Mérési útmutató a „Hálózati protokollok vizsgálata proxy technológiával (Zorp)” című méréshez

2018. április 9.

A mérést kidolgozta:
Pfeiffer Szilárd és Ládi Gergő

Balasys IT Kft.

BME, CrySyS Adat- és Rendszerbiztonság Laboratórium

Tartalomjegyzék

1. Zorp: Elméleti összefoglaló

A következőkben rövid áttekintést nyújtunk a Zorp környezetben használt alapvető fogalmakról, a hozzáférés-vezérlési lista specifikumairól, a hozzáférés engedélyezésének és tiltásának különböző módjairól.

1.1. Forgalom szabályozása

Zorp esetén a forgalom szabályozás követlező módjai közül választhatunk[4, Service]

- forgalom tiltása
  - csomagok eldobása (`DenyService`)
  - csomagok visszautasítása (`DenyService`)
- forgalom engedélyezése
  - csomagszűrő jellegű működés (`PFService`)
  - applikációs szintű proxy (`Service`)

A forgalom tiltásának első változatában tűzfal a beérkezett csomagok mindennemű válasz nélkül dobja el, ezzel egyrészt csökkentve a saját terhelését, hiszen a kliensnek küldött visszautasító válasz is hálózati terhelés, másrészt késleltetésre kényszeríti a klienst, mivel az kénytelen kivárni az adott protokollban (pl: TCP) előírt időt a válasz esetleges megérkezéséig. Így ha ezt a módozatot használjuk kliens oldalon nincs közvetlen bizonyíték arra, hogy az adott szolgáltatás nem működik, erre csupán a várt válasz meg nem érkezése, illetve az ez által okozott időtúllépés utal.

1.1.1. Forgalom tiltása

Csomagok visszautasítása – amennyiben erre a protokoll lehetőséget ad –, annyit jelent, hogy a szerver küld választ a kliens felé az általa kezdeményezett forgalom visszautasításáról. Ennek konkrét módja a `DenyService` paramétereiként adható, illetve adandó meg, az alábbiak szerint.

```
DenyService(
    name='TCPReset',
    ipv4_setting=DenyIPv4.TCP_RESET,
    ipv6_setting=DenyIPv6.TCP_RESET
)

DenyService(
    name='ICMPPortUnreachable',
    ipv4_setting=DenyIPv4.ICMP_PORT_UNREACHABLE,
    ipv6_setting=DenyIPv6.ICMP_PORT_UNREACHABLE
)
```

Az első esetben (`TCPReset`) a klienstől érkező forgalomra válaszként egy TCP resetet küld a *Zorp*, míg a második esetben (`ICMPPortUnreachable`) egy ICMP port unreachable lesz a válasz mind IPv4, mind IPv6 esetén. Értelem szerűen előbbi csak TCP forgalmak esetén értelmes, míg az utóbbi TCP és UDP esetén is használható, ugyanakkor tetszőleges IP alapú forgalomra nem értelmes, hiszen a válasz azt tartalmazza, hogy a célzott port (ami TCP/UDP specifikus) nem elérhető.

1.1.2. Forgalom engedélyezése vizsgálat nélkül

Az egyszerűbb, egyszersmind kevesebb beállítási lehetőséget és ezzel együtt kisebb rugalmassági szintet biztosító, valamint alacsonyabb biztonsági szintet nyújtó forgalomátengedési lehetőség a `PFService`, melynek működése gyakorlatban megegyezik az *IPTables* működésével.

```
PFService(
    name='PacketFilter'
)
```

Ebben az esetben az érkező forgalom teljes egészében kernel-space kerül feldolgozásra, azzal csak a Zorp kernel modulja foglalkozik, a user-space daemon (*Zorp*) számára forgalom voltaképpen nem is létezik. A routing, az esetleges cím- és porthamisítások, illetve a NAT teljes egészében a kernel modul által történik.

1.1.3. Forgalom engedélyezése vizsgálattal

Amennyiben szeretnénk az adott forgalmat mélyebb elemzésnek (deep packet inspection) alávetni, akkor egy harmadik engedélyezési módra lesz szükségünk (`Service`), amiben csak forgalom engedélyezésére, vagy tiltására vonatkozó döntés meghozását, illetve a forgalom elterelését – a user-space felé – végzi a kernel modul, minden egyéb (routing, NAT, cím-, porthamisítás, protokoll elemzés, módosítás, ...) a *Zorp* feladata.

```
Service(
    name='DeepInspection',
    proxy_class=HttpProxy
)
```

A labor gyakorlat szempontjából leginkább említésre méltó különbség az előző (`PFService`) és a mostani (*Service*) átengedési mód között, a protokoll elemzésének lehetősége, illetve az ebben rejlő lehetőségek, valamint az ezzel járó konfigurációs változtatások. A fenti példában a `Service` konfigurációja egy új paraméterrel bővül (`proxy_class`), ami a protokoll elemzését végző modul (proxy) típusát adja meg. Ezen típus tulajdonképpen egy *Python* nyelvű osztály neve, lévén a *Zorp C*/C++(11) nyelven írt protokollelemzői *Python* nyelven egészíthetőek ki.

