Windows Server 2008

TCP/IP Alapok

2. kötet

V1.0

Petrényi József

© 2010, Petrényi József

1.0 verzió, első kiadás

Minden jog fenntartva.

A könyv írása során a szerző és a kiadó a legnagyobb gondossággal és körültekintéssel igyekezett eljárni. Ennek ellenére előfordulhat, hogy némely információ nem pontos vagy teljes, esetleg elavulttá vált.

Az algoritmusokat és módszereket mindenki csak saját felelősségére alkalmazza. Felhasználás előtt próbálja ki és döntse el saját maga, hogy megfelel-e a céljainak. A könyvben foglalt információk felhasználásából fakadó esetleges károkért sem a szerző, sem a kiadó nem vonható felelősségre.

A cégekkel, termékekkel, honlapokkal kapcsolatos listák, hibák és példák kizárólag oktatási jelleggel kerülnek bemutatásra, kedvező vagy kedvezőtlen következtetések nélkül.

Az oldalakon előforduló márka- valamint kereskedelmi védjegyek bejegyzőjük tulajdonában állnak.

Microsoft Magyarország 2010

Köszönetnyilvánítás:

Továbbra is hatalmas köszönet illeti Joseph Davies-t, alias Cable Guy-t az alapos, szemléletformáló írásaiért. A wikipedia most sem hazudtolta meg önmagát, mindenhez hozzá tudott szólni, igaz, nem mindig sikerült érdemben. De becsületesen próbálkozott.

"- Felejtsük el az egészet, kedves Tót - mondta nagylelkűen -, és lássunk hozzá a dobozoláshoz. Minden percért kár.

Leültek. Tót is. Ugyanaz a Tót, aki az imént még lefitymálta és asszonypepecselésnek nézte ezt a munkát, most örült, hogy dobozolhatott... Pedig senki se hívta; épp csak, hogy helyet szorítottak neki. Persze, akárhogy vigyázott, csupa félresikerült, pofoncsapott doboz került ki a keze alól, de szerencsére ezen se akadt föl senki, legföljebb elnézően összemosolyogtak.

Helyreállt a béke. Hosszú negyedórákig senki se beszélt, csak a margóvágó friss kattogása hallatszott.

Később friss levegő jött a hegyekből. Szemközt, a Bábony tisztásain a gyantaszüretelők tűzrakásai hunyorogtak. Tóték ezt se látták. Mindenről elfeledkezve vágták és hajtogatták a dobozokat. Egy óra múlva az őrnagy udvariasan érdeklődött:

- Nem álmosodtak el?

Tót, akinek majd leragadt a szeme, megnyugtatta az őrnagyot, hogy esze ágában sincs lefeküdni.

Tovább dobozoltak. Egy idő múlva az őrnagy megismételte az előbbi kérdést. Tóték egybehangzóan azt állították, hogy nem álmosak.

A harmadik kérdésre Mariska, akinek bal szeme erősen viszketni kezdett, azt válaszolta, hogy ha az őrnagy úr pihenni vágyik, akkor ők is készek abbahagyni a munkát.

- Dehogyis vágyom pihenni! - mondta az őrnagy. - Sajnos, nagyon rossz alvó vagyok.

- Ettől az éles hegyi levegőtől még álmatlanságban szenvedő vendégeink is elálmosodtak - jegyezte meg Tót.

- Nekem az se használ - legyintett Varró őrnagy. - Én a legszívesebben reggelig hajtogatnám a dobozokat.

A koránfekvéshez szokott Tót kezében szétroppant egy doboz. Arcvonásai a fáradtságtól amúgy is összezilálódtak; ijesztő tekintettel meredt az őrnagyra. Ekkor azonban Mariska bokán rúgta, amire Tót - nagy erőfeszítéssel - elmosolyodott.

- Én is csak most kezdek belejönni - mondta.

És tovább dobozoltak. "

Örkény István: Tóték

TARTALOMJEGYZÉK

1 Bevezető 8
2 Az alkalmazás réteg webes protokolljai 9
2.1 HTTP - Aki lenyomta a pockot 9
2.1.1 HTTP Request 20
2.1.1.1 Request-Line 21
2.1.1.2 Message Header (Request) 22
2.1.2 HTTP Response 23
2.1.2.1 Státusz kódok 24
2.1.2.2 Message Header (Response) 26
2.2 FTP - A teherhordó öszvér 28
2.2.1 FTP parancsok 29
2.2.2 FTP kódok 32
2.2.3 Az FTP protokoll lelkivilága 33
2.2.3.1 Aktív mód 34
2.2.3.2 Passzív mód 36
2.2.3.3 FTP és NAT 38
2.3 SMTP - A népszerű öreg 40
2.3.1 Levelek továbbítása, azaz a konkrét SMTP 41
2.3.1.1 SMTP, ESMTP parancsok 45
2.3.1.2 SMTP kódok 46
2.3.2 Az elektronikus levelek szerkezete, azaz IMF 48
2.3.3 Levelezzünk már végre! 68
2.4 Porttáblázat 77
2.5 Sztorik 81
2.5.1 Geek úr nyaral 81
2.5.2 Geek úr a fogorvosnál 88
3 A biztonság kérdése a TCP/IP-ben 90
3.1 SSL, TLS 90
3.2 SSH 95

3.3 Az ismertebb protokollok biztonságosabbá tétele 97
3.3.1 HTTP 97
3.3.1.1 Autentikáció 97
3.3.1.2 HTTPS 103
3.3.1.3 SHTTP 109
3.3.2 FTP 111
3.3.2.1 FTPS 111
3.3.2.2 FTP over SSH 113
3.3.2.3 A bájos félreértések forrása - SFTP 113
3.3.3 A többiek, azaz a STARTTLS 114
3.4 IPSec 116
3.4.1 Authentication Header 117
3.4.1.1 AH Transport mód 118
3.4.1.2 AH Tunnel mode 119
3.4.2 Encapsulating Security Payload 120
3.4.2.1 ESP Transport mód 121
3.4.2.2 ESP Tunnel mód 123
3.4.3 Security Associations 124
3.4.3.1 Main mode (ISAKMP SA) 124
3.4.3.2 Quick mode (IPSec SA) 128
3.4.3.3 IKE 128
3.4.3.4 IKE v2 129
3.4.3.5 AuthIP 131
3.4.4 Összefoglalás 132
3.5 VPN és társai 133
3.5.1 PPTP 136
3.5.2 L2TP 137
3.5.3 L2TP / IPSec 138
3.5.4 SSTP 139
3.5.5 VPN Reconnect 142
3.5.6 Összefoglalás 145
3.6 Autentikáció 146

3.6.1 RADIUS 147
3.6.2 Kétfaktoros autentikáció 151
4 Kivezetés 155
5 Források, linkek 156
6 Javítások 159
7 A szerző 160

1 BEVEZETŐ

Rendhagyó könyv lesz[1].
Ugyanarról a témáról fogok írni még egy könyvet: TCP/IP protokollok, szolgáltatások.
Csakhogy eddig vertikálisan jártuk be a teret: elindultunk alulról és onnan másztunk fel a legfelső szintre. Most viszont eleve a legfelső szintről indulunk és maximum egy szintet lépünk lejjebb. Ez a könyv ugyanis szinte kizárólag a roppant gazdagon vegetáló alkalmazás rétegről szól. (Csak az IPSec kedvéért fogunk lenézni egyszer az IP réteg szintjére.)

Hogyan választottam ki a bemutatandó protokollokat? Web. Internet. Ma már minden ekörül forog. Logikusnak tűnt azokat a protokollokat összeszedni, melyek a legelterjedtebbek, illetve legfontosabbak a neten.
Aztán ha már internet: mi a legnagyobb baj ezekkel a protokollokkal? Hát a biztonság, illetve annak hiánya. Hogyan lehet ~~sárból várat építeni~~ alapban gyenge biztonságú protokollokból megbízható kapcsolatokat építeni? Jó kérdés. Lehet. Nem egyszerű, de nem lehetetlen.

Ilyesmikkel fogok foglalkozni ebben a könyvben.

A kapcsolódó első kötet, illetve az első kötet összefoglaló füzete letölthető innen:
http://mivanvelem.hu/letoltheto-konyvek/

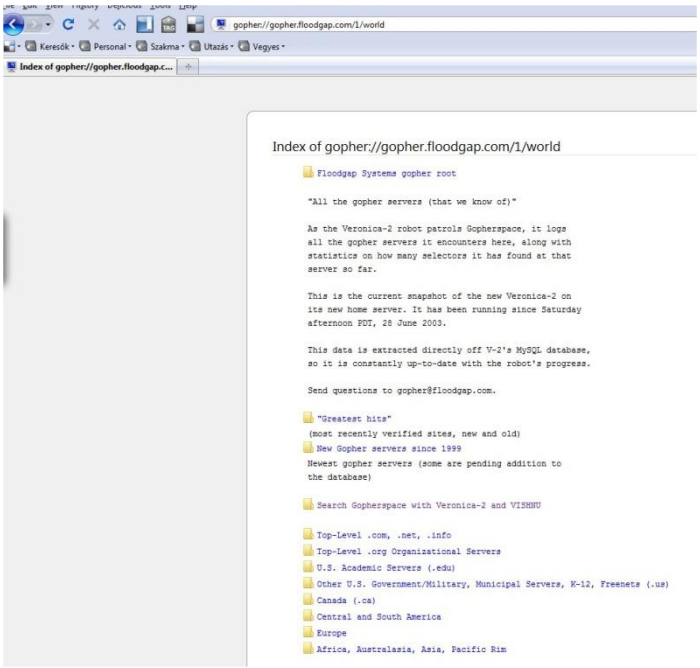
[1] Mondtam. (-:

2 AZ ALKALMAZÁS RÉTEG WEBES PROTOKOLLJAI

2.1 HTTP - AKI LENYOMTA A POCKOT

RFC 2616

Bármennyire is furcsa, de volt olyan időszak, amikor a web nem volt egyenlő a HTTP-vel. Még emlékszem arra, amikor BBS-ek voltak, később FTP szerverek, aki pedig hipertextben utazott, az bőszen nyomta a pockot - azaz a gopher protokollt.



2.1. ÁBRA EGY GOPHER OLDAL

http://en.wikipedia.org/wiki/Gopher_(protocol)

Aztán 1993-ban beindult a HTTP és szép lassan győzött. A gopher ugyanis nem volt képes a szöveg mellett grafikát is megjeleníteni.

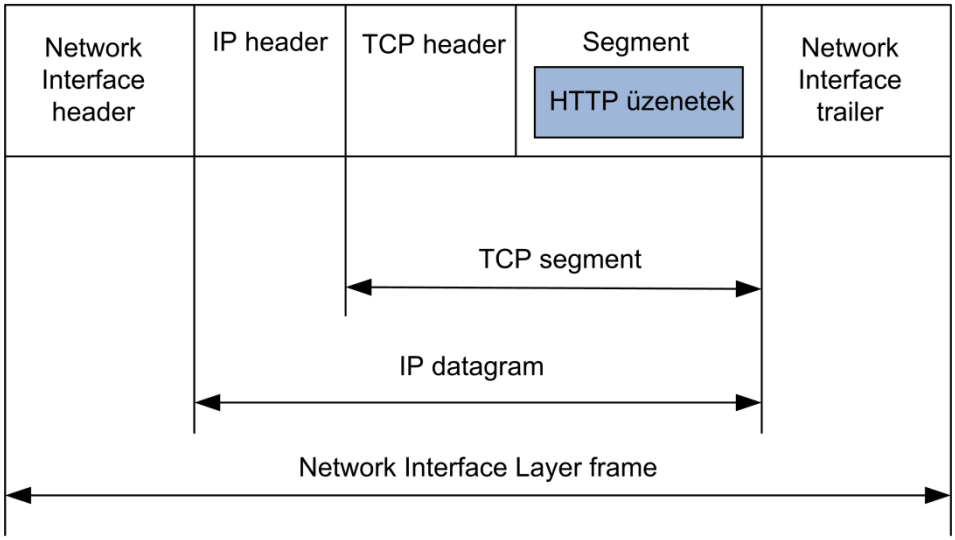
De mit is tud ez a HTTP? Hypertext Transfer Protocol. Azaz amennyiben a kliens oldali böngészőprogramból kérés érkezik egy webszerverhez, az olyan szövegeket képes visszaküldeni, melyekből a kliensgépeken weboldalak állnak össze: linkekkel, képekkel, formázott szövegekkel - és manapság már videókkal is. Úgy nagyjából ezzel le is fedtem most az internet 95%-át. (Warez és pornó nélkül a 10%-át.)

Az előbb elhangzott két nagyon fontos kifejezés. Nem, nem a warez és a pornó.

Az, hogy kliens és szerver.

A HTTP egy olyan protokoll, mely mindig egy kliens és egy szerver között működik. A HTTP szerver (ma már egyszerűen csak webszerver) egy olyan számítógép, mely tele van weblapokkal. Ezek lehetnek statikusak, vagy egy alkalmazás által - legtöbbször valamilyen adatbázisra támaszkodva - dinamikusan generáltak.
A kommunikáció mindig úgy kezdődik, hogy jön egy HTTP kliens (a legtöbbször valamelyik böngésző) és elkér ebből a sok weblapból egyet. A reklámlobby szerencsére még nem olyan erős, hogy kérés nélkül toljanak le a gépedre egy hüvelygomba reklámot[2].
A szerver fogadja a kérést, értelmezi... majd visszaküldi a kért weblapot.

Eddig nem győztük rétegezni a hálózatos kommunikációt. Hol illeszkedik bele ebbe a modellbe a HTTP?

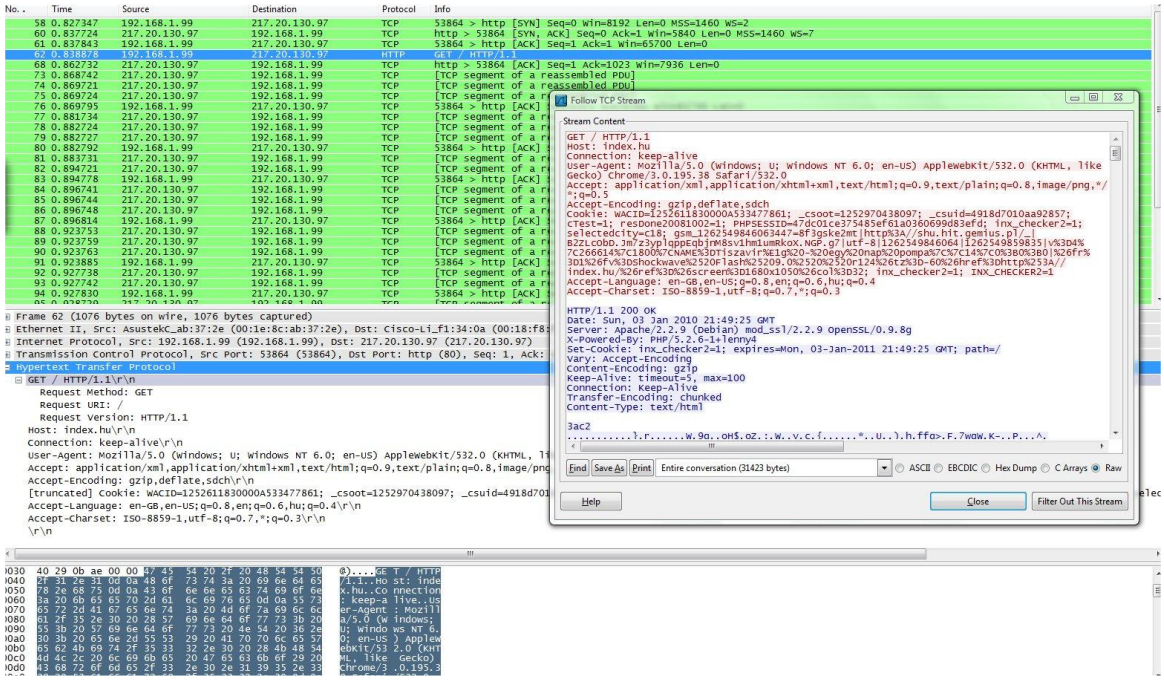


2.2. ÁBRA A HTTP CSOMAGOK HELYE

Mint minden tisztességes alkalmazás-rétegbeli protokoll, a csomagjai a szállítási protokoll (TCP) payload-ján belül utaznak. A kliens ide tolja be a kérését, a webszerver innen veszi ki és ide rakja vissza a választ.

Kicsit előrerohanok.

[2] Az más kérdés, hogy ha lekérsz egy weblapot, akkor már tolják mellé a hirdetést is. De ehhez először neked kellett kérned.



2.3. ÁBRA LEKÉRTEM AZ INDEX NYITÓLAPJÁT

Ez egy olyan kép lesz, melyet a következőkben sokszor fogunk még nézegetni. Most csak azt tessék észrevenni, hogyan is zajlott a kommunikáció:

- Az 58, 60, 61 csomagokban megtörtént az úgynevezett hármas kézfogó, a két fél kiépítette a TCP csatornát.
- A 62-es csomagban a kliens (192.168.1.99) elküldte a kérését. Ezt a kérést látjuk kirészletezve is. A középső ablakban látszik, hogy egy GET kéréssel indul, és a teljes kérés elfér egy TCP szegmensben. Az alsó ablakban bináris formában is megtekinthetjük, sőt, a Wireshark van olyan úr, hogy a teljes kérdés/felelet párost szöveges formában is megmutatja. Mi több, le is tudjuk menteni. (Ezzel azért érdekes trükköket lehet játszani.)
- A 68-as csomagtól szabadult el a pokol, a webszerver nekiállt küldeni az index.hu nyitólapját. Szép nagy nyitólap, elég sok TCP szegmenst kellett befognia a szállítási munkálatokba.
- Habár a képen nem látszik, de hidd el nekem, a 114, 119, 120 csomagokban mindkét fél mindkét oldalról lebontotta a TCP csatornát.

Most alapvetően két úton indulhatunk el. Egyrészt, ragaszkodva a hagyományokhoz, nekiállhatunk request/response csomagokat boncolni, biteket elemezni, flageket értelmezni. Ez akkor korrekt, ha tényleg csak a HTTP protokollt szeretném bemutatni.

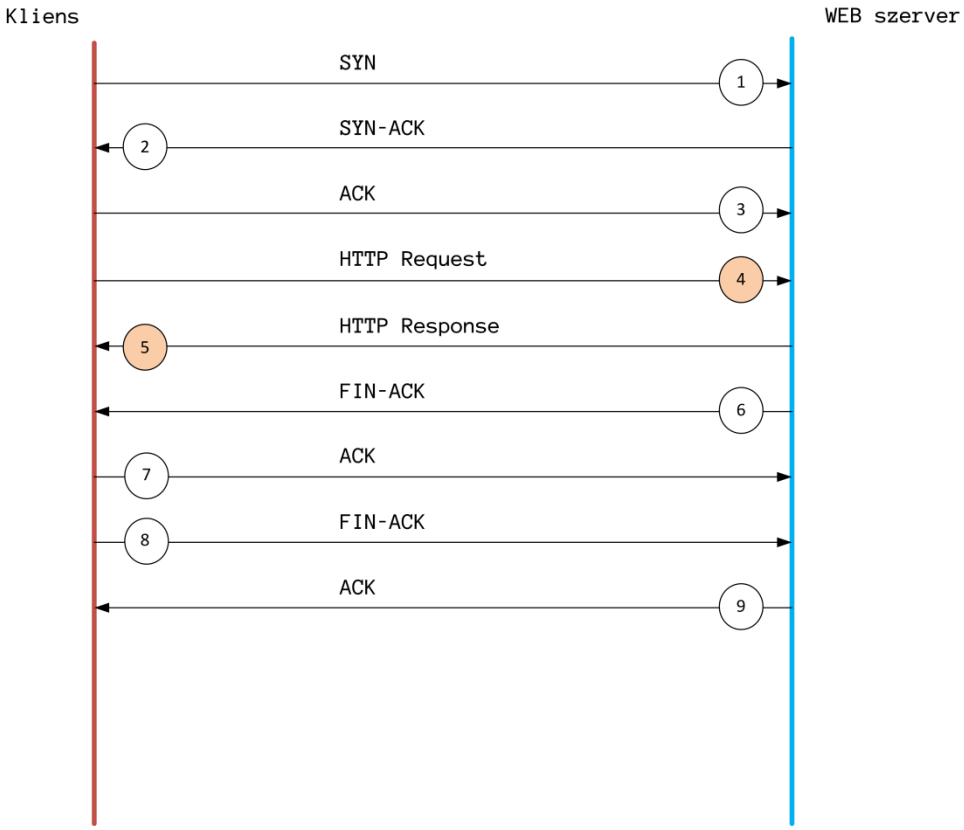
De hiba lenne a HTTP kapcsán csak a protokollról beszélni - hiszen legalább annyira érdekes a környezet is, amelyben használják. Oké, hogy tudjuk, hogyan épül fel egy

request csomag - de azt sem árt tudni, mikor küldi azt a kliens és hogy mondjuk hányszor.

Így most először beszélek nagy általánosságban a HTTP-t használó felek kommunikációjáról, sunyi kis trükkjeiről. Aztán ha már képben leszünk, akkor jöhetnek a csomagboncolások.

Fontos, hogy a kommunikációról lesz szó. Eszem ágában sincs webszerver üzemeltető fejezetet írni. Az Internet Information Server bemutatása önmagában is megtöltene egy könyvet.

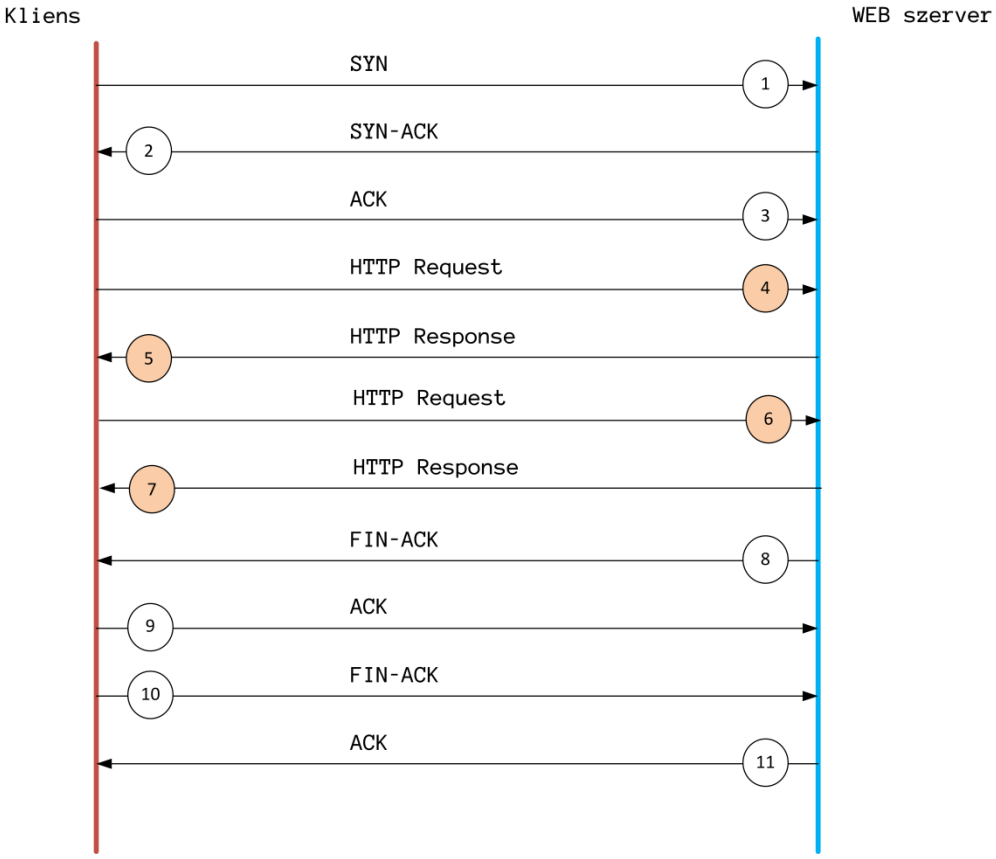
Tehát ott jártunk, hogy tipikus kliens-szerver kommunikáció.



2.4. ÁBRA HTTP KLIENS-SZERVER KOMMUNIKÁCIÓ

Látható, semmi faxni: csatorna kiépül, jön a kérés, megy a válasz, csatorna mindkét oldal felől lebomlik. Nagyjából így is működött a HTTP az 1.0 verzióig. Sok baj nem volt vele - azon kívül, hogy borzasztó lassú volt.

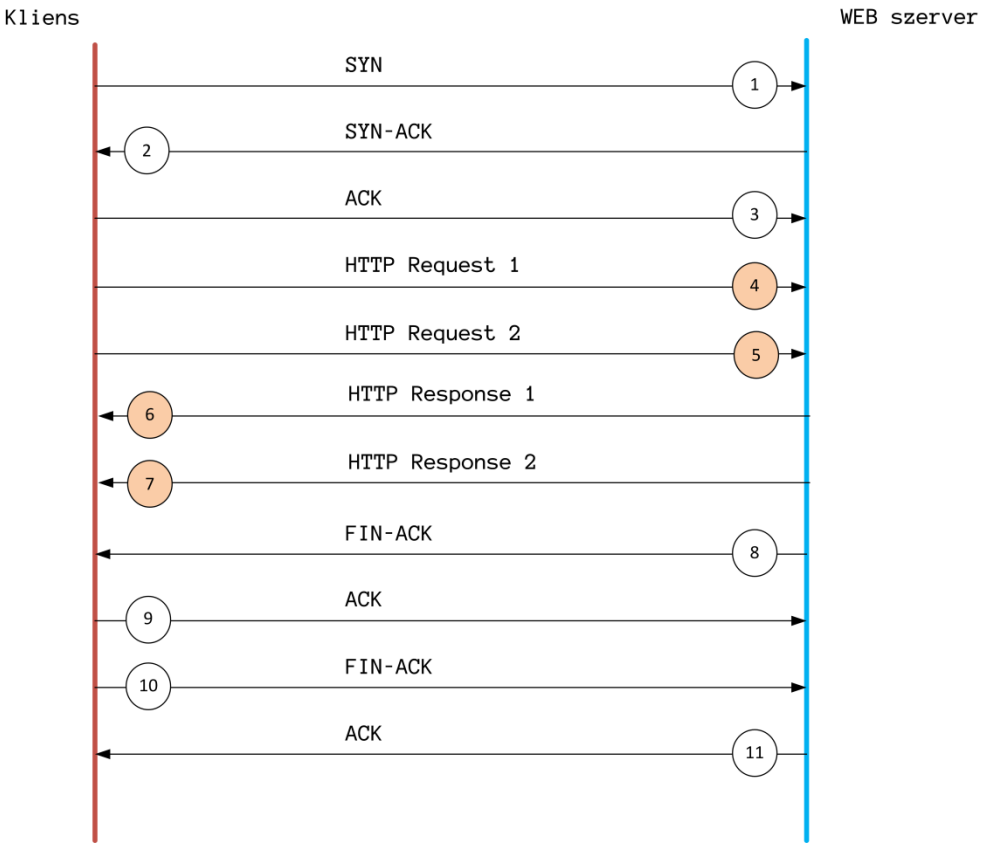
Az 1.1 verzióban gyorsítottak rajta egy kicsit.



2.5. ÁBRA FOLYAMATOS KAPCSOLAT

Egy weboldal ugyanis ritkán jön le egy kéréssel. (Az a gopher.:-) Külön kell kérni a szöveget, külön kell kérni a képeket, külön a kliens oldali szkripteket. Nyilván ha nem nyitunk mindegyik kérésnek egy külön sessiont, akkor valamivel gyorsabban jön le az oldal. Ezt a trükköt hívják úgy, hogy persistence, azaz folyamatos kapcsolat. Értelemszerűen komoly összjáték kell hozzá - elsősorban a szervernek kell mérsékelnie magát, ne kezdje el addig lezárni a kapcsolatot, amíg a kliens meg nem kapta a teljes oldalt.

Gyorsnak gyors... de nézd meg jobban a rajzot. Optimális is?

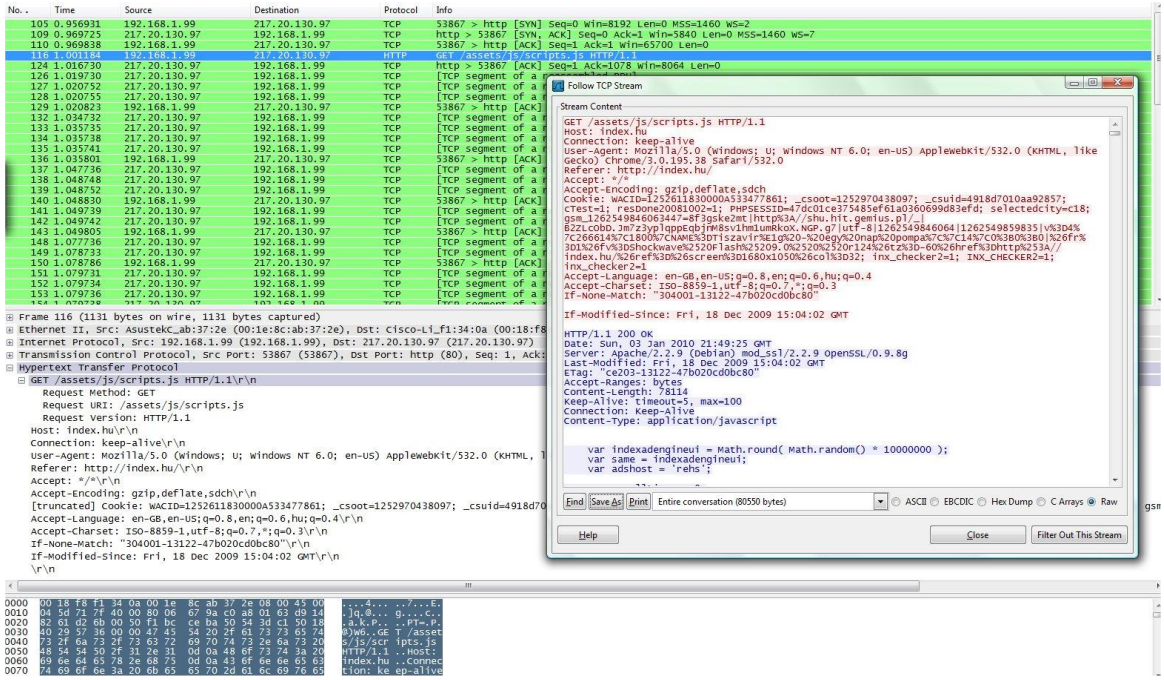


2.6. ÁBRA PIPELINE

Már hogy lenne? Mikor jobb a soros feldolgozás a párhuzamosnál? A kliens géppuska-sebességgel elkezdi lődözni a kéréseket, nem várva meg, amíg megérkezgetnek az előző kérésekre a válaszok, a szerver pedig köpni-nyelni nem tudva folyamatosan nyomja lefelé az ojjektumokat.
Máris gyorsabb lett az index.hu. (De még nem elég gyors. Aki megnézi az ábrát (*2.3. ábra Lekértem az index nyitólapját*), az láthat rajta még egy trükköt. De ezt csak később mesélem el.)

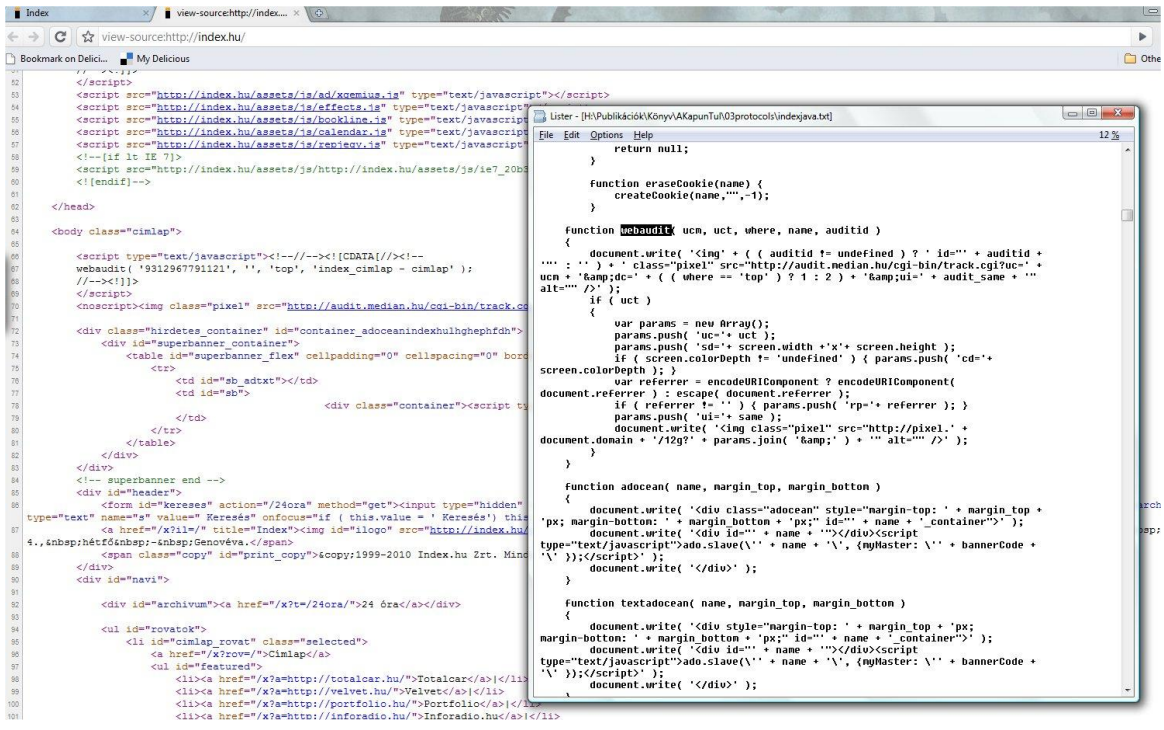
Apró kis kitérő.

Írtam, hogy a kliens külön objektumként kéri le a kliens oldali szkripteket. De ha ez így van, akkor el is lehet ezeket az objektumokat kapni.



2.7. ÁBRA JAVASCRIPT AZ INDEXEN

El bizony. És ha elkaptuk, akkor az ügyes kis Wireshark össze is rakja nekünk a szkriptet. Látjuk? A piros a kérés, a kék a válasz. És ott van benne a tipus:
`content-Type: application/javascript`



2.8. ÁBRA A WEBLAP FORRÁSA KONTRA CAPTURE

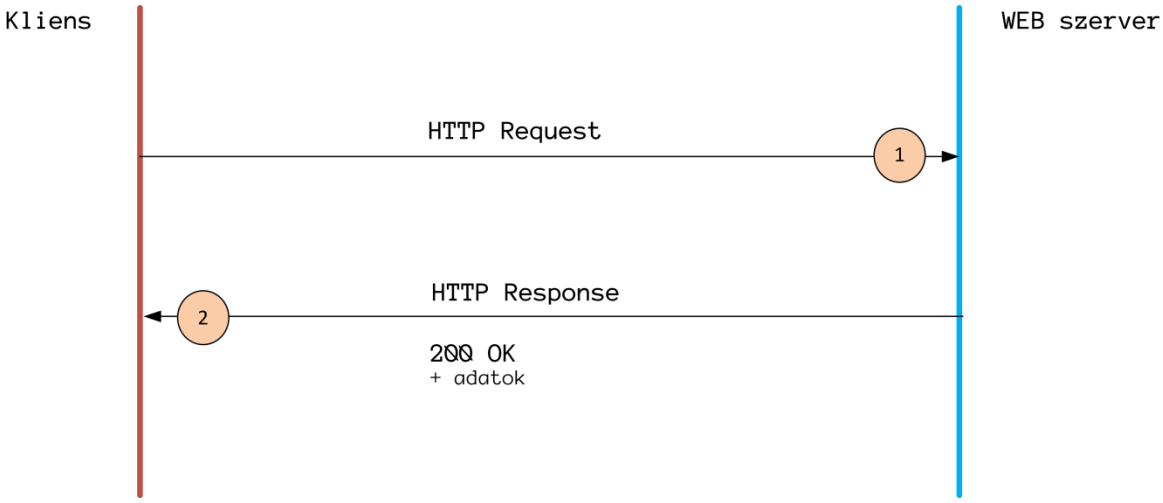
Aki szeret bogarászgatni, az innentől el lesz egy darabig az új játékkal.

Mi viszont beszéljünk komolyabb dolgokról.

Például a sütikről.

A HTTP kapcsolat ugyanis önmagában nem tud állapotot tárolni. A kliens elküldi a kérést, a szerver fogadja, majd visszaadja a választ. Vegyük észre, hogy a szervernek fogalma sincs, hogy ez a kliens most volt itt először, vagy egy perccel ezelőtt is ő kopogtatott. Amennyire a szerver emlékezik... ahhoz képest az aranyhal egy memóriazsonglőr.
Baj ez? Nem annyira. Feltéve, hogy szeretünk állandóan bejelentkezni a kedvenc fórumunkba minden topikváltáskor. Ha nem zavar, hogy a delicous mindig jelszót kér, ha el akarunk menteni egy címet.
Oké, baj. Akkor okosítsuk fel a webszerverünket. Jegyezzen meg minden kérést visszamenőleg mondjuk két hétig, és minden kérésnél fussa át ezeket. Előre a használhatatlan webszerverekért!

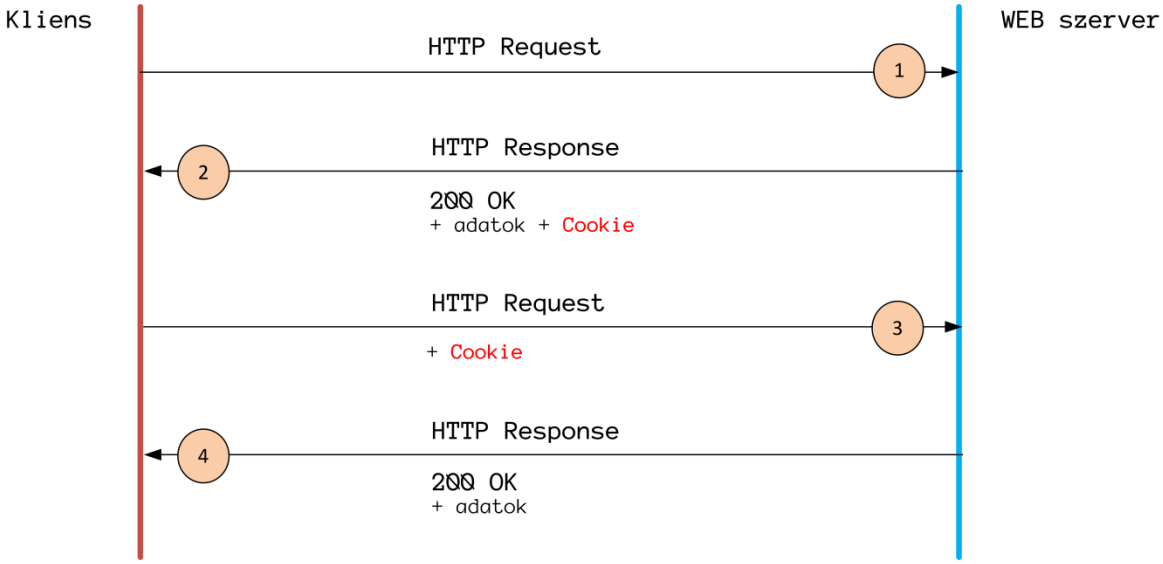
Erre a problémára jelentenek megoldást a sütik. (Cookies.) Hogyan működnek?



2.9. ÁBRA HTTP ÜTÉSVÁLTÁS

Így néz ki egy hagyományos HTTP románc. A kliens lekér egy weboldalt, a szerver a 200-as kóddal jelzi, hogy a kért weblapot megtalálta - majd a válaszüzenetbe be is csomagolja azt.

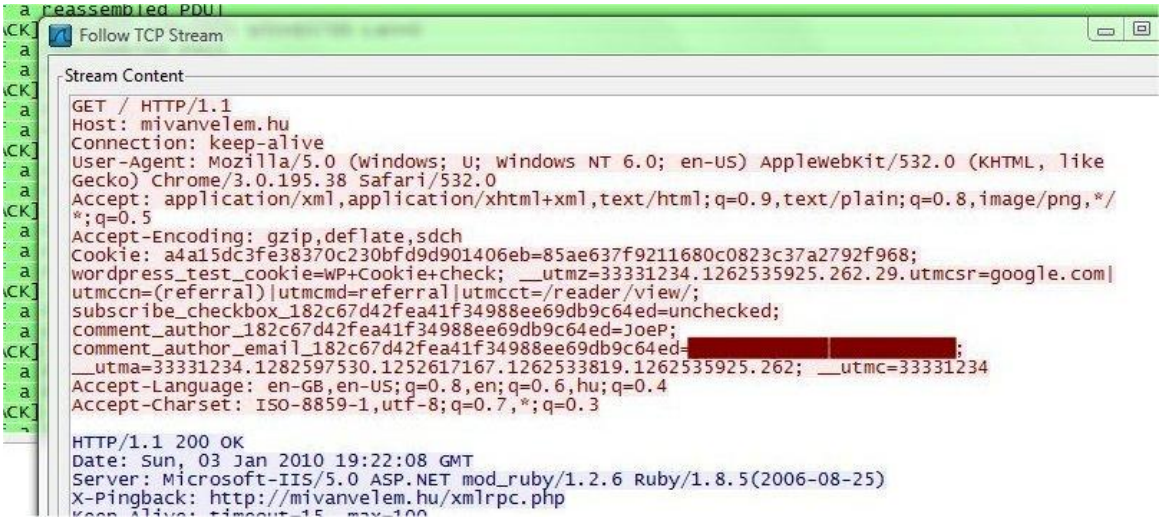
Itt még nincs cookie.



2.10. ÁBRA MEGJELENIK A COOKIE

Itt már van. Amikor a szerver összerakja a választ, belerak egy sütit. A kliens megköszöni, lementi, majd amikor legközelebb megint a szerverhez fordul, akkor már viszi magával.

Ebbe a sütibe van belesütve minden olyan információ, mely fontos lehet az éppen pisztergált webalkalmazás számára: felhasználónév, emailcím, képernyőpozíció... bármi.



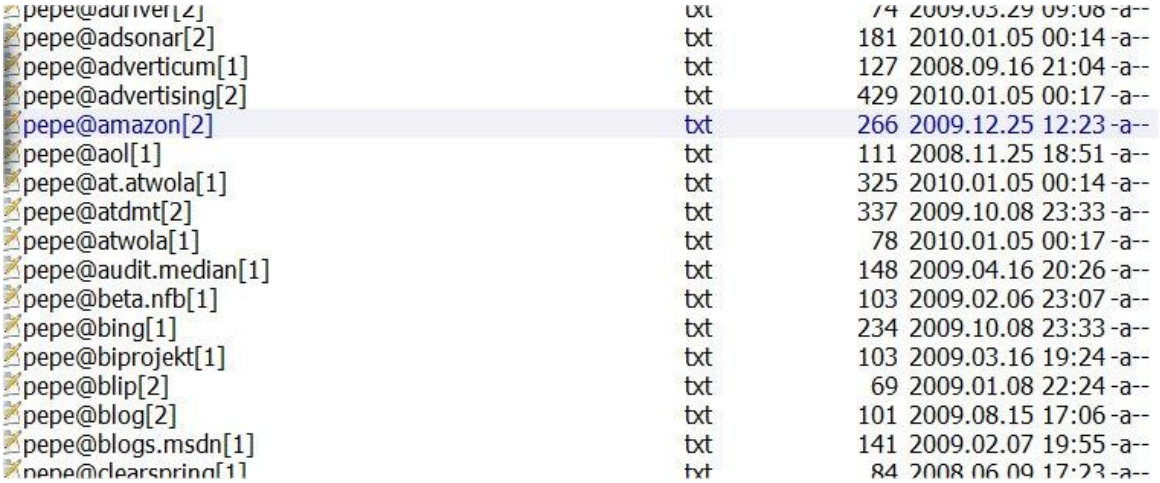
2.11. ÁBRA SÜTI A MIVANVELEM OLDALHOZ

Nézzünk még egy példát. Beléptem a mivanvelem.hu oldalra, ahol rendszeresen szoktam kommentelni. A kliensprogramom már eleve tudta, hogy ehhez az oldalhoz van egy sütim, tehát helyből belerakta a kérésbe. Látod, nem: valami bináris zsizsa, a fene tudja, mit jelent, aztán ott van a kommentelő neve (JoeP), majd némi zsizsa után

jön a kommenteléskor legutóbb megadott emailcímem. (Ezt takartam le két téglalappal.) A süti miatt van az, hogy amikor belépek erre az oldalra, akkor már ki vannak töltve a kommentelő form elemei.

Ha érdekel még egy példa, lapozz vissza a következő ábrához: *2.3. ábra Lekértem az index nyitólapját*. Miket tudsz kihüvelyezni ebből a sütiből? '*Tiszavirág*'. Meg '*egy nap pompa*'. Meg egy lengyel weblap. Úgy látszik, az index szereti az abszurd ízesítésű sütiket. De biztos van rá okuk.

Vajon meg lehet nézni ezeket a sütiket a gépeden is? Hiszen tuti, hogy ott tárolja valahol a böngészőprogramod.



2.12. ÁBRA SÜTITÁR

Ott vannak, bizony. A lokális profilodban. Méghozzá szöveges formában. Elég kellemetlen meglepetés, hogy a sütik nagy része reklámoldal. Vajon miket jegyezhetnek meg ezek rólad? Az amazon.com mondjuk még érthető is, ha szoktál ott vásárolni, akkor tudod, hogy minden belépéskor az ízlésedhez passzoló kínálatot dobnak fel a friss árukból.

Nyugodt vagy?

Mi van akkor, ha egy weblap felolvassa az összes sütidet és komplett profilt készít rólad?
Csoda.
Minden süti webszerverhez van rendelve és a kliens kéréskor csak azt a sütit viszi magával, amelyik a becélzott webszerverhez tartozik

Megnyugodtál?

Kár volt. Ugyanis ha egy reklámszerveren lévő banner van befűzve száz oldalra, akkor ugyanaz a szerver mindegyik - nem hozzátartozó, de sessionon belüli - oldal látogatásakor hozzáfér a sütijéhez és belerakhat abba éppen az érintett oldalakra jellemző infókat. Azaz pont a reklámszerverek azok, amelyek bannereken keresztül tényleg képesek profilt alkotni rólad.

Természetesen a sütiket le tudod tiltani a böngésződben. de ekkor visszatértünk oda, hogy a webszerver abszolút semmit sem fog tudni rólad. Ennél már az is jobb ötlet, ha valamilyen adblock plugint használsz.

Érzem, tűkön ülsz, hogy boncoljunk már végre kéréseket, válaszokat. Nyugi. Lesz még itt annyi táblázat, hogy éjszaka is kockásat fogsz álmodni.

De most ismételjünk egy kicsit. Mi is az, hogy proxy? Az inas, aki beviszi a névjegyet. Hogyan is néz ez ki konkrétan a HTTP esetében?
Hát úgy, hogy a kliens a kérésével nem a webszerverhez fordul közvetlenül, hanem a proxyhoz. Az jó tanárbácsisan megvizsgálja, mit is szeretne kérni a kliens a webszervertől. Értelmezi a kérést. Ehhez persze minimum azt a HTTP verziót kell ismernie, mint a megcélzott webszerver. Aztán ha már pontosan tud mindent, akkor kliensként elmegy az illető webszerverhez, tolmácsolja neki a kérést, majd a kapott választ visszaadja az eredeti kliensnek.

Ravasz kérdés jön: mit csinál akkor a proxy cache?
Hoppá. Erről eddig nem volt szó.

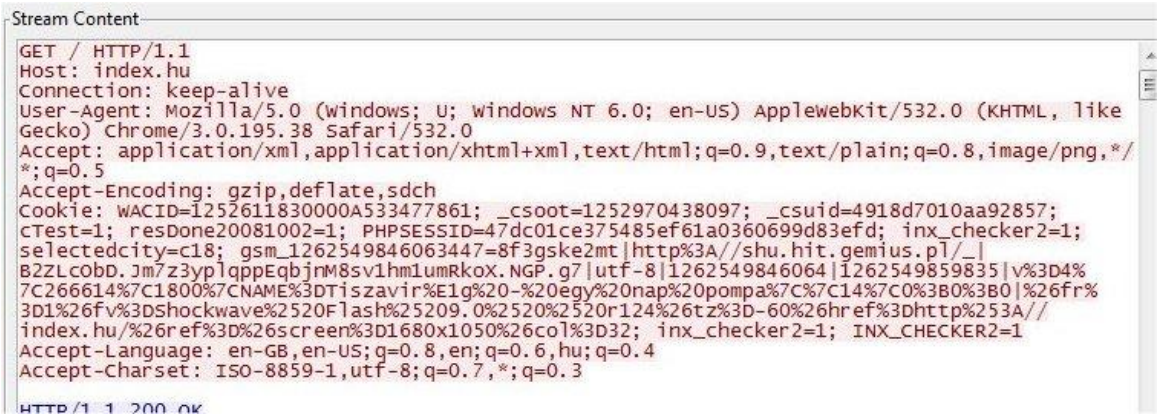
Pedig nem nehéz. Sőt, logikus.

A proxy nem csak egyszerűen visszaadja a kért tartalmat a kliensnek, hanem be is rakja egy átmeneti tárolóba. Ez a cache. Így ha rövid időn belül valamelyik másik kliens is lekéri ugyanezt az oldalt, akkor nem megy el az üveghegyen túlra, hanem a saját tárolójából adja vissza.

Magunk között vagyunk, beszélhetünk őszintén. Én borzasztóan idegenkedek mindenféle cache technikától. De ez érthető, mert én már öreg vagyok és amennyi varjút láttam már karón, annyi a macskaparadicsomban sincs. Mindenhol, ahol valamilyen cache technika játszik, ott az ördög a részletekben bújik meg. Eleinte a cache használat... mondjuk úgy, hogy nem volt teljesen kifinomult. Egyszer megszakadt egy letöltésed és utána órákig csak a hibás fájl jött le a proxyról, akármelyik gépről is próbálkoztál. Hozzászóltál egy fórumon, pörgött a téma, bizsergett az ujjad, kíváncsi voltál a többiek reakciójára - de a proxy cache miatt hosszú tíz percekig nem láttál változást.

Aztán telt-múlt az idő, őszültek a mérnökök - és ma már egész jól használható technológia lett belőle. Meg jó bonyolult is, mondjuk.

2.1.1 HTTP REQUEST

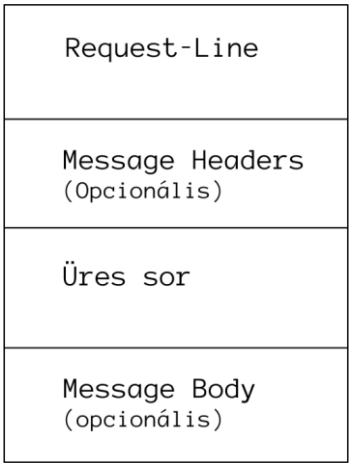


2.13. ÁBRA HTTP REQUEST

Kinagyítottam a korábbi ábra (*2.3. ábra Lekértem az index nyitólapját*) egy részletét. Így néz ki egy vadonban kóborló HTTP request.

Ha nem akarunk sniffer programot telepíteni, de kíváncsiak vagyunk a request/response blokkokra, használjuk bátran a következő linket:
http://web-sniffer.net/

Hivatalosan a HTTP Request négy részre osztható.



2.14. ÁBRA A HTTP REQUEST ELEMEI

REQUEST-LINE: Erről a sorról van szó: GET / HTTP1.1. Általánosítva:
`'<metódus> <URI> <verzió>'`
alakú. Itt mondjuk meg a szervernek, hogy mi a szöszt is akarunk tőle.

MESSAGE HEADERS: Itt sorolunk fel minden olyan információt, melyről a szervernek tudnia kell ahhoz, hogy ki tudjon minket szolgálni.

MESSAGE BODY: Opcionális mező. Ha valami infót akarunk feltölteni a szerverre a kérés során (POST metódus), akkor az itt utazik.

Az egyes mezőket az különbözteti meg egymástól, hogy új sorban kezdődnek, azaz a Carriege Return-Line feed (CR-LF.) karakterekkel záródnak. Még az üres sor is.

2.1.1.1 REQUEST-LINE

```
GET          /            HTTP1.1.
<metódus>    <URI>        <verzió>'
```

Haladjunk hátulról, az az egyszerűbb. A verziószám azt jelenti, melyik a legmagasabb szintű HTTP verzió, melyet a kliens ismer. Az URI azt a weblapot adja meg, amelyikre a kérés vonatkozik. Jelen példában ez a /, azaz a root lap. Végül a metódus maga az ige - mit is akarunk elérni a kéréssel?

