Tartalom

Címlap: Puskás Tivadar, a telefonhírmondó feltalálója

Infokommunikációs rendszerek biztonsága

buttyan@crysys.hu
szabo@hit.bme.hu

Néhány évtizeddel ezelőtt a számítógépes rendszerek és a kommunikációs hálózatok tervezésénél a biztonság nem volt elsődleges fontosságú kritérium, mivel mind a felhasználók, mind pedig az alkalmazások száma erősen korlátos volt, és visszaélésre, támadásra alig akadt példa. Mára a helyzet gyökeresen megváltozott. A különböző infokommunikációs rendszerek a szó szoros értelmében mindenhol jelen vannak, körülvesznek bennünket, átszövik mindennapi életünket. A felhasználók száma nagyságrendekkel nőtt, egyes új alkalmazások pedig az adott rendszer tervezésénél meg sem fordultak a tervezők fejében, gondoljunk csak az Interneten keresztül banki tranzakciók lebonyolítására, vagy az SMS szolgáltatás használatára elektronikus fizetéshez. Mindez sajátos helyzetet teremtett: egyrészt megnőtt a biztonsági követelményeket is támasztó alkalmazások száma, másrészt – az elterjedtség mértékének növekedésével – megnőtt a potenciális támadók száma is; ugyanakkor az alkalmazásokat befogadó legacy rendszerek alapvetően semmilyen védelemet nem nyújtanak. Nyílvánvalónak látszik tehát, hogy a fejlődés nem folytatódhat a biztonsági problémák megnyugtató megoldása nélkül. Az elkövetkezendő évtized minden bizonnyal a biztonság évtizede lesz az infokommunikációs rendszerek kutatása, tervezése és építése területén.

Ez adja a Híradástechnika 2006. évi májusi tematikájának aktualitását is. Ez a szám 9 cikket tartalmaz, melyek az aktuálisan legfontosabbnak tartott biztonsági kérdések széles skáláját lefedik. Természetesen a teljes spektrum tárgyalása az adott keretek között nem lehetséges. A cikkek egy része ismeretterjesztő jellegű és átfogó képet igyekszik adni egy szűkebb területről, másik részük egy-egy fontos területhez kötődő új kutatási eredményről számol be.

A kéretlen reklámlevelek, a spamek sokunk életét megkeserítik. Szabó Géza és Bencsáth Boldizsár cikke bemutatja, hogyan juthatnak a spamerek e-mail címünkhöz, még akkor is, ha azt sehol nem publikáljuk. A cikk javaslatot tesz a DHA (Directory Harvest Attack) technika elleni védekezésre is, és bemutatja a szerzők által tervezett védelmi rendszer működését a gyakorlatban.

Sok laikus tesz egyenlőségjelet a hálózatbiztonság és a tűzfalak közé. A tűzfalak valóban fontos szerepet játszanak a védelem rendszerében, ám áttörésük nem lehetetlen feladat. Szabó István cikke bemutatja, hogy egy külső támadó hogyan tudja kideríteni a tűzfal által használt szűrési szabályokat, melyek megszerzése egy támadás első lépése lehet.

A vírusok és egyéb kártevő programok több évtizede keserítik már életünket, mégsem sikerült még megszabadulnunk tőlük. Sőt, a hálózatok terjedésével, a vírusok és férgek új erőre kaptak, s még gyorsabban képesek terjedni. Adamkó Péter cikke átfogó képet ad a kártékony programok elleni küzdelem mai helyzetéről és a védekezésben használható korszerű eszközökről.

A vírusok és férgek gyakran a rendszerben futó hasznos programok hibáit, hiányosságait használják ki arra, hogy átvegyék az uralmat a rendszer felett. Tóth Gergely és Hornák Zoltán cikke egy olyan módszert mutat be, amellyel tetszőleges programban azonosíthatók a biztonsági szempontból veszélyes hibák. A módszert megvalósító Flinder keretrendszer két lehetőséget biztosít a programozói hibák kiszűrésére: a forráskód alapú tesztelést és az úgynevezett black-box tesztelést (amikor a forráskód nem ismert).

A felhasználók mobilitását támogató vezeték nélküli hálózatok egyre nagyobb népszerűségnek örvendenek. A vezeték nélküli hálózatok azonban általában sebezhetőbbek, mint vezetékes társaik, ezért a biztonságos működés biztosítása még nagyobb hangsúlyt kap. Buttyán Levente és Dóra László cikke áttekintést ad az elterjedten használt WiFi technológia biztonsági kérdéseiről, s a kapcsolódó szabványosítási folyamatokról.

Sok mobil terminál (például telefon vagy PDA) számítási kapacitása korlátozott, ezért nem képes nagy számításigényű kriptográfiai algoritmusok futtatására. Szentgyörgyi Attila és Szüts Péter Lóránt e probléma megoldására tesznek javaslatot cikkükben. Megoldásuk lényege, hogy a nagy számításigényű kriptográfiai algoritmusokat nem a mobil eszközön hajtják végre, hanem egy speciális szerveren, melyet Security Proxy-nak neveznek.

Az elektronikus aláírás elméleti alapjai régóta ismertek. E technológia mégis csak az elmúlt években kezdett elterjedni hazánkban. Ennek egyik oka az, hogy a gyakorlati alkalmazás során számos probléma merül az elektronikus aláírással kapcsolatban. Berta István Zsolt cikke ezen problémákba nyújt betekintést, külön kitérve az elektronikus aláírás hazai helyzetére, többek között a közelmúltban közzétett elektronikus aláírás keretrendszerre.

Az elektronikus aláírás segítségével biztosítható az elektronikus tranzakciók hitelessége és letagadhatatlansága. Egy szolgáltató számára ezek fontos követelmények, ám a felhasználók érdekei gyakran ellentétesek. A felhasználók sokszor anonim módon szeretnék a szolgáltatásokat igénybe venni, amelynek biztosítására különböző anonymizer rendszerek jelentek meg. Szentpál Zoltán és Zömbik László cikke azt vizsgálja, hogy mennyire hatékonyak ezek a rendszerek, és bemutat néhány támadási lehetőséget.

Összefoglalva tehát e tematikus szám olyan kurrens biztonsági témákkal foglalkozik, mint a spam, a tűzfalak, a vírusok és egyéb kártékony programok, a programozói hibák és azok detektálásának lehetőségei, a WiFi hálózatok biztonsági kérdései, a PKI és a digitális aláírás gyakorlati problémái, valamint az anonim kommunikáció kérdése.

Buttyán Levente,
vendégszerkesztő

Szabó Csaba Attila,
főszerkesztő

DHA támadás elleni védekezés központosított szűréssel

SZABÓ GÉZA, BENCSÁTH BOLDIZSÁR

BME Híradástechnikai Tanszék, CrySyS Adatbiztonsági Laboratórium
{szabog, boldi}@crysys.hu

Lektorált

Kulcsszavak: DHA, címkinyerő támadás, feketelista, DNS, központosított védekezés

Cikkünkben a spamvédelmi módszerek területén végzett kutatásainkat és fejlesztési terveinket, eredményeinket kívánjuk vázolni. A tervezett védekezési módszerek komponens alapú fejlesztések, egymással szorosan összefüggő módszerek, melyek egymás szoftverelemeit is jelentős mértékben felhasználják. Elemezzük a rendszerünk által összegyűjtött adatokat és bemutatjuk, hogy milyen tipikus DHA támadók vannak, illetve hogy ezeket egyértelműen meg lehet-e különböztetni egymástól pusztán a támadási statisztikák alapján.

Az elektronikus levelező-szerverek által karbantartott levélcímek megszerzésének egyik lehetséges módja a címkinyerő támadás (Directory Harvest Attack), melynek során egy támadó létező e-mail címeket kísérel meg összegyűjteni címek próbálgatásának segítségével. A levelező szerverek, amennyiben egy érkező levél nem az általuk karbantartott felhasználók címére lett küldve, úgy vagy azonnali, vagy későbbi visszajelzést adhatnak arra nézve, hogy a kapott levélben szereplő felhasználó postafiókja nem létezik a nyilvántartásukban. Ez a folyamat információval szolgál a levelező-szerver által karbantartott e-mail címekről. A támadók ezt az információt használják ki, nagy számú levelet küldve az adott e-mail szervernek. Azokról a címekről, amelyekről nem érkezik válasz, azaz a szerver negatív visszajelzés nélkül elfogadja a levelet, nyilvántartást vesznek fel. Ezek a címek minden valószínűség szerint érvényes felhasználói azonosítókhoz tartoznak, így érdemes lehet rájuk a későbbiekben kéretlen leveleket küldeni.

A DHA támadás lehetősége ismert volt eddig is, ám a sok alternatív levélcím összegyűjtési lehetőség miatt eddig nem kapott kiemelt fontosságot a kéretlen levélküldők által célpont összegyűjtés során használt eszközök között. Ahogy a felhasználók egyre jobban vigyáznak e-mail címeikre, a DHA előtérbe került és a támadók elkezdték előszeretettel alkalmazni.

A cikkben számba vesszük a lehetséges védelmi megoldásokat és ezen alapelveket felhasználva bemutatjuk az általunk implementált több programkomponensből álló, hálózaton keresztül együttműködő rendszert. A rendszer egy feketelistát felhasználó megoldás, ahol a feketelista a támadók IP-címeit tartalmazza egy központi adatbázisban. A levelezőszerverek védelmét egy támadást bejelentő modul és a támadó levélküldését megakadályozó front-end modul látja el. A feketelista szerver tulajdonképpen egy DNS kiszolgáló, ahova a kliensektől érkező bejelentések és lekérdezések is DNS lekérdezés formájában utaznak. A DNS lekérdezésre a szerver egy IP-címmel válaszol, ami a kliens oldalon jelentheti akár azt, hogy a kérdezett IP támadóhoz tartozik, akár azt, hogy ártatlan. A meglevő rendszerek mellé beépítve, azoknál erőforrás megtakarítást lehet elérni, mivel a DHA támadók levelei nem kerülnek a lassabb, erőforrás igényesebb tartalomszűrő mechanizmusok rostája alá, korábban ki lehet tiltani őket a rendszerből. A rendszer valós környezetben való helytállása bizonyítja, hogy megfelelően működik és az alkalmazott védelemi módszer eredményes. A cikkben ezen valós rendszer által összegyűjtött eredményeket elemezzük és rendszerezzük, bemutatva, hogy milyen lehetőségek vannak a támadások csoportosítására az összegyűjtött információk alapján.

1. Bevezetés

Az emberek az egyre növekvő kéretlen levelek áradatának, levélben terjedő vírusok és más kártékony kódok hatására egyre jobban meggondolják azt, hogy kinek is adják oda az e-mail címüket. Átgondolják, hogy meg merjék-e kockáztatni, hogy valamilyen online fórumon használják címüket, vagy akár azt is, hogy a weblapjukon rajta hagyják-e ezt a fontos személyi adatukat. Mindkét esetben ugyanattól kell tartani: a keresőrobotok képesek összegyűjteni a `mailto:user@levelcim` alakú hivatkozásokat. A cél sajnos már minden Internet-felhasználó számára nyilvánvaló: a címeket kéretlen levelek, spamek kiküldésére kívánják felhasználni. A fórumok ilyen szempontból kiemelten veszélyesek, hiszen ha kifejezetten nem korlátozzuk az e-mail címünk szerepeltetését, akkor minden hozzászólásunk mellé odakerülhet.

Egy alternatív e-mail címgyűjtési lehetőség az emberi hiszékenységet kihasználó támadási forma: a hamis oldalak és kérdőívek módszere *(phishing)*. Egy népszerű, látogatócsalogatónak tűnő oldalon valamilyen ügyes fogással (például nyereményjátékok, reklámajándékok) ráveszik az áldozatot, hogy beírja az adatait. Így garantáltan működő címeket tudnak a támadók összegyűjteni.

Ha a felhasználó a fent említett e-mail cím gyűjtési lehetőségeket kizárta, ám gondosan kezelt e-mail címére egyszer csak kéretlen levelek kezdenek el özönleni, akkor e-mail szolgáltatója nagy valószínűséggel egy címkinyerő, azaz *directory harvest attack (DHA)* támadásnak esett áldozatul.

A DHA témája sokszor előkerül, és a kereskedelmi antispam termékek egy hirtelen mozdulattal ki is pipálják az általuk nyújtott szolgáltatások listáján, elfelejtvén megemlíteni, hogy milyen módszert is használnak a támadás kivédésére. A továbbiakban a lehetséges módszereket foglaljuk össze és javaslatot teszünk egy hatékony védelmi mechanizmusra.

1.1. Miért lehetséges a címkinyerő támadás?

A DHA problémája a levéltovábbítási protokollban *(simple mail transport protocol, SMTP)* [1] gyökeredzik: az e-mail szerverek, azaz az SMTP kiszolgálók, vagy más kifejezéssel élve levéltovábbító ügynökök *(mail transport agents, MTA)*, ha megfelelő e-mail címre kapták a levelet, úgy nem adnak semmilyen visszajelzést, egyszerűen csak elfogadják azt. Azonban a szerver, ha egy nem létező címre kap levelet, úgy vagy azonnali, vagy későbbi visszajelzést adhat arra nézve, hogy a felhasználó postafiókja nem létezik.

A címkijutás mellett problémát jelenthet a levelezést kiszolgáló szerver összeomlása. Az e-mail címek megszerzése érdekében a támadó rengeteg téves levelet küld a szervernek, amely így jelentősen, hosszú időre, és akár több támadótól is leterhelésre kerül. A leterhelés leköti a kiszolgáló hálózati kapacitását és processzorát is. Ez végeredményben egy szolgáltatás megtagadási *(denial of service, DoS),* azaz a hardver vagy szoftver megbénításával, illetve működésének zavarásával a felhasználót elérhetetlenné tevő támadást eredményez.

1.2. A támadás fajtái

A DHA támadást, azaz a címlista-kinyerő támadást, két típusba sorolhatjuk: az egyik „brute force” jelleggel az összes lehetséges karakter, illetve szótag kombinációt kipróbálja, mint e-mail címet. A másik jóval szofisztikáltabb: tipikusan előforduló e-mail címeket generál vagy gyűjt emberek vezeték és keresztnevéből, gyakran előforduló szavakból, szóösszetételekből, továbbá ismert e-mail azonosítókból.

Másik lehetséges csoportosítása a DHA támadásnak a felhasznált IP-címek száma alapján történhet: az „alap” változatban a támadó ugyanarról az IP címről próbálkozik, a másik esetben több IP címmel rendelkezik és ezeket felváltva használja a támadáshoz, és ezzel egy eloszott címlista-kinyerő támadást hozva létre *(Distributed DHA)*.

2. DHA-val kapcsolatos munkák

A DHA problémája többnyire ismert, a védekezés viszont jelen pillanatban meglehetősen rendezetlen, mivel mindenki a saját maga által kifejlesztett eszközét próbálja használni. Azon cégek, amelyeket DHA nem ér, többnyire nem is védekeznek. A kereskedelmi termékek funkciójukat tekintve inkább antispam termékek, és nem a DHA támadás ellen vannak kihegyezve. Nagyrészt a valós időben frissített fekete lista *(Real-time Black/Block List, RBL)* alapú megoldásokat támogatják levél érkezésekor, azaz nyilvános RBL-listákon ellenőrzik a feladó címét, hogy támadónak minősítették-e már korábban.

A specifikusan DHA elleni védekezésre felkészített termékek közül az egyik legegyszerűbb megoldást választotta a *Kerio MailServer* [16]: felfigyel a nem létező postafiókoknak küldött levelekre és egy bizonyos szám felett elkezdi szűrni a lehetséges támadókat. A *Secluda Inboxmaster* [17] már konfigurálható SMTP hibaüzenetek beállítását teszi lehetővé, így ha egy spamet detektálnak a levél kézbesítés közben, a szerver egy válaszüzenetet küld a feladónak, hogy nem létező e-mailre próbált levelet küldeni. Ezzel a megoldással az a legfőbb probléma, hogy a DHA támadást nehezen szűri ki, hiszen az ebben a támadásban részvevő e-mailek általában nem tartalmaznak spamet, amit a spamszűrő módszerek így nem jeleznek.

Komolyabb védelmet jelent a jól bevált elemek egybeépítésével operáló a *Styx Mail Filter* [14]: itt egy hardver-szoftver együttest kap a vásárló, ami a kéretlen reklám levelek és vírusos tartalmak szűrését végzi a levelek levelező rendszerbe jutása előtt. Az alapkivitel szabad szoftvereket használ, így megtalálható benne a *ClamAV* [11] víruskereső és *SpamAssasin* [10] spamszűrő. Ez utóbbi egy RBL-alapú megoldást foglal magában, amely kiegészül egy Bayes szabály-tanuló rendszerrel, *Razor*[1] és *DCC*[2] komponensekkel, ami a levelek szűrését elvégzi, de DHA támadást nem jelent az RBL-szerverek felé. Az is előfordul, hogy a termék dokumentációja alapján nem tudni, hogyan működik, de a hatékonyság miatt, nagy valószínűséggel RBL-alapú, ilyen például az *eSafe Advanced Anti-spam Software* [15].

Az egyszer használatos e-mail címek szükségessége esetén egy lehetséges megoldást nyújthat a *mailinator* [9]. Egy azonosítás nélküli e-mail szervert valósítanak meg, ami semmi másra nem jó, mint hogy levelet fogadjon. Bármilyen címzettnek erre a címtartományára érkező levelet elfogad, aminek a postaládáját meg lehet tekinteni belépve az oldalra. A leveleket és az ideiglenes postaládákat óránként ürítik, így arra is jó lehet, hogy például egy fórumra való bejelentkezéshez szükséges megerősítő e-mailt elküldenek erre a helyre, amit megnézünk egyből, és megerősítjük belépésünket. Mivel semmilyen azonosítás nincs, ezért bárki meg

1 Vipul's Razor [12] – egy elosztott, kollaboratív, spamfelismerő és -szűrő hálózat. A rendszernek állandóan frissülő adatbázisa van a felhasználók és a rendszert használó kliensek által beküldött spamek ujjlenyomatáról, azaz azokról a levelekről, amit a felhasználók spamnek ítéltek. Egy levél vizsgálata úgy történik, hogy ellenőrzik a levél ujjlenyomatát, nem szerepel-e a Razor feketelistáján.

2 Distributed Checksum Clearinghouse [13] – A Razor-hoz nagyban hasonló megoldás, de a kliensek itt minden e-mail hash lenyomatát átküldik, és a rendszer azt a lenyomatot ítéli egy kéretlen reklám levél lenyomatának, amit nagyon gyakran jelentenek neki.

tudja nézni bármelyik postafiókot. Ezzel a módszerrel egyben egy *honeypot*[3]-ot is megvalósítanak, és egy órára visszamenőleg meg tudják mutatni, hogy ki az, aki a legaktívabban küldözget nekik levelet.

3. A lehetséges védekezések

A DHA támadás ellen szóba jöhető védekezési mechanizmusok kerülnek bemutatásra a következő részben.

3.1. Új programelemet nem igénylő módszerek

A védekezés egyik lehetséges formája, ha nem telepítünk új programokat a meglévő levelező rendszer mellé, hanem a következő pontokban bemutatott módszerek valamelyikét használjuk.

3.1.1. Egyszer használatos e-mail cím

A védekezés a DHA támadás ellen történhet egyszerűen bonyolultra választott e-mail címekkel, ami a szótáras támadás ellen ideig-óráig véd, de a környezetünk nehezen fogja tudni megjegyezni új e-mail címünket. A védekezés ezen formája brute-force támadások ellen haszontalan lehet, továbbá nem véd a cím kiszivárgásának más lehetőségei ellen.

Az e-mail címmel való védekezés másik lehetséges módja, ha egyszer használatos e-mail címet használunk. A megoldás hibája, hogy a kommunikáció elég egyoldalú lehetőségét teremti csak meg, mivel küldeni gond nélkül fogunk tudni levelet bárkinek, de ha választ is várunk egy levelünkre, akkor a válaszcímeknek léteznie kell mindenképpen. A levelezésünk tehát nem teljesen egyszer használatos, így jelentős karbantartást, adminisztrációt igényel. Teljesen egyszer használatos e-mail esetében válaszra nem számíthatunk.

3.1.2. Különleges szerver konfiguráció

Megoldás az is, ha a szervert úgy konfiguráljuk, hogy fogadjon el minden e-mailt és ne jelezzen vissza róla senkinek, a téves leveleket pedig egyszerűen eldobjuk. Ez több problémát vet fel: a levélküldők nem tudják meg, hogy a cím nem létezik, és elárasztathatják a szervert téves levelekkel. Fontos az is, hogy a legitim felhasználók sem kapnak visszajelzést a tévesen címzett levelekről. Mindezek miatt a visszajelzés letiltása nem javasolható. A legmegfelelőbb természetesen az SMTP protokoll finomítása lenne, de mit tudunk addig is tenni, amíg ez nem következik be?

3.2. Új program elemet igénylő módszerek

Ebben az esetben már valamilyen aktív komponens kerül az eddig használt levelező-rendszer mellé. Két eltérő megoldást lehet alkalmazni, illetve ezek együttesét, növelve egymás hatékonyságát.

3.2.1. Egyénileg védekező rendszer

Az egyik megoldás az egyénileg védekező rendszerek. Ekkor minden résztvevőnek van egy saját önműködő rendszere, amely a döntéseit egyéb rendszerektől függetlenül hozza. A támadásszűrést a levéltovábbítás során keletkező hibaüzenetek alapján lehet elvégezni.

Ha a támadó DHA támadás során levelet küld, akkor téves címzettnek küld e-mailt egy adott IP címről, majd később újra fog próbálkozni ugyanarról az IP címről másik tévesen címzett levéllel. Elosztott DHA esetén is általában több e-mail címet próbál ki a támadó ugyanazon IP-címről még mielőtt IP-címet váltana. Azonban van olyanra is példa, hogy nem küldenek sok levelet egy címről. A támadás elosztottsága függhet a detektált szűrési módszertől és a támadó által várt haszontól. Felmerülhet az a kérdés, hogy mennyire bízhatunk meg a feladó IP-címének valódiságában. Ha tudjuk, hogy a levelet végül egy megbízható levelező szerver továbbította felénk, akkor a szerverrel fel kellett tudnia építeni a kapcsolatot a levélküldőnek, így a címe nem lehet hamis, ha pedig nem megy keresztül több levelező szerveren, akkor velünk is képesnek kell lennie kapcsolatot kiépíteni, ami publikus cím tartományok esetén csak úgy lehetséges, ha legalább a címtartománya (subnet-je) valós (részletes vizsgálat [5]).

3.2.2. Hálózaton alapuló védelem – a javasolt rendszerünk

Másik lehetséges védelmi mechanizmus a hálózaton alapuló védelem. Ekkor a rendszer a hálózat egyéb résztvevőivel együttműködve próbál védekezni a DHA támadás ellen. Javasolt megoldásunk a következő:

Ha egy támadó egy ismeretlen címre küld egy e-mailt a megtámadott szerveren, a megtámadott szerver küld egy hibajelentést a központi szervernek. Ez a hibajelentés tartalmazza a támadó IP-címét, a kipróbált e-mail címet, és a támadás idejét. A központi szerver gyűjti ezeket a jelentéseket, és ha túllép egy küszöböt az ezen IP-ről jövő próbálkozások száma, akkor behelyezi a támadó IP-címét a feketelistára. A szerver a listára kerülés után is jegyzi a támadó kísérleteit, így nem hagyja elévülni a bejegyzést. A feketelista tartalmát csak konkrét cím vonatkozásában lehet lekérdezni a szervertől, azaz a szerver egy igen-nem választ ad arra a kérdésre, hogy egy e-mail feketelistás-e vagy sem.

A feketelistákról (Black Lists) röviden:

Eredetileg a kéretlen leveleket küldőik saját e-mail címükről küldték szét, mintha csak rendes levelet küldenének. Ezeket a felhasználókat (számítógépeket) könnyen azonosítani lehetett, majd kitiltani őket viselkedésük miatt, így a levélküldők elkezdtek levél közvetítőket *(open relay*[4]*)* használni, melyek átvették a levéltovábbítás terhét. Legújabban kompromittált kliens gépeket használnak *(zombie*[5]*)*. Ezzel elérhetik, hogy kevés

3 Általában egy, vagy több hálózati csatlakozással bíró, valamilyen sebezhető operációs rendszert és szolgáltatást emuláló rendszer. A támadók könnyű célpontnak vélik és felfedik ezáltal magukat és szándékukat, így tőlük már az éles rendszer védhetővé válik.

4 Olyan levéltovábbító szerver, mely hajlandó olyan levelek továbbítását is átvenni, hol sem a feladó, sem a címzett nem helyi felhasználó.

5 Valamilyen módon trójai programot juttatnak a felhasználóhoz, mellyel számítógépét irányítani lehet. Ezzel tudta nélkül rávehetik naponta akár több száz levél elküldésére is.

erőforrás lefoglalásával, amit egy-egy felhasználó úgysem vesz észre, összességében nagy mennyiségű kéretlen levelet küldjenek szét. Az ismert spamforrás e-mail címeket, spam küldő gépre vonatkozó IP-címeket feketelistákon szokták összegyűjteni.

Az RBL-listákból több fajta van: van ami e-mail címeket, DNS neveket, DSL címeket, open-relay szervereket, open proxy-kat [7] gyűjt. Ezek teljesen naprakészek és ugyanolyan típusú RBL szerverből több is létezik a világon elszórtan más-más szervezetek által üzemeltetve és esetleg egymástól kicsit eltérően megvalósítva. Az, hogy adott céllal és módszerrel is akár több RBL szerver létezik hasznos lehet védelmi szempontból, mert nem olyan könnyű megtámadni, kiiktatni a szervereket. Az is igaz azonban, hogy az RBL szerverek egy része lassan frissül, pontatlan adatokat tartalmaz. A rendszergazda (illetve esetleg a szoftverfejlesztő) feladata lehet, hogy az RBL listák használata során olyan kombinációt dolgozzon ki, amely megfelelően hatékony a szervezet számára (például pontozás alapú heurisztika).

A feketelistás megoldások több szempontból is szerencsések [6]: technikai szempontból segítik megakadályozni a spam bejutását, másrészt képes társadalmi nyomás kifejtésére, megalapozására a kéretlen levelekért felelős gépek, illetve szolgáltatók irányában, hiszen a letiltott berendezéseket használó legitim felhasználók kénytelenek fellépni szolgáltatójukkal szemben a rendszer megfelelő müködésének visszaállítása érdekében.

4. A javasolt rendszer működésének leírása

A javasolt rendszer egy rendszernapló-elemzőből, és egy ennek eredményét később felhasználó front-end modulból áll. Az eredményeket központi nyilvántartásban összegezzük, azaz nyilvántartjuk azokat a gépeket, amelyek DHA támadásban érintettek. A feketelistán IP-címeket tárolunk, és passzív monitorozást végzünk [2].

Ha a DHA támadó küld levelet a rendszernek, akkor a működés a következő: az első levél átmegy az ellenőrzésen, mivel az IP-címe még nem került be a szerver adatbázisába, hiszen még nem volt olyan résztvevő, aki támadást jelentett volna erről a címről. Az üzenet továbbmegy a levelező-szerverbe, ami egyfelől ellenőrzi, hogy kézbesíteni tudja-e a helyi postafiókokba a levelet. Mivel támadás esetén sikertelen a kézbesítés, a jelentés bekerül a rendszernaplóba. A rendszernapló elemző rendszere az e-mail kiszolgáló jelentéseiből valós időben megnézi, hogy a téves címzéssel rendelkező e-mailek honnan jönnek (milyen IP-címről), és ezekről részletes jelentést tesz a központi adatbázisnak.

A rendszer a DNS protokollt használja a lekérdezésekre és jelentés küldésére a védett szerverek és a RBL-szerver között. A DNS protokoll előnyei közé tartozik a robosztusság. A DNS szervere cache-mechanizmusa növeli a rendszer stabilitását, mivel ideiglenes hiba esetén is tudhat megfelelő választ adni lekérdezésekre. A DNS protokoll többnyire a tűzfal konfigurációkon is átjut, nem igényel újabb nyitott portokat. (A DNS teljesítményének részletes vizsgálatát [3]-ban lehet megtalálni.)

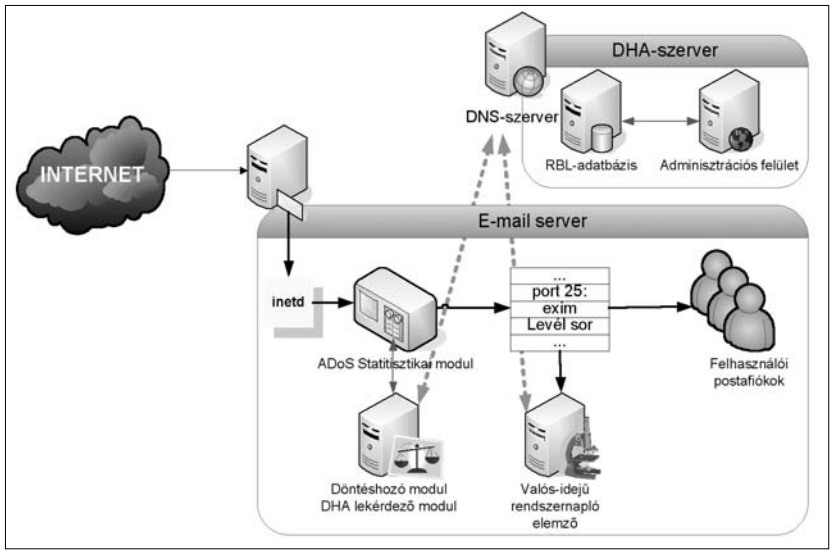
Csökkenthető azon IP-címek támadó adatbázisba kerülésének esélye, akik egyszer-egyszer csak véletlenül elgépelik a címet. A központi adatbázisban nem kerülnek be egyből a bejelentett IP-címek a támadók közé, hanem előtte az előző bejelentéseket is alapul véve a bejelentési időközök gyakoriságát kiszámolja a szerver. Ha ez egy bizonyos értéket átlép, akkor teszi át a bejelentett IP-címet a támadók listájára.

5. A javasolt megoldás implementációja

A rendszer prototípusát *(1. ábra)* linux rendszer felhasználásával hoztuk létre, standard levelezést lebonyolító megoldásokkal. Új levél érkezése esetén az *inetd* rendszerprogram működésbe hozza a levelező szervert. Ez hagyományosan a 25-ös TCP portra érkező kérésekre figyel, és elindít hozzá egy MTA-t (pl. sendmail, postfix, exim stb.) Ám a mi esetünkben nem közvetlenül adjuk át a kérést a levelező szervernek, hanem egyik modulunkon keresztül átvezetjük a kérést. Ez a modul felelős a DHA támadások kiszűréséért, azaz az ismert támadók kitiltásáért.

A DoS frontend [4] előbb statisztikai módszerekkel ellenőrzi, hogy az adott IP címről nem hajtanak-e végre DoS támadást a levelező szerver ellen, és ha ezen a szűrésen átment az IP, akkor kerül sor a DHA támadással kapcsolatos ellenőrzésre. DoS támadás érzékelése esetén a DoS frontend modul eldobja az adott támadó felől jövő TCP kapcsolatokat. A DHA támadó fe-

1. ábra A rendszer prototípusa



lől jövő levél ezután nem megy tovább a levelező rendszer felé, tehát nem kerül kézbesítésre és így nem keletkezik újabb bejelenteni való a szerver felé.

A szűrésen átment TCP kapcsolatot továbbadjuk a levelező szervernek, ami a teszt környezetekben *exim4* levelező szerver. A rendszernapló elemző a linux *syslog* naplóállományában valós időben keresi a levelező rendszer által generált jelentéseket. A különböző levelező szervereknek a rendszernapló bejegyzései eltérőek, így más-más bejegyzések vizsgálatára is fel kellett készíteni az elemző modult. A naplófájl típusát egy külső konfigurációs fájlban lehet beállítani a modul regisztrálásakor használt azonosító és titkos kóddal együtt. Az RBL-adatbázis *MySQL*-ben lett megvalósítva. Az adminisztrációs felület Apache és PHP futtató környezetet igényel. A rendszernapló elemző modult és a RBL-szerver Perl-ben lett implementálva. A rendszer működés közben is megtekinthető [18], illetve a kliens és szerver prototípus is letölthető.

