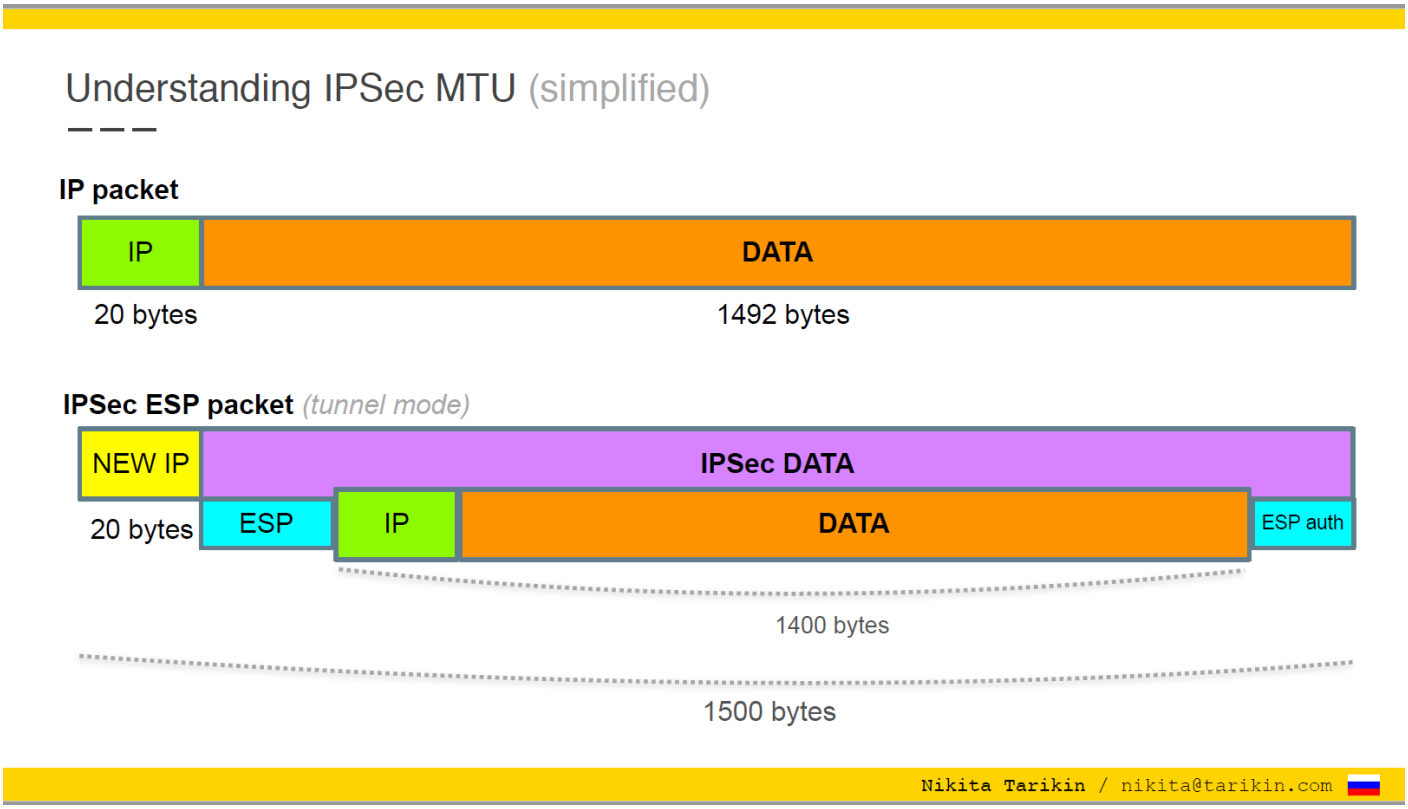
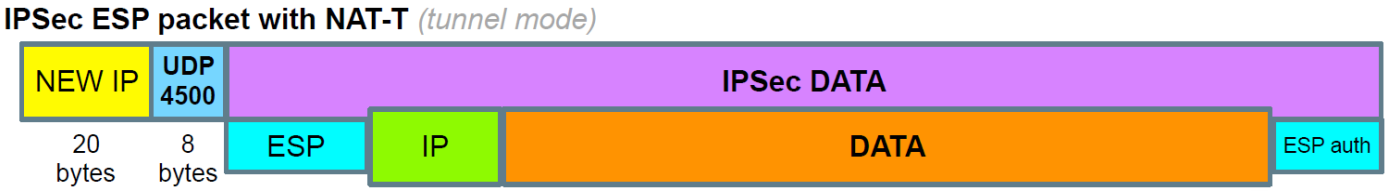
MTU és fragmentáció