2.1. TÁBLÁZAT

Parancs	Leírás
GET	Lekérjük a weboldalt. Ennyire egyszerű. A szerver pedig visszaküld valamit, általában egy OK-t és magát az oldalt. Nyilván ez a leggyakoribb kérés.
HEAD	Lekérjük a weboldalt. Ismerős? De a szerver válasza már más: csak a fejlécet küldi el, magát az oldalt nem.
POST	Feldolgozás céljából adatokat küldünk a szerver számára. Klasszikus példa egy kitöltött form. Az adatok a Message Body részben utaznak, a túloldalon egy webes alkalmazás kapja el őket.
PUT	Feltöltünk egy adatcsomagot, leginkább egy fájlt. Szintén a Message Body részben utazik. A szerver nem dolgozza fel, egyszerűen csak egy elérhető adat lesz belőle.
DELETE	Megkérjük a szervert, hogy a megadott valamit törölje, léccilécci.
TRACE	Megkérjük a szervert, hogy küldje vissza a kérésünket. Ebből fogjuk tudni, hogy mi lett a kérésünkből, mire a szerverhez ért.
OPTIONS	Visszaadja azokat a HTTP metódusokat, melyeket a szerver egyáltalán megért.
CONNECT	Kéri a szervert - ami ebben az esetben általában proxy - hogy kezdjen el bőszen csatornát építeni.

2.1.1.2 Message Header (Request)

2.2. táblázat

Típus	Név
Accept	Az elfogadott tartalomtipusok
Accept-Charset	Az elfogadott karakterkészletek
Accept-Encoding	Az elfogadott kódolások
Accept-Language	Az elfogadott nyelvek
Authorization	A bejelentkezési adatok elküldése
Cache-Control	Előírja, hogy az üzenetváltásban részt vevő felek használhatnak-e az üzenetváltással kapcsolatban átmeneti tárolót - és ha igen, akkor hogyan.
Connection	Megnevezi azt a headert, melyet törölni kell a kérésből, ha az proxyn megy keresztül.
Cookie	A kliensünk már kapott egy sütit (Set-Cookie), és most visszaküldi a szervernek.
Content-Length	A Request Body mérete bájtban.
Content-Type	A Request Body adattipusa (MIME)
Date	Mikor küldték a kérést.
Expect	Ha a szerver visszaküldi ezt, akkor csináld azt. Jelenleg csak a '100 Continue' kódra harap.
From	A kérés küldőjének emailcíme. Nem túl népszerű mező.
Host	A megcélzott virtuális szerver neve. A HTTP/1.1 óta kötelező.

Mi is ez a virtuális szerver? Nos, arról van szó, hogy egy konkrét webszerver nem csak egy webalkalmazást tehet elérhetővé, hanem többet is. Ilyenkor a request header-be írt Host mezőben jelezzük, hogy melyik alkalmazáshoz fordulunk.
Konkrét példa: a mivanvelem.hu és az emaildetektiv.hu blogok ugyanazon a webszerveren laknak. Nézzük meg az ábrát (*2.11. ábra Süti a mivanvelem oldalhoz*), tisztán látható, hogy melyik oldalt támadtam be.

Típus	Név
If-Match	Ha letöltöttem valamit, majd módosítás után vissza akarom tölteni, akkor leellenőrzi, hogy közben másvalaki nem módosította-e?
If-Modified-Since	Cache vezérlés
If-None-Match	Cache vezérlés
If-Range	Cache vezérlés
If-Unmodified-Since	Cache vezérlés
Max-Forwards	Az üzenetet max. hányszor lehet forwardolni.

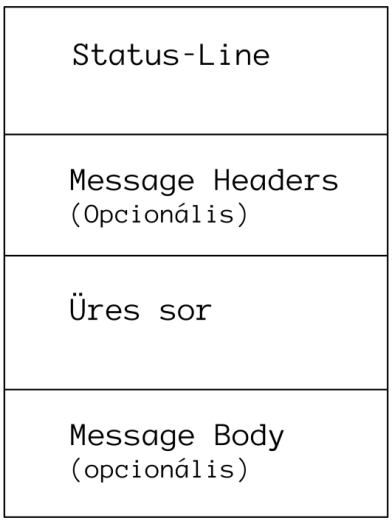
A webszerverek ugyanis képesek ilyen huncutságokra. A keresett weboldal már elköltözött a szerverről, de otthagyott egy ugróoldalt, hogy az érdeklődők az új helyen is megtalálják.

Tíous	Név
Pragma	Ősmaradvány a korai HTTP verziókból.
Range	Nem az egészet kérjük le, csak egy részét.
Referer	Melyik weboldalról érkeztünk a látogatott oldalra. A statisztikákat gyártó alkalmazások imádják. (Én is.)
TE	Transfer Encoding; megmondja a szervernek, hogy a kliens milyen kódolásokat képes fogadni.
Upgrade	Hogy a kliens és a szerver megtalálják a megfelelő magasságú közös HTTP/TLS szintet.
User-Agent	Kiféle-miféle jószág a kliens? (Láthatod, én éppen Chrome vagyok a példákban.)
Via	Közli a szerverrel, hogy milyen proxykon keresztül ért el hozzá a kérés.
Warn	Balhé van.

2.1.2 HTTP RESPONSE

2.15. ÁBRA HTTP RESPONSE

A HTTP Response felépítése nagyon hasonló a kérés felépítéséhez.



2.16. ÁBRA A HTTP VÁLASZ SZERKEZETE

Az utolsó 3 mező formailag teljesen azonos (mégha a header tipusok mások is). A markáns különbség a státusz sor. Itt ugyanis a szerver közli a klienssel, hogy mi is pontosan a helyzet a kérésével.

2.1.2.1 STÁTUSZ KÓDOK

Alapvetően öt nagy kategória létezik.

2.3. TÁBLÁZAT

Kód	Név	Magyarázat
1xx	Informational	A szerver fogta a kérést, de valamiért nem tud válaszolni.
2xx	Success	A szerver fogta a kérést és válaszolni is tud.
3xx	Redirection	A szerver fogta a kérést, de a kért oldal máshol van.
4xx	Client Error	Hibás a kérés, a szerver nem tudja teljesíteni.
5xx	Server Error	A szerver fogta a kérést, de a válasz során hiba keletkezett.

Minden különösebb magyarázat nélkül álljanak itt az egyes kódok:

2.4. TÁBLÁZAT

Érték	Leírás	RFC szám
100	Continue	[RFC2616]
101	Switching Protocols	[RFC2616]
102	Processing	[RFC2518]
200	OK	[RFC2616]
201	Created	[RFC2616]
202	Accepted	[RFC2616]
203	Non-Authoritative Information	[RFC2616]
204	No Content	[RFC2616]
205	Reset Content	[RFC2616]
206	Partial Content	[RFC2616]
207	Multi-Status	[RFC4918]
226	IM Used	[RFC3229]
300	Multiple Choices	[RFC2616]
301	Moved Permanently	[RFC2616]
302	Found	[RFC2616]
303	See Other	[RFC2616]
304	Not Modified	[RFC2616]
305	Use Proxy	[RFC2616]
306	Reserved	[RFC2616]
307	Temporary Redirect	[RFC2616]
400	Bad Request	[RFC2616]
401	Unauthorized	[RFC2616]
402	Payment Required	[RFC2616]
403	Forbidden	[RFC2616]
404	Not Found	[RFC2616]
405	Method Not Allowed	[RFC2616]
406	Not Acceptable	[RFC2616]
407	Proxy Authentication Required	[RFC2616]
408	Request Timeout	[RFC2616]
409	Conflict	[RFC2616]
410	Gone	[RFC2616]
411	Length Required	[RFC2616]
412	Precondition Failed	[RFC2616]
413	Request Entity Too Large	[RFC2616]
414	Request-URI Too Long	[RFC2616]
415	Unsupported Media Type	[RFC2616]
416	Requested Range Not Satisfiable	[RFC2616]
417	Expectation Failed	[RFC2616]
422	Unprocessable Entity	[RFC4918]
423	Locked	[RFC4918]
424	Failed Dependency	[RFC4918]
42	Upgrade Required	[RFC2817]

500	Internal Server Error	[RFC2616]
501	Not Implemented	[RFC2616]
502	Bad Gateway	[RFC2616]
503	Service Unavailable	[RFC2616]
504	Gateway Timeout	[RFC2616]
505	HTTP Version Not Supported	[RFC2616]
506	Variant Also Negotiates (Experimental)	[RFC2295]
507	Insufficient Storage	[RFC4918]
510	Not Extended	[RFC2774]

Forrás:
http://www.iana.org/assignments/http-status-codes

Részletesebben ismertetni egy olyat fogok, mely nem szerepel az előző felsorolásban.
Nem véletlenül. (Ettől függetlenül teljesen hivatalosan létezik, lásd az alábbi RFC-t.)

Maradjunk annyiban, hogy jó látni, hogy ebbe a száraz tudományba is belefér néha egy kis ökörködés. Még ha geek módra is.

RFC 2324

A fenti RFC az IETF egyik április elsejei tréfája. Egész konkrétan a HTCPCP 1.0 protokollt definiálja. A protokoll teljes neve: Hypertext Coffee Pot Control Protocol, azaz hipertextes kávékiöntő vezérlő protokoll. Az a bizonyos hibakód ehhez a protokollhoz kapcsolódik.

418-AS KÓD: I AM A TEAPOT.
4xx-es kód, azaz a szerver nem tudja értelmezni a kliens HTCPCP kérését. Nyilván nem is, hiszen a válaszban a szerver jelzi, hogy ő egy teakiöntő készülék, így nem tud mit kezdeni a kávékiöntő készülékek vezérlési parancsaival.

2.1.2.2 MESSAGE HEADER (RESPONSE)

Egy részükkel már találkozhattunk a kéréseknél. Ha nincs változás, akkor ezeket itt már nem fogom részletezni.

2.5. TÁBLÁZAT

Header	Magyarázat
ACCEPT-RANGES	Ha bedől egy letöltés, akkor mi az az informatikai alapegység, amelyikben a kliens lekérheti a hiányzó részleteket.
AGE	Mennyi ideig lehet az objektum a cache-ben. (sec)
ALLOW	Metódusok, melyek a kért objektumhoz használhatók.
CACHE-CONTROL	
CONTENT-ENCODING	A visszaadott adatok kódolása
CONTENT-LANGUAGE	A visszaadott adatok nyelve
CONTENT-LENGTH	
CONTENT-LOCATION	Hol található meg még az oldal
CONTENT-DISPOSITION	Ha ismert a MIME tipus, akkor megadja a lehetőséget egy "File Download" ablak előugrására.
CONTENT-MD5	A válaszból (message body) képzett MD5 hash, Base64 kódolásban.
CONTENT-RANGE	Ha a szerver részletekben válaszol, akkor itt mondja meg, melyik részletről van szó.
CONTENT-TYPE	
DATE	
ETAG	A lekért objektum verzió jellegű azonosítója. A legtöbbször MD5 hash.
EXPIRES	Mikortól nem lesz már érvényes a válasz.
LAST-MODIFIED	Mikor módosították utoljára a kért objektumot.
LOCATION	Hová van átirányítva a tartalom. (3xx kódok)
PRAGMA	
PROXY-AUTHENTICATE	A proxy autentikációt kér.
RETRY-AFTER	Ha a kért tartalom nem érhető el, mennyi idő próbálkozzon újra a kliens.
SERVER	A szerver neve, pontosabban tipusa..
SET-COOKIE	Megy a süti.
TRAILER	Van olyan, amikor a message body után még jönnek headerek. (Chunked transfer-coding) Ezeknek az utólagos headereknek a tipusai vannak felsorolva a trailer mezőben.

Na, ennek mi lehet az értelme? Hát, az, hogy a szerver, amikor készíti a választ, akkor már tudja, milyen headerek lesznek benne, de ezek még nem álltak össze. Ekkor használja ki a chunked kódolást. A kliens folyamatosan kapja a választ, a szerver menetközben folyamatosan gyártja a csomagokat - így egyiknek sem kell üresjáratban várakoznia.

HEADER	Magyarázat
TRANSFER-ENCODING	Milyen kódolásban megy a válasz. Lehetőségek: chunked, compress, deflate, gzip, identity.
VARY	Cache vezérlés
VIA	
WARNING	
WWW-AUTHENTICATE	Milyen autentikációs módszert kell használni a kért tartalom eléréséhez.

Igértem, hogy megmutatom, mit trükközött még az index.hu, hogy gyorsabb legyen az oldal letöltése.

Nézzük csak meg az ábrát (*2.3. ábra Lekértem az index nyitólapját*).

REQUEST HEADER:

- Accept-encoding: gzip, deflate, sdch.

A szerver pedig boldogan vette tudomásul, hogy a kliens elfogadja tömörített formában is a weboldalt.

RESPONSE HEADER:

- Content-encoding: gzip.

Majd az ablak alján (*2.15. ábra HTTP Response*) láthatod is, hogy a message body már egy ronda, értelmezhetetlen bináris fájl. Tömörített.

2.2 FTP - A TEHERHORDÓ ÖSZVÉR

RFC 959

FTP, azaz File Transfer Protocol. Hozza-viszi a biteket a dróton.

A HTTP-hez meglehetősen sokban hasonlít. Egyrészt ugyanúgy kliens-szerver formában létezik, másrészt nagyjából egykorúak - azaz megalkotásuk pillanatában IT biztonsági szempontokról még csak nem is hallottak. (De erről majd később.)
A felek közötti kommunikáció is hasonlít annyiból, hogy itt is kérések és válaszok vannak - de az irányok már nem lesznek annyira egyértelműek. Emiatt nem is érdemes külön kezelni, egyszerűen parancsoknak nevezzük mindkettőt.
A parancsoknak már nincs annyira rögzített szintaktikája, mint a HTTP esetében.
A session fogalom ugyanúgy létezik - csak itt több is van belőlük egy munkameneten belül.

Mielőtt bármibe is belevágnánk, ideborítom a parancsok és a kódok listáját. Méghozzá minden különösebb magyarázat nélkül, úgy, ahogy a neten is megtalálható. Ezekből a parancsokból ugyanis annyi van, hogy a részletes ismertetésük bőven meghaladja e könyv kereteit.
Nem baj, ha elsőre nem fogod érteni az összeset. Ahogy haladunk majd előre, úgy lesz értelme a fontosabb parancsoknak. A többi meg úgyis csak azért van itt, hogy teljes legyen a kép.

2.2.1 FTP PARANCSOK

2.6. TÁBLÁZAT

Parancs	RFC	Leírás
ABOR		Abort an active file transfer.
ACCT		Account information.
ADAT	RFC 2228	Authentication/Security Data
ALLO		Allocate sufficient disk space to receive a file.
APPE		Append.
AUTH	RFC 2228	Authentication/Security Mechanism
CCC	RFC 2228	Clear Command Channel
CDUP		Change to Parent Directory.
CONF	RFC 2228	Confidentiality Protection Command
CWD		Change working directory.
DELE		Delete file.
ENC	RFC 2228	Privacy Protected Channel
EPRT	RFC 2428	Specifies an extended address and port to which the server should connect.
EPSV	RFC 2428	Enter extended passive mode.
FEAT	RFC 2389	Get the feature list implemented by the server.
HELP		Returns usage documentation on a command if specified, else a general help document is returned.
LANG	RFC 2640	Language Negotiation
LIST		Returns information of a file or directory if specified, else information of the current working directory is returned.
LPRT	RFC 1639	Specifies a long address and port to which the server should connect.
LPSV	RFC 1639	Enter long passive mode.
MDTM	RFC 3659	Return the last-modified time of a specified file.
MIC	RFC 2228	Integrity Protected Command
MKD		Make directory.
MLSD	RFC 3659	Lists the contents of a directory if a directory is named.
MLST	RFC 3659	Provides data about exactly the object named on its command line, and no others.
MODE		Sets the transfer mode (Stream, Block, or Compressed).
NLST		Returns a list of file names in a specified directory.
NOOP		No operation (dummy packet; used mostly on keepalives).
OPTS	RFC 2389	Select options for a feature.
PASS		Authentication password.
PASV		Enter passive mode.
PBSZ	RFC 2228	Protection Buffer Size
PORT		Specifies an address and port to which the server should connect.
PWD		Print working directory. Returns the current directory of the host.
QUIT		Disconnect.
REIN		Re initializes the connection.
REST		Restart transfer from the specified point.
RETR		Retrieve (download) a remote file.
RMD		Remove a directory.
RNFR		Rename from.
RNTO		Rename to.
SITE		Sends site specific commands to remote server.
SIZE	RFC 3659	Return the size of a file.
SMNT		Mount file structure.
STAT		Returns the current status.
STOR		Store (upload) a file.
STOU		Store file uniquely.
STRU		Set file transfer structure.
SYST		Return system type.
TYPE		Sets the transfer mode (ASCII/Binary).
USER		Authentication username.

Forrás:
http://en.wikipedia.org/wiki/List_of_FTP_commands

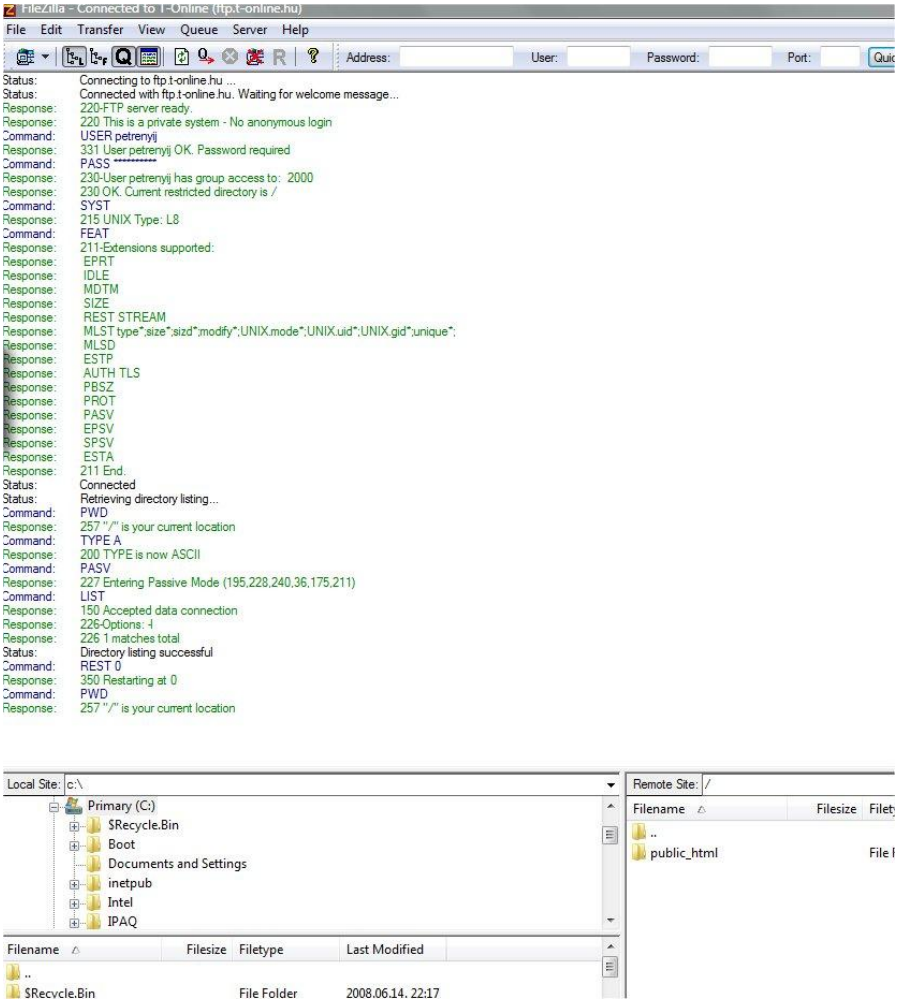
A szép az egészben az, hogy a hőskorban úgy ment az eftépézés, hogy az ember fogta és begépelte ezeket a parancsokat a command prompt-ba.
Grafikus kliens? Ugyanmár.

2.17. ÁBRA FTP PARANCSSORBÓL

A fenti munkamenetben az történt, hogy beléptem egy FTP szerverre, lekértem a könyvtár tartalomjegyzékét, lejjebb léptem egy könyvtárral, majd letöltöttem az index.html fájt. Feltölteni a PUT paranccsal tudtam volna.

(Az ábrán lévő parancsokat ne is keresd az előző táblázatban, ezek a parancsok a Windows beépített fapados telnet kliensének a parancsai. Valójában a GET parancs mögött egy FTP parancskötegnek kell lennie, mely egészen biztosan tartalmazza - többek között - a MODE és a RETR parancsokat.)

Csak az érdekesség kedvéért nézzük meg, hogyan csinálja ugyanezt egy profi.



2.18. ÁBRA FTP GRAFIKUS KLIENSBŐL

Még csak addig jutottunk, hogy beléptünk a szerverre és lekértük a tartalomjegyzéket - de nézzük meg, mennyi parancsot adtunk ki már eddig is.
Hiába, választékosan fogalmazni tudni kell.

2.2.2 FTP KÓDOK

De nem csak parancsokat láthatunk a fenti ábrákon. Vannak kódok is. Olyan kódok, melyeket a szerver ad vissza.

2.7. TÁBLÁZAT

Kód	Magyarázat
100	Series: The requested action is being initiated, expect another reply before proceeding with a new command.
110	Restart marker replay . In this case, the text is exact and not left to the particular implementation; it must read: MARK yyyy = mmmm where yyyy is User-process data stream marker, and mmmm server's equivalent marker (note the spaces between markers and "=").
120	Service ready in nnn minutes.
125	Data connection already open; transfer starting.
150	File status okay; about to open data connection.
200	Command okay.
202	Command not implemented, superfluous at this site.
211	System status, or system help reply.
212	Directory status.
213	File status.
214	Help message.On how to use the server or the meaning of a particular non-standard command. This reply is useful only to the human user.
215	NAME system type. Where NAME is an official system name from the list in the Assigned Numbers document.
220	Service ready for new user.
221	Service closing control connection.
225	Data connection open; no transfer in progress.
226	Closing data connection. Requested file action successful (for example, file transfer or file abort).
227	Entering Passive Mode (h1,h2,h3,h4,p1,p2).
228	Entering Long Passive Mode (long address, port).
229	Entering Extended Passive Mode (|||port|).
230	User logged in, proceed. Logged out if appropriate.
231	User logged out; service terminated.
232	Logout command noted, will complete when transfer done.
250	Requested file action okay, completed.
257	"PATHNAME" created.
331	User name okay, need password.
332	Need account for login.
350	Requested file action pending further information
421	Service not available, closing control connection. This may be a reply to any command if the service knows it must shut down.
425	Can't open data connection.
426	Connection closed; transfer aborted.
434	Requested host unavailable.
450	Requested file action not taken.
451	Requested action aborted. Local error in processing.
452	Requested action not taken. Insufficient storage space in system.File unavailable (e.g., file busy).
500	Syntax error, command unrecognized. This may include errors such as command line too long.
501	Syntax error in parameters or arguments.
502	Command not implemented.
503	Bad sequence of commands.
504	Command not implemented for that parameter.
530	Not logged in.
532	Need account for storing files.
550	Requested action not taken. File unavailable (e.g., file not found, no access).
551	Requested action aborted. Page type unknown.
552	Requested file action aborted. Exceeded storage allocation (for current directory or dataset).
553	Requested action not taken. File name not allowed.

Forrás:
http://en.wikipedia.org/wiki/List_of_FTP_server_return_codes

Látható, hogy a tagozódás teljesen ugyanaz, mint a HTTP esetében (*2.3. táblázat*).

2.2.3 AZ FTP PROTOKOLL LELKIVILÁGA

Kezdjük megint történelmi áttekintéssel.
Kinek mond még valamit az, hogy anonymous FTP? Pedig volt. Sőt, ebből volt több. Például a gyártók a drivereket legtöbbször FTP szervereken tárolták, ahhoz meg nem kellett autentikáció.

> Pontosabban, kellett. De jó volt a kamu is. Anonymous FTP belépésnél a felhasználó az 'anonymous' user volt, a jelszava pedig egy emailcím. Melyet persze a kutya sem ellenőrzött.

Nos, ezek voltak a boldog békeidők. Ma az ilyesmi ritka, mint a fehér holló: hasonló célokra már HTTP alapú letöltések szolgálnak, vagy a Trivial FTP (TFTP).

Sokkal izgalmasabb téma az FTP munkamenetek kavarodása. Mint már céloztam rá, egy viszonylag egyszerű feladat - például egy fájl letöltése - az FTP-ben nem intézhető el egy munkameneten belül, mint például a HTTP-nél. Az FTP ugyanis külön csatornákat használ egyfelől a vezérlőinformációk (Control vagy Command) forgalmazására, másfelől az adatforgalomra (Data). Mindkét csatornát a jól ismert módon (SYN, SYN/ACK, ACK) építik ki a felek.

Pusztán csak azért, hogy ne legyen egyszerű az életed, mindemellett a csatornák kiépítése három különböző forgatókönyv szerint is történhet:

- Aktív mód
- Passzív mód
- Kiterjesztett passzív mód

> RFC 2428

A legkevesebbet a harmadik változattal fogunk foglalkozni, ez gyakorlatilag a passzív mód kiterjesztése IPv6, illetve NAT környezetekre.

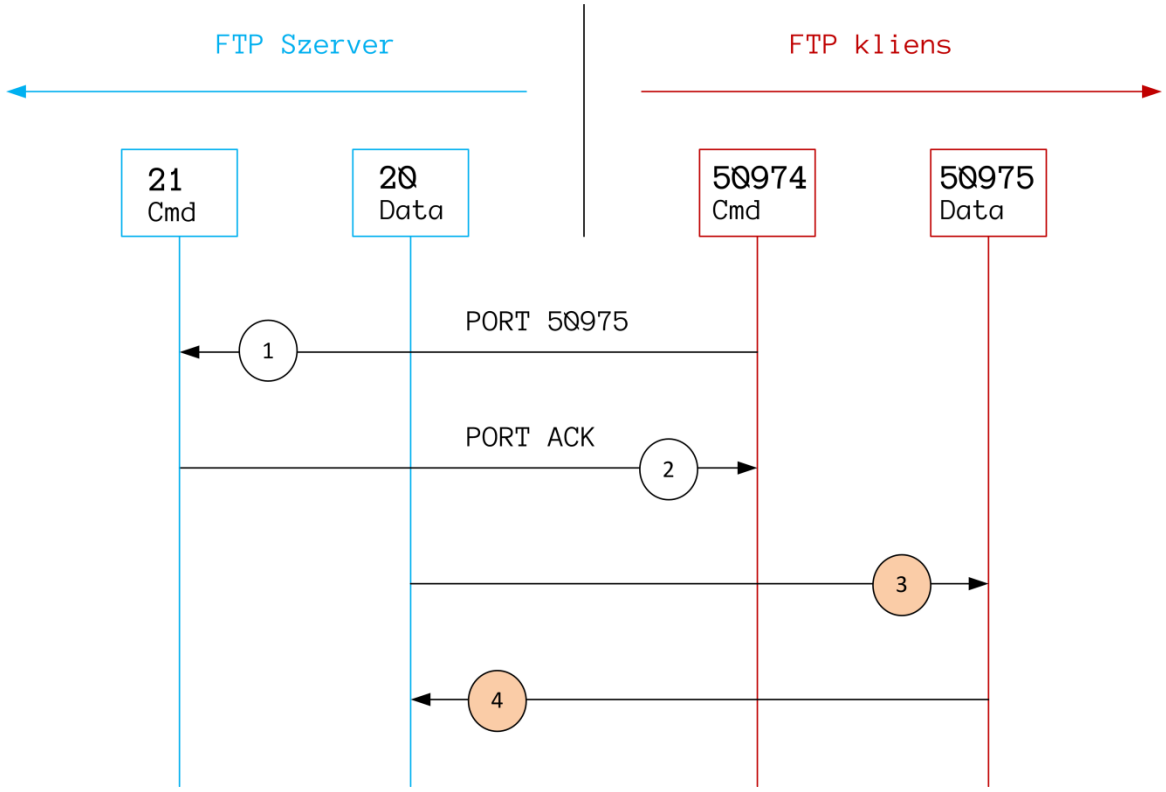
Az meg már kész provokáció, hogy a Data csatornán történő adatáramlás is kétféle lehet:

- ASCII mód : plain text áramlik át a csatornán.
- Bináris mód : bináris kód áramlik át a csatornán.

2.2.3.1 AKTÍV MÓD

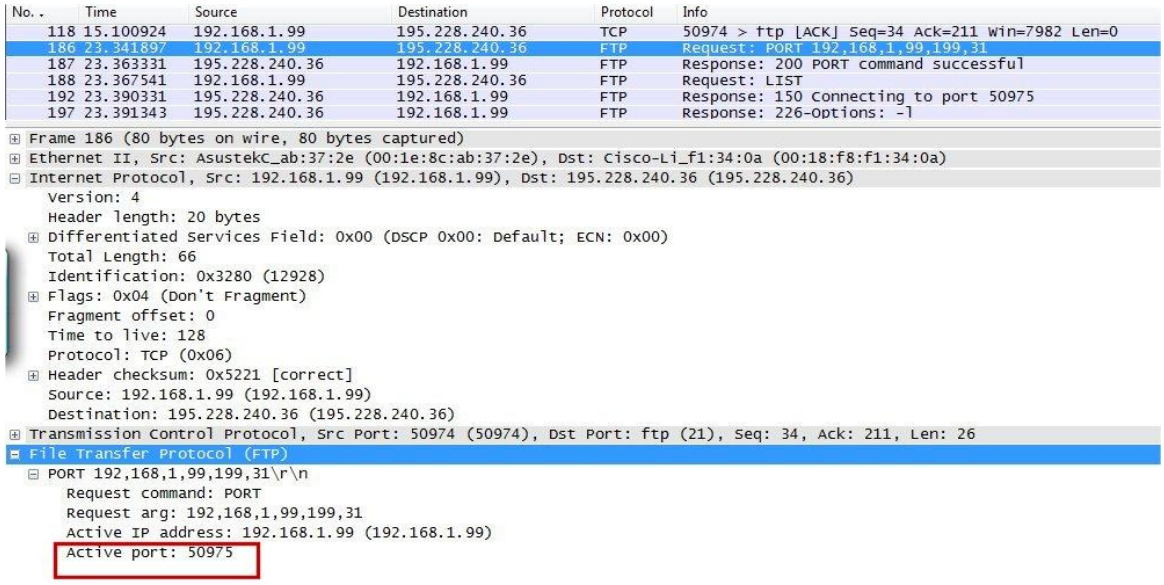
Csak hogy tudjuk, mire készülünk: ez a fejezet azzal fog foglalkozni, hogyan épülnek ki a Command és a Data csatornák abban az esetben, ha a kliens az ún. aktív FTP módot használja.

Így.

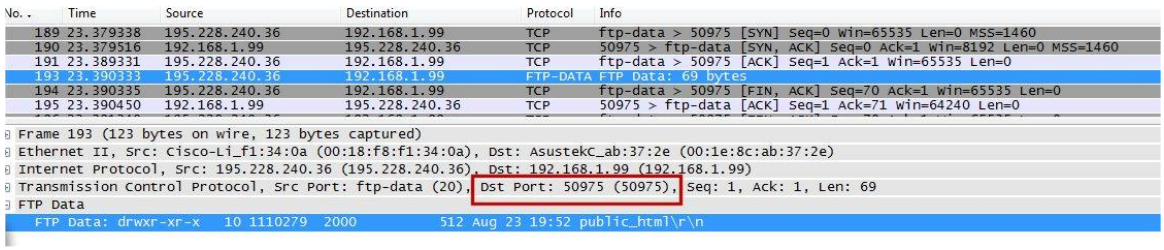


2.19. ÁBRA AKTÍV FTP KLIENS

Első lépésben a kliens - miután kiépítettek egy TCP sessiont - a Command csatornán keresztül küld az FTP szervernek egy PORT parancsot. Ebben benne van az IP címe és egy port szám. Ezen a porton várja a szerver próbálkozását a Data csatorna kiépítésére.



2.20. ÁBRA PORT REQUEST



2.21. ÁBRA DATA FORGALOM

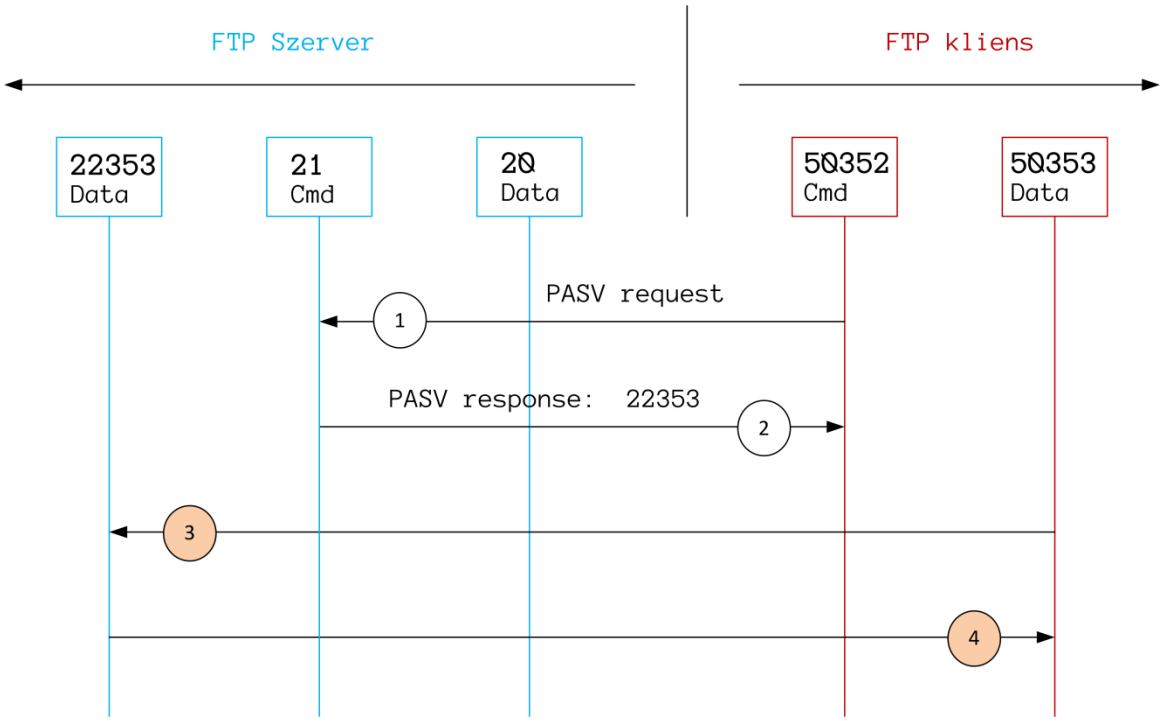
Az első ábrán a kliens (192.168.1.99) értesíti a szervert (186-os csomag), hogy az 50975-ös porton várja a behatolást. A 187-es csomagban a szerver jelzi, hogy vette az adást. A következő ábrán pedig láthatjuk, hogy a szerver nem viccel, a 20-as portjáról tolja a biteket a kliens 50975-ös portjára.
Vegyük észre, hogy a 189-191 csomagokban épül ki az új - Data - csatorna. (Az ábrákon munkamenetekre szűrt forgalmak láthatók - ezért nincsenek benne például a 189-191-es csomagok az első ábrában.)

Le a kalappal az EventHelix fiúk előtt. Az alábbi címről le lehet tölteni egy pdf doksit, mely minden túlzás nélkül gyönyörűen és plasztikusan mutatja be, mi is történik egy látszólag egyszerű fájlletöltés során. (Aktív FTP kliens)
http://www.eventhelix.com/RealtimeMantra/Networking/FTP.pdf
De érdemes átnézni a többi anyagukat is:
http://www.eventhelix.com/RealtimeMantra/Networking/

Mi a módszer hátránya? Nézzük csak meg: a Data csatorna kiépítése úgy történik, hogy a szerver kezdeményez kapcsolatot a kliens egyik portjára. Mit mondanak erre a a betörési kisérletre a mai tűzfalak? Azt, hogy coki.

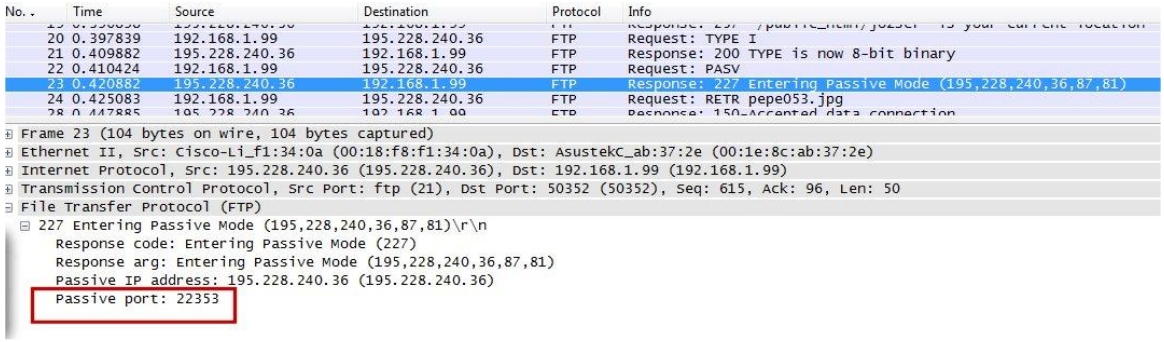
2.2.3.2 PASSZÍV MÓD

Valahogy ki kellene iktatni azt a hiperaktív FTP szervert. Várja csak meg, hogy a kliens kezdeményezze a Data csatorna felépítését is.

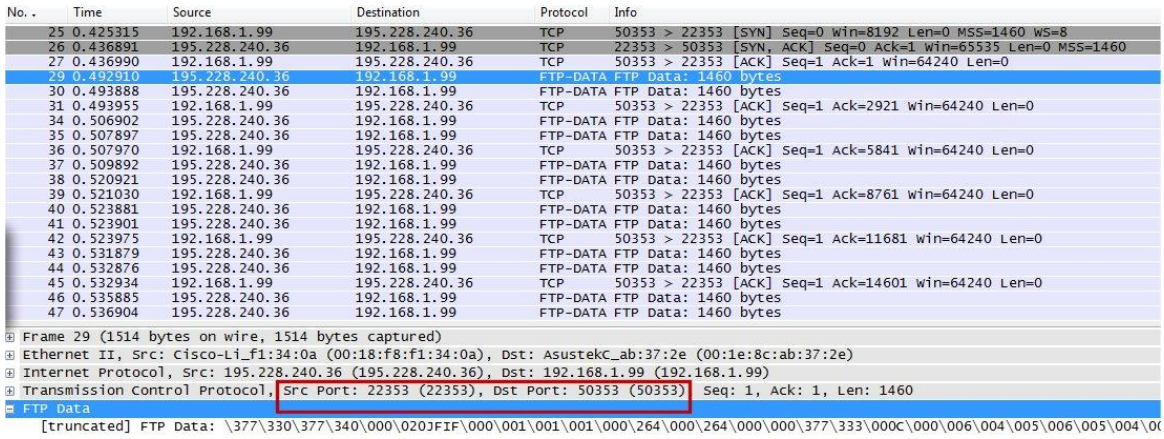


2.22. ÁBRA PASSZÍV FTP KLIENS

A kliens - még a Command csatornán - küld egy PASV parancsot. Ezzel passzív módba kapcsolja az FTP szervert. Válaszként a szerver küld egy portszámot - ahol immár ő várja majd a kliens kezdeményezését a Data csatorna kiépítésére. A kliens új sessiont nyit - immár egy másik portjáról - a szerver előzőleg megadott portjára.



2.23. ÁBRA A SZERVER VÁLASZA A PASV PARANCSRA



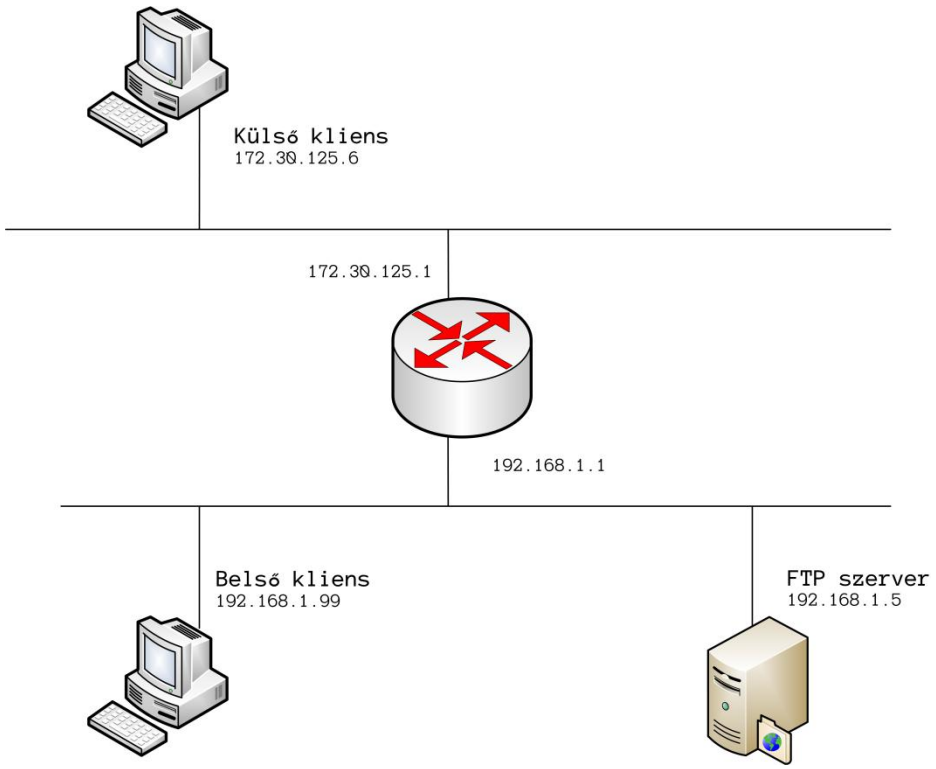
2.24. ÁBRA ADATFORGALOM

Mindez látható a valóságban is. A 22-es csomagban küldi el a kliens a PASV parancsot, a 23-asban válaszol a szerver, megadva a 22353-as portot. A 25-27 csomagokban épül fel az új TCP session és a 29-es csomagtól indul el az adatfolyam a Data csatornán.

2.2.3.3 FTP ÉS NAT

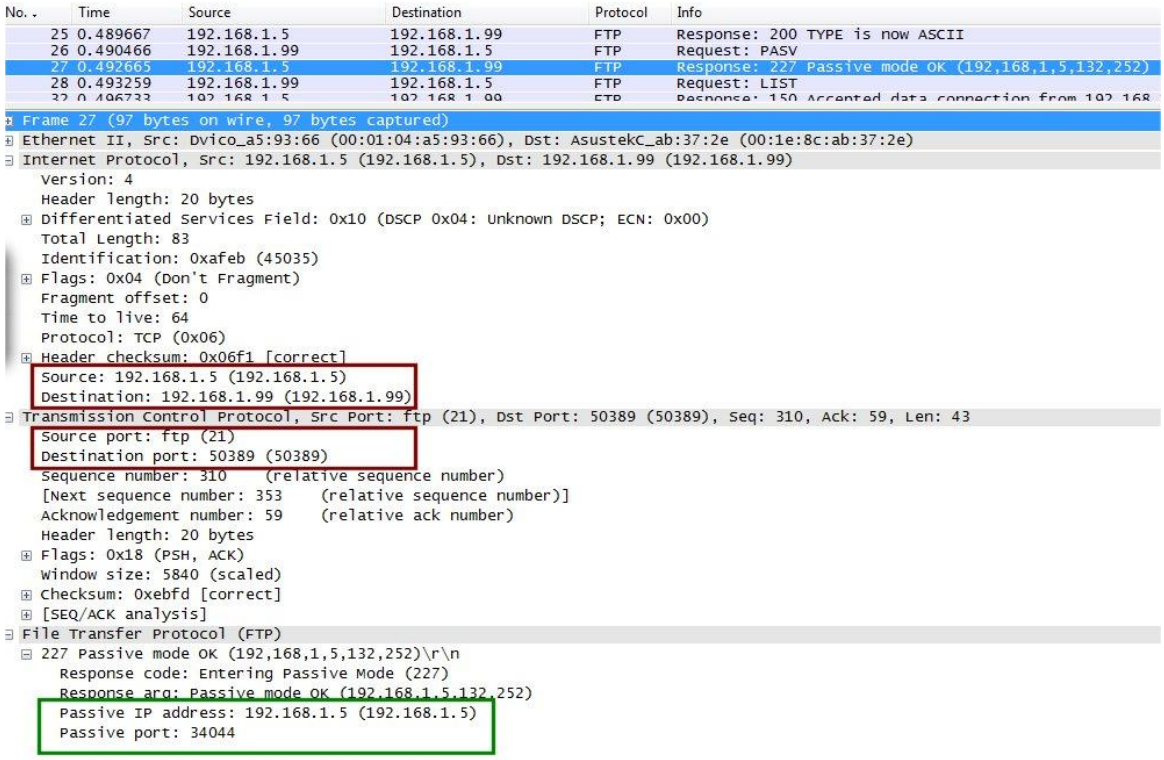
Nem motoszkál valami kellemetlen érzés ott hátul, a kisagyban?
Mindkét módszer esetén nagyon fontos IP címek és portszámok utaznak FTP csomagokban a TCP szegmenseken belül. Aktív esetben a PORT parancsban mondta meg a kliens, hol várja a szervert. Passzív esetben a PASV response parancsban a szerver mondta meg ugyanezt a kliensnek.

Mi van akkor, ha a kettő közé beépül egy NAT router?



2.25. ÁBRA FTP ÉS NAT

Ugye tudjuk, a NAT (oké, PAT) IP, illetve Transport réteg szinten működik. Az IP datagramban a forrás IP címeket, a TCP szegmensben a forrás portszámot írja át.



2.26. ÁBRA IP CÍMEK ÉS PORTOK

A fenti ábra azt mutatja, amikor a belső kliens kapcsolódik a belső FTP szerverhez. Ekkor nincs gond, mert nincs NAT.
De mi van, ha a külső kliens akarja elérni az FTP szervert? A router ki fogja cserélni a 172.30.125.6 IP címet a 192.168.1.1 címre, a kliens portszámát meg egy másikra. Csakhogy az FTP csomagban ott marad a régi IP és az a port, ahol majd a kliens rányomulna a szerverre.

Nem fog működni.

Innentől a NAT routerek társadalma két rétegre tagozódik:

- Azokra, akik ismerik a TCP protokollt és cserélgetik az FTP csomagon belül is a címeket, portokat. Ez már gyakorlatilag FTP applikációs proxy.
- És azokra, akiken nem megy át az FTP.

2.3 SMTP - A NÉPSZERŰ ÖREG

Huh. Végre egy kicsit itthon.

Eddig minden protokoll ismertetését azzal kezdtem, hogy beírtam az aktuális RFC számát. Ám, legyen.

Teljesség nélkül a fontosabb szabványok:

2.8. TÁBLÁZAT

RFC	Megnevezés	Dátum
RFC 0821	Simple Mail Transfer Protocol (STD0010)	1982
RFC 0822	Internet Message Format (STD0011)	1982
RFC 1123	Requirements for Internet Hosts (STD0003)	1989
RFC 1652	SMTP Extension for 8bit-MIME	1994
RFC 1869	SMTP Service Extensions (ESMTP)	1995
RFC 1870	SMTP Extension for Msg. Size Declaration	1995
RFC 1985	SMTP Extension for Remote Message Queue Starting	1996
RFC 2034	SMTP Extension for Enhanced Error Codes	1996
RFC 2045	Multipurpose Internet Mail Extensions (MIME) #1	1996
RFC 2046	Multipurpose Internet Mail Extensions (MIME) #2	1996
RFC 2047	Multipurpose Internet Mail Extensions (MIME) #3	1996
RFC 2048	Multipurpose Internet Mail Extensions (MIME) #4	1996
RFC 2049	Multipurpose Internet Mail Extensions (MIME) #5	1996
RFC 2142	Mailbox Names for Common Services and Roles	1997
RFC 2183	Content-Disposition Header Field	1997
RFC 2298	Message Disposition Notifications	1998
RFC 2476	Message Submission	1998
RFC 2505	Anti-Spam Recommendations for SMTP MTAs	1999
RFC 2554	SMTP Service Extension for Authentication	1999
RFC 2821	Simple Mail Transfer Protocol	2001
RFC 2822	Internet Message Format	2001
RFC 2920	SMTP Extension for Command Pipelining (STD0060)	2000
RFC 3030	SMTP Extensions for Large and Binary MIME	2000
RFC 3207	SMTP Extension for Secure SMTP over TLS	2002
RFC 3461	SMTP Extension for Delivery Status Notifications	2003
RFC 3463	Enhanced Mail System Status Codes	2003
RFC 5321	Simple Mail Transfer Protocol	2008
RFC 5322	Internet Message Format	2008

Nos, rögtön látható, hogy a levelezés már nem lesz olyan egyszerű téma.

Ha alaposan megnézzük a listát, látható, hogy vannak benne ismétlődések. Az RFC 5321 például kinyírta a 2821-et, illetve a 0821-et. Az RFC 5322 pedig a 2822-est és a 0822-est. Erre azért alaposan oda kell figyelni, tekintve, hogy ez a két RFC alkotja az internetes levelezés gerincét.

Amikor SMTP-ről beszélünk, hallgatólagosan két szabványról beszélünk. Már eredetileg is így születtek: a 821-es mellett ott figyelt a 822-es is. Az első foglalkozott azzal, hogyan vándorolnak a levelek. Postás hasonlattal élve, ez az RFC írta le azt, hogyan kell címezni egy borítékot és hogyan kell azt továbbítani a címzetthez.

A 822-es RFC viszont magának a levélnek a tartalmával foglalkozott. Hogyan kell kinéznie egy levélnek: megszólítás, bevezetés, tárgyalás, befejezés, aláírás.

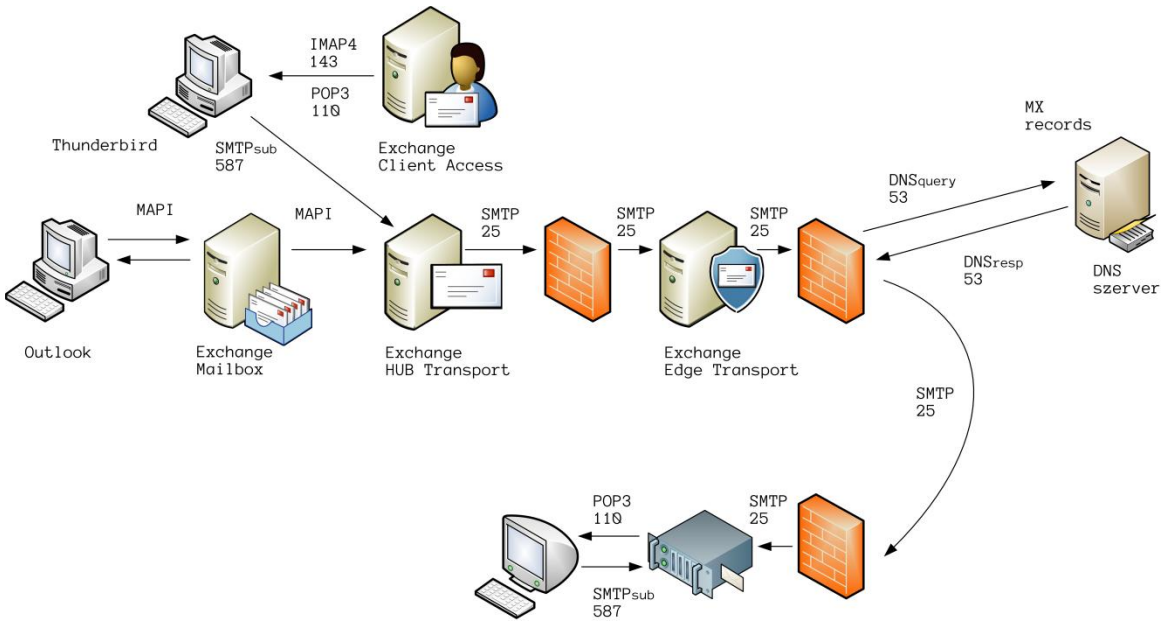
A két szabvány nagyjából párhuzamosan mozgott: a nagy renoválások alkalmával mindkettő egyaránt változott. (A köztes időben pedig hol az egyikhez jöttek ki módosító szabványok, hol a másikhoz.)

Mi is így fogjuk áttekinteni a protokollt.

2.3.1 Levelek továbbítása, azaz a konkrét SMTP

RFC 5321

Hogy képben legyünk, kezdjük egy ábrával.



2.27. ábra Általános levelezési történet

Ez egy teljesen általánosnak mondható séma. Van egy cég, akik Exchange 2007 levelezési infrastruktúrát használnak. Van egy Mailbox szerverük, egy HUB Transport szerverük és egy Client Access szerverük (ez a három lehet ugyanaz is), smarthostként egy Edge Transport szervert használnak. Zömében Outlook klienseik vannak, de néhány erős, önálló egyéniség inkább Mozilla Thunderbird kliensről nyomul.
Az Outlook kliensek MAPI-n keresztül kapcsolódnak a Mailbox szerverhez, mely szintén MAPI-n keresztül a HUB Transport szerverhez.

