE-ban. Ez törlésre kerül a teszted végén.

A tesztjeid a testduite/ könyvtárból fognak futni. Ezentúl ha programokat szeretnél használni, akkor azokat a '../userspace' path-stringgel használd.

A scipted bõvebb információt is megjeleníthet, ha a $VERBOSE be van állítva (a felhasználó megadta a '-v' kapcsolót).

7.3 Hasznos eszközök

Számos hasznos segédprogramot találsz a "tools" könyvtárban: mindegyik nem-nulla visszatérési értékkel jelzi, ha valamilyen problémája akadt.

7.3.1 gen_ip

IP csomagokat tudsz a segítségével elõállítani, amelyek a kimeneten fognak megjelenni. A csomagokat bele tudod küldeni a tap0 és tap1 eszközökbe egyszerû kimenet-átirányítással a /dev/tap0 ill. /dev/tap1-re (ezeket a testsuite elsõ futáskor létrehozza, ha nem léteztek korábban).

gen_ip egy nagyon egyszerû program, ami igen kényes a paraméter-sorrendjére. Elõször az általános opciók jönnek:

FRAG=offset,length

Elkészíti a csomagot, és darabokra tördeli a megadott hosszal és eltolással.

MF

Beállítja a 'More Fragments' bitet a csomagban.

MAC=xx:xx:xx:xx:xx:xx

Beállítja a forrás MAC címet a csomagban.

TOS=tos

Beállítja a TOS mezõt (0-255).

Ezután jönnek a kötelezõ paraméterek:

source ip

Forrás IP cím.

dest ip

Cél IP cím.

length

Teljes hossz, a fejlécet beleszámolva.

protocol

A protokoll sorszáma, pl. 17 = UDP.

Az ezután következõ paraméterek protokoll-függõek: az UDP(17)-hez a forrás és a cél portszám; az ICMP(1)-hez a típus és a kód: ha a típus 0 vagy 8 (ping-válasz, vagy ping), akkor két további argumentumot vár (az ID és a sequence mezõk). A TCP-hez a forrás és célport, és a flagek ("SYN", "SYN/ACK", "ACK", "RST" vagy "FIN"). Itt lehetõség van három további argumentumra is: "OPT=" - vesszõvel elválasztott opció-felsorolás, "SYN=" egy sequence number-rel, valamint "ACK=" szintén egy sequence number-rel. Végül egy opcionális argumentum, a "DATA" jelzi a csomag tartalmát, amit a standard inputról tölt fel.

7.3.2 rcv_ip

Az IP csomagokat tudod vele ellenõrizni. Amennyire csak lehetséges, megpróbálja visszaállítani a gen_ip parancssorát (a fragmentek a kivételek).

Nagyon hasznos a csomagok feldolgozásában. Két kötelezõ argumentuma van:

wait time

A maximális idõ másodpercben, amíg a csomag megérkezésére vár a tandard input-ján.

iterations

Az értelmezendõ csomagok száma.

Egy opcionális argumentuma van: a "DATA". Ennek hatására kinyomtatja a csomag tartalmát a kimenetére a fejléc után.

Egy általános felhasználási mód:

```
# Set up job control, so we can use & in shell scripts.
set -m

# Wait two seconds for one packet from tap0
../tools/rcv_ip 2 1 < /dev/tap0 > $TMPFILE &

# Make sure that rcv_ip has started running.
sleep 1

# Send a ping packet
../tools/gen_ip $TAP1NET.2 $TAP0NET.2 100 1 8 0 55 57 > /dev/tap1 || exit 1

# Wait for rcv_ip,
if wait %../tools/rcv_ip; then :
else
    echo rcv_ip failed:
    cat $TMPFILE
    exit 1
fi
```

7.3.3 gen_err

Egy csomagot vár (pl. a gen_ip-tól) a bemenetén, s ICMP error üzenetre fordítja.

Három argumentuma van: a forrás IP cím, típus és kód. A cél IP cím a csomag forráscíme lesz.

7.3.4 local_ip

A bemenetén érkezõ csomagot elküldi a rendszerbe egy raw-socketen keresztül. Ez biztosítja a helyileg generált csomagot a tesztekhez (elválasztva a hálózati meghajtókba küldött csomagoktól, amik távolról érkezõnek látszanak).

7.4 Tanácsok

Minden tool feltételezi, hogy mindet meg tud csinálni egy írással vagy olvasással: ez igaz az ethertap device-okra, de nem biztos, hogy igaz ha valami trükköset csinálsz a pipe-okkal.

dd-t lehet használni a csomagok darabolására: dd-nek van egy obs (kimeneti blokkméret) opciója, ami lehetõvé teszi a csomagok egy íráson keresztül való megjelenítését.

Elõször a sikerességre tesztelj: pl. teszteld azt, hogy a csomag sikeresen lockolásra került. Elsõ teszt legyen az, hogy a csomag normálisan áthaladt, s **utána** teszteld csak a blokkolást. Különben egy nem kívánt hiba megállíthatja a csomagokat...

Próbálj meg precíz teszteket írni, s ne véletlen dolgokkal tesztelj, s figyeld, mi lesz belõle. Ha egy pontos teszt hibát jelez, akkor a hiba helye használható információt adhat. Viszont ha egy véletlen teszt hibázik, akkor nem lehet vele mit kezdeni.