Természetesen a *Zorp* tartalmaz számos teljes értékű és megfelelő alapértelmezett paraméterekkel rendelkező proxyt. Amennyiben ezt a probléma bonyolultság megköveteli, nem csupán a paraméterek módosíthatók, hanem számos ponton az alapértelmezett működés is felülbírálható, kiegészíthető, sőt lehetőség van teljes egészében *Python* nyelven implementált proxyk létrehozására is, amire a labor keretein belül szintén látunk példát.

1.2. Hozzáférés-vezérlés

1.2.1. Alkalmazandó szabály kiválasztása

Ellentétben számos más tűzfal megoldással a hozzáférés-vezérlési szabályok kiértékelése nem az első illeszkedő szabály megtalálásáig történik (first match), hanem minden egyes esetben minden egyes szabály kiértékelődik és ezek közül a leginkább illeszkedő (best match) fog érvényre jutni[4, Best match].

A kiválasztás során számos feltétel megadható (pl: forrás/cél IP cím, port, ...), illetve azonos típusú feltételekből akár több is. Az azonos típusú feltételek egymással *vagy*, míg a különböző típusú feltételek egymással *és* kapcsolatban állnak. Az alábbi példában lévő szabályra tehát csak azok a forgalmak illeszkednek, amik az `8.8.8.8` címet célozzák, ezen belül is az `53`-as portot. Gyakorlatban az alábbihoz hasonló szabály segítségével lehet a *Google* publikus DNS szerveréhez a forgalmat kiengedni, oly módon, hogy ez egy csomagszűrő esetén történne.

```
PFService(name='PacketFilter')

Rule(
    dst_subnet='1.2.3.4/32', dst_port=53,
    service='PacketFilter'
)
```

A szabály alapvetően két részből áll. Egyrészt a forgalom illeszkedési feltételek (match) megadásából, másrészt a forgalom kiváltandó művelet megadásából (`service`). A művelet csak akkor fog végrehajtódni, amennyiben a feltételek teljesülnek, az összes illeszkedő szabály közül az az egy jut érvényre, amelyiknek az illeszkedése a legjobb. Lássuk a következő példa konfigurációt:

```
PFService(name='PacketFilter')

Rule(
    id=1,
    dst_subnet='8.8.8.8/32'
    service='PacketFilter'
)
Rule(
    id=2,
    dst_port=53,
    service='PacketFilter'
)
```

Az egyes számú szabályunk minden olyan esetben illeszkedik, ha csomag a `8.8.8.8` IP címet célozza, míg a kettes számú akkor, ha a cél port `53`. Kézenfekvő a kérdés, hogy mi történik abban az esetben, ha a forgalomra mindkét feltétel igaz. Ilyenkor feltételek közötti precedencia[4, Conditions] dönt. Amelyik feltétel precedenciája nagyobb, annak az illeszkedése lesz jobb. Ez esetben a kettes számú szabályé, amiből jól látszik, hogy a konfigurációban megadott szabályok sorrendjének nincs jelentősége. Ugyanakkor persze az is igaz, hogy a fenti példa meglehetősen mesterkélt, hiszen mindkét esetben ugyanaz a művelet hajtódik végre, így az eredmény is ugyanaz lesz.

Ha az adott forgalom a szabályrendszer egyetlen elemében megadott feltételeknek sem felel meg, vagyis egyetlen szabályra sem illeszkedik, akkor a sorsáról az *IPTables* dönt, alapértelmezés szerint az adott csomag eldobásra kerül.

1.3. Haladó ismeretek

1.3.1. Testre szabott proxy implementálása

Zorp esetén az egyes proxyk – pontosabban szólva proxy osztályok – testre szabása minden esetben származtatást jelent az alapvetően *C/C++(11)* nyelven írt, de *Python* nyelven kiterjesztett ősosztályból, szintén *Python* nyelven.

```
from Zorp.Http import HttpProxy

class MyHttpProxy(HttpProxy):
    def config(self):
        HttpProxy.config(self)

        self.timeout = 60000

def default():
    Service(name='CustomizedHttpservice', proxy_class=MyHttpProxy)
    Rule(dst_port='80', service='CustomizedHttpservice')
```

A fenti példa nem tesz egyebet, mint a *Zorp* részeként szállított `HttpProxy` timeout értékét írja felül, így ez a testre szabott proxy egy percet fog várakozni, mielőtt a kapcsolatot időtúllépés miatt megszakítaná, az alapértelmezett 5 perces értékkel szemben. Ezt úgy éri el, hogy származtat a megfelelő ősosztályból (`HttpProxy`) és felüldefiniálja annak `config` függvényét. Ebben a függvényben írhatóak felül az alapértelmezett értékek, illetve állíthatóak be olyan paraméterek, melyeknek nincs alapértelmezett értéke.