A Thunderbird felhasználók élete egy kicsit izgalmasabb, ők küldéskor a HUB Transport szerverhez kapcsolódnak SMTP-Submission protokollon keresztül, letöltéskor viszont a Client Access szerverhez vagy POP3, vagy IMAP4 protokollon keresztül.

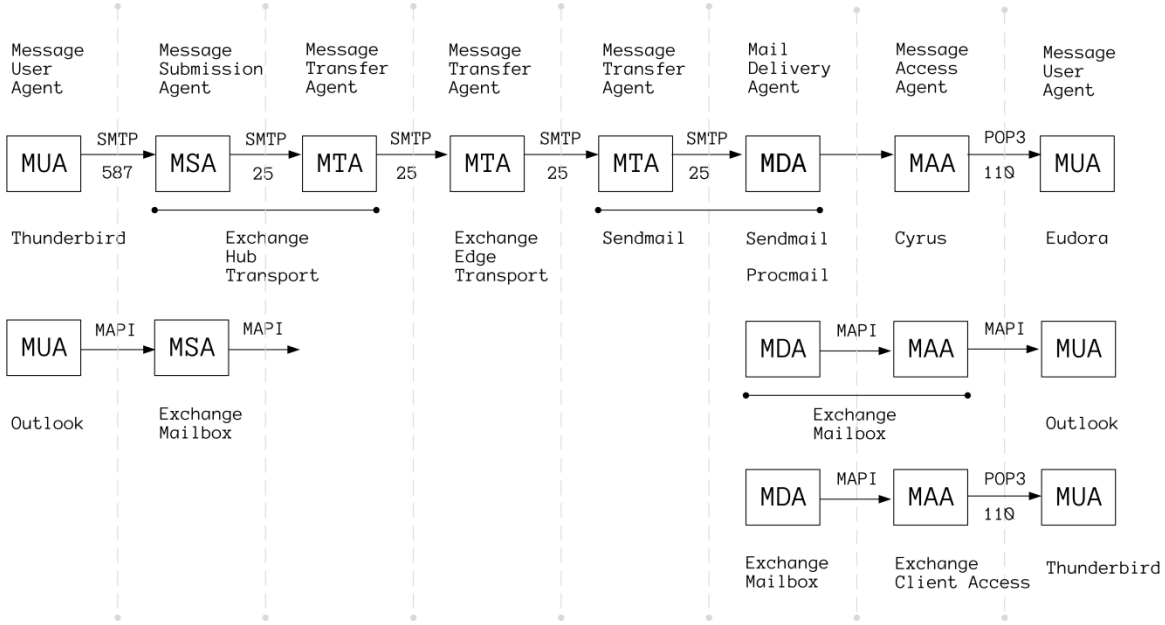
A lényeg, hogy előbb-utóbb minden küldendő levél a HUB Transport szerveren köt ki. Ez a szerver SMTP-MTA (a továbbiakban SMTP) protokollon keresztül továbbítja a levelet a DMZ-ben lévő smarthostnak.

Az Edge Transport szervernek, mielőtt továbbküldené a levelet, meg kell találnia, hová is küldje. A levél címzettjének suffixéből kiolvassa a tartománynevet és egy DNS szervertől lekérdezi a tartományhoz tartozó MX (Mail eXchanger) rekordot. Ez mutat a tartomány levelezőszerverének címére.

Megvan a cím, az Edge Transport szerver SMTP protokollon keresztül áttolja a túlsó SMTP szervernek a levelet, ahonnan a túloldali felhasználók valamilyen módszerrel leszedik azt.

Ugye, nem bonyolult?

Mindjárt az lesz.



2.28. ÁBRA AZ ELEKTRONIKUS LEVELEZÉS SZEREPLŐI

Bár az első ábrán nem látszik, de valójában ennyi komponens teszi szorgalmasan a dolgát, miközben elküldesz egy levelet a melletted ülő kollégádnak, hogy jön-e ebédelni.
Ahelyett, hogy szénné magyaráznám a fogalmakat, próbáljuk összepontozni a két ábrát. Látjuk, hogy a MUA az gyakorlatilag a kliensprogram. Mely konkrétan a felhasználó (fúj) asztalán lévő számítógépben ketyeg.

Amikor a felhasználó levelet akar küldeni, akkor első körben ez a levél feliratkozik küldésre. Ezt végzi az MSA, a Message Submission Agent. Ha Outlookból küldök, akkor gyakorlatilag a Mailbox szervert rugdosom meg, hogy tegye bele a levelet a HUB Transport szerver Submission várakozási sorába. Ha Thunderbird-ből, akkor közvetlenül a HUB Transport szerverhez fordulunk, méghozzá a submission SMTP protokollon keresztül. (Ugye mindenki emlékszik arra a trükös client receive konnektorra?)

Innentől Message Transfer Agent-ek (MTA) hosszú sora következik. Ezek hopról hopra[3] passzolgatják SMTP protokollon keresztül a leveleket, amíg azok el nem jutnak a címzett MTA szerveréhez. (MX rekord.)

Általában az MTA-nak van éppen elég dolga a levelekkel (routolás, higiénia), ezért külön szokták választani azt a funkciót, mely már konkrétan a levelek felhasználókhoz szállítását intézi. Ezt úgy hívják, hogy Mail Delivery Agent (MDA) és a kifejezés az Exchange 2007 szerver esetében a Mailbox szervert takarja.
A felhasználóknak innentől nincs is más dolguk, mint elvinni a levelüket. Ha Exchange 2007 szerverünk van és Outlook kliensünk, akkor egyszerűbb a helyzet, mert a Message Access Agent (MAA) az ugyanaz a szereplő, mint az MDA. (Megjegyzem, ez az állítás Exchange 2010 szerverre már nem igaz. Ott az MAA funkció minden kliens esetén - tehát a MAPI/RPC-t használó Outlook esetében is - átkerült a Client Access szerverre.) Ha nem Exchange szerverünk van, vagy nem MAPI kliensből levelezünk, akkor az MAA egy POP3/IMAP4 szerver, mely Exchange 2007 esetében a Client Access szerver.

Nos, gyakorlatilag megérkeztünk. A túloldali MUA vagy MAPI-n, vagy POP3/IMAP4 protokollokon keresztül egyszerűen letölti a levelét.

Akit mélyebben érdekel az Exchange 2007 lelkivilága, minden részrehajlás nélkül tudom neki ajánlani az alábbi könyvet:
Petrényi József: Exchange 2007 spontán
http://www.microsoft.com/hun/technet/article/?id=e1d46c3f-dbb8-4fdb-865a-73c3b9cce433

[3] Eltekintve persze az Exchange 2007 Direct Relay elvétől.

Habár az előző szövegben elhangzott egy csomó protokollnak a neve (POP3, IMAP4, MAPI, RPC), ezeket itt nem fogom kibontani. Tudjunk róla, hogy léteznek.

Lazításképpen elmesélem, miért pont ezt a címet adtam az alfejezetnek.

Ahogy egy örökkévalósággal ezelőtt is írtam, a hetvenes évek vége felé érdekes konkurenciaharc volt a hálózati architektúrák terén. Egyszerre kezdték szabványba önteni az ún. 7 réteges ISO-OSI modellt és nagyjából ugyanabban az időben formálódott a 4 réteges TCP/IP is. Nagy küzdelem volt. Hogy manapság a 7 réteges modellel már csak a fanatikusok találkoznak, a TCP/IP-vel pedig még a keresztanyám is (csak éppen nem ismeri fel) - az nem kis mértékben a rendkívül népszerű SMTP-nek köszönhető.

- Hohó - mondhatja erre a figyelmes olvasó - Az SMTP első RFC-je 1982-ben készült. Hogyan vitézkedhetett volna ez a hetvenes években?

Úgy, hogy az SMTP-nek voltak előzményei. Már a hatvanas években is léteztek kezdetleges üzenetküldő alkalmazások, melyek 1973 és 1980 között, amikor az ARPANET kezdte kiforrni magát és kezdett belőle kialakulni az internet, már vadul tették a dolgukat. Jon Postel 1980-ban foglalta ezeket össze Mail Transfer Protocol néven - melyből végül 1982-ben alakult ki a Simple Mail Transfer Protocol, azaz az SMTP.

Ez a kezdeti népszerűség később csak fokozódott. Manapság az SMTP és a HTTP fej fej mellett állnak a legnépszerűbb netes protokollok között. És van egy olyan szempont, ahol az SMTP bőven le is körözi a HTTP-t: ez a nélkülözhetetlenség.
Meglehetősen sok ügyfelünk van, de az email mindenhol kiemelten kritikus rendszernek minősül. Ha nincs web, akkor legfeljebb nem böngésznek annyit az alkalmazottak. Ha nem működik a levelezés, akkor viszont megáll az élet.

Ez a nagy népszerűség persze nem csak előny. Gúzsba is köti az SMTP-t. Harminc évvel ezelőtt kicsit más volt a világ, mint ma. A protokoll változott ugyan, de ma már annyi, de annyi SMTP szerver van a világon, hogy a rendszer tehetetlensége miatt gyökeres átalakításokra kevés az esély. Pedig igény lenne rá: a levelezés 75-80%-át manapság a levélszemét teszi ki. Spam, malware, phishing. Ezek mind azért ilyen virulensek, mert az SMTP eredeti hiányosságait használják ki.
(Egyet meg is fogunk nézni.)

Kipihentük magunkat, folytathatjuk.

2.3.1.1 SMTP, ESMTP PARANCSOK

Amikor indult az SMTP, a következő parancsokat tartalmazta:
HELO, MAIL FROM, RCPT TO, DATA, RSET, SEND FROM, SOML FROM, SAML FROM, VRFY, EXPN, HELP, NOOP, QUIT, TURN

Aztán jött néhány ráncfelvarrás (egy szekérderék RFC) és most nagyjából így állunk:

2.9. TÁBLÁZAT

Parancs	Leírás	Visszavonva
8BITMIME	A levél kiterjesztett karakterkészletet tartalmaz	
ATRN	Ugyanaz, mint a TURN, csak autentikáció nélkül nem megy	
AUTH	Autentikáció	
BDAT	Bináris adat	
BINARYMIME	Bináris MIME adat	
BURL	A levél tartalma egy IMAP szerverről jön	
CHECKPOINT	Nagy levelek megbízhatatlan vonalon - a checkpoint jeltzi, hogy eddig minden rendben van	
CHUNKING	DATA helyett BDAT	
DATA	A levél tartalma következik	
DELIVERBY	Megadott időn belüli továbbítás	
DSN	Delivery Status Notification, azaz értesítés a levéltovábbítás alakulásáról	
EHLO	Extended HELO.	
ENHANCEDSTATUSCODES	A szerver a kibővített stuszkód készletet ismeri	
ETRN	Kibővített TURN: az a trükkje, hogy szerverek között is működik.	
EXPN	Kibontja a címlistákat	
FUTURERELEASE	Jövőben elküldendő levél összeállítása	
HELO	Üdv	
HELP	Help	
MAIL FROM	Feladó a borítékon	
MTRK	Message Tracking.	
NO-SOLICITING	Nem kérünk spamszűrést	
NOOP	Semmi	
ONEX	Csak egy üzenet megy	
PIPELINING	Egy munkameneten belül több parancs	
QUIT	Viszlát és kösz a halakat	
RCPT TO	A boríték címzettje	
RSET	Kezdjük újra	
SAML	Send And MaiL (Terminálról)	Igen
SEND	Send (Terminálról)	Igen
SIZE	Levélméret megadása	
SOML	Send Or MaiL (Terminálról)	Igen
STARTTLS	Térjünk át TLS-re.	
SUBMITTER	A kleins megadhat egy olyan emailcímet, mely a levelek feladásáért felelős (Responsible Submitter)	
TURN	A kliens és a szerver szerepet cserélnek - azaz a kliens magára húzza a leveleit.	Igen
UTF8SMTP	Nemzetközi emailcímek	
VERB	Bőbeszédűen. Hibakeresésre.	
VRFY	Létezik-e a postafiók?	

Egyáltalán nem kötelező, hogy egy levelezőszerver az összes parancsot ismerje. Létezik egy minimum készlet, azt igen - de azon felül már egyrészt magán az SMTP szerveren, másrészt a rendszergazdán múlik, mit enged. Például a VRFY parancsot a

'*' paraméterrel határozottan tiltja mindenki (ne tudjanak a spammerek címeket begyűjteni), de van olyan termék (pl. az Exchange is), mely alapértelmezésben magát a parancsot sem engedélyezi.

Jogos lehet a kérdés, hogyan tudják eldönteni a szerverek, hogy ki melyik parancsot ismeri? Nos, ha az egyik szerver a HELO paranccsal köszön be a másiknak, akkor sehogy. Ellenben ha az EHLO parancsot használja, akkor válaszként megkapja az ismert parancsok listáját.

2.3.1.2 SMTP KÓDOK

A szerverek által visszaadott kódokkal ugyanaz a csúfság történt, mint a parancsokkal: valamikor létezett egy szűkebb kör, melyet később RFC-k hosszú során keresztül bővítgettek. Lényeges különbség, hogy itt a kommunikáció során nem állnak neki tisztázni a szerverek, hogy az eredeti küldő oldal érti-e a válaszkódot. Megkapja, oszt azt csinál vele, amit akar.

2.10. TÁBLÁZAT

Kód	Leírás
211	System status, or system help reply.
214	Help message.
220	*Domain* service ready. Ready to start TLS.
221	*Domain* service closing transmission channel.
250	OK, queuing for node *node* started. Requested mail action okay, completed.
251	OK, no messages waiting for node *node*. User not local, will forward to *forwardpath*.
252	OK, pending messages for node *node* started. Cannot VRFY user (e.g., info is not local), but will take message for this user and attempt delivery.
253	OK, *messages* pending messages for node *node* started.
354	Start mail input; end with <CRLF>.<CRLF>.
355	Octet-offset is the transaction offset.
421	*Domain* service not available, closing transmission channel.
432	A password transition is needed.
450	Requested mail action not taken: mailbox unavailable. ATRN request refused.
451	Requested action aborted: local error in processing. Unable to process ATRN request now
452	Requested action not taken: insufficient system storage.
453	You have no mail.
454	TLS not available due to temporary reason. Encryption required for requested authentication mechanism.
458	Unable to queue messages for node *node*.
459	Node *node* not allowed: *reason*.
500	Command not recognized: *command*. Syntax error.
501	Syntax error, no parameters allowed.
502	Command not implemented.
503	Bad sequence of commands.
504	Command parameter not implemented.
521	*Machine* does not accept mail.
530	Must issue a STARTTLS command first.

	Encryption required for requested authentication mechanism.
534	Authentication mechanism is too weak.
538	Encryption required for requested authentication mechanism.
550	Requested action not taken: mailbox unavailable.
551	User not local; please try *forwardpath*.
552	Requested mail action aborted: exceeded storage allocation.
553	Requested action not taken: mailbox name not allowed.
554	Transaction failed.

Szándékosan hagytam meg az angol szöveget: a szerver is így fog válaszolni.

Vegyük észre, hogy a visszajelzések csoportosítása ugyanazt a számozási logikát követi, mint a HTTP, illetve FTP esetében. (Eltekintve a 3xx kategóriától. A rekurzióról ugyanis az SMTP kapcsán túl sok értelme nincs beszélni, azaz a kódmező felhasználható másra.)

2.3.2 AZ ELEKTRONIKUS LEVELEK SZERKEZETE, AZAZ IMF

RFC 5322

IMF: Internet Message Format.
Azaz hogyan kell kinéznie egy virtigli levélnek. (Figyelem, immár a borítékon belül vagyunk.)

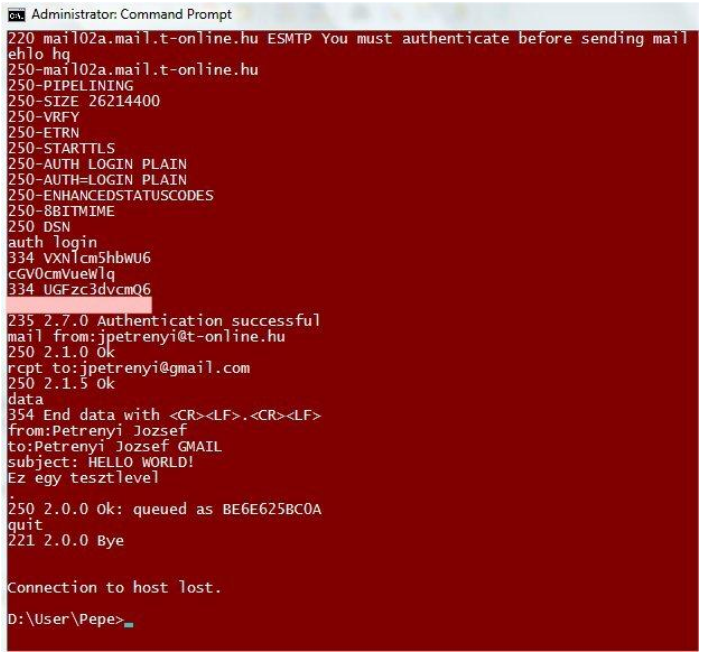
Egy levélen belül alapvetően két nagy részt különítünk el:

- Header: a levélre vonatkozó kisérőinformációk.
- Body: Maga a levél tartalma: szöveg, csatolás.

Az SMTP viszonya a szabványokhoz meglehetősen laza. Egyáltalán nem kötelező kitölteni minden fejléc mezőt, egyáltalán nem kötelező bármit is írni a body részbe, sőt egyáltalán nem kötelező értelmes dolgokat írni magába a levélbe, mint ahogy a céges levelezésekben ez nem is ritkán előfordul. (A borítékra írt információk alapján a levél mindenképpen eljut a címzett levelezőszerverére - maximum a szerver szűri ki, mint spam.)

```
220 mail02a.mail.t-online.hu ESMTP You must authenticate before sending mail
ehlo hq
250-mail02a.mail.t-online.hu
250-PIPELINING
250-SIZE 26214400
250-VRFY
250-ETRN
250-STARTTLS
250-AUTH LOGIN PLAIN
250-AUTH=LOGIN PLAIN
250-ENHANCEDSTATUSCODES
250-8BITMIME
250 DSN
auth login
334 VXNlcm5hbWU6
cGV0cmVueWlq
334 UGFzc3dvcmQ6
Ez_itt_a_jelszavam_helye
235 2.7.0 Authentication successful
mail from:jpetrenyi@t-online.hu
250 2.1.0 Ok
rcpt to:jpetrenyi@gmail.com
250 2.1.5 Ok
data
354 End data with <CR><LF>.<CR><LF>
from:Petrenyi Jozsef
to:Petrenyi Jozsef GMAIL
subject: HELLO WORLD!
Ez egy tesztlevel
.
250 2.0.0 Ok: queued as BE6E625BC0A
quit
221 2.0.0 Bye
```

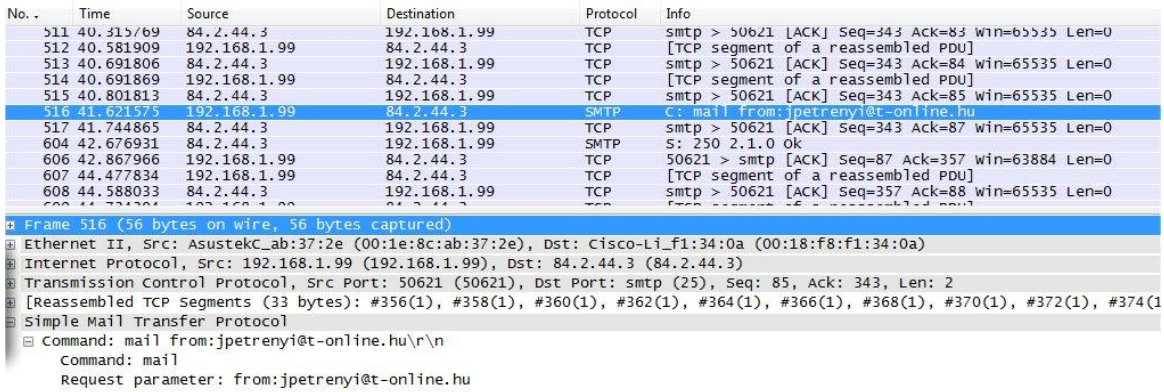
Ez a szövegrészlet egy capture fájlból való. Bekapcsoltam a sniffer programot, összeállítottam egy levelet parancssorból majd elküldtem.



2.29. ÁBRA TESZTLEVÉL PARANCSSORBÓL

A capture fájlban rászűrtem a konkrét kommunikációra, majd összerakattam a programmal a teljes adatáramot. Így kaptam az előző oldalon látható listát.

Itt pedig az elkapott csomagok láthatók.



2.30. ÁBRA PARANCSSORBÓL KÜLDÖTT LEVÉL DARABJA A CAPTURE FÁJLBAN

(Megjegyzem, az a sok szemétnek tűnő TCP csomag a lassú gépelésem miatt van. Istenem, nem vagyok egy Outlook.)

Ez az első élveboncolt levél a könyvben, el lehet rajta csemegézni. (Később látunk majd még eleget.) Megkereshetjük a parancsokat, kibogarászhatjuk a válaszokat.
Jó játék.

> Például észreveheted, hogy a levélben látható 334-es kódú válasz - igen, az az értelmezhetetlen zsizsa - nem szerepel a válaszkódok között. Nem is, ugyanis ez sokkal inkább egy challenge, mint kód. De erről majd később.

A lényeg a szövegben az, hogy mivel én a két saját kezemmel gépeltem be a parancsokat, igyekeztem minimalizálni azokat. Az volt a célom, hogy már értelmezhető levél legyen, még elfogadható mennyiségű parancs eredményeképpen.

Hasonlítsuk össze a próbálkozásomat az Outlook próbálkozásával. A levél feladója, a címzettje és a levél szövege azonos. Sőt, az elkapási módszer is.

```
220 mail01a.mail.t-online.hu ESMTP You must authenticate before sending mail
EHLO hq
250-mail01a.mail.t-online.hu
250-PIPELINING
250-SIZE 26214400
250-VRFY
250-ETRN
250-STARTTLS
250-AUTH LOGIN PLAIN
250-AUTH=LOGIN PLAIN
250-ENHANCEDSTATUSCODES
250-8BITMIME
250 DSN
AUTH LOGIN
334 VXNlcm5hbWU6
cGV0cmVueWlq
334 UGFzc3dvcmQ6
Még mindig a jelszavam helye
235 2.7.0 Authentication successful
MAIL FROM: <jpetrenyi@t-online.hu>
250 2.1.0 Ok
RCPT TO: <jpetrenyi@gmail.com>
250 2.1.5 Ok
DATA
354 End data with <CR><LF>.<CR><LF>
From: "Petrenyi Jozsef" <jpetrenyi@t-online.hu>
To: <jpetrenyi@gmail.com>
Subject: Hello World!
Date: Sun, 10 Jan 2010 11:31:24 +0100
Message-ID: <000001ca91e0$0f568270$2e038750$@hu>
MIME-Version: 1.0
Content-Type: multipart/alternative;
       boundary="----=_NextPart_000_0001_01CA91E8.711B1180"
X-Mailer: Microsoft Office Outlook 12.0
Thread-Index: AcqR4A7u+m0GG6/8SpKyPhj1+sMNtQ==
Content-Language: hu

This is a multi-part message in MIME format.

------=_NextPart_000_0001_01CA91E8.711B1180
Content-Type: text/plain;
```

```
        charset="us-ascii"
Content-Transfer-Encoding: 7bit

Ez egy tesztlevel.


------=_NextPart_000_0001_01CA91E8.711B1180
Content-Type: text/html;
        charset="us-ascii"
Content-Transfer-Encoding: quoted-printable

<html xmlns:v=3D"urn:schemas-microsoft-com:vml" =
xmlns:o=3D"urn:schemas-microsoft-com:office:office" =
xmlns:w=3D"urn:schemas-microsoft-com:office:word" =
xmlns:m=3D"http://schemas.microsoft.com/office/2004/12/omml" =
xmlns=3D"http://www.w3.org/TR/REC-html40">

<head>
<META HTTP-EQUIV=3D"Content-Type" CONTENT=3D"text/html; =
charset=3Dus-ascii">
<meta name=3DGenerator content=3D"Microsoft Word 12 (filtered medium)">
<style>
<!--
 /* Font Definitions */
 @font-face
        {font-family:"Cambria Math";
        panose-1:2 4 5 3 5 4 6 3 2 4;}
@font-face
        {font-family:Calibri;
        panose-1:2 15 5 2 2 2 4 3 2 4;}
 /* Style Definitions */
 p.MsoNormal, li.MsoNormal, div.MsoNormal
        {margin:0cm;
        margin-bottom:.0001pt;
        font-size:11.0pt;
        font-family:"Calibri","sans-serif";}
a:link, span.MsoHyperlink
        {mso-style-priority:99;
        color:blue;
        text-decoration:underline;}
a:visited, span.MsoHyperlinkFollowed
        {mso-style-priority:99;
        color:purple;
        text-decoration:underline;}
span.EmailStyle17
        {mso-style-type:personal-compose;
        font-family:"Calibri","sans-serif";
        color:black;
        font-weight:normal;
        font-style:normal;}
..MsoChpDefault
        {mso-style-type:export-only;}
@page Section1
        {size:612.0pt 792.0pt;
        margin:70.85pt 70.85pt 70.85pt 70.85pt;}
div.Section1
        {page:Section1;}
-->
</style>
<!--[if gte mso 9]><xml>
 <o:shapedefaults v:ext=3D"edit" spidmax=3D"1026" />
</xml><![endif]--><!--[if gte mso 9]><xml>
 <o:shapelayout v:ext=3D"edit">
  <o:idmap v:ext=3D"edit" data=3D"1" />
```

```
 </o:shapelayout></xml><![endif]-->
</head>

<body lang=3DHU link=3Dblue vlink=3Dpurple>

<div class=3DSection1>

<p class=3DMsoNormal><span style=3D'color:black'>Ez egy =
tesztlevel.<o:p></o:p></span></p>

</div>

</body>

</html>

------=_NextPart_000_0001_01CA91E8.711B1180--

.
250 2.0.0 Ok: queued as 5CFB3797A02
QUIT
221 2.0.0 Bye
```

Na, itt aztán van minden, mint a búcsúban. A parancsok még hagyján, azok ugyanazok. De a levél fejléce... na meg az a vadító teste! Az bizony jócskán más.
Kezdjük ott, hogy az Outlook nálam úgy van beállítva, hogy a default formátum a HTML. A legtöbb cifra dolgot rögtön ez magyarázza. Jóval több kisérő információ kellett hozzá. Mivel a HTML esetében nagyon fontos a formázás is, így a stílus külön XML betétként utazik a levéllel. Sőt, látható, hogy a tartalom benne van a levélben plain text formában - és benne van HTML lapként is.
Tiszta öröm.
De tényleg. Remek példa arra, hogy milyen bonyolult is tud lenni egy csatolásokat, nem angol beállításokat használó levél, illetve hogy mennyi érdekes fejléc is létezik a világban.

> RFC 1426, 1521, 1847, 3156, 2045, 2046, 2047, 2048, 2049, 2183, 2231, 2387, 4288, 4289

Illetve remek alkalom, hogy ejtsünk pár szót arról a roppant kellemetlen érzéseket okozó valamiről, ami a MIME.
Már a neve is olyan... izé. Multipurpose Internet Mail Extensions. Ilyen idetekergetem, odatekergetem, bármit is jelenthet. Ha meg meg akarom nézni a szabványtárban, rögtön egy égigérő szabványkupacon kell keresztülrágnom magamat.

Megéri?

Átbogarászni nem biztos. De használni, tudni, hogy nagyjából miről is van szó, azt mindenképpen.

Mielőtt a MIME megjelent volna, az élet egyszerű volt. Levélírásra használhattuk az ASCII tábla alsó felét (7 bites kódkészlet), méghozzá a 127 karakterből rögtön le is kellett vonni a 32 vezérlőkaraktert, A maradékot nevezzük *printable* karaktereknek. De itt még le kell vonnunk a '.' - azaz 'pont' - karaktert is. Ez ugyanis a levél lezáró karaktere. (Nézd meg a példákat: a DATA paranccsal lépünk be a levélösszerakási részbe és a '.' karakterrel zárjuk azt be.)

> Nagyon hamar felismerheted, kik azok az öreg rutinok, akik ebben a korban szocializálódtak: ők azok, akik a mai napig nem használnak ékezeteket a levelezésben.

Csatolás... az nem volt.

Ahogy az internet egyre inkább megszűnt csak a katonák és a tudósok játszótere lenni, úgy szaporodtak az igények is az egyes protokollokkal szemben. Miért ne lehetne más, nemzeti kódkészletet használni a levélben?
Elismerem, ez is nagy kényszer lehetett a protokoll megújítói számára. De a legnagyobb nyomást feltehetően az tette rájuk, amikor megpróbáltak nekik egy pucérnős képet átküldeni, de az istennek sem bírta el a 7 bit[4].

Nos, jó ha tudod, hogy az elektronikus levél formátuma a mai napig 7 bites. Ugyanazt a néhány karaktert használjuk, mint a hősi időkben.
Minden mást, azaz a 8 bites kódolású teljes kódtáblákkal íródott szövegeket, a binárisnak tekintett csatolt fájlokat, mindent, külön át kell kódolni 7 bitesre.

A MIME gyakorlatilag ezzel foglalkozik. A következő funkciói vannak:

- Olyan szöveges levelek kezelése, ahol a levél törzsében kiterjesztett kódú karakterek (például ékezetesek) vannak.
- Nem szöveges csatolások kezelése.
- Olyan levelek kezelése, melyek fejléc imformációiban (címek, subject, stb) találhatóak kiterjesztett kódú karakterek.
- Amennyiben több csatolás van a levélben, vagy több, különböző ASCII kódú rész, vagy a kettő együtt, akkor tudnia kell kezelni a több MIME blokkot is, azaz menedzselnie kell a saját bonyolultságát.

Látható, a MIME egy külön birodalom az SMTP-n belül. Saját fejlécei vannak, képes blokkokra osztani a levél törzsét, annak tartalmától függően. Természetesen ehhez a levelező klienseknek is MIME kompatibilisnek kell lenniük, de ez ma már nem probléma.

[4] Bár az öreg rutinok erre azt mondják, hogy ugyanmár, ott volt az ASCII grafika.

Vizsgáljuk meg néhány példán, hogyan is néz ez ki.

Ha visszalapozol pár oldalt ahhoz a nagyon hosszú levélhez, szépen láthatod, hogyan is épül fel ez az egész. Először jön a levél fejléce.

```
DATA
354 End data with <CR><LF>.<CR><LF>
From: "Petrenyi Jozsef" <jpetrenyi@t-online.hu>
To: <jpetrenyi@gmail.com>
Subject: Hello World!
Date: Sun, 10 Jan 2010 11:31:24 +0100
Message-ID: <000001ca91e0$0f568270$2e038750$@hu>
MIME-Version: 1.0
Content-Type: multipart/alternative;
        boundary="----=_NextPart_000_0001_01CA91E8.711B1180"
X-Mailer: Microsoft Office Outlook 12.0
Thread-Index: AcqR4A7u+m0GG6/8SpKyPhj1+sMNtQ==
Content-Language: hu
This is a multi-part message in MIME format.
```

Láthatod, hogy magába a fejlécbe is belekerültek már MIME információk. Ilyen a MIME verziószáma, de ilyen a Content-Type változó is. (A változó tartalmazza a MIME blokkok határait jelző azonosítót is.)
Plusz a végén ott van egy fenyegetés is, miszerint a levélben több MIME blokk is lesz.

```
------=_NextPart_000_0001_01CA91E8.711B1180
Content-Type: text/plain;
        charset="us-ascii"
Content-Transfer-Encoding: 7bit

Ez egy tesztlevel.
```

Ez az első MIME blokk. A tartalom tipusa text/plain, a használt karaktertábla us-ascii, a kódolás pedig 7bit. Ez azt jelenti, hogy a blokk minden karaktere printable - azaz a MIME-nek ezzel a darabbal az égegyadta világon semmi dolga sincs.

```
------=_NextPart_000_0001_01CA91E8.711B1180
Content-Type: text/html;
        charset="us-ascii"
Content-Transfer-Encoding: quoted-printable
```

A második blokkot nem másolom teljesen ide, mert borzasztó hosszú. De ha átnézed, láthatod, hogy ez ugyanaz a tartalom, mint az első MIME blokkban, csak ez HTML kódú. Azaz tele van formázó utasítással - de a tartalom az ugyanaz a rövid mondat.

Viszont érdemes elmeditálni a kódoláson. Ez már nem 7bit, ugyanis a HTML kódban vannak tiltott karakterek is. Ekkor jön be a képbe a quoted printable kódolás.
Ez a következőképpen működik. Fogja a 8 bites kódot és egy három karakterből álló sorozattal váltja ki. Az első karakter mindig az egyenlőségjel, a másik kettő pedig a karakter ASCII kódja, hexadecimális formában.

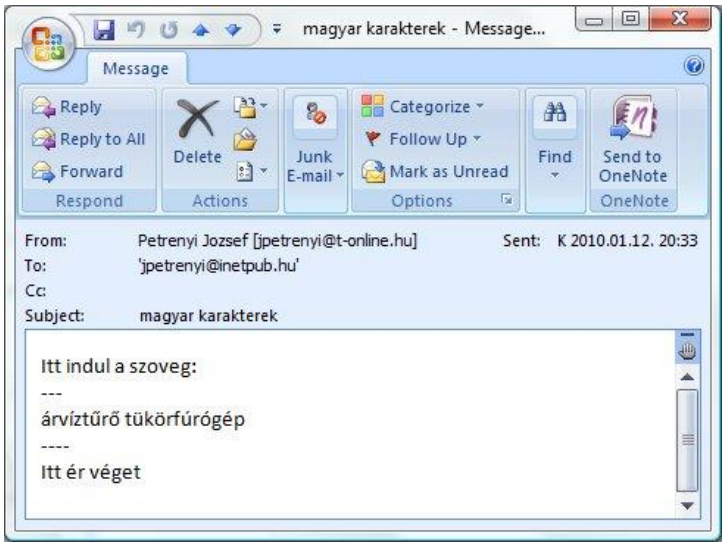
```
<html xmlns:v=3D"urn:schemas-microsoft-com:vml" =
xmlns:o=3D"urn:schemas-microsoft-com:office:office" =
xmlns:w=3D"urn:schemas-microsoft-com:office:word" =
xmlns:m=3D"http://schemas.microsoft.com/office/2004/12/omml" =
xmlns=3D"http://www.w3.org/TR/REC-html40">
```

Rögtön így kezdődik a MIME blokk. Látunk benne quoted printable kódot? Persze, ott van az '=3D' sorozat. Kódoljuk vissza. 3D -> 61; az us-ascii tábla 61-es kódja pedig pont az '=', azaz az egyenlőségjel kódja.

http://en.wikipedia.org/wiki/US-ASCII#ASCII_printable_characters

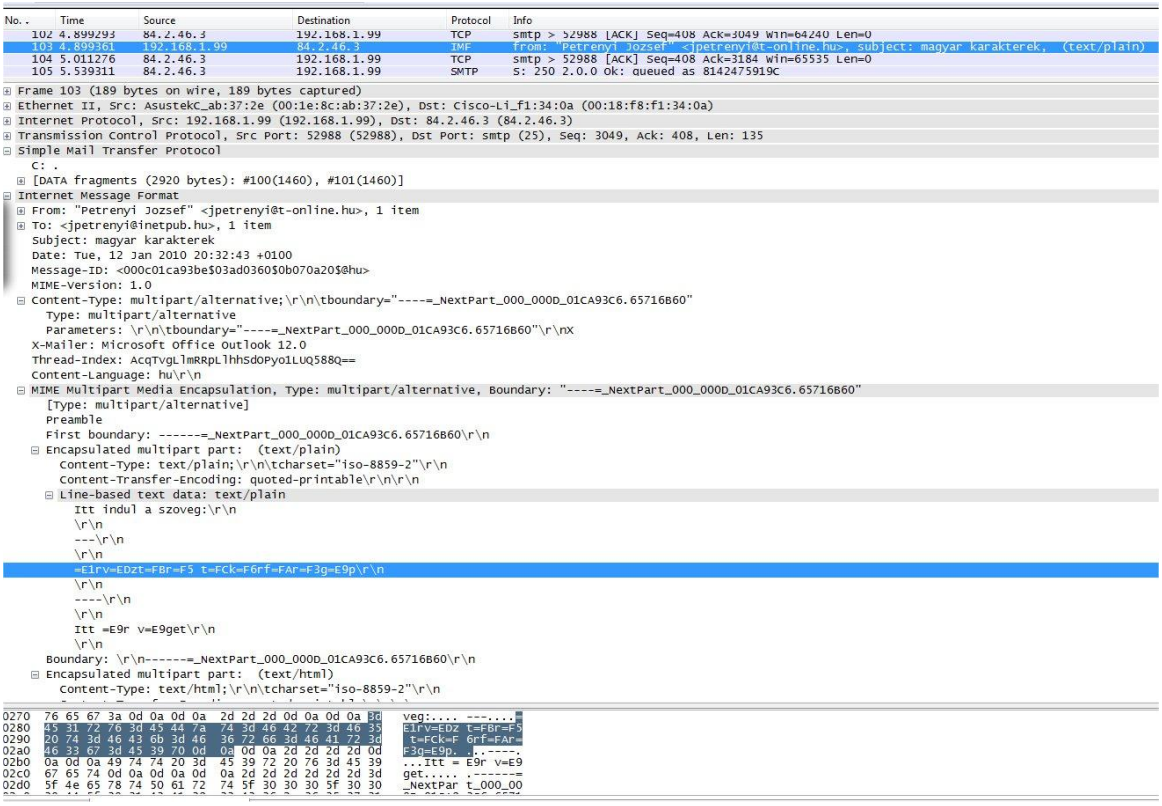
Hoppá. Az egyenlőségjellel jelezzük, hogy a berakott illegális karakter az egyenlőségjel?
Ez csak első ránézésre holdat nyalogató baromság. Ugyanis ha a quoted printable kódolásnál az egyenlőségjel az egy foglalt vezérlőkarakter, akkor tényleg illegális a használata. A számítógép nem tudja intuitíven eldönteni, hogy a szövegben előforduló egyenlőségjel most éppen vezérlőkarakter-e vagy sem.

Nézzünk egy egyértelműbb példát. Írtam egy másik levelet.



2.31. ÁBRA TESZTLEVÉL

Nézzük meg a capture fájlt.



2.32. ÁBRA ÁRVÍZTŰRŐ TÜKÖRFÚRÓGÉP

Milyen protokollnak is érzékeli a Wireshark a csomagot? IMF. Aranyos.

Bármennyire is hülyén hangzik, de a kék csíkkal jelzett szövegrészlet az a bizonyos 'árvíztűrő tükörfúrógép' karaktersorozat.

Nézzük meg a MIME blokkot.

```
------=_NextPart_000_000D_01CA93C6.65716B60
Content-Type: text/plain;
        charset="iso-8859-2"
Content-Transfer-Encoding: quoted-printable

Itt indul a szoveg:

---

=E1rv=EDzt=FBr=F5 t=FCk=F6rf=FAr=F3g=E9p

----

Itt =E9r v=E9get
```

Az iso-8859-2 kódtábla már szigorúan 8 bites. Vannak benne 128-nál magasabb sorszámú ASCII karakterek is. Itt már nem csak a tiltott egyenlőségjel jelenik meg, hanem olyan karakterek is, melyek nem tartoznak a printable csoportba.

Szúrópróbaszerűen kódoljunk vissza néhány karaktert.

2.11. TÁBLÁZAT

Quoted printable kód	Decimális érték	Az iso-8859-2 kódtáblázat kódja
=E1	225	á
=ED	237	í
=FB	251	ű
=F5	245	ő

http://en.wikipedia.org/wiki/ISO/IEC_8859-2

Oké. Ezzel a módszerrel elértük, hogy - igaz, egy karakter helyett hárommal - de le tudjuk írni az összes karaktertábla összes elemét.

De mi van akkor, ha csatolást rakunk a levélbe?

MIME blokk:

```
------=_NextPart_000_0070_01CAB73D.4D8BB510
Content-Type: text/plain;
        name="teszt.txt"
Content-Transfer-Encoding: base64
Content-Disposition: attachment;
        filename="teszt.txt"

VGV0c3r1bGVnZXMgdHh0IGbhamwuDQoNCsFydu16dPty9SB0/Gv2cmb6cvNn6XAu

------=_NextPart_000_0070_01CAB73D.4D8BB510--
```

Szemmel láthatóan bejöttek újabb változók a MIME blokk fejlécébe. A leginkább szembetűnő, hogy egy újabb kódolással találkozunk: ez a base64. Volt már róla szó a könyvben, jól le is dorongoltuk. Ez persze nem azt jelenti, hogy nem szeretjük - pusztán csak arról van szó, hogy mindenkit a maga helyén szeretünk. A base64 kódolás tökéletes arra, hogy bármilyen 8 bites értékből 7 bites értéket állítson elő - de borzasztóan béna, ha arról van szó, hogy egy karaktersorozatot titkosítsunk.

Ezzel el is árultam, mire használja a MIME a base64 kódolást: arra, hogy mindent, ami csatolt fájl, gyorsan bájt szinten lekódol 7 bitesre, majd így már bele tudja illeszteni - MIME blokkokon keresztül - az SMTP DATA mezőbe.

Magáról a kódolásról nem szándékozom sokat írni. Nagyjából arról van szó, hogy 3 bájtos blokkokat négyfelé szedünk (ekkor egy egység hat bites lesz), ezeknek ugye 64 különböző értéke lehet. Minden értékhez egy karaktert rendelünk az ún. Base64 ABC-ből. (Ez egy 64 elemű karaktertáblázat, szigorúan semleges printable karakterekből.) A felhígulási arány 1,37, azaz a 8 bites kódolású fájl 1,37-szer lesz nagyobb base64 kódolás után. (Plusz 814 bájt fejléc.)

http://en.wikipedia.org/wiki/Base64

Gyors ellenőrzés.

2.33. ÁBRA SZÖVEGES FÁJL VISSZAFEJTÉSE

Habár a visszafejtés nem sikerült tökéletesen, de ez már a weblap hibája. Nem ismeri a magyar karaktereket.

Ömlesztve még néhány csatolás.

JPG:

```
-------=_NextPart_000_007F_01CAB73E.6761D2C0
Content-Type: image/jpeg;
        name="feed.jpg"
Content-Transfer-Encoding: base64
Content-Disposition: attachment;
        filename="feed.jpg"

/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDAAYEBQYFBAYGBQYHBwYIChAKCgkJChQODwwQFxQYGBcU
FhYaHSUfGhsjHBYWICwgIyYnKSopGR8tMC0oMCUoKSj/2wBDAQcHBwoIChMKChMoGhYaKCgoKCgo
KCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCj/wAARCAEAAQADASIA
```

MP3

```
------=_NextPart_000_0091_01CAB73F.CEC85AA0
Content-Type: audio/mpeg;
        name="cohen-mbx.mp3"
Content-Transfer-Encoding: base64
Content-Disposition: attachment;
        filename="cohen-mbx.mp3"

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

(A fájl maga rengeteg space karakterrel kezdődik. Amiatt van a sok A karakter.)

XLS:

```
------=_NextPart_000_008B_01CAB73F.3644C660
Content-Type: application/vnd.ms-excel;
        name="abc.xls"
Content-Transfer-Encoding: base64
Content-Disposition: attachment;
        filename="abc.xls"

0M8R4KGxGuEAAAAAAAAAAAAAAAAAAAAAPgADAP7/CQAGAAAAAAAAAAAAAAABAAAAAQAAAAAAAAAA
EAAAHAAAAAEAAAD+////AAAAAAAAAAD/////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////
```

(A vnd a vendor rövidítése.)

EXE:

```
------=_NextPart_000_009D_01CAB740.1CA6BDC0
Content-Type: application/x-msdownload;
        name="myuninst.exe"
Content-Transfer-Encoding: base64
Content-Disposition: attachment;
        filename="myuninst.exe"

TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAA+AAAAA4fug4AtAnNIbgBTM0hVGhpcyBwcm9ncmFtIGNhbm5vdCBiZSBydW4gaW4gRE9TIG1v
ZGUuDQ0KJAAAAAAAAAA1L+4CcU6AUXFOgFFxToBRClKMUXROgFHyUo5Rck6AUR5Ri1FzToBRHlGK
```

Nem beszéltünk még a levelek fejlécében található czifra karakterekről.
Nézzük meg például ezt:

```
From: =?iso-8859-2?Q?Petr=E9nyi_J=F3zsef?= <jpetrenyi@gmail.com>
```

Látjuk rajta, hogy hasonlít a quoted-printable kódolásra... de azért nem teljesen az. Itt ugyanis magába az értékbe kell elrejtenünk, hogy melyik ASCII táblát használtuk, és milyen kódolást. A fenti sorban például az iso-8859-2 táblát és a Q, azaz quoted-printable kódolást. Az egyes mezőket a '?' karakter választja el.

(Vannak még megkötések, a space helyett pl. '_' karakter lép be, az értékek hossza sem lehet végtelen, 75 karakteres blokkokba tördelünk - de ennyire mélyen foglalkozzanak vele a kóderek.)

Azaz összefoglalva: a MIME fő feladata, hogy bárhol, ahol a levélben 7 bitesnél bonyolultabb karaktertipus jelenik meg, azt át tudja kódolni 7 bitesre. (Hogy ne kelljen hozzányúlni az SMTP szerverekhez.)
Ehhez két kódolást használ, már amikor egyáltalán szüksége van rá:

- quoted printable
- base64

Emellett megteszi azt a szívességet is, hogy ha egy konkrét bináris kupacot kell a levélbe belerakni, akkor a content-type változóban be is kategorizálja azt.

Ez a kategorizálás meglehetősen rögzített. Imhol a fő kategóriák.

2.12. TÁBLÁZAT

MIME tipus
Application
Audio
Example
Image
Message
Model
Multipart
Text
Video

A fő kategóriákon belül aztán kismillió altipus létezik.

Content types and subtypes:
http://www.iana.org/assignments/media-types/

Ez az egész MIME kategorizálás végül annyira sikeres lett, hogy átvették a HTTP-be is. Hiszen pucérnős képet ne csak emailben lehessen már küldeni, valahogyan a webről való letöltésnek is működnie kell. (Bár itt a 7/8 bit probléma nem játszik.)

http://en.wikipedia.org/wiki/Internet_media_type

Huh. Azt hiszem, a body részt ezzel ki is végeztük.

Nézzük akkor a fejléceket.

Minden különösebb értesítés helyett:

2.13. TÁBLÁZAT

Fejléc neve	Protokoll	Státusz
A-IM	http	
Accept	http	
Accept-Additions	http	
Accept-Charset	http	
Accept-Encoding	http	
Accept-Features	http	
Accept-Language	http	
Accept-Language	mail	
Accept-Patch	http	
Accept-Ranges	http	
Age	http	
Allow	http	
Also-Control	netnews	obsoleted
Alternate-Recipient	mail	
Alternates	http	
Apply-To-Redirect-Ref	http	
Approved	netnews	standard
Archive	netnews	standard
Archived-At	mail	standard
Archived-At	netnews	standard
Article-Names	netnews	obsoleted
Article-Updates	netnews	obsoleted
Authentication-Info	http	
Authentication-Results	mail	standard
Authorization	http	
Auto-Submitted	mail	standard
Autoforwarded	mail	
Autosubmitted	mail	
Bcc	mail	standard
C-Ext	http	
C-Man	http	
C-Opt	http	
C-PEP	http	
C-PEP-Info	http	
Cache-Control	http	
Cc	mail	standard
Comments	mail	standard
Comments	netnews	standard
Connection	http	
Content-Alternative	MIME	
Content-Base	http	
Content-Base	MIME	
Content-Description	MIME	
Content-Disposition	http	
Content-Disposition	MIME	
Content-Duration	MIME	
Content-Encoding	http	
Content-features	MIME	
Content-ID	http	
Content-ID	MIME	
Content-Identifier	mail	
Content-Language	http	
Content-Language	MIME	
Content-Length	http	
Content-Location	http	
Content-Location	MIME	
Content-MD5	http	
Content-MD5	MIME	
Content-Range	http	

Content-Return	mail	
Content-Script-Type	http	
Content-Style-Type	http	
Content-Transfer-Encoding	MIME	
Content-Type	http	
Content-Type	MIME	
Content-Version	http	
Control	netnews	standard
Conversion	mail	
Conversion-With-Loss	mail	
Cookie	http	
Cookie2	http	
DASL	http	standard
DAV	http	
DL-Expansion-History	mail	
Date	http	
Date	mail	standard
Date	netnews	standard
Date-Received	netnews	obsoleted
Default-Style	http	
Deferred-Delivery	mail	
Delivery-Date	mail	
Delta-Base	http	
Depth	http	
Derived-From	http	
Destination	http	
Differential-ID	http	
Digest	http	
Discarded-X400-IPMS-Extensions	mail	
Discarded-X400-MTS-Extensions	mail	
Disclose-Recipients	mail	
Disposition-Notification-Options	mail	
Disposition-Notification-To	mail	
Distribution	netnews	standard
DKIM-Signature	mail	standard
Downgraded-Bcc	mail	experimental
Downgraded-Cc	mail	experimental
Downgraded-Disposition-Notification-To	mail	experimental
Downgraded-From	mail	experimental
Downgraded-Mail-From	mail	experimental
Downgraded-Rcpt-To	mail	experimental
Downgraded-Reply-To	mail	experimental
Downgraded-Resent-Bcc	mail	experimental
Downgraded-Resent-Cc	mail	experimental
Downgraded-Resent-From	mail	experimental
Downgraded-Resent-Reply-To	mail	experimental
Downgraded-Resent-Sender	mail	experimental
Downgraded-Resent-To	mail	experimental
Downgraded-Return-Path	mail	experimental
Downgraded-Sender	mail	experimental
Downgraded-To	mail	experimental
Encoding	mail	
Encrypted	mail	
ETag	http	
Expect	http	
Expires	http	
Expires	mail	
Expires	netnews	standard
Expiry-Date	mail	
Ext	http	
Followup-To	netnews	standard
From	http	
From	mail	standard

From	netnews	standard
Generate-Delivery-Report	mail	
GetProfile	http	
Host	http	
IM	http	
If	http	
If-Match	http	
If-Modified-Since	http	
If-None-Match	http	
If-Range	http	
If-Unmodified-Since	http	
Importance	mail	
In-Reply-To	mail	standard
Incomplete-Copy	mail	
Injection-Date	netnews	standard
Injection-Info	netnews	standard
Keep-Alive	http	
Keywords	mail	standard
Keywords	netnews	standard
Label	http	
Language	mail	
Last-Modified	http	
Latest-Delivery-Time	mail	
Lines	netnews	deprecated
Link	http	
List-Archive	mail	
List-Help	mail	
List-ID	mail	
List-Owner	mail	
List-Post	mail	
List-Subscribe	mail	
List-Unsubscribe	mail	
Location	http	
Lock-Token	http	
Man	http	
Max-Forwards	http	
Message-Context	mail	
Message-ID	mail	standard
Message-ID	netnews	standard
Message-Type	mail	
Meter	http	
MIME-Version	http	
MIME-Version	MIME	
Negotiate	http	
Newsgroups	netnews	standard
NNTP-Posting-Date	netnews	obsoleted
NNTP-Posting-Host	netnews	obsoleted
Obsoletes	mail	
Opt	http	
Ordering-Type	http	standard
Organization	netnews	standard
Original-Encoded-Information-Types	mail	
Original-From	mail	standard
Original-Message-ID	mail	
Original-Recipient	mail	standard
Original-Sender	netnews	standard
Originator-Return-Address	mail	
Original-Subject	mail	standard
Overwrite	http	
P3P	http	
Path	netnews	standard
PEP	http	
PICS-Label	http	

PICS-Label	mail	
Pep-Info	http	
Position	http	standard
Posting-Version	netnews	obsoleted
Pragma	http	
Prevent-NonDelivery-Report	mail	
Priority	mail	
ProfileObject	http	
Protocol	http	
Protocol-Info	http	
Protocol-Query	http	
Protocol-Request	http	
Proxy-Authenticate	http	
Proxy-Authentication-Info	http	
Proxy-Authorization	http	
Proxy-Features	http	
Proxy-Instruction	http	
Public	http	
Range	http	
Received	mail	standard
Received-SPF	mail	
Redirect-Ref	http	
References	mail	standard
References	netnews	standard
Referer	http	
Relay-Version	netnews	obsoleted
Reply-By	mail	
Reply-To	mail	standard
Reply-To	netnews	standard
Resent-Bcc	mail	standard
Resent-Cc	mail	standard
Resent-Date	mail	standard
Resent-From	mail	standard
Resent-Message-ID	mail	standard
Resent-Reply-To	mail	obsoleted
Resent-Sender	mail	standard
Resent-To	mail	standard
Retry-After	http	
Return-Path	mail	standard
Safe	http	
Security-Scheme	http	
See-Also	netnews	obsoleted
Sender	mail	standard
Sender	netnews	standard
Sensitivity	mail	
Server	http	
Set-Cookie	http	
Set-Cookie2	http	
SetProfile	http	
SLUG	http	standard
SoapAction	http	
Solicitation	mail	
Status-URI	http	
Subject	mail	standard
Subject	netnews	standard
Summary	netnews	standard
Supersedes	mail	
Supersedes	netnews	standard
Surrogate-Capability	http	
Surrogate-Control	http	
TCN	http	
TE	http	
Timeout	http	

To	mail	standard
Trailer	http	
Transfer-Encoding	http	
URI	http	
Upgrade	http	
User-Agent	http	
User-Agent	netnews	standard
Variant-Vary	http	
Vary	http	
VBR-Info	mail	standard
Via	http	
WWW-Authenticate	http	
Want-Digest	http	
Warning	http	
X400-Content-Identifier	mail	
X400-Content-Return	mail	
X400-Content-Type	mail	
X400-MTS-Identifier	mail	
X400-Originator	mail	
X400-Received	mail	
X400-Recipients	mail	
X400-Trace	mail	
Xref	netnews	standard

Forrás:
http://www.iana.org/assignments/message-headers/perm-headers.html

Aláírom, ez kissé már sok volt. De ennyi van. Biztos vagyok benne, hogy a legtöbbel nem fogsz találkozni. Én magam is sokat töprengtem, hogy érdemes-e ekkora táblázatokat belerakni a könyvbe, amikor hivatkozni is lehet a neten lévő anyagokra. Végül úgy döntöttem, jobb az, ha egy könyvben ott van minden. Ha kell, akkor nem leszel rákényszerítve, hogy egy csomó böngészőablak legyen nyitva olvasás közben, illetve ha kinyomtatod, akkor elég csak magát a könyvet.
Ha meg nem kell, akkor legfeljebb átlapozod.

Lehet, most fogsz megkövezni, de a következőkben azokról a header mezőkről fogok beszélni, amelyek _*nincsenek*_ benne a fenti táblázatban.
Nem gondoltad volna, mi? Akkora ez a táblázat, hogy ebben elméletileg a teljes világegyetemnek bele kellett volna férnie.

De van egy olyan fejléc kategória, mely nem szabványos ugyan - ezért nincs benne a táblázatban - mégis nagyon elterjedt: ez a borzasztó népes X-header család.

Az X-headereknek ugyanis nincs definíciójuk. X-headert bárki olyat csinál magának, amilyet csak akar.

Nézd meg például az X-Face header tipust.

Itt van néhány képviselője:

X-Face	About...	ASCII-code
	The Cheshire Cat	X-Face: ,l9N8F.A~!GX.Yz#yYAZtvpk7aF(an!BVo3%Xb,`Q]*$Oh'RtXB 5n4iisiGw8l3nY)lnh(G]9[ud;QeX:W}'r)1F9gEQD4o::mBU9J ZAXI\y_0t$P#*/o|!TSG9OtqY+`NX>cpjf?w~w1RY_}M5731El5 2pUnI}OaRi~PA+n[fiO>=TS|2h~f%c9zy]*i2LajTWLJ_X^1No" P#D_(t*t-5n]wW1
	Dilbert	X-Face: Xw|'+}IE3RcO/4u;MCw_rVT6OB<-PgDq{'[qF%;xVX;*,S9g9:- c#LRR*tVhOl'+uZ&Ial.?6]1IjF^vvo:T<D+:[(_Z+Cp};:9Zotv Or<(#4RP7aF[JT?p9MjTCGa3kP'2C;?ILRb+QYx~WO6{Ala1;v9- ^2_.BDdnY/.^8PeD3SKJY=',t2vXpx7m;Z>xr@C*,<ND6W?uokQW ;.QsPuk>WaFwl7'ig[9
	Goat	X-Face: K?^RDm#'Ec,&g%3z!blP\np<Az>B]y~Y!(oHZmR7#Jf4{r(s&5@ dKA*m5G4'6U'p[id6U$z<0+)=LMn^2J+yohN"Pk@7zd;0I8V~AB ynvf5xL|B9r<cFfBxFpWjP+fQTwv5rd'A*+HE%pH(k'F9T~6Srp U7Ifl+;eGh6|}

Jó nagy adagot találhatsz itt:
http://www.xs4all.nl/~ace/X-Faces/

Oké, ez elég extrém baromságnak tűnik... de mi lehet az értelme?

Például én belerakom minden levelembe a következő fejlécet:
X-Face: *<valamelyik fenti X-Face kód>*

Mit csinálnak vele a levelezőszerverek? Semmit. X-header, azaz nem szabványos, azaz nem kötelesek ismerni. Fogalmuk sincs, mit takar a bájtsorozat. Ugyanígy fognak eljárni velük a levelező kliensek.

De mi van akkor, ha valaki másnak is megtetszik az ötlet? Mondjuk belefejlesztik egy MUA kliensbe, hogy ismerje fel ezt a fejléctipust, kódolja vissza a kódot - és a kialakult ábrát illessze bele mondjuk a levél végére, aláírásként. Innentől akik ilyen levelezőklienst használnak, meg lesznek győződve, hogy mekkora cool übergeek-ek. Hülyeség? Lehet. De a mobiltelefonokra gyártott háttérkép is hülyeségnek tűnt, aztán egy időben mégis mindenkinek olyan figyelt a kijelzőjén.

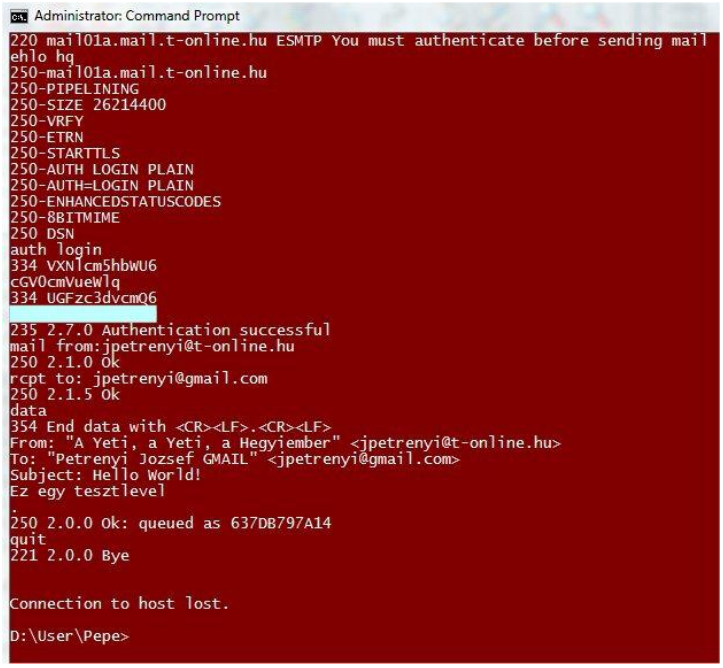
Természetesen az X-header-eknek létezik értelmes felhasználása is. Ezekből szemezgetek néhányat.

2.14. TÁBLÁZAT

X-header	Jelentése
X-Sender	Az Eudora kliens rakja össze, a login névből és a levelezőszerver nevéből
X-Mail From	A Fastmail kliens rakja bele a levélbe, nagyjából senki nem tudja, mi célból
X-Mailer	A levelezőkliens neve, amelyik összerakta a levelet. Széleskörűen elterjedt.
X-envelope-from	Bizonyos levelezőkliensek bemásolják a levélbe is a boríték adatait. (mail from) Így az esetleges külső sérülés esetén is célba tud érni a levél.
X-envelope-to	Lásd, mint fent. (rcpt to)
X-SCL	Spam Confidence Level
X-PCL	Phishing Confidence Level
X-MS-Exchange-Organization-SCL	Lásd fent, csak az MS levelezőkliensekben.
X-MS-Exchange-Organization-PCL	Lásd fent, csak az MS levelezőkliensekben.
X-MS-Has-Attach	Van-e csatolás a levélben
X-MS-Exchange-Organization-PRD	Domain név
X-MS-Exchange-Organization-SenderIDResult	A sender ID azonosítása sikerült-e, vagy sem

2.3.3 LEVELEZZÜNK MÁR VÉGRE!

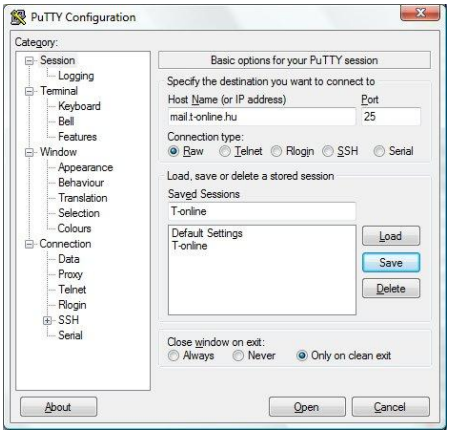
Habár már láttunk levélküldési próbálkozást, de akkor inkább csak a hatásukat vizsgáltuk. Most viszont végig fogunk menni egy levél összerakásán, lépésről lépésre.

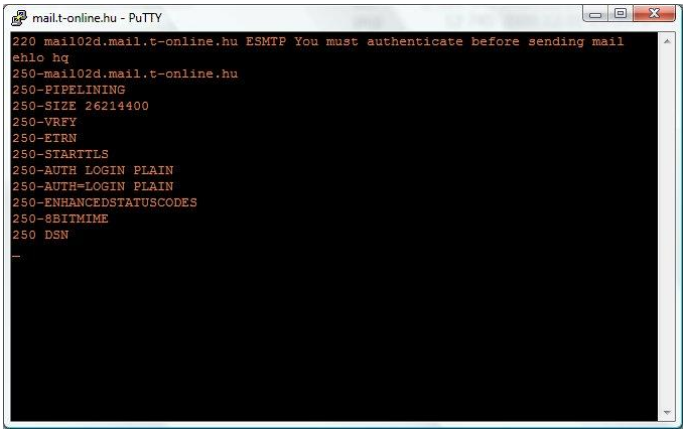


2.34. ÁBRA LEVÉL KÜLDÉSE PARANCSSORBÓL

Először is kezdjük azzal, hogy bár a command prompt elég okos jószág, de smtp szerver funkció azért még nincs beleépítve. Ahhoz, hogy össze tudjunk rakni és el tudjunk küldeni egy levelet, előzetesen nyitnunk kell egy SMTP sessiont egy létező SMTP szerver felé. Erre a célra a beépített telnet kliens programot fogjuk használni.

Igényesebb olvasóknak ajánlom a svájcibicskát, a putty programot.
http://www.chiark.greenend.org.uk/~sgtatham/putty/





Természetesen olyan SMTP szervert kell keresnünk, mely engedi is a session megnyitását. Nekem a T-Online az internetszolgáltatóm, logikusan az ő SMTP szerverüket fogom használni.

```
telnet mail.t-online.hu 25
```

Az SMTP a 25-ös TCP porton dolgozik, értelemszerűen erre a portra kell nyitnunk is. A munkamenet kialakítása után köszönünk, mert udvarias fiúk vagyunk.

```
ehlo hq
```

Mondhattuk volna azt is, hogy helo hq - de ekkor nem kapjuk vissza, milyen ESMTP parancsokat ismer a szerver. A hq jelen esetben a kliens gép neve - és a konkrét szituációban teljesen indifferens adat. Bármit is írhattam volna ide.
De ez nem jelenti azt, hogy éles környezetben is ilyen lazán bánhatunk a paranccsal. Léteznek olyan paranoid SMTP szerverek, melyek leellenőrzik, hogy az itt megadott node-nak tényleg az-e az IP címe, mint amelyikről éppen nyomulunk - és ha nem, akkor eldobják a sessiont.

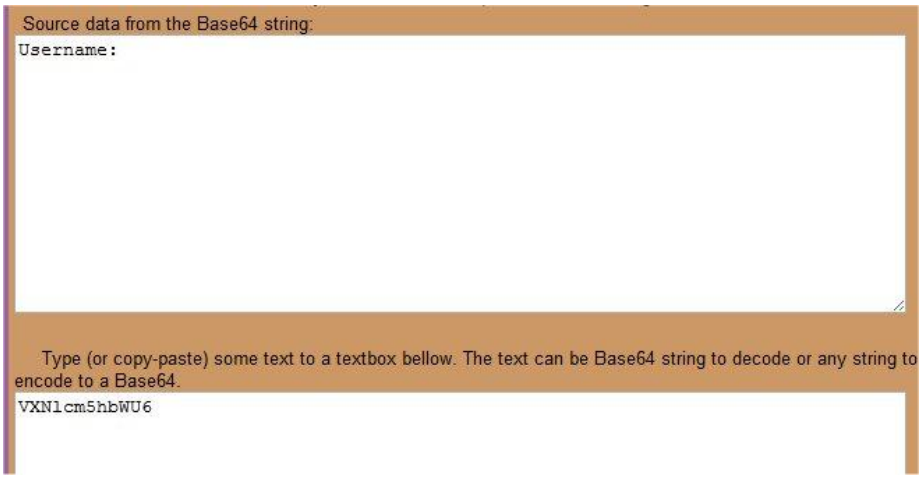
Oké, túlvagyunk a beköszönésen. Vegyük észre, hogy a szerver nem csak azt mondja meg, milyen parancsokat ismer, de arra is figyelmeztet, hogy autentikáció nélkül semmire sem megyünk.
Kezdjük el.

```
auth login
```

Erre a szerver káromkodik valamit. Majd szemtelenül villog a prompt, jelezve, hogy erre válaszoljál valamit, köcsög. Még ha feltételezzük is, hogy ez egy kérdés lenne, akkor is milyen nyelven van? Ezt mindenképpen tisztázni kell ahhoz, hogy válaszolni tudjunk.

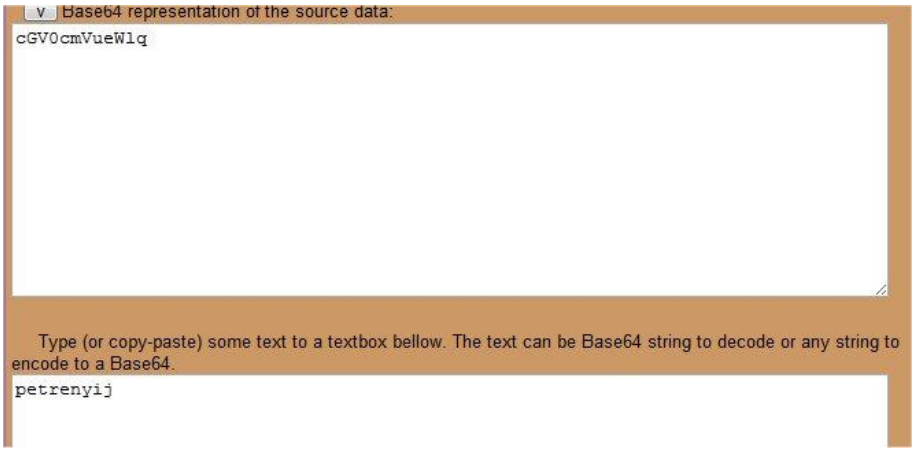
Azért annyira nem veszélyes a helyzet. Mint ahogy titkosírásnál is az az első, mondhatni ösztönös reakciónk, hogy megpróbáljuk visszafelé olvasni a szavakat, ugyanez igaz az informatikai titkosításokra is. Megnézzük, hátha Base64.
Ennek a legegyszerűbb módja, ha keresünk a weben egy Base64 encoder/decoder oldalt.

Például valamelyiket:
http://www.motobit.com/util/base64-decoder-encoder.asp
http://www.opinionatedgeek.com/dotnet/tools/Base64Decode/Default.aspx



2.35. ÁBRA BASE64 DECODE

Így legyen lottó ötösünk. A szerver azt kérdezi tőlünk bézhatvannégyül, hogy mi a felhasználói nevünk. Mondjuk meg neki.



2.36. ÁBRA BASE64 ENCODE

Látható (*2.34. ábra Levél küldése parancssorból*), ezt a karaktersorozatot nyomtam vissza neki. Jött a következő kérdés. Gondolom, nem huppansz le a földre, ha

elárulom, hogy ekkor a jelszavamat kérdezte, melyet hasonló kódolás után adtam meg neki. Az eredmény látszik:

```
235 2.7.0 Authentication successful
```

> RFC 2554

Ha nem megérzés alapján dolgozunk, hanem a tudományosabb módszereket preferáljuk, akkor akár el is olvashattuk volna az SMTP autentikációra vonatkozó RFC-t. Ott szépen le van írva, hogy a kódolás Base64.

Bent vagyunk végre, kezdhetünk dolgozni.

```
mail from:jpetrenyi@t-online.hu
rcpt to: jpetrenyi@gmail.com
```

Ráírjuk a borítékra a feladót és a címzettet. Ránézésre nem egy bonyolult dolog, de nagyon sokszor hasalunk el ezen a lépésen. Nem, nem azért, mert bénák vagyunk és nem tudunk gépelni - hanem azért, mert ekkor szokott aktiválódni a szerverek open relay elleni védelme.

> Relay : Továbbpasszolás.
> Open relay : Mindenkitől elfogad levelet és továbbpasszolja.
> Close relay : Csak egy kiválasztott körtől fogad el levelet.
>
> Jegyezzük meg: Open Relay fúj. A szpemmelés melegágya.

Nálam mindkét parancsra az a válasz érkezett, hogy

```
250 2.1.5 Ok
```

Ez azt jelenti, hogy mivel az IP címem a T-online tartományába esik, számomra engedélyezve van a relé.

Csináljunk egy ellentesztet.

2.37. ÁBRA CLOSE RELAY

Beléptem egy tetszőleges SMTP szerverre. Megvolt a köszöngetés, itt nem kértek jelszót. Látható, hogy a mail from parancson simán átjutottam. Ilyenkor a szerver külső embernek tart - akinek csak belső címre lenne szabad levelet küldenie. Mivel én az rcpt to parancsban is külső címet adtam meg, így ez már relének számít - ami az én IP címemről szemmel láthatóan nincs engedélyezve.

Jöhet magának a levélnek az összerakása.

```
data
```

A szerver figyelmeztet, hogy majd a '.' karakterrel kell lezárnom.

```
354 End data with <CR><LF>.<CR><LF>
```

Fejlécekkel kezdünk.

```
From: "A Yeti, a Yeti, a Hegyiember" <jpetrenyi@t-online.hu>
To: "Petrenyi Jozsef GMAIL" <jpetrenyi@gmail.com>
Subject: Hello World!
```

Most szépen kitöltöttem minden fejlécelemet, úgy, ahogy elő van írva. Vegyük észre, hogy a borítékon lévő címekhez képest egy kicsit trükköztem a belső címekkel.

A fejlécek után a body jön.

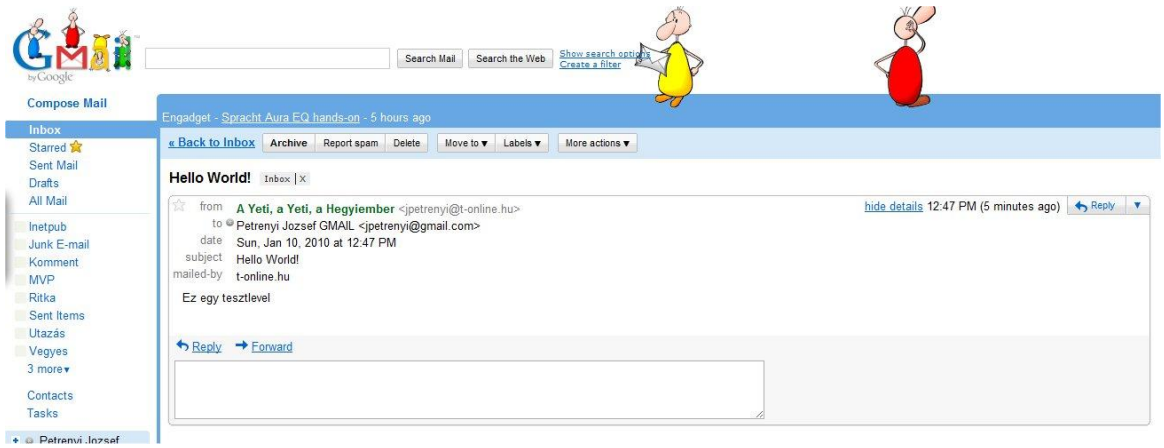
```
Ez egy tesztlevel
.
```

Nos, igen. Nem bonyolítottam túl az életet mindenféle MIME tagolással. Oldschool módra csak 7 bites karaktereket használtam, így nem is volt rá szükségem.
Vegyük észre a záró pöttyöt.

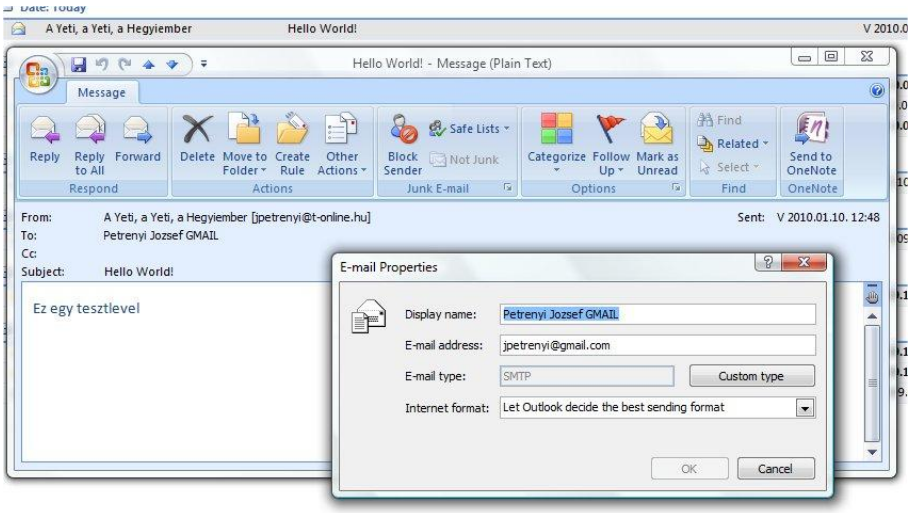
```
250 2.0.0 Ok: queued as 637DB797A14
quit
221 2.0.0 Bye
```

A szerver közölte, hogy átvette a levelemet és berakta a várakozási sorába. Egyszer majd el is küldi. Mindenesetre az én dolgom itt véget ért, kiléptem.

Nézzük meg, mit csináltunk:



2.38. ÁBRA TESZTLEVÉL GMAIL-BEN



2.39. ÁBRA TESZTLEVÉL OUTLOOKBAN

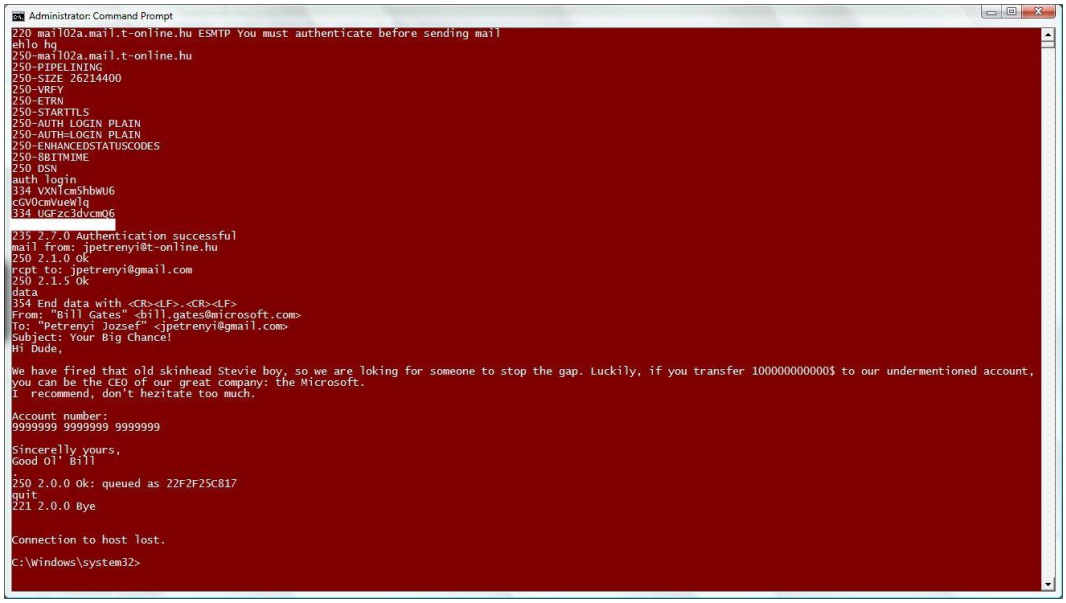
Abszolút normális levélnek tűnik.

Mit okozott az apró trükközésünk:

- A To mezőben eltakartuk a címzett SMTP címét, így csak az általunk kitalált név látszik. A címet előcsalogatni csak a névre kattintással lehet.
- A From mezőben azért még a rendes cím látszik. Még.

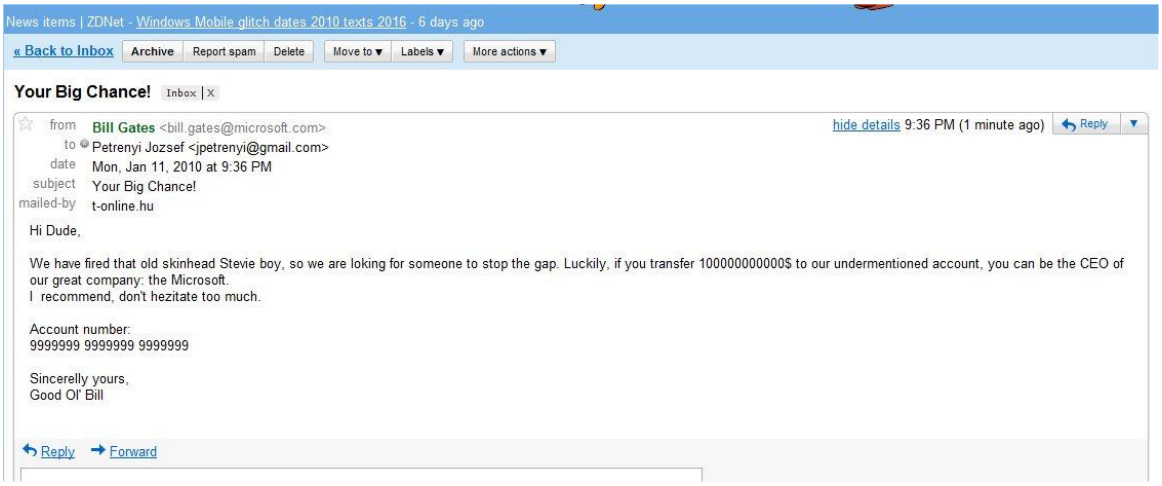
Eddig jó fiúk voltunk. A következőkben nézzük meg, milyen kevéssel kell több ahhoz, hogy rossz fiúk lehessünk.

Írunk még egy levelet.



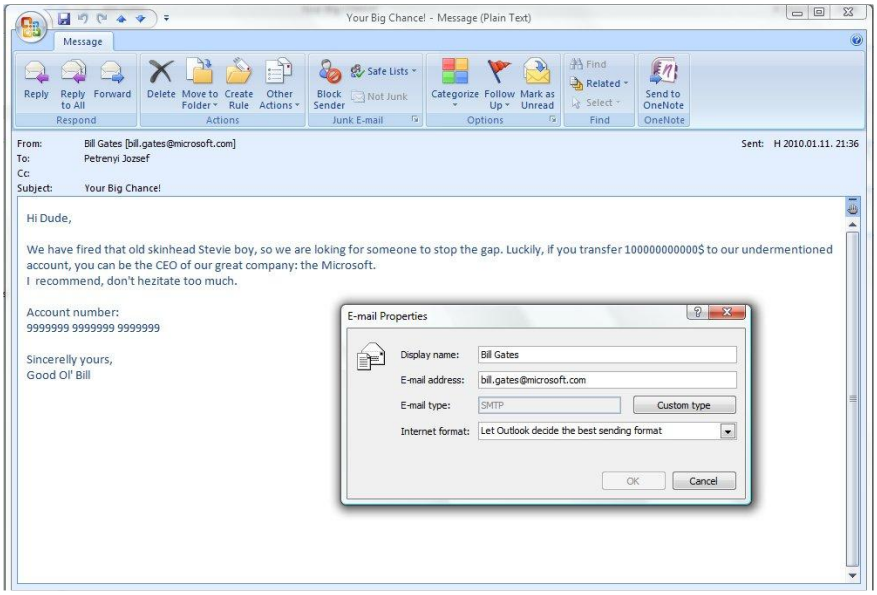
2.40. ÁBRA IRÁNY A SÖTÉT OLDAL

Nézzük meg, mi is történt. Ahogy eddig, most is összeraktunk egy levelet. Csakhogy most - ahhoz képest, mint amit a borítékra írtunk - gyökeresen eltérő feladót adtunk meg a borítékon belül.
Mit fognak erre lépni a levelező kliensek?



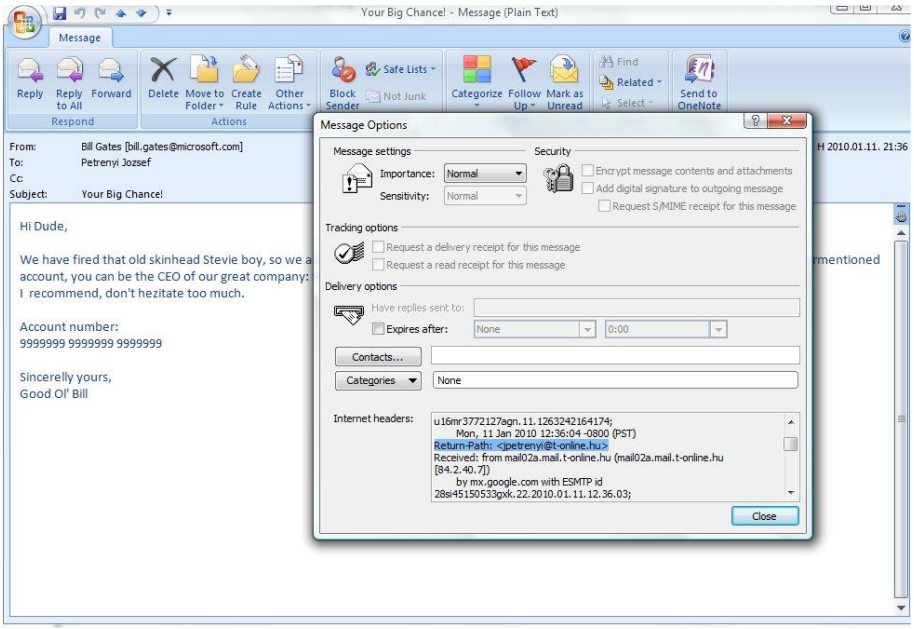
2.41. ÁBRA KAMU LEVÉL A GMAIL-BEN

Rossz hírem van. Simán beszopják.
Nem azt a feladót mutatják meg, aki tényleg küldte a levelet - hanem azt, akinek a nevét a levélbe írtuk. Pedig amit látsz, az már az ún. details nézet. Oké, egy halvány célzás azért van arra, hogy valami nem klappol, az a 'mailed-by t-online.hu' gyanús lehet - de ha itt egy addig sose látott levelezőszerver neve szerepel, egy átlagos felhasználó mennyire fogna gyanút?



2.42. ÁBRA KAMU LEVÉL OUTLOOKBÓL

Outlookban még rosszabb a helyzet. Ha itt ráncolom a szemöldökömet és kérem le a details nézetet, semmi nem jelzi, hogy átverés történt.



2.43. ÁBRA KAMU LEVÉL FEJLÉCE

Egyedül akkor tudom lebuktatni a csalót, ha megnézem a levél fejlécét, ott nem riadok vissza a borzasztóan teki krikszkrakszoktól és megkeresem a Return-Path fejléc mezőt. Itt látszik az igazi feladó: jpetrenyi@t-online.hu.

Ezen a borzasztóan egyszerű trükkön alapul a legtöbb megtévesztő levél. Így lehet elérni, hogy felháborodott leveleket kapjál olyan emailekért, melyeket nem is te küldtél.

> Egyszer kaptam méltatlankodó levelet egy általam bálványként tisztelt magyar karikaturistától, hogy azonnal álljak le a mindenféle hülye emailek küldözgetéséről. Alig bírtam tisztázni magamat. Pedig csak annyi történt, hogy volt valaki, akinek a címlistájában mindkettőnk neve szerepelt, a gépe pedig megfertőződött egy olyan féreggel, mely a címlistából vett címeket véletlenszerűen párosítva küldözgetett kamu leveleket.

És ez csak a tesztelési fázisa volt egy technikának, melyet ma phishing-nek nevezünk: hitelesnek látszó emailekkel vesznek rá gyanútlan embereket arra, hogy pénzt utaljanak át, jelszavakat adjanak meg.

2.4 PORTTÁBLÁZAT

A portok... ezek meglehetősen furcsa lények. Egy részüket RFC írja elő, más részüket szokásjog alakította ki. Nem véletlen, hogy a foglalt portokat well-known portoknak, azaz jól ismert portoknak is nevezik.

Ráadásul hiába írja elő RFC az egyes protokollok által használt portot - ha semmi nem szankcionálja azokat, akik eltérnek ettől. Nem visz el a spanyol inkvizíció, ha a webes alkalmazásomat nem a 80-as porton publikálom ki. Jó, persze, nehezebb lesz megjegyezni a címét, mert oda kell tenni mögé a port számot: http://furamuki.hu:53. Az is igaz, hogy ezzel kizártam mindenkit, aki tűzfal mögül jött volna. Nem valószínű, hogy az én kis webshopom kedvéért ütnek egy plusz lyukat a céges tűzfalakba. Az anonymous proxikat meg nem mindenki ismeri. De még ha ki is lyukasztják, akkor sem biztos, hogy el fognak tudni érni. Én éppen nemrég futottam össze egy olyan vicces webmesterrel, aki a 8443-as porton adta a HTTPS-t. Az ISA2006 ettől egyből dobott is egy hátast, mert a HTTPS proxy modulja (webproxy) ezt a trükköt nem bírta lekezelni. Arról meg aztán nem is beszélve, hogy az 53-as port a DNS szolgáltatáshoz tartozik, tehát ha oda rakom a HTTP-t, akkor ezen a gépen nem futhat majd DNS szerver alkalmazás.

Szóval ezeket mind végig kell gondolni, és ezen szempontok alapján kell dönteni arról, hogy használjuk-e az ajánlásokat, vagy sem.
Ha a józan ész dominál, akkor igen.

2.15. TÁBLÁZAT

Port No.	Protocol	Service Name	Aliases	Comment
7	TCP	echo		Echo
7	UDP	echo		Echo
9	TCP	discard	sink null	Discard
9	UDP	discard	sink null	Discard
13	TCP	daytime		Daytime
13	UDP	daytime		Daytime
17	TCP	qotd	quote	Quote of the day
17	UDP	qotd	quote	Quote of the day
19	TCP	chargen	ttytst source	Character generator
19	UDP	chargen	ttytst source	Character generator
20	TCP	ftp-data		File Transfer
21	TCP	ftp		FTP Control
23	TCP	telnet		Telnet
25	TCP	smtp	mail	Simple Mail Transfer
37	TCP	time		Time
37	UDP	time		Time
39	UDP	rlp	resource	Resource Location Protocol
42	TCP	nameserver	name	Host Name Server
42	UDP	nameserver	name	Host Name Server
43	TCP	nicname	whois	Who Is
53	TCP	domain		Domain Name
53	UDP	domain		Domain Name Server
67	UDP	bootps	dhcps	Bootstrap Protocol Server (DHCP server)

68	UDP	bootpc	dhcpc	Bootstrap Protocol Client (DHCP client)
69	UDP	tftp		Trivial File Transfer
70	TCP	gopher		Gopher
79	TCP	finger		Finger
80	TCP	http	www, http	World Wide Web
88	TCP	kerberos	krb5	Kerberos
88	UDP	kerberos	krb5	Kerberos
101	TCP	hostname	hostnames	NIC Host Name Server
102	TCP	iso-tsap		ISO-TSAP Class 0
107	TCP	rtelnet		Remote Telnet Service
109	TCP	pop2	postoffice	Post Office Protocol - Version 2
110	TCP	pop3	postoffice	Post Office Protocol - Version 3
111	TCP	sunrpc	rpcbind portmap	SUN Remote Procedure Call
111	UDP	sunrpc	rpcbind portmap	SUN Remote Procedure Call
113	TCP	auth	ident tap	Authentication Sevice
117	TCP	uucp-path		UUCP Path Service
119	TCP	nntp	usenet	Network News Transfer Protocol
123	UDP	ntp		Network Time Protocol
135	TCP	epmap	loc-srv	DCE endpoint resolution
135	UDP	epmap	loc-srv	DCE endpoint resolution
137	TCP	netbios-ns	nbname	NETBIOS Name Service
137	UDP	netbios-ns	nbname	NETBIOS Name Service
138	UDP	netbios-dgm	nbdatagram	NETBIOS Datagram Service
139	TCP	netbios-ssn	nbsession	NETBIOS Session Service
143	TCP	imap	imap4	Internet Message Access Protocol
158	TCP	pcmail-srv	repository	PC Mail Server
161	UDP	snmp	snmp	SNMP
162	UDP	snmptrap	snmp-trap	SNMP TRAP
170	TCP	print-srv		Network PostScript
179	TCP	bgp		Border Gateway Protocol
194	TCP	irc		Internet Relay Chat Protocol
213	UDP	ipx		IPX over IP
389	TCP	ldap		Lightweight Directory Access Protocol
443	TCP	https	MCom	
443	UDP	https	MCom	
445	TCP			Microsoft CIFS
445	UDP			Microsoft CIFS
464	TCP	kpasswd		Kerberos (v5)
464	UDP	kpasswd		Kerberos (v5)
500	UDP	isakmp	ike	Internet Key Exchange (IPSec)
512	TCP	exec		Remote Process Execution
512	UDP	biff	comsat	Notifies users of new mail
513	TCP	login		Remote Login
513	UDP	who	whod	Database of who's logged on, average load
514	TCP	cmd	shell	Automatic Authentication
514	UDP	syslog		
515	TCP	printer	spooler	Listens for incoming connections
517	UDP	talk		Establishes TCP Connection
518	UDP	ntalk		
520	TCP	efs		Extended File Name Server
520	UDP	router	router routed	RIPv.1, RIPv.2
522				Netmeeting User Location Service
525	UDP	timed	timeserver	Timeserver
526	TCP	tempo	newdate	Newdate
530	TCP	courier	rpc	RPC
530	UDP	courier	rpc	RPC
531	TCP	conference	chat	IRC Chat
532	TCP	netnews	readnews	Readnews
533	UDP	netwall		For emergency broadcasts
540	TCP	uucp	uucpd	Uucpd
543	TCP	klogin		Kerberos login
544	TCP	kshell	krcmd	Kerberos remote shell
548	TCP			Macintosh, File services (AFP/IP)

550	UDP	new-rwho	new-who	New-who
556	TCP	remotefs	rfs rfs_server	Rfs Server
560	UDP	rmonitor	rmonitord	Rmonitor
561	UDP	monitor		
563	TCP			NNTP-TLS
636	TCP	ldaps	sldap	LDAP over TLS/SSL
749	TCP	kerberos-adm		Kerberos administration
749	UDP	kerberos-adm		Kerberos administration
993	TCP			IMAP (SSL)
995	TCP			POP3 (SSL)
1109	TCP	kpop		Kerberos POP
1167	UDP	phone		Conference calling
1433	TCP	ms-sql-s		Microsoft-SQL-Server
1433	UDP	ms-sql-s		Microsoft-SQL-Server
1434	TCP	ms-sql-m		Microsoft-SQL-Monitor
1434	UDP	ms-sql-m		Microsoft-SQL-Monitor
1503	TCP			Netmeeting T.120
1512	TCP	wins		Microsoft Windows Internet Name Service
1512	UDP	wins		Microsoft Windows Internet Name Service
1524	TCP	ingreslock	ingres	Ingres
1701	UDP	l2tp		Layer Two Tunneling Protocol
1720	TCP			Netmeeting Audio Call Control
1723	TCP	pptp		Point-to-point tunneling protocol
1731	TCP			Netmeeting Audio Call Control
1801	UDP			Microsoft Message Queue Server
1812	UDP	radiusauth		RRAS (RADIUS authentication protocol)
1813	UDP	radacct		RRAS (RADIUS accounting protocol)
2049	UDP	nfsd	nfs	Sun NFS server
2053	TCP	knetd		Kerberos de-multiplexer
2101	TCP			Microsoft Message Queue Server
2103	TCP			Microsoft Message Queue Server
2105	TCP			Microsoft Message Queue Server
2504	UDP	nlbs		Network Load Balancing
3268	TCP			GC LDAP
3269	TCP			GC LDAP SSL/TLS
3389	TCP			Terminal Server
3527	UDP			Microsoft Message Queue Server
5060	TCP			Session Initiation Protocol (SIP nonencrypted)
5060	UDP			Session Initiation Protocol (SIP nonencrypted)
5061	TCP			Session Initiation Protocol (SIP TLS)
5061	UDP			Session Initiation Protocol (SIP TLS)
6665	TCP			Microsoft Chat Server to Server
6667	TCP			Microsoft Chat Client to Server
8000	TCP			Cybercash Administration
8001	TCP			Cybercash Coin Gateway
8002	TCP			Cybercash Credit Gateway
9535	TCP	man		Remote Man Server

Természetesen a táblázat bőven nem teljes. Mint mindenhol, itt is igaz az, hogy a népszerűbb, mondhatni celeb portok nagyobb nyilvánosságot kapnak, jobban figyelembe veszik az emberek.

Ha például a saját alkalmazásomban a 2002-es portot használom, aztán az első sniffelésnél azt írja ki a Wireshark, hogy ez a globe protokoll portja... akkor mennyire kell a kardomba dőlnöm? Mekkora az esélye annak, hogy használom is valaha ezt a protokollt, ha még csak nem is hallottam róla? Egyáltalán, barátságos protokollról van-e szó, vagy ellenségesről? Jelen esetben az utóbbiról, ugyanis a portot egy

TransScout nevű virnyák használta. Nyilván nem RFC alapján foglalta le... de a sniffer program már jólt ismertnek tekintette, hogy ez a port ehhez a vírushoz tartozik. Ergo egy túlbuzgó antivírus program simán elkaszálhatja a nagyreményű alkalmazásomat.

Van egy nagyon durva határ. Azt szokták mondani, hogy 1024 alatt vannak a nagyon fontos portok. Innen akkor sem mazsolázunk saját portot a magunk számára, ha éppen szabad az érték. Bármikor jöhet egy őrült egy új protokollal és kiszórják neki az addig szabad címet.
1024- 65535 között vannak az úgynevezett kliens portok. (Azért ennyi a plafon, mert két bájtnyi helyen lehet tárolni a portszámokat a TCP/UDP fejlécekben.) Itt már jóval inkább szabad a vásár, bátrabban lehet rabolni. Itt is belefuthatunk persze szerencsétlen véletlenekbe (3268: GC LDAP, 3389: RDP, hogy csak két nagyon fontos portot említsek) - de itt azért már jóval kisebb eséllyel.

2.5 Sztorik

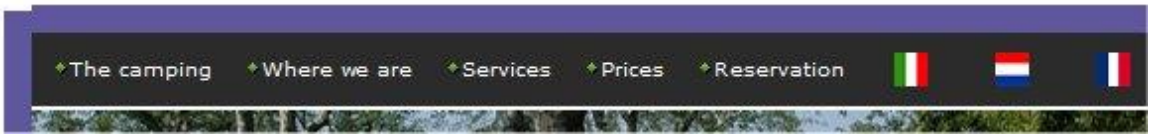
Pihenjünk egy kicsit. Elmesélek két sztorit.
Mindkettő most, frissen, írás közben esett meg velem. Mindkettő kapcsolódik valamilyen módon az eddig írtakhoz. Mintha csak megrendelésre jöttek volna.

2.5.1 Geek úr nyaral

Nem tudom, te hogyan vagy vele, én nagyon aggódós utazásszervező vagyok. Akkor nyugszom meg, ha előre minden le van foglalva, ki van fizetve. Igaz, ekkor meg azon szoktam parázni, hogy időben odaérünk-e mindenhová.

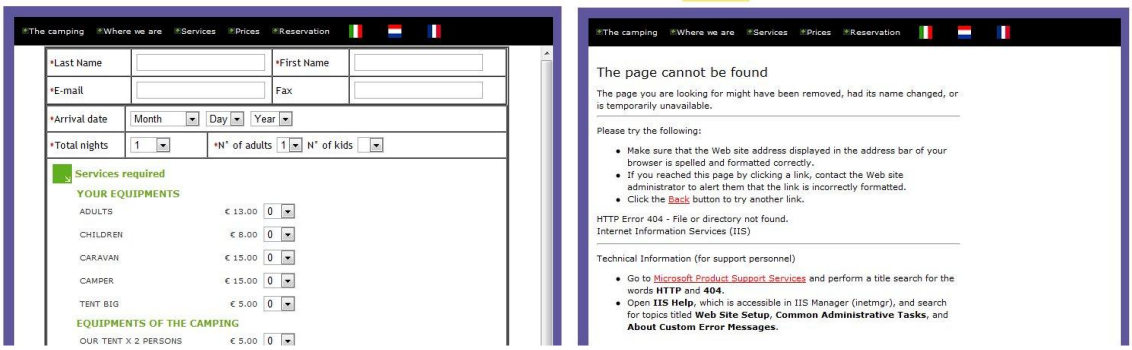
Tudtam, hogy hová akarunk menni a nyáron. Igaz, még január volt, de mivel konkrét eseményre terveztünk, az időpont biztos volt. Maga az esemény miatt jókora tömeg várható, a kiválasztott kemping pedig picike. Ráadásul éppen erős is a forint - ergo minden jel afelé mutat, hogy már most foglaljam le a szállást.

Elmentem a weblapra, megkerestem a Reservation menüpontot.



2.44. ábra Reservation

Erre feljött egy iframe-be ágyazott webes form. Kitöltöttem az adatokat, rányomtam a 'send' gombra. Kaptam egy kövér 404-est.

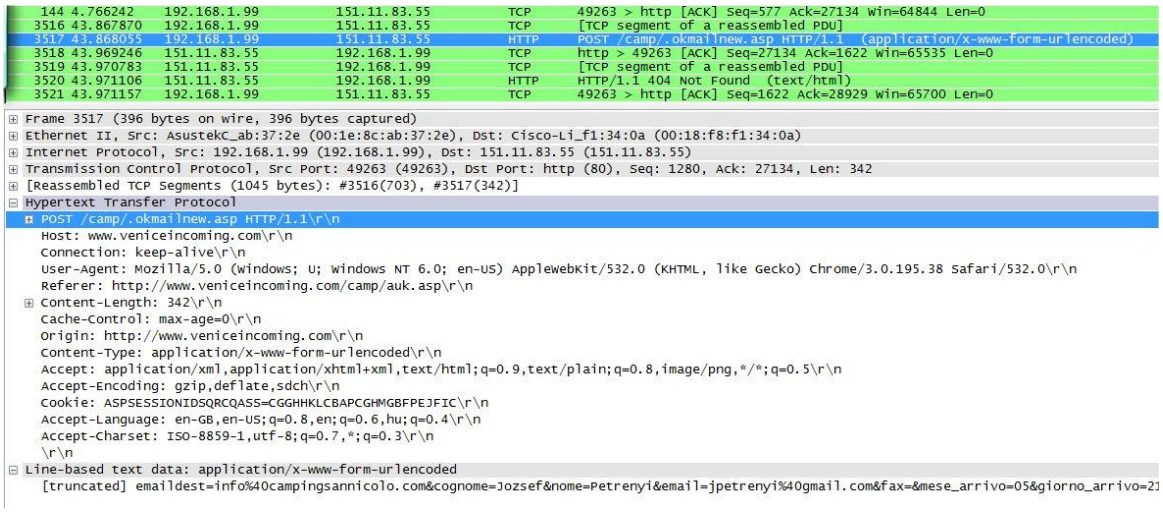


2.45. ábra Adatbevitel

Ez bizony baj. Lemenjek úgy, hogy nincs biztos szállás? Egy ilyen frekventált időpontban?

Akkor már inkább varázsoljunk.

Első lépés: keressük meg, melyik az az oldal, amely végül nem érhető el. Szerencsére a Wireshark mostanában gyakorlatilag bootoláskor indul, így nem okozott gondot a beröffentése. Eljátszottam ugyanezt a foglalást.



2.46. ÁBRA SIKERTELEN POSZTOLÁS

A webes formok adatait az OK gomb megnyomására egy POST paranccsal szokták elküldeni a webes alkalmazás számára. Rakjuk össze a HOST headert és a POST paraméterét: www.veniceincoming/camp/.okmailnew.asp. Ennek az alkalmazásnak kellene átadni a kép alján található szép nagy emaildest változó értékét.
Csakhogy az alkalmazás sztrájkol. Vagy kiment pisilni. Nincs.

Ami elsőre gyanús, az a '.' karakter az URI-ban. Nem szoktak. Írjuk be anélkül az egészet a böngésző címsorába: és rögtön egy alkalmazásoldali hibaüzenetet kapunk.
Első lépésnek jó. Megvan az alkalmazás, a hiba meg jogos, hiszen tényleg nem adtam át semmilyen paramétert.

Nosza.

Alapvetően két stratégia közül választhatunk. Kezdjük az 'ökköllel szöget falbaverős' technikával.

Megkértem a Wireshark-ot, hogy az elkapott csomagokból rekonstruálja a teljes forgalmat.

```
GET /camp/auk.asp HTTP/1.1
Host: www.veniceincoming.com
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US) AppleWebKit/532.0 (KHTML, like
Gecko) Chrome/3.0.195.38 Safari/532.0
Referer: http://www.campingsannicolo.com/uk/contattaci-italiano.htm
Accept:
application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Encoding: gzip,deflate,sdch
Cookie: ASPSESSIONIDSQRCQASS=CGGHHKLCBAPCGHMGBFPEJFIC
Accept-Language: en-GB,en-US;q=0.8,en;q=0.6,hu;q=0.4
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
```

Ez volt a kemping weblapjának lekérése.

```
HTTP/1.1 200 OK
Date: Tue, 19 Jan 2010 18:19:33 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
Content-Length: 26933
Content-Type: text/html; Charset=windows-1252
Cache-control: private


<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
```

Innen jött egy bazi nagy HTML lap. Miért is? Ugye elment egy GET kérés, arra jött egy 200 OK válasz. Nemrég tanultuk, hogy ennek a válasznak a message blokkjában jön vissza a lekért oldal. Mi választja el a message és a header blokkokat? Üres sor. És tényleg.