6. A javasolt rendszer vizsgálata

Az alábbiakban a megvalósított rendszer más megoldásokkal kerül összevetésre, illetve a védekezési eljárás és a védett rendszerek támadhatóságát vizsgáljuk meg.

6.1. A rendszer előnyei

Több rendszer az egyéni védekezés módszerét, a gépek önálló védelmét valósítja meg, amivel az a nyilvánvaló baj, hogy ha egyetlen gépet védenek és nem egy központot használnak, akkor egy széles körben elosztott támadás ellen nem véd. Az általunk javasolt központosított megoldás a központ leállása esetén hasonló módon képes önálló döntést végrehajtani.

Több komplett antispam rendszer állítja magáról, hogy védelmet biztosít a DHA támadások ellen, de a megoldás által használt módszert nem ismerteti. Érdekes, hogy a kereskedelmi szoftvergyártók az RBL-szerverekkel együttműködnek egy levél érkezésekor, azaz kiszűrik az ismert támadók IP-címeit, de támadás esetén többnyire nem járulnak hozzá ezen listák automatikus bővítéséhez.

A rendszerünk támadást bejelentő megvalósítása is RBL-alapú és integrálható más rendszerekbe. A központi nyilvántartás segítségével a komponenseinket használó összes résztvevő profitál egymás bajából is, azaz egy támadó nemcsak egy helyen lesz kitiltható, de másoknak sem fog tudni károkat okozni, amennyiben a támadást a szűrést végző rendszeren bejelentik központunknak. A rendszer kliens oldalának megvalósítása komponens-alapú, aminek több előnyös következménye is van:

– a támadók bejelentési és a tiltási mechanizmusa különválasztható;
– egy már meglevő rendszer is kiegészíthető vele, illetve akár csak bizonyos komponenseivel, így növelve a meglévő hatékonyságát is, nem kell a teljes rendszert átalakítani;
– a komponensek transzparensek kívülről, így a kiesésük esetén nem teszik a rendszert használhatatlanná;
– a DHA védelmi komponens segíti, hogy a gépek védelme komplex, átfogó legyen és a meglévő vírus és spamszűrőket kiegészítve nyújtson védelmet.

6.2. Téves riasztások kezelése

A rendszer támadóknak tekinthet olyan levélküldőket is, akik csak véletlenül elgépelik a címet és így nem létező postafióknak küldenek levelet. A központi adatbázisba sajnos ilyen esetben is támadóként kerül bele a felhasználó által használt SMTP kiszolgáló. A téves riasztások alacsonyan tartása érdekében több módszer használható, hogy az egyedi téves levelek elválaszthatóvá váljanak a valódi támadóktól: egyrészt a központi adatbázisban az is nyilvántartható, hogy az egyes IP-címek mennyire „veszélyesek". Azaz pontozni lehet őket aszerint, hogy hány bejelentés érkezett arra az IP címre vonatkozóan.

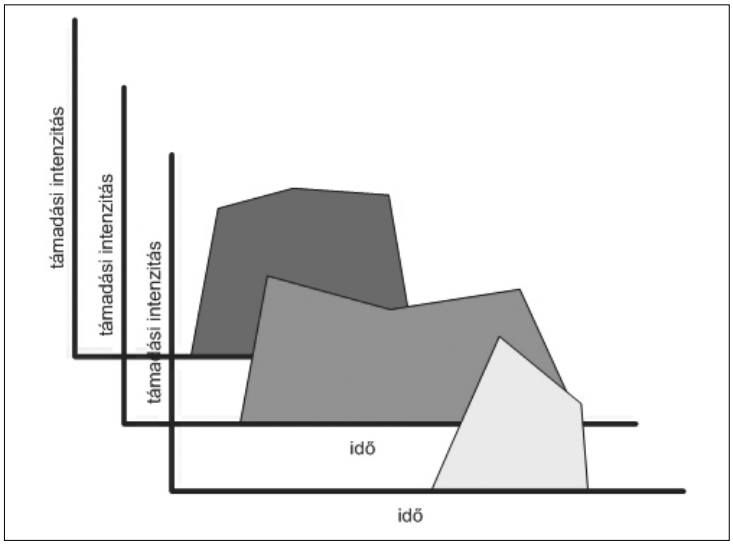
Másrészt alkalmazható öregítés *(aging)* a központi adatbázisban. Az öregítés, azaz az adat eltávolítása adott idő elteltével nagyon fontos szerepet játszik a rendszerben, ezért jól kell megválasztani a használt metódust: ha egy támadót eltávolítunk a listáról, akkor tovább támadhat, ha viszont túl sokáig van rajta, akkor akár a rendes felhasználók forgalmát is megakadályozhatja (például olyan IP címek esetén, amikor a felhasználók gyakran cserélődnek ugyanazon IP cím mögött).

A téves riasztások a rendszer legnagyobb gyengeségét jelenthetik, hiszen a nagy számú téves riasztásból adódóan előfordulhat, hogy a rendszergazda inkább kikapcsolja a védelmet. A téves riasztások problémájának megoldása emiatt a rendszer működésének szempontjából kulcsfontosságú.

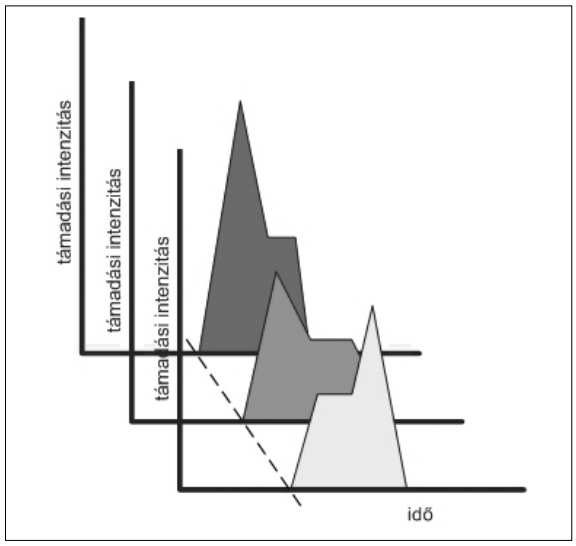
6.3. A védelem eredményessége

A központosított szűrés eredményeképpen a támadó csak nagyon korlátozott számú próbát tehet a védett címtartományokon. A tipikus támadás során sok emailt küldenek ki rövid idő alatt, így a rendszer hamar besorolja a támadókat a tiltólistára. A tiltás után a támadónak ki kell várnia az öregítést, azaz azt az időt, amíg a rendszer kiszedi az IP-t az ismert támadók listájáról, majd a folyamat újrakezdődhet. A támadó gépe egy bizonytalan végeredménnyel záródó próbálkozás alapján tiltólistára kerülhet, és utána azt nem tudja használni nagyobb haszonnal kecsegtető támadásokra sem, például spam kiküldésére. A támadónak tehát védelmünk alkalmazása esetén nem lesz érdeke a DHA támadás mindaddig, míg az abból származó haszon meg nem haladja az alternatív módszerekkel elérhető hasznot.

Természetesen a rendszerünk nem nyújt védelmet a nem védett rendszereknek, így azokon korlátlanul próbálkozhat a támadó. A védelem csökkentheti a támadó nyereségét, gazdaságtalanná téve a DHA támadást, ezáltal a nem védett rendszereknek közvetve okozhat hasznot.



2. ábra
Tudatos támadók jellemző viselkedése



4. ábra
Több vírussal fertőzött gép, mint támadók jellemző viselkedése egymáshoz képest

7. A támadók csoportosítása támadási intenzitás alapján

A DHA támadókról feltételezzük, hogy megkülönböztethetőek. Egyik ilyen megkülönböztető jegyük a támadási statisztikájuk. Megvizsgáltuk a valós rendszerekből gyűjtött támadási statisztikákat, és megpróbáltunk tipikus támadó modelleket kialakítani.

Az egyik tipikus támadó modell a tudatos támadók *(3. ábra)*, akiknek a viselkedésén látszik, hogy nem véletlenül küldenek néha egy-egy levelet, hanem nagy intenzitású, alkalmi támadást indítanak.

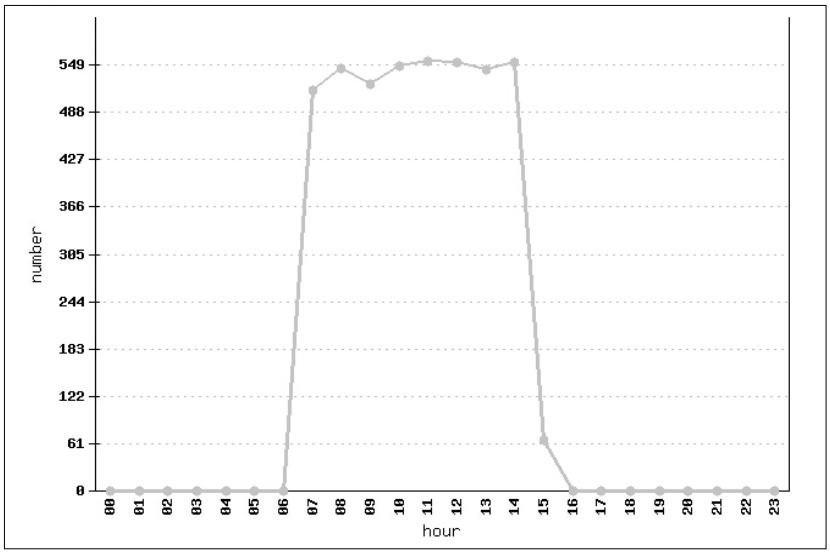
Az időpontokat megvizsgálva azt látjuk, hogy a támadók otthoni számítógépüket bekapcsolva hagyva, tipikusan munkanapokon támadnak. Hétvégén a támadások általában szünetelnek. Ekkor az internet-elérést nyílván másra is használják. Jellemző rájuk egy állandó levélküldési sebesség, mivel a sávszélességüknek egy fix hányadát használják a támadásra. Ez általában nem nagyon ingadozik, félgázzal sohasem támadnak, ha más dolguk van, akkor teljesen leállnak és akkor kezdik újra, amikor megint meg van hozzá az erőforrásuk.

Megvizsgáltunk egy-két ebbe a „tudatos” kategóriába sorolható magyarországi támadót. Az IP számhoz tartozó, úgynevezett RIPE bejegyzés alapján fix című kábelnetes és változó című, ám egy ADSL-poolba (tartományba) tartozó címekről volt szó.

Ezek nem nagyvállalatok által bérelt IP-tartományokból jöttek, és a sávszélességüknek is csak kis hányadát használták a DHA-ra. A sávszélesség kis része is elegendő volt azonban napi 5-6 ezer levél küldésére, ami körülbelül 500 levél/óra sebességet feltételez. Több tudatos támadó között a támadás időpontját illetően általában nincs kapcsolat, ez látható a *3. ábrán* (jelentősége főként a többi ábrával összevetve érthető meg.)

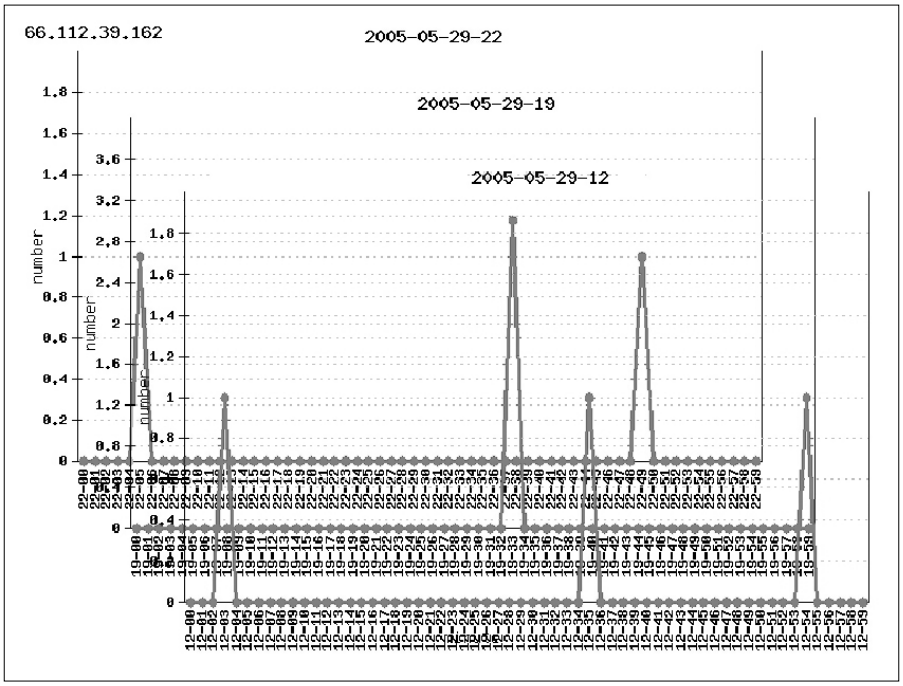
A *vírusokkal fertőzött, így a vírus által támadó gépek* és a trójaiakon keresztül távirányított gépek (ún. zombie-k) levélküldési sebessége kategorizálásunk szerint nagyon ingadozó *(4. ábra)*.

3. ábra Egy tudatos támadó napi statisztikája



Ennek egyik oka, hogy nagyon elterjedtek, valamint hogy hatalmas mennyiségű zombie-gép áll a támadók rendelkezésére, amelyek sávszélességei elég változóak. A támadó nagy mennyiségű gépet irányít, ezeket úgy próbálja meg felhasználni, hogy igyekszik jól kiaknázni azok erőforrásait, ugyanakkor a támadás a lehető legnagyobb mértékben elosztott legyen.

A támadás viszonylag szabályos időközönkénti felbukkanása védett rendszerünkben belső időzítő, illetve koordináció meglétét sugallja. Az időzítés alapján feltételezhető, hogy azonos vírusok azonos időpontban indíthatnak támadást a rendszer ellen. Ezt szemlélteti a következő oldali *5. ábra*.

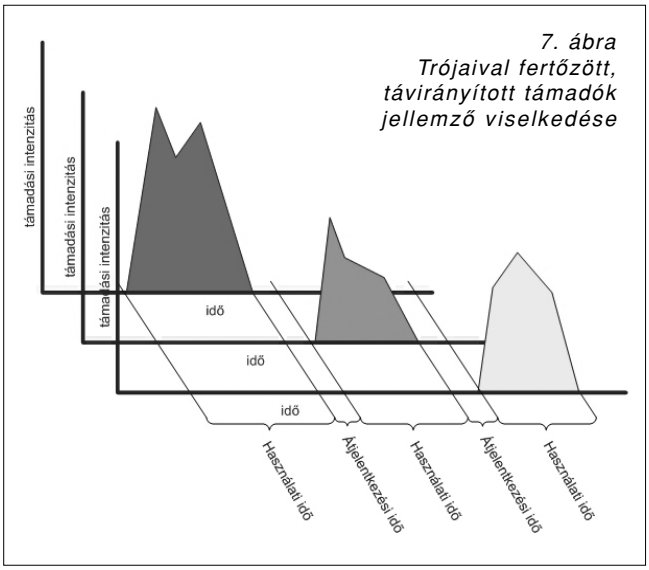


5. ábra
Ugyanazon támadó 3 különböző órában mutatott aktivitása

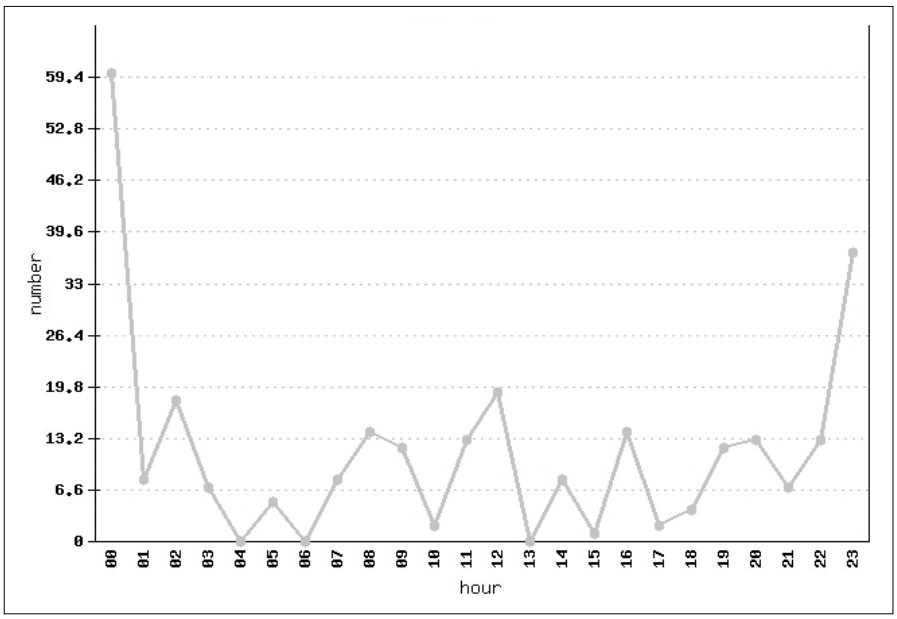
A másik oka a levélküldési sebesség ingadozásának az a cél, hogy jelenlétük rejtve maradjon a tulajdonos gépén, így célszerűen ne zavarja a rendes munkája során a felhasználót, hanem az amúgy is rengeteg szabadidőben használjon mind számítási, mind hálózati erőforrásokat. Egy feltehetőleg vírussal fertőzött gép napi támadási statisztikáját mutatja a *6. ábra*, ahol jól látszik az ingadozó levél küldési sebesség.

A *trójain keresztül távirányított gépek* is főleg napközben aktívak, azonban a távirányító ismeretlen, ezért nem tudni milyen időzónában tartózkodik, így a távirányított gépek időzónáinak összevetése nem sok információval szolgálhatna. Mivel az azonos időzónában levő gépek nagy valószínűséggel egyszerre aktívak, ezért nem célszerű a támadóknak teljesen más időzónában levő gépeket irányítani. Egy támadó feltételezhető aktivitását a zombie-gépein szemlélteti a *7. ábra*.

Eddig a trójaiak egy nagyon kezdő szintű felhasználásáról volt szó, sajnos azonban ezen a területen is vannak *professzionális támadók,* akik az általuk irányított zombie-számítógépeket nagyfokú, jól kialakított koordinációs rendszeren keresztül működtetik, amit az irodalomban többnyire „botnet”-nek, azaz robotok hálózatának hívnak. Ez a támadási módszer lehetővé teszi a támadóknak, hogy egyszerre ne csak egy trójait irányítsanak, hanem rengetegnek adjanak parancsot, ez tehát egy alternatív megoldás egy elosztott DHA támadás indítására az időkoordinációs módszer mellett.



7. ábra
Trójaival fertőzött, távirányított támadók jellemző viselkedése



6. ábra
Vírussal fertőzött gép támadásának napi statisztikája

Távirányított elosztott DHA-ra jó példa a rendszerünket ért egyik nagy arányú támadás, ami a *8. ábrán* látható. Az azt megelőző, illetve rákövetkező órákban átlagosan 1000 támadó szándékú levél érkezett, míg aztán 3 órakor hirtelen majdnem 9000! A támadások vizsgálatához a rendszerünk szűrési funkcióját kiiktattuk, hogy az ne befolyásolja a megfigyelés folyamatát (a támadó ne észlelje levelei szűrését). A támadók jellemzőinek összefoglalását láthatjuk az *1. táblázatban*.

Támadó típusa	Időpont	Relatív aktivitás	Intenzitás	Támadás sebessége	Jellemző aktivitás
tudatos támadó	napközben	nincs összefüggés	egyenletes egy támadót tekintve	akár 500 óránként	*3. ábra*
vírus okozta támadás	bármikor	egy időzónából egyszerre/ nincs összefüggés	még egy támadót tekintve is ingadozik	1-2 óránként	*4. ábra*
távirányított támadó	napközben	időben egymáshoz képest eltoltan	még egy támadót tekintve is ingadozik	10-20 óránként	*7. ábra*

1. táblázat A támadók lehetséges besorolása és jellemzőik

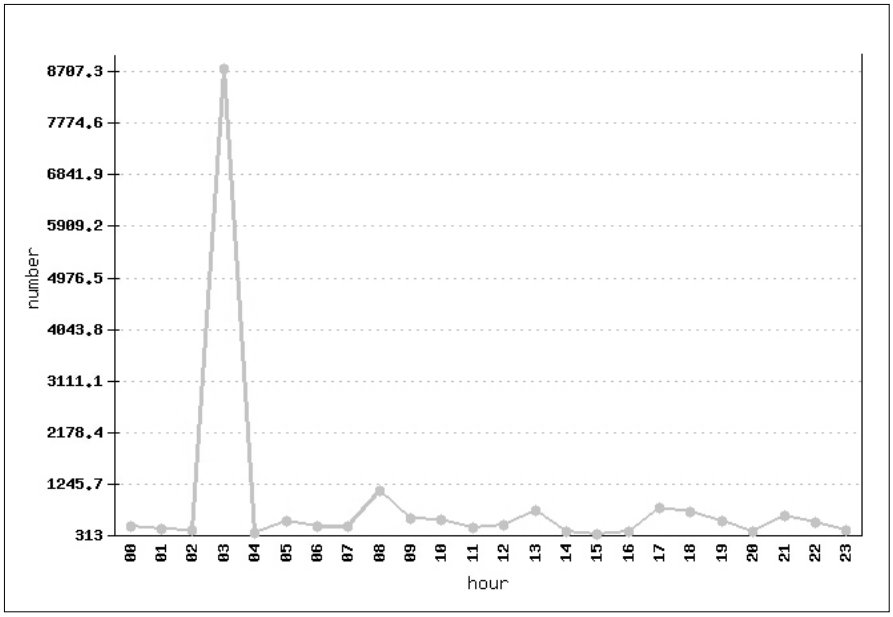
8. Összefoglalás

A cikkben megvizsgáltuk a DHA támadásokat, melyek brute-force jellegű támadások egy levelező-szerver által karbantartott e-mail címek kinyerésére. Számba vettük a lehetséges védekezési technikákat és javasoltunk egy új megoldást a támadások kiszűrésére.

A javasolt megoldásnál a rendszerünk a hálózat egyéb résztvevőivel együttműködve védekezik a DHA támadás ellen, úgy hogy az egyes e-mail kiszolgálók egy központi szerver által karbantartott RBL-listát töltenek fel feltételezett támadókról, valamint ezt a listát kérdezik le új kapcsolatfelépítés engedélyezése előtt. Ezek figyelembevételével bemutattuk az általunk kialakított komponensekből felépülő védelmi mechanizmust, és elemeztük azt.

A rendszerünk felépítése a következő: egyrészt áll egy rendszernapló elemzőből és egy SMTP szervereken működő szűrő komponensből. Másik része egy szerver alkalmazás, ami a szűrők által szolgáltatott eredményeket központi nyilvántartásban összegzi, azaz nyilvántartja azokat a gépeket, amelyek DHA támadásban érintettek. A rendszer által szolgáltatott adatokat alapul véve csoportosítottuk a támadókat és a fenyegetettségeket, amit a levelező szerverekre jelentenek.

8. ábra DHA eredménye egyik rendszerünk ellen



Irodalom

[1] J. Klensin: Simple Mail Transfer Protocol 2001, RFC 2821.

[2] Jaeyon Jung, Emil Sit: An Emprical Study of Spam Traffic and the Use of DNS Black Lists 2004.

[3] Jaeyon Jung, Emil Sit, Hari Balakrishnan, Robert Morris: DNS performance and the effectiveness of caching IEEE/ACM Transactions on Networking, 10(5), October 2002.

[4] Bencsáth Boldizsár, Vajda István: Ados frontend 2005. január

[5] Joshua Goodman: IP Addresses in Email Clients, Microsoft Research, Redmond, WA 98052.

[6] Terri Oda, Tony White: Developing an Immunity to Spam, Carleton University

[7] Open Relay Database, http://www.ordb.org

[8] Distributed Sender Blackhole List, http://dsbl.org

[9] Mailinator, http://www.mailinator.com/mailinator/index.jsp

[10] The Apache SpamAssassin Project, http://spamassassin.apache.org/

[11] Clam AntiVirus, http://www.clamav.net/

[12] Vipul's Razor, http://razor.sourceforge.net/

[13] Distributed Checksum Clearinghouse, http://www.rhyolite.com/anti-spam/dcc/

[14] Styx Mail Filter – vállalati levelezőszerverek védelmére kifejlesztett integrált hardver- és szoftver megoldás, http://www.albacomp.hu/sajtokozlemeny.asp?szam=25 (2005. január)

[15] eSafe Advanced Anti-spam Software, ftp://ftp.ealaddin.com/pub/Marketing/eSafe/White_paper%/WP_eSafe_Anti_Spam/esafe_antispam_whitepaper.pdf

[16] Kerio MailServer, http://www.kerio.com/kms_antispam.html

[17] Secluda Inboxmaster, http://press.arrivenet.com/tec/article.php/308157.html

[18] VIRUSFLAGS (a rendszer működő prototípusa), http://www.virusflags.org

Tűzfalszabályok felderítése

SZABÓ ISTVÁN
KFKI-LNX Zrt.
szabo.istvan@kfki-lnx.hu

Kulcsszavak: firewalk, tűzfal felderítés, tűzfalszabály felderítés, hálózati felderítés, portscan

Jelen publikáció a hálózati felderítés egy konkrét területére fókuszál: a tűzfalszabályok vizsgálatára és felderítésére. A szükséges elméleti háttér után bemutatjuk, hogy milyen módszerek állnak rendelkezésre a szabályrendszer megismeréséhez, részletesen elemezzük a FireWalk technikát, ismertetjük ennek a javasolt kiterjesztését, és végül a lehetséges ellenlépéseket.

1. Bevezetés

Napjainkban sajnos nehéz a számítógépes bűnözők élete. Temérdek biztonsági berendezés, tűzfal, egyre jobban képzett rendszergazdák hada keseríti meg az életüket. Egy jól képzett támadó azonban mindenre képes: elszánt, egy lépéssel a jó oldal előtt áll és gondosan előkészül az akciókra.

Sun Tzu, az emberiség történelmének egyik legnagyobb hadvezére óta tudjuk: *„Ha ismered ellenségedet és ismered önmagad, nem kell félned száz csatától sem."* A támadás tervezésének szerves része a célpont minél jobb megismerése. Egy profi hacker vagy cracker is ennek fényében jár el. Írásom célja annak bemutatása, hogyan tudjuk ezt minél jobban megakadályozni.

Legfőbb célunk az elméleti és gyakorlati ismeretek átadása mellett az, hogy felhívjuk a figyelmet a hálózati biztonság aktualitására. A crackerek, vírusok, trójai programok nagy része akár több éve ismert elvek, megoldások segítségével is elhárítható lenne. Nem lenne szabad utat engedni a céltalan rombolásnak.

2. Elméleti háttér

A szabály-felderítési módszerek elemzése előtt foglaljuk össze a lényegi részt, a FireWalk technika feldolgozásához szükséges specifikus tudnivalókat.

Szó lesz az ICMP üzenetekről, IP TTL mezőről, az ehhez kapcsolódó algoritmusokról, valamint a tűzfalak típusairól. Végül bemutatjuk a felderítés alapeszközét, a portscan-t (portpásztázás). Ez utóbbi módszer használható egy célgép nyitott, illetve zárt portjainak a feltérképezéséhez, ha nem számolunk tűzfal jelenlétével. Amennyiben a célgép tűzfallal van védve, akkor csak a nyitott portokat lehet meghatározni ezzel a technikával. A zárt portokról nem lehet eldönteni, hogy a célponton van-e zárva, vagy az őt védő tűzfal bontja a kapcsolatot.

2.1. ICMP

Az ICMP (RFC 792 vagy pl. [5]), az Internet Control Message Protocol rövidítése. Az IP-re épülve végez hibajelzési és hibakeresési feladatokat. Az ICMP üzenet IP csomagba van ágyazva, az ICMP fejléc az IP fejléc után következik.

Feladata, hogy különböző jelzéseket adjon a hálózat, illetve eszköz működéséről. Az üzenetek nagy része automatikusan keletkezik, ilyen lehet például, ha a célpont nem elérhető. Létezik olyan üzenet is, amit főleg diagnosztikára használunk, ilyen az echo-request és reply, melyet a ping programmal küldhetünk diagnosztikához.

Az ICMP üzenetek az információt két mezőben szállítják. A jelzés fő kategóriáját a típus (type) határozza meg. Ezen belül pontosít a kód, utóbbit nem minden típus esetén használják.

A FireWalk-hoz kapcsolódó ICMP típusok:

- *3-as típus –*
  destination unreachable (cél elérhetetlen).
  Ilyen üzenet több okból keletkezhet:
  ha a routing-táblában nincs meg a célhálózat,
  illetve számunkra a fontos eset az,
  amikor tűzfal szabály tiltja az adott forgalmat.
  Ritka, hogy a tűzfal ilyet küldjön,
  biztonsági szempontból nem javasolt,
  elégtelen konfiguráció esetén azonban előfordul.
- *11-es típus –*
  time-exceeded (időzítő lejárt).
  Ezt akkor küldik a routerek, amikor egy csomag
  TTL (time to live – „életkor") értéke 0-ra csökken.
  Az eldobó router így jelez a küldőnek,
  hogy nem ért célt az általa kibocsátott IP csomag.
  Fontos, hogy az ICMP üzenetet
  nem az előző router kapja meg,
  hanem a tényleges forrás (tipikusan egy PC)!

2.2. IP TTL

A TTL mező része az IP fejlécnek, 1 byte hosszú, így értéke 0-255 lehet. Az IP protokoll tervezőinek az volt a célja, hogy a route-olt hálózatban egy esetleges konfigurációs hiba esetén ne keringjenek a végtelenségig a csomagok.

Ezért implementálták a TTL mezőt, melynek értéke induláskor egy magas szám (tipikusan 128 vagy 255),

ezt az értéket minden router eggyel csökkenti. Ha 0-ra csökken, akkor a csomag eldobásra kerül. Ez a legtöbb esetben véd a hurkok ellen és az egy byte is elégnek bizonyul, hiszen még az Interneten is minden elérhető maximum 20-40 ugrással.

TTL=0 esetén a router, amelyiknél lejárt az időzítő, értesíti a küldőt arról, hogy nem ért célba a csomag és a hiba okáról is tudósít. Mindezt az információt egy ICMP time-exceeded (type 11, code 0) üzenet formájában küldi el.

2.3. Portscan

Mielőtt a támadás bekövetkezne, szükséges az egyes hostok által nyitva tartott TCP portok azonosítása, az általuk nyújtott szolgáltatások feltérképezése. Erre alkalmas módszer a portscan [3].

A támadó egymás után csatlakozik a célpont portjaihoz és megnézi, hogy ezek közül melyek vannak nyitva. Ezt teheti egyszerű TCP kapcsolat-felépítésekkel is, ennek azonban több hátránya van: lassú és biztosan felkelti a rendszeradminisztrátorok, esetleg egy IDS/IPS figyelmét.

Ezért ennél kifinomultabb módszerek születtek a portok feltérképezésére. A 80-as, 90-es évek hackerei nem kevés kreativitással egészen zseniális módszereket dolgoztak ki arra, hogyan lehetne ezt a célt kevés erőforrással, minél nehezebben detektálható módon végrehajtani.

A lehetőségek száma szinte végtelen, jelen cikknek nem is célja, hogy sorra végignézzük az összes módszert. A lényeg megértéséhez azonban hasznos lehet egy áttekintés a legismertebb, legeredményesebb módszerekről.