1.3.2. Python nylelvű proxy implementálása

Zorp esetén lehetőség van arra is, hogy saját, vagy a szállított proxyk által egyáltalán nem támogatott protokollokhoz – teljes egészében *Python* nyelven – írjunk proxyt. Ehhez egy létező létező proxyból `AnyPyProxy` való származtatásra lesz szükség, ami voltaképpen csak egy váz, ahol az alapvető paraméterek a már megismert metódus szerint (`config`) – a függvény felüldefiniálásával – történik, míg a hálózati forgalom tényleges kezelését, a protokoll elemzését, esetleges módosítását a `proxyThread` függvény részeként kell implementálni.

```
class FullyCustomizedProxy(AnyPyProxy):
    def config(self):
        self.client_max_line_length = 65535

    def proxyThread(self):
        self.readData()

        self.analyzeData()
        if (necessary):
            self.modifyData()

        self.writeData()
```

Fontos megjegyezni, hogy ezen proxy használatakor hálózati forgalom mindennemű kezelésének (olvasás/írás kliens/szerver oldalon, hibakezelés, ...) feladata ránk hárul, amennyiben a proxyt a teljes forgalom kezelésére használjuk, nem pedig egy már létező protokoll adat részének további elemzésére, ahogy azt a következő fejezetben látni fogjuk.

1.3.3. Proxyk egymásba ágyazása

A *Zorp* lehetővé teszi proxyk egymásba ágyazását (*stacking*), melynek jelentősége abban áll, hogy minden proxy a saját maga által ismert részeket – `HttpProxy` esetén ez a *HTTP* protokoll – értelmezi, majd a számára csak adatként kezelendő részeket tovább tudja adni egy másik protokoll értelmező számára további elemzésre. Ez az egymásba ágyazás nem feltétlenül kell, hogy egylépcsős legyen, bonyolultabb esetekben lehetnek akár többszörös egymásba ágyazások is.

```
class AnyPyProxyRequest(AnyPyProxy):
    def proxyThread(self):
        pass

class AnyPyProxyResponse(AnyPyProxy):
    def proxyThread(self):
        pass

class MyHttpProxy(HttpProxy):
    def config(self):
        HttpProxy.config(self)
        self.request_stack["*"] =
            (HTTP_STK_DATA, (Z_STACK_PROXY, AnyPyProxyRequest))
        self.response_stack["*"] =
            (HTTP_STK_DATA, (Z_STACK_PROXY, AnyPyProxyResponse))
```

A fenti példában egy olyan testre szabott *HTTP* proxy implementáció látható, ami mind a *HTTP* kérések (*request*) mind a *HTTP* válaszok (*response*) adat részét továbbítja egy-egy – az `AnyPyProxy` ősosztályból származó – újabb proxy felé, ahol azok kezelése kell megtörténjen.

2. Gyakorlati összefoglaló

2.1. Architektúra

A mérés alapvetően virtualizált környezetben (*VMware*) történik, egy kliens-szerver architektúrában, ahol a kliens és a szerver között egy tűzfal (*Zorp*) helyezkedik el, amivel a tulajdonképpeni mérési feladatok elvégzendőek. A kapcsolatok minden esetben

- a kliens gépről kezdeményeződnek,
- a tűzfal gépen keresztül jutnak el a szerverre.

A szerverek – az utolsó mérési feladattól eltekintve – az erre a célra szolgáló virtuális gépen futnak.

2.2. Gépek beállítása

Az importálást követően minden virtuális gép négy hálózati interfésszel rendelkezik a következők szerint:

1. internet kapcsolat

   célja: az internet elérésének biztosítása frissítések telepítésének céljából

   működése a VMware által biztosított virtuális DHCP szerver segítségével kap IP címet, viszont elkerülendő az esetleges routing problémákat alapértelmezés szerint nem aktív

   azonosítása az operációs rendszerben `eth0` néven érhető el

2. kliens oldal

   célja: olyan kapcsolat biztosítása a kliens és a tűzfal között, amin kizárólag ezen két gép közötti forgalom zajlik

   működése a VMware által megvalósított belső hálózatok kiépítésére szolgáló típussal, mely az architektúrát felvázoló ábra szerint kap IP címeket

