IPSEC/IKE2 VPN elmélet

Alapfogalmak

Most egy kis elméleti rész következik. Ahol lehet, ott be fogom írni hogy RouterOs-ben az adott rész beállításai melyik menüpont alatt érhetők el. Ezen felül gyakorlati, konkrét javaslatokat teszek az egyes beállításokra a mi konkrét példáinkra (két telephely folyamatos összeköttetése VPN alagúton, illetve road warrior setup).

Ez a protokoll több fő részre bontható:

- Az `IKE` felelős azért, hogy a kommunikációs csatorna két oldalán levő felek közösen és biztonságosan megegyezzenek olyan kriptografikus kulcsokban, amiket később a kommunikáció során föl lehet használni az adatforgalom titkosítására. Az `IKE` az Internet Key Exchange rövidítése. Két fő verziója van. Az egyes verziót már nem nagyon használjuk. A kettes verziót úgy alakították ki, hogy kiküszöbölje az első verzió használata közben tapasztalt hibákat és hiányosságokat. Az IKEv2 normál UDP csomagokat és alapértelmezésben az 500-as portot használja.
- A titkosított adatforgalom úgy zajlik, hogy a tunnel két végén levő eszközök az egymás irányába küldött csomagokat a küldés előtt enkapszulálják (azaz beágyazzák) `IPSEC` csomagok belsejébe, a beérkező csomagokat pedig dekapszulálják (kicsomagolják).
- Az enkapszulációra kétféle szabvány terjedt el: az Authentication Header (`AH`) és az Encapsulating Security Payload (`ESP`). Az `AH` arra alkalmas, hogy garantálja a csomagok eredetét. (Harmadik fél nem képes meghamisítani őket.) Az `ESP` ezen felül titkosítást is végez - harmadik fél nem képes hozzájutni az eredeti csomagok adattartalmához akkor sem, ha a két fél közötti csatorna összes `IPSEC` csomagját el tudja olvasni.

Bár erről később még részletesen írok, de elöljáróban megemlítek két fogalmat. Ezek szükségesek ahhoz, hogy elkerüljük az előre hivatkozásokat, és úgy előre-hátra ugrálás nélkül lehessen olvasni ezt a cikket.

IPSEC Peer

Az IPSEC protokoll használatakor mindig van két kitüntetett csomópont. Az egyik a csomagok beágyazását (enkapszuláció) és titkosítását végzi. A másik a csomagok kicsomagolását (dekapszuláció), a titkosítás feloldását és az eredetiség vizsgálatot végzi. A gyakorlatban a kommunikáció kétirányú szokott lenni, ezért mindkét fél párhuzamosan végzi mindkét feladatot. Ezeknek a feleknek a neve `ipsec peer`. Fontos látni, hogy az enkapszulációt és dekapszulációt végző felek nem feltétlenül ugyan azok, mint akik a csomagokat eredetileg küldik vagy fogadják. A mi site-to-site példánkban az internet és a telephely (`branch01` és `office`) határán álló router-ek a `peer`-ek, mivel ők végzik a csomagok be- és kicsomagolását. Ugyanakkor a csomagok feladói és címzettjei a telephelyen belül található gépek. A gyakorlatban előfordulhatnak olyan hálózati topológiák is, ahol a csomagok olyan útvonalon utaznak, ahol az út egyes szakaszait IPSEC

csomagokba ágyazva, egy másik részét normál módon teszik meg. Mi most csak a feladatleírásban szereplő problémára koncentrálunk.

> RouterOS esetén az ehhez tartozó menüpont a `/ip ipsec peer`. Itt lehet megadni az IPSEC kommunikációhoz a távoli peer-eket. A helyi peer-t nem kell külön megadni, mert az maga a router.

IPSEC Policy

A másik fogalom az `ipsec policy`. Amikor egy csomagot továbbítani kell egy forrás címről egy cél címre, akkor a router elsősorban forgalomirányítási táblázatokat (routing table) használ annak megállapítására, hogy melyik csomagot melyik interface-en keresztül küldje ki. A csomagokat tűzfal szabályokkal tudjuk szűrni és manipulálni. A szűrés azt jelenti, hogy bizonyos csomagokat eldobunk ahelyett, hogy továbbítanánk őket. A manipuláció azt jelenti, hogy módosítjuk őket a továbbítás előtt. Ilyen manipuláció lehet például a forrás- vagy célcím megváltoztatása (srcnat és dstnat), a csomagok sorba állítása, várakoztatás és priorizálás stb. Bizonyos módosítások automatikusak, minden router a rajta áthaladó összes csomagot módosítja az IP protokoll szabályainak megfelelően. (Pl. TTL csökkentése) Az IPSEC protokoll a csomagok feldolgozását a fentiektől jól elkülöníthető rétegben végzi. A csomagok feldolgozása ebben a rétegben úgynevezett ipsec szabályok, szakszóval `ipsec policy`-k segítségével történik. Egy ilyen policy kicsit hasonlít egy tűzfal szabályra:

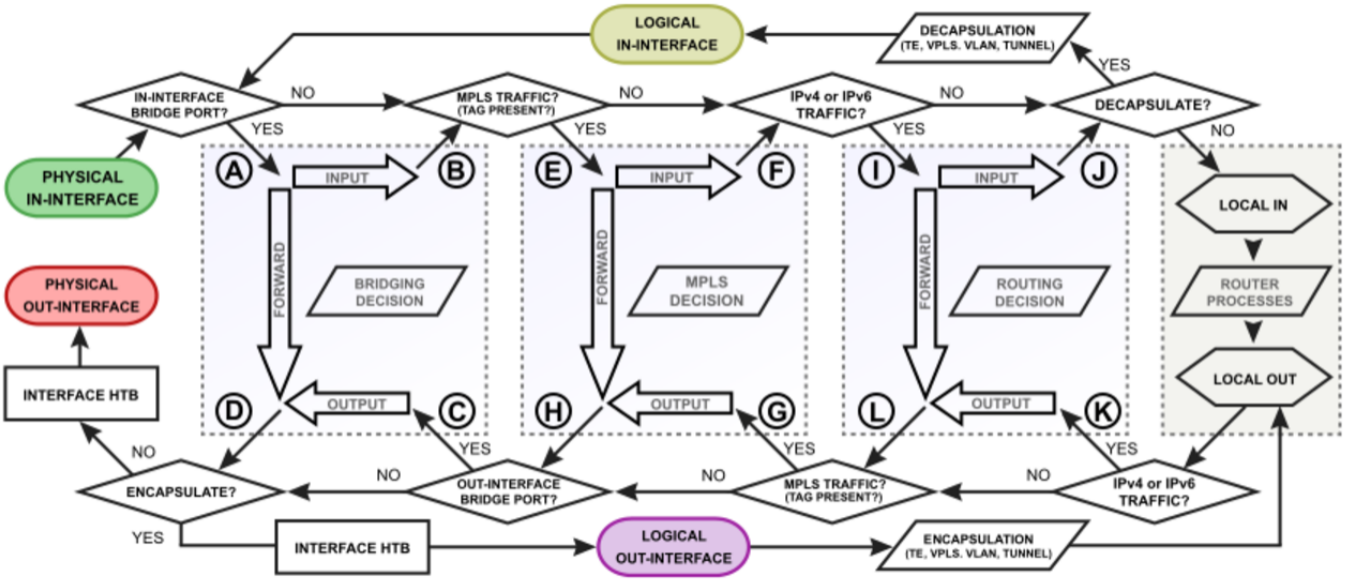
- IP csomagokon működik
- Minden szabályban feltételeket adunk meg. Ezek a feltételek vonatkozhatnak a csomag cél címére, forrás címére, az csomag típusára (TCP, UDP) stb.
- A feltételeken felül meg van adva egy művelet (`action`) is. Ez RouterOS esetén lehet `discard` (csomag eldobása) `encrypt` (csomag enkapszulálása és titkosítása) vagy `none` (ne csináljon vele semmit). Megjegyzés: az `encrypt` művelet egy kicsit megtévesztő lehet, mivel `AH` esetében nem történik titkosítás. Ami mindig megtörténik az az enkapszuláció.
- Amikor egy csomag illeszkedik a policy-ben megadott szabályokra, akkor a policy-ben megadott művelet végrehajtódik. Az `encrypt` művelet a csomagot beágyazza (enkapszulálja) egy IPSEC csomag belsejébe. Ennek során titkosíthat és ellenőrző összegeket írhat be az így előállított új csomagba.
- Fontos látni, hogy a beágyazás során az eredeti csomag "elhasználódik". Helyette egy teljesen új csomag képződik. Ez az új csomag szintén egy rendes IP csomag, de a protokoll száma már nem az eredeti (pl. UDP vagy TCP) hanem `IPSEC AH` vagy `IPSEC ESP`. Van rendes forrás- és célcíme, és a router az eredeti csomag helyett ezt a csomagot továbbítja. A fogadó oldal a nála megtalálható kulcs segítségével tudja kicsomagolni az eredeti csomagot. A kicsomagoláskor szükség lehet a titkosítás feloldására és az ellenőrző összeg ellenőrzésére. Az `AH` és `ESP` csomagok működés szempontjából datagram típusúak. A továbbítás nem feltétlenül sorrendhelyes. Ha a csomag valami miatt elveszik, akkor az IPSEC nem biztosít erről visszajelzést, és nem próbálja meg az újraküldést. Az eredeti feladó és címzett számára az IPSEC transzparens, nem látható. Ha az eredeti csomag kapcsolatállapot alapú (TCP) volt, akkor az esetleges csomag újraküldéseket is az