Ahogy azt már említettük, a portscan ugyan kiválóan alkalmas a nyitott portok felderítéséhez, de nem képes különbséget tenni a zárt és szűrt portok között, ha a forgalom tűzfalon is átmegy! Erre csak a FireWalk módszer alkalmas.

- *TCP egyszerű scan:* sima TCP kapcsolatfelvétel, azaz a három utas TCP kézfogás használata. Egyszerűen implementálható, de nagyon primitív módszer.
- *TCP SYN scan:* a kapcsolat létesítés harmadik lépését kihagyja. A célpont persze az első SYN-re a szabványnak megfelelően ACK, SYN-el, vagy RST-vel válaszol, attól függően, hogy a port nyitva van-e. Sok rendszer nem naplózza az ilyen portscant, mert ez nem egy érvényes kapcsolatfelvétel.
- *TCP FIN scan:* a SYN csomagokat kívülről gyakran nem engedi be a tűzfal, de a FIN-ek átcsúsznak rajta. Ekkor használható a FIN scan, itt nem SYN, hanem FIN csomagokat küld a támadó. A cél RFC szerint RST-vel válaszol, ha a port csukva van, illetve nem válaszol, ha az nyitva volt. Egyes operációs rendszerek rosszul implementálták az RFC-t és mindig RST-vel válaszolnak, így immunisak ezzel a módszerrel szemben.
- *Fragmentation (töredezés) scan:* a kapcsolódáshoz használt csomagok sok kis darabra tördelésével megnehezítjük a védelmi berendezések munkáját. Gyakran a tördelt csomagokat nem is kísérelik meg összerakni, így ezek átmehetnek a tűzfalon, illetve nem detektálódnak az IDS által. Biztonságosan konfigurált rendszerek figyelmen kívül hagyják a sok darabban érkező csomagokat és azok azonnal eldobásra kerülnek.
- *UDP scan:* mivel az UDP nem kapcsolatorientált, ezért sokkal nehezebb megállapítani, hogy mely UDP portok vannak nyitva. RFC szerint nem kell válaszolni egy zárt UDP portra történő kapcsolatkérelemre, de a legtöbb implementáció mégis megteszi. Ez lehetővé teszi, hogy a támadó azonosítson néhány biztosan zárt portot, de a többiről nem ad információt.
- *FTP proxy scan:* szabvány szerint az FTP szerverek támogatják a proxy módot, azaz meg lehet határozni, hogy melyik IP-re menjenek az adatok a szervertől a klienshez. Ez manapság inkább biztonsági hiányosságnak nevezhető, hiszen amellett, hogy felhasználható portscan-re, tetszőleges mennyiségű adatot küldhetünk akárhova az Internetre, hamisított levelet írhatunk stb. A portpásztázás úgy lehetséges, hogy célként az áldozatot jelöli meg a támadó, PORT parancscsal pedig kiválasztva minden pásztázandó portot, adatot küld. Az FTP szerver tájékoztatást nyújt arról, hogy sikerült-e az átvitel vagy nem, ebből a támadó megtudja, hogy nyitva van-e a célport.
- *Zombie scan:* ez a technika talán a legkreatívabb az összes közül. Az IP sorszám mező (nem TCP sorszám!) lehetőséget ad egy olyan portscan végrehajtására, ahol a támadó rejtve marad. Szükséges egy olyan zombi gépet találni valahol az Interneten, ami kevés forgalmat generál, és a következő biztonsági hibával rendelkezik: az IP sorszám mező minden küldött csomagnál konstans értékkel nő. Ez a sorszám, a TCP sorszámmal ellentétben nem változik kapcsolatonként, hanem egy globális érték, minden egyes elküldött csomagnál nő. A támadó folyamatosan kommunikál a zombival, és figyeli, hogy mennyivel nő ez az érték a zombitól jövő válaszokban. Közben kapcsolódni próbál a célpont egyik portjára, de az IP forrás mező értékét kicseréli a zombi címére. Ha a port nyitva van, akkor a cél egy SYN, ACK-t küld a zombinak, mire ő egy RST-vel válaszol. Ha zárva, akkor egy RST-t küld, erre a zombi nem válaszol. Látható, hogy előbbi esetben egy csomagot kibocsátott a zombi, míg utóbb nem, ebből a támadó tudhatja, hogy a port nyitva volt-e.
- *Snow blind (hóvakság) scan:* sok hamis címről indít scant a támadó, a valós IP címe mellett. Így a célpontnál gyakorlatilag lehetetlen kitalálni, hogy melyik volt az igazi.

2.4. A tűzfalak típusai

A hálózatbiztonság alapeszközének tekintett eszközöket több szempont alapján lehet csoportosítani [4,5]. Fontosabb csoportosítási szempontok:

- *Implementáció típusa szerint:* hardver-szoftver.

Előbbi esetén a tűzfal célhardveren fut, nem egyszerű PC-n. Hardveres megoldásra példa a Cisco PIX tűzfalcsalád. Szoftveres például a Linux iptables.

• *Elemzés módja szerint:* csomagszűrő, kapcsolatalapú, alkalmazásszintű és proxy.

Csomagszűrőnek nevezzük a kapcsolatokat nyilván nem tartó, egyszerű IP, TCP adatok alapján szűrést végző eszközöket. A kapcsolatalapú tűzfalak figyelik az átmenő forgalmat és ha egy kapcsolat engedélyezett a forrás irányából, akkor nem kell külön a visszairányú forgalmat is engedélyezni. Alkalmazásszintű tűzfalak a magasabb rétegek információit is elemzik, például a HTTP kérésekben az URL hosszát és tartalmát. Proxy tűzfalak nem engednek direkt kapcsolatot a védett hálózatra, hanem a tűzfal folytatja le a védett gépek helyett a kommunikációt, (mint egy web proxy) az egyes átmenő protokollokat mélyrehatóan elemzi és ha kell, megszakítja.

Tipikus esetben a négy mód közül többet is megvalósít egy jó tűzfal, például: kapcsolatalapú + alkalmazásszintű, proxy + kapcsolatalapú. Az ilyen hibrid megoldások előre meghatározott forgalmat az egyik, illetve másik módon szűrik. (Példa: egy proxy-kapcsolatalapú hibrid az ismert, proxyzásra alkalmas protokollokat proxy a többi protokollt kapcsolatalapú tűzfal módon szűri.)

• *TTL mező kezelés alapján.*

Ez a FireWalk vizsgálatakor kulcsszerepet játszik. Fontos kérdés, hogy csökkenti-e a TTL mezőt vagy nem?

3. Tűzfalszabályok felderítése

A támadó egyik fontos célja lehet a hálózati topológia megismerése mellett a szűrési szabályok felderítése.

Ez az információ több előnnyel is kecsegtet: meghatározható, milyen forgalom megy át a tűzfalon, egycsomagos támadások ilyen forgalomnál gond nélkül végrehajthatók. Másrészt ha ismert a tűzfal szabályrendszere, akkor a szűrt portok megkerülésével célszerű támadásokat indítani. (A tűzfalszabályok megkerüléséről röviden lesz még szó a továbbiakban.)

Az előző fejezetben bemutatott portscan technikák képesek meghatározni egy hálózati berendezésen futó alkalmazásokat, illetve a zárt portokat, ha nincs tűzfal a célhoz vezető úton. Amennyiben van (ez a tipikus eset), akkor csak a nyitott portokat képes meghatározni.

Tekintsük egy port scannelését, valamilyen SYN alapú módszerrel. (A FIN scan jellegű módszerek által nyerhető információ még kevesebb, lásd fentebb.)

A lehetséges válaszok a SYN csomagra:

• *SYN, ACK* – ez egyértelműen jelzi, hogy a port nyitva van.
• *RST* – két eset lehetséges: zárva van a port és a célpont válaszol, vagy tűzfalszabály tiltja a forgalmat és a tűzfal reset-tel bont.
• *Nincs válasz* – a port szűrve van, a tűzfal drop módra van konfigurálva, illetve a célponton futó operációs rendszer vagy személyes tűzfal nem válaszol.

Az első esetben nincs további teendő, megállapítható a port állapota. Az utóbbi két esetben azonban további vizsgálat szükséges.

3.1. A FireWalk technika

A FireWalk pont a fenti kérdést képes eldönteni. A tradicionális FireWalk algoritmus a következő [1]:

0) *Bemenetek:* célpont IP-je, tűzfal IP-je. (A módszer feltételezi, hogy csak egy tűzfal van a célhoz vezető úton.)
1) *Tűzfal távolságának meghatározása:* hány ugrásra (hop) van. Ez történhet egyszerű traceroute-tal. Ez a számot jelöljük N-nel.
2) *Scannelés N+1 TTL értékkel.* Ekkor (optimális esetben, ld. lejjebb) a tűzfal a következőképpen reagál:
   – A célportot átengedi a szabályrendszer. Ekkor a router továbbítani próbálja a csomagot és lejár a TTL. ICMP time-exceeded üzenetet generál a forrásnak.
   – A célport szűrve van: ekkor azonnal eldobásra kerül a csomag és nincs válasz.
3) *2. pont ismétlése* az összes portra.

A fenti módszer elvben tökéletes, a gyakorlatban viszont ritkán használható.

Optimális eset alatt a következőt értjük:

– A tűzfal csökkenti a TTL-t és a szűrés a TTL csökkentés előtt valósul meg. Az utóbbi jellemzően így történik, de a TTL csökkentés nem minden tűzfalra igaz. Példa a Cisco PIX tűzfalcsalád: ezek nem csökkentik a TTL-t. Ilyen eszközök ellen nem lehet ezt a típusú felderítést végrehajtani, hiszen annak lényegi része (TTL csökkentés) ellen immunisak.
– A tűzfal a szűrt forgalmat egyszerűen eldobja.
– A tűzfal generál ICMP üzeneteket. Ennek köszönhetően kapunk vissza a 2. pontban ICMP time-exceeded üzenetet. Ha a tűzfal nem generál ICMP-t, akkor egyik esetben sem kapunk választ, nem lehet eldönteni, hogy szűrve van-e a port.

3.2. A FireWalk továbbfejlesztése

Amennyiben ezek nem teljesülnek, nehezebb végrehajtani a támadást. Ilyenkor más szempontokat is figyelembe kell venni:

Ha RST-vel bontja a kapcsolatokat a tűzfal, akkor az algoritmus 2. lépésének egyszerű változtatásával ebben az esetben is végrehajtható a támadás:

2) *Scannelés N+1 TTL értékkel.*
   Ekkor a tűzfal a következőképpen reagál (RST szűrés esetén):
   – A célportra irányuló kapcsolatkéréseket átengedi a szabályrendszer. Ekkor a tűzfal továbbítani próbálja a csomagot és lejár a TTL. ICMP time-exceeded üzenetet generál a forrásnak.
   – A célport szűrve van: ekkor azonnal elutasításra kerül a csomag, és RST válasz érkezik.

Ez volt a két alapeset. Innentől különböző korlátozások mellett vizsgáljuk meg a módszer kivitelezhetőségét.

1) *A tűzfal nem generál ICMP-t.*
   Ilyenkor rögtön problémát okoz a szükséges N TTL érték megállapítása. Mivel nem mindig ismert a tá-

volság, vagy lépésről lépésre kell a módszert alkalmazni, vagy feltételezhetjük hogy a vizsgálandó tűzfal az első, ahonnan már nem jön ICMP válasz, esetleg a traceroute kimenetből következtethetünk. (A szolgáltatói hálózat után tipikusan van egy ügyfél oldali router, utána pedig egy tűzfal. Ez persze nem törvényszerű, de kiindulásnak jó lehet.)

RST bontás esetén akkor is el lehet dönteni, hogy a port szűrve van-e: ilyen esetben nincs válasz. Ez megkülönböztethető a másik választól, az RST-től.

DROP mód esetén azonban nem ilyen egyszerű a helyzet, egyik esetben sincs válasz: engedélyezés esetén az ICMP time-exceeded nem keletkezik, szűrés esetén egyszerű eldobás miatt nincs válasz. Ekkor a TTL további növelésével érhető el a kívánt siker: még eggyel megnövelt TTL (N+2) esetén, ha van még egy router a célpont és a tűzfal között. Ekkor szűrés esetén nincs válasz, egyébként a következő router fog ICMP time-exceeded üzenetet generálni, ami átmegy a tűzfalon, hiszen csak az ICMP generálás van tiltva, az átmenő forgalomban engedélyezett az ICMP.

Ennek az esetnek változata, amikor nincs még egy ugrás (router) a tűzfal és a cél között. Ekkor a csomag eljut a célig, és vagy itt, vagy a tűzfalon eldobásra kerül. A két esetet nem lehet megkülönböztetni.

Szintén elképzelhető, hogy ugyan van még egy ugrás, de ezen a routeren is korlátozva van az ICMP generálás. Ilyenkor szükség lehet a következő ugrás vizsgálatára.

2) *Az átmenő ICMP tiltva van.*
   Sem RST, sem DROP esetben nincs jelentősége, mivel az alapalgoritmushoz nincs szükség átmenő ICMP-re.
3) *Mind az átmenő,*
   mind a tűzfalon generált ICMP tiltva van.
   RST bontás esetén lehetséges a támadás kivitelezése, az első pontban leírtak szerint. Itt nincs szükség átmenő ICMP-re.
   DROP esetben nem lehetséges a támadás végrehajtása, hiszen nem tudunk különbséget tenni az első pontban leírt „nincs válasz”, illetve ICMP time-exceeded között.

Összegezve: ha a tűzfal RST-vel bont és csökkenti a TTL-t, akkor az ICMP szűréstől függetlenül a támadás kivitelezhető. DROP bontás esetén is végrehajtható, ha nincs semmilyen ICMP szűrés implementálva. Részleges szűrés esetén: ha a tűzfalon generált ICMP van tiltva, és még van legalább egy router a célponthoz vezető úton a tűzfaltól számítva, akkor kivitelezhető. Ha csak az átmenő ICMP van szűrve, akkor is lehetséges a támadás. (További háttérismeret a routing, switching témakörhöz: [6,7].)

3.3. Egyéb módszerek

A tűzfalak szabályrendszerének kitalálására léteznek egyéb módszerek is, azonban a FireWalk a leghatékonyabb, legtöbb esetben alkalmazható.

Következzen ezekről egy rövid lista:

• *Tördelés*

IP csomagok kis darabokra tördelésével elérhető, hogy bizonyos tűzfalak figyelmen kívül hagyják az így álcázott csomagokat. A tördelt adatokat a tűzfalnak össze kell állítania ahhoz, hogy értelmezni és szűrni tudja, ez igen erőforrásigényes művelet. Ez ellen védekezni kell, a legtöbb hálózatba be vannak engedve a tördelt csomagok. Erre manapság csak speciális esetben van szükség, célszerű tiltani illetve korlátozni a tördelést.

• *Forrásport-hamisítás*

Csak egyszerű csomagszűrők ellen hajtható végre. Azt használja ki, hogy ezen tűzfalakon engedélyezni kell a válaszforgalmat is. (Ezzel szemben a kapcsolatalapúak nyomon követik a kimenő/bejövő kapcsolatokat, és a válaszforgalmat automatikusan engedélyezik. Proxy tűzfalak ellen a beépített protokollvizsgálat miatt szintén nem működik.) Ilyen esetben ismert portok forrásként beállításával tetszőleges port elérhető a tűzfalon keresztül. Például: ha engedélyezve van két interfész között az FTP, akkor a támadó ha FTP portról scannel, a forgalmát a tűzfal (illetve az adminisztrátor által létrehozott szabályrendszer) egy – nem létező – FTP kapcsolat válaszforgalmának minősíti, és engedélyezi. Védekezésként érdemes kapcsolatalapú tűzfalat használni.

• *TCP, IP nem használt mezők, illegális értékek*

A kreatív hackerek újra és újra találnak olyan, gyakorlatban nem használt, értelmetlen fejlécbeállításokat, amik átengedésre késztetik, vagy leállítják a tűzfalat. Ezek ellen a fölösleges szolgáltatások tiltásával, korlátozásokkal lehet védekezni.

• *Támadás a tűzfal ellen*

Ritkán fordul elő, de a tűzfal operációs rendszere ellen is jöhet ki exploit (biztonsági hiányosságot kihasználó program). Így elképzelhető, hogy a támadó hozzá tud férni a tűzfal konfigurációjához, és ebből meg tudja szerezni, esetleg módosítani tudja a szabályokat. A védekezési lehetőség azonos az előző pontban írtakkal.

4. Összefoglalás

Láttuk, hogy milyen módon lehet egy tűzfal szabályrendszerét felderíteni. A fentieket összegezve megállapíthatjuk, hogy a következő általános elvek sok veszélytől védenek [2]:

– Fölösleges szolgáltatások tiltása.
– Mindent tiltani, kivéve ami engedélyezett – és nem fordítva!
– Erős szűrési szabályok konfigurálása az átmenő forgalomra.
– A lehető legkevesebb információt nyújtani a tűzfalról, illetve a védett hálózatról.
– Szoftver (operációs rendszer) gyakori frissítése.
– Lehetőleg célhardver alkalmazása.

Ezeknek az elvi, absztrakt szabályoknak az átültetése a gyakorlatba erős védelmet nyújt a FireWalk és az összes többi módszerrel szemben. Az előbbi módszer lehetséges alkalmazási területének bővítésére hívtuk fel a figyelmet és számba vettük a különböző eseteket, védelmi megfontolásokat.

Konkrétan a következőkre van szükség ahhoz, hogy FireWalk támadást ne lehessen végrehajtani a tűzfal ellen:

– A tűzfalat DROP módra kell állítani.
– Mind a tűzfalon generált, mind az átmenő ICMP forgalmat szűrni kell.
– Ezek mellett javasolt olyan tűzfal használata, amelyik nem csökkenti a TTL-t.

Az ICMP szűrés megvalósítására egyes tűzfalak képesek intelligens módon is, a kapcsolatalapú módszerhez hasonlóan engedik át az ICMP üzenetekhez tartozó legitim válaszokat. Amennyiben az ICMP teljes szűrése nem megoldható – például szükség van rá diagnosztikai vagy egyéb célból –, akkor javasolt ilyen konfiguráció használata.

Mivel a proxy tűzfalak alkalmazása rendkívül hatásos védelmet biztosít több felderítés, illetve támadás ellen, felmerül a kérdés, hogy mi értelme van akkor egyáltalán nem proxy tűzfal használatának? Nos, ez a kérdés a hálózati biztonsági viták egyik meghatározó témája volt az elmúlt 10-15 évnek, napjainkban a használt tűzfalak több mint 90%-a nem ilyen.

Íme a legfőbb okok, amik a proxy tűzfalak háttérbe szorulását okozták:

• *Teljesítmény*

Lassabbak, mint kapcsolatalapú társaik, hiszen összetettebb elemzést végeznek, és kétszer annyi kapcsolatot kell kezelniük.

• *Alkalmazhatóság*

Ugyan a proxy technika jól működik ismert protokolloknál, a nem szokványos alkalmazásokon elveszik a biztonsági többlet, rosszabb esetben az adott protokoll nem működik proxy-n keresztül. A HTTP, FTP SMTP stb. igen, de saját alkalmazások protokolljai, például egyes VPN megoldások nem alkalmasak proxy átvitelre.

• *Drágább üzemeltetési költségek*

Bonyolultabb konfiguráció és szabályrendszer, az egyes protokollok mély ismerete szükséges. Nehezen azonosítható hibák gyakrabban fordulnak elő mint a nem proxy megoldásoknál.

Még egyszerűbb tűzfal-implementációk esetében is igaz, hogy a szabályrendszert gyakran kell módosítani, átalakítani, emiatt szükséges a rendszeres karbantartás. Ekkor célszerű a nem használt szabályokat törölni, felülvizsgálni a biztonsági politikát.

Ha kész a változtatás, akkor mindenképpen javasolt a tűzfal auditálása. Ennek a folyamatnak szükséges lépése a teljes védett hálózat portscannelése, felderítési kísérlet végzése. A feladatra több program is a segítségére lehet a rendszeradminisztrátornak, talán legismertebb az nmap ingyenes portscanner. A tűzfalak auditálása egy külön területté nőtte ki magát. Részletesen elemezni és ismertetni itt nincs lehetőségünk, az Interneten számos remek írás olvasható a témában.

A fentiek figyelembe vételével nehéz órákat okozhatunk a rendszerünket éppen feltörni készülő crackernek, illetve az auditot készítő biztonságtechnikai kollégának.

A cél ugyanaz mindkét esetben: a védett hálózat legyen minél kevésbé megismerhető a külső szemlélő számára.

Irodalom

[1] David Goldsmith, Michael Schiffman: A Traceroute-Like Analysis of IP Packet Responses to Determine Gateway Access Control Lists (Az eredeti Firewalk-ot ismertető publikáció)
[2] David M. Piscitello: Firewall Best Practices – Egress Traffic Filtering
[3] www.insecure.org – Port Scanning Techniques: http://www.insecure.org/nmap/man/man-port-scanning-techniques.html
[4] Andrew G. Mason, Mark J. Newcomb: Cisco Secure Internet Security Solutions
[5] Richard A. Deal: Cisco Router Firewall Security
[6] Richard Froom, Balaji Sivasubramanian, Erum Frahim: Building Cisco Multilayer Switched Networks (BCMSN), Second Edition
[7] Catherine Paquet, Diane Teare: Building Scalable Cisco Internetworks (BSCI), Second Edition

Vírusok, férgek, rootkitek, a legújabb fenyegetések

ADAMKÓ PÉTER

Budapesti Műszaki és Gazdaságtudományi Egyetem, Informatikai Központ
ap331@hszk.bme.hu

Kulcsszavak: vírus, féreg, rootkit, algoritmus, detektálás, kernel

Körülbelül másfél éve kezdtek elterjedni a Windows-on működőképes rootkitek, megkeserítve ezzel a felhasználók és biztonsági szakemberek életét egyaránt. Fény derült arra, milyen nehéz az új típusú kártevők ellen védekezni. Ami nem volt világos, hogy lehetséges-e ezen új technológiát ötvözni a régi kártevőkével. Cikkünkben a férgek és rootkitek legveszélyesebb tulajdonságait mutatjuk be, valamint ismertetjük a rendelkezésre álló detektálási technikákat, szoftvereket.

1. Helyzetünk

A számítógépes vírusok több, mint 20 éve keserítik meg napjainkat. Eközben folyamatosan átalakultak, változtak s érték el pillanatnyi fejlettségüket. Eleinte különböző adathordozó médiákon terjedtek, azonban később a különböző hálózatok összekapcsolásával sokkal gyorsabb terjedésre nyílt lehetőségük, végül pedig az Internet elterjedésével betörtek az emberek mindennapjaiba.

Ekkorra végeredményben el is tűntek a klasszikus értelemben vett vírusok, helyüket az úgynevezett programférgek vették át (legnagyobb különbség, hogy a férgek önálló programok, immár nincs szükségük más hordozóra a működéshez, terjedéshez). Az elmúlt 4 évben sorozatban lehettünk tanúi az egész világot behálózó féregjárványoknak, melyek mintha csak az utóbbi időben csillapodtak volna. Általánosan igaz, hogy minden egyes esetben valamilyen széles körben használt szoftver hibáját használták ki, gyakran ötvözve a felhasználók megtévesztésével szolgáló technikákkal.

Kimondottan érdekes téma, hogy mitől tudtak enynyire elterjedtté válni ezek a kártevők? Emberi hanyagság, lustaság, esetleg jó algoritmusok és kiemelkedő programozás eredménye? Mára már ott tartunk, hogy egy-egy féreg detektálása és a védelmi szoftverek frissítése közötti idő rohamosan csökken, akár pár óra is lehet. Egészében véve elmondhatjuk, hogy a különböző IT-biztonsági megoldásokat szállító cégek kidolgozták a tömeges és nagyméretű féregtámadások elleni védelmet. Azonban ezzel kapcsolatban még mindig felmerülnek kérdések:

Mi történik, ha a frissítés előtt a kártékony program már megfertőzte a sebezhető gépek többségét (a Slammernek kevesebb, mint 15 percre volt szüksége)?

Mi történik azokkal a programokkal, amelyek nem mutatnak tipikus vírusviselkedést (gyors terjedés, komoly hálózati forgalom, fájlműveletek stb.), vagy még nincsenek benne a leíró adatbázisban?

Mi történik azokkal a programokkal, melyeket rejtőzködő életmód folytatására terveztek?

Biztonsági szakértők szerint az elkövetkező támadások már nem világméretűek lesznek, hanem egy-egy szervezet, cég, intézmény IT infrastruktúrája ellen fognak irányulni. Kétségtelen, hogy a rejtőzködés és a gyors terjedés némileg ellentmondanak egymásnak, de az is igaz, hogy egy gyors terjedésű, rootkit technológiát használó programféreg akár eltávolíthatatlanná is tudja magát tenni, miközben sokkal nehezebben detektálhatóvá válik. Sok víruskereső program már most is küszködik a rootkitek detektálásával.

A dolog aktualitását mutatja, hogy az évek óta jelenlevő Bagle immár rootkit technológiát használ. Nem véletlen, hogy a 2005-ös évben a Kaspersky Labs adatai alapján a rootkitek száma 413%-os növekedést ért el.

2. Algoritmusok

Mi is tette annak idején annyira pusztítóvá ezeket a férgeket? Egy gyors féreg sikeressége azon múlik, milyen gyorsan találja meg új célpontjait, milyen gyorsan képes őket megfertőzni.

Az új célpontokat jellemzően valamilyen szkennelés révén találja meg. A szekvenciális szkennelés (Blaster) során egy véletlen címtől kezdve szekvenciálisan nézi végig az IP címeket az algoritmus. Ez rosszabb hatékonyságú, mint az egyszerű véletlenszerű szkennelés, hiszen sok redundancia van benne, azonban ha gyorsan talál egy sérülékeny gépekkel teli hálózatot, akkor azt végigpásztázza és végigfertőzi. Ha mindez a terjedés elején történik, akkor az meggyorsítja az algoritmust, esetenként még a véletlen szkennelés gyorsaságát is megközelítheti, de ez nagyban függ a szerencsétől.

Maga a fertőzési sebesség szintén fontos tényező. Például annak megállapítása, hogy nem sebezhető a rendszer, gyorsabb lehet, mint maga a fertőzés. A fertőzés gyorsasága nagyban függ a megvalósítás módszerétől. A TCP kapcsolat felépítése, például TFTP esetén (a féreg törzse TFTP-vel töltődik fel a fertőzés

után) sokkal lassabb, mint mondjuk egy UDP csomagon keresztül terjedő féreg esetén. A Slammert az tette olyan gyorssá, hogy nem kellett állapotokat tárolnia, visszajelzésekre várnia, csak elküldték egy rendszer felé, és véget is ért a támadás. Ellenben az általános működésű férgek először fertőznek, majd a féreg testét, és a többi funkciót tartalmazó részt is feltöltik a megtámadott hosztra.

További fontos dolog, hogy mennyi sérülékeny gép van elérhető állapotban, egy nagyobb populációban ugyanis nagyobb a találati esély is. Legvégül pedig a szülő és gyerek férgek feloszthatják egymás között a szkennelési tartományt, így párhuzamosítva a műveletet. Így a célpontkeresés bináris fában történő keresésé válik, ami gyorsabb, mint a lineáris keresés.

Javít a gyorsaságon az is, ha a féreg fel tudja ismerni a már megfertőzött rendszereket, s ezeket a szkennelés során kihagyja illetve, ha egyes címtartományokról tudja, hogy azok nem tartalmaznak sérülékeny gépeket.

Ezek alapján a következőt mondhatjuk el egy fejlett féreg viselkedéséről: gyors, állapotmentes célfelismerő algoritmust használ, elkerüli az üres, vagy már megfertőzött címtartományokat, párhuzamos szálakat használ, gyorsan fertőz és kisméretű [4].

Ilyen algoritmusokat már 2001-ben kidolgoztak, s igazán szerencsések voltunk, hogy csak 2003-ban jelent meg olyan programféreg, mely használta ezeket. A Warhol és a Flash algoritmusok mindegyike lehetőséget ad arra, hogy az Interneten fellelhető sérülékeny gépeket kevesebb, mint 15 perc alatt végigfertőzzék.

Azonban, ha jobban belegondolunk, a jelenlegi sávszélességi állapotok mellett csak az elsőnek van létjogosultsága. A legtöbb féreg problematikája a továbbterjedés irányítása volt, melyre különböző módszereket használtak: címlisták szerzése a fertőzött gépről, hálózati szkennelés, random címek generálása, előre beállított szerverekről történő címfrissítés, talált email címekre való továbbküldés. Viszont egy kimerítő keresés meglehetősen nagy hálózati forgalmat generál...

Minden féreg a kezdeti terjedési szakaszában állítható meg a legkönnyebben, amikor még nem érte el azt a fertőzési számot, mely után a továbbterjedés robbanásszerűen megugrik. Természetesen ehhez az kell, hogy korai szakaszban fogjunk be egy példányt, s alkossunk rá felismerési mintát (ez általában valamilyen hash érték létrehozása). Ez órák kérdése lehet, de alapvetően a fent említett algoritmusok esetén nincsen elég idő erre. Éppen ebben rejlik nagyfokú veszélyességük.

Az algoritmusok egy előszkenneléssel kapott címlista generálásával (DNS lekérdezések, Google, web-robotok stb.) meghatározzák a kezdeti célpontokat, s egy időben támadva őket, gyorsan túllépnek a kezdeti szakaszon, létrehozva egy 10000-es kezdeti populációt. A következő lépés a további gépek megfertőzése, s ez az a pont, ahol a két algoritmus elkülönül egymástól. Az egyik megpróbálja „gazdaságosan” végigszkennelni a hálózatot, az egész IP címteret felosztva a féregpéldányok között. Minden fertőzés után a szülő és a gyerek között felezi a keresési teret. Ha egy már megfertőzött gépet talál, úgy egy új random, vagy szekvenciában rögzített pontról kezdi újra a szkennelést, hiszen tudja, a már fertőzött példány felelős a tér következő részének átkutatásáért, és már előrébb is tart ebben.

Ez egyes kutatók szerint gyorsabb is lehet, mint a hagyományos random szkennelés, de ha nem is gyorsabb, mindenképpen kevesebb üzenetet generál. A Flash algoritmus teljesen kihagy mindenféle szkennelést, mivel már az előszkennelés során feltérképezte az összes sebezhető hosztot (az egész Interneten), s terjedése során ezt a listát menedzseli, s osztja szét a különböző féregpéldányoknak. Bár ez még gyorsabb lehet, mint a Warhol technika, azonban ez a lista már kellően nagyméretű ahhoz, hogy sávszélességi problémákat vessen fel. [4], [5]

Milyen védelmi lehetőségeink vannak? A fent említett terjedési algoritmusok leginkább jellemző tulajdonsága a hálózati forgalomban létrehozott anomáliák. Ezek detektálhatóak különböző hálózatfigyelő szoftverekkel, NIDS-ekkel. További lehetőség a vállalati hálózat határán különböző szűrések végzése. Általában itt helyezkedik el a különböző vírusvédelmi szoftverek egy része, melyek ismert minták után szűrik az átmenő forgalmat. A publikus szervereken is szinte kötelező érvényű a szűrés, tipikus példa erre a levelezőszervereken felállított védelem.

Különösen fontos ez, hiszen mára már a férgek két legszignifikánsabb fajtája a levelező (mass-mailer), és a hálózati (network) férgek típusa. Az ismert vírusok keresésén kívül a gyakran ismétlődő minták, csatolt fájlok is felkelthetik figyelmünket (van olyan szoftver, mely ezekből úgynevezett szürkelistát hoz létre, s amíg nem tisztázódik vírusmentessége, nem továbbítja), sőt általában a csatolt fájlok kiterjesztései is, vagy gyakran ismétlődő nevei.

Véleményem szerint ez mindössze addig megoldás, amíg egy jobb polimorf motorral rendelkező programféreg meg nem jelenik. További segítség lehet, ha csak olyan levelező szerverekről fogadunk el leveleket, melyek címe feloldható. Legvégül pedig fontosak a munkaállomáson telepített vírusvédelmi szoftvereket is.