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

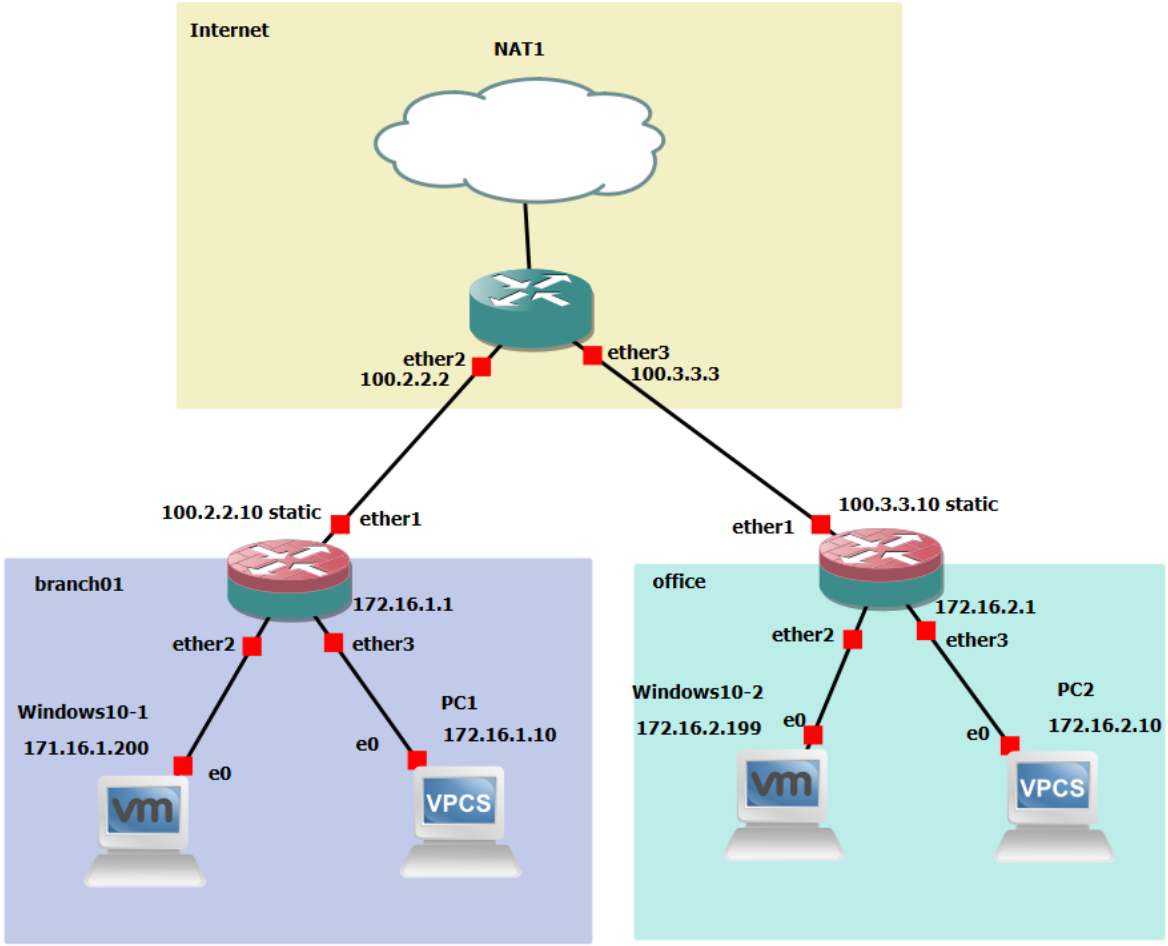
A TCP kapcsolatok esetében elterjedt megoldás a fenti probléma kiküszöbölésére a TCP maximális szegmens méret csökkentése vagy más néven `MSS Clamping`. A TCP MSS csökkentése közvetve csökkenti a TCP csomagok maximális méretét, ezáltal képes csökkenteni a fragmentációt. Ez a következő módon működik. 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.)

Revision #4
Created 9 January 2021 11:50:25 by Gandalf
Updated 9 January 2021 13:12:41 by Gandalf