erdeti (pl. TCP) protokoll valósítja meg függetlenül attól, hogy a csomagok az út egyrészében IPSEC csomagokba vannak ágyazva.
- Nem létezik `decrypt` művelet. A bejövő csomagok esetén a RouterOS automatikusan felismeri az IPSEC csomagokat, és megpróbálja kicsomagolni őket. Más szóval a bejövő csomagokra nem kell külön policy-kat megadni.
- Több ilyen szabályt (`ipsec policy`) meg lehet adni. Ezek egymás után lefutnak, és ha van olyan ami illeszkedik, akkor a később következő szabályok már nem futnak le.
- A tűzfal szabályokkal ellentétben az `ipsec policy`-nál nincsenek `chain`-ek.

> RouterOS esetében a policy-k az `/ip ipsec policy` menüpont alatt találhatók.

Enkapszuláció és dekapszuláció

A ki- és becsomagolási folyamatot jobban nyomon lehet követni ezen az ábrán:



Forrás: https://wiki.mikrotik.com/wiki/Manual:Packet_Flow

A három nagy szürke doboz balról jobbra sorrendben:

- Switch (layer 2 routing), ebben vannak az A,B,C,D virtuális portok
- MPLS, ebben vannak az E,F,G,H virtuális portok
- Router (layer 3 routing), ebben vannak az I,J,K,L virtuális portok
- Belső RouterOS folyamatok (local in, local out, router processes)

Látható, hogy a `decapsulate?` és `encapsulate?` döntést jelképező dobozok a switching és routing részeken kívül helyezkednek el. A kimenő csomagok közvetlenül a fizikai interfész elérése előtt érik el az `encapsulate?` dobozt. Tehát az enkapszuláció a switching és routing után történik. Ennek során az eredeti (enkapszulált) csomag "elhasználódik". Helyette egy új csomag jön létre, ami a `local out` belső processzben képződik, és innen kezdődik a feldolgozása. Hasonlóan, a

dekapszulációról való döntés is a switching és routing után történik. Ha dekapszulációra van szükség, akkor ezt a `local in` belső RouterOS processz végzi el. Ennek hatására az eredeti IPSEC csomag "elhasználódik", helyette megjelenik az eredeti (dekapszulált) csomag a `local out` processz kimenetén. Ezután megkezdődik a feldolgozása. Az IPSEC csomag és a dekapszulált csomag is áteshet routing/switching műveleteken. Sőt, ha többszörös beágyazás van, akkor ezek többször is megismétlődhetnek. Ha például az internet kapcsolat PPPoE protokollt használ, és ezen keresztül érkezik be egy IPSEC csomag, akkor az eredeti csomagot két dekapszuláció után kapjuk meg.

Policy template és generált policy-k

A legegyszerűbb esetben mindkét peer-en kézzel adjuk meg a policy-kat. Tehát kézzel írjuk elő hogy mik azok a csomagok, amiket IPSEC csomagokba kell ágyazni, és biztonságosan továbbítani a másik peer-hez. Ezt a módszert akkor lehet használni, ha kevés számú, előre megismerhető peer-t használunk, és kevés számú policy-t. Van azonban egy másik módja is a policy-k előállításának, ez pedig a `policy template` ("szabály sablon") megadása. RouterOS-ben úgy tudunk ilyeneket létrehozni, hogy policy `template` attribútumát `yes` értékre állítjuk. RouterOS-ben ezek a policy-k úgy jelennek meg a `/ip ipsec policy` menü alatt, hogy egy `T` betű van a státusz oszlopban. Egyelőre ezekről elegendő tudni annyit, hogy a rendszer a policy template-ek alapján konkrét dinamikus policy-kat generál a kapcsolat létrehozásakor. Ezeket automatikusan kitörli akkor, amikor a kapcsolat befejeződik. Ezek működését egy másik cikkben (road warrior VPN) fogom elmagyarázni.

IKE - a kulcs csere alapjai

Az `IKE` protokoll célja, hogy két egymással kommunikálni kívánó fél számára biztonságos módon meghatározzon olyan kriptografikus kulcsokat, amikkel a biztonságos kommunikáció elvégezhető. Az IKE protokoll nem csak a kulcsokat határozza meg, hanem a kulcsokhoz tartozó titkosítási algoritmusokat és egyéb olyan paramétereket is, amik szükségesek a kommunikáció lebonyolításához. Ezeket a dolgokat összefoglaló néven Security Association-nak vagy röviden `SA`-nak nevezzük. Egy SA mindig két konkrét félhez (peer) tartozik. Ha ez első olvasásra nem teljesen világos akkor gondolj úgy az SA-ra, mint egy olyan titkosító kulcsra ("jelszóra"), amit csak az egy mással kommunikáló felek ismernek. Fontos látni, hogy az IKE mindig két fél között egyezteti az algoritmusokat és a kulcsokat. Emiatt az `SA`-k mindig párban jönnek létre (a kommunikációs csatorna két oldalán.) Ennek hiányában az IPSEC kommunikáció nem lehetséges.

Az IKE protokoll egy démont (folyamatosan a háttérben futó programot) használ. Ez a démon az idő nagy részében nem csinál semmit, inaktív. Alapvetően kétféle módon lehet aktiválni:

- Ha van olyan adatforgalom ami illeszkedik egy `ipsec policy`-re, akkor az enkapszuláció elvégzéséhez szükség van egy megfelelő `SA`-ra. Ha ez az `SA` még nem áll rendelkezésre, akkor ez felébreszti az IKE démont. Ez az IKE démon fölveszi a kapcsolatot a másik oldalon levő IKE démonnal.
- A másik oldalon levő IKE démon fogadja ezt a kapcsolatot. Ez a másik mód amin keresztül egy IKE démont aktiválni lehet (távolról egy másik IKE démon által).
- A RouterOS-ben van egy olyan beállítás is (`send-initial-contact`), amivel elő lehet írni, hogy a két oldalon konfigurált peer akkor is fölvegye egymással a kapcsolatot, ha épp nincsen olyan továbbítandó csomag, amit enkapszulálni kellene. Az SA előre kiépítése felgyorsítja a válaszidőt az első csomagok elküldésénél.

Miután felébredtek, a két oldalon levő két démon elkezdi egyeztetni az algoritmusokat és a kulcsokat. Ezt UDP üzenetek segítségével valósítják meg. Az egyeztetés végeredménye (szerencsés esetben) az lesz, hogy mindkét félnél létrejön egy-egy `SA`, amivel már le lehet bonyolítani a kommunikációt. A kulcsok meghatározásához a Diffie-Hellman néven ismert kulcs csere algoritmust használják. Ez az algoritmus képes előállítani ugyan azt a titkos és véletlenszerű kulcsot a kommunikációs csatorna két oldalán úgy, hogy közben a csatornát potenciálisan figyelő többi fél nem képes hozzájutni ezekhez a kulcsokhoz (annak ellenére, hogy a kommunikáció nincs titkosítva). Az algoritmus rövid neve a továbbiakban DH algoritmus, vagy egyszerűen csak DH. A DH algoritmusnak több változata ismert, amelyek különböző bonyolultsági fokkal rendelkeznek. A magasabb bitszámot használó, erősebb titkosítást biztosító algoritmusok magától értetődően magasabb számítási igénnyel rendelkeznek. A különböző változatait a kulcs bitek számának megfelelően szokás csoportosítani. Ezek a csoportok az úgynevezett DH csoportok.

Az IKE démon az UDP/500 -as portot használja a kulcsok egyeztetésére. Ha szeretnél IPSEC/IKEv2 kommunikációt, akkor ezt a portot nyitva kell hagynod. Ez a kommunikáció normál UDP csomagokat használ, amelyek adattartalma nincs titkosítva vagy aláírva. (Ezen felül a 4500-as portot is ha NAT mögött van valamelyik fél, erről alább írok.)