3. Rootkitek

A rootkitek már régóta jelen vannak életünkben, a Windows-t használók számára azonban csak az elmúlt egy-két évben váltak ismerté. Szeretnénk leszögezni, hogy teljes mértékben egyetértünk Greg Hoglund-dal, aki szerint ez mindössze egy olyan eszköz, mely képes arra, hogy elrejtsen folyamatokat, adatokat egy rendszerben.

Sok felügyeleti eszköz, védelmi program és biztonságos dokumentumkezelést biztosító szoftver is alkalmazza ezeket a technikákat (tűzfalak, CSA, ISeeSec).

Most a vírusokhoz kapcsolódóan mégis a támadási lehetőségeik kapcsán tárgyalom a rootkitek működését.

Az x86-os architektúra négy privilégiumszintet határoz meg (Ring0-3), ezek közül az operációs rendszerek általában kettőt használnak, a felhasználói és a kernel szintet. A privilégiumok meghatározzák, hogy a programok milyen műveleteket hajthatnak végre, hiszen a kernelt meg kell védeni a felhasználói programoktól, viszont bizonyos szolgáltatásokat biztosítani kell számukra. Míg felhasználói módban egyes memóriaterületek védettek, addig kernel módban az egész memóriához, s a processzor összes utasításához hozzáférhetünk. Ennek eredménye, hogy kernel módban jól lehet nyomon követni, elfogni, s módosítani a rendszerben lévő üzeneteket, állapotokat.

A felhasználói módú támadó rootkitekre jellemző, hogy többnyire önálló alkalmazásként futnak, vagy pedig egy már létezőt cserélnek le.

Ezen rootkitek és a védelmi szoftverek között állandó harc dúl, hogy melyik tudja átvenni az operációs rendszer fölötti irányítást, s ennek kimenetét az határozza meg, hogy melyik tud mélyebb szinten a rendszerbe nyúlni (hiszen az operációs rendszer különböző szintjei egymáson keresztül érik el az adott szolgáltatásokat). Például az operációs rendszer teljes körű irányítással rendelkezik egy-egy alkalmazás memórianézetéről, hiszen a fizikai és virtuális memória közötti leképzés absztrakcióját ő maga végzi el.

Általánosságban két típusú rootkitről beszélhetünk, perzisztens és memória alapúakról. Legnagyobb különbség, hogy az előbbi túléli a hoszt újraindulását. Ehhez két dolog szükséges: valahol tárolniuk kell a kódjukat a rendszerben (többnyire a merevlemezek egyikén), valamint valahol be kell tölteniük magukat a rendszerbe (például beépülve az indítási szekvenciába). Ezeket a változtatásokat valahogy rejteniük is kell. A memória alapú rootkitek csak a memóriában léteznek, s újraindításkor törlődnek. Bár ez gyengeségnek hathat, azonban a szervereket viszonylag ritkán indítják újra, ennek fejében lényegesen nehezebb detektálni őket, s jóval kevesebb nyomot is hagynak. Ettől függetlenül el kell rejteniük a memóriában található végrehajtható kódjukat, valamint el kell rejteniük a memóriabeli módosításokat is (ezek nélkül akár a legegyszerűbb mintaillesztéses memóriaszkennelés is elárulhatja őket) [2].

A rootkit elrejtésére egy polimorf technika is jó lenne, azonban ez még mindig jól láthatóan otthagyná a memóriában végzett változtatásokat, így a rendszerkomponensek változásai egy integritásellenőrzés során kimutathatóak lennének. Ennek következtében a rendszer rootkitről látott képét kell módosítani.

3.1. Rootkit technológiák

A legelterjedtebb technikák közé tartozik az úgynevezett kampók (hook) használata, valamint a kernel objektumainak (DKOM) manipulálása. Az első során az operációs rendszer normális végrehajtási útvonalát módosítja a rootkit, és így módosítani tudja azokat az információkat, melyeket egy rendszerhívás visszatérésekor ad.

3.1.1. Felhasználói módban használható módosítási lehetőségek

Import Address Table

A Windowst úgy tervezték, hogy hardvertől független legyen, valamint kompatibilis legyen más környezetekkel (pl. POSIX). Fontos volt, hogy a fejlesztőknek ne kelljen mindig újraírniuk kódjaikat egy-egy rendszerfrissítésnél, ezért különböző alrendszereket használ, melyeket dll-ként implementál. Ezeken az interfészeken keresztül érhetőek el a kernel szolgáltatásai.

A különböző alkalmazások nem közvetlenül hívják a Windows rendszerszolgáltatásokat, hanem az alrendszereken (OS/2, POSIX, Win32) keresztül. Ezek a könyvtárak exportálják azon interfészeket, amelyekkel a programok kapcsolódnak hozzájuk. A leggyakrabban használt a Win32, mely a Kernel32, a User32, a Gdi32.dll és az Advapi dll-ekből áll. Az Ntdll.dll olyan speciális rendszertámogatási könyvtár, amelyet az alrendszer dll-jei használnak. Mikor egy bináris fájl betöltésre kerül, a töltő automatikusan átnézi a fájl egy részét, az Import Address Table-t (IAT), amelyben megtalálható a dll-k listája, valamint azok a függvények, amelyeket belőlük használni fog a program. Ezután a töltő a függvények címeit a memóriába teszi. Általában a Kernel32 és az Ntdll függvényei találhatóak meg itt, de más speciális függvények is jelen lehetnek (például a Ws2_32.dll-ben található socketkezelő függvények). A kernel eszközmeghajtói szintén importálnak függvényeket a dll-ekből (pl. Ntoskrnl.exe, Hal.dll). A bináris fájlok IAT-jaiban található címek módosításával egy program magára irányíthatja a végrehajtást, s befolyásolhatja az eredeti függvény futását: egy program egy könyvtár listázását végzi, és néhány műveletet hajt végre rajtuk.

Tételezzük fel, hogy felhasználói módban fut, és Win32-es alkalmazás. Ekkor a Kernel32, a User32 és a Gui32 dll-eket fogja használni. A könyvtár listázásához a FindFirstFile és amennyiben az sikeres, a FindNextFile API függvényeket fogja meghívni. Amikor az alkalmazás meghívja a függvényt, akkor az import táblából kikeresi a címet, s a megfelelő függvényre ugrik a Kernel32.dll-ben. A rootkit az IAT címet átírva a saját függvényére irányíthatja a hívást, és tulajdonképpen bármilyen utasítást végrehajthat: meghívhatja az eredeti függvényt, s annak visszatérési adatait szűrheti stb. Fontos tudnunk, hogy amikor átírja ezt a címet, a rootkit átlépi a folyamat virtuális címtartományát, s ezzel detektálhatóvá válik.

Inline kampó

Az inline kampó létrehozása során az ellenőrizni kívánt függvény elején kell módosítani néhány bájtnyi kódot, általában egy feltétel nélküli ugrást, mellyel a rootkit kódjára ugrik a processz, majd onnan vissza. A leg-

több függvény ugyanazzal a kódrészlettel kezdődik, majd ezt követik a függvény tényleges utasításai.

A rootkitnek a függvény első 5 bájtját (1 bájt az ugrási utasítás, 4 a cím) kell átírni egy ugrási utasításra, s az ugrási címre. Ezt a rootkitnek érdemes lehet elmentenie, mivel a Service Pack2 előtti időkben az azonos kódrészlet mindössze 3 bájt volt, s mivel az ugrás + cím 5 bájtnyi helyet foglal, tudnia kellett, hogy milyen utasítást írt át (hogy megőrizze a függvény eredeti funkcionalitását).

A Service Pack2 utáni időben 5 bájtra növekedett ennek a kódnak a hossza, legális haszna, hogy lehetővé teszi a „hot patching"-et, vagyis, hogy újraindítás nélkül lehessen módosítani a függvényeket. Viszont így probléma nélkül lehet módosítani rossz indítékkal is őket.

Dll injektálás

Dll injektálás révén betölthető a rootkit kódja egy másik folyamat memóriaterületére. Az NT/2000/XP/2003 Windows családban a HKEY_LOCAL_MACHINE\Software\Microsoft\WindowsNT\CurrentVersion\Windows\AppInit_DLL kulcsból olvassa ki a rendszer, hogy az egyes alkalmazások milyen dll-eket töltenek be indításukkor. Ezt módosítva tetszőleges dll tölthető az alkalmazással együtt a memóriaterületre.

Lehetőség van eseményvezérelten hozzáférni más folyamatok üzeneteihez. A SetWindowsHookEx API függvényt használva más folyamatok eseményeihez férhetünk hozzá, s tetszőleges feldolgozási függvényt hívhatunk meg adott esemény bekövetkeztekor.

Legvégül pedig lehetőség van távoli szálat létrehozni a CreateRemoteThread API függvény segítségével.

3.1.2. Kernelszintű módosítási lehetőségek

Interrupt Descriptor Table

Az interrupt vektor tábla (Interrupt Descriptor Table) kampók használata még a DOS-os időkbe nyúlik vissza. Ez a tábla határozza meg, hogy az egyes események, megszakítások (például billentyűlenyomás) után milyen feldolgozó egységnek adja át a vezérlést (billentyűlenyomás, memóriahiba stb.) Fontos azonban tudni, hogy nem tér vissza, s a vezérlést sem kapja vissza, így szűrésre nem alkalmas, azonban blokkolhatja egyes szoftvereknek (pl. IDS) címzett megszakítási kéréseket.

IRP kampó

Az összes eszközmeghajtó programban jelenlévő függvénytábla szintén alkalmas kampók létrehozására, mivel ebben találhatóak meg azok a mutatók, melyek az egyes feldolgozó függvények helyeit mutatják meg. A függvények különféle I/O kérés csomagokat (I/O Request Packets) dolgoznak fel, mint például olvasás, írás, lekérdezés. Egy TCP/IP lekérdezést változtatva például elrejthetjük a nyitott portok listáját. Az IDT-nél tapasztaltaknak megfelelően ez sem hívja meg az eredeti függvényt, s nem ad módot a szűrésre.

System Service Descriptor Table

A rootkitek ahelyett, hogy alrendszereken keresztül kapcsolódnak a kernelhez, átírhatják közvetlenül annak a táblának (System Service Descriptor Table) a függvénycímeit, mely az aktuális függvényimplementációk címeit tartalmazza az adott operációs rendszerben. Ezek a címek az Ntoskrnl.exe-ben található Nt*** függvények címeit tartalmazzák.

Ezzel lényegében nem egy programra, hanem az egész rendszerre kiterjedő kampó valósítható meg. Az előző IAT példát véve, hasonló funkcionalitás elérésére a rootkit átírhatná az NTQueryDirectoryFile címét, a hívást önmagára irányítva.

Runtime patching

A kampók használata meglehetősen régi technika, s mivel a hívási táblákat módosítják, viszonylag jól észrevehetőek. A programok futási időben a memóriában történő módosítása (runtime patching) lehet a következő módszer, amelyet alkalmazni fog egy magát elrejteni kívánó program. A memóriában található adatstruktúrák vezérlik az egyes programok futási logikáját, ennek módosításával a program működése is megváltoztatható. Fontos dolog azonban, hogy ekkor a memórialapok írási/olvasási jogosultságával is rendelkeznie kell a rootkitnek.

A „detour patching" során a rootkit az eltéríteni kívánt függvény belsejében helyez el egy magára vonatkozó ugrási utasítást, ezáltal az összes függvényre mutató táblára hatott, másrészt nem kell számon tartania az összes ilyen táblát.

Layer driver

A réteges meghajtók (layer driver) újabb helyként szolgálhatnak a rootkitek elrejtésére. Az eszközmeghajtók lehetőséget adnak egymás sorba láncolására, s az egymástól bejövő adatok alapján dolgoznak. Ez módot ad arra, hogy egy-egy meghajtó program frissítéséhez ne kelljen mindig újraírni az egészet. Minden meghajtó program az operációs rendszer fontos funkcióit valósítja meg: hálózati kommunikáció, fájlműveletek stb. Sok víruskereső is egy eszközmeghajtót telepít fel a fájlok megnyitásakor történő szkenneléséhez: az operációs rendszer által biztosított fájl meghajtók által szolgáltatott adatok (például egy fájl megnyitásakor) a víruskereső meghajtójához kerülnek, amely képes mintakeresést végezni rajtuk.

Értelemszerűen ezt a rootkitek is használhatják adatszűrésre, és ha a megfelelő meghajtók előtt helyezkednek el a sorban, akkor a róluk szóló adatokat módosítva adhatják tovább.

DKOM

A közvetlen kernel objektumok manipulálásának módszere (DKOM) arra épül, hogy a rootkit kihasználja, hogy az operációs rendszer ezeket az objektumokat használja nyomonkövetésre és auditálásra. Ha ezeket módosítja, az egész rendszer önmagáról alkotott képét változtatja meg.

Általános esetben a kernel objektumok (folyamatok, tokenek) az Object Manageren keresztül érhetőek el, mely az általános funkciókat, mint például létrehozás, törlés, védelmi szintek beállítása biztosítja számukra. A rootkitek pont ezt kerülik meg, átlépve az összes ellenőrzést az adott objektumon. Bár nehezen detektálható módszer, hátránya, hogy csak a memóriában tárolt objektumokon képes módosításokat végrehajtani, így például a folyamatok, vagy portok listáját tudja változtatni. Ellenben a fájlrendszert egyetlen objektum sem képviseli a kernelben, ezért fájlokat elrejteni nem tud. Ezen kívül képes eszközmeghajtókat elrejteni, valamint folyamatok privilégiumszintjét változtatni.

Például a hírhedt FU rootkit a folyamatok nyilvántartására használt objektumot módosítja: minden folyamat egy EPROCESS adatstruktúrában van tárolva, s ezek kétszeresen láncolt listában helyezkednek el. Ebben a listában PID, vagy név alapján keresve megkeresi a kívánt folyamatot, s kiláncolja a listából [1,2].

4. Milyen védekezési eszközeink vannak a rootkitek ellen?

Lényegében két módszer közül választhatunk, megpróbáljuk jelenlét, vagy viselkedés alapján kiszűrni őket. A jelenlétet többféle módon is kimutathatjuk:

Mintakeresés

A minta alapú keresés már régen a víruskeresők sajátosságai közé tartozik. A rendszerben található fájlok végigszkennelése önmagában korlátozott hatékonyságú (a rootkitek rejtőző tulajdonságai miatt), a rendszermemória szkennelése már sikeresebb lehet. A legtöbb nyilvános kernel rootkit tipikusan a nem lapozható memóriában foglal helyet, s ritkán tesz arra erőfeszítést, hogy kódját valamilyen formában elfedje. Így nyilvánvalóan a kernel memória átvizsgálása révén leleplezhetőek. A kulcsszó itt a nyilvánoson van, hiszen egy nem ismert kódra nincsen minta az adatbázisban. Másrészt a virtuális memória menedzselésére képes rootkitek (pl. Shadow Walker) ellen sem használható, hiszen az képes a memóriaolvasások vezérlésére.

További memória alapú módszer a különböző kampók keresése. Az általános módszer az lehet, hogy végignézzük a különböző rendszertáblákat (IVT, SSDT, IDT, IAT stb.), ahol is a különböző függvények egy adott címtartományba kell, hogy essenek. Például az SSDT összes függvénycímének a kernelben található ntoskrnl.exe-re kellene mutatnia.

Betöltési helyek ellenőrzése

A memóriába töltés lehetőségeit számba véve sok olyan módszert találhatunk, melyekkel a rootkit a memóriába töltheti magát. Ezen helyek megfigyelésével és a megfelelő függvényekre kampók létesítésével (ZwSetSystemInformation, ZxOpenProcess stb.) történhet. Viszont ezekből a pontokból meglehetősen sok van.

Ma már különböző víruskereső alkalmazások is használják ezt a módszert (Nod32), így mutatva ki a rootkitek jelenlétet. Természetesen ebben benne van a hamis pozitív találatok lehetősége, hiszen számos nem kártékony alkalmazás is használ kampókat.

Viselkedés vizsgálat

Ennek során annak kimutatása szükséges, hogy az operációs rendszer valamilyen hamis információt adott meg számunkra. Itt elsősorban az elrejtett fájlok és folyamatok kimutatására teszünk kísérletet. [2]

Keresztnézeti (Cross-view) detektálás

Egy viszonylag új és sok sikerrel kecsegtető viselkedésdetektálásra alkalmas technika. A módszer feltételezi, hogy az operációs rendszer már módosított és megbízhatatlan.

Ekkor ugyanazt a lekérdezést végzi el, csak többféle módon. Először a szoftver az API függvényeken keresztül (Zw*** függvények stb.) kérdezi le a kívánt adatot (fájl, processz lista, regisztrációs adatbáziskulcsok), majd ugyanezt az adatot kinyeri valamilyen algoritmus révén, mely nem függ az API hívásoktól. Bármilyen különbség a rootkit jelenlétéről árulkodik. Az API kampók, vagy a DKOM használatából adódó hamis adatokat az alacsony szintű lekérdezésekből származó adatok leplezik le.

Meg kell jegyeznünk azt is, hogy ez a fajta detektálási módszer sebezhető a létező rootkitek támadási módszereivel szemben, hiszen nagymértékben függ az implementációtól, főleg az alacsony szintű rendszerinformációk kinyerése esetén. Például az NTFS diszk elrendezési információinak kinyeréséhez először szüksége van egy handler-re az adott kötethez, majd el kell olvasnia a szektorokat. Ha az API függvényt használja a feladathoz, akkor arról feltételeznie kell, hogy nem megbízható. Tehát az implementáció sikeressége a diszkvezérlővel folytatott közvetlen kommunikáción alapszik [3].

Most pedig lássuk ezen elméletek néhány gyakorlati megvalósítását:

VICE

Ez egy olyan eszköz, mely egy eszközmeghajtó programot telepít fel, a felhasználói és kernel módú programok figyelésére. Ellenőrzi az SSDT-t, s megvizsgálja azokat a címeket, amelyek nem az ntoskrnl.exe-be mutatnak. Emellett hozzáadhatunk fájlokat a driver.ini-hez és ekkor ellenőrzi a meghajtóhoz tartozó IRP főbb funkciós tábláját. Ha valamelyik mutató nem a meghajtóra mutató címmel rendelkezik, akkor az IRP-re egy másik meghajtó, vagy valamilyen kernel kód kampót hozott létre.

Felhasználó módban ellenőrzi az alkalmazások dll-jeit IAT-beli kampók után kutatva. Az inline függvénymódosításokat a dll-ben és az SSDT-ben is detektálja. Ekkor megpróbálja visszakeresni, hogy melyik függvény hozta létre a kampót és hol található.

A rootkitek ki tudják kerülni a programot, mivel mindig ugyanazzal a folyamatnévvel fut, s ezt a nevet érzékelve a rootkit nem hozza létre a kampókat. Egy másik támadás a VICE eszközmeghajtója és a felhasználói módban futó modul közötti kommunikációt zavarja meg. Azonban a program legnagyobb hibája a nagyszámú hamis pozitív találat, amit visszaad (hiszen létezhetnek a rendszerben legitim kampók is: hot patching, dll forwarding) [6].

Patchfinder

Ez a szoftver a statikus analízissel szemben futási idejű végrehajtási útvonalak vizsgálatával keresi a rootkiteket. A program működési elve azon alapszik, hogy egy rootkit, ha módosítja a végrehajtási útvonalat (például: kampó beillesztésével szűri egy adott szolgáltatás visszatérési eredményeit), akkor hozzáad kódot. Meg lehet számolni, hogy az egyes rendszerszolgáltatások hány műveletből állnak. Egy kompromittált rendszerben mindenképpen több műveletből kell állnia, mint egy tiszta rendszerben. A program az x86-os processzor „single step” funkcióját használja a műveletek számolására.

Sajnos a Windows operációs rendszerben több végrehajtási útvonal is lehet, amely nem determinisztikus viselkedést eredményez (vagyis egymás után megismételt számolás más-más eredményt adhat különböző rendszer feltételeknek köszönhetően). A program egy hisztogramot hoz létre, s az első csúcsértékkel számol, mely rootkit jelenlétében érezhetően elmozdul. Mindenesetre ez a módszer is képes hamis pozitív találatokat adni, valamint a programot támadó rootkitekkel szemben is sebezhető.

Rootkit Revealer

Perzisztens rootkitek ellen dolgozták ki, ezért a rendszerleíró adatbázis és a fájlrendszer lekérdezésével dolgozik. Az API függvényektől kapott adatokat a nyers fájlrendszer struktúrával és a regisztrációs adatbázist alkotó puszta fájlokkal veti össze.

Mint a legtöbb detektálási eszköz, a Rootkit Revealer is sebezhető azon rootkitekkel szemben, melyek blokkolják vagy eltérítik a hozzáférést a diszkrendszerhez vagy az adatbázis fájlokhoz. Szintén produkál hamis pozitív találatokat, amennyiben fájlok vagy adatbázis kulcsok jönnek létre, módosulnak két lekérdezés között.

Klister

A DKOM-ot használó rootkitek ellen készült, melyek az FU-hoz hasonlóan rejtik el futó folyamataikat, vagyis a PsActiceProcessList-ből kiláncolják az elrejteni kívántakat. Ekkor elvileg a processz futásának is le kellene állnia, vagyis az operációs rendszer időzítőjétől nem szabadna több CPU időt kapnia, gyakorlatilag azonban az operációs rendszer több különálló időzítőt is fenntart, melyek segítségével kiosztja a CPU időket.

Ezt a redundanciát kihasználva a Klisters összehasonlítja a folyamatok listáját a különböző időzítőkben található listákkal. A különbség nyilvánvalóan rootkit jelenlétre utal. Bár ez a módszer egyelőre csak rejtett folyamatok felderítésére szolgál, a jövőben kiteljesíthető a redundáns kernel adatstruktúrák teljes vizsgálatára.

Icesword

Egyike a legjobb ingyenesen elérhető rootkit detektoroknak, SSDT kampókat, rejtett fájlokat, folyamatokat, regisztrációs adatbázis kulcsokat és könyvtárakat érzékel, valamint nyitott portokat és rejtett socket kommunikációt. Nagy előnye, hogy sok információt ad a felhasználóknak, megmutatva, melyik rendszer-hívásokon vannak kampók, milyen meghajtó programok rejtettek stb.

Gyengesége, hogy az alacsonyszintű processz információkat egy kernel adatstruktúrára (PspCidTable) alapozva nyeri ki, s ez sebezhetővé teszi egy olyan rootkittel szemben, mely erre az egy táblára hoz létre kampót.

Blacklight

Elrejtett fájlok, folyamatok felderítésére szakosodott. Hasonló képességei vannak, mint az előző programnak, sőt, az alacsonyszintű lekérdezési algoritmusa is hasonló.

Strider GhostBuster

A Rootkit Revealer-hez hasonlóan működik, magas szintű API lekérdezéseket alacsonyszintű adatstruktúrákkal (NTFS master file table, Windows registry hive) hasonlít össze. Különlegessége, hogy keresés során először egy futó rendszeren, majd a rendszer újraindítása során egy általa produkált tiszta rendszeren (WinPE által produkált live cd) fut le. Ekkor viszonylag könnyen kimutathatja a fájlrendszerben található különbségeket, ami a perzisztens rootkitek ellen működhet. Viszont egy szerverrendszer újraindítása nem mindig kézenfekvő.

Ez azonban arra is képessé teszi, hogy adatokat találjon utólagos analízishez egy kompromittált rendszerben.

System Virginity Verifier

Ez a program egyesíti magában a VICE rendszerintegrációs ellenőrzését, valamint heurisztikus keresést is végez a találatok között, hogy a hamis pozitív találatok arányát csökkentse. A memóriaintegritás ellenőrzése a fontos rendszerkönyvtárak és meghajtó programok a merevlemezen található kódjainak és a nekik megfelelő memóriaképek összehasonlítása révén történik.

Copilot

A Copilot egyedülálló a maga nemében, hiszen hardveres megoldást nyújt a rootkit detektálásra. Jelenleg egy PCI csatlakozású kártya, mely figyeli a rootkit aktivitást a rendszerben. A PCI kártya használatának célja, hogy egy olyan tiszta elemet építsen a rendszerbe, mely bármilyen szoftveres változtatás után is független

és lehetőleg változatlan marad. Ennek érdekében saját CPU-val rendelkezik és a fizikai memóriát DMA-n keresztül éri el.

A memóriában különböző rootkitre utaló nyomokat keres: SSDT kampókat, a kernel függvényeinek és adatstruktúráinak módosításait stb. A Copilot saját hálózati interfésszel is rendelkezik, hogy biztonságos csatornán kommunikálhasson a vezérlő komponensével. A hardver miatt nagyfokú biztonságot nyújt, azonban sokkal költségesebb, mint egy szoftveres megoldás, és persze frissítése is nehézkesebb [6].

RAIDE

Egy új kezdeményezés a Rootkit Revealer készítőitől, mely a detektáláson kívül az esetleges „gyógyításra" is vállalkozik. Kombinálja a legtöbb létező eszköz detektálási módszereit, igyekszik viszonylag sok fontos és részletes információt nyújtani a felhasználók számára. Képes a következők elvégzésére:

Utólagos bizonyítékok gyűjtése (processz dump készítése), különböző, már létező motorok technikáit használja: rejtett folyamatok (Blacklight), kampók detektálása (SVV, VICE) stb. A program a felhasználói és kernel komponensei közötti kommunikációra megosztott memóriát használ, mely sokkal védettebb, mint a többi szoftver kommunikációja, révén ez csak titkosított adatot tartalmaz, sokkal nehezebb visszafejteni, megzavarni. Véletlen folyamatneveket és eseményneveket használ tovább nehezítve az esetleges támadó rootkitek dolgát [7].

Tripwire

A Tripwire egyedi CRC hash-t használ minden egyes fájl megjelölésére. Egy ellenőrzésnél minden fájlra újragenerálja az értéket, majd összehasonlítja az adatbázisában szereplővel.

Azon a megfigyelésen alapszik a működése, hogy a rendszer fájljai nem változhatnak meg (kivéve frissítés), tehát ez betörésre utaló nyom. A Tripwire nagyon eredményes volt azokkal a kezdeti rootkitekkel szemben, melyek egyszerűen csak helyettesítették a rendszerfájlokat saját binárisaikkal. Mivel azóta a fájlrendszerből a memóriába mozogtak a rootkitek, s képesek fájlolvasási műveleteknél a valóstól eltérő képet mutatni, lényegében használhatatlanná tették ezt a fajta védelmet.

5. Konklúzió

Kétségtelen, hogy egyre újabb és újabb ötletek kerülnek napvilágra, s egyre mélyebben nyúlnak bele az operációs rendszerekbe. Ezt kombinálva a vírusterjedési módszerekkel igen veszélyes kártevőt kapunk. Nem elég, hogy gyorsan terjed, de fertőzés után még szinte érzékelhetetlen is lesz. Ráadásul egyáltalán nem biztos, hogy érzékelés után biztosan el tudjuk távolítani, hiszen kernel modulokat kivenni egy rendszerből nem mindig egyszerű megoldás.

Ezen felüll az egyes vírusvédelmi szoftvereknek a rendszer stabilitásával is törődniük kell mielőtt akcióba lépnek. Amiatt pedig, hogy nem bízhatnak meg a kapott adatokban, sokkal mélyebbre kell majd nyúlniuk a rendszerben, ez pedig egyre nehezebb lesz...

Terjedelmi okokból a Linux operációs rendszeren futó rootkiteket nem tárgyaltuk. A teljes cikk elérhető a http://viruslab.ik.bme.hu/ honlapon.

Irodalom

[1] Ed Skoudis, Lenny Zeltser:
Malware: Fighting Malicious Code,
Prentice Hall PTR, 2003.

[2] Greg Hoglund, Jamie Butler:
ROOTKITS, Subverting the Windows Kernel,
Addison Wesley Professional, 2005.

[3] Joanna Rutkowska:
Thoughts about Cross-View based Rootkit Detection,
http://invisiblethings.org/papers/
crossview_detection_thoughts.pdf, 2005.

[4] Stuart Staniford, Vern Paxson, Nicholas Weaver:
How to 0wn the Internet in Your Spare Time,
11th USENIX Security Symposium, 2002.

[5] Tom Vogt:
Simulating and Optimalising Worm Propagation
Algorithms,
http://www.megasecurity.org/papers/
WormPropagation.pdf, 2003.

[6] James Butler, Sherri Sparks:
Windows rootkits of 2005,
http://www.securityfocus.com/infocus/1854, 2006.

[7] James Butler, Peter Silberman:
RAIDE: Rootkit Analysis Identification Elimination,
BlackHat Europe 2006.

Biztonsági tesztelés forráskód alapú hibainjektálással

TÓTH GERGELY, HORNÁK ZOLTÁN

SEARCH-LAB Kft.
{gergely.toth, zoltan.hornak}@search-lab.hu

Lektorált

Kulcsszavak: biztonsági tesztelés, hibainjektálás, forráskód alapú tesztelés

Az informatikai rendszerekben megbújó kihasználható biztonsági szoftverhibák hatalmas károkat okoznak világszerte. A bemutatásra kerülő biztonsági hibákat hibainjektálás alapú teszteléssel felderítő módszer célja, hogy költséghatékony alternatívát nyújtson a jelenlegi drága biztonság-növelő lehetőségeknek, a formális validálásnak, illetve a kimerítő tesztelésnek. Az új módszer lényege, hogy a használt adatstruktúrák leírójára támaszkodva képes generikus teszt algoritmusokat futtatni, melyek közvetlenül a tesztelni kívánt kódrészletekbe injektálnak automatizáltan generált teszt vektorokat, majd képes megfigyelni, hogy ezen szélsőséges esetekben a vizsgált rendszer le tudja-e kezelni a hibát, vagy biztonsági hibához hasonló reakciót vált-e az ki.

1. Bevezetés

A programozói hibákat három csoportra oszthatjuk:

(1) általános, *funkcionalitást befolyásoló hibák*;

(2) *biztonsági szempontból veszélyes hibák,* melyek bizonyos esetekben azt okozhatják, hogy az adott rendszer biztonsági tulajdonságai sérülnek (de ebben a kategóriában még nem feltétlenül kihasználható a hiba, csupán fennáll annak veszélye);

(3) végül pedig *kihasználható biztonsági lyukak*, melyeken keresztül közvetlen támadás intézhető a rendszer ellen.

A gyakorlatban akkor nevezhetünk egy rendszert biztonságosnak, ha nincs benne kihasználható biztonsági lyuk. Ennek bizonyításához azonban komplex formális módszerek vagy kimerítő tesztelés szükséges.

Ha azonban nem csak közvetlenül a kihasználható biztonsági lyukak felderítésére koncentrálunk, hanem azok okait, a biztonsági szempontból veszélyes hibákat kívánjunk felfedezni és kiküszöbölni, akkor – bár látszólag szélesebb körű tesztelést kell végrehajtani – költséghatékonyabban aránylag magas fokú biztonságot tudunk elérni.

A leggyakoribb biztonsági szempontból veszélyes hiba típusokhoz ugyanis léteznek algoritmusok, melyek ezeket költséghatékonyan, mégis nagy megbízhatósággal detektálni tudják és a legtöbb felhasználási területen az így elérhető biztonsági szint is lényeges javulást eredményezne.

A bemutatásra kerülő módszer – illetve a *Flinder* keretrendszer – lényege, hogy a tesztelendő programban (Target of Evaluation, ToE) felkutassa a biztonsági szempontból veszélyes hibákat, mielőtt azok kihasználható biztonsági lyukakká válnának. A Flinder erre két lehetőséget biztosít: az úgynevezett black-box (amikor a forráskód nem ismert) és a forráskód alapú tesztelést.

1.1. Black-box tesztelés

Black-box tesztelés során a Flinder a ToE és az úgynevezett Input Generátor (IG) eszköz közötti (például egy kliens-szerver alkalmazás esetén a kliens és a szerver közötti) üzeneteket manipulálja oly módon, hogy az algoritmikusan módosított teszt vektorok minél nagyobb valószínűséggel okozzanak anomáliát (például lefagyást vagy védelmi hibát) a ToE-ben, amit utána detektálni lehet és így fény derülhet a biztonsági hibára.