```
        </body>

</html>

POST /camp/.okmailnew.asp HTTP/1.1
Host: www.veniceincoming.com
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US) AppleWebKit/532.0 (KHTML, like
Gecko) Chrome/3.0.195.38 Safari/532.0
Referer: http://www.veniceincoming.com/camp/auk.asp
Content-Length: 342
Cache-Control: max-age=0
Origin: http://www.veniceincoming.com
Content-Type: application/x-www-form-urlencoded
Accept:
application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Encoding: gzip,deflate,sdch
Cookie: ASPSESSIONIDSQRCQASS=CGGHHKLCBAPCGHMGBFPEJFIC
Accept-Language: en-GB,en-US;q=0.8,en;q=0.6,hu;q=0.4
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3

emaildest=info%40campingsannicolo.com&cognome=Jozsef&nome=Petrenyi&email=jpetrenyi%40gmail
.com&fax=&mese_arrivo=05&giorno_arrivo=21&anno_arrivo=2010&totale_notti=3&numero_adulti=4&
```

```
numero_bambini=&V-TADU=0&V-B2%2F10=0&V-R%2BAUTO=0&V-CAMP=0&V-TG=0&N-TX2=0&N-TX4=0&N-
R4%2F5=1&N-SP=0&N-LAV=0&N-BICI=0&N-ELE=1&N-PARK=0&N-PARK-M=0&Note=&Submit=Send
```

Itt történik a lényeg. Először vége van a nagy HTML oldalnak. (Nyilván közben kitöltöttem a mezőket és rányomtam a Send gombra.) A POST paranccsal indul az adatok feltöltése. Egy csomó fejléc mező után jön az üres sor, majd a message blokkban ott figyel az a karakterlánc, melyet a form rakott össze és melyet az alkalmazásnak már jó étvággyal el kellene fogyasztania.

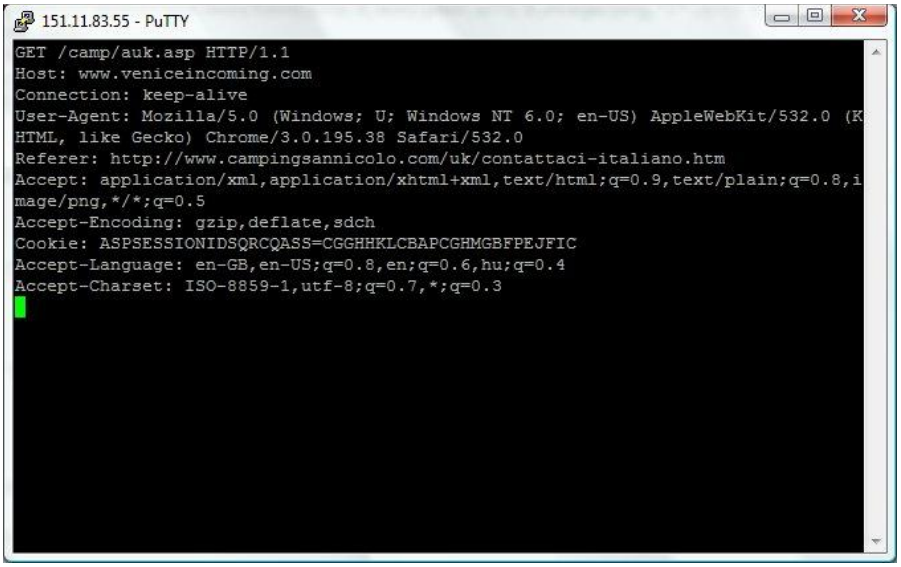
```
HTTP/1.1 404 Not Found
Content-Length: 1635
Content-Type: text/html
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
Date: Tue, 19 Jan 2010 18:20:12 GMT

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<HTML><HEAD><TITLE>The page cannot be found</TITLE>
```

De nem teszi meg, ehelyett visszajön egy 404-es hibakód. (Emlékszünk, kliens oldali hiba.) Azaz nincs olyan weblap - jelen esetben alkalmazás, melyet meg akartunk hívni.
Az üzenet message blokkjában még ott van egy formázott HTML üzenet is, hogy a nem kockafejűek is értsék, miről is van szó.

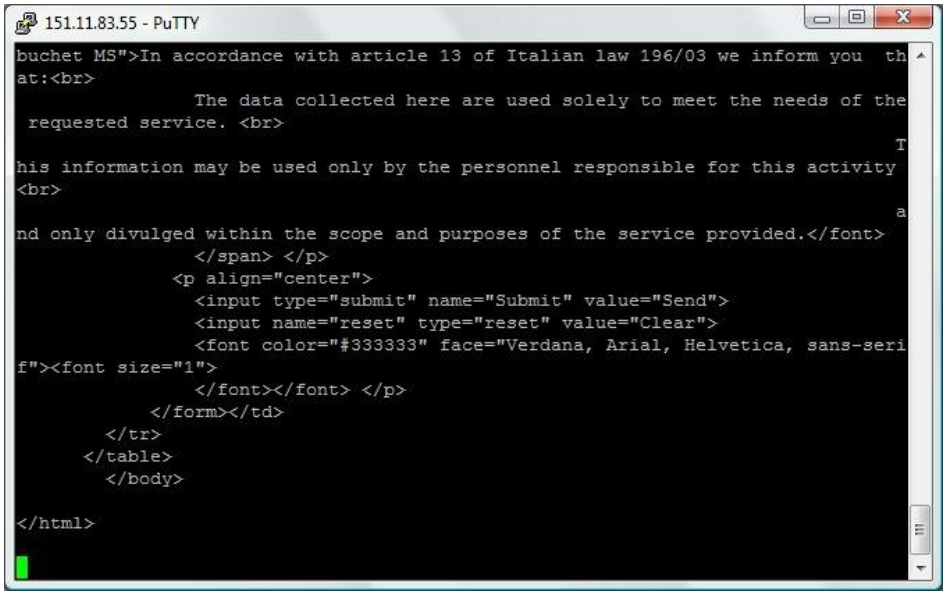
Nos, itt van előttünk a teljes folyamat. Minden parancsot, minden fejlécet, minden message blokkot ismerünk. Azt is tudjuk, hogy a POST parancsban van elgépelve az alkalmazás neve.

Mire várunk? Putty.



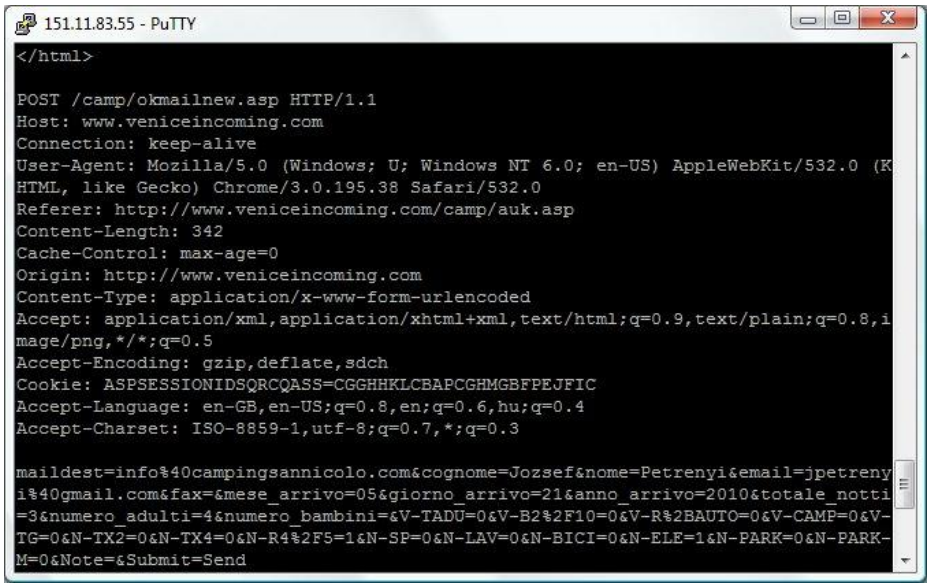
2.47. ÁBRA WEBLAP MANUÁLISAN I

Rácsatlakoztam a webszerver 80-as portjára. Soronként átmásoltam a parancsokat a Wireshark kimenetéből. Ahogy egy üres enter-rel jeleztem, hogy vége a parancsnak, rögtön meg is kaptam a weblapot.



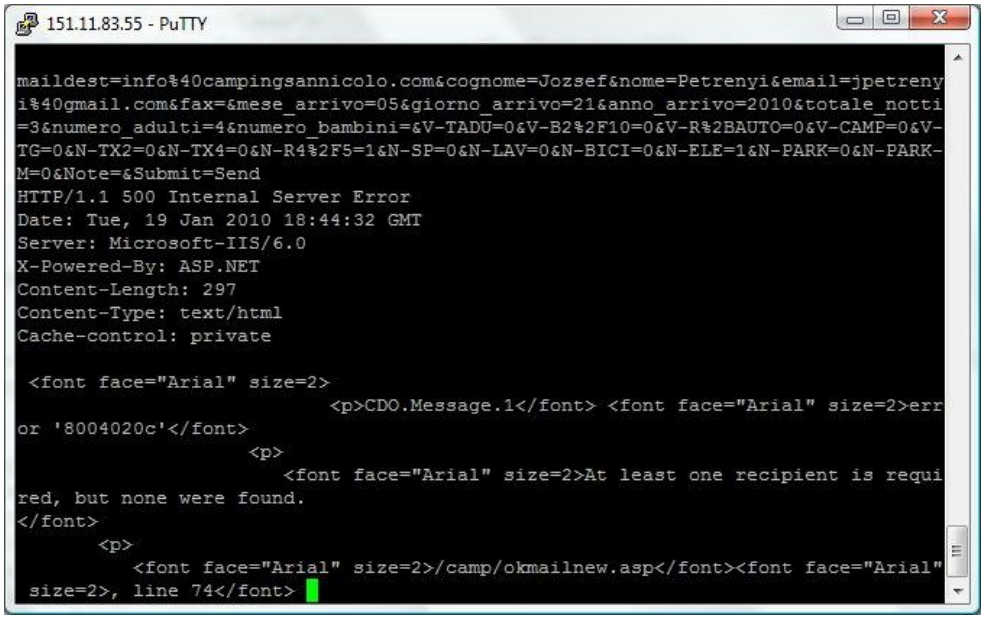
2.48. ÁBRA WEBLAP MANUÁLISAN II

Ha ez most egy böngésző lett volna, akkor értelmezte volna a HTML kódot és szép, színesszagos weblapom lett volna. De most szöveges kliensem van, abban meg így néz ki a weblap.
Képzeletben kitöltöttük a formot, vissza szeretnénk küldeni.



2.49. ÁBRA WEBLAP MANUÁLISAN III

Vegyük észre, a küldés első sorát, a POST parancsot nem ész nélkül másoltam be. Töröltem a '.' karaktert. A többi maradt. A message blokk utáni üres enter után el is ment a csomag.



2.50. ÁBRA WEBLAP MANUÁLISAN

Sajnos kiábrándító eredménnyel. Hiába találtuk meg az alkalmazást, hiába adtunk át neki egy teljesen szabályos adatcsomagot, az alkalmazás hibát dobott. Maximum annyi vigaszt meríthetünk, hogy ez egy 500-as hibakód, azaz kliensként már jók vagyunk, most a szerver dobta el az agyát.

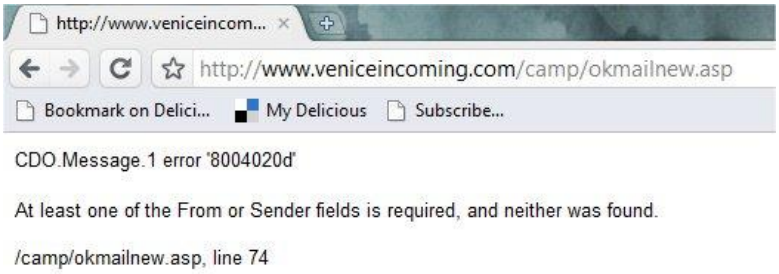
Mondtam, hogy van egy másik módszer is. Egyszerűbb is, elegánsabb is... de kevesebbet lehet belőle tanulni.

Első lépésben lementjük az iframe-ben lévő weboldal forráskódját. Remélem, ezt nem kell külön részleteznem.

```
16  <tr>
17   <td width="100%" height="500" valign="top" bgcolor="#EBE8E3">
18    <form action=".okmailnew.asp" method="POST" name="FormName">
19    <input type="hidden" name="emaildest" value="info@campingsannicolo.com">
20     <table width="599" border="1" cellpadding="4" cellspacing="0">
21      <tr>
```

2.51. ÁBRA AZ IFRAME FORRÁSKÓDJA

Megkeressük a form parancs action paraméterét. És tényleg, ott van a hibás hivatkozás az alkalmazásra. Kijavítjuk, méghozzá úgy, hogy nem a relatív hivatkozást hagyjuk benne - hiszen a HTML fájl most már itt van a lokális vinyónkon - hanem az abszolút URI-t. Elmentjük, megnyitjuk a böngészőnkben. Kitöltjük a formot, send.
Hiba.



2.52. ÁBRA A BOSSZANTÓ HIBA

Ugye látjuk, hogy ez nagyjából ugyanaz a hiba, mint amilyet a Puttyban kaptunk - csak itt a böngésző jelenítette meg a szöveget.

A hibaüzenetből egyébként annyit lehet látni, hogy ez egy asp alkalmazás, méghozzá a nevéből itélve egy levélküldő alkalmazás. Valószínűleg az lett volna a terv, hogy az alkalmazás emailben elküldi a kempingnek a regisztrációs adataimat - csakhogy a form és a progi nem igazán vannak összhangban, nincs olyan cím, ahová a csomagot küldeni kellene.

Ez ellen már nem segít semmilyen trükközés. Megkerestem a kemping emailcímét és elküldtem nekik szép, ember által is érthető szöveges formában a foglalási igényemet.

2.5.2 GEEK ÚR A FOGORVOSNÁL

Nemrégiben kellett elmennem a fogorvosomhoz. Mivel nem voltam előre bejelentve, egy kicsit várnom kellett. A dokinőről tudni kell, hogy egy kifejezetten mediterrán személyiség, gyors hangulatváltásokkal, nagy hangerővel. A váróterem együtt van a titkársággal, sőt, maga a rendelőajtó is csak a legritkább esetekben van zárva. Így egy légtérben rohangál fel-alá mindenki, harsányan kiabálva, nevetve. Még a betegek is igyekeznek hangos nyögésekkel kivenni a részüket a kavalkádból.

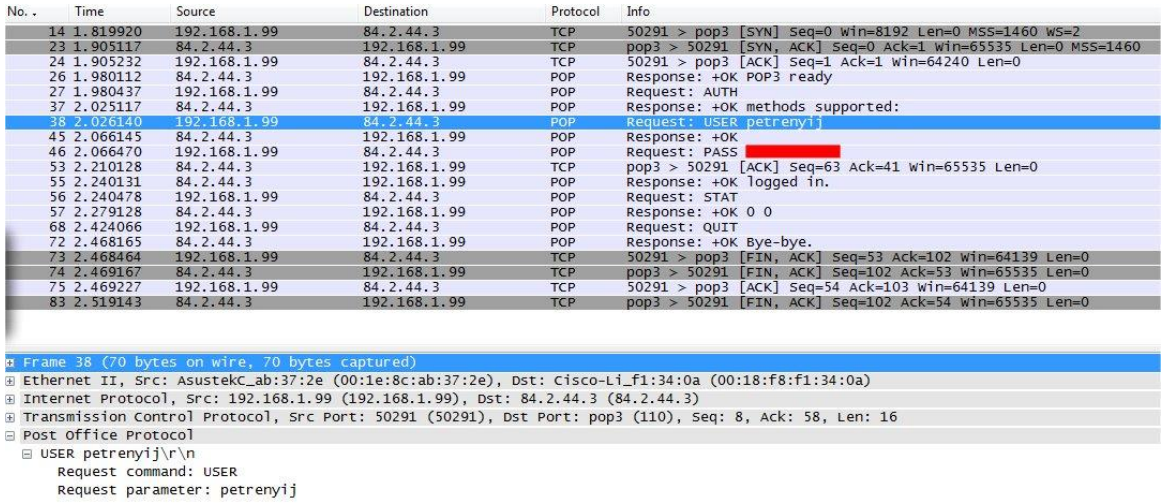
Nos, amikor itt voltam, a szokásosnál is nagyobb hajcihő volt. A fogorvos szerette volna lemondani a rendelő internet előfizetését, de a titkárnő nem tudott zöldágra vergődni az ISP ügyfélszolgálatával. Állandóan usernevet és jelszót kértek tőle - de arra már senki sem emlékezett. Nyilván volt valahol szerződés, de az évek alatt dossziéstól nyelte el valamelyik szekrény feneketlen gyomra. Mondanom sem kell, mindenki kiabált mindenkivel. A vége az lett, hogy úgy döntöttek, nem fizetik a díjakat, aztán az ISP egyszer csak rájön, hogy kvázi lemondták a szolgáltatást.

Már az utcán jutott eszembe a megoldás. Bosszantott is, mert hatalmas piros pontot szerezhettem volna az orvosnál.

Oda kellett volna kéredzkedni a számítógépükhöz. Nyolc éve vagyok törzsbeteg náluk, mások kocsit vesznek annyi pénzből, amennyit én már ott hagytam, simán odaengedtek volna.

Első lépésben letöltöttem volna egy Wiresharkot. Telepít, elindít. Majd benéztem volna a levelezőprogramba és rányomtam volna a send/receive gombra.
Ennyi.
Ezzel már gyakorlatilag készen is vagyunk.

Egyszerűen szégyen, de Magyarországon gyakorlatilag minden ISP titkosítás - azaz SSL - nélküli POP3 hozzáférést biztosít. (Eltekintve a Gmailtől, de az ugye nem túlzottan magyar.) Ez azt jelenti, hogy amikor bejelentkezek a postafiókomba, plain text formában megy át a dróton a usernév és a jelszó.



2.53. ÁBRA POP3 FORGALOM

Kitakartam pirossal, de hidd el nekem, hogy a 46-os csomagban a jelszavam ugyanolyan jól olvasható, mint a 38-as csomagban a usernevem.

Most jön a következő szégyen. Legalábbis az általam ismert ISP-k esetében a POP3 usernév/jelszó megegyezik a fő jelszóval. Azaz ugyanezzel a usernével, jelszóval lehet belépni a webes admin felületre.

Sorolhatnám még a sztorikat, de inkább már csak utalok. Nem túl régen keringett egy levél a neten. Közel 1000 magyar ftp szerver usernév/jelszó adata volt benne. Egyszerűen valaki lerakott egy stratégialiag jó helyre egy sniffert és begyűjtötte. Mind titkosítatlanul közlekedett. Nem értem... miből gondoljuk, hogy az internet barátságos hely? És akkor a bongóról még nem is beszéltem.

3 A BIZTONSÁG KÉRDÉSE A TCP/IP-BEN

3.1 SSL, TLS

Az eddig tárgyalt protokolloknak mind volt egy közös tulajdonságuk: vének, mint az országút. Ez önmagában persze még nem lenne olyan nagy baj - csakhogy azóta nem is kicsit változott a világ. Bármi elterjed, népszerű lesz - ott megjelennek a rossz arcok és el kell kezdenünk gondolkozni a védekezésről. A gond az, hogy ezek a protokollok strukturálisan alkalmatlanok a hatékony védelemre, átalakítani meg az elterjedtségük miatt nagyon nehézkes őket.

Már sokadszorra írom le, hogy öreg, meg bizonyos szempontból alkalmatlan - de eddig soha nem mondtam meg konkrétan, hogy mégis, hol szorít a cipő.
Egyrészt az autentikációnál. A HTTP még csak-csak, ott legalább vannak választási lehetőségeink. Igaz, mindegyiknek van valamilyen hátulütője. Az FTP-nél már teljesen bajban vagyunk: ha elkapjuk a hálózati forgalmat, tisztán kiszedhetjük az FTP jelszavunkat. SMTP? Láthattad. Az összes védelem az a papírtigris erősségű Base64 autentikáció. A többi, eddig éppen csak említett internetes protokoll sem lehet büszke magára: a POP3, az NNTP, a telnet szintén sima szöveges formában küldik át a felhasználói nevet és a jelszót. (Az IMAP4 egy kicsit más.)
Másrészt pedig ott van a forgalom titkosítása. Volt szó róla eddig? Nem véletlenül nem.

Mi lehet az a varázsszer, amelyik egyszerre fogja megoldani az összes protokoll biztonsági problémáját? Hát a csatorna. Azaz alagút. Azaz kapszula. Illetve ez mind így együtt.
Ugyanis magát a protokollt nem tudjuk már tovább tákolni. Járhatóbb megoldás az, hogy a két végpont között atombiztos csatornát építünk ki, aztán ezen belül úgy mennek a jelszavak, ahogy kedvük tartja. (Azért tudjunk róla, hogy vannak olyan eszközök - pl. ISA, TMG - melyek bridzselik az SSL-t és közben bele is tudnak nézni.)

A csatornázást már ismerjük. Beszéltünk róla az IPv4/IPv6 átalakításoknál.
Ugye az megvan még, hogy bár a csatorna/alagút vizuálisan nagyon erős fogalmak és tényleg sokat segítenek a kommunikáció elképzelésében - de a technológiát illetően nem igazán fedik a valóságot?
Pontosabb hasonlat lenne, ha azt mondanánk, hogy a feladó beleteszi egy dobozba a cuccot, lezárja lakattal és odaadja a postásnak. A postás elviszi a címzettnek, aki kibontja a dobozt, kiveszi a tartalmát, belerak valami mást és visszaküldi a feladónak.

A hasonlat persze sánta: a lakat nem mindig lakat. Van, amikor csak egy kibontásjelző: nem akadályoz meg senkit abban, hogy belepiszkáljon a dobozba, de a címzettnek jelzi, hogy ez bizony nem az eredeti tartalom (Authentication Header, AH). Van, amikor még csak lakat sincs: ha olyan országon vezet át a csomag útja, ahol halálbüntetés jár a piros színű csomagokért, akkor a határon gyorsan becsomagoljuk a csomagunkat egy kék dobozba, a határ túloldalán meg kicsomagoljuk (IPv6 over IPv4). És természetesen a lakat lehet igazi postásálló/gazfickóálló lakat is: akármilyen vad vidéken is vezet át a postás útja, a doboz tartalmához senki nem férhet hozzá.

Mindegyik módszer egyfajta csatornázás. Látunk is rájuk példákat hamarosan.

Az SSL az egy ilyen titkosítós-lakatos fajta csatorna. Maga a titkosítás kulcsokon alapul[5]. A kulcs egy jó nagy szám, mellyel konkrét információt lehet nyitni-zárni.

Ismétlünk.

Amikor kulccsal akarok titkosítani egy kommunikációt, akkor azt kétféleképpen tehetem meg: vagy egy szimmetrikus kulcsú titkosítást használok, vagy egy asszimetrikusat.

- A szimmetrikus kulcsú titkosítás gyors. Piszkosul gyors. Sokkal gyorsabb, mint az asszimetrikus. Ilyenkor zárásra/nyitásra ugyanazt a kulcsot használjuk.
- Az asszimetrikus titkosításnál egy kulcspárt használunk: amit az egyik kulccsal bezártunk, azt csak és kizárólag a másik kulcs tudja kinyitni. És vicaversa. A módszer előnye, hogy az egyik kulcsot (publikus kulcs) simán lehet küldözgetni a legvadabb vidéken keresztül is, ha a párja (privát kulcs) biztonságban lapul. Hátránya, hogy lassú.

A szimmetrikus kulcsú titkosításnak is van egy súlyos hibája: a kulcsot valahogy el kell juttatni mindkét kommunikáló félhez, még akkor is, ha azok többezer kilométer távolságra vannak egymástól, közöttük pedig ott feszül a hekker óceán.
Gondolom, nem kell hozzá zseninek lenni, hogy beugorjon a megoldás: hát küldjük el a szimmetrikus titkosítás kulcsát (session key) asszimetrikus titkosítással, ez lassú ugyan, de csak egyszer, maximum kétszer kell használni - utána pedig hadd szóljon, ami a csövön kifér, szimmetrikus titkosítással.

[5] A titkosítási módszerekbe, a kulcshosszúságok szerepébe nem mennék mélyebben bele. Nagyon hosszú téma - gyakorlatilag a titkosításokról és a PKI-ról külön könyveket szoktak írni.

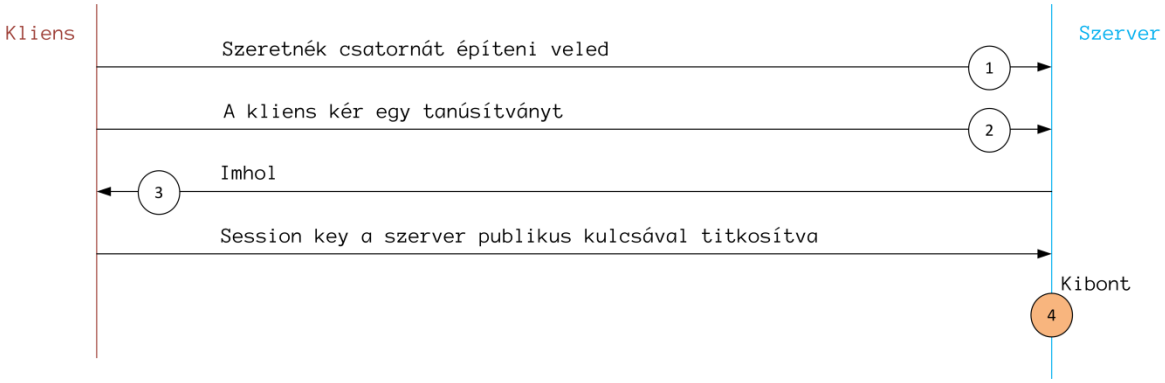
Ezt hívják úgy, hogy SSL, azaz Secure Sockets Layer. Pontosabban úgy, hogy TLS, azaz Transport Layer Security.

Evolúció a számítástechnikában is létezik.

RFC 2246, 4346, 5246

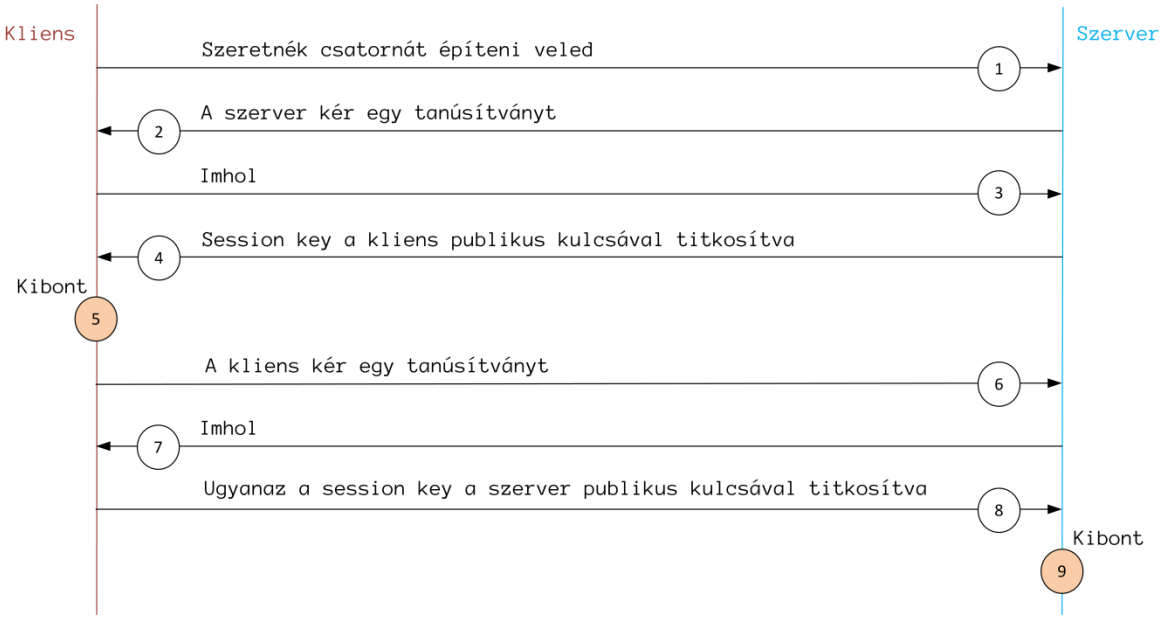
Az elv mindkettőnél ugyanaz, de van egy óriási különbség: a TLS az SSL utódja. SSL-ből volt az 1.0 (mely soha nem lett publikálva), volt a 2.0 (mely bugzott, mint macska éjjel), aztán jött a 3.0, mely már egészen jó volt - de nem folytatták a sort, mert jött helyette a TLS 1.0, aztán az 1.1 és most éppen az 1.2 az aktuális. Hogy mi minden változott az egyes verziókban, arra most nem térnék ki.

A TLS-nek van egy óriási előnye: lehetővé teszi a kölcsönös autentikációt.
De először nézzük meg az egyszeres autentikációt.



3.1. ÁBRA TLS EGYSZERES AUTENTIKÁCIÓVAL

A kliens szeretne kommunikálni a szerverrel. Elkéri a tanúsítványát. (Melyben ugye ott van hitelt érdemlően a szerver publikus kulcsa.) Ezzel a publikus kulccsal betitkosítja a session key-t, majd visszaküldi. (Előtte természetesen megállapodnak abban a maximális erősségű titkosításban, melyet még mindketten ismernek.) Az aszimmetrikus titkosítás lényegéből következően ezt a csomagot csak a szerver tudja kicsomagolni.
Miután mind a két félnél megvan a session key, életre kel a szimmetrikus titkosításon alapuló csatorna, indulhat a kommunikáció.



3.2. ÁBRA TLS KÖLCSÖNÖS AUTENTIKÁCIÓVAL

Látható, a kölcsönös autentikáció trükkje az, hogy a kliensnek is rendelkeznie kell érvényes tanúsítvánnyal.

Zavaró lehet, hogy vegyesen használtam a kulcs, illetve a tanúsítvány szavakat. Fussunk ezekkel egy kört.

Képzeld el, hogy küldök neked egy publikus kulcsot, miszerint én vagyok a Nemzeti Bank. Innentől minden titkodat ezzel a kulccsal titkosítva küldd el, lécci.
Mi lenne az eredménye? Csak én tudnám elolvasni a bizalmas információdat - pedig most már elárulhatom, hogy nem is én vagyok a Nemzeti Bank.

Éppen ezért a kulcspárokra ráépült egy hitelesítési szint. Létrejöttek minden gyanú feletti hitelesítő szervezetek. Amit ezek mondanak, az kurva fix - hogy az egyik kedvenc szerzőmtől idézzek[6].

A legismertebb hitelesítéssel foglalkozó nemzetközi cég a Verisign. Nálunk a Netlock űzi nagyban ezt az ipart.
A lényeg, hogy ezek a cégek minden gyanú felett állnak. Az ő publikus kulcsaik már beleépültek az operációs rendszerbe. (Legalábbis a Windowsokba biztosan.) Ha ők

[6] *"Básník Egon Bondy,pokaždé když mu Vladimír četl ze svého deníku, dupal podrážkami svých malýchstřevíců do podlahy a křičíval: Kurva fix! Než já najdu jeden takový obraz, takprstíčkem musím překopat celý náměstí! A von jich celý stovky sype takhle z rukávu.Vladimíre, proboha! Pište básně, kurva fix... Já už se s Vladimírem zlobit nebudu. Jáho zabiju a bude pokoj! Dyk to, co říká, je to samý, jak žiju já, jako marxist levý, vestavu permanentní revoluce, ve stavu trvalý vzpoury proti otci, protože doposavad zakaždýho zabitýho otce nastoupil syn, a tak stav se otcem, musel čekat, až ho zaseněkdo zabije... ale my jsme, kurva fix, jen a jen synové..."*

azt mondják egy publikus kulcsra, hogy az a Nemzeti Banké, akkor mérget vehetsz rá, hogy tényleg azé is.

De hogyan adja át egy hitelesítő ezt az infót? Itt jönnek képbe a tanúsítványok.
A tanúsítvány ugyanis egy korrekten összecsomagolt dosszié.

- Benne van a kérdéses publikus kulcs.
- Bele lehet tenni a privát kulcsot is, de ez veszélyesebb, mint napos csibe a root jelszóval. (Biztonsági mentés céljára, illetve SSL bridge kiépítéséhez szokták használni.)
- Rá van pecsételve egy érvényességi idő.
- Benne van a kérdéses cég egy csomó azonosító adata: név, országkód, város, stb.
- Benne van a hitelesítő cég egy csomó azonosító adata.
- Úgynevezett egyéb adatok.
- Majd ez az egész - pontosabban egy kis darabja - alá van írva a hitelesítő cég privát kulcsával. (Azaz ha a hitelesítő cég mindenhol meglévő publikus kulcsával el tudom olvasni a betitkosított darabot, akkor a tanúsítvány egész biztosan ugyanaz, mint amelyet a hitelesítő cég kiállított.)

A tanúsítványoknak szintjei vannak. Csak a legpimflibb szinten lehet ügyet intézni a weben. A többihez szépen be kell vándorolni a cég hivatalos papírjaival, igazolva, hogy tényleg mi vagyunk mi.

Léteznek kevésbé erős tanúsítványok is. Mi magunk is gyárthatunk hitelesítő szervert. (Certificate Authorithy, CA szerver) Ha mi magunk megbízunk benne, akkor megbízhatunk az általa kiosztott tanúsítványokban is. Ha a partnereink is megbíznak benne - megjegyzem, nem szoktak - akkor használhatják ők is. (Ha cégen belül bohóckodunk, ott még elmegy a saját CA. Cégen kivül már égő.)

Oké, térjünk vissza a TLS-hez. Ott jártunk, hogy kiépül a biztonságos csatorna. (Doboz, lakatok, türelmetlen postás.)

Mi utazhat a csatornában?

Stay tuned.

3.2 SSH

Most ugyanis egy másik protokollról lesz szó, mely látszólag nagyon hasonlít ugyan az SSL-re, de ha jobban megnézzük, láthatóvá válnak a különbségek.

Kezdjük történelemórával.

A Unix világban nagyon népszerűek voltak az RSH és barátai: RCP, RLOGIN[7]. (Hogy végülis melyik protokoll melyik csomagnak volt része, abba most nem mennék bele. Nekünk Windows oldalról lényegtelen.) Nagyjából ugyanebbe a körbe tartozott a telnet is. Ez utóbbit nézzük meg jobban, mert ilyesmi van a Windowsban is[8].

> RFC 15, 854

A telnet egy kliens-szerver alapú alkalmazás. A telnet szerver a 23-as TCP porton várja a kliensek jelentkezését. Maguk a kliensek tipikusan szöveges programok, mivel a szerver is parancssor alapú hozzáférést biztosít. Láttunk is már példát rájuk, ilyen volt a Putty, de ilyen a command promptból indított telnet parancs is.

> Ne keverjük össze a telnet protokollt a telnetelni igével. A telnet protokoll a 23-as TCP porton ad remote shell-t, a 'telnetelni' kifejezés viszont azt jelenti, hogy a telnet segédprogram segítségével tetszőleges porton keresztül hozzáférni egy szerver olyan protokolljához, mely szintén biztosít parancssor jellegű hozzáférést.
> A *'telnet mail.t-online.hu 25*' parancs - dacára annak, hogy a telnet programot használjuk - például egy SMTP sessiont indít. Ugyanez igaz a Putty-ra is.

Mindegyik protokoll távoli varázslásokra adott lehetőséget. És egyik protokoll sem volt biztonságos: a felhasználónév/jelszó párosok kódolatlanul utaztak a csomagokban.

> RFC 4250-4254

Aztán jött a Secure Shell, azaz SSH (port 22). Egyszerűen fogták a fenti protokollmatuzsálemeket és beléjük hegesztettek egy biztonságos csatornaépítési képességet. Az elv ugyanaz, mint az SSL-nél: a csatornaépítéshez kulcsokat használunk. Vigyázat: nem tanúsítványokat! Azaz az SSH-hoz nem kell kiépített PKI rendszer, a kliensek maguknak gyártanak kulcsokat.

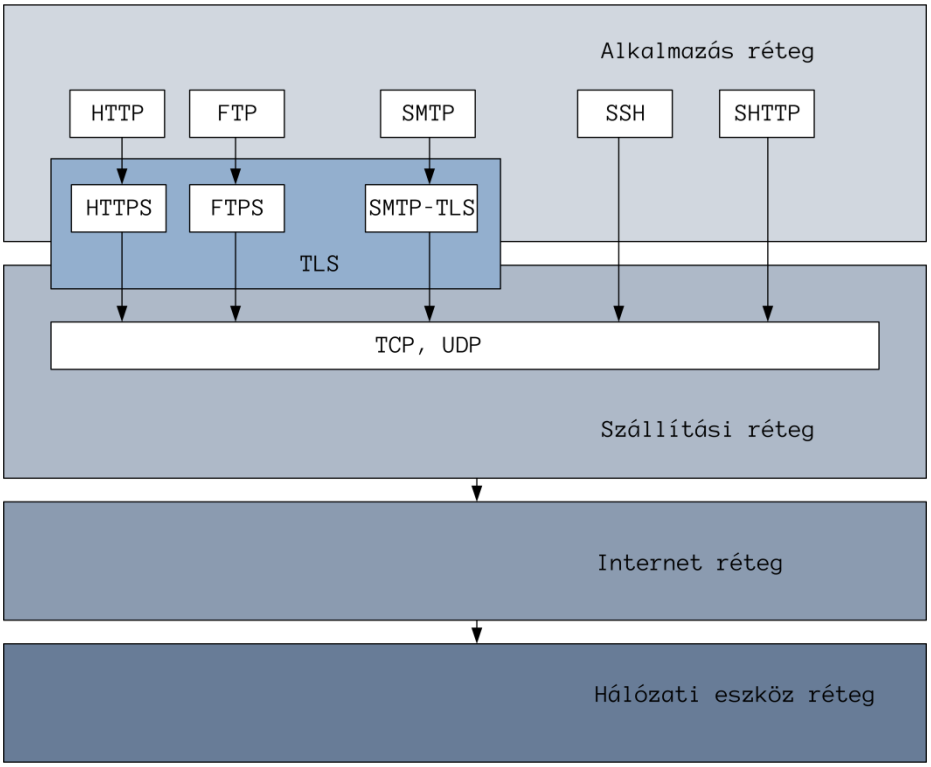
[7] Remote Shell, Remote Copy, Remote Login
[8] Figyelem, Vista óta már a kliens sem része az alap telepítésnek. Külön feature, melyet utólag kell hozzáadni a rendszerhez.

Így végül az SSH gyakorlatilag leváltotta a telnetet, az RSH-t és az RCP-t (SCP).

> Itt is érdemes elkülöníteni a protokollt az ugyanolyan nevű segédprogramtól. Szerencsére az OpenSSH óta ez nem olyan nehéz.

Összefoglalva:

- Az SSL/TLS egy csatornaépítő protokoll. PKI-t használ, user/password alapú autentikációt nem tud. Magát a kiépített csatornát bármelyik másik protokoll képes használni, feltéve, ha felkészítették rá. Példák: HTTPS, FTPS, POP3S, NNTP-TLS, SMTP-TLS. A protokoll a szállítási réteg határán dolgozik - azaz a HTTP felől úgy tűnik, hogy a szállítási rétegben, a TCP felől pedig úgy, mintha az alkalmazás rétegben dolgozna.
- Az SSH egy távoli menedzselést lehetővé tevő protokollcsalád. Saját magának épít csatornát az SSL-hez hasonló módon, de nem kell hozzá PKI rendszer. Más protokollokat nem igazán támogat9. Az alkalmazás rétegben dolgozik.



3.3. ÁBRA A TLS/SSL ÉS AZ SSH HELYE A TÉRKÉPEN

[9] Erről azért még beszélünk.

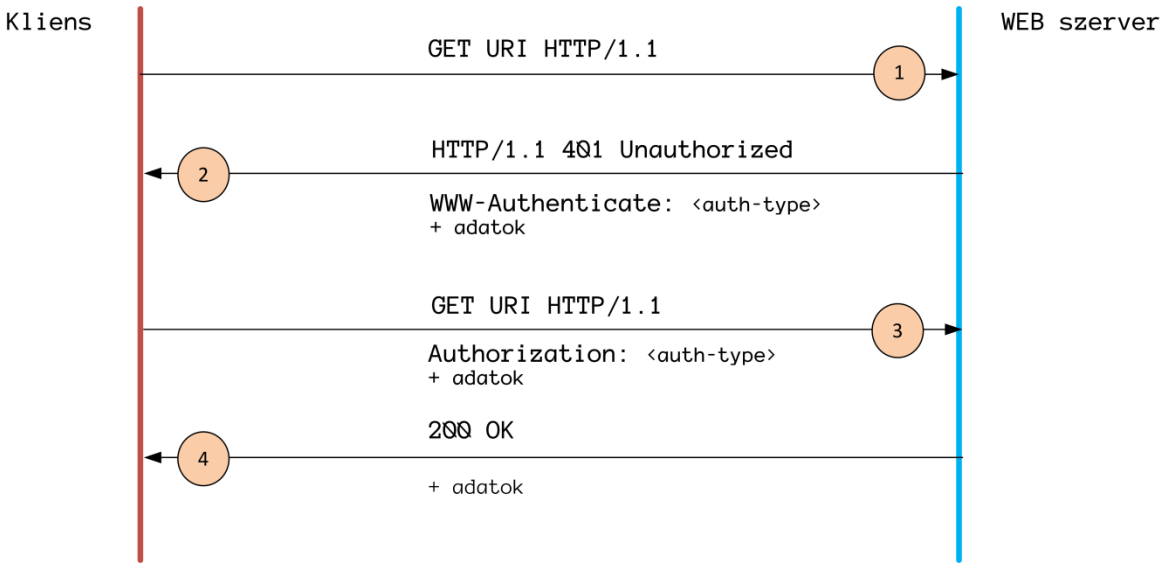
3.3 AZ ISMERTEBB PROTOKOLLOK BIZTONSÁGOSABBÁ TÉTELE

3.3.1 HTTP

Ez egy olyan megfoghatatlan kifejezés, hogy 'biztonságosabbá tétele'. Eleve a HTTP azon protokollok közé tartozik, amelyikben beépítve is van valamilyen biztonsági funkció: tud autentikálni.
De ez csak a biztonság egyik oldala[10]. Hiába jó az autentikáció - ha a forgalmat nem lehet titkosítani, akkor nem küldhetek érzékeny adatokat. Márpedig egy webáruházban ténferegve, a Paypal-en vagy a webbankunkban kattogtatva nem győznek repkedni a kényes információk.

3.3.1.1 AUTENTIKÁCIÓ

Mikor történik egy HTTP forgalom során autentikáció? Ha a szerver kéri. Mikor kéri a szerver? Amikor az alkalmazás fejlesztői úgy gondolták, hogy innentől bizonyos weboldalak csak akkor legyenek elérhetőek, ha a próbálkozó meg tud adni egy megfelelő jogosultsággal bíró felhasználói nevet és egy jelszót - vagy legalább hitelt érdemlően bizonyítani tudja, hogy országos cimborája a webmesternek.



3.4. ÁBRA HOPPÁ, AUTENTIKÁCIÓ

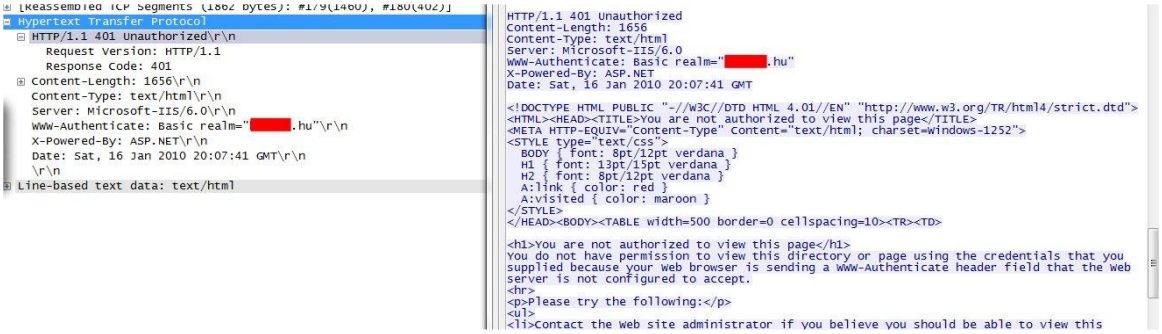
A kliens elmegy a webszerverhez, lekér egy weblapot. A webszerver kliens oldali hibát (401) jelez vissza, melyben megadja, milyen autentikációs módszerrel lehet

[10] A tízezerből. A Koh-i-noor-nak nem csiszoltak annyi oldalt, mint amennyi a biztonságnak van.

megkörnyékezni. A kliens összeállítja az autentikációs csomagot és elmegy újból a weboldalért. Ha jó volt a csomag, akkor meg is kapja.

Melyek lehetnek azok az autentikációs módszerek?

3.3.1.1.1 BASIC



3.5. ÁBRA BASIC AUTENTIKÁCIÓ

A fenti ábrán egy IIS6 webszerveren futó ASP.NET alkalmazás van olyan remekül beállítva, hogy Basic autentikációt kér. Bár a képen nem látszik, de az interneten keresztül.

A klienstől elment a kérés, valami ilyesmi formában:

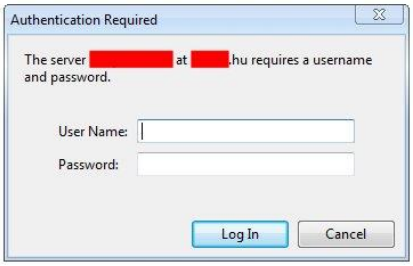
```
GET /micsodacsoda/index.html
```

Erre jött a 401-es kódú válasz:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic
      realm="ecpecckimehecc.hu"
```

Ezzel jelzi a szerver, hogy autentikációt kér, abból is a Basic tipusút. Az autentikáció az ecpecckimehecc.hu tartományból fog történni.

A kliens először is bekéri az adatokat.



3.6. ÁBRA AUTENTIKÁCIÓS ABLAK

Majd bősz matekozásba kezd. Összerakja a felhasználói nevet és a jelszót egy karakterláncba (kettősponttal elválasztva), majd az egészre ráküldi a félelmetes Base64 kódoló algoritmust. Végül megismétli az előző kérést, de immár mellécsomagolja a kódot is.

```
GET /micsodacsoda/index.html
Authorization: Basic
      dXNlcm5hbWU6cGFzc3dvcmQ=
```

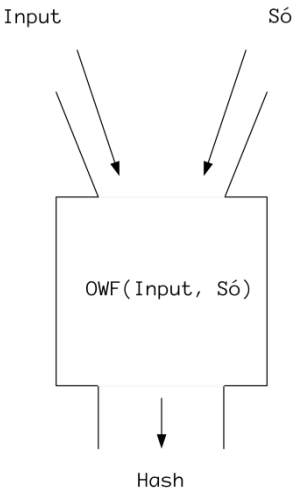
Amennyiben az autentikáció sikeres volt a fenti usernév/jelszó párossal, megnyílik előttünk az oldal.

Meg mindenki más előtt is, aki el tudja kapni a vad interneten ezt a forgalmat. Bónuszként jár hozzá egy érvényes felhasználói név az eccpecckimehecc.hu tartományból. Jelszóval.

Gondolom, sejted, hogy nem ez a legbiztonságosabb módszer. Szigorúan csak akkor javallott a használata, ha a kommunikáció más módszerrel már védve van.

3.3.1.1.2 DIGEST

Kezdjük rögtön egy definícióval: a digest az a hash másik neve.



Elcsépelt példa, már használtam egy másik könyben is a hasonlatot (és már akkor is mástól loptam): a hash az gyakorlatilag a darált hús. Azaz van egy szép nagy szám, azt ledaráljuk a húsdarálón - és kapunk egy kisebb számot. Ez teljesen jellemző arra a nagy számra, amelyből kiindultunk, de nem lehet belőle kiindulva visszakapni a kiindulási számot. (Mint ahogy a fasírtmasszából sem tudjuk visszaállítani a disznót. Különösen a menzán nem, mert az a massza még csak nem is látott húst.)

A végterméket nevezzük hash-nek, a húsdarálót pedig OWF-nek. Nem, nem WTF-nek.

> OWF: One Way Function. Ide tartozik minden olyan függvény/eljárás, mely csak az egyik irányba működik.

Finom a darált hús só nélkül? Nem igazán. Ezért a kiindulási számhoz (mely ugye simán lehet egy karakterlánc, egy blob[11], de akár egy komplett bináris fájl is) hozzá szoktak tenni egy többé-kevésbé változó másik számot, az úgynevezett sót is.

Digest autentikációból legalább kétfajta létezik:

- Az eredeti Digest autentikáció a HTTP 1.0-ban.
- A fejlettebb Digest autentikáció a HTTP1.1-ben.

A két eljárás nem kompatibilis egymással. Mindkettő gyökeresen más OWF-et használ. Ha most azt hiszed, hogy oldalakon keresztül nekiállok részletezni, hogyan gyötrik meg ezeket a szerencsétlen számokat, hogyan választják ki a sót - akkor tévedsz. Bonyolult dolgok ezek.

> A korábban említett TCP/IP Alapok című könyvben ki van bontva néhány algoritmus a PPP fejezetben.

Azért pár mondatban összefoglalom a lényeget: az autentikáló szerveren megvan a felhasználó neve és jelszava. A szerver küld egy sót a kliensnek. A begépelt usernévvel, jelszóval és a sóval a kliens eltáncolja az OWF-et, majd a digestet visszaküldi a szervernek. A szerver szintén végig tudja számolni az eljárást - és ha ugyanazt a végeredményt kapja, akkor sikerült az autentikáció. Látható a módszer előnye: a vad interneten nem utazik át visszafejthető formában a jelszó.

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Digest
      realm="ecpecckimehecc.hu"
      nonce="asldenwpbsdeifbgiuitrdubnsdtrbuin"
```

A 401-es üzenet WWW-Authenticate fejléce ebben az esetben kibővült egy újabb paraméterrel. A nonce maga a só, amellyel a kliensnek dolgoznia kell.

Sehol nincs szigorúan rögzítve, minek kell lennie a sónak. Abszolút a szervereken múlik, milyen algoritmussal generálják. (Azt azért sejthetjük, hogy nem lehet konstans érték.)

[11] Binary Large OBject: bazi nagy bináris adatkupac.

A kliens veszi a lapot, bekéri a felhasználói adatokat. Feltűrt ingujjal összegyúrja az egészet, majd az eredményt visszaküldi a szervernek.