Az IKEv2 protokoll az `SA`-t két fázisban hozza létre.

IKE első fázis

Az első fázisban a két fél megegyezik azokban az algoritmusokban, amiket használni fognak a kommunikációra a következő fázisban. Ezen felül meghatározásra kerül néhány olyan kulcs is, ami majd a második fázis összes kommunikációját védi. Ezeket a kulcsokat a fent említett DH algoritmussal határozzák meg. Ha gyakran és sok alagutat kell fölépíteni, akkor a DH algoritmus egy korlátozó tényező lehet (mivel számításigényes). Ebben az írásban az a célunk, hogy két fix címmel rendelkező router között permanens kapcsolatot alakítsunk ki. Itt nem lehet számítani nagy számú kapcsolat gyakori felépítésére, ezért érdemes magasabb (biztonságosabb) DH csoportot választani. A magas számítási igény miatt az első fázisban meghatározott kulcsok általában hosszú lejáratúak (több óra). Az első fázisban a sikeres egyeztetéshez a következőkben kell megegyezniük a feleknek:

- autentikációs módszer (pl. előre megadott titkos jelszó vagy tanúsítvány)
- DH csoport
- titkosítási algoritmus
- kulcs csere algoritmus
- hash algoritmus
- NAT-T
- DPD és leárati idő (opcionális)

Ezeket alább részletezem.

Authetikációs (azonosítási) módszer

A felek valamilyen módon igazolják azt, hogy tényleg azok, akinek mondják magukat. Ez a felek azonosítása, vagy szakszóval autentikáció. A legegyszerűbb esetben erre egy egymás között megosztott titkos kulcsot használnak. Egy egyszerű jelszó, angolul "pre shared key" vagy röviden `PSK`. Egy másik, nagyon elterjedt módszer az X.509 tanúsítványok használata. Ez utóbbit nehezebb beállítani, de jóval biztonságosabb. Ha nagy biztonságra törekedünk, akkor az előre megosztott kulcsok (PSK) használatát kerülni kell.

> Keress rá arra hogy `psk ike vulnerability` és találni fogsz egy csomó cikket ami arról szól, hogy a PSK azonosítás mennyire sebezhető.

A PSK használata mégis indokolt lehet némely esetben:

- Ha a végfelhasználónak nincs elég szakértelme a tanúsítvány alapú azonosítás beállításához.
- Ha a távoli csomópont nem támogatja a tanúsítvány alapú azonosítást (pl. régi szoftver verzió).
- Ha alacsonyabbak a biztonsággal kapcsolatos elvárásaink.

Vannak egyéb más azonosítási módszerek is, erre most nem térek ki. Mi ebben az írásban alhálózatok összekötését akarjuk megvalósítani. Teljes telephelyek közötti mindenféle kommunikációt kívánunk titkosítani, ezért itt érdemes erősebb, tanúsítvány alapú azonosítást használni.

> RouterOS-ben az azonosításhoz tartozó beállítások a `/ip ipsec identity` menüpont alatt találhatók. Ezen belül az `auth-method` attribútum határozza meg az azonosítási módszert.

DH csoport

Ez a már föntebb említett Diffie-Hellman algoritmus bonyolultsági fokát adja meg. A magasabb bitszámú algoritmus magasabb biztonságot ígér, de számításigényesebb.

Titkosítási algoritmus

Ez egy titkos kulcsú, szimmetrikus titkosítási algoritmus kiválasztását jelenti. Mi ebben a példában az `AES256-CBC` algoritmust választjuk, mert ez hardver támogatással rendelkezik, és elég biztonságos.

Kulcs csere algoritmus

Itt `IKE2`-őt választunk. (Akit érdekel az utánanézhet a többi típusnak, ez a cikk az IKEv2-ről szól.)

Hash algoritmus

Ez az algoritmus egy kriptografikus hasítófüggvény aminek az a legfontosabb jellemzője, hogy az invertálása túl nagy bonyolultságú ahhoz, hogy értelmes keretek között megvalósítható legyen. Mi ebben a példában az `SHA2-256` algoritmust választjuk, mert ez hardver támogatással rendelkezik, és elég biztonságos. Ezt a hash algoritmust használjuk többek között a fent említett ellenőrző összegek számításához. Közvetve ez garantálja az enkapszulált csomagok integritását. (Az ellenőrző összeget általában nem direkt módon a hash függvény határozza meg, hanem az ebből származtatott MAC kód.)

Néhány szó a NAT-T-ről

Az IPSEC protokollhoz külön módszert kellett kidolgozni a NAT mögött levő végpontok elérésére. Az IKE protkoll első és második fázisát nem befolyásolja az, ha a peer-ek közötti útvonalon áthaladó csomagok NAT-on esnek át. Ez azért van, mert az alap IKE protokoll normál UDP csomagokat

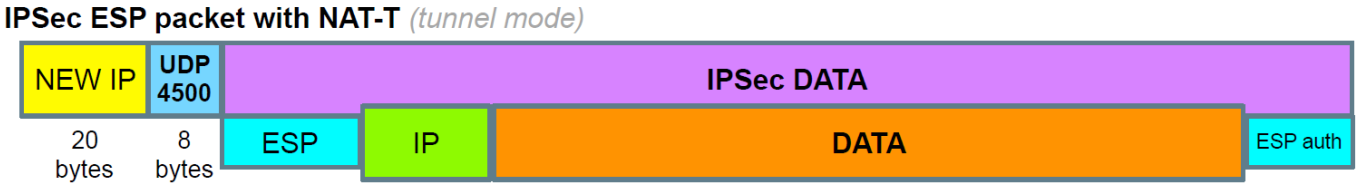
használ, és ezek forrás- és célcíme szabadon módosítható. Azonban az IPSEC által titkosított adatforgalomnál ez már problémát okoz.

Az egyik probléma a csomagok módosításával kapcsolatos:

- `ESP` csomagok esetén azért nem lehet NAT-ot végezni, mert a beágyazott eredeti csomag tartalmazhatja a forrás- és célcímet. Ezekhez a NAT-ot elvégezni kívánó router nem fér hozzá, mert az eredeti csomag titkosítva van.
- `AH` csomagok esetén az eredeti címek hozzáférhetőek, nincsenek titkosítva. Viszont a csomag integritását egy olyan hash érték biztosítja, ami érvénytelenné válik akkor, ha a csomagban levő forrás vagy célcím módosításra kerül. Így tehát az AH csomagokon elvégezhető a NAT, de a célállomásra való beérkezés után a célállomás ezeket a csomagokat eldobja, mert észreveszi hogy a csomagot valaki módosította.

A másik probléma azzal kapcsolatos, hogy a két fél közötti kommunikáció olyan útvonalon haladhat végig, amiben részt vesznek olyan router-ek, amik szintén futtatnak IKE démont és tartalmaznak IPSEC policy-ket. Ha egy ilyen router-re beérkezik egy IPSEC csomag, akkor a router esetleg elkezdheti feldolgozni (megpróbálja kicsomagolni). Valahogy el kell érni azt, hogy ezek a csomagok csak a cél csomóponton legyenek dekapszulálva.

Ezen problémák elkerülésére lefoglalták a 4500-as portot és kialakítottak egy plusz protokollt arra, hogy az IPSEC csomagok át tudjanak haladni NAT-olt hálózatokon. Az eredeti `ESP=50` illetve `AH=51` protokoll mezőt lecserélik `UDP=17`-re, és beszúrnak egy érvényes UDP header-t az IP header és az ESP/AH header közé, valamit módosítják a cél portszámot 4500-ra. Ezekkel a változtatásokkal ezek a csomagok normál UDP csomagként továbbíthatók. A hozzáadott extra UDP header miatt elvégezhető rajtuk a NAT anélkül, hogy az enkapszulált csomag integritása megsérülne.



Mivel ez a módosítás csökkenti a hasznos payload méretét (azonos MTU mellett), ezért ezt csak akkor szokás használni, ha arra lehet számítani hogy a felek között NAT történik. Ha a felek között biztosan nem történik NAT, akkor érdemes kikapcsolni ezt a funkciót, ezzel növelve az egy csomagban kiküldhető (hasznos) adat mennyiségét.

Bár ezek a problémák a kulcs csere (IKE) használatakor nem jelentkeznek, de mégis az IKE protokoll az, ami előre megegyezik abban, hogy a felek használjanak-e NAT-T módot vagy ne. Így amikor a csomagok enkapszulációja szükségessé válik, akkor a router már előre tudja, hogy el kell-e végezni a NAT-T csomag transzformációt.