A Flinder az üzenetmódosítást a következőképpen végzi: beépül a tesztelendő program (ToE) és az azt input üzenetekkel működésre bíró Input Generátor (IG) közé, majd megfigyeli és manipulálja a közöttük folyó kommunikációt. Miután a ToE és IG közötti üzeneteket a Flinder elkapta, szükség van a nyers bináris formátum értelmezésére, egységes formátumra való transzformációjára. A Flinder *értelmező* (parser) moduljának feladata, hogy a bináris üzenetet egy belső, általános fa struktúrába, úgynevezett MSDL-be (Message Structure Description Language) konvertálja. Ehhez egy *formátum leíróra* van szüksége, ami a bináris üzenetek formátumát specifikálja, ez pedig az MFDL (Message Format Description Landuage) követelményeinek megfelelő XML dokumentum (egy példát az *2. ábrán* mutatunk be).

Ezek után a konkrét teszt vektort a megfelelő algoritmus az MSDL módosításával generálja, amit a *szerializáló* modul (az értelmező ellentettje) transzformál visszabináris formába, ami így jut el a címzetthez, a ToE-hez. A tesztelés és a kommunikáció során használt protokoll állapotát a Flinder egy *állapotgépben* tárolja, melynek segítségével nem csak egy kérés-válasz jellegű kommunikációt, hanem több üzenetváltást is magába foglaló protokollt is értelmezni tud. Fontos megjegyezni, hogy az MSDL általános volta miatt a konkrét bináris formátumtól függetlenül generikus tesztelő algoritmusok futhatnak (így tehát ugyanaz az algoritmus tud futni a DER kódolású üzenetek tesztelésénél és a szöveg alapú HTTP-nél is).

1.2. Forráskód alapú tesztelés

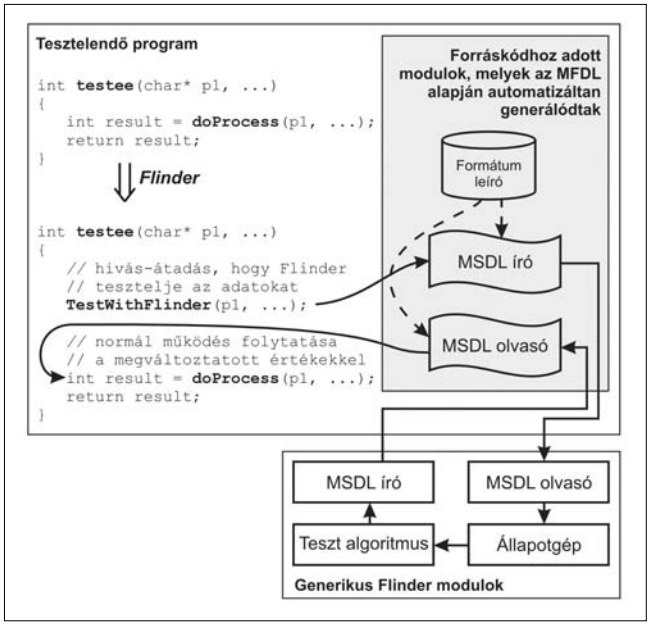
Forráskód alapú tesztelés során a cél, hogy a tesztelő algoritmusok a teszt vektorokat a program belső függvényeibe (például mint függvényparaméterek) tudják bejuttatni, a mi szóhasználatunkban *injektálni*. Ehhez a munkamódszer adaptálásra szorul: az architektúrát úgy kell módosítani *(1. ábra)*, hogy a tesztelni kívánt programhoz hozzá kell fordítani az MFDL formátum leíró alapján automatizáltan generált forráskódú értelmezőt (MSDL író) és szerializálót (MSDL olvasó), melyek képesek a Flinder számára a tesztelendő adatstruktúrákat MSDL formátumba átírni (MSDL író), majd a teszt vektort a Flindertől fogadva az adott nyelv belső adatstruktúrájának megfelelően visszafordítani (MSDL olvasó). Maga a teszt vektor előállítása így továbbra is általános: eredeti MSDL-ből kell az algoritmikusan generált teszt MSDL-t előállítani.

A black-box tesztelés menetét és a felmerült problémákat [1]-ben publikált írásunk elemzi, ebben a cikkben a forráskód alapú tesztelésre fókuszálunk, mivel ez integrálható legjobban a szoftverfejlesztési életciklusba. Itt kell megemlíteni, hogy a Flinder tervezésénél fontos szempont volt, hogy minél több modult tudjunk mind black-box, mind pedig forráskód alapú tesztelésnél is használni – ezért is hasonlít a két megközelítés architektúrája.

2. A Flinder fő tulajdonságai

A Flinder fő célja a biztonsági szempontból veszélyes hibák felderítése függetlenül attól, hogy azok kihasználhatók-e, vagy sem. Erre a feladatra számos módszer és termék létezik már, azonban a Flinder több újdonsággal is rendelkezik, melyek automatizmusokkal és új megközelítésekkel csökkentik a tesztelők által végrehajtandó munka mennyiségét:

1. ábra Forráskód alapú hibainjektálás Flinderrel



• *Formátum leíró alapú tesztelés:*

A Flinder által alkalmazott módszer lényege, hogy a teszt vektorok előállításához csak a formátum leírót használja, melyet más leírónyelvekből (pl. XML Schema [4]) akár automatizáltan is elő lehet állítani. Bár az MFDL pontos specifikációját terjedelmi okokból ez a cikk nem tartalmazza, egy könnyen megérthető példát az *2. ábra* tartalmaz.

A már meglevő módszerekkel szemben a nem megfelelő működés felismeréséhez a Flinder nem igényli a helyes működés részletes specifikációját – a rendszer képes a teszt vektorokat helyes bemenetek és a formátum leírók alapján generálni.

• *Generikus, cserélhető teszt algoritmusok:*

Az architektúra további előnye, hogy a tesztelő algoritmusok generikus adatstruktúrán, az MSDL-en dolgoznak, így nem szükséges azok adaptációja az éppen tesztelni kívánt környezethez, ezek az algoritmusok úgynevezett *plug-in* rendszerben működnek a Flinderen belül. Fontos hangsúlyozni, hogy számos gyakori biztonsági hiba osztályhoz lehet ilyen általános tesztelő algoritmust készíteni (puffertúlcsordulás, egész szám túlcsordulás, vagy kódolási hibák detektálására).

• *Állapotgép:*

Megemlítendő még, hogy a Flinder bonyolultabb tesztek futtatásának segítéséhez egy úgynevezett *protokoll állapotgépet* is futtat, mely az éppen tesztelni kívánt forgatókönyv *UML state machine* [2] alapú leírását tartalmazza. Ily módon a rendszer képes komplex folyamatok illetve protokollok követésére is. Forráskód alapú tesztelésnél ez az állapotgép használható akár a program futásának nyomon követésére is.

• *Kriptogrammok támogatása:*

Flinder annak érdekében, hogy kriptográfiai eszközöket felvonultató protokollok implementációját is tesztelni tudja (pl. SSL) támogatja a különböző kriptográfiai primitíveket (lenyomatkészítés, titkosítás, digitális aláírás stb.). Az MFDL megfelelő kialakításával az értelmező képes például egy eredetileg titkosított üzenetet először dekódolni, majd tovább értelmezni (természetesen a szerializáló is támogatja ezeket a műveleteket), így akár egy teljesen rejtjelezett protokoll implementációja is tesztelhető a Flinderrel.

3. Forráskód alapú tesztelés Flinderrel

A Flinder általános ismertetése után most következzék a konkrét tesztelési módszer ismertetése:

1. Első lépésként el kell készíteni a tesztelendő adatstruktúráknak megfelelő formátum leírót. Egy egyszerű példát C nyelvű adatstruktúrákhoz a 2. ábrán láthatunk (a következő oldalon).

2. Ezután az MFDL alapján a Flinder olyan függvények forráskódját generálja, melyek képesek az MFDL-nek megfelelő adatstruktúra illetve objektum példányokat MSDL-lé konvertálni, illetve MSDL alapján egy konkrét példány változóit értékekkel feltölteni.

3. A generált forráskódot a tesztelendő programhoz fordítva előáll ezeket a belső kapcsolódási lehetőségeket (úgynevezett *hook*-okat) tartalmazó ToE, melyet a Flinder különböző konfigurációkkal fog futtatni.

4. A Flinder futtatja a ToE-t, majd a megfelelő pontokon a tesztelő algoritmus módosítja a generált MSDL-t. A teszt vektor visszainjektálása után Flinder figyeli a korrekt futást (nem terminálódik-e abnormálisan a ToE, az állapotgépnek megfelelő állapotba kerül-e a tesztelt program stb.)

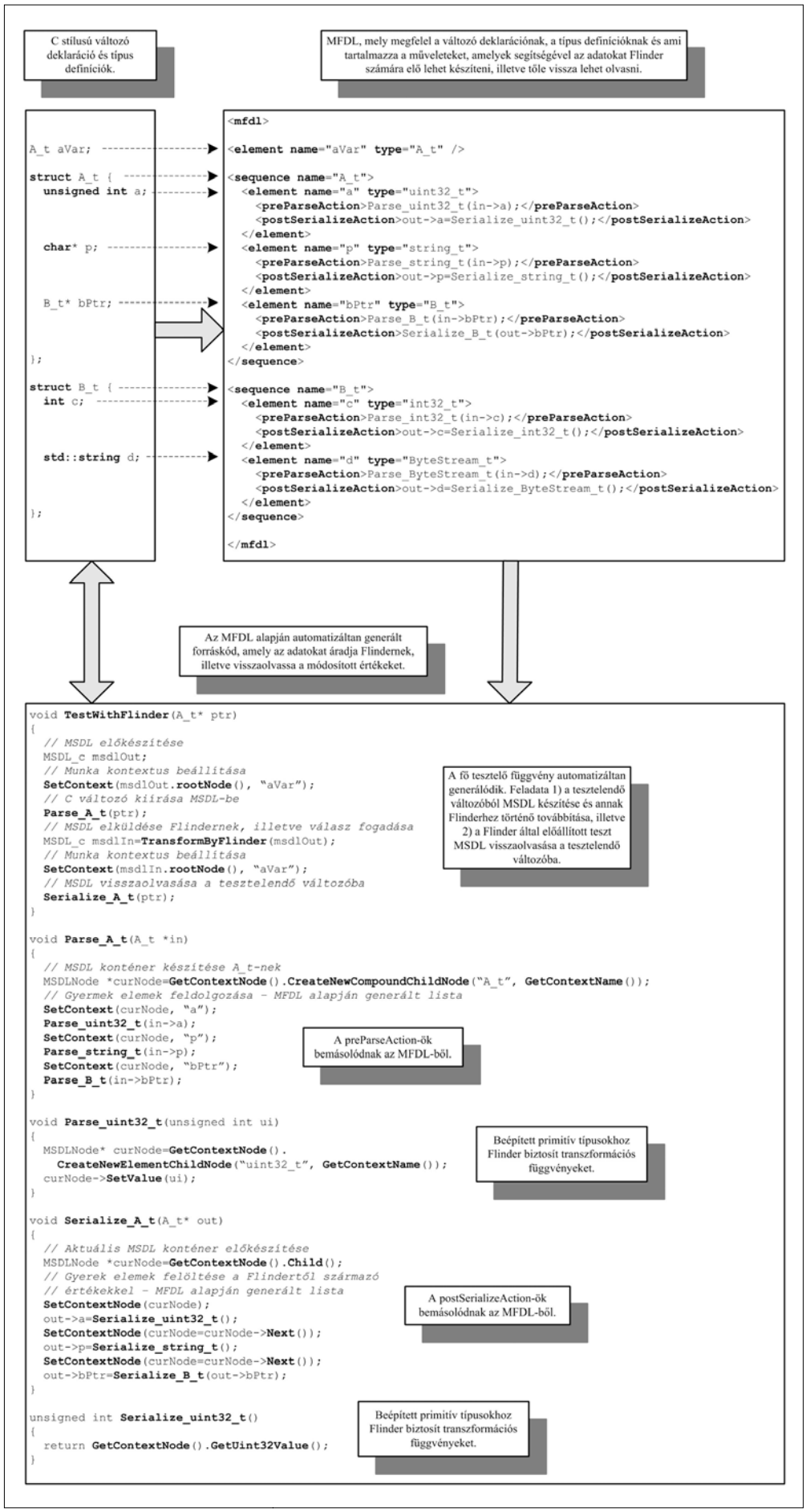
Ezt a lépést a Flinder többször is végrehajthatja, amennyiben egy többlépéses, úgynevezett iteratív tesztelő algoritmust hajt végre.

5. Végül a tesztelések eredményeiről a Flinder jegyzőkönyvet készít, melyben részletesen összefoglalja a végzett teszteket, rögzíti a felhasznált teszt vektorokat és a ToE reakcióit.

3.1. MFDL példa

A *2. ábrán* a C nyelvű adatstruktúra-MFDL megfeleltetés kerül bemutatásra, valamint részlet látható az MFDL alapján generált C nyelvű MSDL író és olvasó funkciókból.

Fontos megemlíteni, hogy bár az MFDL automatikusan generálható a típus definíciókból, a tesztelő számára megtartottuk annak lehetőségét, hogy az MSDL-változó konverziót befolyásolja.



2. ábra
Példa adatstruktúra, hozzá tartozó MFDL és generált kódrészlet

Erre szolgálnak az MFDL-ben elhelyezendő *preParseAction* és *postSerializeAction* mezők, melyek olyan, a forráskód nyelvének megfelelő kódrészleteket tartalmazhatnak, amik az adott mező konvertálását hivatottak elvégezni.

3.2. Teszt algoritmus példa

Az MFDL példát folytatva tekintsünk át egy egyszerű tesztelő algoritmust. Tételezzük fel, hogy a *3. ábrán* látható tipikus, puffer túlcsordulásos biztonsági hibát tartalmazó kódrészletet teszteljük a Flinderrel.

```
void ErroneousFunction(A_t *aPtr)
{
  // Flinder tesztelés
  TestWithFlinder(aPtr);

  // 10 bájt hosszú lokális tömb
  char locBuf[10];

  // Amennyiben aPtr->p hosszabb, mint 10 bájt,
  // úgy ez a másolás a hosszellenőrzés mellőzése
  // miatt felülírhatja a stacken tárolt függvény
  // visszatérési címet, ami biztonsági hibát
  // eredményez.
  strcpy(locBuf, aPtr->p);
}
```

3. ábra Puffertúlcsordulásos hibát tartalmazó függvény

Amennyiben a fenti függvény futtatása során az A_t típusú struktúra p mezője 10 bájtnál hosszabb sztringre mutat, úgy a másolás túl fogja írni a lokális fix méretű tömböt. A túlírás adott esetben elérheti a veremben tárolt vezérlési adatstruktúrákat is, sőt akár a függvény visszatérési címét is módosíthatja. Ily módon egy támadó eltérítheti a program futását a függvényből való visszatéréskor. Ha pedig nem támadó szándékkal véletlenszerű értékkel íródik felül ez a terület, akkor a program nagy valószínűséggel védelmi hibát fog okozni. (A puffertúlcsordulásos hibák veszélyéről és kihasználásuk módszeréről [3] részletes áttekintést nyújt.)

A Flinder ezen hibatípus felderítésére egy egyszerű algoritmust használ: az MFDL alapján a kapott MSDL-ben megkeresi a változó méretű mezőket (a példánkban ezek az A_t típus p tagváltozója és a B_t típus d tagváltozója) és minden meghíváskor megduplázza azok méretét (így például p tartalma először „XX”, majd „XXXX” stb. lesz).

Ez az algoritmus gyorsan képes a programozók által feltételezett értéktartományon kívülre jutni és túlméretes értékeket generálni, amik a méretellenőrzés hiánya vagy hibája miatt védelmi hibát okozhatnak. A hibásan lekezelt vagy nem ellenőrzött méretkorlátokat így a Flinder már detektálni tudja, és fény derülhet a biztonsági hibára. Gyorsítási lehetőség még, ha a különböző mezőket egyszerre nyújtjuk, de opció lehet a mezők soros tesztelése is.

Fontos megjegyezni, hogy mivel a tesztalgoritmusok (ebben az esetben a puffertúlcsordulásos hibákat kereső algoritmus) az általános MSDL-en dolgoznak, ugyanaz az algoritmus alkalmazható függetlenül attól, hogy mi is volt a konkrét bemenet. Terjedelmi okokból további algoritmusok ismertetését mellőzzük, de hasonló elven számos gyakori típushibára készíthető effektív algoritmus, melyek együttes alkalmazása lényegesen javíthatja a tesztelt programok biztonsági tulajdonságait.

4. Összefoglalás

A cikkben biztonsági hibák automatizált felderítésére mutattuk be a Flinder keretrendszert, mely forráskód alapú hibainjektáláson alapul.

A módszer lényege, hogy egy állapotgéppel követett programfutás közben generikus tesztelő algoritmusok képesek egy univerzális formátum leíró alapján a program belső függvényeibe teszt vektorokat injektálni, melyek célja a biztonsági hibák miatti abnormális működés előidézése, amit aztán a Flinder detektálni tud. A Flinder újdonsága, hogy a teszteléshez nem szükséges a helyes működés formális specifikációja, a tesztelőnek helyes bemeneteket, a kommunikáció állapotgépét (UML) és egy XML-alapú formátumleírót kell csak elkészítenie. Ezek alapján már számos gyakori biztonsági hiba típusra automatizáltan végezhető el a tesztelés.

Köszönetnyilvánítás

A projektet a Gazdasági Versenyképesség Operatív Program támogatta (GVOP-3.3.1-2004-04- 0094/3.0).

Irodalom

[1] Tóth G., Hornák Z.:
Semi-automated detection of security-relevant programming bugs with Flinder, 2006.
www.flinder.hu
[2] Object Management Group:
UML – Unified Modeling Language, 2005.
[3] Aleph One: Smashing the stack for fun and profit, Phrack 7, 1996.
[4] W3C – World Wide Web Consortium:
XML Schema, 2004.

WiFi biztonság – A jó, a rossz és a csúf

BUTTYÁN LEVENTE, DÓRA LÁSZLÓ

BME Híradástechnikai Tanszék, CrySyS Adatbiztonsági Laboratórium
{buttyan, doralaca}@crysys.hu

Lektorált

Kulcsszavak: WLAN, WEP, 802.11i, 802.1x, EAP, TKIP, CCMP, AES, hitelesítés, hozzáférés-védelem, integritás-védelem, rejtjelezés

Napjainkban a vezeték nélküli hálózatok egyre nagyobb teret nyernek. A vezeték nélküli hálózatok nagy előnye az eszközök, s ezzel a felhasználók mobilitásának támogatása, hátrányuk viszont, hogy – a fizikai kapcsolatok hiánya és a rádiós csatorna jellege miatt – több potenciális támadásnak vannak kitéve, mint vezetékes társaik általában. Fontos tehát, hogy a vezeték nélküli hálózatok megfelelő védelmi mechanizmusokkal legyenek ellátva, melyek minden körülmények között (azaz rosszindulatú támadások esetén is) biztosítják a biztonságos működést. Ebben a cikkben a 802.11 vezeték néküli LAN szabványhoz, közismertebb nevén a WiFi-hez kapcsolódó biztonsági problémákkal és megoldásokkal foglalkozunk.

1. Bevezetés

Cikkünk ismeretterjesztő jellegű, saját eredményeket nem tartalmaz, csupán tömör áttekintést ad a WiFi biztonsághoz kapcsolódó szabványokról és a mások által elért tudományos és gyakorlati eredményekről. Az olvasó a tématerület bővebb kifejtését [1]-ben találja.

A továbbiakban először a WEP működését és hibáit mutatjuk be. A WEP az első biztonsági architektúra, melyet 802.11 hálózatok számára javasoltak, ám hamar kiderült, hogy nem nyújt megfelelő védelmet. Utána a 802.11i szabványt ismertetjük, mely a WEP utódjának tekinthető. Áttekintjük a 802.11i-ben javasolt biztonsági architektúra elemeit: a hitelesítési és hozzáférés-védelmi mechanizmust, a kulcsmenedzsmentet, valamint a TKIP és az AES-CCMP protokollokat.

2. WEP

Az IEEE 802.11 vezeték nélküli LAN szabvány tervezői kezdettől fogva fontosnak tartották a biztonságot. Ezért már a 802.11 korai verziója [2] is tartalmazott biztonsági mechanizmusokat, melyek összességét WEP-nek (Wired Equivalent Privacy) nevezték el. Ahogy arra a név is utal, a WEP célja az, hogy a vezeték nélküli hálózatot *legalább* olyan biztonságossá tegye, mint egy – különösebb biztonsági kiegészítésekkel nem rendelkező – vezetékes hálózat. Ha például egy támadó egy vezetékes Ethernet hálózathoz szeretne csatlakozni, akkor hozzá kell férnie az Ethernet hub-hoz. Mivel azonban a hálózati eszközök általában fizikailag védve, zárt szobában találhatók, ezért a támadó nehézségekbe ütközik. Ezzel szemben egy védelmi mechanizmusokat nélkülöző vezeték nélküli LAN-hoz való hozzáférés – a rádiós csatorna nyitottsága miatt – triviális feladat a támadó számára. A WEP ezt a triviális feladatot hivatott megnehezíteni. Fontos azonban megjegyezni, hogy a WEP tervezői nem törekedtek „tökéletes" biztonságra, mint ahogy a zárt szoba sem jelent tökéletes védelmet egy Ethernet hub számára.

A tervezők tehát nem tették túl magasra a lécet, ám a WEP még ezt a korlátozott célt sem érte el. Pár évvel a megjelenése után, a kriptográfusok és az IT biztonsági szakemberek súlyos biztonsági hibákat találtak a WEP-ben [3,4,5], s nyílvánvalóvá vált, hogy a WEP nem nyújt megfelelő védelmet. A felfedezést tett követte, és hamarosan megjelentek az Interneten a WEP feltörését automatizáló programok. Válaszul az IEEE új biztonsági architektúrát dolgozott ki, amit a 802.11 szabvány *i* jelzésű kiegészítése tartalmaz [6], melyet később tárgyalunk. Ebben fejezetben a WEP működését és hibáit tekintjük át. Ezt azért tartjuk szükségesnek, mert – bár a WEP-en már túlhaladt a kor – a legtöbb forgalomban levő hálózati eszköz még mindig támogatja a WEP-et. Azok a felhasználók, akik WEP-et használnak, jobb ha tisztában vannak annak korlátaival.

2.1. A WEP működése

Vezeték néküli hálózatok esetében két alapvető biztonsági probléma merül fel. Egyrészt a rádiós csatorna jellege miatt a kommunikáció könnyen lehallgatható. Másrészt – s ez talán fontosabb – a hálózathoz való csatlakozás nem igényel fizikai hozzáférést a hálózati csatlakozóponthoz (Access Point, vagy röviden AP), ezért bárki megpróbálhatja a hálózat szolgáltatásait illegálisan igénybe venni. A WEP az első problémát az üzenetek rejtjelezésével igyekszik megoldani, a második probléma megoldása érdekében pedig megköveteli a csatlakozni kívánó mobil eszköz (Station, vagy röviden STA) hitelesítését az AP felé.

A hitelesítést egy egyszerű kihívás-válasz alapú protokoll végzi, mely négy üzenet cseréjéből áll. Elsőként a STA jelzi, hogy szeretné hitelesíteni magát (authenticate request). Válaszul az AP generál egy véletlen számot, s azt kihívásként elküldi a STA-nak (authenticate challenge). A STA rejtjelezi a kihívást, s az eredményt visszaküldi az AP-nak (authenticate response). A STA a

rejtjelezést egy olyan titkos kulccsal végzi, melyet csak a STA és az AP ismer. Ezért ha az AP sikeresen dekódolja a választ (azaz a dekódolás eredményeként visszakapja saját kihívását), akkor elhiszi, hogy a választ az adott STA állította elő, hiszen csak az ismeri a helyes válasz generálásához szükséges titkos kulcsot. Más szavakkal, a válasz sikeres dekódolása esetén az AP hitelesítette a STA-t, és ennek megfelelően dönthet arról, hogy a csatlakozást engedélyezi vagy sem. A döntésről az AP a protokoll negyedik üzenetében tájékoztatja a STA-t (authenticate success vagy failure).

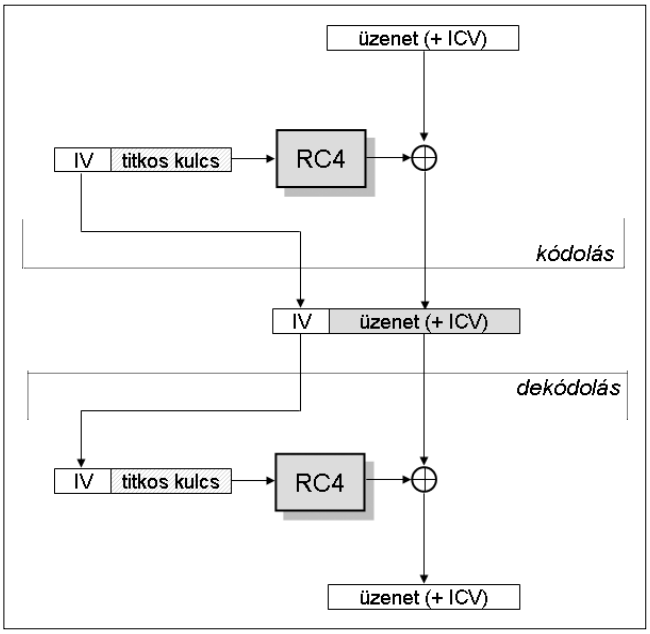
Miután a hitelesítés megtörtént, a STA és az AP üzeneteiket rejtjelezve kommunikálnak. A rejtjelezéshez ugyanazt a titkos kulcsot használják, mint a hitelesítéshez. A WEP rejtjelező algoritmusa az RC4 kulcsfolyam kódoló. A kulcsfolyam kódolók úgy működnek, hogy egy kis méretű, néhány bájtos titkos kulcsból egy hosszú álvéletlen bájtsorozatot állítanak elő, és ezen sorozat bájtjait XOR-olják az üzenet bájtjaihoz. Ez történik a WEP esetében is. Az M üzenet küldője (a STA vagy az AP) a titkos kulccsal inicializálja az RC4 kódolót, majd az RC4 által előállított K álvéletlen bájtsorozatot XOR-olja az üzenethez. Az $M \oplus K$ rejtjelezett üzenet vevője ugyanazt teszi mint a küldő: a titkos kulccsal inicializálja az RC4 algoritmust, amely így ugyanazt a K álvéletlen bájtsorozatot állítja elő, amit a rejtjelezéshez használt a küldő. Ezt a rejtjelezett üzenethez XOR-olva – az XOR művelet tulajdonságai miatt – a vevő az eredeti üzenetet kapja vissza: $(M \oplus K) \oplus K = M$.

A fent leírtak majdnem megfelelnek a valóságnak, van azonban még valami, amit a WEP rejtjelezés kapcsán meg kell említeni. Könnyen látható, hogy ha a rejtjelezés a fentiek szerint működne, akkor minden üzenethez ugyanazt a K álvéletlen bájtsorozatot XOR-olnánk, hiszen a kódolót minden üzenet elküldése előtt ugyanazzal a titkos kulccsal inicializáljuk. Ez több szempontból is hiba lenne.

Tegyük fel például, hogy egy támadó lehallgat két rejtjelezett üzenetet, $M_1 \oplus K$-t és $M_2 \oplus K$-t. A két rejtjelezett üzenetet XOR-olva, a támadó a két nyílt üzenet XOR összegét kapja: $(M_1 \oplus K) \oplus (M_2 \oplus K) = M_1 \oplus M_2$. Ez olyan, mintha az egyik üzenetet a másik üzenettel, mint kulcsfolyammal rejtjeleztük volna. Ám ebben az esetben M_1 és M_2 nem álvéletlen bájtsorozatok. Valójában tehát $M_1 \oplus M_2$ egy nagyon gyenge rejtjelezés, és a támadó az üzenetek statisztikai tulajdonságait felhasználva könnyen meg tudja fejteni mindkét üzenetet. Az is elképzelhető, hogy a támadó esetleg (részlegesen) ismeri az egyik üzenet tartalmát, s annak segítségével a másik üzenet (részleges) tartalmához azonnal hozzájut.

Ezen problémák elkerülése érdekében, a WEP nem egyszerűen a titkos kulcsot használja a rejtjelezéshez, hanem azt kiegészíti egy IV-nek (Initialization Vector) nevezett értékkel, mely üzenetenként változik. A rejtjelezés folyamata tehát a következő: az IV-t és a titkos kulcsot összefűzzük, a kapott értékkel inicializáljuk az RC4 kódolót, mely előállítja az álvéletlen bájtsorozatot, amit az üzenethez XOR-olunk. A dekódolás folyamata ezzel analóg. Ebből következik, hogy a vevőnek szüksége van a kódolásnál használt IV-re. Ez a rejtjelezett üzenethez fűzve, nyíltan kerül átvitelre. Ez elvileg nem jelent problémát, mert az üzenet dekódolásához csupán az IV ismerete nem elegendő, ahhoz a titkos kulcsot is ismerni kell. A méreteket illetően megemlítjük – s ennek később még lesz jelentősége – hogy az IV 24 bites, a titkos kulcs pedig (általában) 104 bites. (Különböző marketing anyagokban ezt gyakran úgy interpretálják, hogy a WEP „128 bites biztonságot” nyújt. Ez természetesen félrevezető – mint a marketing anyagok általában –, hiszen a 128 bitből 24 nyíltan kerül átvitelre.)

A WEP rejtjelezés teljes folyamatát az *1. ábra* szemlélteti.



1. ábra A WEP rejtjelezés folyamata

Az ábra azt is mutatja, hogy a rejtjelezés előtt, a küldő egy integritásvédő ellenőrző összeggel (Integrity Check Value, ICV) egészíti ki a nyílt üzenetet, melynek célja a szándékos módosítások detektálásának lehetővé tétele a vevő számára. A WEP esetében az ICV nem más, mint a nyílt üzenetre számolt CRC érték. Mivel azonban a CRC önmagában nem véd a szándékos módosítások ellen (hiszen egy támadó a módosított üzenethez új CRC értéket tud számolni), ezért a WEP a CRC értéket is rejtjelezi. A mögöttes gondolat az, hogy így a támadó nem tudja manipulálni az üzeneteket, hiszen a titkos kulcs hiányában nem tudja a módosított üzenethez tartozó rejtjelezett CRC értéket előállítani. Mint azt alább látni fogjuk, ez a gondolatmenet nem teljesen hibamentes.

Végezetül a WEP kulcsokról szólunk röviden. A szabvány lehetővé teszi, hogy minden STA-nak saját titkos kulcsa legyen, amit csak az AP-vel oszt meg. Ez azonban megnehezíti a kulcsmenedzsmentet az AP oldalán, mivel ekkor az AP-nek minden STA kulcsát ismernie és gondoznia kell. Ezért a legtöbb implementáció nem támogatja ezt a lehetőséget. A szabvány előír egy úgynevezett default kulcsot is, amit az AP és a hálózat-

hoz tartozó *minden* STA ismer. Eredetileg ezt a kulcsot azon üzenetek védelmére szánták, melyeket az AP többesszórással (broadcast) minden STA-nak el szeretne küldeni. A legtöbb WEP implementáció azonban csak ezt a megoldást támogatja. Így a gyakorlatban egy adott hálózathoz tartozó eszközök egyetlen közös kulcsot használnak titkos kulcsként. Ezt a kulcsot manuálisan kell telepíteni a mobil eszközökben és az AP-ben. Nyílvánvaló, hogy ez a megoldás csak egy külső támadó ellen biztosítja a kommunikáció biztonságát; az eszközök (elvileg) dekódolni tudják egymás üzeneteit.