   azonosítása az operációs rendszerben `eth1` néven érhető el

3. szerver oldal

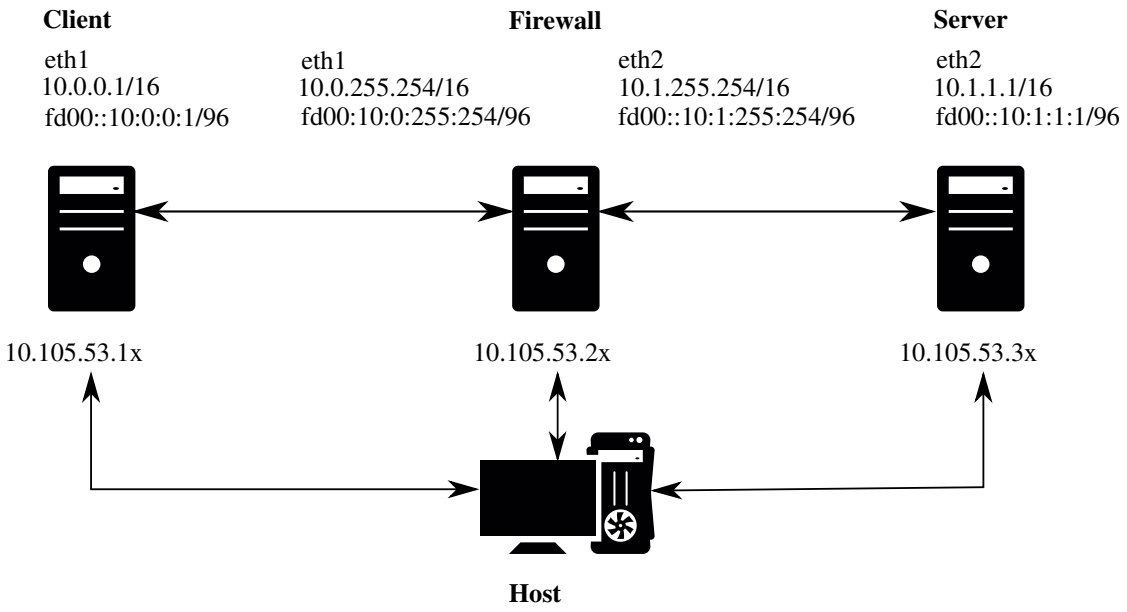
   célja: olyan kapcsolat biztosítása a szerver és a tűzfal között, amin kizárólag ezen két gép közötti forgalom zajlik

   működése a VMware által megvalósított belső hálózatok kiépítésére szolgáló típussal, mely az architektúrát felvázoló ábra szerint kap IP címeket

   azonosítása az operációs rendszerben `eth2` néven érhető el

2.3. Gépek elérése

A virtuális gépek kialakítása az 1. ábrán láthatóak szerint történik.



1. ábra. Virtuális gépek kialakítása, ahol az x helyére rendre a mérőhely számát kell behelyettesíteni

- az autentikáció felhasználónév jelszó párossal történik

- két felhasználóval rendelkezik (`root`, `balasys`)
- a felhasználónév és a jelszó megegyezik

A bejelentkezés a gépekre vagy a gépen közvetlenül felhasználónév (`balasys`) jelszó (`balasys`) páros megadásával, vagy *SSH* segítségével, az alábbi parancsot használva – a jelszó megadása után – történik.

```
ssh virtualis_gep_ip_cime -l felhasznalonev
```

Az egyes gépek az `eth0` interfészeiken – a virtuális környezet *DHCP* szervere által kiosztott –, IP címeken keresztül érhetőek el. A konkrét címek kiderítéséhez az alábbi parancs kiadása szükséges, közvetlenül a virtuális gépre bejelentkezve.

```
ip -4 addr show dev wlan0
```

A virtuális gépek billentyűzetkiosztása angol. Amennyiben a magyar nyelvű kiosztás használata a mérés során kényelmesebbnek tűnik, a váltás következő paranccsal végezhető el:

```
sudo loadkeys hu
```

2.4. Konfigurációs fájlok

A *Zorp* konfigurációs fájljai – hasonlóan más alkalmazásoknál használt könyvtárszerkezethez – az `/etc/zorp` könyvtárban kapnak helyet. Alapvetően három állomány kódosítására van szükség a gyakorlatban.

instances.conf az egyes *Zorp* példányokhoz kapcsolódó beállítások kapnak itt helyet (pl: naplózás részletessége, szabályokat leíró konfigurációs fájl, . . .)

zones.py a zónák definícióit tartalmazza

policy.py a tényleges tűzfalszabályokat, illetve a általuk indítandó szolgáltatások (`Service`), illetve a tényleges forgalom-ellenőrzést és manipulációt végző osztály (`Proxy`) leírását tartalmazza

A mérés során csak az utóbbi állomány módosítására lesz szükség, ehhez tetszőleges szövegszerkesztő (`nano`, `vim`, . . .) használható.

3. Feladatok

3.1. IPTables

A mérés *IPTables*-ös része közvetlenül a szerver tűzfalát állítja, hogy semmiképp ne keveredjen a zorp-os feladatokkal. Éles környezetben az *IPTables* szabályokat érdemes a dedikált tűzfalon felvenni, amennyiben van ilyen gépünk, és nem keverni a szolgáltatásokat ellátó szerverekkel.

A feladatok megoldásában segítséget nyújthat a 11. ábra, a Hálózatbiztonság (VIHIMB00) tárgy tűzfalakról szóló előadásának anyaga [3] valamint a következő leírások: [2, 1].