Az algoritmusok egyeztetésének módja

RouterOS-ben (és általában más szoftverekben is) megadható több algoritmus. A két oldalon levő IKE démon a megadott lehetséges algoritmusok közül olyan választ, ami mindkét oldal számára megfelelő. Ezen belül a sorrend is lényeges - ha több lehetséges algoritmust adunk meg, akkor azoknak magasabb a prioritása, amelyeket előrébb sorolunk a listában.

> RouterOS-ben az ehhez tartozó menüpont az `/ip ipsec profile` és az `/ip ipsec proposal` . A profile az első fázishoz tartozik, a proposal a második fázishoz.

Ha nem található olyan algoritmus amit mindkét fél megadott a listájában, akkor a kulcscsere nem hajtható végre, és az IPSEC kapcsolat nem hozható létre.

Az algoritmusok kiválasztásánál nem csak azt kell figyelembe venni, hogy mennyire biztonságos kapcsolatot szeretnénk. Figyelembe kell venni az elérhető számítási teljesítményt (ami az adatforgalom sebességétől is függ!), valamint a távoli peer lehetőségeit. Ha például megnézzük a Windows 10 kliensek lehetőségeit akkor azt látjuk, hogy a kettes fázisban kizárólag az SHA1 hasító függvényt támogatja. Így például ha azt szeretnénk hogy Windows 10 kliens tudjon csatlakozni, akkor kénytelenek leszünk engedélyezni az SHA1 hasító függvényt. (Megjegyzés: bár maga az SHA1 már rég nem tekinthető biztonságosnak, de az SHA1 alapú MAC kódok igen.)

A mi esetünkben site-to-site kapcsolatot akarunk kialakítani két MikroTik Router között, így előre ismerjük a peer-ek képességeit, és biztonságosnak mondott algoritmusokat tudunk használni.

IKE második fázis

A második fázisban a felek (peers) létrehoznak egy vagy több SA párt. Ezek lesznek később használva az IPSEC csomag enkapszulációhoz (titkosításra és MAC generálásra). Minden IKE démon által előállított SA-nak van egy maximális életkora (lifetime). Az életkor lehet megadva lejárati időben, maximális adatforgalomban, illetve lehet akár mindkettő is. Az életkor elérése után az SA nem használható tovább a kommunikációhoz, új SA-t kell kiépíteni.

Kétféle életkor létezik. egy "soft" és egy "hard". Amikor az SA eléri a "soft" életkort, akkor az IKE démon kap egy értesítést, és újra elkezdi futtatni a második fázist. Ennek az a célja, hogy az életkor végén a kommunikáció folytatásához szükséges "friss" SA azonnal rendelkezésre álljon. Így a kommunikáció nem lesz várakoztatva a kulcsok egyeztetése miatt (az IPSEC forgalomnak nem kell várakoznia az IKE-re).

A második fázisban a feleknek a következőkben kell megegyeznie:

- Mód (alagút vagy transzport)
- IPSEC protokoll (ESP vagy AH)
- Azonosítás (autentikáció) módja
- PFS (DH) csoport
- Életkor

Ezeket alább részletezem.

Mi az a transzport és alagút mód?

A csomagok enkapszulációja során az eredeti IP csomagnak van egy forrás- és egy célcíme. Transzport mód esetében a forrás- és célcím nem kerül beágyazásra. Ezt a módot akkor lehet jól használni, ha egy olyan biztonságos kapcsolatot akarunk kiépíteni, aminél a titkosítandó csomagok forrás- és célcíme ugyan az, mint a peer-ek címe. (A mi esetünkben ez azt jelenti, hogy az egyik router az eredeti csomag küldője, a másik az eredeti csomag végleges címzettje.)

Azokban az esetekben, amikor az eredeti csomagok feladója és/vagy címzettje nem egyezik meg az enkapszulációt/dekapszulációt végző peer-ekkel, alagút módot kell használni. A mi példa feladatunk ilyen! A kapcsolatot két alhálózat között kell megvalósítanunk. Az egyik alhálózatból induló csomagok valahol beérkeznek egy router-be, ami enkapszulálja őket. Ezután az adatok titkosított IPSEC csomagok belsejében haladnak tovább egy másik router-hez. Ez a másik router elvégi a dekapszulációt, és innentől kezdve a csomagok újra titkosítatlan módon haladnak tovább az eredeti célcím irányába. Tehát az eredeti csomagok forrás- és célcíme nem egyezik meg azoknak a router-eknek a címével, amelyek az enkapszulációt/dekapszulációt végzik. Egy ilyen típusú összeköttetés kialakításához a transzport mód nyilván nem használható, mivel az eredeti csomagok forrás- és célcímeit nem lehet használni az összeköttetés kialakításához. Az IPSEC csomag célba juttatásához az kell, hogy a célcíme a távoli peer címe legyen, és ez nem egyezik meg az eredeti címzettel. Az eredeti címek általában olyan belső/foglalt hálózati címek amikhez az interneten nem lehet útvonalat választani. Ugyanakkor az eredeti forrás- és célcímek megőrzése szükséges ahhoz, hogy kicsomagolás után a távoli alhálózaton belül továbbíthatóak legyenek az eredeti csomagok az eredeti (végleges) célcímre. Tehát az IPSEC csomagok forrás- és célcíme a peer-ek címeivel egyezik meg, de valahol el kell tárolni az eredeti forrás- és célcímet is. Tunnel módban pontosan ez történik. Az eredeti csomag teljes egészében, az eredeti forrás- és célcímekkel együtt enkapszulálásra kerül. Az új IPSEC csomag forrás és célcímei eltérhetnek az eredeti csomag forrás- és célcímeitől. Így az IPSEC csomag forrás- és célcíme olyan publikus címeket tartalmazhat, amikhez az interneten lehet útvonalat választani, a kicsomagolás után pedig újra elérhetőek lesznek azok a (belső privát) címek, ami alapján az útvonalválasztás a helyi/belső alhálózaton belül elvégezhető.

A tunnel mód bármikor használható a transzport mód helyett. Azonban fontos megjegyezni azt, hogy a tunnel mód további header-ek hozzáadását igényli, és így csökkenti az egy csomagban továbbítható hasznos adat (payload) maximális méretét, és fölösleges csomag fragmentációt okozat. Ezért ha biztosan nincs rá szükség, akkor ne használjunk tunnel módot.

IPSEC csomag formátumok: AH és ESP

Az AH csomag formátum arra alkalmas, hogy garantálja a küldő fél eredetiségét. (Azt, hogy valóban a másik fél küldte a csomagot.) Más szóval ezt a csomag integritásának nevezik. Az AH nem titkosítja az adatokat! Ez úgy működik, hogy az eredeti csomag egyes részeire kiszámítunk egy ellenőrző összeget. Az összeg kiszámításához felhasználjuk a választott hash algoritmust és az SA-ban található titkos kulcsot. Hogy a csomag mely részeiből számítjuk ki ezt az összeget, az függ

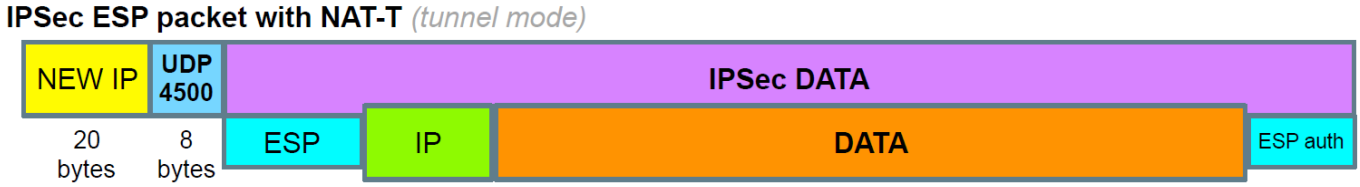
attól is, hogy transzport vagy alagút módot használunk.

- Transzport módnál az AH header-t az IP header után szúrjuk be. Az IP adat és IP header is részt vesz az ellenőrző összeg számításában. Azok az IP mezők amik megváltozhatnak a továbbítás során (pl. TTL) nullára vannak állítva az összeg kiszámítása előtt. Ez csökkenti a header-ek méretét, és növeli a hasznos payload méretét. Ugyanakkor ez azzal jár, hogy az IP header-ben levő címek nem módosíthatók.
- Alagút módnál a teljes IP csomagot enkapszuláljuk. Az eredeti IP csomag teljes egésze (ide értve az forrás és célcímet valamint a TTL mezőket is) részt vesz az ellenőrző összeg számításában. Ez garantálja az eredeti csomag forrás és célcímének integritását, de növeli a header-ek méretét, és csökkenti a hasznos payload méretét. Az IP header-ben levő címek módosíthatók.