```
GET /micsodacsoda/index.html
Authorization: Digest
      username="EgahazaDideki"
      realm="eccpecckimehecc.hu"
      nonce="asldenwpbsdeifbgiuitrdubnsdtrbuin"
      uri="/micsodacsoda/index.html"
      response="ufhbvuefvbfujfb456546sfs6ef5gb4se6sefb486"
```

Ez az autentikációs módszer már jóval fejlettebb, mint a Basic. A Windows világban viszont van néhány kellemetlensége. IIS5 alatt például a webszervernek DC-nek kell lennie - és a jelszavakat reverzibilis titkosítással kell tárolnia. (Nemnemsoha.)
IIS6 alatt már nem ennyire durva a helyzet, az Advanced Digest autentikációnál már MD5 hash formában tároljuk a jelszavakat a DC-n. (AltSecId tulajdonság a user objektumon.) Itt már csak azon kell túltennünk magunkat, hogy ezt az autentikációt bármelyik böngésző tudja használni, feltéve, ha úgy hívják, hogy Internet Explorer, a felhasználónak és az elérendő szervernek ugyanabban a tartományban kell lennie (ha nem, akkor trust kell), az IIS szervernek és a DC-nek minimum Windows 2003-nak kell lennie és a felhasználó természetesen domaintag kell legyen. És az MD5 már nem is annyira cool.

A Digest és a Basic általános autentikációs eljárások voltak. A következőkben rástartolunk az IIS autentikációs módszereire. (Mondjuk úgy, hogy ezek elvi leírások lesznek. Mivel ez nem IIS könyv, nem fogok belemenni abba, hogy melyik autentikációs módszert melyik IIS verzió tudja, meg hogy milyen változások történtek az egyes módszerekben a különböző verziók között.)

3.3.1.1.3 Anonymous autentikáció

Egyszerű, mint a faék. A szerver senkitől nem kérdez semmit. Mindenki, az anonymous felhasználóhoz rendelt felhasználó nevében fog hozzáférni a kért lapokhoz. Ez a felhasználó alapértelmezésben az IUSR_computername felhasználó, de módosítható.

3.3.1.1.4 Integrated Windows autentikáció

Határozottan erős autentikáció. A körülményektől függően hol NTLM, hol Kerberos autentikációt használ.
Eddig tartottak a jó hírek.

Hátulütők:

- NTLM autentikáció:
- Nem megy át a HTTP proxyn.
- Csak Internet Explorer böngészővel működik.
- Kerberos:
- A kliensnek közvetlenül kell elérnie egy tartományvezérlőt. Az ilyesmi az interneten nem igazán megszokott.

Maradjunk annyiban, hogy ez remek módszer, feltéve, hogy a cég belső hálózatáról próbálkozunk.

3.3.1.1.5 .NET PASSWORD AUTENTIKÁCIÓ

A .NET Framework része egy úgynevezett .NET Passport single sign-in szolgáltatás, melyen keresztül elérhetjük az alkalmazásainkból a Microsoft .NET passport rendszerét. (Melyet mostanában Live ID-nak hívnak.)

3.3.1.1.6 CLIENT CERTIFICATE MAPPING AUTENTIKÁCIÓ

Ez se túl bonyolult. Ha a kliensnek megfelelő tanúsítványa van, akkor minden egyéb autentikálás nélkül hozzáfér a kért tartalomhoz. Létezhet one-to-one, illetve many-to-one verzióban, ez utóbbi esetben több felhasználót rendelhetünk ugyanahhoz a tanúsítványhoz.

3.3.1.1.7 FORM-BASED AUTENTIKÁCIÓ

A szerver autentikáció esetén átirányítja a klienst egy webes formhoz, melyet egy autentikáló alkalmazás kezel. Tipikus példa egy OWA bejelentkező képernyő.

3.3.1.1.8 EXTENDED PROTECTION - CBT

RFC 2743

Egy ún. Generic Security Service (GSS) API-n keresztüli autentikáció. A műveletben kliens oldalról egy Channel Binding Token (CBT) vesz részt. Ez a token autentikálja a felhasználót. Az autentikáció szigorúságát (CbtHardeningLevel) külön lehet konfigurálni. (Csak CBT-vel lehessen, vagy annak hiányában más módszerrel is.)
Kizárólag TLS felett működik (HTTPS) - azaz előzetesen kell neki a biztonságos csatorna.

3.3.1.1.9 Extended Protection - SPN

Meglehetősen hasonló az előbbihez, de itt a token nem csatornát köt be, hanem egy szolgáltatáshoz rendelődik. Az összekötés a Service Principle Name (SPN) alapján történik. Akkor használják, ha a kapcsolat alatt nincs TLS, illetve olyan trükkös esetekben, amikor vagy egy proxy, vagy egy NLBS farm beleszól a TLS csatorna működésébe.

3.3.1.1.10 Összefoglaló táblázat az autentikációkról (IIS6)

3.1. táblázat

Módszer	Biztonság	Jelszó kódolást	Proxyn átmegy?	Böngésző?
Anonymous authentication	None	N/A	Igen	Bármelyik
Basic authentication	Alacsony	Base64 kódolású szöveg	Igen	A legtöbb
Digest authentication	Közepes	Hashed	Igen	Min. Internet Explorer 5
Advanced Digest authentication	Közepes	Hashed	Igen	Min. Internet Explorer 5
Integrated Windows authentication	Magas	Hashed, NTLM esetében. Egyébként Kerberos ticket.	Nem, maximum PPP-vel használva	NTLM: Internet Explorer 2.0 felett Kerberos: Windows 2000 + min. Internet Explorer 5
.NET Passport authentication	Magas Naná.	Titkosított	Igen. SSL	Internet Explorer és Mozilla

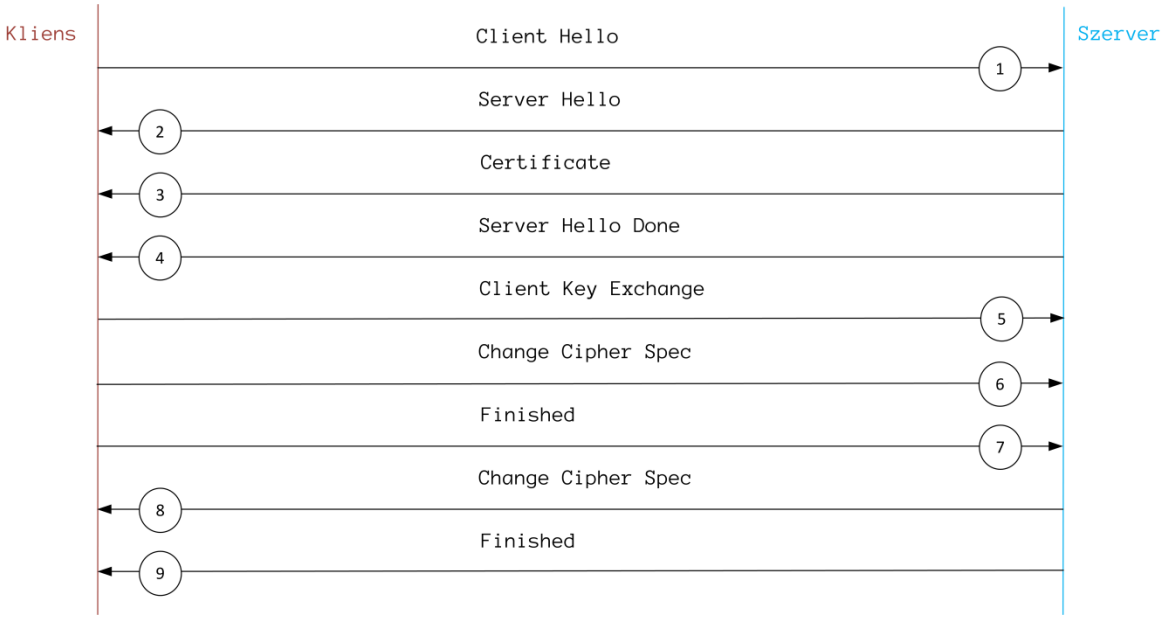
3.3.1.2 HTTPS

A TLS-ről már írtam. Óvatlanul azt is elárultam a fejezet végén, hogy a TLS csatornán átzavart HTTP-t nevezzük HTTPS-nek.

De azt még nem mondtam, hogy igazából a HTTP két szép szeméért találták ki eredetileg az SSL protokollt. Hogy biztonságosabbá tegyék az egyik legsűrűbben használt protokoll forgalmát. Legyen egy olyan burok, egy vastagfalú külső cső a konkrét kommunikáció körül, mely biztosítja, hogy az adatfolyamba a két kommunikáló félen kívül senki nem láthat bele. (A TMG még mindig kuncog a sarokban.)
Aztán annyira jól sikerült a produkció, hogy apránként az összes fontosabb protokollt alkalmassá tették arra, hogy használja.

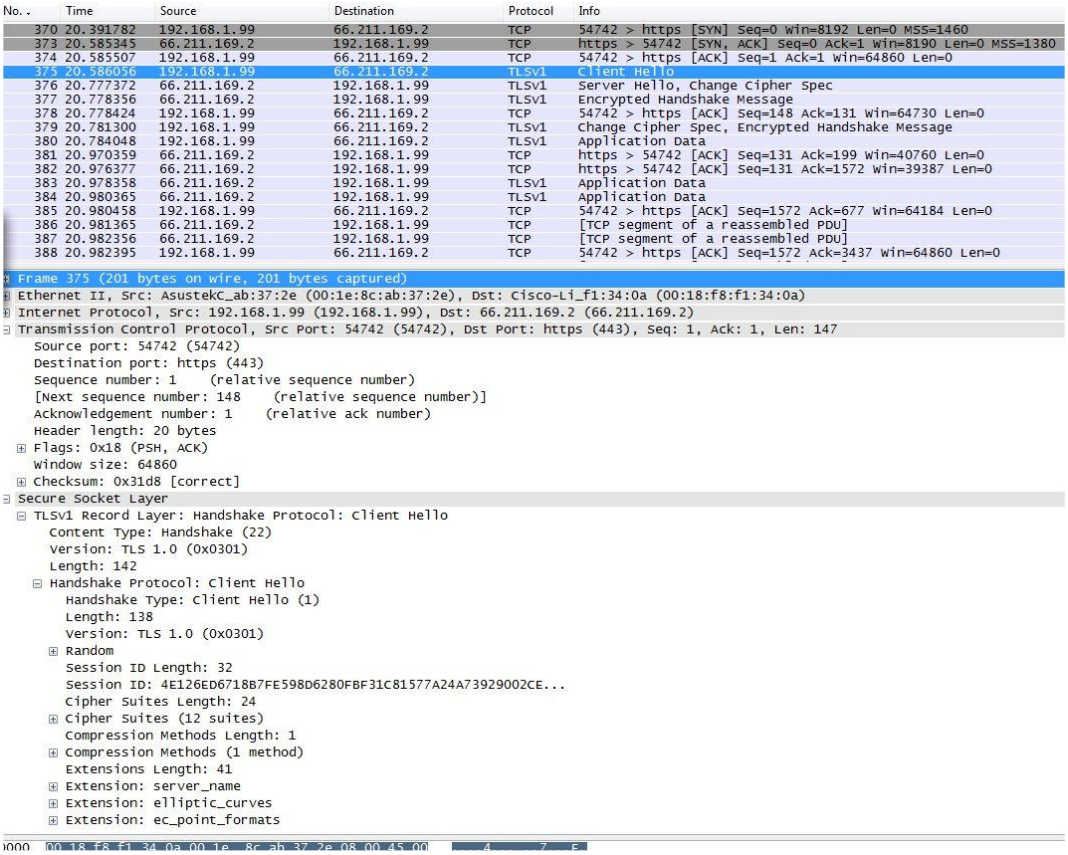
Gondolom az elv már mindenkinek tiszta. Ezen nem is akarok túlzottan rágódni.

Inkább nézzük, milyen buktatókba tudunk beleszaladni akkor, amikor a HTTP-t a TLS csatornán vezetjük keresztül.



3.7. ÁBRA HTTPS RÉSZLETESEN

A fenti ábrához hasonlót láthattunk már. (*3.1. ábra TLS egyszeres autentikációval*) Ugyanarról van szó, csak most konkrétan a HTTP-t vizsgáljuk meg - és egy kicsit mélyebben.



3.8. ÁBRA A CLIENT HELLO CSOMAG

A capture ábrán láthatjuk, hogy a kapcsolat a szokásos módon kezdődött: a szállítási réteg szintjén kiépült egy TCP session.

1.
A kliens küld egy Client Hello parancsot a szervernek. Ezt a parancsot vehetjük szemügyre részletesen is a fenti ábrán. Láthatjuk, hogy a content type az egy kézrázás (22-es kód), a kézrázás protokollja pedig a Client Hello. Láthatunk még néhány érdekességet: én például komolyan meghatódtam, amikor a kliensem által felkínált listában (cipher suites) megláttam az elliptic_curves lehetőséget. Istenem, eddig még csak hallottam arról, hogy létezik ilyen... most meg látom is.

2.
A szerver küld egy Server Hello üzenetet. Ebben mondja meg, hogy milyen SSL/TLS verziót fognak használni. (Jelen példában TLS 1.0, mert ennyit tud a kliensem.)

3.
A szerver egy Certificate üzenetben elküldi a tanúsítványát.

4.
A szerver egy Server Hello Done üzenettel jelzi, hogy részéről befejezte a kézrázást.

5.
A kliens küld egy Client Key Exchange üzenetet, benne a session kulccsal. Ezt a csomagot a szerver publikus kulcsával titkosítja be.

6.
A kliens küld egy Change Cipher Spec üzenetet. Ezzel jelzi, hogy innentől mindent a session kulccsal fog titkosítani.

7.
A kliens küld egy Finished üzenetet. Ezzel jelzi, hogy részéről véget ért a csatornaépítés,

8.
A szerver is küld egy Change Cipher Spec üzenetet. Értelemszerűen a jelentése ugyanaz, mint korábban a kliensnél - innentől kezdve a szerver is mindent a session kulccsal titkosítva fog küldeni.
9.
Végül a szerver is küld egy Finished üzenetet. A csatorna elkészült. Innentől aztán már abszolút semmit sem látunk a forgalomból. (Pontosabban látjuk, csak nem tudjuk értelmezni.)

Kérdés egy nyalókáért: mindenki látja, mi itt a probléma?

Ha nem, akkor elmondom. Mit látunk a Client Hello csomagban? Pontosabban, mit nem? Például határozottan nem látunk GET üzenetet, logikusan emiatt nem látunk HOST nevű headert sem. Ez természetes, hiszen itt még csak a TLS csatorna kiépítését láttuk. A HTTP majd csak később lesz beleterelve.
Igenám, de akkor milyen tanúsítványt fogunk kapni a szervertől? Konkrétan milyet akkor, ha ez egy olyan webszerver, mely több virtuális host-ot kezel?
Mit tenne az ISP-m, ha szólnék neki, hogy mostantól a mivanvelem.hu és az emaildetektiv.hu weblapjaimat HTTPS-en keresztül szeretném elérhetővé tenni? Először valószínűleg magához nyúlna. Aztán meg én[12], amikor közölné az árait.

Nem megy. A tanúsítvány vagy a mivanvelem.hu vagy az emaildetektiv.hu névre van kiállítva. De amikor a kliens benyögi a Client Hello üzenetet, akkor még nem lehet tudni, melyiket akarja elérni. Emiatt a szerver csak egy olyan tanúsítványt tud visszaküldeni, amely az IP címére lett kiadva.

Mi következik ebből?

- Az ISP mindegyik virtuális host-ot más IP címre rakja. (Ezért lenne drága.)
- Az ISP ugyanazt a tanúsítványt rendelné mindegyik virtuális host-hoz.
- Ha mindegyik virtuális host ugyanabban a gyökértartományban van, akkor használhat wildcard tanúsítványt.
- Ha nem - mint nálam - akkor kénytelen lesz Subject Alternate Name tanúsítványt használni. Ezzel nem csak az a baj, hogy k. drága, hanem az is, hogy ha bejön egy újabb host, akkor a teljes SAN tanúsítványt kell megújítani. Ez pedig érinti az összes virtuális host-ot.

RFC 4366

Erre a csapdahelyzetre jelent megoldást a Client Hello parancs egy kiterjesztése, az ún. Server Name Indication (SNI). Ebbe a mezőbe írja bele a kliens, hogy melyik virtuális host-ot célozta be, így a szerver el tudja dönteni, melyik tanúsítványt küldje vissza. Ehhez persze olyan böngészőre is van szükségünk, mely ismeri az SNI-t: minimum Firefox 2, Opera 8 vagy Internet Explorer 7.

Elemezgessük még egy kicsit ezt a folyamatot. Alaposan átnézted, amit írtam? Össze is vetetted a capture fájl által mutatott folyamattal? És még nem hőbörögsz?

[12] Mármint saját magamhoz.

A 2. pontban azt írtam, hogy a szerver küld egy Server Hello üzenetet. Valójában viszont (376. csomag) küld mellé egy Change Cipher Spec üzenetet is. Tudjuk, hogy ez mit jelent: a szerver innentől a session kulccsal titkosítva küld mindent. és valóban: a 377. csomagban elméletileg egy Certificate üzenetnek kellene érkeznie, de már csak annyit látunk, hogy Encrypted Handshake Message.

Na de ezt... hogyan? Milyen színű session kulcsot használ a szerver, amikor a kliensem még el sem kezdett gondolkodni arról, hogy generáljon egyet?

A megoldás kulcsa az első Client Hello csomagban rejlik. Ott van olyan, hogy Session ID mező, melynek értéke egy bazi hosszú szám. A kliens ugyanis észleli, hogy erről a gépről már kapcsolódtunk a szerverhez, azaz már van egy érvényes session kulcs mind a két gépen. Nincs más dolga, mint hogy közölje a szerverrel, hogy melyik session volt az, amelyiknek a kulcsát újra tudjuk hasznosítani.
Így a szerver már a második lépésben át tud váltani titkosított kommunikációra. Nyilván innentől borul a táncrend, hamarosan a kliens is követi.

Még mindig a csatornaépítés.

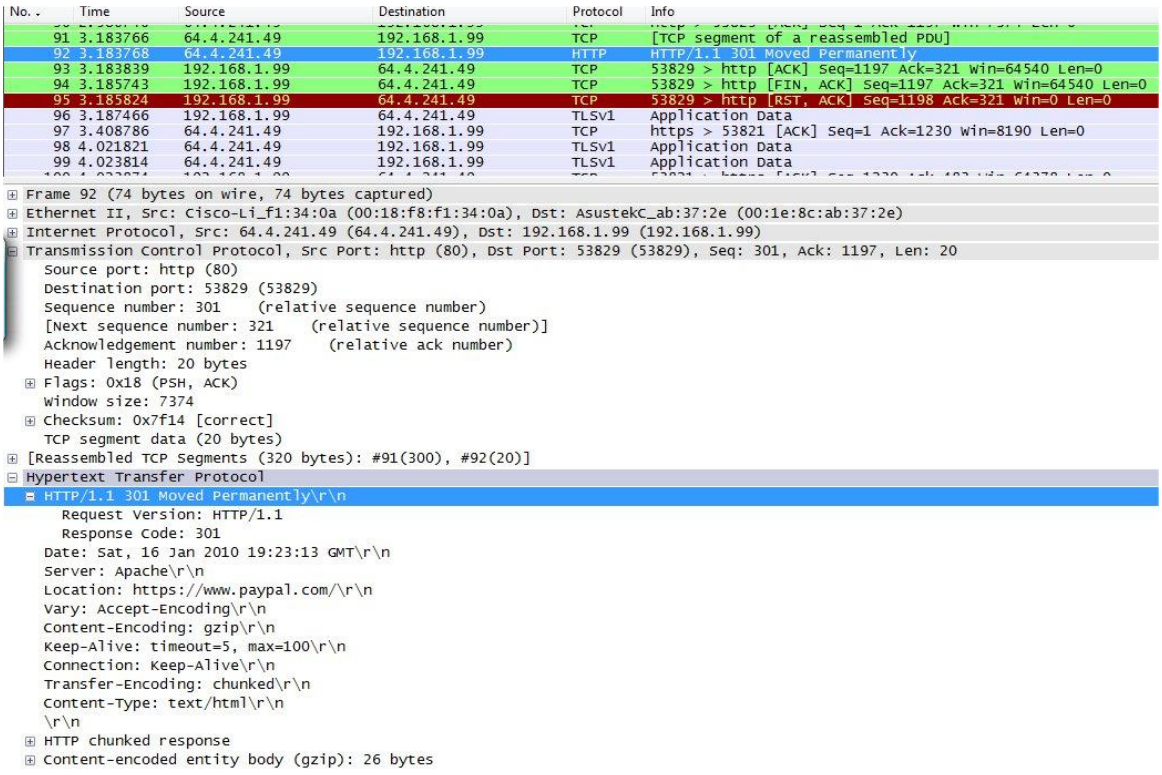
Az ábrán (*3.7. ábra HTTPS részletesen*) az egyszeres autentikációs TLS csatorna kiépítését láthattuk. Természetesen csatornaépítés itt is létezik kölcsönös autentikációval is - de ilyet elég nehéz csak úgy találni a neten. Gondold el, hogy például csak az férne hozzá a Paypal-hez vagy az Amazonhoz, akinek az otthoni gépén érvényes - és igazi - tanúsítványa lenne.

Nem hiszed el, de még mindig a csatornaépítést fogjuk boncolgatni.

A TLS tud egy érdekes trükköt az SSL-hez képest. Azt ugye már a hátulgombolós csöppségek is tudják, hogy a HTTP a 80-as porton megy, a HTTPS meg a 443-ason. (Bár láttam már hiperaktív rendszergazdákat, akik átpakolgatták, de a kettősség, mármint a külön port megmaradt.)
A TLS elvileg képes megugrani azt, hogy egy alkalmazás ugyanazon a 80-as porton forgalmazzon titkosított és titkosítatlan csomagokat - azaz menetközben tudjon TLS-re váltani.

Legalább annyit mondjunk, hogy vov.

De mielőtt megnéznénk, hogyan csinálja ezt a trükköt, lássuk, hogyan csináltuk ezt eddig, a hagyományos módon.



3.9. ÁBRA HTTP REDIRECT

Beírtam a böngészőmbe, hogy http://www.paypal.com. Erre azt mondta a szerver, hogy

```
HTTP/1.1 301 Moved Permanently
      Location: https://www.paypal.com
```

Azaz csak azért fut a webszerver 80-as portján egy alkalmazás, hogy az odaérkező kéréseket átdobja a szerver 443-as portjára.

Ennél sokkal elegánsabb az a módszer, amikor menetközben upgrade-ljük fel a kommunikációt HTTP-ről HTTPS-re. Ehhez a következő formában kell kliens oldalról kiadni a GET parancsot:

```
GET http://kintisvagyokbentisvagyok.hu/kint.aspx HTTP/1.1
      Host: kintisvagyokbentisvagyok.hu
      Upgrade: TLS/1.0
      Connection: Upgrade
```

Erre ezt fogja mondani a szerver:

```
HTTP/1.1 101 Switching Protocols
      Upgrade: TLS/1.0, HTTP/1.1
      Connection: Upgrade
```

És belekezdenek egy TLS csatorna kiépítésébe, a már korábban látott módon. Amint felépült a csatorna, a szerver válaszol a kliens eredeti GET kérésére.

Már ha tud.

Képzeljük el, mi van, ha a szerver nem ismeri ezt az Upgrade koreográfiát? Majd mindenféle TLS csatorna kiépítése nélkül visszatolja a kényes weblapot, csak úgy, HTTP-n keresztül?

Nyilván a kliensnek kell még ennél is óvatosabbnak lennie. Először nem a GET-et kell betolni az ajtón, hanem be kell próbálkozni az OPTIONS paranccsal.

```
OPTIONS * HTTP/1.1
      Host: kintisvagyokbentisvagyok.hu
      Upgrade: TLS/1.0
      Connection: Upgrade
```

Ha a szerver partner, akkor kiépül a csatorna, a kliens elküldi az igazi GET parancsot. Ha nem, akkor a kliens nem küld semmit.

3.3.1.3 SHTTP

Megint történelemóra.

Tudni kell, hogy az SSL-t a Netscape fejlesztette ki. Azaz nem volt róla RFC - de ettől függetlenül megért 3 verziót és meglehetős népszerűségre tett szert.

RFC 2660

Erre gondolták azt az IETF-nél, hogy nekünk is kell egy ilyen. Igaz, első körben ők csak a HTTP-t akarták felturbózni.
Így született a Secure HTTP, azaz SHTTP.

Habár maga a protokoll ránézésre egészen kellemes tulajdonságokkal bírt, elterjedni nem igazán tudott. A kor két óriása (Netscape és Microsoft) a HTTPS mellett tette le a voksát, a böngészőpiac maradék 0.0001%-a meg nem érdekelte a kutyát sem.

Mik is ezek a kellemes tulajdonságok?

- Kinézetre teljesen ugyanolyan szerkezetű, mint a HTTP. Ez határozott szándék volt, hogy a fejlesztőknek ne kelljen sok újat tanulniuk.
- Nem igényel külön portot. A 80-as abszolút jó neki.
- A titkosítási módszereknek valami hatalmas arzenálját vonultatja fel.
- Nem kell hozzá PKI, csak szimmetrikus titkosítással dolgozik.

Aztán a végén az IETF hagyta az egészet a fenébe és beemelte az SSL3,0-át TLS1.0 néven az RFC-k közé.

Részletesebb leírás:
http://www.javvin.com/protocolHTTPS.html

3.3.2 FTP

Tudom, hogy nem ez a legerőteljesebb FTP szerver implementáció, de nézzünk meg egy korai IIS szervert. Milyen autentikációt engedett az FTP-nek? A Digest és az Integrated Windows autentikációk szóba sem jöhettek.
Maradt az Anonymous, illetve a Basic.

De minek.

És igazából máshol sem volt jobb a helyzet. Az FTP vagy nem kért autentikációt, vagy plain text-ben küldte át a jelszót.

3.3.2.1 FTPS

Nyilván az FTP is az elsők között volt, amelyik ment a HTTP után az SSL csatornába.

Méghozzá rögtön meg is cifrázta a lépéseit, ugyanis két különböző módon képes SSL csatornát építeni:

EXPLICIT MÓD

> RFC 2228, 4217

Ebben az esetben a kliens határozottan javasolja a szervernek, hogy kezdjenek el beszélgetni az autentikációs lehetőségekről. Ezt az AUTH paranccsal jelzi. A parancsnak paramétere a csatorna típusa (SSL, illetve TLS). Ha a kliens nem találja el, mit tud a szerver, akkor egy AUTH 504-es kódot kap vissza. Hogy elkerülje ezt a frusztrációt, a kliens kezdheti egy FEAT paranccsal, amelyre a szerver megmondja, melyiket ismeri.
Utána pedig az ismert módon nekiállnak titkosított csatornát építeni.

Ebben a módban a kliensnek lehetősége van kapcsolgatni. Ha be akarja kapcsolni a titkosítást a command csatornára, akkor azt az AUTH TLS v. AUTH SSL paranccsal teheti meg. A kikapcsolás a CCC paranccsal történi. (Clear Control Channel.) Az adatcsatorna titkosításának bekapcsolása a PROT paranccsal történik, kikapcsolása pedig a CDC paranccsal.

IMPLICIT MÓD

- Semmi cicó, nálam van a slukker - mondja a kliens. Azaz senki nem tárgyal semmit, ha a kliens beszól egy Client Hello-val, akkor a következő pillanatban már falazzák is a csatornát.

3.10. ÁBRA KAPCSOLÓDÁS IMPLICIT FTPS SZERVERHEZ

Láthatod, még semmi sem történt... de a kliensem már vadul építette a csatornát. Ahol a logban zöld lesz a felirat, onnan már titkosított csatornában folyik a kommunikáció. (Nem is tettem be képet a capture fájlból. Nem látni semmit.)
Variálni maximum a titkosítás szintjében lehet (PROT P), de kikapcsolni, azt nem. Az egész session titkosítva lesz.

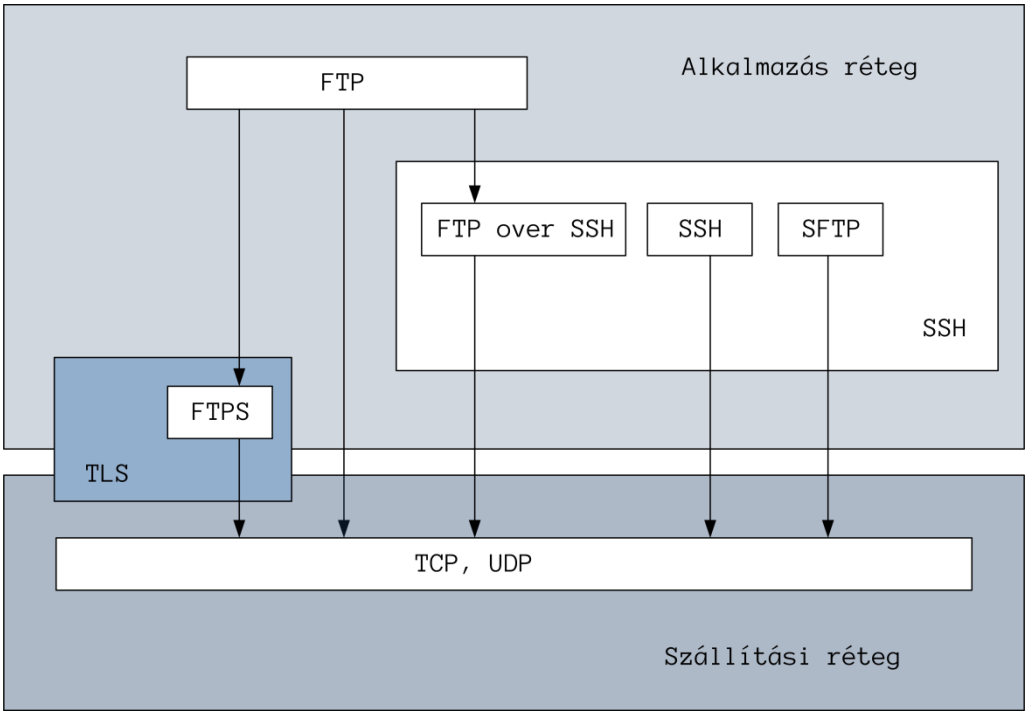
Említsünk meg egy apró problémát - mely mindkét módot egyaránt érinti. A tűzfal. Pontosabban az FTP protokoll proxy. Ez ugye csak akkor tud működni, ha a command csatornából ki tudja olvasni, hogy a szerver melyik portját fogja felajánlani a kliensnek data csatorna létesítése céljából. (Mivel tűzfalazunk, beszéljünk eleve csak a passzív FTP-ről.) Ha TLS/SSL csatornába zavarom a kommunikációt, akkor mit fog tudni kiolvasni a tűzfal? Semmit.
Ezt a problémát lehet úgy körbedolgozni, hogy erősen lekorlátozzuk az FTP szerveren a szóbajöhető portokat, majd a tűzfalon engedélyezzük ezt a már nem túl nagy számú porthalmazt.

3.3.2.2 FTP OVER SSH

Írtam, hogy az SSH az egy önálló, zárt világ. Nem kooperál más protokollokkal. Egyetlen kivételre hajlandó csak: az FTP-t valamiért nagyon szereti.
Az SSH-ban meg lehet oldani, hogy az FTP forgalom teljes egészében - beleértve a command és data csatornák forgalmait - SSH-ba csomagolva utazzon. Ezt nevezik FTP over SSH-nak. (Lásd a legördülő menüt: *3.10. ábra Kapcsolódás Implicit FTPS szerverhez*)

3.3.2.3 A BÁJOS FÉLREÉRTÉSEK FORRÁSA - SFTP

Az FTP over SSH nem keverendő össze az SFTP protokollal (SSH File Transfer Protocol), mely csak nevében hasonlít az FTP-re. Igazából ez az SCP utódja: RCP -> SCP -> SFTP. Azaz az SFTP része az SSH-nak, annak gyakorlatilag a fájltovábbító, fájlmenedzselő alkalmazása. (Ezt szokták összekeverni a Simple File Transfer Protocol-lal, mivel ugyanaz a rövidítésük. Az élet nem habostorta.)



3.11. ÁBRA MINDENFÉLE FTP-K

Egy igazi ingyenes FTP szerver, mely ismeri mind az FTPS-t, mind az SFTP-t.
http://filezilla-project.org/client_features.php

3.3.3 A TÖBBIEK, AZAZ A STARTTLS

Habár az SSL/TLS olyan protokoll, melyen keresztül bármilyen más protokollt át lehet vezetni, de azért ez nem olyan egyszerű. Láthattad eddig is, hogy bizony bele kell nyúlni az eredeti protokollba. Minimum annyira, hogy legyen benne egy olyan parancs, amelyik elindítja a TLS csatorna építését.

Ez a STARTTLS.

Gyakorlatilag ezt a bővítést használja az SMTP, a POP3, az IMAP4 és az NNTP is.

3.12. ÁBRA STARTTLS

Láthatod, a parancs neve szerepel az EHLO parancsra visszaadott listában. Ha beírom, akkor a szerver a 220-as kóddal jelzi, hogy fel van készülve a csatornaépítésre, kezdhetem az ásást.

Erre már nem is érdemes több szót vesztegetni.

Ha már a levelezésnél járunk, beszéljünk még egy másik fajta védelemről is: nem csak teljes forgalmakat lehet csatornába terelni, a legtöbbször megoldható egyes levelek titkosítása, illetve digitális aláírása is. Ezekre több módszer is létezik, tényleg csak felsorolásként kettő: S/MIME, PGP/GPG.

Hogyan történhet egy levél titkosítása? Használhatunk szimmetrikus kulcsot? Persze, ha már van belőle egy példány a túloldalon. Hogyan juthat át? Motoros futárral? Nem igazán biztonságos. Asszimetrikus kulccsal titkosított csomagban? Dehát az a TLS.

Ergo nem tudunk szimmetrikus kulcsokat biztonságosan használni. Marad az, hogy a teljes titkosítás/aláírás móka asszimetrikus titkosítással, azaz kulcspárokkal történik.

Nem akarok ebbe túlzottan belemenni, itt csak pár mondatban összefoglalom a lényeget:

LEVÉL ALÁÍRÁSA

Előveszem a privát kulcsomat. Ezzel betitkosítom a küldendő levelemet - illetve a gyakorlatban csak egy kis darabját. Mivel a publikus kulcsomat az egész világ ismeri, így bárki ki tudja nyitni a levelet. Azaz nem titkos. Viszont ha az én publikus kulcsommal tudták kinyitni, akkor az csakis az én privát kulcsommal lehetett lezárva - ergo hiteles. Azaz aláírt.

LEVÉL TITKOSÍTÁSA

Első körben megszerzem *_a címzett_* publikus kulcsát. (Tanúsítvány.) Ha megvan, akkor ezzel betitkosítom a küldendő levelet. Ki tudja elolvasni? Bárki, akinél ott van a címzett privát kulcsa. Jó esetben ez egyedül csak a címzett.

(Érted már, miért szoktak jót derülni az informatikusok azon, amikor bejön Vezérigazgató István és közli, hogy 10 perc múlva titkosított levelet akar küldeni az Arrogáns zRT-nek? Ha ugyanis a címzettnek nincs tanúsítványa, a fejünk tetejére is állhatunk, akkor sem tudunk neki titkosított levelet küldeni.)

3.4 IPSec

Igen bajban vagyok, ha valahogy definiálnom kell, mi is az az IPSec. Talán még az hangzik a legpontosabban, ha azt mondom, hogy az IPSec az egy nagy köteg IETF szabványgyűjtemény. Olyan szabványokból áll, melyek hálózati forgalom titkosítására vonatkoznak. Pontosabban nem is a titkosítás kifejezés a leginkább megfelelő - maradjunk annyiban, hogy a hálózati csomagokat birizgálja. Úgy, hogy azoknak jól essen.

Még mindig pontosítanom kell. Mi az, hogy hálózati csomagokat? Az SSL szintén gyönyörűen titkosít.
A különbség az, hogy az SSL az alkalmazásból kijövő csomagot. azaz a szállítási réteg payloadját titkosítja. Az IPSec viszont a szállítási rétegből kijövő csomagot, azaz az IP réteg payloadját pisztergálja. Egy szinttel lejjebb ténykedik.

Ezért hívják ***_IP_***Sec-nek.

És akkor rakjuk már rendbe ezt a titkosít/birizgál dolgot is. A levelezésnél már láthattuk, hogy kulcspárokkal kétféleképpen játszhatunk:

- Aláírtunk. Ekkor mindenki el tudta olvasni a levelet - de maga az a tény, hogy el tudták olvasni, igazolta, hogy mi és csak mi küldhettük a levelet és annak tartalma nem változott meg.
- Titkosítottunk. Ekkor csak a kiválasztottak tudták elolvasni a levelet.
- Illetve a kettő együtt.

Az IPSec esetében - céljait tekintve - hasonló a felállás. (Technológiailag nem.)

- Aláírunk : Authentication Header, azaz AH.
- Titkosítunk : Encapsulating Security Payload, azaz ESP

Még mindig csak vadul definiálgatunk. Ugyanis mindkét trükköt helyből kétféleképpen is be tudjuk mutatni:

- Transport mód: Két node között biztosít biztonságos/autentikált kapcsolatot. Tipikusan cégen - azaz biztonságosabbnak tekintett hálózaton - belül használják.
- Tunnel mód: Általában két router közötti csatorna kiépítésére használják, megbízhatatlan közegben. (Mint pl. az internet.)

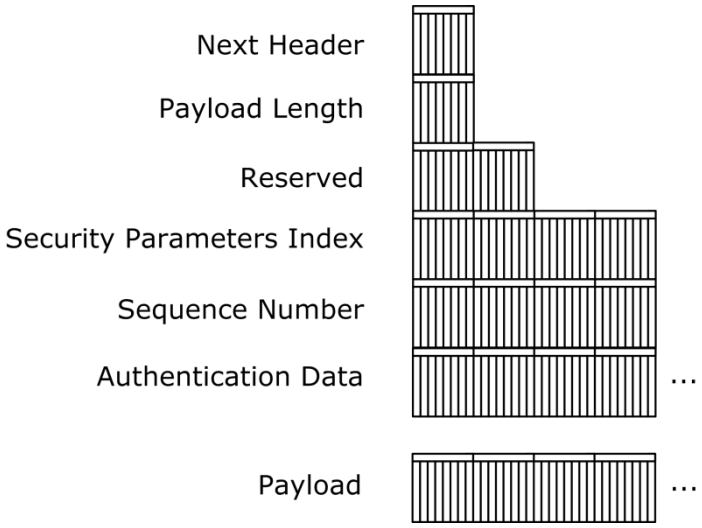
Ez annyi, mint négy IPSec aleset.

Húzzunk bele.

3.4.1 AUTHENTICATION HEADER

RFC 4302

Az autentikációs fejléc gyakorlatilag azt a trükköt tudja, hogy az IP datagram bizonyos részeiből és magának az autentikációs fejlécnek egyes részeiből készít egy tál darált húst: azaz a mezők tartalmát keresztülpasszírozza egy OWF-en. A Windows Server 2008 / Vista esetében ez vagy HMAC-MD5, vagy HMAC- SHA1 hash gyártást jelent.



3.13. ÁBRA AUTHENTICATION HEADER

NEXT HEADER: Spoilerezek egyet: az AH egy - valamilyen - IP fejléc és egy IP payload közé furakszik be. Emiatt elromlik az IP fejlécben a PROTOCOL mező értéke - hiszen a következő blokk nem a payload, hanem az AH. Ergo az IP fejlécben lévő PROTOCOL mező értéke 51 lesz (az AH kódja), az autentikációs fejlécben lévő NEXT HEADER mező értéke pedig a következő blokkra utaló kód. (Ábrán érthetőbb lesz.)

PAYLOAD LENGTH: Egy bájt, az AH bájtokban mért értéke.

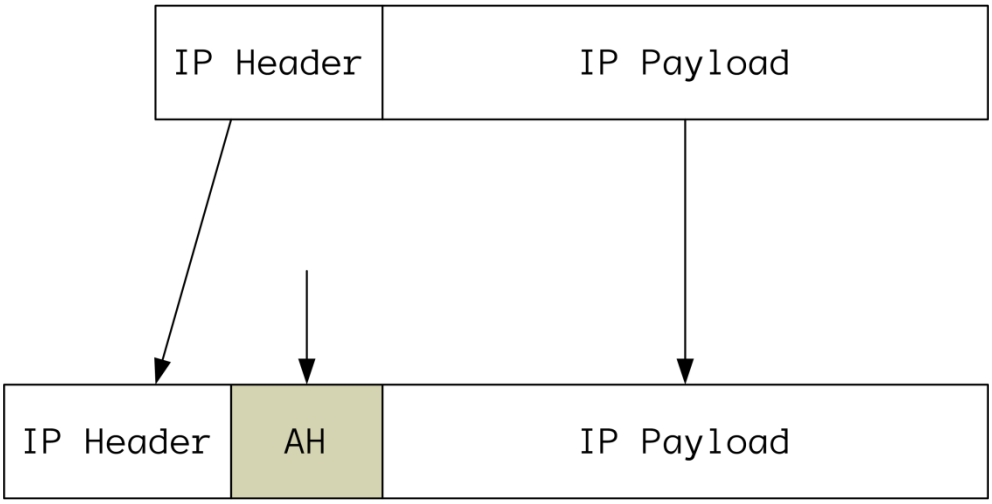
RESERVED: Nulla. Majd valamikor használjuk valamire.

SECURITY PARAMETERS INDEX: Egy kód, mely az IP fejlécben lévő Destination IP címmel együtt az SA-t (Security Association, lsd. később) azonosítja.

SEQUENCE NUMBER: Sorszám. Fontos, hogy bele van véve a darálásba, így a csomagfolyamot nem lehet átvágni visszajátszásos technikákkal.

AUTHENTICATION DATA: Maga a lényeg. A blokk mérete bájtban mérve néggyel osztható kell legyen, ha nem úgy jön ki, akkor kipótolják. Ebben a mezőben utazik maga a darálék, azaz a hash.

3.4.1.1 AH TRANSPORT MÓD



3.14. ÁBRA AH TRANSPORT MÓD

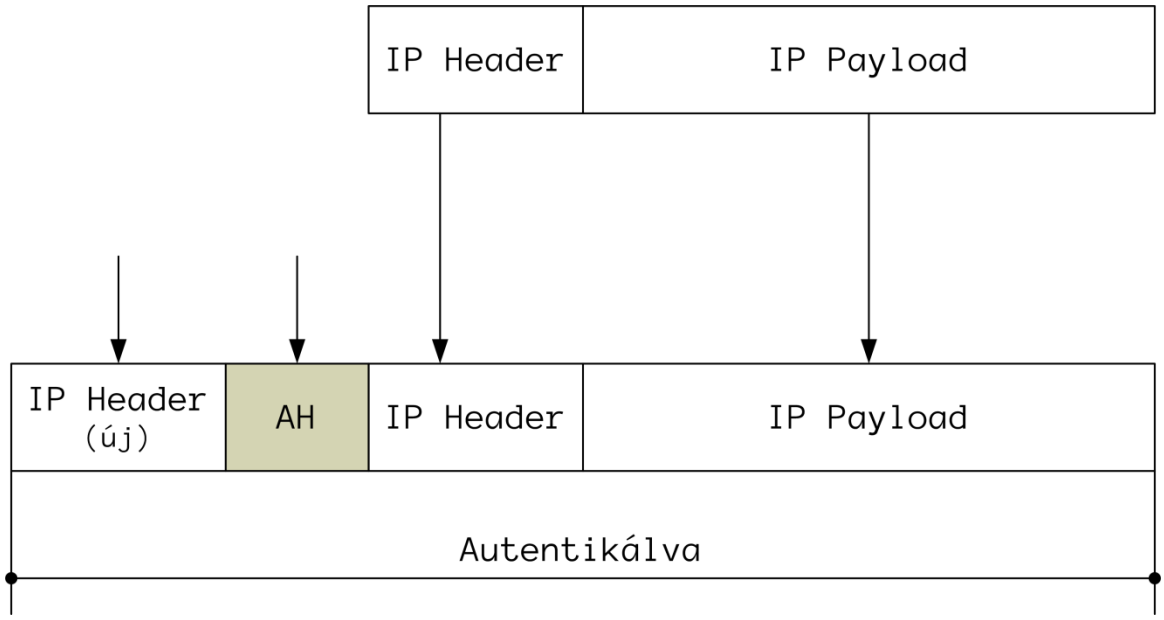
A legegyszerűbb módszer. Az IP fejléc és a payload közé csusszan be az autentikációs fejléc. Logikusan az IP fejléc Protocol mezőjébe 51 kerül, az AH Next Header mezőjébe pedig a payload tartalma, mondjuk 17, azaz UDP.

Nézzük, hogyan keletkezik az ICV. (Az ICV egy újabb neve a darált húsnak. Jelen esetben az Integrity Check Value kifejezést takarja.)

- Az IP fejlécből beledolgoznak minden mezőt, mely nem változik továbbítás közben. Amelyek változnak (TOS, Flags, Fragment Offset, TTL, Header Checksum), azoknak az értéke nulla lesz a számolás során.
- Belekerül maga az AH is, bár az Authentication Data mező helyett 0 szerepel. (Egyébként az örökkévalóságig darálnánk.)
- Végül beledobjuk a teljes IP payload blokkot.

Azaz az AH igazolja, hogy sem az IP payload, sem az IP fejléc üzemszerűen nem változó mezői és legfőképpen maga az igazoló blokk értékei nem változtak meg szállítás közben, a cucc bitről-bitre ugyanaz, mint amit feladtak.

3.4.1.2 AH TUNNEL MODE



3.15. ÁBRA AH TUNNEL MÓD

Az AH megint furakodik, de most az IP fejléc megmakacsolta magát. Összezártak a payload-dal. Az AH így a csomag elejére került - de mivel ő azért nem egy IP fejléc, így a csomag elé oda kellett bigyeszteni egy igazit.
Jelen esetben az eredeti IP datagram nem változott semmit. Az új IP fejléc Protocol mezőjébe kerül bele az 51-es kód, az AH Next Header értéke meg a mögötte jövő IP datagramra mutat.

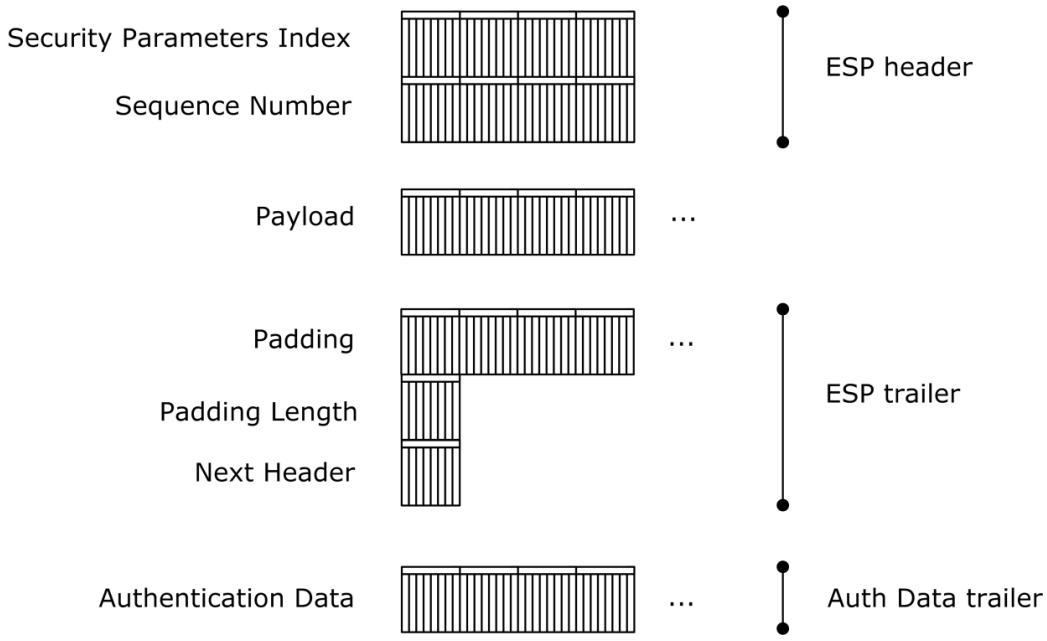
Töltelék recept:

- A külső IP fejlécből bekerül minden mező, melyek értéke üzemszerűen nem változik.
- Bekerülnek az AH mező értékei, kivéve persze önmaga, az Authentication Data.
- Végül beledobjuk a teljes, eredeti IP datagramot.

Fontos megérteni a különbséget a Tunnell és a Transport mód között. A Tunnelnél az eredeti IP fejléc - azaz az eredeti feladó és címzett IP címei - bent maradnak a belső IP fejlécben. A külső IP fejlécbe a tunell két végpontját biztosító hídfőállások IP címei kerülnek. Ahogy megérkezik a csomag a bejárati kapuhoz, kap egy AH-t és egy új fejlécet, melyben cél címként a kijárati kapu IP címe lesz. A kijárati kapunál meg leszedik róla az új fejlécet, az AH-t - és az eredeti IP csomag mehet tovább az útján.
Transport mód esetén ilyen variálás nincs, a végső címzett ugyanaz a host, akinél az IPSec is végződik.

3.4.2 ENCAPSULATING SECURITY PAYLOAD

RFC 4303



3.16. ÁBRA ENCAPSULATING SECURITY PAYLOAD

SECURITY PARAMETERS INDEX: Mint korábban.

SEQUENCE NUMBER: Sorszám.

PADDING: A titkosított résznek szintén akkorának kell lennie, hogy a bájtban számolt mérete néggyel osztható legyen. Ezért toldozunk, ha kell.

PADDING LENGTH: Mennyivel toldottunk.

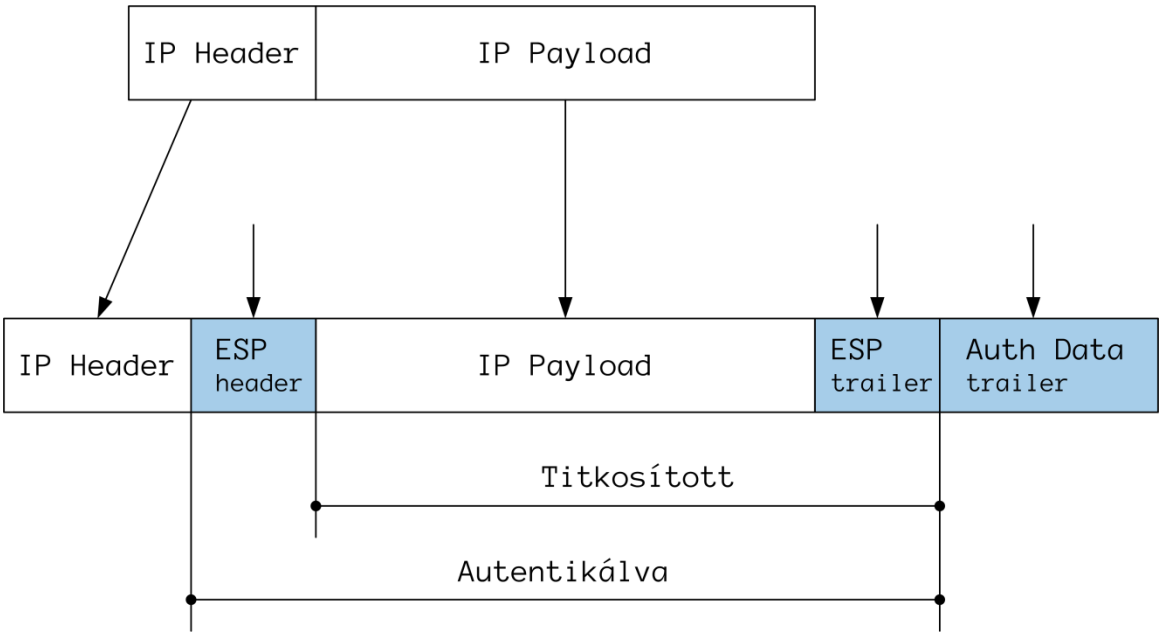
NEXT HEADER: Szintén, mint korábban.

AUTHENTICATION DATA: Teljesen azonos, ugyanúgy az ICV-t tartalmazza.

Ejtsünk pár szót arról, milyen titkosítások jöhetnek szóba Windows Server 2008, illetve Vista alatt:

- Advanced Encryption Standard 128 bites kulccsal (AES-128)
- AES 192
- AES-256
- Triple Data Encryption (3DES) 56 bites kulccsal
- Data Encryption Standard (DES) 56 bites kulccsal. (Broáf.)

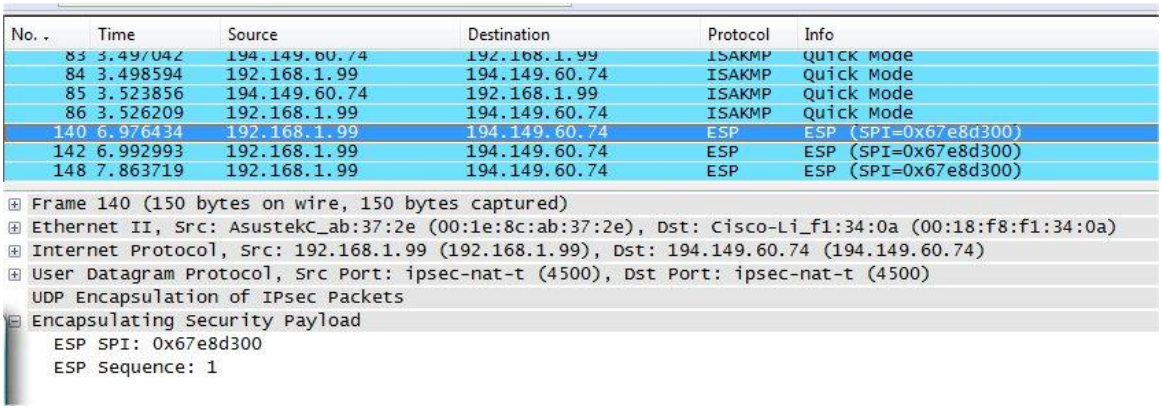
3.4.2.1 ESP TRANSPORT MÓD



3.17. ÁBRA ESP TRANSPORT MÓD

Látható, itt egy egész siserehad vetődött rá az IP datagramra. Az ESP fejléc befurakodott az IP fejléc és az IP payload közé. Emiatt az IP fejlécben lévő Protocol mező tartalma 50-es értékre (ESP) váltott. A csomag végére pedig bejött az ESP trailer, illetve egy Authentication Data blokk.

A titkosítás módjára utaló információk az ESP headerben utaznak. Azaz eddig a blokkig nem lehet titkosítani, utána viszont már igen. A teljes IP payload és az ESP trailer is le van kódolva.



3.18. ÁBRA ESP CSOMAG

Láthatod, a Wireshark is csak az ESP headert tudja mutatni: SPI és sorszám.

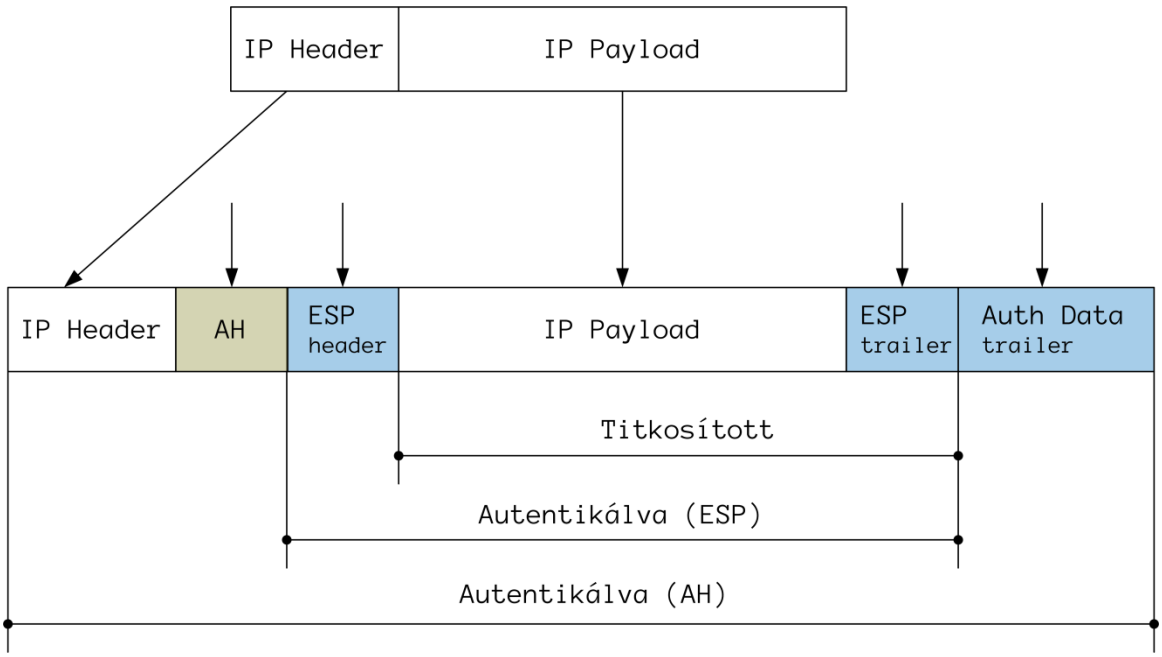
Az ICV a következőkből áll össze:

- A teljes ESP header
- A teljes IP payload
- Az ESP trailer, de az Authentication Data mező már nem.

Nézzük meg, ezzel mit csináltunk. Egyrészt igazoltuk, hogy a payload, illetve a titkosításhoz használt mezők garantáltan nem lettek módosítva. Másrészt a payload, azaz a hasznos teher, titkosítva ment át a dróton.

Viszont az is tény, hogy semmilyen módon nem védtük meg az eredeti IP fejlécet. Még annyira sem, mint az AH Transport módban.

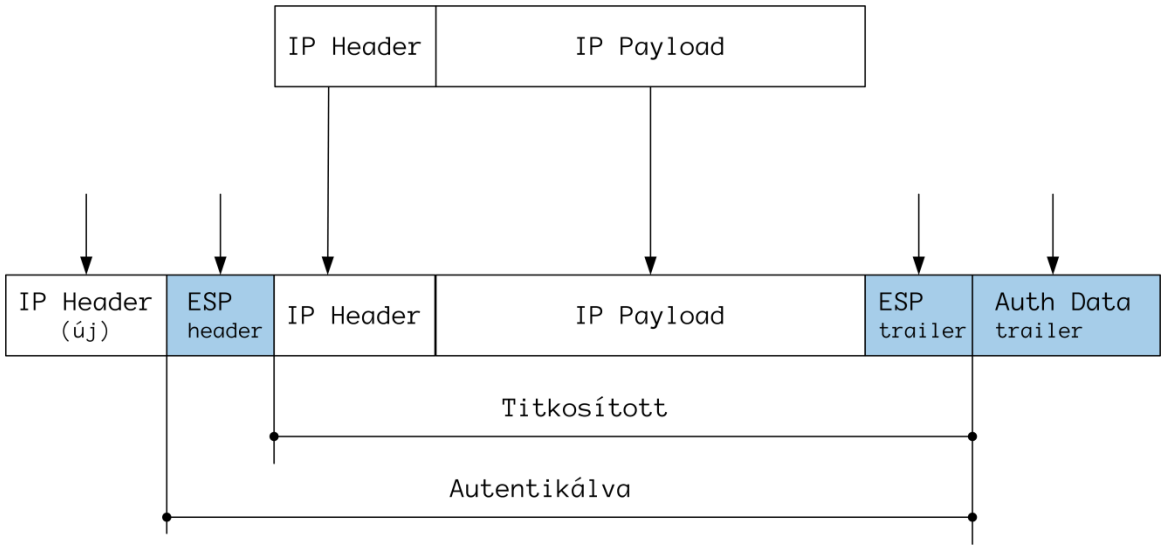
Emiatt szokták a két Transport módszert kombinálni.



3.19. ÁBRA AH ÉS ESP TRANSPORT MÓDBAN

Látható, hogy az előző ábrához képest annyi a különbség, hogy bejött egy plusz autentikációs fejléc. Azaz az ESP továbbra is autentikálja és titkosítja a saját kis szemétdombját, de az egész meg lett még fejelve egy AH blokkal.
Nyilván ebben a körben az IP fejléc PROTOCOL mezőjébe az 51-es érték kerül, az AH blokk NEXT HEADER mezőjébe 50-es, és csak az ESP Trailer NEXT HEADER mezője fogja mutatni a Payload tartalmát.
Az AH szempontjából az ESP csomag az IP payload - azaz ennek függvényében készíti el az AH Transport módnál már ismertetett ICV-t.

3.4.2.2 ESP TUNNEL MÓD



3.20. ÁBRA ESP TUNNEL MÓD

Na, itt aztán védjük ezerrel az eredeti IP fejlécet.

Látható, Tunnel módban ugyanaz a lényeg: ahogy az IP datagram megérkezett a csatorna bejáratához, teljesen ugyanúgy kell ki is mennie a kijáraton. Emiatt az eredeti IP datagram titkosítva van, autentikálva van, sőt, autentikálva vannak a titkosításhoz szükséges mezők is.

(Jelzem, ez a módszer sem védi meg az új fejlécet a módosítástól.)

3.4.3 SECURITY ASSOCIATIONS

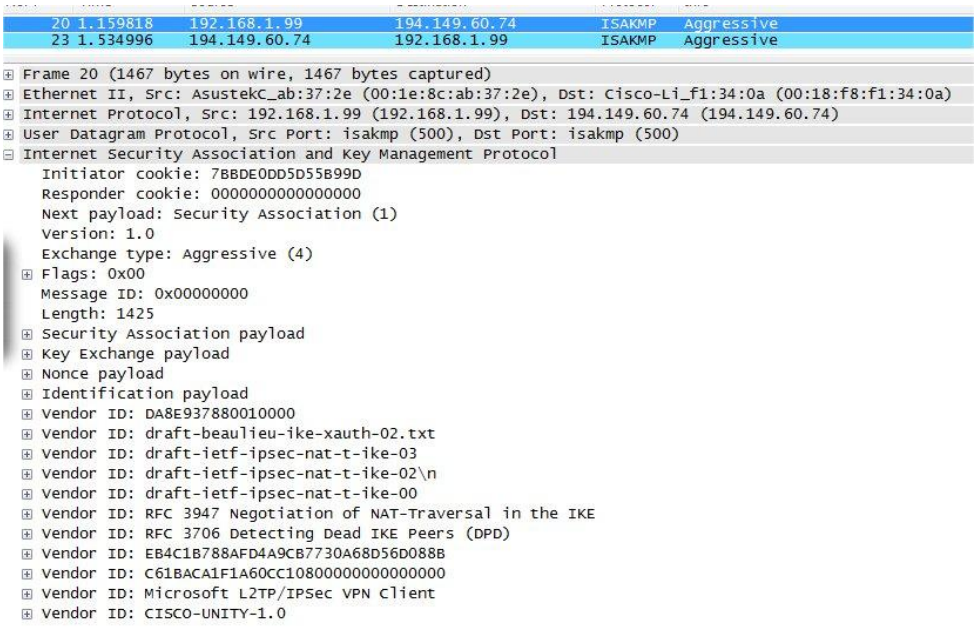
Egy újabb köteg szabvány következik. Ahhoz ugyanis, hogy két host beszélni tudjon egymással ipszekül, ahhoz le kell tisztázniuk, melyik az a nyelvjárás, melyet mindketten ismernek. Ha megtalálták, akkor az így kialakult kapcsolatot - melyben egyaránt vannak biztonsági szolgáltatások, különböző védelmi mechanizmusok és kölcsönös kulcscserélési algoritmusok - nevezzük biztonsági szövetkezéseknek, ángliusul Security Associations-nek. Az IPSec kommunikációban alapvetően két SA alakul ki: először az Internet Security Association and Key Management Protocol (ISAKMP) SA, mely kialakítja azt a biztonságos csatornát, ahol már beszélgethetnek a hostok a finomabb részletekről. Utána jön a második SA, az IPsec SA, aholis a korábban említett finomabb részletekben egyeznek meg a felek. Ez után indulhat csak be a forgalom, a korábban említett technikák valamelyikével.

3.4.3.1 MAIN MODE (ISAKMP SA)

RFC 2408

Alapvetően az SA kialakítása három lépésben történik:

- A felek - plain textben - megbeszélik, milyen védekező mechanizmusokat ismernek.
- Még mindig plain textben nekiállnak kulcsokat cserélni.
- Elkezdik egymást autentikálni. (Na, ez már titkosítva megy.)



3.21. ÁBRA ISAKMP SA KIALAKÍTÁSA, PLAIN TEXT

No.	Time	Source	Destination	Protocol	Info
24	1.557430	192.168.1.99	194.149.60.74	ISAKMP	Aggressive
25	1.581501	194.149.60.74	192.168.1.99	ISAKMP	Transaction (Config Mode)
26	1.582051	192.168.1.99	194.149.60.74	ISAKMP	Transaction (Config Mode)
28	1.982365	194.149.60.74	192.168.1.99	ISAKMP	Transaction (Config Mode)
29	1.982845	192.168.1.99	194.149.60.74	ISAKMP	Transaction (Config Mode)
35	3.150534	192.168.1.99	194.149.60.74	ISAKMP	Transaction (Config Mode)
36	3.174782	194.149.60.74	192.168.1.99	ISAKMP	Transaction (Config Mode)
37	3.177537	192.168.1.99	194.149.60.74	ISAKMP	Quick Mode
38	3.228274	194.149.60.74	192.168.1.99	ISAKMP	Quick Mode
39	3.231307	192.168.1.99	194.149.60.74	ISAKMP	Quick Mode
40	3.280526	192.168.1.99	194.149.60.74	ISAKMP	Quick Mode
41	3.281293	192.168.1.99	194.149.60.74	ISAKMP	Quick Mode
42	3.281598	192.168.1.99	194.149.60.74	ISAKMP	Quick Mode
43	3.281898	192.168.1.99	194.149.60.74	ISAKMP	Quick Mode
44	3.282151	192.168.1.99	194.149.60.74	ISAKMP	Quick Mode
45	3.283923	192.168.1.99	194.149.60.74	ISAKMP	Quick Mode
47	3.284617	192.168.1.99	194.149.60.74	ISAKMP	Quick Mode
48	3.284941	192.168.1.99	194.149.60.74	ISAKMP	Quick Mode
49	3.285080	192.168.1.99	194.149.60.74	ISAKMP	Quick Mode
50	3.366420	194.149.60.74	192.168.1.99	ISAKMP	Quick Mode
54	3.367507	194.149.60.74	192.168.1.99	ISAKMP	Quick Mode
62	3.382262	194.149.60.74	192.168.1.99	ISAKMP	Quick Mode
67	3.444728	194.149.60.74	192.168.1.99	ISAKMP	Quick Mode
68	3.444849	194.149.60.74	192.168.1.99	ISAKMP	Quick Mode
69	3.448309	192.168.1.99	194.149.60.74	ISAKMP	Quick Mode
70	3.448423	192.168.1.99	194.149.60.74	ISAKMP	Quick Mode
71	3.448518	192.168.1.99	194.149.60.74	ISAKMP	Quick Mode
72	3.448578	194.149.60.74	192.168.1.99	ISAKMP	Quick Mode
73	3.454119	192.168.1.99	194.149.60.74	ISAKMP	Quick Mode
74	3.455713	192.168.1.99	194.149.60.74	ISAKMP	Quick Mode
76	3.456408	192.168.1.99	194.149.60.74	ISAKMP	Quick Mode
79	3.492813	194.149.60.74	192.168.1.99	ISAKMP	Quick Mode
81	3.496837	192.168.1.99	194.149.60.74	ISAKMP	Quick Mode
83	3.497042	194.149.60.74	192.168.1.99	ISAKMP	Quick Mode
84	3.498594	192.168.1.99	194.149.60.74	ISAKMP	Quick Mode
85	3.523856	194.149.60.74	192.168.1.99	ISAKMP	Quick Mode
86	3.526209	192.168.1.99	194.149.60.74	ISAKMP	Quick Mode
140	6.976434	192.168.1.99	194.149.60.74	ESP	ESP (SPI=0x67e8d300)
142	6.992993	192.168.1.99	194.149.60.74	ESP	ESP (SPI=0x67e8d300)

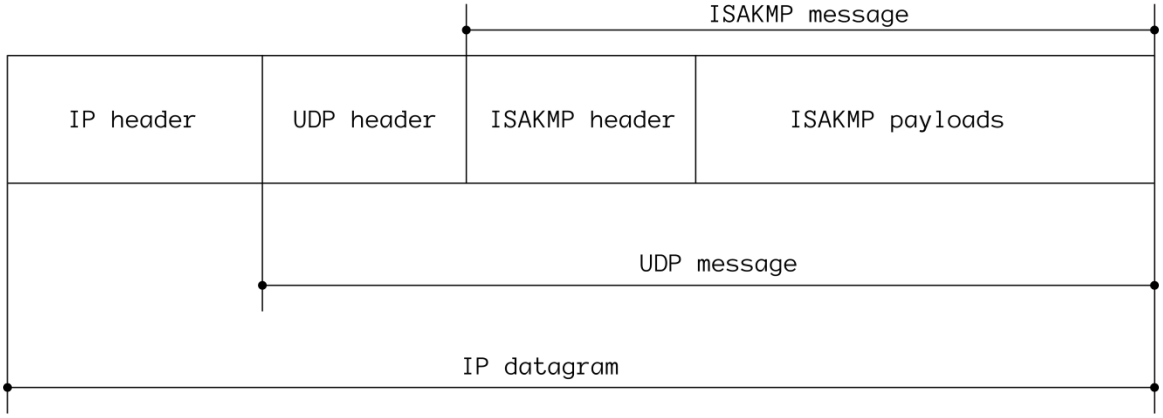
```
⊞ Frame 37 (882 bytes on wire, 882 bytes captured)
⊞ Ethernet II, Src: AsustekC_ab:37:2e (00:1e:8c:ab:37:2e), Dst: Cisco-Li_f1:34:0a (00:18:f8:f1:34:0a)
⊞ Internet Protocol, Src: 192.168.1.99 (192.168.1.99), Dst: 194.149.60.74 (194.149.60.74)
⊞ User Datagram Protocol, Src Port: ipsec-nat-t (4500), Dst Port: ipsec-nat-t (4500)
⊞ UDP Encapsulation of IPsec Packets
⊟ Internet Security Association and Key Management Protocol
    Initiator cookie: 7BBDE0DD5D55B99D
    Responder cookie: A16251353FF596F2
    Next payload: Hash (8)
    Version: 1.0
    Exchange type: Quick Mode (32)
  ⊞ Flags: 0x01
    Message ID: 0x53589fc4
    Length: 836
    Encrypted payload (808 bytes)
```

3.22. ÁBRA ISAKMP KIALAKÍTÁSA, MÁR TITKOSÍTVA

Jól látható maga az SA-k kialakítása: elindul a plain text kommunikáció, majd áttérnek titkosítottra (ekkor már csak annyit látunk a payload-ból, hogy Encrypted payload), végül elkészültünk, jöhet maga az ESP.

Az ábrákról kiolvashatunk még egy fontos információt: az ISAKMP, legalábbis amíg nem titkosított, addig UDP-t használ, méghozzá az 500-as porton - illetve NAT esetén a 4500-as porton.

Még egy megjegyzés: az ábrákon is látható Aggressive mód kifejezés alatt nagyjából hasonlót kell érteni, mint a Main módnál. A tartalma ugyanaz, csak a Main mód esetében a felek identitása védve marad.



3.23. ÁBRA AZ ISAKMP CSOMAG ELHELYEZKEDÉSE

Remélem, senkit nem lepett meg, hogy az ISAKMP csomag is fejlécből és payload-ból áll.

Ellenben most az egyszer nem mennék bele a fejléc élveboncolásába. Valahol abba kell hagyni, mert a végén még az ember szenvedélyévé válik.
Az ábrán (*3.21. ábra ISAKMP SA kialakítása, plain text*) látható egy kifejlett példány.

Piszkáljuk inkább a payload-ot. Ugyanezen az ábrán látható belőlük is egy jó nagy kupac. Igen, egy csomagban nem csak egyféle lehet. És hogy még bonyolultabb legyen a helyzet, mindegyik payload tipusban vannak speciális mezők. Az - egyébként az ISAKMP fejlécben is megtalálható - Next Payload mező értéke mondja meg, hogy milyen payload tipus jön a láncban.