3.1.1. Előkészületek

Jelentkezzünk be a kliensre és a szerverre SSH használatával. Listázzuk ki és dokumentáljuk a szerver filter táblájának jelenlegi tartalmát (javasolt kapcsolók: -nvL)!

A következő feladatok elvégzésénél érdemes egy bash scriptbe gyűjteni a parancsokat. A script elején érdemes a policy-t mindkét irányban megengedőre állítani, minden szabályt törölni, majd hozzáadni az előbb kilistázott szabályokat, illetve kibővíteni a scriptet az alábbi feladatok megoldásával. Így jól követhető, hogy milyen szabályok vannak beállítva, illetve a beállítás egy könnyen reprodukálható folyamattá válik (a cél, hogy az elkészült script sorrendben oldja meg az összes feladatot).

3.1.2. SSH forgalom beengedése a mérőgépek felől

A netfilter jelenleg is aktív, de minden csomagot (így az előző feladatban indított SSH kapcsolatot is) átenged. A továbbiakban át szeretnénk állítani szigorúbb üzemmódba, azonban előtte *be kell konfigurálnunk, hogy a mérőgépek felől érkező SSH kapcsolatokat engedje át, különben kizárjuk magunkat!*

Vegyünk fel tehát egy szabályt, amely a labor tartományából (10.105.0.0/16), az *eth0* interfészre, az SSH szerver portjára érkező csomagokat átengedi.

3.1.3. Válaszüzenetek kiengedése a szerverről

Azt már beállítottuk, hogy a bejövő SSH csomagok átmenjenek a tűzfalon, azonban a kimenő forgalomra is gondolnunk kell.

Vegyünk fel egy szabályt, amely minden olyan forgalmat minden interfészen, minden porton és minden IP címről kienged, amelynek volt előzménye, azaz valamilyen, már folyamatban lévő kommunikációhoz vagy egy kiépült kapcsolathoz tartozik.

3.1.4. Szabályok ellenőrzése

Listázzuk ki és dokumentáljuk a filter tábla jelenlegi tartalmát! Megjelent-e a két újonnan felvett szabály? Hány csomag illeszkedett eddig az egyes szabályokra? Pár másodperc eltelte után ismét nézzük meg a tábla tartalmát. Változott-e a szabályok számlálója? Ha nem, akkor valamit elrontottunk! Addig ne folytassuk a mérést, amíg a hiba javításra nem kerül, ellenkező esetben kizárjuk magunkat.

3.1.5. Alapértelmezett policyk átállítása

Állítsuk át a tűzfalat úgy, hogy minden olyan bejövő és kimenő csomagot dobjon el, amely nem illeszkedik egyik felvett szabályra sem!

3.1.6. Tipikus szabályok beállítása I.

Vegyünk fel szabályokat, amelyek lehetővé teszik, hogy bárki bárhonnan elérhesse a szerveren futó webszervert HTTP és HTTPS protokollok használatával a szokásos portokon.

3.1.7. Tipikus szabályok beállítása II.

Vegyünk fel egy szabályt, amely lehetővé teszi, hogy a belső hálózaton belül (10.1.0.0/16) bárki elérhesse a szerveren futó TFTP szolgáltatást. Mivel valójában nem fut TFTP szolgáltatás a gépen, átmenetileg egy nc parancs segítségével fogadjunk kapcsolatokat a megfelelő porton. A kliensről ellenőrizzük valóban lehet-e kapcsolódni a szerverhez.

3.1.8. Tipikus szabályok beállítása III.

A nagyfőnök szeretné, ha az ő gépéről (10.105.250.100) érkező minden forgalom átmenne a tűzfalon, viszont fél attól, hogy megint órákon át a szerveren található weblapokat fogja böngészni, így azt is kéri, hogy ezt az egyet tiltsuk neki (a HTTPS-sel nem kell most foglalkozni).

3.1.9. Tipikus szabályok beállítása IV.

Rájöttünk, hogy nem is fut már TFTP szolgáltatás a szerverünkön, így érdemes volna törölni az erre vonatkozó szabályt. Tegyük ezt meg anélkül, hogy tudnánk, pontosan hanyadik ez a szabály a chainen (ne a létrehozó szabályt törölje a scriptből, hanem adjon hozzá törlő szabályt)!

3.1.10. Haladóbb konfiguráció I.

Pingeljük folyamatosan a szervert a kliensről! Mit tapasztalunk?

3.1.11. Haladóbb konfiguráció II.

Vegyünk fel egy szabályt, amely engedélyezi az ICMP forgalmat a 10.1.0.0/16-os hálózaton belülről érkező címekről ugyanebbe a hálózatba tartozó címekre. Mit látni most a kliens konzolján?

3.1.12. Haladóbb konfiguráció III.

A főnök panaszkodik, hogy lassú a szerver. Szerinte az a baj, hogy túl sok a feldolgozott ping üzenet. Kéri, hogy korlátozzuk a beengedett ICMP üzenetek számát úgy, hogy percenként körülbelül 20 üzenet kerüljön csak feldolgozásra.