2.2. A WEP hibái

A WEP tulajdonképpen a rossz protokolltervezés mintapéldája. Az alábbi tömör összefoglalóból látható, hogy lényegében egyetlen kitűzött biztonsági célt sem valósít meg tökéletesen:

Hitelesítés: A WEP hitelesítési eljárásának több problémája is van. Elsőként mindjárt az, hogy a hitelesítés egyirányú, azaz a STA hitelesíti magát az AP felé, ám az AP nem hitelesíti magát a STA felé. Másodszor, a hitelesítés és a rejtjelezés ugyanazzal a titkos kulccsal történik. Ez azért nem kívánatos, mert így a támadó mind a hitelesítési, mind pedig a rejtjelezési eljárás potenciális gyengeségeit kihasználhatja egy, a titkos kulcs megfejtésére irányuló támadásban. Biztonságosabb lenne, ha minden funkcióhoz külön kulcs tartozna.

A harmadik probléma az, hogy a protokoll csak a hálózathoz történő csatlakozás pillanatában hitelesíti a STA-t. Miután a hitelesítés megtörtént és a STA csatlakozott a hálózathoz, bárki küldhet a STA nevében üzeneteket annak MAC címét használva. Úgy tűnhet, hogy ez annyira nem nagy gond, hiszen a támadó, a titkos kulcs ismeretének hiányában, úgysem tud helyes rejtjelezett üzenetet fabrikálni, amit az AP elfogad. Ám ahogy azt korábban említettük, a gyakorlatban az összes STA egy közös titkos kulcsot használ, s így a támadó megteheti azt, hogy egy STA_1 által küldött – és a támadó által lehallgatott – rejtjelezett üzenetet STA_2 nevében megismétel az AP felé; ezt az AP el fogja fogadni.

A negyedik probléma egy gyöngyszem a protokolltervezési hibák között. Emlékeztetünk arra, hogy a WEP rejtjelezési algoritmusa az RC4 folyamkódoló. Nemcsak az üzeneteket kódolják az RC4 segítségével, hanem a STA ezt használja a hitelesítés során is az AP által küldött kihívás rejtjelezésére. Így a támadó a hitelesítés során küldött üzenetek lehallgatásával könnyen hozzájut a C kihíváshoz és az arra adott $R = C \oplus K$ válaszhoz, melyből $C \oplus R = K$ alapján azonnal megkapja az RC4 algoritmus által generált K álvéletlen bájtsorozatot. A játéknak ezzel vége, hiszen K segítségével a támadó bármikor, bármilyen kihívásra helyes választ tud generálni a STA nevében (s ezen az IV használata sem segít, mert az IV-t a rejtjelezett üzenet küldője, jelen esetben a támadó választja). Sőt, mivel a gyakorlatban minden, az adott hálózathoz tartozó eszköz ugyanazt a titkos kulcsot használja, a támadó ezek után bármelyik eszköz nevében csatlakozni tud a hálózathoz. Persze a csatlakozás önmagában még nem elegendő, a támadó használni is szeretné a hálózatot. Ehhez olyan üzeneteket kell fabrikálnia, amit az AP elfogad. A rejtjelezés követelménye miatt ez nem triviális feladat (hiszen magához a titkos kulcshoz még nem jutott hozzá a támadó), de a WEP hibáinak tárháza bőven tartogat még lehetőségeket.

Integritás-védelem: A WEP-ben az üzenetek integritásának védelmét az üzenetekhez csatolt ellenőrző összeg (ICV) hivatott biztosítani. Az ICV nem más, mint az üzenetre számolt CRC érték, mely az üzenettel együtt rejtjelezésre kerül.

Formális jelöléseket használva, a rejtjelezett üzenet a következő módon írható fel: $(M \,||\mathrm{CRC}(M)) \oplus \mathrm{K}$, ahol M a nyílt üzenet, K az RC4 által az IV-ből és a titkos kulcsból előállított álvéletlen bájtsorozat, CRC(.) jelöli a CRC függvényt, és || jelöli az összefűzés műveletét. Ismeretes, hogy a CRC lineáris művelet az XOR-ra nézve, azaz $\mathrm{CRC}(X \oplus Y) = \mathrm{CRC}(X) \oplus \mathrm{CRC}(Y)$. Ezt kihasználva, a támadó a rejtjelezett WEP üzenetek bármely bitjét módosítani tudja (át tudja billenteni), bár magát az üzenetet nem látja. Jelöljük a támadó szándékolt módosításait ΔM-mel. Ekkor a támadó az $((M \oplus \Delta M) \,||\mathrm{CRC}(M \oplus \Delta M)) \oplus K$ rejtjelezett üzenetet szeretné előállítani az eredetileg megfigyelt $(M \,||\mathrm{CRC}(M)) \oplus K$ rejtjelezett üzenetből. Ehhez egyszerűen $\mathrm{CRC}(\Delta M)$-et kell kiszámolnia, majd a $\Delta M \,||\mathrm{CRC}(\Delta M)$ értéket kell az eredeti rejtjelezett üzenethez XOR-olnia.

A következő egyszerű levezetés megmutatja, hogy ez miért vezet célra:

$$((M \,||\, \mathrm{CRC}(M)) \oplus K) \oplus \Delta M \,||\, \mathrm{CRC}(\Delta M)) =$$
$$((M \oplus \Delta M) \,||\, (\mathrm{CRC}(M) \oplus \mathrm{CRC}(\Delta M))) \oplus K =$$
$$((M \oplus \Delta M) \,||\, \mathrm{CRC}(M \oplus \Delta M)) \oplus K$$

ahol az utolsó lépésben kihasználtuk a CRC linearitását. Mivel $\mathrm{CRC}(\Delta M)$ kiszámolásához nincs szükség a titkos kulcsra, ezért láthatóan a támadó könnyen tudja manipulálni a WEP üzeneteket, az integritás-védelem és a rejtjelezés ellenére.

Az üzenetfolyam integritásának védelme kapcsán szokás említeni az üzenetvisszajátszás detektálását, mint biztonsági követelményt. A WEP esetében ennek vizsgálatával egyszerű dolgunk van, mert a WEP-ben egyáltalán nincs semmilyen mechanizmus, mely az üzenetek visszajátszásának detektálását lehetővé tenné. A tervezők nemes egyszerűséggel erről a biztonsági követelményről megfeledkeztek. A támadó tehát bármely eszköz korábban rögzített üzenetét vissza tudja játszani egy későbbi időpontban, s ezt a WEP nem detektálja. Nyílvánvaló, hogy ez miért gond, ha arra gondolunk, hogy a rögzített üzenet akár egy bejelentkezési folyamatból is származhat, s például egy felhasználói név/jelszó párt tartalmazhat.

Titkosítás: Mint azt korábban említettük, folyamkódolók használata esetén fontos, hogy minden üzenet más kulccsal legyen rejtjelezve. Ezt a WEP-ben az IV használata biztosítja; sajnos nem teljesen megfelelő módon. A probléma abból adódik, hogy az IV csak 24 bites, ami azt jelenti, hogy mintegy 17 millió lehetséges IV van. Egy WiFi eszköz körülbelül 500 teljes hosszúságú keretet tud forgalmazni egy másodperc alatt, így

a teljes IV teret közel 7 óra leforgása alatt kimeríti. Azaz 7 óránként ismétlődnek az IV értékek, s ezzel az RC4 által előállított álvéletlen bájtsorozatok is. A problémát súlyosbítja, hogy a gyakorlatban minden eszköz ugyanazt a titkos kulcsot használja, potenciálisan különböző IV értékekkel, így ha egyszerre *n* eszköz használja a hálózatot, akkor az IV ütközés várható ideje a 7 óra *n*-ed részére csökken. Egy másik súlyosbító tényező, hogy sok WEP implementáció az IV-t nem véletlen értékről indítja, hanem nulláról. Ezért beindítás után a különböző eszközök ugyanazt a nullától induló és egyesével növekvő IV sorozatot használják, legtöbbször ugyanazzal a közös titkos kulccsal. Azaz, a támadónak várakoznia sem kell, azonnal IV ütközésekhez jut.

A WEP teljes összeomlását az RC4 kódoló nem megfelelő használata okozza. Ismeretes, hogy léteznek úgynevezett gyenge RC4 kulcsok, melyekre az a jellemző, hogy belőlük az RC4 algoritmus nem teljesen véletlen bájtsorozatot állít elő [7]. Ha valaki meg tudja figyelni egy gyenge kulcsból előállított bájtsorozat első néhány bájtját, akkor abból következtetni tud a kulcsra. Ezért a szakemberek azt javasolják, hogy az RC4 által előállított bájtsorozat első 256 bájtját mindig dobjuk el, s csak az utána generált bájtokat használjuk a rejtjelezéshez. Ezzel a gyenge kulcsok problémáját meg lehetne oldani. Sajnos a WEP nem így működik. Ráadásul a változó IV érték miatt előbb-utóbb biztosan gyenge kulcsot kap a kódoló, s az IV nyílt átvitele miatt, erről a támadó is tudomást szerezhet.

Ezt kihasználva, néhány kriptográfus olyan támadó algoritmust konstruált a WEP ellen, melynek segítségével a teljes 104 bites titkos kulcs néhány millió üzenet lehallgatása után nagy valószínűséggel megfejthető. A WEP minden korábban leírt hibája eltörpül ezen eredmény mellett, ugyanis ezzel a támadással magához a titkos kulcshoz jut hozzá a támadó. Ráadásul a támadás könnyen automatizálható, és néhány „segítőkész" embernek köszönhetően, az Internetről letöltött támadó programok használatával amatőrök által is rutinszerűen végrehajtható.

3. A 802.11i specifikáció

A WEP hibáit felismerve, az IEEE új biztonsági megoldást dolgozott ki, melyet a 802.11i specifikáció tartalmaz [6]. A WEP-től való megkülönböztetés érdekében, az új koncepciót RSN-nek (Robust Security Network) nevezték el. Az RSN-t körültekintőbben tervezték meg, mint a WEP-et. Új módszer került bevezetésre a hitelesítés és a hozzáférés-védelem biztosítására, mely a 802.1X szabvány által definiált modellre épül, az integritás-védelmet és a titkosítást pedig az AES (Advanced Encryption Standard) algoritmusra támaszkodva oldották meg.

Sajnos azonban az új RSN koncepcióra nem lehet egyik napról a másikra áttérni. Ennek az az oka, hogy a használatban levő WiFi eszközök az RC4 algoritmust támogató hardver elemekkel vannak felszerelve, és nem támogatják az RSN által előírt AES algoritmust. Ezen pusztán szoftver upgrade-del nem lehet segíteni, új hardverre van szükség, s ez lassítja az RSN elterjedésének folyamatát.

Ezt a problémát az IEEE is felismerte, és egy olyan opcionális protokollt is hozzáadott a 802.11i specifikációhoz, mely továbbra is az RC4 algrotimust használja, és így – szoftver upgrade után – futtatható a régi hardveren, de erősebb mint a WEP. Ezt a protokollt TKIP-nek (Temporal Key Integrity Protocol) nevezik.

A WiFi eszközöket gyártó cégek azonnal adaptálták a TKIP protokollt, hiszen annak segítségével a régi eszközökből álló WEP-es hálózatokat egy csapásra biztonságossá lehetett varázsolni. Meg sem várták amíg a 802.11i specifikáció a lassú szabványosítási folyamat során végleges állapotba kerül, azonnal kiadták a WPA (WiFi Protected Access) specifikációt [8], ami a TKIP-re épül. A WPA tehát egy gyártók által támogatott specifikáció, mely az RSN egy azonnal használható részhalmazát tartalmazza. A WPA-ban a hitelesítés, a hozzáférés-védelem, és a kulcsok menedzsmentje megegyezik az RSN-ben használt módszerekkel, a különbség csak az integritás-védelemre és a rejtjelezésre használt algoritmusokban mutatkozik.

A továbbiakban áttekintjük a 802.11i-ben definiált hitelesítési, hozzáférés-védelmi, és kulcsmenedzsment módszereket, melyek tehát megegyeznek az RSN-ben és a WPA-ban. Ezt követően röviden összefoglaljuk a TKIP (WPA) és az AES-CCMP (RSN) protokollok működését.

3.1. Hitelesítés és hozzáférés-védelem

A 802.11i-ben a hitelesítés és hozzáférés-védelem modelljét a 802.1X szabványból kölcsönözték [9]. Ezt a szabványt eredetileg vezetékes LAN-ok számára tervezték, de az elvek végülis vezeték nélküli WiFi hálózatokban is ugyanúgy alkalmazhatóak.

A 802.1X modell három résztvevőt különböztet meg a hitelesítés folyamatában: a hitelesítendő felet (supplicant), a hitelesítőt (authenticator), és a hitelesítő szervert (authentication server). A hitelesítendő fél szeretne a hálózat szolgáltatásaihoz hozzáférni, és ennek érdekében szeretné magát hitelesíteni, azaz kilétét bizonyítani. A hitelesítő kontrollálja a hálózathoz történő hozzáférést. A modellben ez úgy történik, hogy a hitelesítő egy úgynevezett port állapotát vezérli. Alapállapotban a porton adatforgalom nincs engedélyezve, ám sikeres hitelesítés esetén a hitelesítő „bekapcsolja" a portot, ezzel engedélyezve a hitelesítendő fél adatforgalmát a porton keresztül. A hitelesítő szerver az engedélyező szerepet játsza. Tulajdonképpen a hitelesítendő fél hitelesítését nem a hitelesítő, hanem a hitelesítő szerver végzi (ebből a szempontból az elnevezések kicsit szerencsétlenek), és ha a hitelesítés sikeres volt, engedélyezi, hogy a hitelesítő bekapcsolja a portot.

WiFi hálózatok esetében a hitelesítendő fél a mobil eszköz, mely szeretne a hálózathoz csatlakozni, a hitelesítő pedig az AP, mely a hálózathoz történő hozzáférést kontrollálja. A hitelesítő szerver egy program, mely

kisebb hálózatok esetében akár az AP-ben is futhat, nagyobb hálózatoknál azonban tipikusan egy külön erre a célra dedikált hoszton futó szerver alkalmazás. WiFi esetében a port nem egy fizikai csatlakozó, hanem egy logikai csatlakozási pont, amit az AP-ben futó szoftver valósít meg.

Vezetékes LAN esetében, a hitelesítendő fél egyszer hitelesíti magát, mikor fizikailag csatlakozik a hálózathoz. További védelmi lépésekre nincsen szükség (legalábbis hozzáférés-védelem tekintetében), hiszen a használatba vett portot más úgysem használhatja. Ahhoz ugyanis a hitelesítendő fél és a hitelesítő között létrejött fizikai kapcsolatot kellene megbontani (például a hálózati csatlakozót kihúzni egy Ethernet hub-ból). Ezt a hitelesítő hardvere detektálja és a portot azonnal letiltja. WiFi esetében más a helyzet, mert nincsen fizikai kapcsolat a STA és az AP között.

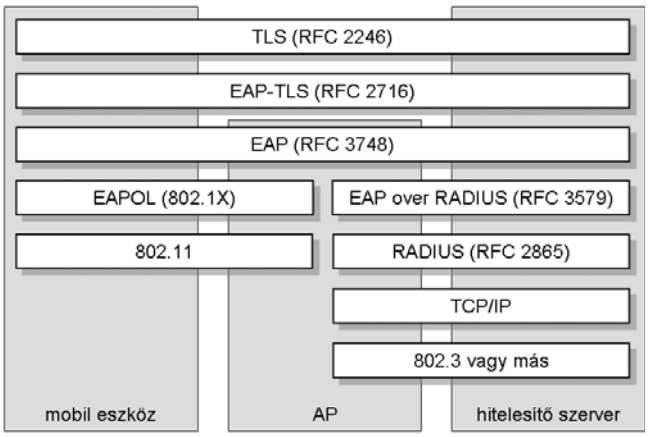
Ezért a 802.11i azzal a követelménnyel egészíti ki a 802.1X modellt, hogy a hitelesítés során létre kell jöjjön egy titkos kulcs a STA és az AP között, melyet azok a további kommunikáció kriptográfiai védelmére használhatnak. Ezen kulcs hiányában egy támadó nem tudja a STA és az AP között kialakult logikai kapcsolatot csalárd módon saját céljainak megvalósítására felhasználni.

Maga a hitelesítés az EAP (Extensible Authentication Protocol) segítségével történik [10]. Az EAP egy igen egyszerű protokoll, aminek az az oka, hogy nem maga az EAP végzi a hitelesítést. Az EAP csak egy illesztő-protokoll, amit arra terveztek, hogy tetszőleges hitelesítő protokoll üzeneteit szállítani tudja (ezért „extensible”). Egy adott hitelesítő protokoll EAP-ba történő beágyazásának szabályait külön kell specifikálni. Több elterjedt hitelesítő protokollra létezik már ilyen specifikáció (pl. EAP-TLS, LEAP, PEAP, EAP-SIM).

Négy fajta EAP üzenet létezik: request, response, success, és failure. Az EAP request és response üzenetek szállítják a beágyazott hitelesítő protokoll üzeneteit. Az EAP success és failure speciális üzenetek, melyek segítségével a hitelesítés eredményét lehet jelezni a hitelesítendő fél felé.

Ahogy azt fentebb említettük, a 802.1X modellben, a hitelesítendő fél hitelesítését a hitelesítő szerver végzi. WiFi esetében ez azt jelenti, hogy az EAP protokollt (és az abba beágyazott tényleges hitelesítő protokollt, például a TLS-t) lényegében a mobil eszköz és a hitelesítő szerver futtatják. Az AP csak továbbítja az EAP üzeneteket a mobil eszköz és a hitelesítő szerver között, de nem érti azok tartalmát. Az AP csak az EAP success és failure üzeneteket érti meg, ezeket figyeli, és ha success üzenetet lát, akkor engedélyezi a mobil eszköz csatlakozását a hálózathoz.

Az EAP üzeneteket a mobil eszköz és az AP között a 802.1X-ben definiált EAPOL (EAP over LAN) protokoll szállítja. Az AP és a hitelesítő szerver között a WPA a RADIUS protokoll [11] használatát írja elő. A RADIUS-t az RSN opcióként ajánlja, de más alkalmas protokoll használatát is lehetővé teszi a specifikáció. Végülis tetszőleges protokoll használható, amely az EAP üzenetek szállítására alkalmas. Elterjedtsége miatt azonban várhatóan a legtöbb hálózat RADIUS-t használ majd. Az így kialakuló protokoll architektúrát a *2. ábra* szemlélteti.



2. ábra
Hitelesítési protokoll-architektúra a 802.11i-ben TLS használata esetén

Ahogy azt korábban említettük, a hitelesítés eredményeként nemcsak a hálózathoz való hozzáférést engedélyezi a hitelesítő szerver, hanem egy titkos kulcs is létrejön, mely a mobil eszköz és az AP további kommunikációját hivatott védeni. Mivel a hitelesítés a mobil eszköz és a szerver között folyik, ezért a protokoll futása után ezt a kulcsot csak a mobil eszköz és a hitelesítő szerver birtokolja, és azt még el kell juttatni az AP-hez. A RADIUS protokoll biztosít erre használható mechanizmust az MS-MPPE-Recv-Key RADIUS üzenet-attribútum formájában, mely kifejezetten kulcsszállítás céljára lett specifikálva. A kulcs rejtjelezett formában kerül átvitelre, ahol a rejtjelezés egy a hitelesítő szerver és az AP között korábban létrehozott (tipikusan manuálisan telepített) kulcs segítségével történik.

3.2. Kulcsmenedzsment

A hitelesítés során, a mobil eszköz és az AP között létrehozott titkos kulcsot páronkénti mesterkulcsnak (Pairwise Master Key, PMK) nevezik. Azért „páronkénti”, mert csak az adott mobil eszköz és az AP ismeri (na meg a hitelesítő szerver, de az megbízható entitásnak tekinthető), s azért „mester”, mert ezt a kulcsot nem használják közvetlenül rejtjelezésre, hanem további kulcsokat generálnak belőle. Egészen pontosan a PMK-ból mind a mobil eszköz, mind pedig az AP négy további kulcsot generál: egy adatrejtjelező kulcsot, egy adatintegritás-védő kulcsot, egy kulcsrejtjelező kulcsot, és egy kulcsintegritás-védő kulcsot. Ezeket együttesen páronkénti ideiglenes kulcsnak (Pairwise Transient Key, PTK) ne- Az AES-CCMP protokoll az adatok rejtjelezéséhez és az adatok integritás-védelméhez ugyanazt a kulcsot használja, ezért AES-CCMP használata esetén csak három kulcs generálódik a PMK-ból. A PTK előállításához a PMK-n kívül felhasználják még a két fél (mobil eszköz és AP) MAC címét, és két véletlenszámot, melyet a felek generálnak. Ezt a *3. ábra* szemlélteti.

A véletlenszámokat az úgynevezett *négyutas kézfogás* (four way handshake) protokollt használva juttatják el egymáshoz a felek. Ennek a protokollnak további fontos feladata az, hogy segítségével a felek közvetlenül meggyőződjenek arról, hogy a másik fél ismeri a PMK-t. A négy utas kézfogás protokoll üzeneteit az EAPOL protokoll Key típusú üzeneteiben juttatják el egymáshoz a felek. Az üzenetek tartalma és a protokoll működése vázlatosan a következő:

1. Első lépésként az AP elküldi az általa generált véletlenszámot a mobil eszköznek. Mikor a mobil eszköz ezt megkapja, rendelkezésére áll minden információ a PTK előállításához. A mobil eszköz tehát kiszámolja az ideiglenes kulcsokat.

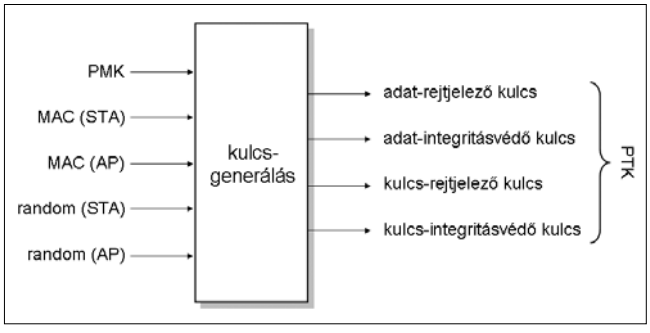
2. A mobil eszköz is elküldi az általa generált véletlenszámot az AP-nek. Ez az üzenet kriptográfiai integritás-ellenőrző összeggel (Message Integrity Code, MIC) van ellátva, amit a mobil eszköz a frissen kiszámolt kulcsintegritás-védő kulcs segítségével állít elő. Az üzenet vétele után az AP-nek is rendelkezésére áll minden információ a PTK kiszámításához. Kiszámolja az ideiglenes kulcsokat, majd a kulcsintegritás-védő kulcs segítségével ellenőrzi a MIC-et. Ha az ellenőrzés sikeres, akkor elhiszi, hogy a mobil eszköz ismeri a PMK-t.

3. Az AP is küld egy MIC-et tartalmazó üzenetet a mobil eszköznek, melyben tájékoztatja a mobil eszközt arról, hogy a kulcsokat sikeresen telepítette, és készen áll a további adatforgalom rejtjelezésre. Ez az üzenet tartalmaz továbbá egy kezdeti sorozatszámot. A későbbiekben ettől az értéktől kezdik majd sorszámozni a felek az egymásnak küldött adatcsomagokat, és a sorszámozás segítségével detektálják a visszajátszásos támadásokat. Az üzenet vétele után a mobil eszköz a kulcsintegritás-védő kulccsal ellenőrzi a MIC-et, és ha az ellenőrzés sikeres, akkor elhiszi, hogy az AP ismeri a PMK-t.

4. Végül a mobil eszköz nyugtázza az AP előző üzenetét, mely egyben azt is jelenti, hogy a mobil eszköz is készen áll a további adatforgalom rejtjelezésére.

A továbbiakban a mobil eszköz és az AP az adatintegritás-védő és az adatrejtjelező kulccsal védik egymásnak küldött üzeneteiket. Szükség van azonban még olyan kulcsokra is, melyek segítségével az AP többesszórással küldhet üzeneteket biztonságosan minden mobil eszköz számára.

3. ábra
A PTK generálása a PMK-ból, a felek MAC címéből, és a véletlenszámokból



Értelemszerűen, ezeket a kulcsokat az összes mobil eszköznek és az AP-nek is ismernie kell, ezért ezeket együttesen ideiglenes csoportkulcsnak (Group Transient Key, GTK) nevezik. A GTK egy rejtjelező és egy integritásvédő kulcsot tartalmaz. AES-CCMP esetén a két kulcs ugyanaz, ezért csak egy kulcsból áll a GTK. A GTK-t az AP generálja, és a négy utas kézfogás során létrehozott kulcsrejtjelező kulcsok segítségével titkosítva juttatja el minden mobil eszközhöz külön-külön.

3.3. TKIP és AES-CCMP

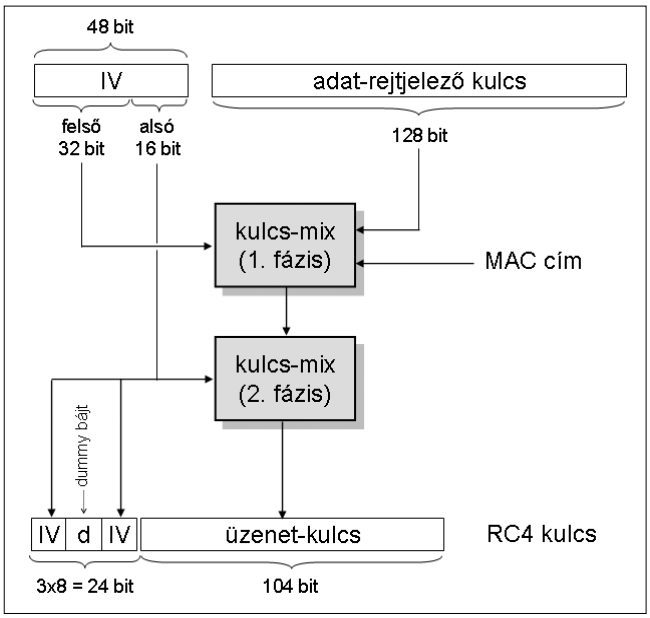
A TKIP (Temporal Key Integrity Protocol) és az AES-CCMP (AES CTR Mode and CBC MAC) a fent leírt kulcsmenedzsment megoldásra támaszkodó protokollok, melyek az üzenetek integritás-védelmével és rejtjelezésével foglalkoznak. Mint azt korábban említettük, a TKIP egy olyan köztes megoldás, amely a régi, WEP-es hardveren is működik, de a WEP-nél jóval magasabb szintű biztonságot nyújt. Az AES-CCMP új hardvert igényel, de cserébe egyszerűbb, tisztább megoldást nyújt, mint a TKIP. A TKIP a WEP hibáit a következő módokon igyekszik javítani:

Integritás-védelem: A TKIP egy új integritás-védelmi mechanizmussal egészíti ki a WEP-es megoldást (utóbbi általában hardverben van implementálva, úgyhogy benne hagyták a TKIP-ben is). Az új mechanizmust Michael-nek hívják. A Michael SDU szinten működik (azaz a felsőbb protokollszintről a MAC szintre érkező adatokon, fragmentálás előtt végzi az integritásvédő ellenőrző összeg számítását), ami lehetővé teszi a hálózati kártya meghajtó programjában (device driver) történő megvalósítást. Ez azért fontos, mert így a Michael bevezetése egyszerű upgrade-del megoldható.

Az üzenet-visszajátszás detektálása érdekében a TKIP az IV-t használja sorozatszámként. Ennek megfelelően, a TKIP előírja, hogy az IV értékét minden üzenetben eggyel növelni kell (a WEP-ben ez nem volt kötelező). A vevő nyilvántartja az utolsó néhány vett IV értéket. Ha egy frissen érkezett üzenet IV-je kisebb, mint a legkisebb nyilvántartott IV, akkor a vevő eldobja az üzenetet, míg ha az üzenet IV-je hagyobb, mint a legnagyobb nyilvántartott IV, akkor a vevő megtartja az üzenetet és módosítja a nyilvántartását. Ha egy vett üzenet IV-je a legkisebb és a legnagyobb nyilvántartott IV közé esik, akkor a vevő ellenőrzi, hogy az adott IV szerepel-e a nyilvántartásában. Ha igen, akkor eldobja az üzenetet, ha nem, akkor megtartja azt és módosítja a nyilvántartását.

Titkosítás: Emlékeztetünk arra, hogy a WEP titkosítás legfőbb hibáját az IV kis mérete és a gyenge RC4 kulcsok használata jelentette. A TKIP-ben ezért az IV méretét 24 bitről megnövelték 48 bitre. Ez egyszerű megoldásnak látszik, ám a nehézséget az okozza, hogy a WEP-et támogató hardverek adott hosszúságú (128 bites) értékkel inicializálják az RC4 algoritmust, s így a megnövelt IV-t, a rejtjelező kulccsal együtt, valamilyen módon „bele kell gyömöszölni” ebbe az adott hosszúságba. A gyenge kulcsok problémáját a TKIP úgy oldja meg, hogy minden üzenet rejtjelezését más

kulccsal végzi. Így a támadó nem tud a sikeres támadáshoz szükséges számú, azonos (potenciálisan gyenge) kulccsal kódolt üzenetet megfigyelni. Az üzenetkulcsokat a TKIP a négyutas kézfogás során generált adatrejtjelező kulcsból állítja elő. A TKIP IV mechanizmusát és az üzenet-kulcsok előállítását a *4. ábra* szemlélteti.



4. ábra Az RC4 kulcs előállítása a TKIP-ben

Az AES-CCMP tervezőinek bizonyos értelemben könnyebb dolguk volt, mint a TKIP tervezőinek, hiszen nem volt megkötés arra vonatkozóan, hogy a protokollnak milyen hardveren kell futnia. A tervezők ezért egyszerűen megszabadultak az RC4 algoritmustól, s helyette az AES blokkrejtjelezőre építették fel a protokollt. Definiáltak egy új AES használati módot, mely a régóta ismert CTR (Counter) mód és a CBC-MAC (Cipher Block Chaining – Message Authentication Code) kombinációja. Ebből származik a CCMP rövidítés. CCMP módban az üzenet küldője először kiszámolja az üzenet CBC-MAC értékét, ezt az üzenethez csatolja, majd az üzenetet CTR módban rejtjelezi. A CBC-MAC számítás kiterjed az üzenet fejlécére is, a rejtjelezés azonban csak az üzenet hasznos tartalmára és a CBC-MAC értékre vonatkozik. A CCMP mód tehát egyszerre biztosítja a teljes üzenet (beleértve a fejlécet is) integritásának védelmét és az üzenet tartalmának titkosságát. A visszajátszás ellen az üzenetek sorszámozásával védekezik a protokoll. A sorszám a CBC-MAC számításhoz szükséges inicializáló blokkban van elhelyezve.

4. Összegzés

Cikkünkben ismeretterjesztő jellegű áttekintést adtunk a WiFi biztonsághoz kapcsolódó szabványokról, a WEP-ről és a 802.11i-ről. Bemutattuk a WEP működését és hibáit. Ismertettük a 802.11i-ben definiált hitelesítési és hozzáférés-védelmi mechanizmust, kulcshierarchiát, valamint a TKIP és az AES-CCMP protokollokat. Láttuk, hogy a TKIP protokoll célja, hogy lehetővé tegye a 802.11i-ben definiált új biztonsági architektúra alkalmazását a régi eszközökön, melyek hardvere csak a WEP-et támogatja. Ez a követelmény nagy mértékben megkötötte a tervezők kezét, ezért működő, de nem túl szép megoldás született. Ezzel szemben az AES-CCMP protokoll biztonságos és elegáns, viszont új hardvert igényel.

Manapság a boltban vásárolt új WiFi eszközök már támogatják 802.11i-ben definiált biztonsági architektúrát. Új eszközök vásárlása esetén tehát érdemes az RSN-t használni, amit a gyártók WPA2-nek is neveznek. Ha csak régebbi eszközök állnak rendelkezésre, akkor a WPA használatát tanácsoljuk.