```
⊟ Internet Security Association and Key Management Protoco
    Initiator cookie: 7BBDE0DD5D55B99D
    Responder cookie: A16251353FF596F2
    Next payload: Security Association (1)
    Version: 1.0
    Exchange type: Aggressive (4)
  ⊞ Flags: 0x80
    Message ID: 0x00000000
    Length: 444
  ⊟ Security Association payload
      Next payload: Key Exchange (4)
      Payload length: 52
      Domain of interpretation: IPSEC (1)
      Situation: IDENTITY (1)
    ⊞ Proposal payload # 1
  ⊟ Key Exchange payload
      Next payload: Nonce (10)
      Payload length: 132
      Key Exchange Data (128 bytes / 1024 bits)
  ⊞ Nonce payload
  ⊞ Identification payload
  ⊞ Hash payload
  ⊞ Vendor ID: CISCO-UNITY-1.0
  ⊞ Vendor ID: draft-beaulieu-ike-xauth-02.txt
  ⊞ Vendor ID: RFC 3706 Detecting Dead IKE Peers (DPD)
  ⊞ Vendor ID: draft-ietf-ipsec-nat-t-ike-02\n
  ⊞ NAT-D (draft-ietf-ipsec-nat-t-ike-01 to 03) payload
  ⊞ NAT-D (draft-ietf-ipsec-nat-t-ike-01 to 03) payload
  ⊞ Vendor ID: Microsoft L2TP/IPSec VPN Client
  ⊞ Vendor ID: 54A5F6283FF496F2331E2260CABD2A27
  ⊟ Vendor ID: 1F07F70EAA6514D3B0FA96542A500407
      Next payload: NONE (0)
      Payload length: 20
      Vendor ID: 1F07F70EAA6514D3B0FA96542A500407
```

Az oldalsó ábrán lenyitottam pár payload blokkot. Az összeset nem, sőt, a kinyitottaknál sem mentem le a teljes mélységekig - borzasztóan hosszú listát kaptunk volna.

Az ISAKMP esetén tényleg minden fontos dolgot megbeszélnek a felek.

Pusztán csak felsorolás jelleggel, mivel csak egy kicsit nem lehet részletezni az egyes tipusok tartalmát - nagyon pedig nem akarok belemászni.

Payload tipusok:

- SA payload
- Proposal payload
- Transform payload
- Vendor ID payload
- Nonce payload
- Key Exchange payload
- Notification payload
- Delete payload
- Identification payload
- Hash payload
- Certificate Request payload
- Certificate payload
- Signature payload

3.4.3.2 QUICK MODE (IPSEC SA)

Az ISAKMP SA után mindkét fél kezében ott lesz egy eljárásköteg. Melyek azok az eljárások, melyeket mindkét fél ugyanúgy ismer.
Az IPSec SA kialakításakor ezek közül választják ki azokat, amelyeket immár használni is fognak. (Gyakorlatilag az IPSec SA kialakításakor mindkét irányra külön-külön megállapodások jönnek létre.) Mivel itt már tényleg a konkrét eljárásokról van szó, ennek a kommunikációnak muszáj titkosnak lennie.

Bármilyen meglepő, formailag ebben a fázisban is ISAKMP csomagokat használnak a felek. A kommunikáció négy lépésből áll, ezek során a következő tipusú payload-ok közlekednek: SA, Identification, Nonce, Hash és Notification.

3.4.3.3 IKE

RFC 2409

Elérkeztünk ahhoz a részhez, ahol az egyébként boszorkányosan ügyes versenytáncosnak is összegubancolódik a lába.

Az Internet Key Exchange ugyanis megint egy szabványgyűjtemény, mely ráadásul átfedésben van az eddigiekkel. Egész pontosan az IKE az két halmaz egybegyúrása:

- Egyfelől az ISAKMP, mint SA létrehozására szolgáló protokoll,
- Másfelől az Oakley Key Determination Protocol, mely a Diffie-Hellman algoritmust használja kulcsok cseréjére.

Ha visszaemlékszel, akkor az ISAKMP SA létrehozásakor írtam, hogy van egy lépés, amikor még plain textben kulcsokat cserélnek a felek. Nos, az IKE azt tudja, hogy ez a kulcscsere történhet a Diffie-Hellman algoritmussal. (Mert extrém esetben lehet preshared key is.)

Természetesen az IKE célja is SA létrehozása - méghozzá mindkettőé. Van neki Main módja és van Quick módja. Ugyanúgy ISAKMP csomagokat használ, sőt, a Main mód után az SA-t ugyanúgy IPSEC SA-nak nevezik. Értelemszerűen az ISAKMP csomagok itt is az 500-as UDP portot használják.

3.4.3.4 IKE v2

RFC 4306, 4301, 4309

Az eredeti IKE - bár egész jó cucc volt - de azért bírt néhány hiányossággal. Több kisérlet is született a jobbítására, végül ezeket 2005-ben az IETF összefogta egy RFC-be, a 4306-osba. Ez lett az IKEv2.

Mit tud?

3.4.3.4.1 RFC

Eleve kevesebb RFC-t jelent. Mint írtam, az IKE-t nekiálltak toldozgatni, foldozgatni. Ez mind-mind RFC-ket jelentett. Ezzel szemben a 4306 egybefogta az egészet. (Azóta persze ezt is toldozgatják.)

3.4.3.4.2 MOBIKE

Ez egy későbbi kiegészítés az IKEv2-höz, melynek segítségével a bolyongó - mobil - felhasználók is tudnak SA-t létrehozni.

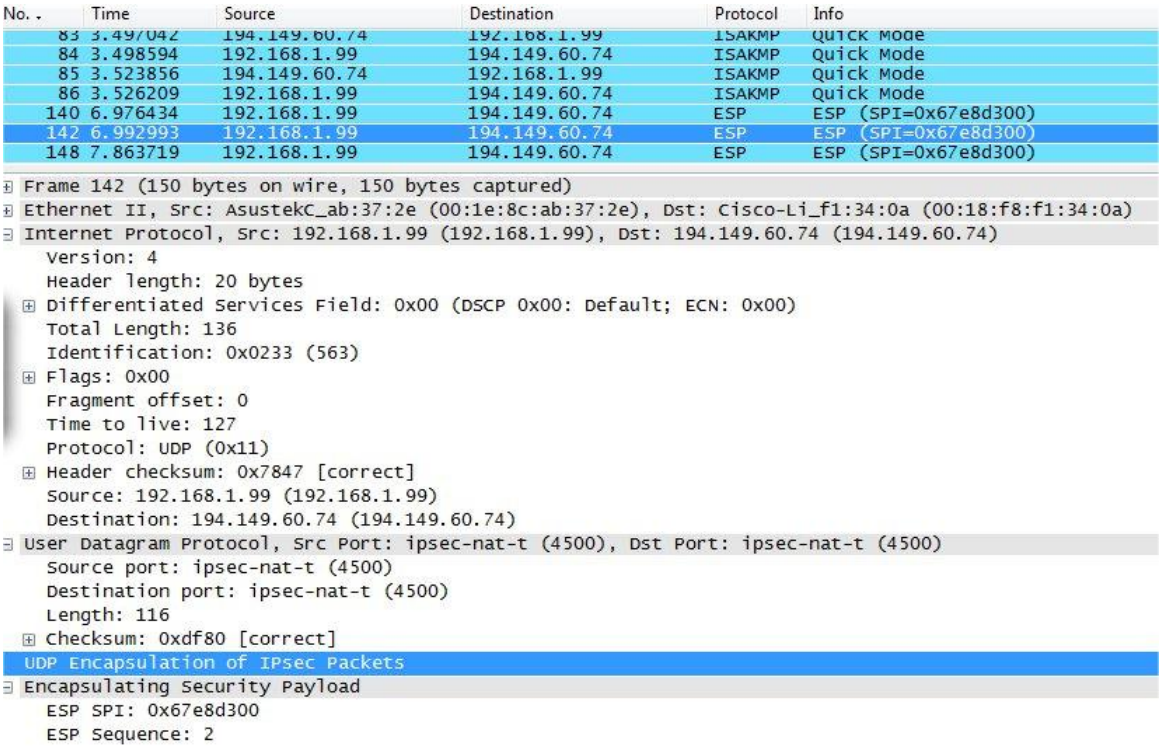
3.4.3.4.3 NAT TRAVERSAL

Ez önmagában is egy érdekes történet.

Gondoljunk bele, mi van akkor, ha egy natoló router mögött ülve akarunk IPSec csatornát kiépíteni? A router a NAT során kapásból kicseréli az IP fejlécben a feladó IP címét. Ha a portokkal is játszik, akkor még a TCP/UDP fejlécben is cseréli a feladó port számát. Márpedig akár az AH, akár az ESP csatornázást nézzük, az aláírás elkészítésében benne van mind az IP fejléc, mind az IP payload, azaz az UDP csomag vagy TCP szegmens.

Akkor ezt most hogy?

Ugyanúgy, ahogy az IPv6-ot csatornáztuk bele IPv4-be, natoló router esetén. Ott Teredo-nak hívták a módszert, itt NAT-T-nek. A lényeg ugyanaz: a teljes forgalmat UDP csomagokba rakjuk (portszám: 4500), és ez már kényelmesen átmegy a NAT/PAT routerrel megerősített hálózaton.



3.24. ÁBRA NAT-T MŰKÖDÉS KÖZBEN

3.4.3.4.4 SCTP

Az IKEv2 már támogatja a VOIP-nál használt SCTP protokollt.

3.4.3.4.5 MEGBÍZHATÓSÁG ÉS ÁLLAPOTMENEDZSMENT

Az IKEv2-ben bevezettek egy, a TCP-hez hasonló technikát: sequence és acknowledge number párok segítségével gondoskodnak a megbízható csomagszállításról. Ennek segítségével felderíthető az impotencia is. Ez alatt azt értem, amikor a kommunikáció úgy akad el, hogy mindkét fél a másikra vár, mert azt hiszik, az van soron. Feloldásukra még nincs szabvány, de áthidaló megoldások - Dead-Peer-Detection - már léteznek.

3.4.3.4.6 DOS ELLENI VÉDELEM

Az IKEv2 már hamar meg tudja állapítani, hogy a másik fél - a requester - létezik-e és tényleg komolyan gondolja-e a csatornázást.

A Windows Server 2008 R2, illetve Windows 7 termékek már ismerik: az IKEv2 a későbbiekben részletezett VPN Reconnect fantázianevű funkció része.

3.4.3.5 AUTHIP

Azaz Authenticated Internet Protocol. Egy olyan kiterjesztése az IKE-nek, mely kifejezetten az autentikációra megy rá:

- Ismeri a felhasználói szintű autentikációt. (Kerberos vagy tanúsítvány)
- Tud kezelni multiple credential autentikációt.
- Van benne asszimetrikus autentikáció.
- És úgy általában, az IKE-nél fejlettebb autentikációs módszereket használ.

Van a módszernek egy hátulütője is: szigorúan csak Microsoft környezetben működik, ott is csak a Vista, illetve az azutáni operációs rendszerekben.
Ergo nem IETF RFC, hanem belső Microsoft megoldás.

Fontos látni, hogy az AuthIP egyáltalán nem kompatibilis az IKEv2-vel. Olyannyira nem, hogy az AuthIP, habár ISAKMP protokollt használ, de teljesen más - egyszerűsített - csomagtipussal.

3.4.4 Összefoglalás

Nem nagyon szoktam fejezeteket összefoglalásokkal zárni, de most kivételt teszek. Egyrészt ez az IPsec téma önmagában elég kaotikus, nem árt röviden áttekinteni. Másrészt a későbbiekben építünk erre a tudásra, tehát jó, ha van egy világos kép - egy váz - arról, mi is ez tulajdonképpen.

Tehát az IPSec az egy szabványgyűjtemény. A célja az, hogy biztonságos csatornát építhessünk ki két kommunikáló fél között. A biztonság azt jelenti, hogy vagy aláírjuk a csomagokat, garantálva ezzel az eredetiségüket, vagy az egész kommunikációt tikosítjuk. Az elsőt hívjuk AH-nak, a másodikat ESP-nek. A kettőt kombinálhatjuk.
Mindkét módszernek létezik Transport, illetve Tunnel módja. Az első inkább a host-ok közötti kommunikációra jellemző, a második pedig inkább a routerek közöttire.

Ahhoz, hogy akár az AH, akár az ESP működni tudjon, a feleknek meg kell egyezniük a közösen ismert eljárásokat illetően. Ezt a megállapodást nevezzük SA-nak.
Az IPSec kommunikációhoz két SA-t kell létrehozni. Az első célja a biztonságos csatorna létrehozása, a másodiké már ezen a csatornán a konkrét egyeztetés.

Az első SA-t nevezik ISAKMP SA-nak, illetve IKE SA-nak. Mindkét esetben az ISAKMP protokollt használjuk, annak a csomagtipusával. Az IKE SA annyival több az ISAKMP SA-nál, hogy kulcscserére az Oakland Key Determination protokollt használja.
Az IKE szabványcsomagnak létezik újabb, erősen feljavított változata, az IKEv2. Emellett léteznek alternatív kiterjesztett IKE megoldások is, ilyen Windows környezetben az AuthIP.

A második SA-t IPSec SA-nak nevezik. A kommunikáció itt is ISAKMP csomagok segítségével történik, de már titkosított csatornán.

Windows környezetben az IPSec működését IPSec házirendekkel szabályozhatjuk.

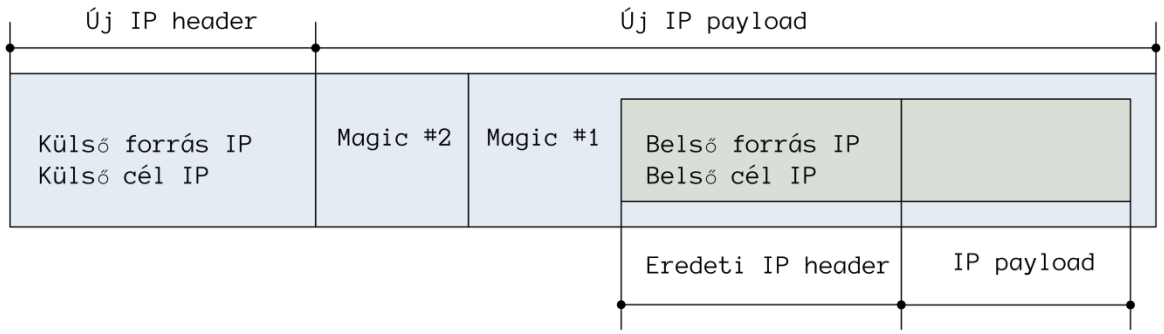
3.5 VPN ÉS TÁRSAI

El sem hiszed, de megint csatornázni fogunk.

Azért ejtsünk előtte pár szót arról, hogy mit is értünk Virtual Private Network, azaz VPN alatt.
Hát, alapvetően azt, amit az angol név is sugall. Van egy védett (private) hálózatunk és van ezen kívül a vad külső világ. A védett hálózatunkban mindenki barát, a social engineering művelőit már a portás főbelövi a bejáratnál, szóval itt tényleg magunk között vagyunk, sokkal lazábbak lehetünk.
De ójaj, embereink időnként kénytelenek kimerészkedni a nagyvilágba. természetesen a laptopjaikkal meg a mindenféle kütyüikkel együtt. Mondtam már, hogy nekünk borzasztó perverz embereink vannak? Képesek arra, hogy egy fárasztó nap után a távoli szállodából is szeretnének a belső hálózat ólmelegében lubickolni egy cseppet. Nem is beszélve azokról az embereinkről, akik akár munkaidőn kívül, akár azon belül a kényelmes otthonukból szeretnének dolgozni.

Nyilván felmerül az igény, hogyan lehetne a belső biztonságos hálózat határait úgy kihúzni, hogy távoli pontokra is elérjen. Úgy, hogy a két pont között a kiszámíthatatlan közeg, az internet jelenti az átviteli közeget.

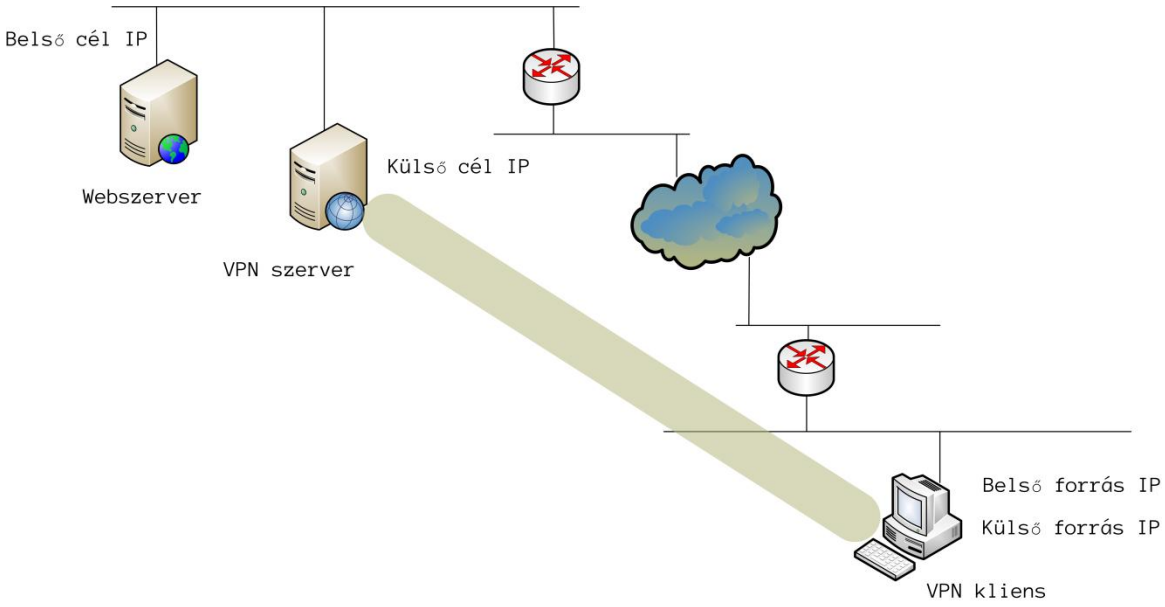
Az eddigi fejezetek alapján nyilván senki nem huppan földre a meglepetéstől, ha azt mondom, hogy igen, itt is valamiféle csatornázások lesznek.



3.25. ÁBRA ÁLTALÁNOS VPN SÉMA

Ez egy teljesen általános séma. Van az eredeti IP csomagunk, ezt valamilyen varázslattal betitkosítjuk. Aztán elképzelhető, hogy rákűldünk még egy varázslatot, majd teszünk elé egy újabb IP fejlécet - és kész az új IP datagram.

Mielőtt belemennénk a részletekbe, vizsgáljuk meg, felépítésük alapján milyen VPN struktúrákat ismerünk.

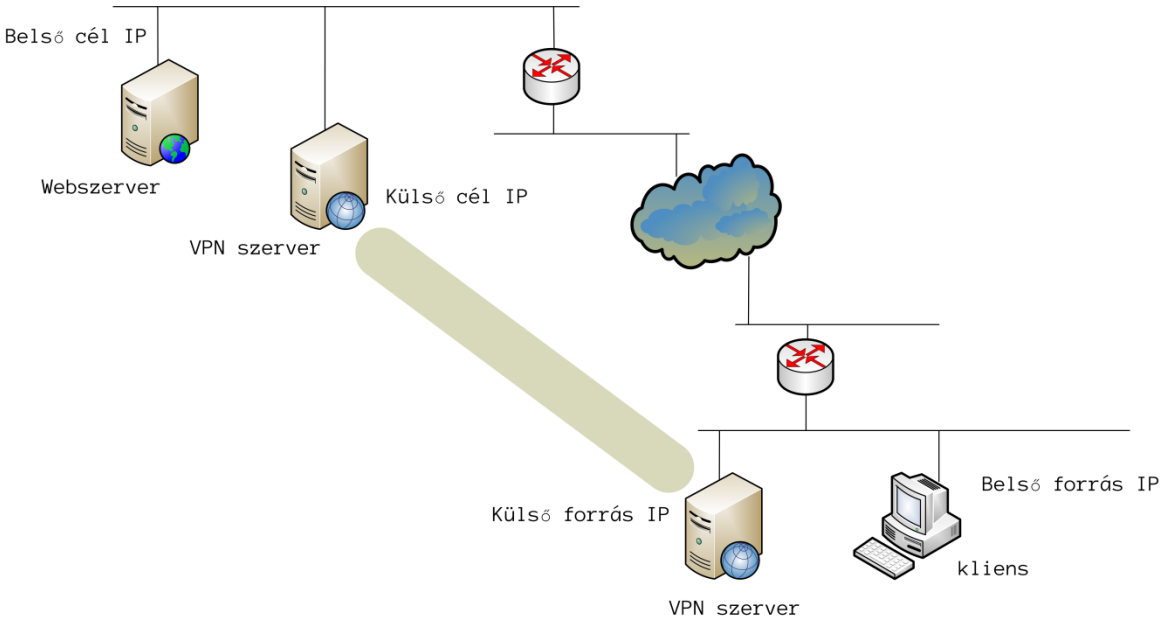


3.26. ÁBRA VPN REMOTE ACCESS

Az első a client-server tipusú, más néven Remote Access VPN. Erről beszéltünk eddig. Bolyong a kliens a külvilágban, majd egy benzinkút büféjéből megpróbál a helyi wifin keresztül rácsatlakozni cégének VPN szerverére. Ha ez sikerül neki, akkor utána úgy fogja érezni magát, mintha bent ülne a munkahelyén. El tud érni bármilyen belső erőforrást, például a képen látható webszervert is.

Az ábráról nem csak a struktúra olvasható le, hanem az IP címek viszonyai is. Az eredeti IP datagramban a webszerver IP címe szerepel megcélzott címként és a kliens IP címe feladóként. A VPN varázslat után viszont a külső IP fejlécben a megcélzott cím a VPN router címe lesz. Az fogja levenni a csomagról a ráküldött varázslatot és továbbítja az immár teljesen hétköznapi csomagot a webszervernek.

Visszafelé ugyanez pepitában, csak éppen ekkor a VPN szerver varázsol és a VPN kliens távolítja el a rontást.



3.27. ÁBRA SITE-TO-SITE VPN

De nem csak ez az egy forgatókönyv létezhet. Mi van akkor, ha nekünk van egy központi telephelyünk és mellette szanaszét az országban 30-40 kicsi telephely? Mindegyikhez bérelt vonali kapcsolatot kiépíteni meglehetős pazarlás, különösen akkor, ha nem létfontosságú a kapcsolat minősége. Ilyenkor jöhet szóba az, hogy leteszünk a telephelyekre egy-egy VPN szervert és úgynevezett site-to-site VPN-eket építünk ki. Ebben az esetben nem kell a kliens gépekre VPN kliens szoftvert telepíteni, viszont lokális alhálózat szinten gondoskodni kell a routolásról: ugyanis ekkor a központ felé nem a helyi router lesz a gateway, hanem a VPN szerver.

Nézzük meg, hogyan alakulnak ilyenkor az IP címek. Amikor a kliens feladja a csomagot, akkor a belső feladó IP cím továbbra is a saját IP címe lesz, a belső cél IP cím pedig a webszerveré. De a külső feladó IP cím már a helyi VPN szerver IP címe lesz, a külső cél IP cím pedig a túloldali VPN szerver IP címe. Nem is lehetne másképpen, hiszen a varázslást nem a kliens, hanem a VPN szerver végzi, ő teszi hozzá a külső IP fejlécet is a csomaghoz. A varázslást a túloldali VPN szerver szedi le, a csomag onnantól hétköznapi csomagként viselkedik.

Amennyiben saját cégünk telephelyeit kötjük össze ilyen módon, akkor Intranet VPN-ről beszélünk. Ha különböző cégek kötik össze ilyen módon a hálózataikat (például full IT outsourcing), akkor pedig Extranet VPN-ről.

És akkor nézzük meg konkrétan is a varázslásokat.

3.5.1 PPTP

RFC 2637

Point-to-Point Tunelling Protocol.

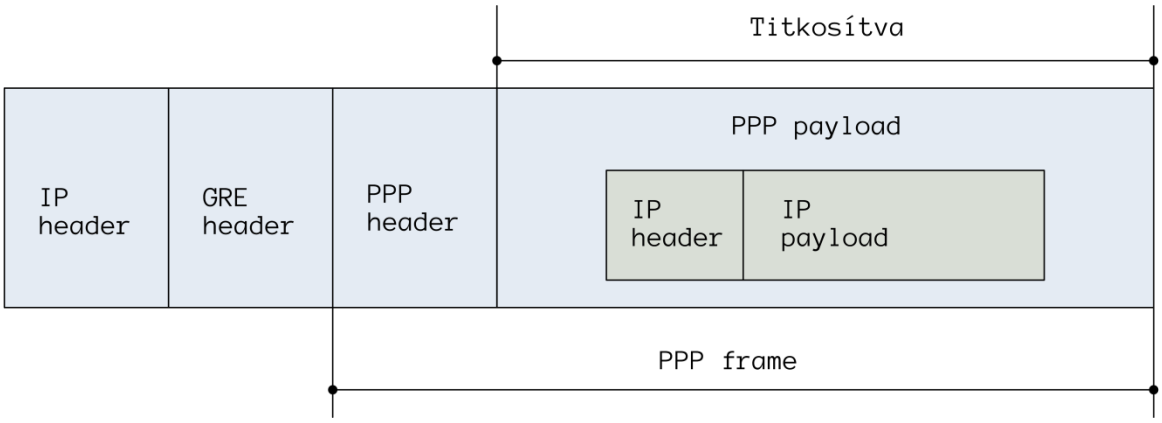
Akinek elkezd halványan derengeni, hogy de hasonlít már ez a név a korábban megismert PPP-hez (Point-toPoint Protocol), az nem jár messze a valóságtól.

A PPP-ről a korábban már többször is ajánlott könyvben írtam részletesen.

Nos, a PPTP valahogy így néz ki:

- A küldendő cuccot PPP keretbe csomagoljuk.
- Ezt a PPP keretet bekapszulázzuk. A módszer neve GRE.
- Az így keletkező csomag lesz egy új IP datagram payloadja.

Egy ábra száz szónál is szebben beszél.



3.28. ÁBRA PPTP CSOMAG SZERKEZETE

Induljunk ki legbelülről. Van az eredeti IP datagram. (Ez ugye áll egy IP fejlécből és egy IP payloadból, mely lehet ICMP csomag, UDP datagram vagy TCP szegmens.) Ezt a csomagot szeretnénk átjuttani a vad és szeszélyes interneten.
Először belecsomagoljuk egy PPP keretbe. Emlékszünk, ez egyszerre jelent tömörítést/titkosítást (MPPC/MPPE), illetve egy PPP autentikációt, mely meglehetősen sokféle lehet, az egyszerű PAP autentikációtól az EAP/TLS-ig.
A PPP keretet még el kell küldenünk szabályos csomagként. Ez a csomag úgy fog kinézni, hogy kap egy új IP fejlécet, ahol az IP címek a VPN struktúrájának megfelelően alakulnak ki. Az IP payload pedig egy GRE csomag.
Figyelem, a GRE protokollként kezelendő, ugyanolyan kódja van, mint a TCP-nek (6), az UDP-nek (17), az ICMP-nek (1) vagy a nemrég megismert AH-nak (51). A GRE

protokoll kódja konkrétan 47 - és tényleg protokollként működik. Hogy mást ne mondjak, neki is vannak sequence illetve acknowledgment number értékei.

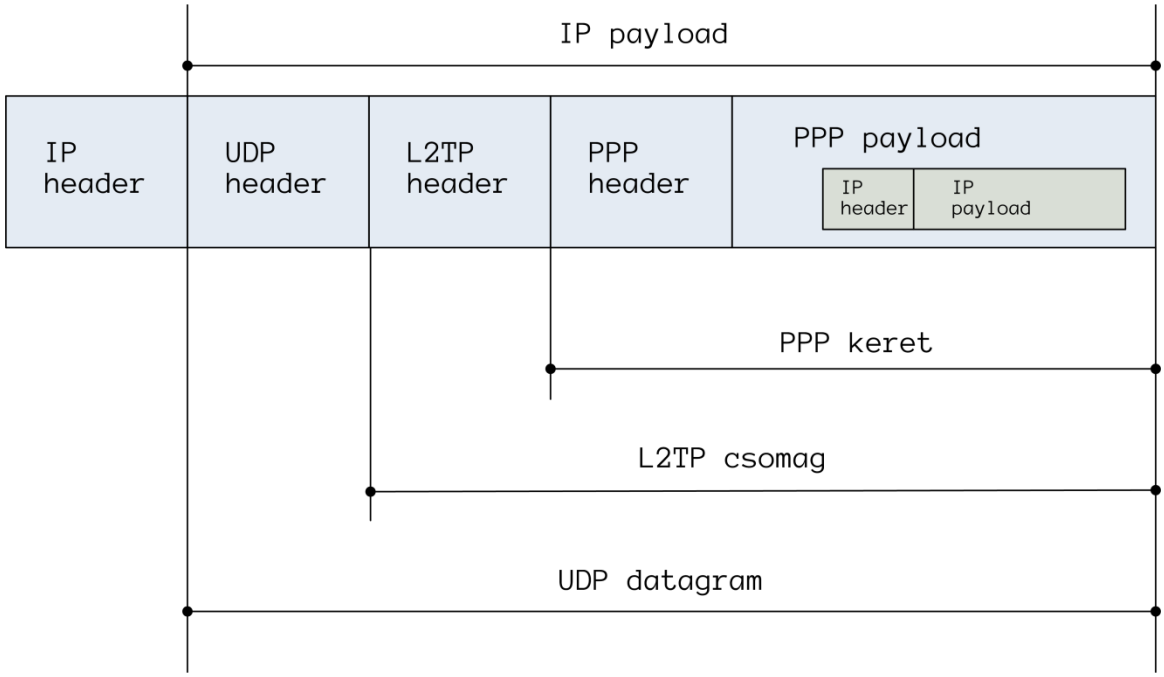
A PPTP-nek - az FTP-hez hasonlóan - van egy control csatornája, a Control Connection. Ez sima TCP protokollon keresztül megy, a PPTP szerver az 1723-as portot használja.

A PPTP volt a legelső VPN protokoll. A maga idején kifejezetten jónak számított.
Aztán jöttek a bajok. A technika fejlődésével meggyengült az MPPE titkosítás. A PPP autentikációk közül is sorra dőltek ki az egyszerűbbek.
Végül megjelent a kihívója, az L2TP/IPsec VPN protokoll. (Bár az igazi áttörést a NAT-T bevezetése jelentette, addig ugyanis a PPTP - ha egy kis trükközéssel is ugyan - de átment a natoló routereken, ezzel szemben az L2TP/IPsec nem.)

3.5.2 L2TP

RFC 2661

Az L2TP az önmagában egy tunneling protokol - vagy ha úgy jobban tetszik, akkor egy csatornázó, avagy egy kapszulázó. Ilyen értelemben a GRE protokollal esik egy kategóriába. (Az L2TP protokoll kódja 115.) És ennyi.



3.29. ÁBRA L2TP CSOMAG SZERKEZETE

Ha összehasonlítod a korábbi ábrával (*3.28. ábra PPTP csomag szerkezete*), láthatod, hogy olyan óriási nagy különbség nincs. GRE helyett L2TP kapszulázást használunk, plusz az egészet még beletesszük egy UDP datagramba. Így áll össze a csomag.
Említsük meg, hogy az L2TP nem csak IP protokollon keresztül működik, az RFC szerint értelmezve van Frame Relay és X.25 protokollokon keresztül is.

Mivel jobb ez, mint a PPTP? Biztonság terén nem sokkal. Sem az UDP, sem az L2TP nem ad semmilyen biztonsági pluszt a PPTP-hez képest. Az autentikációt és a titkosítást mindkét esetben a PPP végzi.
Kényelemben viszont annál inkább van különbség. Itt most ne arra gondoljunk, hogy a PPP csomag Pullman kocsiban utazik. Magát a csatornát könnyebb nekünk kezelni.

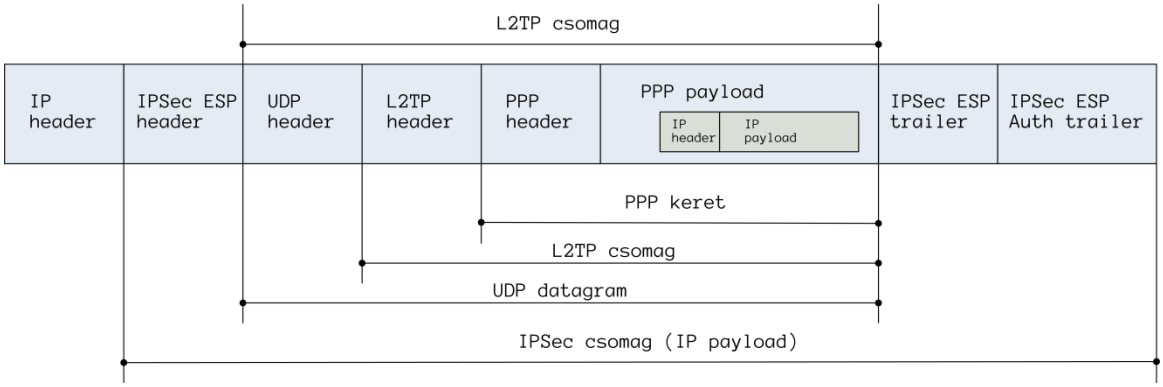
- Mind az adat, mind a control üzenetek ugyanazt az UDP stream-et használják, nem kell külön control TCP csatorna.
- Mind az L2TP kliens, mind az L2TP szerver az 1701-es UDP portot használja.

3.5.3 L2TP / IPSec

Jó. Mi van akkor, ha mi nem csak kényelmesebbek akarunk lenni, hanem szeretnénk biztonságosabb VPN csatornát építeni?

Ekkor jön az, hogy kombináljuk az L2TP csatornát az IPSec csatornatitkosító változatával. Ebből lesz az L2TP/IPSec.

> Alapértelmezésben a Windows Server 2008-ban nem létezik L2TP IPSec nélkül - azaz nincs olyan választási lehetőségünk, hogy csak L2TP csatornázást szeretnénk. Ellenben olyat viszont csinálhatunk, hogy L2TP/IPSec csatornát választunk, majd később a registryben letiltjuk rajta az IPSec komponenst. De a szakirodalom szerint ilyet csak tesztelési, illetve hibakeresési okokból cselekedjünk.



3.30. ÁBRA L2TP/IPSEC CSOMAG SZERKEZETE

Egyre bonyolultabb, mi? Annyira azért nem kell megijedni, egyszerűen csak elkezdtük egymásba csúsztatni az ábrákat. Látható középen a korábbi ábráról (*3.29. ábra L2TP csomag szerkezete*) ismerős L2TP csomag. Ha úgy nézzük, hogy ez egy egység, akkor az is látható, hogy tulajdonképpen erre az egységre nézve itt egy IPSec ESP transzformációt hajtunk végre, méghozzá Tunnel módban (*3.20. ábra ESP Tunnel mód*).

Ezzel gyakorlatilag - az ESP Tunnelhez hasonlóan - kellőképpen titkosítottuk és alá is írtuk a csomagot. A NAT Traversal-nak köszönhetően pedig manapság már a NAT sem akadály.

Esetleg felmerülhet a kérdés, hogy nem árt-e meg a jóbol is a sok? De bizony. L2TP/IPSec VPN csatorna esetén a PPP szintű MPPE titkosítás ki van kapcsolva. Gyengébb, mint az IPSec ESP, ergo felesleges.

3.5.4 SSTP

Azt mondtam volna, hogy a NAT nem akadály?

Nos, a helyzet azért nem ennyire egyszerű. A NAT igenis akadály. Csak éppen már megugorható.
A gond igazából az, hogy nem egykönnyen. PPTP esetében például a NAT-ba kell egy erre a célra felingerelt NAT editor. IPSec-nél meg a NAT-T.

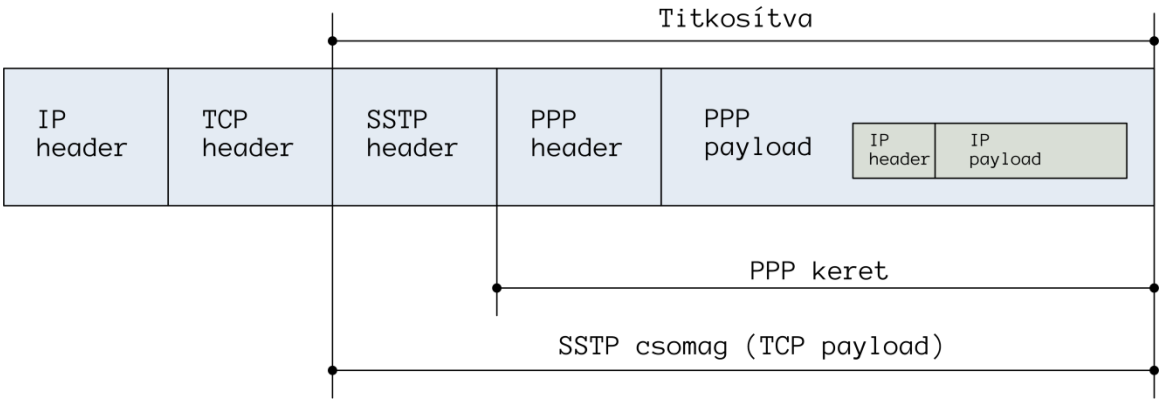
És akkor mi van a tűzfalakkal? A proxy szerverekkel? A tűzfalaknak engedniük kell mind a GRE, mind az ESP protokollokat. Biztos, hogy egy átlagos szállodai szobában ez engedélyezve van? Háát..
A proxyk pedig nemes egyszerűséggel nem támogatják sem a PPTP-t, sem az L2TP/IPSec forgalmat.

Anya, baj van. Marha biztonságos szeretnék lenni, de ez a zord világ nem hagyja.

Pedig itt van a kezünkben a helyzet kulcsa. Annyit beszéltünk már róla... hát miért nem zavarjuk át azt a szerencsétlen VPN forgalmat egy SSL/TLS csatornán?

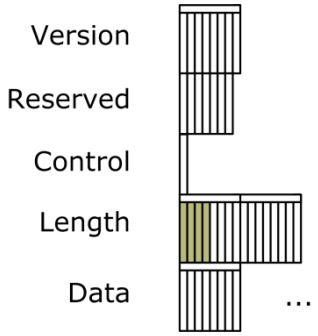
Biztonságos? A kulcsmérettől függ. Azaz ha nem bökjük el, akkor igen.

Tűzfalak, NAT routerek, proxyk? Mit kekeckednének egy teljesen átlagos 443-as porton menő forgalommal? Simán átengedik.



3.31. ÁBRA SSTP CSOMAG SZERKEZETE

Látható, hogy a PPP-be csomagolást most sem ússzuk meg. De legalább tudjuk, hogy becsületes VPN protokollal állunk szemben. Aztán ezt a PPP keretet dobjuk bele egy SSL titkosított HTTP csomagba. (A PPP MPPE itt is ki lett kapcsolva.)
Tudni kell, hogy az SSTP abszolút nem támogatja a site-to-site VPN-eket, azaz a csatorna mindig az SSTP kliens és az SSTP szerver között épül ki.
Az egyetlen ismeretlen szereplő az ábrán az SSTP kapszulázás, illetve az ahhoz tartozó fejléc. (Hiszen a HTTPS-t már ismerjük elég jól, a PPP-t meg szintén.)



3.32. ÁBRA SSTP HEADER

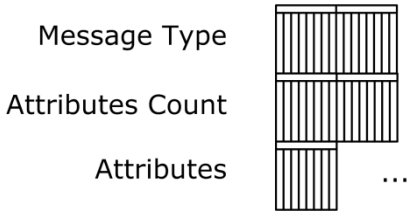
VERSION: Az alkalmazott SSTP verziószáma.

RESERVED: Majd kitaláljuk, mire lesz jó.

CONTROL: Az SSTP-nek is van Control, illetve Data üzemmódja. Ez a flag jelzi, hogy az aktuális csomag melyik tipusba tartozik.

LENGTH: Az első négy bit szintén foglalt, csak még nem tudjuk, mire. A maradék 12 bit a csomag teljes hosszát mutatja. (Header+payload)

DATA: Ha a control flag magas, akkor itt utazik a vezérlőüzenet. Ellenkező esetben pedig a PPP keret.



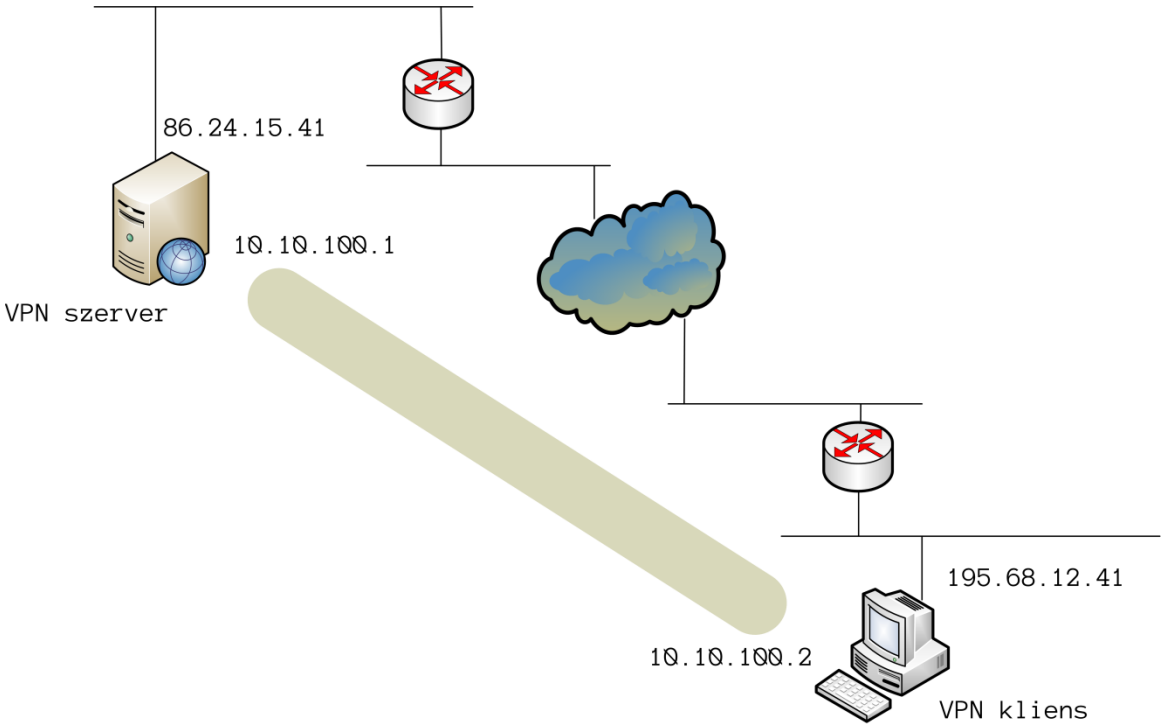
3.33. ÁBRA SSTP CONTROL ÜZENET

A PPP szerkezetet nyilván nem fogom itt bemutatni, az már máshol megtörtént. A Control csomagé viszont nem.

MESSAGE TYPE: A Control üzenet tipusa.

ATTRIBUTES COUNT: A Control üzenethez mennyi érték tartozik.

ATTRIBUTES: Maguk a konkrét értékek, listaszerűen.
Nézzük meg egy kicsit részletesebben is, hogyan jön létre egy SSTP kapcsolat.



3.34. ÁBRA SSTP KAPCSOLAT KIALAKULÁSA

1. A kliens a 195.68.12.41-es IP címéről felkeresi a szerver 86.24.15.41-es IP címét, a 443-as porton. Létrehoz egy TCP sessiont.

2. Ezen a session-on belül elindul egy SSL csatorna kiépítése. A kliens elkéri a szerver tanúsítványát a session key titkosításához. Leellenőrzi. A szerver nem ellenőrzi vissza a klienst.

3. A kliens küld egy HTTPS Request-et, immár a titkosított SSL csatornában.

4. A kliens küld egy SSTP control üzenetet. Ha összefütyültek, akkor kezdődhet a PPP kapcsolat kialakítása.

5. A HTTPS feletti SSTP-ben megtörténik a PPP kapcsolat felépítése. Tudjuk, hogy MPPE titkosítás itt már nincs, autentikációban viszont a teljes PPP skálát használhatjuk, beleértve az EAP verziókat is (SSL-ben EAP-TLS... finom). A PPP autentikáción keresztül a szerver autentikálja a kliens felhasználót.

6. Amint a PPP is felállt, a számítógépeken megjelenik egy új IP cím. Az ábrán ezek a 10.10.100.2 a kliensnél és 10.10.100.1 a szervernél. Finito.

3.5.5 VPN Reconnect

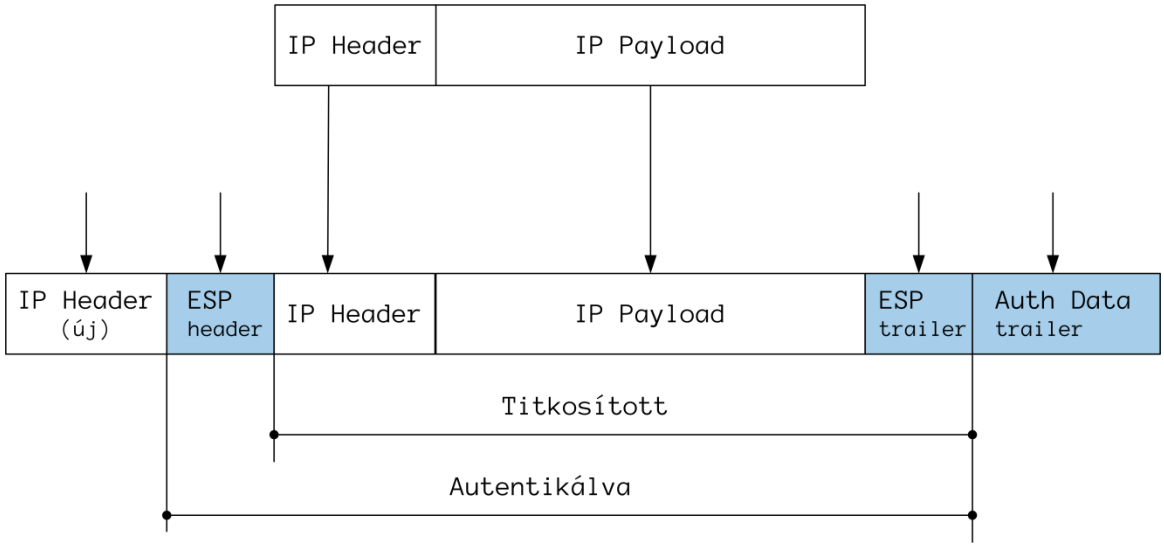
Érdekes játék ez a security. Időnként olyan, mint egy nagy doboz Lego: előveszem a darabokat és ha így rakom össze, akkor kávédaráló, ha másképpen, akkor meg tank lesz belőle.

A VPN Reconnect is ilyen legós-összedugdosós VPN protokoll. Megfogtuk az L2TP/IPSec modellt, leszedtük róla az L2TP kockát, félretettük az UDP kockát, a megmaradt IPSec blokkban meg kicseréltük az IKE SA-t IKEv2-re.

Ami keletkezett, azt elneveztük VPN Reconnect-nek.

Most az egyszer az átnevezésnek tényleg volt értelme is. Ha emlékszünk, írtam, hogy az IKEv2-hez kijött később egy kiegészítés MOBIKE néven. Nos, ez pont a mobil kliensek kiszolgálására hivatott. Azaz olyan kliensekére, akik nagy sebességgel mozognak, miközben tolják az iwiw-et, VPN-en keresztül belépve a céges hálózatukba. Aztán ha mozgás közben megszakad a kapcsolatuk, átváltanak egy másik cellára - és borzasztóan nem szeretnék, ha erről bármilyen módon is tudomást szereznének. Azaz a VPN legyen olyan kedves, oldja meg az újracsatlakozást magától.

Csak, hogy ne legyen egyszerű az élet, MS berkekben a VPN Reconnect és az IKEv2 teljesen szinonim kifejezések. De mi tudjunk róla, hogy az egyik egy VPN protokoll, a másik pedig egy SA.



3.35. ÁBRA IPSEC ESP TUNNEL

A csomagszerkezet bemutatásával nagyon könnyű dolgom volt, egyszerűen idemásoltam a korábbi ábrát. (*3.20. ábra ESP Tunnel mód.*) Hiszen ez gyakorlatilag nem más, mint egy mezei IPSec ESP Tunnel, ahol az első SA-t az IKEv2 hozza létre.
De azért ne becsüljük le ennyire. Nem csak a mobil kliensek kiszolgálása a nagy erőssége a protokollnak, hanem az autentikáiós eljárások bővülése (full EAP), illetve ráncfelvarrása is. Hogy mást ne mondjak, a VPN Reconnect esetében már nincs szükség a VPN kliensen tanúsítványra, elég, ha a szerveren van.

Nézzünk néhány forgatókönyvet.

Mind a kliensnek, mind a VPN szervernek egyaránt van IPv4 és IPv6 címe. Ebben az esetben a VPN kapcsolat először az IPv6-os címek között jön létre - de ez később bármikor megváltoztatható, méghozzá úgy, hogy a VPN megszakadását a kliens nem fogja észlelni.

Ha a VPN Reconnect úgy van konfigurálva, hogy szerepel benne célként a céges VPN szervernek mind a külső, mind a belső lába, akkor simán előfordulhat, hogy Kereskedő Béla éppen sétál be a cégéhez, közben vadul győzködve az egyik ügyfelét egy OCS alapú mobil kliensről - és nem veszi észre, hogy ahogy belépett az ajtón, a kliense immár a VPN szerver belső lábára kapcsolt át.

A cég informatikusa éppen a Teched-en koptatja a székeket. Járkál egyik épületből a másikba, laptopja természetesen a helyi wifin keresztül folyamatosan rákapcsolva a céges VPN hálózatra. Hiába sétál ki az egyik Access Point hatóköréből és sétál be egy másikéba, hiába változik meg az IP címe - a VPN Reconnect csak kitart és kitart. És ez igaz akkor is, ha olyan termekbe téved a szerencsétlen, ahol már igen gyenge a wifi, ahol állandóan szakadozik. A kliens nem fog neki visszaszámolós reconnect ablakot feldobni, megoldja minden patália nélkül az újrakapcsolást.

Mindez nem csak a MOBIKE következménye. Nem részleteztem, mert ehhez egy közel 100 oldalas RFC-t kellett volna feldolgoznom, de az IKEv2-ban - az IKEv1-hez képest - mind koreográfiában, mind a felhasználható eszközkészletben nagyon komoly változások történtek. Például nem lehetne ilyen rugalmasan kezelni a hálózati változásokat, ha az egyes tánclépéseket - autentikáció, kulcscsere, csatornaépítés - nem gondolták volna ennyire át, nem egyszerűsítették volna le a végtelenségig.

3.5.6 Összefoglalás

Még egy összefoglalás. De nem azért, mert átláthatatlanul bonyolult lett volna a fejezet. Sokkal inkább azért, hogy egymás mellett lássuk, melyik VPN protokoll mit tud, hol használható optimálisan.

3.2. táblázat

	PPTP	L2TP/IPSec	SSTP	VPN Reconnect
Operációs rendszer	XP, 2003, Vista, WS08, W7, WS08 R2	XP, 2003, Vista, WS08, W7, WS08 R2	Vista, WS08, W7, WS08 R2	W7, WS08 R2
Előfordulás	Remote Access Site-to-site	Remote Access Site-to-site	Remote Access	Remote Access
NAT	NAT editor	NAT-T	No problemo	NAT-T
Tűzfal	Engedélyezés után	Engedélyezés után	Legtöbbször simán	Engedélyezés után
Web Proxy	Non passarant	Non passarant	Autentikáció nélkül	Non passarant
Mobil támogatás	Nincs	Nincs	Nincs	Van
Autentikáció	User autentikáció a PPP-ben	Gép autentikáció az IPSec-ben, utána user autentikáció a PPP-ben	User autentikáció a PPP-ben	Gép vagy user autentikáció az IKEv2-ben

A Microsoft ajánlások a következő ökölszabályokat javasolják arra az esetre, ha olyan VPN szerverünk van, mely az összes VPN protokollt ismeri (Halló, mi is a címe a könyv második részének?), és a klienseink is leginkább Windows7-ből állnak.

- A legmagasabb prioritású a VPN Reconnect legyen. A kliensek mindenképpen ezt próbálják meg legelőször kialakítani. Ha működik, akkor sokkal jobb a többinél.
- De nem minden helyzetben működik. Egy webes proxy, egy túl szigorú tűzfal, egy NAT-T-t nem ismerő natoló router hamar elgázolják a karrierét. Ilyenkor próbálkozzunk az SSTP-vel, az még a falon is átmegy. (Kivéve, ha felhasználói autentikációt alkalmazó WEB Proxy-val találkozik. Az még neki is sok.)