Elsőként töröljük az előzőleg felvett szabályt, most a sorszám ismeretében. Utána írjuk át úgy, hogy megfeleljen a főnök elvárásainak.

Mit látni most a kliens konzolján?

3.1.13. Haladóbb konfiguráció IV.

Csörög a telefon. Már megint a főnök hív... Szerinte a szerver által küldött csomagok TTL-jéből meg lehet állapítani, hogy a szerveren Linux operációs rendszer fut. Igaza lehet-e?

Kéri, hogy *IPTables* segítségével oldjuk meg, hogy inkább Windowsra hasonlítson a szerver (hint: Windows esetén a TTL mező kezdőértéke 128).

Mi vehető ebből észre a kliens konzolján?

3.1.14. Policy visszaállítása

A mérés következő feladataiban nem netfiltert fogunk használni. Így, hogy ne okozzon gondot, állítsuk vissza alaphelyzetbe, azonban előtte, még utoljára listázzuk ki és dokumentáljuk a filter tábla tartalmát. Hova tűnt a TTL-t beállító szabály?

Állítsuk be, hogy azon kimenő és bejövő csomagokat, amelyre nem vonatkozik szabály, engedje be a rendszer.

3.1.15. Az összes felvett szabály törlése

Töröljük az összes felvett szabályt! Hány parancsra van ehhez szükség? Miért?

3.2. Zorp

3.2.1. Működés ellenőrzése

A kialakított hálózati topológia működésének ellenőrzésére hajtsa végre az alábbiakat:

1. jelentkezzen be mindhárom virtuális gépre *SSH* segítségével
2. ellenőrizze a kliens és a szerver gépeken a tűzfal elérését az `arping -I interfésznév cél_IP_cím` parancs segítségével
3. a jegyzőkönyvezze
   - a kiadott parancsot
   - a parancs kimenetének azon részét, ami az elküldött kérést és az arra érkező választ tartalmazza

Ezt követően kísérelje meg a kliens gépről a szerver gép elérését:

1. ellenőrizze szerver gép elérését IPv4 protokollon a következő paranccsal:
   `ping szerver.ip.cime`
2. ellenőrizze web szerver elérését IPv4 protokollon a következő paranccsal:
   `wget -O /dev/null http://szerver.ip.cime`

3. ellenőrizze web szerver elérését IPv4 protokollon a következő paranccsal[1]
   `curl --insecure -o /dev/null https://szerver.ip.cime`

3.2.2. Alapértelmezett tiltás beállítása

Módosítsa a *Zorp* konfigurációs állományokat, hogy minden forgalom tiltásra kerüljön oly módon, hogy a kliens értesüljön a tiltásról. A tiltás megvalósítására egyetlen szabályt használjon! A konfiguráció módosítását követően indítsa újra a *Zorp*ot, majd ismételje meg a működés ellenőrzésekor (3.2.1) végrehajtottakat és dokumentálja azokat az ott leírtaknak megfelelően, illetve jegyzőkönyvezze a konfigurációs változtatásokat (`Rule`,`Service`)!

3.2.3. Specifikus szabályok felvétele

Módosítsa a konfigurációs állományokat úgy, hogy az ICMP forgalom átengedése `PFService`, a *HTTP* forgalom pedig `Service` segítségével valósuljon meg, ahol a proxy típusa legyen a *Zorp* részeként szállított `HttpProxy`! Ehhez alkosson az általános tiltó szabálynál specifikusabb, az ICMP, illetve a 80-as porton zajló TCP forgalomra jobban illeszkedő (1.2.1) szabályokat.[2]

Jegyzőkönyvezzen az előző mérésnél leírtak szerint!

3.2.4. Saját proxy osztály létrehozása

A fentiekben ismertetetteknek (1.3.1) megfelelően hozzon létre egy testre szabott `HttpProxy` változatot, melynek segítségével cserélje le a `User-Agent` *HTTP* fejléc értékét, az alábbi kódrészlet megfelelő helyre történő beillesztésével.

```
self.request_header["User-Agent"] = (HTTP_HDR_CHANGE_VALUE, "Forged Browser")
```

Ellenőrizze a kapcsolat kiépülését a szerver alkalmazás (*Apache*) napló állományában (ennek helye: `/var/log/apache2/access.log`), illetve a tűzfal gép rendszernapló állományában (`/var/log/syslog`). Jegyzőkönyvezze a tűzfalon keresztüli elérést bizonyító szerver oldali naplósort, illetve a konkrét HTTP parancs (`GET`) átviteléről szóló tűzfal napló bejegyzést!

Jegyzőkönyvezzen az előző mérésnél leírtak szerint!

3.2.5. Titkosított forgalom átvitele proxy segítségével

Hozzon létre egy újabb szabályt a HTTPS forgalom (port 443) átvitelére! A proxy típusa a *Zorp* részeként szállított `PlugProxy` legyen!