Köszönetnyílvánítás

Jelen anyag elkészítését a Mobil Innovációs Központ (www.mik.bme.hu) és a 2 027 04 számú NKFP projekt (Adaptív médiafolyam szolgáltatási architektúra a legújabb mobil távközlési rendszerek céljaira) támogatta.

Irodalom

[1] J. Edney, W. Arbaugh, Real 802.11 Security – WiFi Protected Access and 802.11i. Addison-Wesley, 2004.
[2] IEEE Std 802.11. IEEE Standard: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, 1999.
[3] J. Walker, Unsafe at any key size: An analysis of the WEP encapsulation. IEEE 802.11-00/362, 2000.
[4] N. Borisov, I. Goldberg, D. Wagner, Intercepting mobile communications: the insecurity of 802.11. Proc. of the 7th ACM Conference on Mobile Computing and Networking, 2001.
[5] W. Arbaugh, N. Shankar, J. Wan, K. Zhang, Your 802.11 network has no clothes. IEEE Wireless Communications Magazine, 9(6):44–51, 2002.
[6] IEEE Std 802.11i. IEEE Standard Amendment 6: Medium Access Control (MAC) Security Enhancements, 2004.
[7] S. Fluhrer, I. Mantin, A. Shamir, Weaknesses in the key scheduling algorithm of RC4. Proc. of the 8th Workshop on Selected Areas in Cryptography, 2001.
[8] Wi-Fi Alliance. Wi-Fi Protected Access. http://www.wi-fi.org/white_papers/whitepaper-042903-wpa/ (ellenőrizve 2006. ápr.19-én)
[9] IEEE Std 802.1X-2001. IEEE Standard: Port-based Network Access Control, 2001.
[10] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, H. Levkowetz: Extensible Authentication Protocol, RFC 3748., 2004.
[11] B. Aboba, P. Calhoun, RADIUS (Remote Authentication Dial In User Service) Support for Extensible Authentication Protocol (EAP), RFC 3579, 2003.

Biztonságos szolgáltatások kihelyezése mobil eszközök számára

SZENTGYÖRGYI ATTILA, SZÜTS PÉTER LÓRÁNT

Budapesti Műszaki és Gazdaságtudományi Egyetem, Távközlési és Médiainformatikai Tanszék
{szgyi, szuts}@alpha.tmit.bme.hu

Lektorált

Kulcsszavak: WLAN, mobil eszközök, PDA, Security Proxy, titkosítás, adatbiztonság

Napjainkban egyre többen használnak kézi számítógépet (Personal Digital Assistant, PDA) és okostelefont (Smartphone). A hordozhatóságból származó előnyt azonban beárnyékolja az eszközök számára rendelkezésre álló limitált erőforrások mennyisége: a processzor teljesítménye és a memóriaterület mérete. Az erőforráshiány következtében a biztonságos hálózati szolgáltatások igénybevételénél adatátviteli sebesség csökkenést tapasztalhatunk. Emellett számos olyan biztonságos kommunikációt használó szolgáltatással is találkozhatunk, amelyek a mobil eszközök számára mindeddig nem álltak rendelkezésre. A problémák megoldásához egy olyan eljárást kerestünk, amely a biztonságos kommunikáció megtartása mellett nem csökkenti az adatátvitel sebességét, és lehetővé teszi új, a mobil eszközök számára eddig elérhetetlen szolgáltatások igénybevételét. Vizsgálódásaink eredményeként megszületett egy erre a célra alkalmas megoldás; a Security Proxy (SP).

1. A Security Proxy működése

A Security Proxy [9] a mobil eszközök megbízásából (Proxy) biztonságos kapcsolatot épít ki az Internet egy távoli szerverével, és így a készülék felől érkező titkosítatlan adatokat titkosítva továbbítja (Security).

A Security Proxy az adatok titkosítását két szinten végezheti. Egyrészt létrehozhat a távoli szerverhez egy titkosított csatornát (tunnel), amelyen a mobil eszköz csomagjait titkosítva, ám változtatás nélkül küldheti tovább. Ezt a működést hívjuk *tunnel* üzemmódnak. A tunnel mód egy tipikus alkalmazása a virtuális magánhálózatokhoz [2] (Virtual Private Network, VPN) történő csatlakozás.

Amikor tunnel üzemmódban egy mobil eszköz a Security Proxyt igénybe véve csatlakozni kíván egy virtuális magánhálózathoz, akkor ezt jeleznie kell a Security Proxynak, ami ezek után VPN kliensként működve kapcsolódni fog a távoli VPN szerverhez. Miután létrejött a kapcsolat, és erről értesítette a mobil eszközt, a készülék megkezdheti a kommunikációt a távoli számítógépen keresztül elérhető hálózattal. Ezt a folyamatot mutatja be az *1. ábra*.

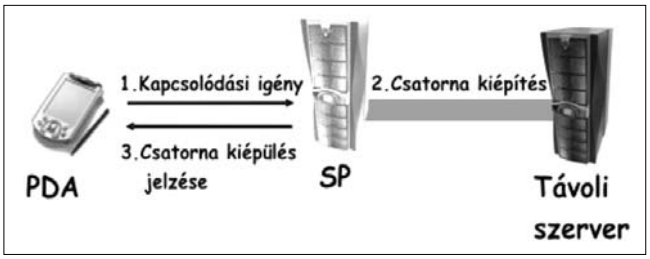
A Security Proxy alkalmazási szinten is működhet, ezt az üzemmódot hívjuk *proxy* módnak. Proxy mód esetében, amikor egy mobil eszköz csatlakozni szeretne egy titkosítást igénylő távoli alkalmazáshoz, akkor egy jóval gyengébb, vagy titkosítást egyáltalán nem használó programmal (pl. Telnet [5]) a Security Proxyhoz fordul, ami egy biztonságosabb szoftver (például SSH [4]) segítségével továbbítja az adatokat a távoli számítógéphez. Ezt a folyamatot szemlélteti a *2. ábra* a Telnet-SSH átalakítás esetében.

A két működési mód közötti alapvető különbség, hogy a tunnel üzemmód esetén a mobil eszköz az alkalmazásszintű protokolltól függetlenül a teljes hálózati forgalmát átküldheti a titkosított csatornán, míg proxy módban minden alkalmazásszintű protokoll támogatását egyenként kell megvalósítani.

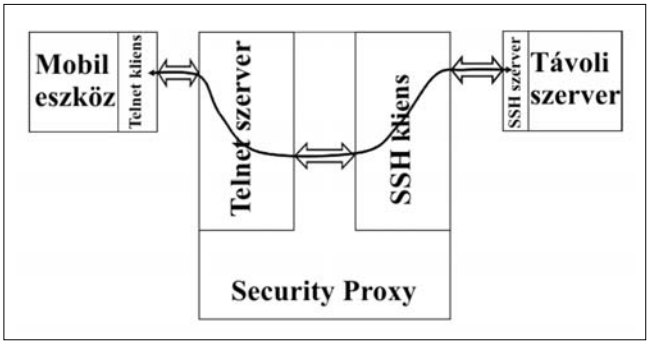
A Security Proxy transzparens módon látja el a feladatát. A mobil eszközön futó alkalmazás nem tudja, hogy a csomagjai a Security Proxyn haladnak keresztül, és a távoli szerver sincs tisztában azzal, hogy a kapott csomagok közvetlen forrása nem a mobil eszköz. Ez azért előnyös, mert a Security Proxy szolgáltatásainak igénybevételéhez nem szükséges módosítani sem a távoli szerveren futó alkalmazásokat, sem a mobil eszközön meglévő szoftvereket.

Biztonságos hálózati alkalmazásoknál előfordulhat, hogy egy mobil eszköz a Security Proxyn keresztül egy távoli hálózatot vagy alkalmazást szeretne elérni, de a

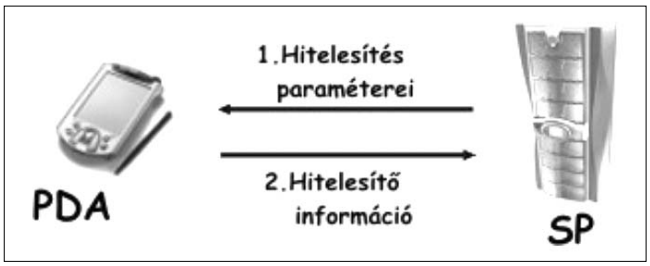
1. ábra Security Proxy tunnel módja VPN kapcsolattal



2. ábra A proxy mód



Security Proxy nem rendelkezik a hitelesítéshez szükséges titokkal – ami lehet egy jelszó, vagy egy privát kulcs. Ebben az esetben a titkosított csatornát a Security Proxy nem tudja kiépíteni a jogosultság hiánya miatt. A jogosultság megszerzéséhez hozzá kell jutnia a készülék által őrzött információhoz.



3. ábra A kihelyezett hitelesítési eljárás

Mivel a felhasználótól a titkának kiadását nem várhatjuk el, de a Security Proxy számára biztosítani kell a titokhoz való közvetett hozzáférést, ezért a Security Proxynak egy üzenetet kell küldenie a mobil eszköznek, amely tartalmazza azokat a paramétereket, amelyek a titok hozzáadásával hitelesítik a felhasználót a távoli hálózatban. Miután a készülék megkapta a hitelesítéshez szükséges adatokat, a titok hozzáadásával előállítja a hitelesítő információt, majd ezt visszaküldi a Security Proxynak *(3. ábra)*. A Security Proxy ezt felhasználva a mobil eszköz nevében már ki tudja építeni a távoli szerverrel a biztonságos kapcsolatot.

Profilokat használhatunk annak érdekében, hogy a mobil eszköz beállításait ne kelljen minden egyes alkalommal megadni, és a szolgáltatásokat több Security Proxynál is egyszerűen igénybe lehessen venni. A profilokat egy böngészőn keresztül lehet módosítani, amit azután le kell tölteni a mobil eszközre. A szolgáltatások igényléséhez a profilt tartalmazó fájlt csak fel kell tölteni a Security Proxyra, ami ezt követően kiépíti a fájlban megadott szolgáltatásokat.

3. Security Proxy illesztése meglévő hálózatokhoz

A Security Proxyt meglévő hálózatok kiegészítő szolgáltatásaként terveztük, ezért a hozzá tartozó adatforgalmat el kell különíteni a hálózat többi forgalmától. A hálózatban így szükség van a mobil eszköz és a Security Proxy között egy átjáróra (*Gateway,* GW), amely képes a készülék bizonyos csomagjait a Security Proxy felé irányítani.

Vezetéknélküli hálózatok esetében az átjáró lehet például a mobil eszköz vezetéknélküli hozzáférési pontja is. Az átjárónak így információt kell cserélnie a Security Proxyval, és ez alapján kell módosítania a készülék csomagjainak útvonalát. Ezt szemlélteti a *4. ábra*.

Amikor egy mobil eszköz tunnel módot szeretne használni, akkor ezt jelzi a Security Proxynak. A Security Proxy ekkor üzenetet küld az átjárónak, hogy azokat a csomagokat, amelyeknek forráscíme a mobil eszköz IP címe, a célcíme pedig a Security Proxytól induló csatorna végén lévő alhálózat valamely IP címe, a Security Proxy felé irányítsa. Ebben az esetben tehát az útvonalválasztó a csomagok forrás- és célcíme alapján választja ki a helyes irányt. Proxy mód használata esetén a mobil eszköztől származó csomagok célcíme csak a távoli kiszolgáló IP címe lehet, így nincs szükség egy teljes címtartomány figyelésére. Mindemellett figyelni kell a mobil eszköz IP címét is, vagyis a csomagok forráscímét.

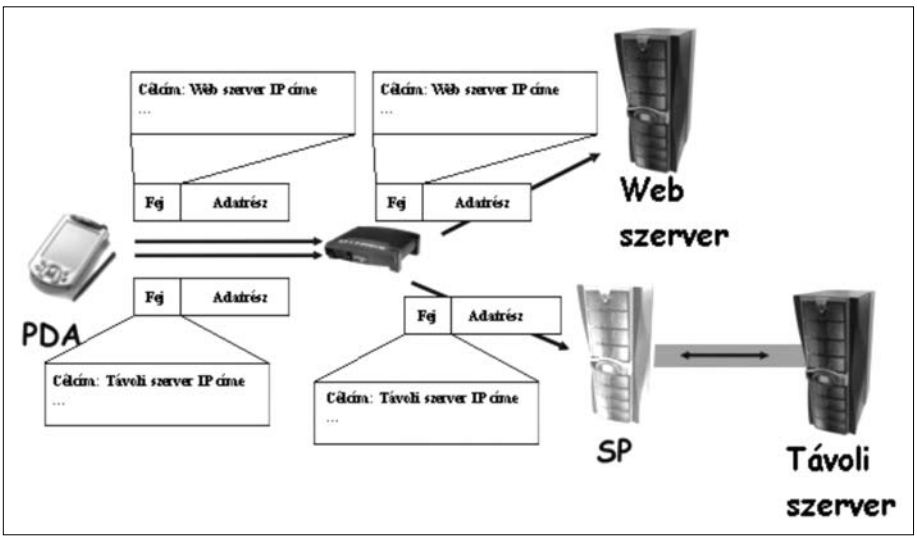
4. A Security Proxy Control Protocol

Az SPCP egy általunk készített protokoll, melynek célja a Security Proxy és a hálózati elemek közötti kommunikáció biztosítása. Ez a kommunikáció magában foglalja a kapcsolatfelépítést és -bontást a mobil eszköz és a Security Proxy között, a kihelyezett hitelesítési eljárás használatát, illetve a Security Proxy és az átjáró kommunikációját. Ez a kommunikáció kérdés-válasz alapú.

Mobil eszköz és Security Proxy kommunikáció

A mobil eszköz a kapcsolatot egy kapcsolatfelépítés (CONNECT) üzenet elküldésével veheti fel a Security Proxyval. Ha a kapcsolat felépült és a készülék egy új szolgáltatást kíván igénybe venni (set), vagy egy korábbi megrendelését szeretné törölni (del), akkor egy módosító (MODIFY) üzenetben jelezheti mindezt. Amikor a mobil eszköz egy ilyen üzenetet küld, akkor a Security Proxy kiépíti a távoli szerverrel a titkos csatornát, majd jelzi a készülék átjárójának, hogy hozza létre a megfelelő bejegyzéseket a routing táblájában, a sikeres létrehozás visszaigazolása után pedig a Security Proxy nyugtázza a mobil eszköz felé a szolgáltatás kiépítését. Kihelyezett hitelesítés használata során a mó-

4. ábra Security Proxy által vezérelt átjáró



dosító (MODIFY) üzenet után a Security Proxy egy státusz (STATUS) üzenetben elküldi a hitelesítéshez szükséges paramétereket. A készülék ezt feldolgozza, majd megismétli az előző módosító (MODIFY) üzenetét kiegészítve a hitelesítő információkkal.

A feltöltés (UPLOAD) üzenettel a mobil eszköz adatokat tölthet fel a Security Proxyra. Ilyen adat lehet a szolgáltatások paramétereit tartalmazó profil, illetve a hitelesítéshez szükséges tanúsítvány [6].

A kapcsolat lebontását a kapcsolatbontás (DISCONNECT) üzenettel lehet kezdeményezni. A kapcsolatbontás (DISCONNECT) üzenet hatására a lefoglalt erőforrások azonnal felszabadíthatók, és az átjáró konfigurációja visszaállítható anélkül, hogy meg kellene várni, amíg a készülék és a Security Proxy közötti kapcsolat időtúllépés miatt megszűnik.

A parancsok feldolgozásának sikerességét, illetve az esetleg fellépő hibákat a státusz (STATUS) üzenet segítségével jelezzük.

Átjáró és Security Proxy kommunikáció

A Security Proxy és az átjáró között routing (ROUTE) üzenetek haladnak. Ezekkel az üzenetekkel lehetővé válik a csomagok szeparációja attól függően, hogy a készülék éppen a Security Proxy valamely szolgáltatása számára küldi a csomagot, vagy a csomag független a Security Proxytól. Ha a Security Proxy egy routing (ROUTE) üzenetet küld az átjárónak, akkor az egy válaszüzenetben jelzi, hogy sikerült-e megfelelően kitölteni (set) vagy törölni (del) a routing-táblájának a szolgáltatáshoz tartozó bejegyzéseit.

Az *5. ábra* egy egyszerű kapcsolatfelépítési eljárást mutat proxy üzemmód esetén.

5. A Security Proxy és a biztonság

A Security Proxy Control Protocol alapesetben nem nyújt biztonságot a támadásokkal és az üzenetmódosításokkal szemben, így az adatforgalom biztonságát és hitelességét hálózati szinten kell biztosítani.

A mobil eszköztől érkező csomagok a készülék és a hozzáférési pont között egy WPA TKIP [8] által titkosított csatornán mennek át, a hozzáférési pont és a Security Proxy között pedig egy kiépített biztonságos alagúton (pl. IPSec [2]) haladnak *(6. ábra)*.

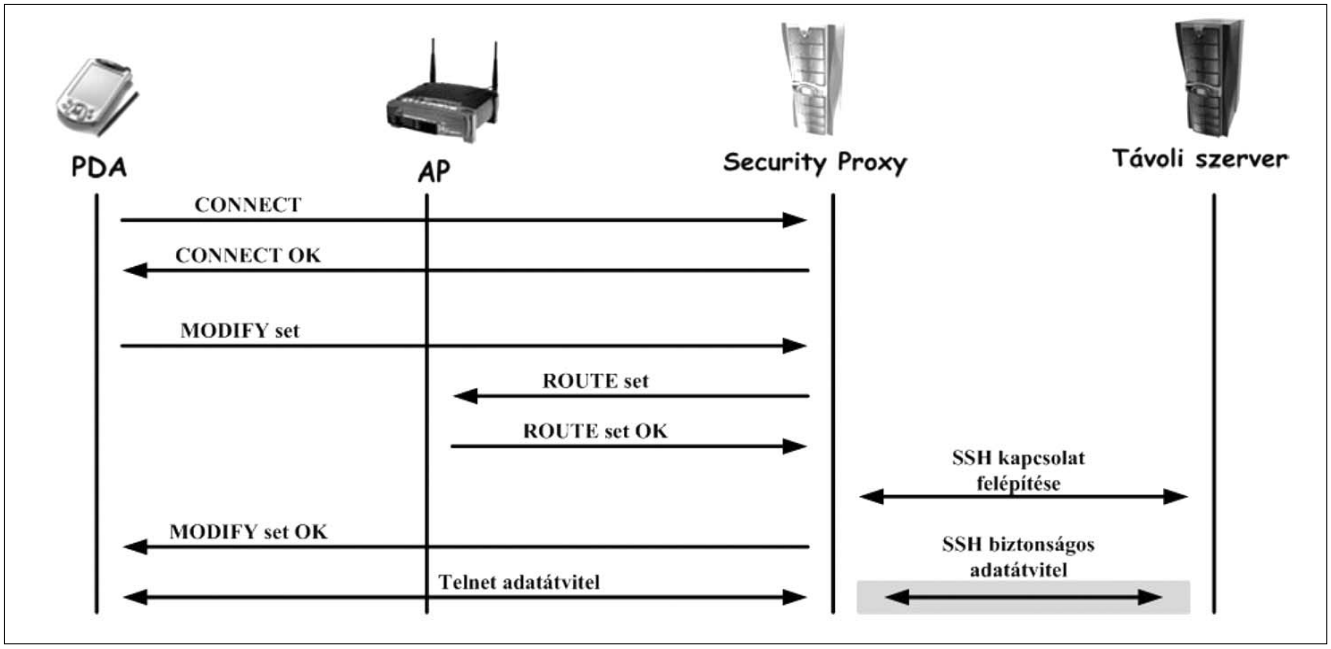
A mobil eszköz a hozzáférési pontot könnyen hitelesítheti például egy tanúsítvány segítségével. Mivel azonban a WPA TKIP titkosítás megszűnik a hozzáférési pontnál, ezért az hozzáférhet a mobil eszköz csomagjaihoz: láthatja és értelmezheti az IP fejlécet, de akár módosíthatja is a csomag tartalmát. Ez egyrészt előnyös, hiszen képessé válik a forgalom irányának befolyásolására, másrészt viszont megszemélyesítéses támadásra *(man-in-the-middle/evil twin attack)* adhat okot.

Ez a támadás azonban nem hajtható végre a hálózatban, ugyanis a hitelesítéshez szükséges közös titkot csak a hálózat hitelesítő szervere (pl. RADIUS [7]) és a mobil eszköz ismeri, és a titkot egyikük sem adja ki a támadónak, így tehát a hálózat biztonságos lesz.

6. Teszthálózat és elért eredmények

A Security Proxyt az Infopark teszthálózatába [1] illesztettük be, amelyben található egy vezetéknélküli hozzáférési pont, egy RADIUS hitelesítő szerver, egy VPN szerver és az Internet és a belső hálózat között elhelyezkedő tűzfal. A Security Proxy közvetlenül a tűzfalhoz kapcsolódik, és a tűzfalon keresztül kommunikál a mobil eszköz átjárójaként szolgáló vezetéknélküli hozzáférési ponttal. A tűzfalat úgy állítottuk be, hogy a Security Proxy által az Internet felé kezdeményezett kapcsolatokat engedélyezze.

5. ábra
Kapcsolatfelépítés proxy üzemmódban (Telnet / SSH)



6. ábra
Titkosítás a mobil eszköz és a Security Proxy között

A méréseket ebben a hálózatban végeztük el. A mérések során két főbb esetet vizsgáltunk. Először a mobil eszköz egy IPSec szolgáltatást vett igénybe. Ekkor megmértük az adatátviteli sebességet Security Proxy használatával és anélkül. Az eredmény a *7/a. ábrán* látható.

Ha a hálózatban nem használtunk titkosítást, akkor az adatátviteli sebesség jóval nagyobb, mint amikor a kézi számítógép IPSec titkosítást használt. Amikor azonban a titkosító funkciót áthelyeztük a Security Proxyhoz, akkor láthatóan nem történt adatátviteli sebesség csökkenés.

A másik esetben egy olyan biztonsági szolgáltatást tettünk elérhetővé a kézi számítógép számára, amelyet eddig nem vehetett igénybe. A *7/b. ábrán* látható, hogy ha a Security Proxy végzi az openVPN [3] biztonságos csatornához szükséges titkosítást, akkor – az IPSec esetéhez hasonlóan – nem tapasztalható adatátviteli sebességcsökkenés.

Megállapítható tehát, hogy mindkét esetben, amikor a mobil eszköz igénybe vette a Security Proxy szolgáltatásait, adatsebesség csökkenés nélkül tudta használni a biztonságos szolgáltatásokat.

7. Összefoglalás

Cikkünkben bemutattuk a Security Proxy működésének elméletét és a hozzá kapcsolódó kihelyezett hitelesítési eljárást. Ezzel kapcsolatban találkoztunk a Security Proxy és a hálózati elemek információcseréjéhez létfontosságú SPCP-vel, és bemutattuk a Security Proxy vezetéknélküli hálózatokba történő biztonságos integrációját és az így született mérési eredményeket. Látható tehát, hogy a Security Proxy egy olyan komplex, bővíthető megoldás, mely bármely vezetéknélküli hálózatban lehetővé teszi a kisteljesítményű mobil eszközök számára a biztonságos szolgáltatások igénybevételét.

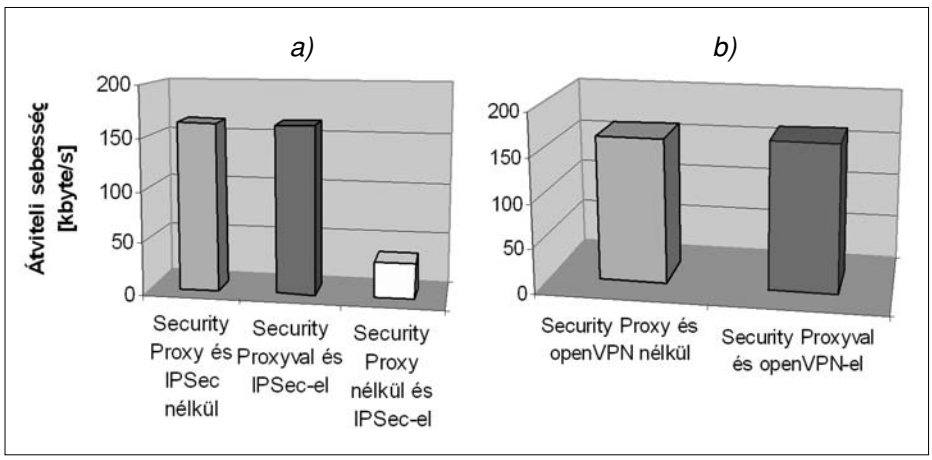
Köszönetnyilvánítás

Köszönetet mondunk a Budapesti Műszaki és Gazdaságtudományi Egyetem Távközlési és Média-Informatikai Tanszékének, hogy támogatta TDK dolgozatunk elméletének és prototípusának megvalósítását, valamint a cikk létrejöttét, továbbá külön szeretnénk köszönetet mondani a rengeteg segítségért konzulenseinknek, Dr. Fehér Gábornak és Korn Andrásnak.

Irodalom

[1] DR. FEHÉR GÁBOR (SZERK): Biztonságos vezetéknélküli szuperhálózat elosztott környezetben, 2004. http://qosip.tmit.bme.hu/cgi-bin/twiki/view/SECLab/ResultS/Rendszerterv_final.pdf
[2] NAGANAND DORASWAMY, DAN HARKINS: IPSec: The new Security Standard for the Internet, Intranets, and Virtual Private Networks, Second Edition, Prentice Hall PTR, 2003.
[3] CHARLIE HOSNER: OpenVPN and the SSL VPN Revolution 2004, http://www.sans.org/rr/whitepapers/vpns/1459.php
[4] T. YLONEN, C. LONVICK: SSH Protocol Architecture, Internet-Draft, 2005.
[5] J. POSTEL AND J. REYNOLDS: Telnet Protocol Specification, RFC 854, 1983.
[6] R. HOUSLEY, W. FORD, W. POLK, D. SOLO: Internet X.509 Public Key Infrastructure, Certificate and CRL Profile, RFC 2459, 1999.
[7] JONATHAN HASSEL: „RADIUS", O'Reilly, 2002.
[8] JON EDNEY, WILLIAM A. ARBAUGH: „Real 802.11 Security: Wi-Fi Protected Access and 802.11i", Addison-Wesley, 2003.
[9] Security Proxy weboldala: http://alpha.tmit.bme.hu/~szgyi/sp.html

7. ábra
a) Mérések IPSec-el, b) Mérések openVPN-el



Mi alapján fogadhatunk el egy elektronikus aláírást?

BERTA ISTVÁN ZSOLT

Microsec Kft.
istvan.berta@microsec.hu

Lektorált

Kulcsszavak: PKI, digitális aláírás, tanúsítvány, hitelesítés szolgáltatók, időpecsét, KGYHSZ

Az elektronikus aláírás elméleti alapjai régóta ismertek. E matematikai, kriptográfiai szempontból bevált technológia már a hazai jogrendszerbe is beépült, és ezzel az elektronikus aláírás a kézzel írott aláírással egyenértékűvé vált. Az elektronikus aláírás használatához mind a matematikai, mind a jogi alapok rendelkezésre állnak, de e technológia mégis csak az elmúlt években kezdett elterjedni hazánkban. A gyakorlati alkalmazás során számos probléma merült fel ezen új technológiával kapcsolatban. Ezek nagy része megjelenik a kézzel írott aláírások esetében is, de e problémák ott sokkal kisebb jelentőséggel bírnak.

1. Bevezetés

Elektronikus aláírás segítségével elektronikus dokumentumokat hitelesíthetünk. Ha egy dokumentumot elektronikusan írunk alá, az egyenértékű azzal, hogy a kézzel írott aláírásunkkal látjuk el. Az elektronikusan aláírt dokumentumról bármennyi másolatot készíthetünk, és tetszőleges számú helyre elküldhetjük úgy, hogy minden példány hiteles, az „eredetivel" egyenértékű, azonos bizonyító erejű lesz. Az elektronikus dokumentumokat papír alapú társaiknál sokkal gyorsabban elküldhetjük, és a fogadó fél – akár automatizáltan, számítógépes programok segítségével – sokkal gyorsabban feldolgozhatja őket. Ma sok ügyet azért lehet kizárólag papíron intézni, mert eredeti, hiteles dokumentumokra van szükség. Az elektronikus dokumentumok a papíroknál könnyebben kezelhetőek, a rendszer jobban kézben tartható, és az elektronikus aláírással ellátott iratok a papírokkal azonos bizonyító erővel rendelkeznek. Egy jól megtervezett informatikai rendszerben, amely értelmesen használja az elektronikus aláírásra épülő technológiát, gyorsan és gördülékenyen intézhetjük ügyeinket. Ezzel szemben, ha az elektronikus aláírást ügyetlenül alkalmazzuk, drága, szövevényes és bonyolult rendszerhez jutunk, amelyben az értelmetlenül használt elektronikus aláírások csak nyűgöt és költséget jelentenek, szélsőséges esetben akár bizonyító erejüket is elveszíthetik.

Az elektronikus aláíráshoz szükséges technológiák és kriptográfiai algoritmusok több évtizede rendelkezésre állnak [1]. A 2001-ben megjelent, elektronikus aláírásról szóló törvény, az [2] a kriptográfiai algoritmusok nyújtotta *letagadhatatlanságot* jogi fogalomnak, *bizonyító erőnek* feleltette meg. E törvény *„minősített"* és *„fokozott biztonságú"* elektronikus aláírást különböztet meg. (Emellett „egyszerű" elektronikus aláírást is említ, de ezzel itt nem foglalkozunk.) A fokozott biztonságú aláírással hitelesített dokumentum *magánokirat* (tehát a kézzel írott aláírással azonos bizonyító erővel rendelkezik), míg a fokozott biztonságú aláírásnál erősebb minősített aláírással hitelesített dokumentum *teljes bizonyító erejű magánokirat* (ilyen például a két tanú előtt vagy közjegyző előtt aláírt dokumentum is). Minősített aláírással kapcsolatos jogvita esetén a bíróságnak vélelmeznie kell, hogy az aláírt dokumentum tartalma az aláírás elkészítésének időpontja óta nem változott meg, viszont a fokozott biztonságú aláírásra nem vonatkozik ilyen szabály [2]. Mind a technológia, mind a hozzá kapcsolódó jogszabályok rendelkezésre állnak, és a technológiát a jogszabályok szerint szolgáltató cégek is megjelentek. Négy olyan minősített *hitelesítés szolgáltató* is működik Magyarországon, akiktől bárki vehet minősített elektronikus aláíráshoz használható tanúsítványt.

Az elektronikus aláírás gyakorlatban történő alkalmazása számos olyan problémát vet fel, amelyet sem a technológia, sem a jogszabályok alapján nem könnyű megválaszolni. Cikkünkben az egyik ilyen problémakört, nevezetesen egy elektronikus aláírás elfogadásakor szükséges lépéseket gondoljuk végig. Emellett azt is megvizsgáljuk, hogy a közigazgatás által a közelmúltban közzétett elektronikus aláírás keretrendszer [3] hogyan viszonyul e kérdésekhez.