- Amennyiben pedig feltétlenül site-to-site VPN kialakítására kell adnunk a fejünket, akkor jöhet az L2TP/IPSec.
- PPTP-t ma már egyáltalán nem ajánlott telepíteni. Elméletileg elképzelhető olyan szituáció, hogy site-to-site VPN-t kell kialakítanunk, de van útközben egy natoló router, mely a NAT-T-t nem ismeri, ellenben NAT Editorral le tudja kezelni a PPTP-t... de ilyenkor érdemesebb inkább a natoló eszközt cserélni.

3.6 AUTENTIKÁCIÓ

Így végiggondolva, eléggé mostohán szoktunk bánni ezzel az autentikációval. Felbukkan itt, írunk róla ott, egyik helyen ilyen módszerekkel csinálják, másik helyen olyannal - de önálló témaként nem nagyon szoktunk vele foglalkozni.
Nem véletlenül. Az autentikáció ugyanis legfőképpen viszony. Valaki meg akar bizonyosodni másvalakinek/másvalaminek a valódiságáról.

- Jónapotkívánokjogosítványtforgalmiengedélytkérem.
- Vagy a diszkóban az UV csuklószalag.

Informatikában is nagyjából erről van szó, de nagyon nem mindegy, ki kér, kitől kér és leginkább mikor kér. Akár egy egyszerű folyamatban is előfordulhat, hogy a csatorna kiépítéséhez a gépek autentikálják egymást, de a gépen futó alkalmazás eléréséhez maga az alkalmazás is autentikálja a felhasználót. (Pl. OWA.) Habár ugyanazt a szót használjuk mindkét lépésre, de a kettő között ég és földnyi különbség is lehet.
Másik jó példa erre egy L2TP/IPSec csatorna: a csatorna kiépítéséhez a két host autentikálja egymást, de a VPN mögötti belépés autentikációjához már a PPP protokoll autentikációs módszerei közül valamelyikkel vizsgáljuk meg a felhasználót.

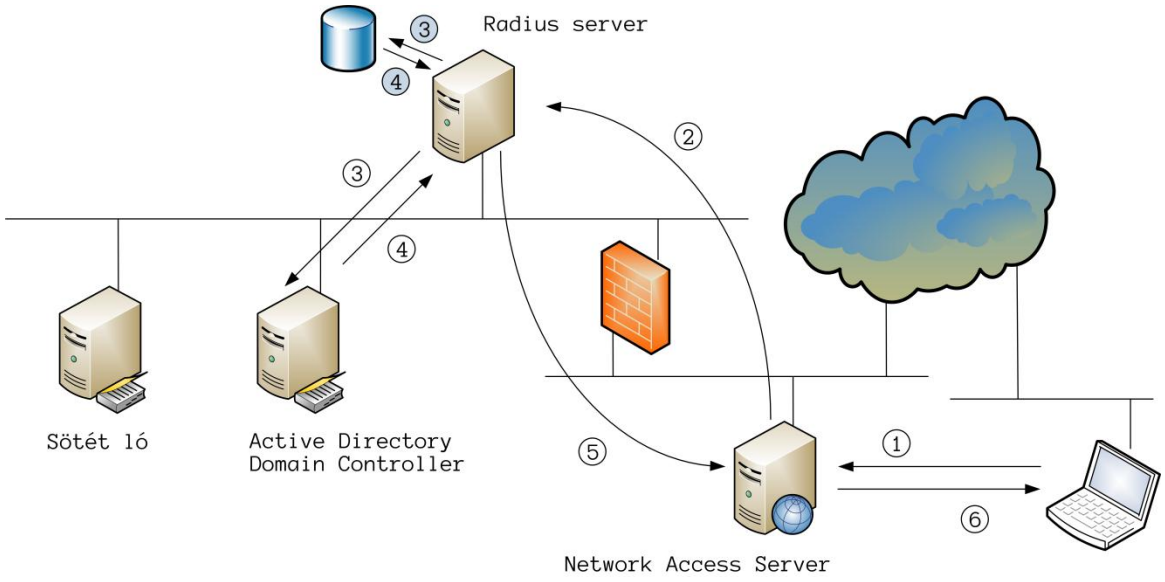
És akkor nem beszéltünk még azokról az eszközökről, kliens-szerver alapú alkalmazásokról, melyek kifejezetten arra lettek kinemesítve, hogy egyfajta autentikációs motorként - Authentication Provider - levegyék az autentikáció feladatának terhét az úgynevezett Network Access[13] szerverekről.

Több ilyen rendszer létezik, a Windows világban a RADIUS terjedt el, IAS, illetve a Windows Server 2008-ban már NPS néven.

[13] Tipikus NAS például az eddig VPN szerverként emlegetett eszköz. Figyelem, nem keverendő össze a teljesen azonos rövidítéssel bíró Network Attached Storage eszközökkel.

3.6.1 RADIUS

RFC 2865 - és egy csomó egyéb



3.36. ÁBRA RADIUS SZERVER MŰKÖDÉS KÖZBEN

1. Az odakint bolyongó laptopos ember be akar lépni a munkahelyi hálózatába. Felveszi a kapcsolatot a NAS-sal, mely jelen esetben egy VPN szerver. (De lehet RAS szerver, illetve más szituációban egy port szintű azonosítást használó switch is.)

2. Mivel az autentikáció outsourcolva lett, a VPN szerver - mint Radius kliens - felveszi a kapcsolatot a Radius szerverrel. Átadja neki a szükséges paramétereket. (Valójában ez a lépés egy kicsit bonyolultabb, a Radius szerver bizonyos esetekben még visszakérdez, ilyenkor a NAS szerver is visszakérdez... szóval maradjunk annyiban, hogy ezek itt addig variálnak, amíg összeáll a küldendő csomag.)
A Radius szerver első körben leellenőrzi, hogy a Radius kliensnek egyáltalán van-e joga hozzá fordulni?

3. A Radius szerver saját adatbázisából keresi ki a felhasználót.

4.Leellenőrzi, hogy sikeres volt-e az autentikáció.

Alternatív változat:

3.Egy külső címtárszerverben keres rá a felhasználóra. Ilyen lehet egy LDAP szerver, egy Active Directory tartományvezérlő, de akár egy SQL szerver is.

4. A külső szerver visszajelez a Radius szervernek, hogy sikeres volt-e az autentikáció.

5. A Radius szerver visszajelez a Radius kliensnek, hogy sikerült-e, vagy sem az autentikáció.

6. A NAS szerver vagy beengedi, vagy képen törli a laptopos embert.

Nos, kérem, nagyjából erről lenne szó. Tényleg nagyjából, mert hamarosan részleteiben is végig fogunk menni a folyamaton.

Beszéljünk egy kicsit az extrákról. A Radius ugyanis nem csak autentikálni tud, hanem teljes értékű AAA szerver.
Azaz ismeri a következő funkciókat:

- AUTENTIKÁCIÓ : Erről beszéltünk eddig. Bejöhetsz, vagy sem?
- AUTORIZÁCIÓ : Ha már bejöttél, mihez van jogod? Milyen erőforrás van hozzád rendelve, ezek közül melyeket kell helyből megkapnod?
- ACCOUNTING : Adatokat gyűjtünk: mennyi ideig használtad a hozzáférést? Ezen adatok alapján az ISP például számlázni tud neked. (Vagy ha van belső számlázás a cégnél, akkor az IT a többi részlegnek.)

A Radius kliens és Radius szerver közötti kommunikáció során hat különböző csomag jelenhet meg:

ACCESS-REQUEST
A Radius kliens küldi a Radius szervernek. Jelzi, hogy autentikációs és autorizációs igénye van. Ebben az üzenetben utazik a NAS által összeállított autentikációs csomag.

ACCESS-CHALLENGE
Amennyiben olyan az autentikáció tipusa, hogy a Radius szervernek plusz információkra van szüksége, akkor ebben a csomagban jelez vissza a Radius kliensnek. (Tipikusan ilyesmi történik a challange-response tipusú autentikációk esetén.)

ACCESS-ACCEPT
A Radius szerver visszajelez a kliensnek, hogy az autentikáció sikeres volt.

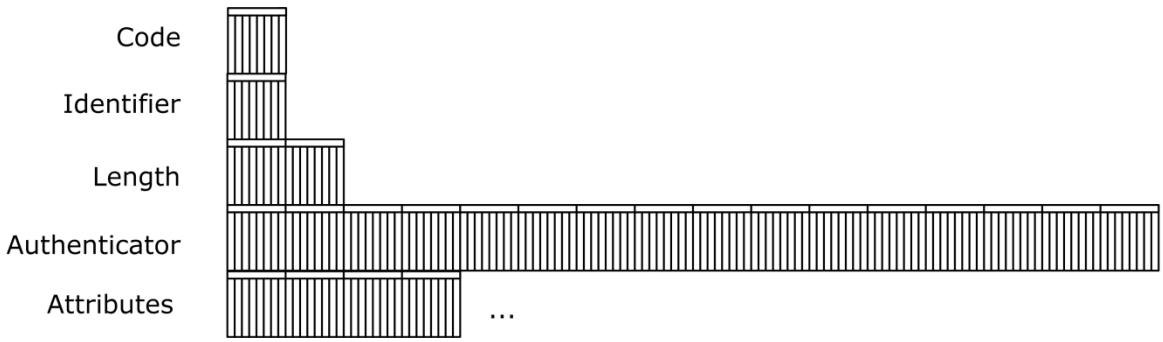
ACCESS-REJECT
A Radius szerver visszajelez a kliensnek, hogy az autentikáció sikertelen volt.

ACCOUNTING-REQUEST
A Radius kliens jelez a szervernek, hogy accounting információkat ad meg.

ACCOUNTING-RESPONSE
A Radius szerver jelez vissza a kliensnek, hogy tudomásul vette a jelzést és fel is dolgozta.

Ejtsünk még pár mondatot erről az accounting folyamatról.
Amikor létrejött a kapcsolat, azaz a laptopos ember belépést nyert a belső hálózatba, a Radius kliens küld egy ACCOUNTING-REQUEST üzenetet a Radius szervernek. Ez az üzenet jelzi, hogy el kell indítani egy accounting folyamatot, megadja, milyen szolgáltatás lett igénybevéve és kicsoda által. Erre jelez vissza a Radius szerver egy ACCOUNTING-RESPONSE üzenettel, miszerint az accounting folyamat rögzítése megtörtént.
Amikor véget ért egy kapcsolat, a Radius kliens küld egy újabb ACCOUNTING-REQUEST csomagot, benne egy Accounting Stop üzenettel. Ebben az üzenetben van benne minden lényeges információ: ki volt, milyen szolgáltatásokat vett igénybe, mennyi ideig használta, mindenféle statisztikák, letöltött/feltöltött bájtok... meg ilyesmik. A Radius szerver mindezeket befogadja, leállítja az accounting folyamatot és minderről értesíti a klienst egy újabb ACCOUNTING-RESPONSE csomaggal.

Mind a hat csomag UDP-ben utazik. A Radius szerver tipikusan a 1812-es porton figyel az autentikációs üzenetekre és a 1813-ason az accounting üzenetekre.



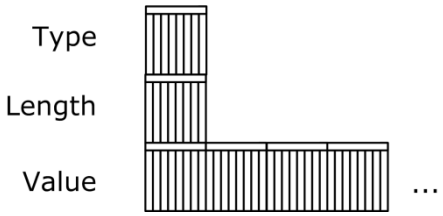
3.37. ÁBRA RADIUS CSOMAG

CODE: A Radius üzenet tipusa.

IDENTIFIER: Egy azonosító, mely az egymáshoz tartozó üzeneteket kapcsolja össze. Általában igaz, hogy a Response üzenetben ebbe a mezőbe a request üzenetben lévő kódot másolják át.
LENGTH: A teljes Radius üzenet hossza.

AUTHENTICATOR: Egy olyan mező, melyben egy titkosított jelszó (közös titok) utazik. A Radius kliens szokta visszaellenőrizni ezzel a Radius szervert.

ATTRIBUTES: Minden, mi szem-szájnak ingere. Minden lényeges, az autentikációt, az autorizációt és accountingot érintő információ ebben a mezőben utazik.



3.38. ÁBRA RADIUS ATTRIBUTES

Minden attribútumnak van egy típusa, mivel az attribútumok hossza nem rögzített, így nyilván fontos paraméter a hossz is, az érték mezőben pedig már a konkrét információk utaznak. A felsorolásuktól eltekintek, hihetetlenül sok van belőlük.

Akit érdekel:
http://www.iana.org/assignments/radius-types

Annyi van belőlük, hogy már csak végiggörgetni a képernyőn is épp elég fárasztó. Pedig ez még nem is az összes, ezek csak az általános attribútumok.

Külön kategóriákba esnek az úgynevezett vendor-specifikus attribútumok. Ezek a 26-os tipusú attribútum alatt jelennek meg, gyakorlatilag al-attribútumként. Ezeknek a felsorolását is kihagyom.

A vendorok listáját itt találhatjuk:
http://www.iana.org/assignments/enterprise-numbers

RFC 2548

Végül a fenti RFC-ben találjuk a Microsoft, mint vendor által használt plusz Radius attribútumokat.

3.6.2 KÉTFAKTOROS AUTENTIKÁCIÓ

A fenti ábrán (*3.36. ábra RADIUS szerver működés közben*) az a Sötét Ló nevű számítógép nem csak úgy viccből került oda.

Ugyanis nem csak egyszeres autentikációt nyújtó Authentication Provider szerverek (Radius, Takacs) léteznek a világban. Vannak olyanok is, amelyek képesek a kétfaktoros autentikációra.

Jogos lehet a kérdés, hogy mi is ez a kétfaktoros autentikáció és miben különbözik mondjuk a Kétfarkú Kutya párttól?

Mint látható, a Radius csak egyszeres autentikációt nyújt. Azaz ellenőrzi a felhasználó nevét, jelszavát... oszt jól van. A kétfaktoros autentikációhoz ez kevés. Oda kell még egy olyan elem, egy fizikailag is megfogható elem, mely az egyszeres autentikációval együtt adja meg a belépés lehetőségét. Ahogy az angol mondja, "*something you have, something you know*", azaz van valamid és tudsz valamit.
Ilyenekre gondolok:

- Kliens tanúsítvány smartcard-on.
- Valami OTP.

Az első esetben a felhasználó betolja smartcardot a gépébe, megpróbál rácsatlakozni a szupertitkos weblapra, beüti a usernevét, jelszavát, az autentikációs eljárás felolvassa a tanúsítványt - és ezek együttes autentikációja adja meg a hőn áhított belépést.

A másik már jóval cifrább.

RFC 2289

Először is, nem, nem a nagy magyar bankról van szó. Az OTP a One-Time Password rövidítése. Az elv lényege, hogy a felhasználó egy jelszót csak egyszer használ.
Hogy lesz ebből kétfaktorú autentikáció?
Nézzük a legegyszerűbb OTP rendszert. A felhasználó kap egy nyomtatott, számozott listát, rajta sorban egy csomó jelszó. Mindegyikkel csak egyszer lehet belépni, és csak a megadott sorrendben lehet felhasználni a jelszavakat. Ekkor a felhasználó beírja a usernevét, a hagyományos jelszavát (eddig tartott a 'valamit tudsz' komponens), majd előveszi a papírlapot és kikeresi a legelső, még nem áthúzott jelszót, majd azt is beírja egy másik rubrikába. (Maga a papír a 'valamid van' komponens.)

Habár ezt a módszert még manapság is használják webes banki elérésekre, de azért ennél már léteznek elegánsabb módszerek. Ezek mind valamilyen külső hardver komponensen alapulnak.
De mielőtt belemennénk a kütyük ismertetésébe, beszéljünk arról, hogyan is keletkeznek ezek a jelszavak. Mert ezek keletkeznek, nem pedig a központi IT tölti fel a listát az eszközre, mielőtt kiadja. (Mellesleg ezeket a generált kódokat már nem jelszavaknak hívják, hanem passcode-nak.)

- A passcode generálása a pontos időn alapul. Ehhez nyilván kell egy pontos óra is az eszközben. Innentől kezdve csak az algoritmus kérdése, hogy a szerveren is ugyanaz a passcode generálódjon, mint a kütyüben. A módszer előnye, hogy a kód csak rövid ideig használható.
- A passcode generálása a matekon alapul. Ekkor a kütyübe az első kódot még a rendszergazda fiúk töltik be, a következők pedig mindig az előzőekből keletkeznek egy jól definiált algoritmus alapján.

Nézzük akkor az eszközparkot.

A nyomtatott papírról már beszéltem.

A másik leginkább kézenfekvő eszköz a mobiltelefon. Ez az eszköz ráadásul kétféleképpen is használható.

- Az egyik szerint telepítünk rá egy célszoftvert, mely képes legenerálni az éppen aktuális kódot.
- A másik szerint nem telepítünk semmilyen szoftvert, hanem amikor a felhasználó begépeli a usernevét és a jelszavát, akkor küldünk neki egy kódot SMS-ben a hozzá regisztrált mobiltelefonra.

Végül itt van a tokenhadsereg. Tucatnál is több gyártó gyárt ma már egymással nyilván nem kompatibilis rendszereket, mindegyik a maga tokenjét használva. A token egy kis hardverkütyü, mely gombnyomásra kiadja a következő kódot. Ha idő alapú, akkor pontos óra kell bele, ha matekozós, akkor induló passcode.

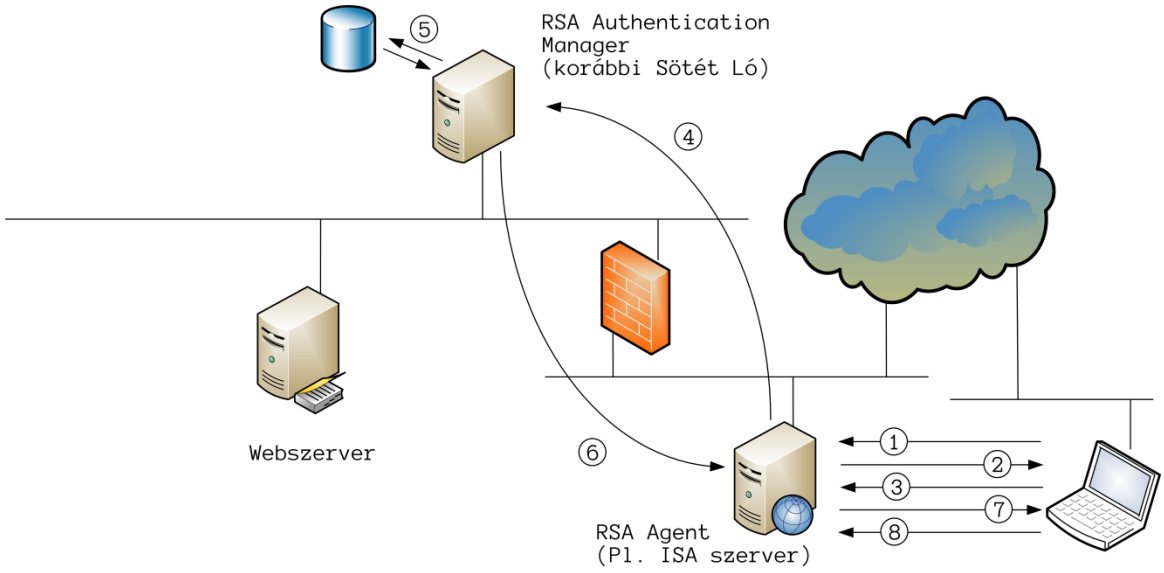
Térjünk vissza arra a Sötét Lóra és a Windows világba. A kétfaktorú autentikációt kétféleképpen tudjuk megvalósítani:

- A Radius szerver felturbózásával. Ezt nevezzük Radius-OTP megoldásnak.
- A külső gyártó OTP Manager szerverével. Na, ez a Sötét Ló.

Ha megnézzük például az ISA2006 szerver esetében a választható kétfaktorú autentikációkat, akkor ezeket látjuk:

- Tanúsítvány alapú
- Radius OTP
- RSA SecureID

A Radius-OTP esetében nincs nagy változás az ábrán, egyszerűen a Radius szervert kicseréljük egy Radius-OTP szerverre és minden megy tovább a maga útján.
Az RSA SecureID esetében viszont már más egy kicsit a helyzet.



3.39. ÁBRA RSA SECUREID

1.
A kliens küld egy kérést az RSA Agent felé. Legyen ez egy ISA2006 szerver, ahol form-based autentikáció van beállítva, azaz a Form-Based Authentication Filter aktivizálja magát.

2.
A filter visszatol egy formot a kliensnek. Tőcsdki - mondja.
Ez a form háromféle is lehet, attól függően, hogy mit kérünk:

- USERNÉV, JELSZÓ: Ez speciel nem a mi esetünk, ekkor ugyanis nincs OTP autentikáció.
- USERNÉV, PASSCODE: Ez már OTP, de csak egyfaktorú.
- USERNÉV, JELSZÓ, PASSCODE: Igen, ez az igazi kétfaktorú autentikáció.

3.
A kliens kitölti a formot és egy HTTP POST paranccsal visszaküldi.

4.
Az RSA Agent továbbítja az autentikációs csomagot az RSA Authentication Manager szereplőnek.

5.
Az RSA Authentication Manager molyol rajta egy sort.

6.
Visszaküldi az eredményt az RSA ügynöknek.
7.
Ha nem sikerült az autentikáció, akkor az ügynök küld egy HTTP COKI üzenetet a kliensnek.
Amennyiben sikerült az autentikáció, akkor viszont egy HTTP COOKIE-t küld. Azaz ekkor még nincs direkt beengedés. Ez a süti tartalmazza a kliens autentikációs adatait, méghozzá úgy betitkosítva, hogy csak az RSA Agent tudja elolvasni. Emellett a süti még úgy van paraméterezve, hogy tilos a merevlemezen tárolni. Csak a kliensgép memóriájában lesz jelen.
De ezzel még nincs vége. Az RSA Agent küld a csomagban egy REFRESH fejlécet is, ezzel kényszerítve a klienst, hogy próbálja meg újra a behatolást.

8.
A kliens meg is próbálja. És igen... mivel most már van egy csodakukija, az RSA Agent beengedi.

Azt kérdezed, mi volt ez a matyóhímzés itt a végén a sütivel?
OTP, kérem szépen, OTP. A kliens csak egyszer adhat meg egy passcode-ot, többször nem. Mi van akkor, ha akármiért is újra kell autentikálni? Mondjuk, egy újabb kérésnél? (Aztán Single-Sign-On, hogy mást ne mondjak.) Ilyenkor igazából az RSA ügynököt nem az érdekli, hogy van-e még újabb passcode-ja a kliensnek, hanem megelégszik azzal az információval, hogy egyszer már volt egy érvényes. (Ezért kell a sütit csak a memóriában tárolni. Hogy annyira sokáig azért ne éljen ez a státusz.)

Egy utolsó gondolat. Van a képen egy webszerver. Most ő lett a sötét ló. Mit keres ez egyáltalán itt?
Hát, egyelőre csak errefelé ólálkodik. De ebből az adok-kapokból leeshet neki is. Ezt úgy hívják, hogy Authentication Delegation. Vagy más kifejezéssel úgy, hogy Publishing Rules.

Ugye, ismerős? De ez már egy másik könyv témája.

4 KIVEZETÉS

Minden normális könyvben arra szokta buzdítani a szerző a kedves olvasót, hogy ha kérdése van, akkor ne hezitáljon, tegye föl bátran.
Ez ilyen szempontból is rendhagyó könyv. Én azt mondom, hogy eszed ágában se legyen tőlem bármit is kérdezni. Nem azért, mert én egy undok vadmalac vagyok... hanem azért, mert ez egy alulról korlátos könyv.

Elmagyarázom.

Aki van annyira bátor, hogy könyvet ír, annak nagyon kell tudnia. Ezt a tudást legtöbbször nem is tudja teljesen kiírni magából. Már nyomdában van a könyv, amikor eszébe jut, hogy ez is kimaradt, meg az se lett teljesen kibontva. Azaz ha bárki kérdez valamit, ami nem került bele a könyvbe, a szerző magától értetődően el tudja magyarázni.
Ez a felülről korlátos könyv. Ilyen volt az Exchange 2007-ről szóló könyvem.

Ennek a mostani könyvnek már a legelején jeleztem, hogy rendhagyó kísérletre készülök. Olyan területről írok, amelyhez nem értek. Pontosabban értek valamennyire, de messze nem annyira, mint amilyen mélyen el akarok merülni a témában. Menetközben tanulok... és a megértett dolgokat gyermeki örömmel adom át, mint megannyi újdonságot.
Leírtam mindent, amit tudok... sőt valójában többet is írtam le, mint amit ténylegesen tudok. Hiszen a keretek és datagramok szerkezetét, a szabványok lényegét én is kézikönyvekből, weblapokról szedtem össze.
Ergo ez egy alulról korlátos könyv.

Kérdezni kérdezhetsz, persze... de én is csak annyit tudok tenni, amennyi neked is rendelkezésedre áll: gugli vagy bing.

Ellenben ha te a téma szakértője vagy és csak a viccek miatt olvastad végig a könyvet, aztán úgy találtál benne ordító hibákat - nos, te *ne tétovázz*, írd meg ezeket egyből a jpetrenyi@gmail.com címre. Az online megjelenésnek ugyanis pont az az óriási előnye, hogy a tartalom módosítható, aktualizálható.

A végére egy jó hír: a könyv működik. Amikor gyűjtöttem az anyagot, szép lassan kezdett érthető lenni a TCP/IP működése. Amikor írtam a könyvet, már azt hittem, hogy értem - csak ekkor még darabokban láttam mindent, hiszen ebben a fázisban a részletek kidolgozása volt a jellemző. És amikor lektoráláskor olvastam el egyben az egészet, akkor állt csak össze a teljes kép. De összeállt.

5 FORRÁSOK, LINKEK

Ez a könyv úgy készült, hogy elolvastam hozzá RFC-ket, izmos kéziratokat - majd ahol úgy éreztem, hozzáolvastam a netről is. Rengeteg link gyűlt össze a végére. Ezeket a linkeket találjátok meg a ebben a fejezetben, minimálisan strukturálva.

SZAKKÖNYVEK:

Alapvetően ezekre az írott forrásokra támaszkodtam:

- Joseph Davies: Windows Server 2003 TCP/IP Fundamentals for Microsoft Windows
- Joseph Davies: Windows Server 2008 TCP/IP Fundamentals for Microsoft Windows
- Joseph Davies, Tony Northrup with the Microsoft Networking Team: Windows Server 2008 Networking and Network Access Protection (NAP)
- Joseph Davies: Windows Server 2008 TCP/IP Protocols and Services
- Stephen Thomas: HTTP Essentials

RFC

A tiszta forrás, ahol minden megvan:

- http://tools.ietf.org/html/
- http://www.rfc-editor.org/rfc.html
- http://www.networksorcery.com/enp/
- http://en.wikipedia.org/wiki/Request_for_Comments

KIEMELT LINKEK:

mICK publikációi:

- http://inetcom.hu/mick/publikaciok/aktivpasszivftp.htm
- http://inetcom.hu/mick/publikaciok/ipsecnat.htm
- http://inetcom.hu/mick/publikaciok/https.htm
- http://inetcom.hu/mick/publikaciok/vpn1.htm
- http://inetcom.hu/mick/publikaciok/vpn2.htm
- http://inetcom.hu/mick/publikaciok/ipsec1.htm
- http://inetcom.hu/mick/publikaciok/ipsec2.htm
- http://inetcom.hu/mick/publikaciok/ipsec3.htm
- http://inetcom.hu/mick/publikaciok/ipsecports.htm
- http://inetcom.hu/mick/publikaciok/mime1.htm
- http://inetcom.hu/mick/publikaciok/mime2.htm
- http://inetcom.hu/mick/publikaciok/mime3.htm

TCP/IP Guide:

http://www.tcpipguide.com

Internetworking Technology Handbook:
http://www.cisco.com/en/US/docs/internetworking/technology/handbook/ito_doc.html
TCP/IP Networks:
http://www.citap.com/documents/tcp-ip/tcpip001.htm
Protocols Directory:
http://www.protocols.com/protocols.htm
Internetworking Guide:
http://technet.microsoft.com/en-us/library/cc951210.aspx

WINDOWS SERVER 2008 ÉS VISTA

New Networking Features in Windows Server 2008 and Windows Vista
http://technet.microsoft.com/en-us/library/bb726965.aspx
Windows Server 2008 Networking:
http://technet.microsoft.com/en-us/library/cc753940(WS.10).aspx
Windows Vista Networking Tecnologies:
http://en.wikipedia.org/wiki/Windows_Vista_networking_technologies

ÖMLESZTETT LINKEK

Végül ömlesztve egy csomó link abból a segédfájlból, ahová menetközben szórtam a hasznos infókat tartalmazó találatokat.

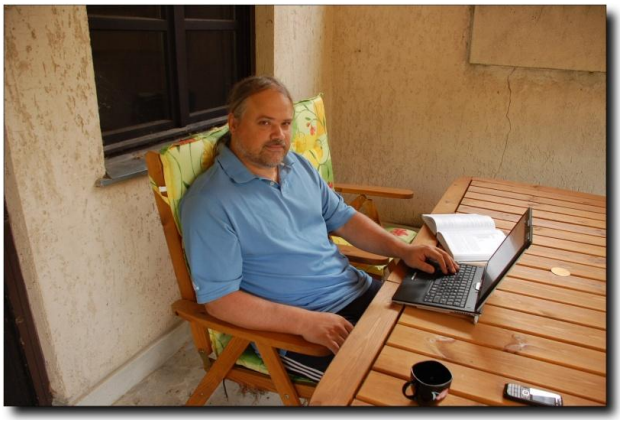
http://en.wikipedia.org/wiki/Http
http://www.w3.org/Protocols/
http://www.ietf.org/rfc/rfc2616.txt
http://en.wikipedia.org/wiki/List_of_HTTP_status_codes
http://en.wikipedia.org/wiki/List_of_HTTP_headers
http://en.wikipedia.org/wiki/Ftp
http://en.wikipedia.org/wiki/SSH_file_transfer_protocol
http://en.wikipedia.org/wiki/FTPS
http://www.eventhelix.com/RealtimeMantra/Networking/FTP.pdf
http://en.wikipedia.org/wiki/Smtp
http://en.wikipedia.org/wiki/Extended_SMTP
http://www.vanemery.com/Protocols/SMTP/smtp.html
http://en.wikipedia.org/wiki/E-mail
http://www.xs4all.nl/~ace/X-Faces/
http://content.hccfl.edu/pollock/Unix/EmailNotes.htm
http://www.opinionatedgeek.com/dotnet/tools/Base64Decode/Default.aspx
http://technet.microsoft.com/en-us/library/cc959828.aspx
http://technet.microsoft.com/en-us/library/cc959829.aspx
http://technet.microsoft.com/en-us/library/cc959833.aspx
http://technet.microsoft.com/en-us/library/cc959827.aspx
http://technet.microsoft.com/en-us/library/cc958813.aspx
http://www.iana.org/assignments/message-headers/perm-headers.html
http://209.85.135.132/search?q=cache:vcwrTogaB9UJ:www.avolio.com/columns/E-mailheaders.html+x+header+smtp&cd=1&hl=en&ct=clnk&gl=hu

http://gsexdev.blogspot.com/2004/08/using-x-headers-with-exchange.html
http://en.wikipedia.org/wiki/Ftp
http://en.wikipedia.org/wiki/Telnet
http://en.wikipedia.org/wiki/Secure_Shell
http://en.wikipedia.org/wiki/Pop3
http://de.wikipedia.org/wiki/SMTPS
http://technet.microsoft.com/en-us/library/cc754104(WS.10).aspx
http://technet.microsoft.com/en-us/library/bb430764.aspx
http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/809552a3-3473-48a7-9683-c6df0cdfda21.mspx?mfr=true
http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/be052923-6022-4007-833f-587c2fa33e78.mspx?mfr=true
http://msdn.microsoft.com/en-us/library/cc251568(PROT.13).aspx
http://technet.microsoft.com/en-us/library/cc730981(WS.10).aspx
http://technet.microsoft.com/en-us/library/cc733010(WS.10).aspx
http://www.javvin.com/protocolHTTPS.html
http://en.wikipedia.org/wiki/Ftps
http://en.wikipedia.org/wiki/E-mail_encryption
http://en.wikipedia.org/wiki/Pretty_Good_Privacy
http://en.wikipedia.org/wiki/S/MIME
http://en.wikipedia.org/wiki/STARTTLS
http://www.windowsecurity.com/articles/Securing_Data_in_Transit_with_IPSec.html
http://en.wikipedia.org/wiki/IPsec
http://en.wikipedia.org/wiki/Internet_Key_Exchange
http://en.wikipedia.org/wiki/AuthIP
http://en.wikipedia.org/wiki/ISAKMP
http://www.iana.org/assignments/protocol-numbers/
http://blogs.technet.com/rrasblog/archive/tags/IKEv2/default.aspx
http://blogs.technet.com/rrasblog/archive/tags/SSTP/default.aspx
http://www.microsoft.com/downloads/details.aspx?familyid=7E973087-3D2D-4CAC-ABDF-CC7BDE298847&displaylang=en
http://en.wikipedia.org/wiki/Sstp
http://technet.microsoft.com/en-us/library/cc771298(WS.10).aspx
http://www.isaserver.org/tutorials/Configuring-TMG-Beta-3-SSTP-VPN-Connections-Part1.html
http://blogs.technet.com/rrasblog/archive/2007/01/10/how-sstp-based-vpn-connection-works.aspx
http://en.wikipedia.org/wiki/RADIUS
http://en.wikipedia.org/wiki/One-time_password
http://blogs.isaserver.org/pouseele/2006/12/26/playing-with-radius-authentication-and-isa-server-2006/
http://en.wikipedia.org/wiki/Base64
http://en.wikipedia.org/wiki/MIME
http://technet.microsoft.com/en-gb/magazine/2009.07.cableguy.aspx

6 JAVÍTÁSOK

Hah... milyen szép tiszta fehér még ez az oldal.

7 A SZERZŐ



Itt szeretnék adni az érzésnek. Mármint az exhibicionizmus nevűnek.

Valamikor vegyészmérnöknek készültem, sőt, meglepő módon még diplomát is kaptam belőle. Pár évnek kellett csak eltelnie és társaságban már azt bizonygattam, hogy a kén-monoxid sokkal veszélyesebb, mint a kén-dioxid. Ha jót akarsz magadnak, nem engem kérdezel meg arról, mely anyagok mérgezőek.

Valójában soha nem is tekintettem magamat vegyésznek: az utolsó két évet a veszprémi egyetem kibernetika tanszékén töltöttem - és aki ismeri a helyet, tudja, hogy akkoriban arrafelé tanyáztak az égőszeműek, az elhivatott rendszeresek... azaz a Rendszermérnök szakos hallgatók. Gyakorlatilag nekünk csak a testnevelés és a nyelvi óráinkon nem volt matek - az összes többin tisztán. Modelleztünk (matek), optimalizáltunk (lineáris/nemlineáris algebra), mindezeket számítógépre vittük (numerikus matek), elemeztük (statisztika, valószínűségszámítás)... kikapcsolódásképpen ipari/általános számítástechnikát, ipari/kisgépes/nagygépes programozási nyelveket tanultunk. A hab a tortán a fizikai kémia volt, melyet gondolom az egyetem vegyipari jellege miatt csempésztek bele álcázásképpen a tanrendbe, de nálunk azt is statisztikus alapon oktatták.

Ehhez képest az első munkahelyemen, a veszprémi IKV távfűtési részlegén mindösszesen egy darab C128-as számítógép volt. Arra kellett ügyviteli szoftvereket. fejlesztenem. Mondjuk, még jó, hogy nem a főnök Casio menedzserkalkulátorára[14].

[14] A geek vonal egyébként már korán kiütközött. Az első programozható zsebszámológépem - Stylandia, egy szégyentelen tajvani koppintás - 48 lépést ismert. Erre gyártottam különböző programokat, úgy, hogy a gombnyomásokat cetlikre írtam fel. Különösen büszke voltam a másodfokú egyenlet megoldóképletére - hé, mondtam már, hogy 48 lépés? - és a Stirling formulával megvalósított faktoriális számolásra. Ez utóbbi ugyanis képes volt a 69-nél nagyobb számok esetében is faktoriálist számolni.

Aztán jött a rendszerváltozás, a tanácsi rendszer felbomlott, az IKV szintén - és ahogy a távfűtés önálló cég lett, megszabadultunk az összes kontraszelektált idiótától. Innentől kezdve tényleg informatikával foglalkoztam, rendszerépítés, ügyvitelszervezés, üzemeltetés, programozás (Clipper/Pascal), hálózat. Ahogy egy kis cégnél szokás az ilyesmi.

Az OOP már az üzemeltetési oldalon talált[15]. Nem szándékosan álltam át, egyszerűen a következő munkahelyemen erre a tudásra volt szükség. Ha fejlesztőt kerestek volna, akkor ma fejlesztenék.

A szakmai karrierem 2002-ben indult be igazán, amikor egy csoportos leépítés előjátékaként kirúgtak a munkahelyemről. Jó tíz hónapig kerestem új munkahelyet. Ekkor fogadtam meg, hogy ilyesmit többször nem engedélyezek a sorsnak. Rágyúrtam a szakmára, tanultam, mint az őrült, sorra nyomtam a - valódi - vizsgákat, a cégemnél is szépen haladtam előre, kaptam az egyre izgalmasabb feladatokat. 2005 körül vágtam bele egy szakmai/civil blogba, az itteni szakmai írások keltették fel később a Microsoft figyelmét. Így lettem a Technet Magazin egyik rendszeres szerzője. Innentől szépen apránként épültek egymásra a dolgok: cikkek, blogbejegyzések, oktatási anyagok... közben egy MVP cím... majd egy Exchange 2007 könyv, mely eredetileg Netacademia produkciónak indult, de a kézirat végül a Microsoft Magyarországnál landolt. Erre a mostani könyvre már a Microsofttól kaptam a megbízást - és ahogy most kinéznek a dolgok, valószínűleg nem ez lesz az utolsó.

Végül álljanak itt az elérhetőségek:

Szakmai blog:

EMAIL ÉS A DETEKTÍVEK	:	http://emaildetektiv.hu
MICROSOFT TECHNET PORTÁL	:	http://tinyurl.com/ydmbgdk

Privát blog

MI VAN VELEM?	:	http://mivanvelem.hu

Email:

jpetrenyi@gmail.com

[15] Ha csak nem számítjuk a kicsit bénácska objektumorientált Clippert, a CA-VO-t.