Bővebb infó: https://wiki.mikrotik.com/wiki/Manual:IP/IPsec#Authentication_Header_.28AH.29

Az ESP csomag formátum garantálja a csomagok integritását, és titkosítja a bennük levő adatokat. Teljesen más mezőket használ mint az AH. Ezeket a mezőket három csoportba lehet osztani:

- **ESP Header** - ez a titkosított adat előtt van. A helye attól függ, hogy transzport vagy alagút módot használunk.
- **ESP Trailer** - Ez a titkosított adatok után következik. Ez tartalmaz egy padding-ot ami szükséges ahhoz, hogy a titkosítandó adat mérete a titkosító algoritmus blokk méretének egész számú többszöröse legyen.
- **ESP Authentication data** - Ez tartalmaz egy Integrity Check Value (ICV) értéket. Ez az AH protokollhoz hasonló módon van kiszámítva.



Bővebb infó:

https://wiki.mikrotik.com/wiki/Manual:IP/IPsec#Encapsulating_Security_Payload_.28ESP.29

Az AH használata akkor indokolt, ha az átvinni kívánt adatok nem titkosak (nyilvánosak) és csak annyi a célunk, hogy kívülálló/közbeékelődő támadók ne legyenek képesek más nevében adatokat küldeni. A mi példánkban a célok között szerepel az, hogy a két site közötti kommunikációt mások ne tudják lehallgatni. Emiatt ESP-t kell használunk.

Második fázis, azonosítás (autentikáció) módja

Itt pont ugyan azok igazak, mint az első fázisnál. (Az azonosítás módja az első és második fázisnál azonos.)

PFS (DH) csoport

Ennek megértéshez vissza kell térnünk az első fázishoz. Az első fázisban csak azokat a kezdeti kulcsokat határozzuk meg DH algoritmussal, amik a második fázis biztonságos lebonyolításához szükségesek. A második fázisban létrehozott kulcsok nem tartanak örökké. Ahogy azt korábban írtam, a második fázisban létrehozott SA-knak van egy soft és egy hard lifetime limit-je, és időnként meg kell őket újítani. A második fázisban létrehozott SA nem ugyanazokat a kulcsokat tartalmazza, mint amiket az első fázis használt. Minden második fázisban létrehozott SA saját, egyedi titkos kulcsokat tartalmaz, amik előre nem kitalálhatóak. Tehát valójában kétféle SA van: egy szülő SA ami az első fázishoz tartozik, és ehhez kapcsolódó további "gyermek" SA-k, amik a második fázishoz tartoznak. Az első fázis rendszerint egyszer játszódik le, ezért az ehhez tartozó kulcsot szokás *állandó kulcsnak* is nevezni. A második fázis sokszor lejátszódhat, mivel a kommunkikáció hosszú ideig eltarthat, és ezalatt többször szükség lehet a (korlátos életidővel rendelkező), második fázis által generált SA-k megújítására.

Ahogy korábban említettem, a DH algoritmus nagyon számításigényes. Ezért a második fázisban létrehozott SA-k titkos kulcsait alapesetben az első fázisban létrehozott SA kulcsaiból származtatják. Mivel az első fázisban DH algoritmussal létrehozott titkos kulcsok közvetlenül semminek a titkosítására nincsenek fölhasználva, ezért szinte semmi esély nincs arra, hogy ezt a kezdeti kulcsot bárki brute force módszerrel (a kommunikáció rögzítésével és annak próbálkozásos visszafejtésével) meghatározza. Ez a működés `pfs-group=none` beállításnak felel meg RouterOS-ben. Ennél a beállításnál a második fázis egyáltalán nem használ DH algoritmust. Emiatt gyors, kisebb CPU igényű. Mégis viszonylag biztonságos.

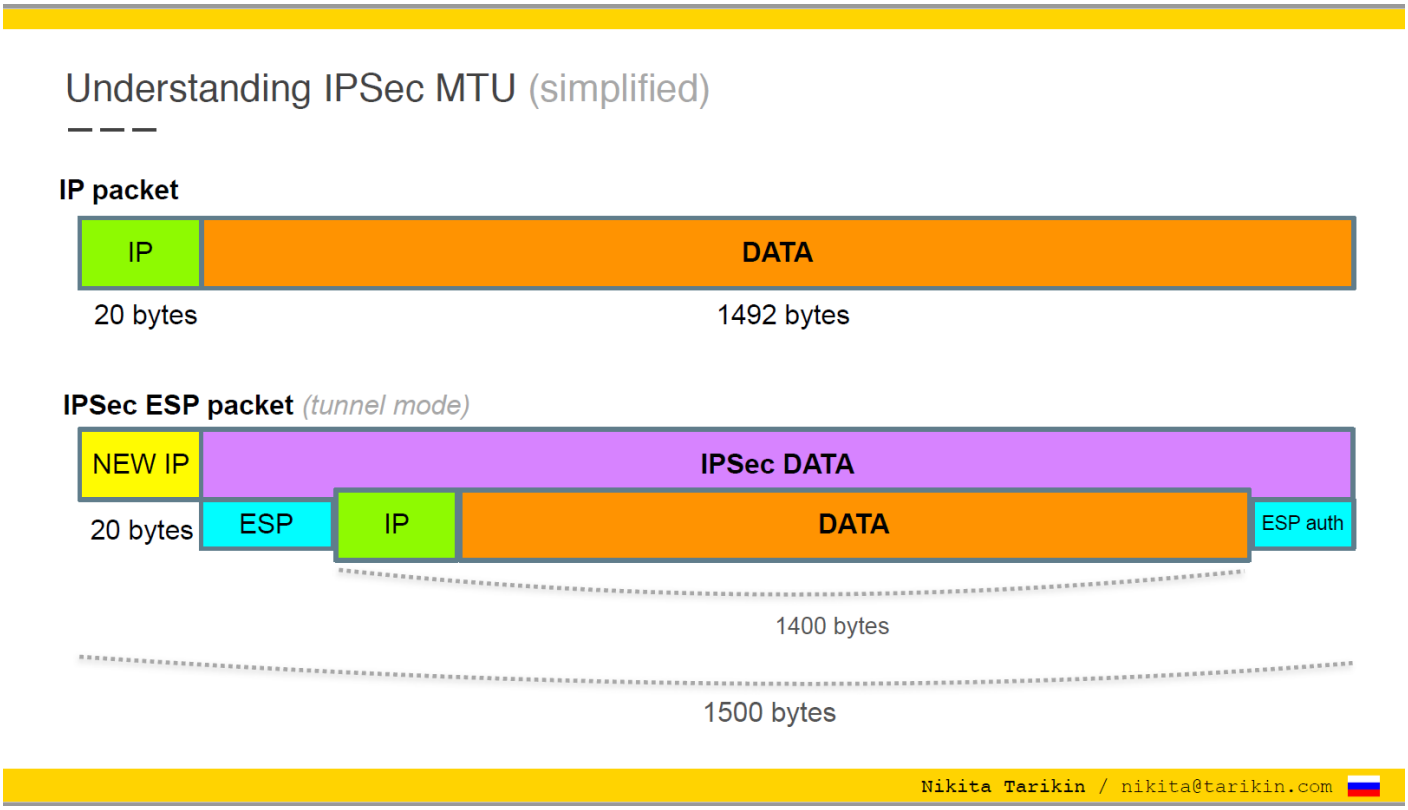
Előfordulhat azonban az, hogy az első fázisban meghatározott állandó kulcshoz valaki valamilyen más módon hozzáfér. Ha ez bekövetkezik, akkor az ebből származtott összes második fázisban használt kulcs meghatározása nagyon egyszerűvé válik. Mivel az első fázisban meghatározott kulcs nagyon hosszú ideig használatban lehet (nincs lejárata!), ezért az állandó kulcs feltörése után visszamenőleg megfejthetővé válik az összes kommunikáció, amit korábban a támdó rögzített. Ez különösen akkor veszélyes, ha az első fázis által meghatározott kulcsok megbízhatóan, leállás nélkül működő csomópontok között, határozatlan ideig kiépített alagút kiépítésére vannak használva. Ennek a problémának kivédésre szolgál a Perfect Forward Secrecy vagy röviden `PFS`. Ennek az a lényege, hogy a második fázishoz használt kulcsot nem az első fázisban meghatározott első kulcsból származtatjuk, hanem egy külön DH algoritmussal határozzuk meg. Ez csökkentheti az átviteli sebességet, és a válaszidőket is növelheti. Főleg akkor, ha a második fázis lejárati ideje rövid, és egyetlen eszközön párhuzamosan egyszerre sok VPN alagutat kell működtetni. Ugyanakkor ez a módszer garantálja azt, hogy a korábban rögzített kommunikáció ne legyen teljes egészében visszafejthető akkor, ha valaki megszerzi az első fázisban meghatározott állandó kulcsot. PFS group használata esetén minden IPSEC SA kulcsát egy külön DH algoritmus futtatással határozzuk meg, aminek semmi köze nincsen az IKE első fázisában használt állandó kulcshoz. Ha bármelyik kulcs feltörésre kerül (például brute force módszerrel), akkor csak a kommunikációnak azon része lesz hozzáférhető, aminek titkosításához a feltört/megtalált kulcsot használták.