1. hajtsa végre az ellenőrzést az előző mérésnél leírtak szerint
2. ellenőrizze és jegyzőkönyvezze a szerver által felmutatott tanúsítvány kibocsátójának (issuer) nevét (CN): `openssl s_client -connect 10.1.1.1:443`

3.2.6. Titkosított forgalom átvitele titkosítás bontásával

Illessze be az alábbi – a titkosítás kibontására szolgáló – konfigurációs részletet a korábban létrehozott saját proxy osztály alá. Ezt követően módosítsa a `Service` definícióját, hogy az használja az `EncryptionPolicy` osztályt, illetve cserélje le a jelenleg használt `PlugProxy` osztályt a korábban létrehozott saját `HttpProxy` osztályra.

```
EncryptionPolicy(
     name="https_encryption_policy",
     encryption=TwoSidedEncryption(
          client_certificate_generator=DynamicCertificate(
```

[1] Megjegyzés: `--insecure` helyett a `-k` akkor is letölti a certificatet ha az nem megbízható.

[2] a megfelelő protokoll azonosítók a *Python* `socket` moduljában `IPPROTO_TCP`, `IPPROTO_ICMP`, illetve `IPPROTO_ICMPV6` néven érhetőek el

```
                private_key=PrivateKey.fromFile(
                    key_file_path="/etc/zorp/certs/key.pem",
                    passphrase=""
                ),
                trusted_ca=Certificate.fromFile(
                    certificate_file_path="/etc/zorp/certs/trusted.pem",
                    private_key=PrivateKey.fromFile("/etc/zorp/certs/trusted.pem",
                        passphrase="passphrase"
                    ),
                ),
                untrusted_ca=Certificate.fromFile(
                    certificate_file_path="/etc/zorp/certs/untrusted.pem",
                    private_key=PrivateKey.fromFile("/etc/zorp/certs/untrusted.pem",
                        passphrase="passphrase"
                    ),
                ),
            ),
            client_verify=ClientNoneVerifier(),
            server_verify=ServerCertificateVerifier(
                trusted=FALSE
            )
        )
)

...

Service(
    name="https_service",
    encryption_policy="https_encryption_policy",
    proxy_class=MyHttpProxy
)
```

A feladat megoldásához töltse le a tűzfal számára a titkosításhoz szükséges kulcsot és tanusítványokat (http://www.crysys.hu/downloads/vihimb01/2017/certs.zip) és másolja be a tűzfal virtuális gépének `/etc/zorp/certs` mappájába a kicsomagolt pem fájlokat, majd állítsa be a megfelelő jogosultságokat (user:rw-, group:r–, others:—)!

Hajtsa végre az ellenőrzést az előző mérésnél leírtak szerint!

4. Jegyzőkönyv

A jegyzőkönyvet a mérés után egy héten belül el kell küldeni a mérésvezetőnek pdf formátumban. A jegyzőkönyvnek az alábbiakat kell tartalmaznia:

- Hallgató(k) neve és Neptun kódja
- Mérés neve
- Mérés időpontja
- Mérőhely száma
- Feladatok megoldása

A megoldások leírásánál törekedni kell a tömör, de érthető válaszra. **A leírásból a megoldásnak reprodukálhatónak kell lennie!**

5. Lehetséges beugró kérdések

5.1. IPTables-höz kapcsolódó kérdések

1. Mire való a netfilter? Mi köze van iptables-nek a netfilterhez?

2. Sorolj fel három, a netfilterhez köthető táblát!
3. Mire való a filter tábla?
4. Mire valók a láncok (chain)?
5. Sorolj fel legalább három beépített láncot!
6. Mire való az INPUT chain?
7. Mire való az OUTPUT chain?
8. Milyen sorrendben kerülnek a szabályok kiértékelésre?
9. Mi történik egy csomaggal, ha eléri a lánc végét, és eddig egyik szabály sem illeszkedett rá?
10. Mi történik a következő parancs hatására?

```
iptables -D INPUT 3
```

11. Mi történik a következő parancs hatására?

```
iptables -A OUTPUT -j ACCEPT -d 8.8.8.8 -p udp --dport 53
```

5.2. Zorphoz kapcsolódó kérdések

1. Milyen szolgáltatások használhatóak forgalom átengedésére/tiltására *Zorp* esetén?
2. Mi az alapvető különbség egy `Service` és egy `PFService` segítségével átengedett forgalom tűzfalazásában?
3. Miben különbözik a *Zorp* által használt best match kiértékelés a más tűzfalaknál megszokott first match metodikától?
4. Képes-e a *Zorp* titkosított kapcsolatok (pl: HTTPS) protokolljának layer 7 elemzésére?
5. Forgalom tiltása esetén – a kliensek szempontjából – milyen metódusok lehetségesek?
6. Milyen logikai kapcsolatban állnak egymással egy szabály (`Rule`) azonos forgalmi paraméterre (pl. célport) vonatkozó részei?
7. Milyen logikai kapcsolatban állnak egymással egy szabály (`Rule`) különböző forgalmi paraméterre (pl: célport és IP) vonatkozó részei?
8. Milyen objektum-orientált programozási paradigma alkalmazása szükséges a proxy testre szabásához?