Ha egy teszt üzenet nélkül jelez hibát, használhatod a '-x' opciót a legelsõ sorban (pl: '#! /bin/sh -x'), hogy lásd, hogy melyik programok futnak.

Ha a teszt véletlenszerûen áll meg, ellenõrizd a külsõ behatásokat (próbáld meg leállítani a külsõ interface-eidet). Például Andrew Tridgell-el közös hálózaton ülve felfigyeltem, hogy Windows broadcast-el gyötörték a gépemet.

8 Motiváció

Ahogy az ipchains-t fejlesztettem, rájöttem, hogy a csomagszûrést rossz helyen valósítottuk meg. Most nem találom, de írtam egy levelet Alan Cox-nak, aki csak annyit mondott, hogy 'miért nem fejezed be elõször amit csinálsz, annak ellenére, hogy talán igazad van'. Röviden a gyakorlatiasság gyõzött a helyes út felett.

Amiután befejeztem az ipchainst - ami egy kis módosításnak indult az ipfwadm kernel részein - s egy nagyobb újraírás felé fordultam, megírtam a HOWTO-t, kifinomult képet kaptam arról, hogy a Linux társadalomnak milyen problémái vannak a csomagszûréssel, maszkolással, port-átirányítással és hasonlókkal.

Ez az öröme a saját támogatás nyújtásának: közelrõl érezheted, hogy a felhasználók mit szeretnének megvalósítani, mivel szenvednek. A szabad program a legnagyobb jutalom a legtöbb felhasználó kezében, s ez azt jelenti, hogy egyszerûnek kell lennie. Az architektúra, s nem a dokumentáció volt a legnagyobb hiba.

Nos, megvolt a tapasztalat az ipchains kódjából, s az ötlet az emberek próbálkozásaiból. Csak két problémám volt:

Nem szeretnék visszatérni a biztonsági rendszerekbe. Biztonsági konzulensnek lenni egy állandó huzavona a lelkiismereted és a pénztárcád között. Alapszinten: te a biztonság érzését árulod, hadilábon áll az aktuális biztonsággal. Talán katonai területen, ahol értik a biztonságot, más lennék.

A második problémám az, hogy nemcsak a kezdõ felhasználó okozhat gondot, hanem növekvõ számú vállalat és ISP használja ezt a terméket. Ezektõl a felhasználóktól megbízható információkra van szükség, ha a jövõ otthoni felhasználóira tervezek.

Ezek a problémák megoldódtak amikor belefutottam David Bonn-ba, a WatchGuard-tól az Usenix-en 1998 júliusában. Egy Linux kernel programozót kerestek - a végén megegyeztünk, hogy egy hónapra elmegyek a seattle-i irodájukba, hogy hátha sikerül összehozni egy megállapodást, miszerint õk támogatják az új kódot cserébe a támogatásomért. A megállapodás többrõl szólt, mint szerettem volna. Nem szeretnék külsõ konzulensként szerepelni - legalábbis mostanában.

A WatchGuard-hoz való kötõdés lehetõvé tette egy nagy ügyfélkör elérését, s mindemellett független maradhattam tõlük, ami lehetõvé tette, hogy mindenkit egyformán támogassak.

Egyszerûen megírtam a netfiltert, az ipchains-t portoltam a tetjére, s elkészültem vele. Sajnos ez meghagyta a masquerading kódot a kernelben: de a masquerading kód függetlenítése a csomagszûréstõl egyike a legnagyobb nyereségeknek, de az megmaradt, hogy a masquerading-ot ki kell emelni a kernelbõl, s át kell helyezni a netfilter rendszerbe.

Az ipfwadm 'interface-address' szolgáltatásával (amit kivettem az ipchains-bõl) kapcsolatos tapasztalataim megmutatták, hogy nem lesz egyszerû munka a masquerading kód kivágása a kernelbõl, s nem lehet másra bízni a netfilter-be való portolását.

A meglevõ kódból a lehetõ legkevesebbet kellett megtartanom, s inkább új lehetõségekkel kibõvíteni azt, felbátorítandó a felhasználókat, hogy használják azokat. Ez a transzparens proxy, masquerading és port forwarding lecserélését jelentette. Más szavakkal: egy teljes NAT layer-t.

Bár elhatároztam, hogy portolom a meglevõ masquerading réteget a teljesen új NAT rendszer implementálása helyett, de a kódon meglátszott a kora, valamint a karbantartás hiánya. Nem volt felelõse a karbantartásának. Úgy látszik, hogy a komoly felhasználó nem használta a modult, s az otthoni felhasználók közül pedig senki sem vette a fáradságot a követésre. Bátor emberek, mint pl. Juan Ciarlante készítettek javítgatásokat hozzá, de elérte azt az állapotot, hogy szükségessé vált az újraírása.

Fel szeretném hívni a figyelmedet, hogy nem én voltam az alkalmas ember a NAT újraírására: soha többé nem használtam a masquerading-et, s nem tanulmányoztam a meglevõ kódot. Talán ezért tartott tovább, mint szerettem volna. De az eredmény egészen jó - legalábbis a saját véleményem szerint - s biztos vagyok benne, hogy sokat tanultam belõle. Nincs kétségem afelõl, hogy a második verzió sokkal jobb lesz, miután megláttuk, hogy miként használják az emberek.

9 Köszönetnyilvánítás

Köszönet azoknak, akik segítettek, különösen Harald Welte-nek a Protokollsegítõk (Protocol Helpers) fejezetért.