Van itt erről egy jó kérdés: https://security.stackexchange.com/questions/196832/which-pfs-group-is-recommended-for-ipsec-configuration

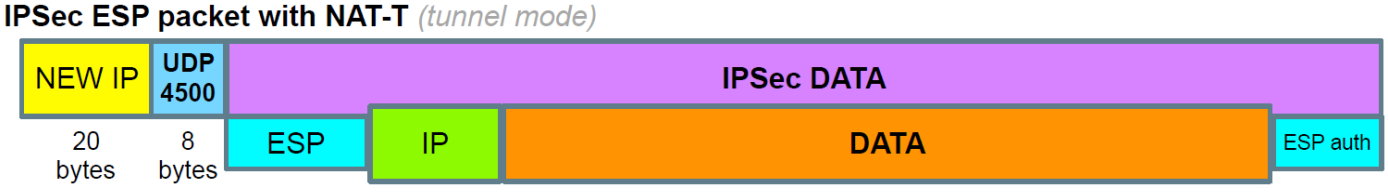
Mi ebben a példában `modp2048` beállítást fogunk használni az első és a második fázishoz is. Ez egy viszonylag erőforrás igényes beállítás. A választást azzal indokoljuk, hogy itt egy site-to-site kapcsolatról van szó. Egyetlen alagutat építünk ki két alhálózat összekötésére, és csak ezt az egy alagutat kell működtetnünk. Ezek a peer-ek megbízhatóan, hosszú időn keresztül működnek. A két alhálózatban levő gépek ugyan ezt az egy alagutat használják megosztva. Így a felépítendő SA-t száma alacsony, viszont az egy SA-n keresztül nagyobb adatmennyiség van lebonyolítva. Ez indokolja a magasabb PFS DH csoport használatát.

MTU és fragmentáció

Amikor az eredeti IP csomagokat IPSEC csomagokba enkapszuláljuk, akkor természetesen az IPSEC csomagok mérete nagyobb lesz, mint az eredeti IP csomagok mérete. Hogy pontosan mennyivel nagyobb, azt ismét Nikita Tarikin előadásáról származó ábrával mutatom be. Az alábbi ábra egy rendes IP csomag illetve egy IPSec ESP csomag felépítését mutatja (tunnel módban):



Ha a tunnel módon felül még NAT-T beállítást is használunk, akkor további 8 byte-ot elvesz az IP fejléc és az IPSEC adat blokk közé beszúrt extra UDP fejléc:



Ethernet (layer 2) szinten a maximális csomagméret általában 1500 byte. Egy normál IPv4 csomag esetében az IPv4 header 20 byte-ot foglal el. (Ebben van verziószám, TTL érték, al-protokol száma, forrás- és célcím stb. Részletek itt találhatók. Emiatt egy ethernet-en keresztül küldött normál IP csomagban a hasznos adat (payload) mérete legfeljebb 1492 byte lehet.

Ha megnézzük az alagút módban használt IPSEC/ESP csomagot akkor azt látjuk, hogy az eredeti (teljes) beágyazott és titkosított IP csomagon felül még el kell tárolnunk az ESP fejlécet, az ESP tail-t (amiben pl. a csomag integritását biztosító ellenőrző összeg szerepel). Ezen felül egy új IP header is bekerül az elejére (amiben az sa-src-address és az sa-dst-address közötti forgalmazáshoz szükséges). Ha pedig még NAT-T módot is használunk, akkor még újabb 8 byte-ot föl kell használunk.

Ha az eredeti ethernet csomag mérete 1500 byte volt, és az IPSEC/ESP csomag (ethernet) mérete is legfeljebb 1500 byte lehet, akkor az enkapszulációt természetesen nem lehet elvégezni. Egyszerűen azért, mert az eredeti csomag nem fér bele a legnagyobb méretű IPSEC csomagba sem. IPv6 esetében még rosszabb a helyzet, az IPv6 header mérete ugyanis 40 byte.

Amikor a router ilyen feladattal szembesül, akkor az eredeti IP csomagot töredékekre bontja (idegen szóval `fragmentálja`), és ezeket a töredékeket külön IPSEC csomagokban továbbítja. A távoli peer ezeket a töredékeket fogadja, egyesével kicsomagolja, és ezekből állítja elő az eredeti csomagot. A távoli peer addig nem tudja helyreállítani az eredeti csomagot, amíg az összes töredék meg nem érkezik. Ezért tárolnia kell őket egy ideig. Előfordulhat, hogy nem érkezik meg minden töredék. (Az IPSEC csomagok datagram típusúak, nincs arra garancia hogy megérkeznek.) Ilyenkor a távoli peer eldobja a használhatatlan töredékeket. De ezt csak egy idő után teszi meg, és addig is fölöslegesen tárolja őket. A fragmentálás jelentősen rontja a továbbított és a hasznos adatmennyiség arányát. A helyzet tovább romlik akkor, ha egy adott útvonalon több helyen kell fragmentálni. A többszörös fragmentáció hatására a kapcsolat átviteli sebessége jelentősen csökkenhet, a válaszidők jelentősen növekedhetnek, és az átvitelben résztvevő eszközök terhelése növekedik. Többszörös fragmentáció előfordulhat például akkor, ha az útvonalon van egy PPPoE alagút, vagy ha egy nem optimális router az IPSEC csomagokat újabb IPSEC csomagokba enkapszulálja stb.

MTU Path Discovery

A fenti okokból a fragmentációt lehetőség szerint el kell kerülni! Ebből a célból használják a Path MTU Discovery (rövidítve `MTUPD`) nevű eljárást. Ez máshogy működik IPv4 és IPv6 esetén. Azonban mindkettőre igaz, hogy kizárólag kapcsolatállapot alapú protokollokkal működik. Az ilyen protokollok a kommunikációt úgy végzik el, hogy először kiépítenek egy utat (path), és az adatokat ezen az úton át küldik el egymásnak. A csomagkapcsolt (datagram) protokollok esetén az MTU Path discovery nem működik.

- IPv4 esetében úgy működik az MTUPD, hogy a kiküldött IP csomagnál a küldő fél beállítja a `DF` (don't fragment) bitet. Bármely eszköz ami egy ilyen csomagot nem tud fragmentálás nélkül továbbítani, eldobja a csomagot és visszaküld egy "Fragmentation eeded" ICMP csomagot a feladónak. Ebben az üzenetben szerepel az elérhető legnagyobb MTU is. A küldő fél ezzel az új MTU-val próbálja újra a csomag küldését. Ez a folyamat addig ismétlődik, amíg a csomag el nem jut (fragmentálás nélkül) a célcímhez.
- Az IPv6 router-ek nem támogatják a fragmentálást, és az IPv6 fejlécben nincsen `DF` bit. Az IPv6 protokoll esetében az MTU Path discovery úgy működik, hogy eredetileg azt feltételezzük, hogy az útvonal MTU-ja ugyan az, mint a link MTU-ja. (A link MTU-ját az a

helyi interface adja meg, amin keresztül a küldő fél kiküldi a csomagot.) Ezután az IPv4 protokollhoz hasonlóan, bármely eszköz ami a túl nagy méret miatt nem tudja továbbítani a csomagot, visszaküld egy ICMPv6 "túl nagy csomag" üzenetet a feladónak.
- Ha egy korábban kialakított kapcsolatban csökken az MTU értéke, akkor az első túl nagy csomag ICMP hibát okoz, és a küldő fél automatikusan ennek megfelelően csökkenti az kiküldött csomagméreteket. Tehát az MTUPD képes dinamikusan alkalmazkodni a path MTU megváltozásához.
- Hasonlóan, ha a kialakított kapcsolatban növekedik az MTU értéke, akkor az operációs rendszer ezt felismeri, és módosítja a kapcsolat MTU-ját. Ezt az OS úgy éri el, hogy időnként újrapróbálkozik nagyobb méretű ICMP DF csomagokkal. Az újrapróbálás ideje Linux és Windows operációs rendszerek esetében 10 perc.