Hivatkozások

[1] Iptables man page. http://ipset.netfilter.org/iptables.man.html.

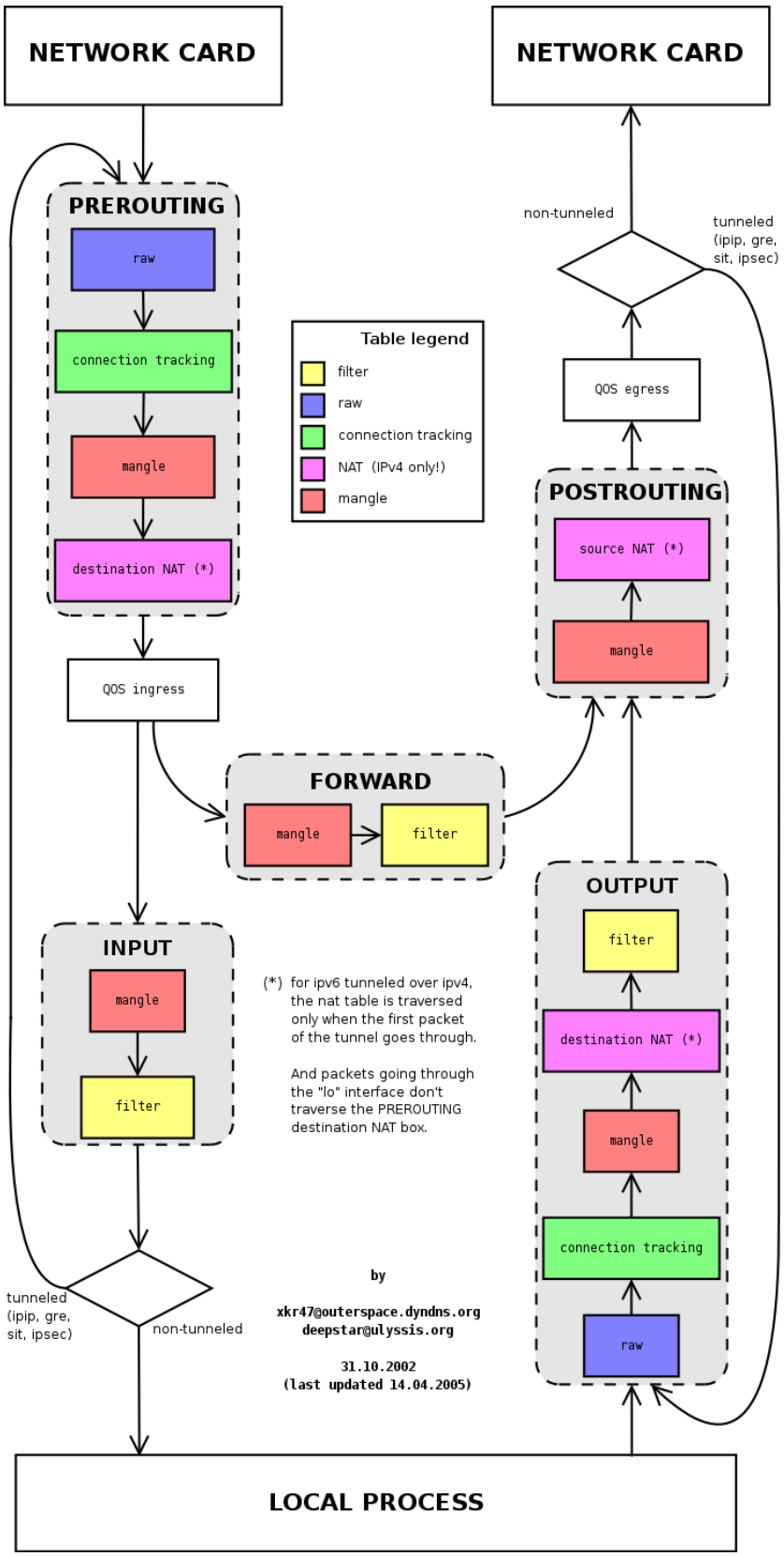
[2] Justin Ellingwood. A deep dive into iptables and netfilter architecture, 2015. https://www.digitalocean.com/community/tutorials/a-deep-dive-into-iptables-and-netfilter-architecture.

[3] Tamás Holczer. Hálózatbiztonság (VIHIMB00) - Tűzfalak, 2018. https://www.crysys.hu/downloads/vihimb00/2018/slide_04_Firewall.zip.

[4] Szilárd Pfeiffer. *Zorp GPL Tutorial*. BalaSys Ltd., Alíz utca 2. H-1117 Budapest, 2013-2016.

Tárgymutató

A. Függelék



2. ábra. A netfilter működése