Sajnálatos módon az MTUPD algoritmus nem mindig működik megfelelően. Ennek az az oka, hogy egyes eszközök letiltják az ICMP csomagok fogadását és/vagy továbbítását. (Általában biztonsági megfontolásokból.) Nem lehet előre megmondani, hogy a két telephelyet összekötő útvonalon van-e (vagy lesz-e a jövőben) olyan eszköz, ami megakadályozza az ICMP csomagok átjutását, és ezzel ellehetetleníti az MTU automatikus meghatározását.

TCP MSS Clamping

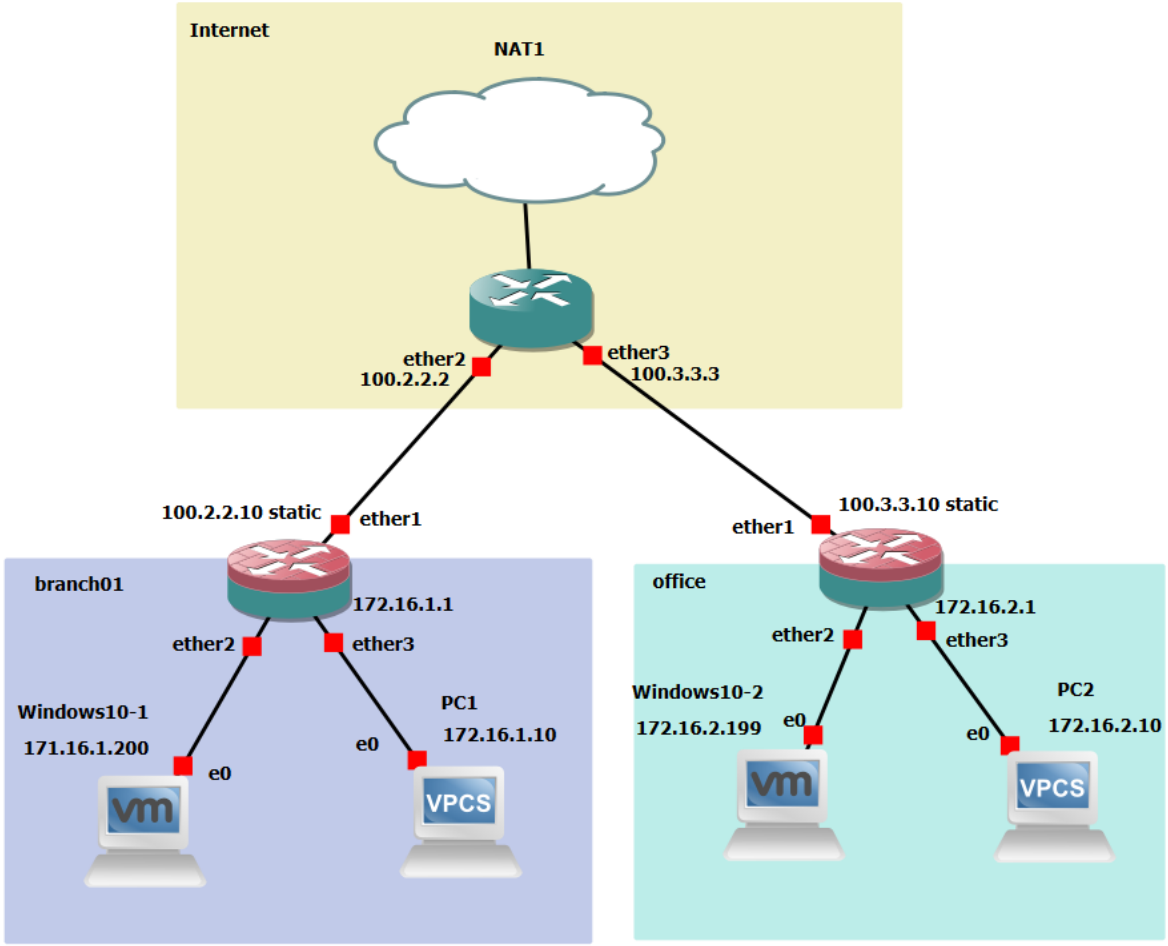
A TCP kapcsolatok esetében elterjedt megoldás a fenti probléma kiküszöbölésére a TCP maximális szegmens méret csökkentése vagy más néven `MSS Clamping`. A TCP MSS csökkentése közvetve csökkenti a TCP csomagok maximális méretét, ezáltal képes csökkenteni a fragmentációt. Ez a következő módon működik. A TCP protokoll a kapcsolat kialakítását az úgynevezett `SYN` csomagok elküldésével kezdi. Ezek tartalmazzák az egy TCP csomagban elküldhető maximális adat méretet, más néven `MSS`-t. Amikor egy `TCP SYN` csomag áthalad egy olyan eszközön, ami tudja magáról hogy nem képes az alapértelmezett 1500 byte -os ethernet frame fragmentálás nélküli továbbítására, akkor **módosítja** a továbbítandó `TCP SYN` csomagban található `MSS` értéket egy alacsonyabb értékre. (Hogy pontosan mi a megfelelő érték, arról később írok). Ezzel a módszerrel TCP kapcsolatok esetén az adott eszközön egész biztosan el lehet kerülni a fragmentációt. Ez a módszer az MTUPD-től teljesen független módon működik, és bármely TCP kapcsolatra alkalmazható.

Az MTU érték magasan tartása növeli a hatékonyságot. Emiatt sok szabályt dolgoztak ki, amivel az overhead-et (header/adat arányt) alacsonyan lehet tartani. Az IPSEC/ESP csomagok overhead-je függ az átviteli módtól (transzport/alagút), a NAT-T beállítástól (kell-e extra UDP header-t beszúrni), a használt titkosítási algoritmustól (például a cipher blokk méretétől) stb. Ráadásul a használt titkosítási algoritmust nem lehet mindig előre rögzíteni. Ha például sok különböző kliens kapcsolódik és különböző proposal-okat használnak, akkor az IKE démon számos különböző algoritmus kombinációt elfogadhat. Ennek bonyolultságával kapcsolatban némi támpontot adhat ez a weblap. Itt látható, hogy például a paddig értéke egy byte-tal kevesebb mint a cipher blokk mérete (legrosszabb eset). Illetve első pillantásra furcsának tűnhet az SHA1 HMAC értékhez beírt 96 bit, de jobban utánanézve kiderül, hogy bizonyos algoritmus kombinációknál nem a teljes HMAC ellenőrző összeget írják be a csomagba, hanem annak csak egy részét. Az utólag hozzáadott újabb

típusú algoritmusokra (SHA256+) további RFC-k használhatók. Lehet találni a neten mindenféle MTU kalkulátorokat, azonban ezek nem mindig képesek kiszámítani a megfelelő MTU értéket az általad használt algoritmusokra (feltéve hogy előre ismered őket!), és kétséges a megbízhatóságuk.

Sokkal célravezetőbb az empirikus módszer. Az empirikus módszerhez az szükséges, hogy legyen az alagút mindkét oldalán egy elérhető, ping-elésre képes gép, valamint hogy a ping üzenetek átmenjenek az egyik oldalról a másikra. Windows operációs rendszer alatt erre a a `ping` parancs a `-f` és `-l` kapcsolókkal használható. A `-f` kapcsoló jelentése: do not fragment. A `-l` kapcsolóval lehet megadni (növelni) a csomag méretét.

Az empirikus módszert a feladatként meghatározott site-to-site VPN kapcsolat teszt hálózatán végezzük el.



Először egy olyan útvonalon ping-elünk, ahol nincsenek alagutak. Például a branch01 -ben levő Windows10-1 gépről pingeljük az ISP router-ének 10.2.2.2 címét. Kezdetben 1472 byte mérettel próbálkozunk. Az 1472 úgy jön ki, hogy egy ICMP ping csomagban 20 byte IP header és 8 byte ICMP header van. A teljes (alapértelmezett) 1500 byte-os ethernet keretből így 1472 byte marad:

```
C:\Users\User>ping 10.2.2.2 -n 3 -f -l 1472

Pinging 10.2.2.2 with 1472 bytes of data:
Reply from 10.2.2.2: bytes=1472 time<1ms TTL=63
Reply from 10.2.2.2: bytes=1472 time<1ms TTL=63
Reply from 10.2.2.2: bytes=1472 time<1ms TTL=63

Ping statistics for 10.2.2.2:
    Packets: Sent = 3, Received = 3, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Users\User>ping 10.2.2.2 -n 3 -f -l 1473

Pinging 10.2.2.2 with 1473 bytes of data:
Packet needs to be fragmented but DF set.
Packet needs to be fragmented but DF set.
Packet needs to be fragmented but DF set.

Ping statistics for 10.2.2.2:
    Packets: Sent = 3, Received = 0, Lost = 3 (100% loss),

C:\Users\User>
```

Ahogy látható, az 1472 byte méret átment, az 1473 byte méretre "packet needs to be fragmented but DF set" ICMP hibaüzenetet kaptunk. Ezzel megtudtuk azt, hogy az internet felé menő normál (nem alagutazott) útvonal milyen MTU értékkel rendelkezik. Jelen esetben ez ethernet (layer 2) szinten 1500. Ezután elkezdjük ping-elni az alagút másik oldalán levő valamelyik (bármelyik) gépet egy olyan csomagmérettel, amiről még úgy gondoljuk, hogy fragmentáció nélkül átmegy az alagúton:

```
C:\Users\User>ping 172.16.2.10 -n 1 -f -l 1400

Pinging 172.16.2.10 with 1400 bytes of data:
Reply from 172.16.2.10: bytes=1400 time=1ms TTL=62

Ping statistics for 172.16.2.10:
    Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 1ms, Maximum = 1ms, Average = 1ms
```

Tehát most már tudjuk, hogy az MTU érték valahol 1472 és 1400 között van. A pontos értéket felező módszerrel viszonylag gyorsan meg tudjuk határozni:

```
C:\Users\User>ping 172.16.2.10 -n 1 -f -l 1430

Pinging 172.16.2.10 with 1430 bytes of data:
Packet needs to be fragmented but DF set.
```

```
C:\Users\User>ping 172.16.2.10 -n 1 -f -l 1415

Pinging 172.16.2.10 with 1415 bytes of data:
Packet needs to be fragmented but DF set.
```

```
C:\Users\User>ping 172.16.2.10 -n 1 -f -l 1407

Pinging 172.16.2.10 with 1407 bytes of data:
Reply from 172.16.2.10: bytes=1407 time=1ms TTL=62
```

```
C:\Users\User>ping 172.16.2.10 -n 1 -f -l 1411

Pinging 172.16.2.10 with 1411 bytes of data:
Packet needs to be fragmented but DF set.
```

```
C:\Users\User>ping 172.16.2.10 -n 1 -f -l 1408

Pinging 172.16.2.10 with 1408 bytes of data:
Reply from 172.16.2.10: bytes=1408 time=3010ms TTL=62
```

```
C:\Users\User>ping 172.16.2.10 -n 1 -f -l 1409

Pinging 172.16.2.10 with 1409 bytes of data:
Reply from 172.16.2.10: bytes=1409 time=1ms TTL=62
```

```
C:\Users\User>ping 172.16.2.10 -n 1 -f -l 1410

Pinging 172.16.2.10 with 1410 bytes of data:
Reply from 172.16.2.10: bytes=1410 time=1ms TTL=62
```

```
C:\Users\User>ping 172.16.2.10 -n 1 -f -l 1411

Pinging 172.16.2.10 with 1411 bytes of data:
Packet needs to be fragmented but DF set.
```

Alagúton legfeljebb 1410 byte adat ment át, alagút nélkül pedig 1472 byte. A kettő között a különbség 1472-1410 = 62 byte. Tehát ennyivel kell csökkenteni az MSS értékét ahhoz, hogy elkerüljük a TCP csomagok fragmentációját.

Egy normál 1500 byte-os TCP/IP csomag felépítése olyan, hogy az IP fejléc elvesz 20 byte-ot, és a TCP fejléc elvesz még 20 byte-ot. Ezért a teljes 1500 byte-os ethernet keretben az MSS értéke nem haladhatja meg az 1460 byte-ot. Az általunk felépített konkrét alagút ennél 62 byte-tal kisebb helyet biztosít. Ezért a fragmentáció nélkül továbbítható legnagyobb MSS érték 1460 - 62 = 1398 byte.

Ezt a számítás egy sorba kiírva:

TCP MSS = 1460 - (1472 - <mért ICMP MTU>) = **<mért ICMP MTU> - 12**

> Mindig az 1472 byte-os alap értékhez viszonyítunk. Ha például az internet kapcsolatod PPPoE-t használ, akkor az első tesztben azt fogod tapasztalni, hogy az ICMP MTU a VPN

> alagút nélkül 1472 alatt van. Ettől függetlenül a TCP MSS mindig 12 byte-tal kevesebb lesz, mint az általad mért maximuális ICMP ping méret. Ez azért van így, mert a TCP csomag elméleti maximális 1460-as értéke is az 1500 byte-os full ethernet frame-re vonatkozik.

Bár ez a módszer egy kicsit fáradságos, viszont sokkal megbízhatóbban meghatározza a valós MTU értéket, mint egy elméleti számítás.

A fent leírt módszer akkor is célra tud vezetni, ha az útvonalon található router-ek közül valamelyik blokkolja az ICMP üzenetek visszaküldését. Ha a visszajövő ICMP üzenetek blokkolva vannak, akkor az MTUPD biztosan nem működik. Azonban ICMP alapú ping helyett lehet használni UDP vagy TCP alapú pinget. (Erre vannak programok.) Ha nem érkezik vissza válasz a TCP vagy UDP alapú, megnövelt méretű pingre (timeout) akkor ez feltételezhetően azért azért van, mert fragmentáció nélkül nem lehetett átküldeni a csomagot. Tehát a "packet needs to be fragmented but DF set" üzenetet ilyenkor a timeout helyettesítheti (de persze csak akkor, ha van olyan csomagméret, aminél jön vissza ping válasz!)

Ha sokféle algoritmus engedélyezel, és nem vagy biztos abban, hogy az enkapszuláció és az alagutazás pontosan mekkora overhead-et okoz, akkor viszonylag biztonságosan használhatod az 1360 byte-os MSS értéket. Ebbe bőven belefér a NAT-T mód által beszúrt extra 8 byte-os UDP header, és a legerősebb titkosítási algoritmusok overhead-je is. Bár az igaz, hogy ez így nem mindig optimális, de egész biztosan nem okoz fragmentációt. (Annál minden jobb.)

TCP MSS Clamping RouterOS-ben

Tehát olyan szabályt veszünk föl, ami képes lecsökkenteni a `TCP SYN` csomagok 1360 byte-nál nagyobb `MSS` értékét 1360-ra. Nagyon fontos hogy olyan szabályt vegyünk föl, ami nem képes növelni az MSS értékét. Ha a csomag már eleve 1360 -nál kisebb MSS értékkel érkezik be, és ezt felülírjuk 1360-ra, akkor ezzel **megnöveljük azt**, és így végül pont az ellenkezőjét érjük el annak, mint amit szeretnénk (növeljük a fragmentációt ahelyett hogy csökkentenénk).

A megfelelő szabály az office gépen így néz ki:

```
/ip firewall mangle add action=change-mss chain=forward new-mss=1360 src-address=172.16.1.0/24
protocol=tcp tcp-flags=syn tcp-mss=!0-1360 ipsec-policy=in,ipsec passthrough=yes
comment="IKE2: Clamp TCP MSS from 172.16.1.0/24 to ANY"
```

A branch01 gépen pedig így:

```
/ip firewall mangle add action=change-mss chain=forward new-mss=1360 src-address=172.16.2.0/24
protocol=tcp tcp-flags=syn tcp-mss=!0-1360 ipsec-policy=in,ipsec passthrough=yes
comment="IKE2: Clamp TCP MSS from 172.16.2.0/24 to ANY"
```

Egy kis magyarázat hozzá:

- Az `src-address`, `ipsec-policy`, `protocol=tcp` és `tcp-flags=syn` együttes használatával kiszűrjük azokat a `TCP SYN` csomagokat, amik **már átjöttek** az alagúton. (Az `src-address` szűrőben az office router szabályánál ezért szerepel a branch oldal alhálózata.) Ez a szabály így azért jó, mert garantáltan csak azokra a TCP kapcsolatokra módosítja az MSS-t, amik már átjöttek az alagúton. (Azokra nem szeretnénk módosítani, ami nem megy át alagúton!)
- A `tcp-mss=!0-1360` szabály szó szerint azt jelenti, hogy "a tcp mss értéke nem nulla és 1360 között van". Ez könnyebben értelmezhető formában annyit tesz, hogy nagyobb mint 1360.
- A `passthrough=yes` azért került oda, mert az MSS megváltoztatása után nem szeretnénk átugrani a többi mangle szabályt. (Bár a jelenlegi példában nincsen több mangle szabály, de ha lennének akkor valószínűleg nem akarnánk átugrani őket.)