2. Milyen formátumú az elektronikus aláírás?

Tételezzük fel, hogy kaptunk egy elektronikusan aláírt dokumentumot. Első kérdés, hogy egyáltalán tudjuk-e értelmezni az elektronikus aláírást. Más szóval, van-e olyan szoftverünk, amely érti az adott formátumú aláírást. Nagyon sok különböző elektronikus aláírás formátum létezik: nemcsak e-mailek, de például Word dokumentumok és PDF fájlok is tartalmazhatnak aláírást, és a különböző szoftverek (illetve ugyanazon szoftver különböző verziói) által létrehozott fájlok különböző formátumú elektronikus aláírást tartalmazhatnak. Ezek közül a levelezőprogramok által készített S/MIME aláírások tekinthetőek valamelyest szabványosnak; ekkor

gyakori, hogy az egyik levelezőprogrammal készített aláírást egy másik levelezőprogram is megérti. Sajnos, az általános célú alkalmazások – beleértve a levelezőprogramokat is – lényeges, alapvető műszaki követelményeknek sem felelnek meg a hosszú távon is letagadhatatlan elektronikus aláírással kapcsolatban. A legtöbb alkalmazás nem építi fel a tanúsítványláncot, sok nem ellenőrzi a lánc egyes elemeinek visszavonási állapotát, valamint nem vizsgálja meg, hogy a visszavonási állapotról beszerzett információ az aláírás időpontjára vonatkozik-e egyáltalán. (E kérdésekről a későbbi fejezetekben szólunk.) Ezen aláírás formátumok nem tartalmaznak időbélyeget sem, pedig ha nem tudjuk biztonságos módon megállapítani, hogy az aláírás mikor készült, akkor az aláírás letagadhatatlansága műszakilag nem biztosítható (lásd 4. fejezet). Léteznek olyan aláírás formátumok, amelyek alkalmasak az aláírás hosszú távú érvényességének biztosítására azáltal, hogy bennük az időbélyegek, a tanúsítványlánc és a tanúsítványra vonatkozó visszavonási információk is eltárolhatók (lásd 6. fejezet).

Aki elektronikusan aláírt dokumentumot szeretne fogadni, annak célszerű meghatároznia és a küldő fél tudomására hoznia, hogy milyen formátumú elektronikus aláírást fogad el. Különben abba a helyzetbe kerülhet, hogy olyan levelet kap, amelyet nem tud elolvasni. Az aláírás és az aláírt dokumentum többféleképpen csatlakozhatnak egymáshoz:

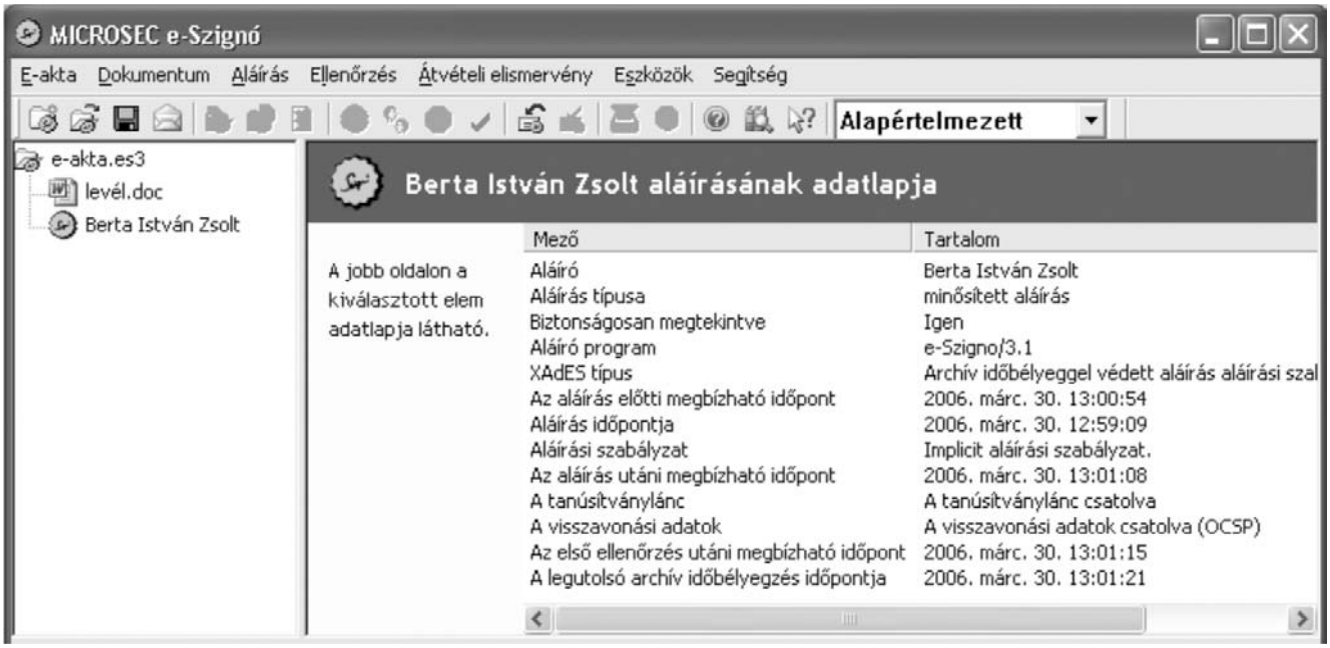
• Egyik lehetőség, hogy *az aláírás az aláírt dokumentum fájljába kerül*. Ilyen a .doc vagy .pdf fájlban lévő aláírás. E megoldásnak hátránya, hogy ilyenkor a felhasználó kénytelen az adott szoftver aláírás-ellenőrző mechanizmusát használni. E megoldás biztonsági szempontból nem egységes, mert lehet, hogy a befogadó az egyes fájltípusok esetén ellenőrzéskor más és más biztonsági követelményeket alkalmaz (a tanúsítványlánc felépítésére, a visszavonási információk összegyűjtésére stb). A befogadó kénytelen az adott típusú fájlt kezelő alkalmazás által nyújtott (gyakran igen alacsony) szintű biztonsággal beérni. Ugyanakkor, e megoldás sokszor praktikus lehet, mert egy meglévő informatikai rendszert nem szükséges jelentősen átalakítani az elektronikus aláírás bevezetéséhez; az eddig használt fájlformátumok (.doc, .pdf) továbbra is használhatóak.

• Az *aláírás(ok)* és az *aláírt dokumentum(ok)* egy aláírás-fájlban, például úgynevezett *e-aktában* helyezkednek el *(1. ábra)*. Ilyenkor a felhasználó az aláírásfájlt kezelő, professzionális aláírás-létrehozó (vagy -ellenőrző) alkalmazás biztonsági beállításai szerint hozhatja létre és ellenőrizheti az aláírásokat, így e megoldás biztonsági szempontból testre szabható, és megfelelő aláírás-létrehozó és -ellenőrző alkalmazás esetén erős biztonságot jelenthet. Egyes esetekben e megoldás kényelmetlen is lehet, mert az elektronikus aláírást bevezető szervezetnek át kell alakítani a belső folyamatait, hogy minden folyamat aláírás-fájlokat használjon. Ekkor az aláírt fájlok olvasásához is aláírás-ellenőrző szoftverre van szükség.

• Az *aláírás és az aláírt dokumentum(ok) külön fájlokban vannak*. E megoldás egyesíti a két előző előnyeit: professzionális, testre szabható alkalmazást használhatunk, így tetszőleges fájlt aláírhatunk, és egyúttal a hagyományos fájlformátumokra épülő meglévő folyamatokat nem kell átalakítani. E megoldásnak van egy veszélye is: ha az aláírt fájlokat és az aláírásokat szétválasztjuk egymástól, és egy részük megsérül, vagy összekeveredik, akkor nagyon-nagyon nehéz lehet biztosítani az aláírások letagadhatatlanságát.

A magyar közigazgatás a [4] szabvány által leírt XML elektronikus aláírás formátumot választotta. Ez egy nagyon jó választás – egy professzionális aláírás-formátum, amely alkalmas az aláírások letagadhatatlanságá-

1. ábra Elektronikus aláírás ellenőrzése az e-Szignó program ingyenes változatával



nak hosszú távú biztosítására is. E szabvány szerint a második és a harmadik változat könnyen megoldható, az első változathoz az egyes szoftverek gyártóinak kell megvalósítaniuk a XAdES aláírások kezelését.

3. Érvényes-e az aláírás?

Ha szoftverünk ismeri az elektronikus aláírás formátumát, akkor meg tudja állapítani, hogy műszaki szempontból *érvényes-e*, tehát valóban kapcsolódik-e hozzá a jogszabályok szerinti bizonyító erő. Így, mielőtt egy aláírt dokumentum alapján fontos döntést hozunk, célszerű megvizsgálni, hogy az aláírás érvényes-e egyáltalán. A [2] szerint ekkor meg kell néznünk azon *hitelesítési rendet,* amely szerint az aláíró tanúsítványát kibocsátották, és eszerint kell eljárnunk. Ha nem így teszünk, és esetleg érvénytelen aláírás alapján hozunk fontos döntést, akkor a felmerülő károkért sem az aláíró, sem az aláíró tanúsítványát kibocsátó hitelesítés szolgáltató nem vállal felelősséget. A hitelesítés szolgáltatók hitelesítési rendjei általában a nemzetközi szabványok – például [5,6] – által leírt lépéseket tartalmazzák:

1. Ellenőrizni kell, hogy az aláírás az aláíró tanúsítványához és az aláírt dokumentumhoz tartozik-e. Ez a kriptográfiai ellenőrzés, amelyet szinte minden alkalmazás támogat.
2. Ellenőrizni kell, hogy az aláíró tanúsítványa visszavezethető-e egy megbízható hitelesítés szolgáltató tanúsítványára. Ezt nevezzük a tanúsítványlánc felépítésének. (Erről bővebben az 5. fejezetben szólunk.)
3. Ellenőrizni kell, hogy nem vonták-e vissza a tanúsítványláncban szereplő tanúsítványok bármelyikét. (Erről a 6. fejezetben szólunk bővebben.)

Egy professzionális aláírás-ellenőrző alkalmazás a fenti lépések mindegyikét elvégzi. A fenti lépéseken túl azt is el kell dönteni, hogy az adott típusú tanúsítvány elfogadható-e az adott célra. Ez nemcsak műszaki lépéseket jelent, ennek a problémának jogi, gazdasági és szabályozási vonzatai is vannak. A 7–9. fejezetek ilyen kérdésekről szólnak.

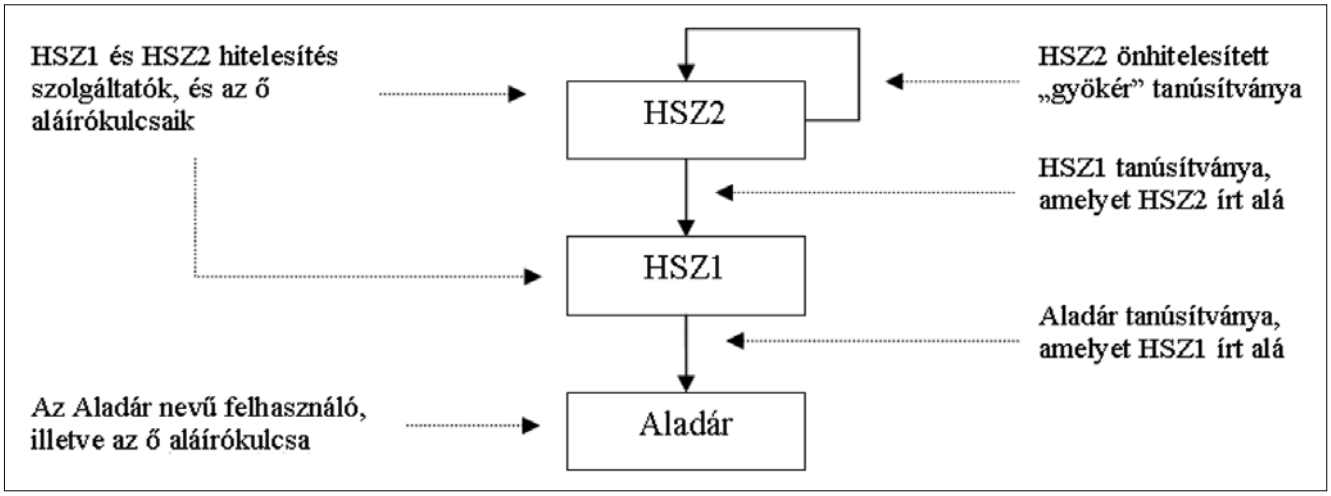
4. Bizonyítható-e, hogy mikor készült az aláírás?

Az aláírás akkor érvényes, ha az aláíró tanúsítvány érvényes volt az aláírás létrehozásának időpontjában. Ebből következően, ha nem bizonyítható az aláírás időpontja, akkor az érvényessége is megkérdőjelezhetővé válhat. Ez akkor fordulhat elő, ha az aláíró tanúsítványa később lejár, esetleg felfüggesztik vagy visszavonják. Ilyenkor később nem lehet majd bizonyítani, hogy az aláírás akkor készült-e, amikor a tanúsítvány még érvényes volt. Ez azt jelenti, hogy hiába győződünk meg egy aláírt dokumentum érvényességéről, *ha az aláírás időpontja nem bizonyítható, akkor az aláíró később letagadhatja az aláírását.* Többféle módon bizonyíthatjuk, hogy az aláírás mikor készült, de erre az elektronikus időbélyeg jelenti a legkézenfekvőbb módszert.

Az időbélyeg az időbélyegzett dokumentum lenyomatát és az időbélyegzés időpontját tartalmazza. Ezen időpontot egy *időbélyegzés szolgáltató*, egy bevizsgált és biztonságos szervezet szolgáltatja, mielőtt aláírja az időbélyeget, mely valójában az időbélyegzés szolgáltató általi igazolása arról, hogy az időbélyeggel ellátott dokumentum az időbélyegen szereplő időpontban létezett. Az aláíráson lévő időbélyeg azt igazolja, hogy az nem készülhetett az időbélyegen szereplő időpontnál később. Ha egy aláírt dokumentumon nincsen időbélyeg, akkor célszerű a legrövidebb időn belül elhelyezni rajta egyet (amíg érvényes a tanúsítvány); így meggátolhatjuk, hogy az aláíró később letagadja az aláírását.

Az időbélyegek jelentik az elektronikus aláírásokra épülő rendszer egyik alapkövét, így a közigazgatási keretrendszer is – nagyon helyesen – kimerítően foglalkozik az időbélyegekre vonatkozó követelményekkel. Ugyanakkor, nemcsak a [2] szerinti időbélyeget, hanem az időjelzést is elismeri. Az *időjelzés* abban különbözik az időbélyegtől, hogy az időjelzést kibocsátókra nem érvényes az időbélyegzés szolgáltatókra vonatkozó erős követelményrendszer, így a közigazgatás az időbélyegnél sokkal olcsóbban állíthat elő – esetleg gyenge minőségű – időjelzést.

2. ábra Egy egyszerű tanúsítványlánc



5. Melyik gyökér tanúsítványra vezetjük vissza az aláírást?

Tegyük fel, hogy az Aladár nevű felhasználó (akinek tanúsítványát a HSZ1 hitelesítés szolgáltató bocsátotta ki) elektronikusan aláírt üzenetet küld Bélának *(2. ábra)*. Ha Béla a 3. fejezetben leírtak szerint *ellenőrzi Aladár aláírását, akkor egy „megbízható" tanúsítványra (gyökér tanúsítványra) próbálja meg visszavezetni az aláírást.* Béla akkor tekintheti egy hitelesítés szolgáltató tanúsítványát megbízhatónak, ha *ismeri* a hitelesítés szolgáltatót, és *megbízik benne,* valamint *hitelesen,* biztonságos módon *hozzájutott* a hitelesítés szolgáltató *tanúsítványához* [1]. Könnyen előfordulhat, hogy Béla nem ismeri HSZ1 tanúsítványát. Ilyenkor Béla (pontosabban, a számítógépes program, amelyet Béla használ) egy olyan tanúsítványláncot keres, amely a Béla számítógépén (például a Béla Windows-ában) lévő megbízható tanúsítványok egyikétől indul, és az Aladár tanúsítványát kibocsátó HSZ1 tanúsítványáig tart. (Ennek algoritmusát a [6] ismerteti részletesen.)

Sajnos a gyakorlatban a helyzet a 2. ábránál sokkal bonyolultabb. A *3. ábrán* látható, hogy egy hitelesítés szolgáltató egy kulcsához egyszerre több tanúsítvány is tartozhat. Ha ezeket különböző időben bocsátották ki, akkor az egyes tanúsítványok különböző információkat tartalmaznak arról, hogy a hitelesítés szolgáltató adott kulcsához mely hitelesítés szolgáltatók mely kulcsaival bocsátottak ki tanúsítványt. A 3. ábra alapján jogosan merül fel a kérdés, hogy melyik szolgáltató tanúsítványa a megbízható gyökértanúsítvány.

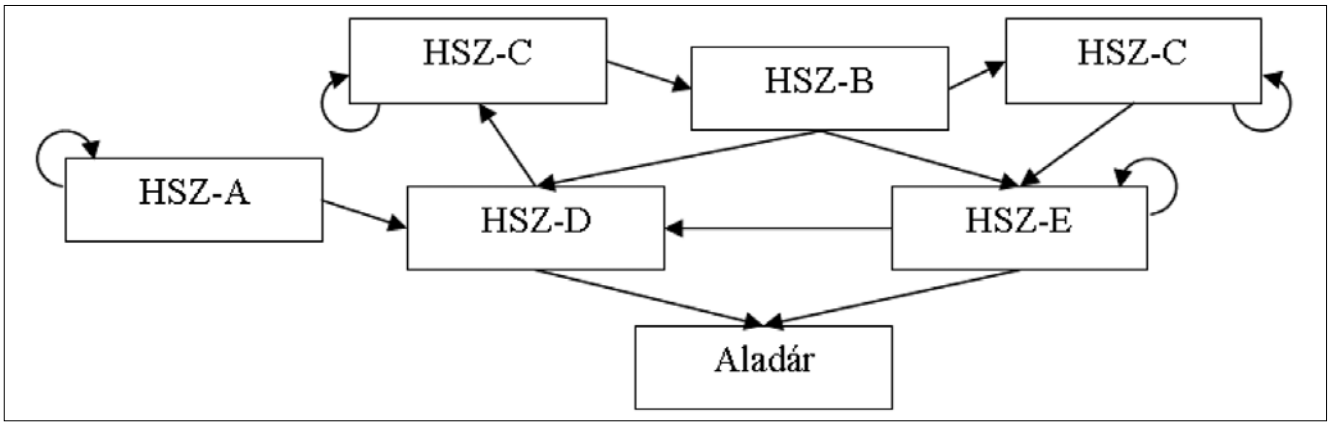
Önhitelesített tanúsítványt – amely gyökér tanúsítványként is működhet – bárki bármikor kibocsáthat önmagának, de ettől ez még nem lesz megbízható tanúsítvány. *Egy tanúsítvány attól „megbízható", hogy mások megbíznak benne,* tehát elfogadják azokat az aláírásokat, amelyeket erre a tanúsítványra vezetnek vissza. Bizonyos szolgáltatók dönthetnek úgy, hogy bizonyos kulcsaikat önhitelesített gyökér tanúsítványként is közzéteszik, így lehetővé teszik, hogy mások aláírásokat vezessenek vissza e tanúsítványokra. A felhasználók, azaz a piac dönti el, hogy megbízik-e ebben a tanúsítványban, és elismeri-e gyökértanúsítványnak.

E döntés kimenetele függhet attól, hogy mekkora a bizalom a szolgáltató informatikai rendszere iránt, mekkora anyagi felelősséget vállal a szolgáltató a rendszer helyes működéséért, mennyire könnyen érhetőek el a visszavonási információk, mennyire könnyű hitelesen hozzáférni a gyökér tanúsítványhoz, és hány aláírás és időbélyeg vezethető vissza rá. Természetesen függ a jogszabályoktól is, például attól, hogy minősített tanúsítványokat bocsát-e ki a szolgáltató e tanúsítvány alapján. *Aláíráskor az aláíró visszavezetheti az aláírást valamelyik önhitelesített tanúsítványra, a befogadó pedig keres egy olyan gyökértanúsítványt, amelyikre az aláírás szintén visszavezethető.* A két gyökér nem feltétlenül ugyanaz.

Béla határozza meg, hogy mely gyökereket fogad el megbízható tanúsítványnak, de – attól függően, hogy milyen szoftvert használ az aláírás ellenőrzésére – szoftverében már eleve telepítve vannak bizonyos gyökerek. Sajnos nem általános, hogy a magyar hitelesítés szolgáltatók tanúsítványai szerepelnének a külföldi szoftverekben. Ezért az a jellemző, hogy az Aladár tanúsítványát kibocsátó HSZ1 tanúsítványa a jogszabályok szerint érvényes, de a Béla szoftverében lévő egyik megbízható tanúsítványra sem vezethető vissza. Ha egyáltalán visszavezethető Aladár aláírása egy olyan tanúsítványra, amelyben Béla megbízik, akkor sem nyilvánvaló, hogy Béla szoftvere megtalálja ezt a láncot. Megoldást jelent e problémára, ha Aladár, az aláíró maga építi fel a tanúsítványláncot, és az aláírásával együtt azt is elküldi Bélának. Ugyanakkor, ha Béla kizárólag az Aladártól kapott tanúsítványláncra hagyatkozik, akkor előfordulhat az a szerencsétlen eset, hogy Béla nem bízik meg abban a tanúsítványban, amelyikre Aladár visszavezette az aláírását, pedig van egy másik olyan megbízható tanúsítvány Béla tanúsítványtárában, amelyre az aláírás visszavezethető lett volna.

A [7] megjelenése előtt a magyar szolgáltatók önálló gyökér tanúsítványokkal rendelkeztek. Így csak az fogadhatta el az összes magyar elektronikus aláírást, aki minden magyar hitelesítés szolgáltató minden gyökér tanúsítványát megbízható tanúsítványnak fogadta el. A [7] szerint egy közigazgatási gyökér hitelesítés szolgáltató [8] jött létre, amely tanúsítványt bocsát ki a köz-

3. ábra Egy kusza, de realisztikus szituáció



igazgatásnak is szolgáltató hitelesítés szolgáltatók bizonyos kulcsai számára. Így bizonyos aláírások esetén a KGYHSZ tanúsítványa közös megbízható gyökér tanúsítványt jelent. Sajnos a KGYHSZ időbélyegzőket és OCSP válaszadókat nem tanúsíthat felül, emiatt az ezeket tartalmazó aláírás (és hosszú távú aláírás mindig tartalmaz ilyet) ellenőrzésekor a hitelesítés szolgáltatók eddigi gyökér tanúsítványaira is szükség van. Tehát a KGYHSZ csak részmegoldást nyújt a problémára.

6. Minek a visszavonási állapotát ellenőrizzük és hogyan?

Ha egy tanúsítványhoz tartozó titkos aláírókulcs elvész vagy illetéktelen kezekbe kerül (például, ha a tanúsítványhoz tartozó kártyát ellopják), szólni kell a hitelesítés szolgáltatónak, és az visszavonja a kibocsátott tanúsítványt. A visszavonás azt jelenti, hogy a hitelesítés szolgáltató közzéteszi, hogy a tanúsítvány érvénytelen. Ezért mielőtt elfogadunk egy tanúsítványt, meg kell győződnünk róla, hogy a tanúsítvány nincs visszavonva (vagy felfüggesztve). Magyarországon két technológia terjedt el, amelyek segítségével megtudhatjuk egy tanúsítvány visszavonási állapotát: az egyik a visszavonási lista (CRL), a másik pedig a kérdés-válasz alapú online tanúsítvány-állapot szolgáltatás (OCSP). A visszavonási lista a visszavont tanúsítványok sorozatszámát tartalmazza. Minden magyar hitelesítés szolgáltató bocsát ki visszavonási listát, és a visszavonási listák ingyenesen elérhetőek. Az online tanúsítvány-állapot szolgáltatás egy olyan protokollt jelent, amellyel rákérdezhetünk, hogy egy tanúsítvány az adott pillanatban érvényes-e, és kérdésünkre a szolgáltató azonnali hiteles választ ad, így sok esetben gyorsabb, praktikusabb, mint a CRL. A négy magyar kereskedelmi hitelesítés szolgáltatóból három nyújt OCSP szolgáltatást.

Két okból van szükség visszavonási információkra (CRL-ekre és OCSP válaszokra): Egyrészt, aláírás befogadásakor a visszavonási információk alapján állapítjuk meg, hogy a tanúsítvány az aláírás pillanatában érvényes volt-e. Másrészt, később a visszavonási információk segítségével igazolhatjuk, hogy az aláírást valóban megalapozottan fogadtuk el. Ezáltal a visszavonási információk védik az aláírás befogadóját, és biztosítják az aláírás letagadhatatlanságát. A gyakorlatban előfordulhat, hogy a visszavonási információkat később nem könnyű összegyűjteni (például azért, mert az [6] szerint CRL nem feltétlenül tartalmazza a már lejárt tanúsítványokat), így elterjedt, hogy a visszavonási információkat nem a befogadó gyűjti össze, hanem az aláíró csatolja őket az aláírásához. Ezt általában a befogadó kényszeríti ki: kijelenti, hogy csak olyan aláírásokat fogad, amelyek tartalmazzák az aláírásra vonatkozó visszavonási információkat.

Milyen információk tartoznak ide? Az aláíró tanúsítványától a megbízható gyökér tanúsítványig tartó tanúsítványlánc minden elemére vonatkozó visszavonási lista vagy OCSP válasz, amely igazolja, hogy az adott tanúsítvány az aláírás időpontjában érvényes volt. Ha az aláírás időbélyeget is tartalmaz (lásd 4. fejezet), akkor az időbélyegre vonatkozó visszavonási információkra is szükség van. Emellett a visszavonási információkon is aláírás szerepel, így az aláírás ellenőrzéséhez a visszavonási információkon lévő aláírásokra vonatkozó visszavonási információk is szükségesek.

Ha azt szeretnénk eldönteni, hogy elfogadhatunk-e egy aláírást, akkor arra a kérdésre keressük a választ, hogy a tanúsítvány *az aláírás időpontjában érvényes volt-e.* Ebből következően, pusztán az aláírás időpontja előtt kibocsátott visszavonási listák és OCSP válaszok alapján nem fogadhatunk el aláírást, hanem az aláírás időpontját követő visszavonási listára vagy OCSP válaszra van szükségünk. Tegyük fel, hogy egy szolgáltató minden nap éjfélkor bocsátja ki a CRL-ét. Ilyenkor, ha egy aláírás délután 2-kor jön létre, akkor egészen addig nem lehet CRL alapján megállapítani, hogy érvényes-e, amíg a következő CRL meg nem jelenik. Lehet, hogy a tanúsítványt reggel 9-kor visszavonták, de CRL alapján várhatóan csak éjfélkor szerzünk tudomást erről. Ez azt jelenti, hogy az aláírt dokumentummal kapcsolatban érdemi döntést éjfél előtt nem hozhatunk.

A szakirodalom „kivárási időnek" (grace period) nevezi azt az időszakot, amikor az aláírás már rendelkezésre áll, de a visszavonási információk beszerzéséig nem lehet megállapítani az érvényességét. OCSP használata esetén a fenti probléma esetleg meg sem jelenik. OCSP segítségével a délután 2-kor létrehozott aláírás érvényességét akár 2:01-kor is megkérdezhetjük, és a kérdésre a hitelesítés szolgáltató azonnali, aláírással hitelesített választ ad. Egyes szolgáltatók vállalják, hogy soron kívül kibocsátanak egy új CRL-t, ha egy tanúsítvány visszavonási állapota megváltozik. Ha azt látjuk, hogy nem jelent meg új CRL, akkor a tanúsítvány érvényes ugyan, de nem rendelkezünk olyan visszavonási információval, amelyet az aláíráshoz csatolhatnánk, hogy később bizonyíthassuk annak érvényességét. OCSP esetén nincsen ilyen probléma, mert érvényes tanúsítvány esetén is hiteles OCSP választ kapunk, amelyet csatolhatunk az aláíráshoz. További probléma a CRL-lel, hogy a szabványok nem írják elő, hogy az alkalmazásoknak figyelnie kell a soron kívül kibocsátott CRL-eket, így nem biztos, hogy minden szabványos alkalmazás mindig ugyanarra az eredményre jut, ha tanúsítványt CRL alapon ellenőriz.

A szabványok szerint *a tanúsítványláncban szereplő minden tanúsítvány visszavonási állapotát ellenőriznünk kell.* Ezen ellenőrzés történhet *elektronikusan* CRL vagy OCSP alapján, vagy *manuálisan,* amikor a szolgáltató vállalja, hogy például újsághirdetést tesz közzé, ha aláírókulcsa illetéktelen kezekbe kerül. Ha sok embernek gyakran kell elvégeznie az ellenőrzést, akkor az elektronikus megoldás mindenképpen olcsóbb. A manuális megoldás viszont néha elkerülhetetlen, az önhitelesített gyökér tanúsítványok visszavonási állapotát például kizárólag így lehet ellenőrizni. Egy nagy rendszerben az a jó, ha a lehető legkevesebb olyan tanúsítvány van benne, amelyet csak manuálisan lehet ellenőrizni, s amelyekre a többi tanúsítványt visszavezet-

jük. *Ha több szolgáltatói tanúsítvány szerepel a tanúsítványláncban, akkor az egyes szolgáltatóknál jelentkező kivárási idők legnagyobbika számít.* Tegyük fel, hogy a 2. ábrán szereplő HSZ1 naponta bocsát ki visszavonási listát, HSZ2 pedig havonta (amely igen gyakori gyökér hitelesítő egységek esetén). Ekkor, ha Aladár aláírását HSZ2 gyökér tanúsítványára szeretnénk visszavezetni, és a tanúsítványlánc elemeit CRL alapján ellenőrizzük, akkor akár 1 hónapot kell várnunk Aladár aláírásának ellenőrzéséhez. OCSP segítségével ez az ellenőrzés is mindössze néhány másodperc. Ha gyorsan szeretnénk ellenőrizni egy aláírást, akkor az OCSP egyértelműen jobb technológia a CRL-nél, de ha sok aláírt dokumentumot szeretnénk tárolni, akkor a CRL a praktikusabb: ilyenkor egyetlen CRL sok aláírás érvényességét igazolhatja, míg OCSP esetén külön OCSP válasz szükséges minden egyes aláíráshoz [9,10].

A közigazgatási keretrendszer mind a CRL, mind az OCSP technológiát elismeri. Kötelezi a hitelesítés szolgáltatókat, hogy egy visszavonás bejelentésétől számított 4 órán belül új CRL-t tegyenek közzé, ezzel felgyorsítja a CRL alapú ellenőrzést. Sajnos egyúttal az aláírás formátumáról szóló specifikációban mind a CRL, mind az OCSP használata esetén 4 óra kivárási időt ír elő. Ezen előírás akkor is érvényes, ha a hitelesítés szolgáltató 4 óránál sokkal gyorsabban fel tud dolgozni egy visszavonási kérelmet. A felhasználónak ekkor is 4 órát kell várnia, mert csak ekkor tudja később igazolni, hogy egy aláírást valóban jogosan fogadott el. A keretrendszer nem ösztönzi a hitelesítés szolgáltatókat, hogy versenyezzenek, hogy melyik biztosít gyorsabb, megbízhatóbb, jobb minőségű visszavonás kezelést. Ehelyett rákényszeríti a felhasználókra a gyenge technológiákat alkalmazó hitelesítés szolgáltatók nyújtotta minőséget. Ez azt eredményezi, hogy az elektronikus aláírással történő közigazgatási ügyintézés legalább 4 óráig mindenképpen eltart, pusztán az elektronikus aláírás ellenőrzése miatt. Eközben a négy magyar kereskedelmi hitelesítés szolgáltató közül három is (Microsec, Magyar Telekom, Máv Informatika) felelősséget vállal azért, hogy azonnal (azaz nem 4 óra alatt) tudja megváltoztatni az általa kibocsátott tanúsítványok visszavonási állapotát.

A keretrendszer szerint minden aláírást a KGYHSZ tanúsítványára kell visszavezetni. Így a tanúsítványláncban három tanúsítvány szerepel: az aláíró tanúsítványa, az „első szintű” hitelesítés szolgáltató tanúsítványa és a KGYHSZ gyökér tanúsítványa. Az első szintű hitelesítés szolgáltatók 24 óránként adnak ki CRL-t, és közülük három OCSP szolgáltatást is nyújt. A KGYHSZ (amely csak az első szintű hitelesítés szolgáltatóknak ad tanúsítványt) 35 naponta, de visszavonás esetén 1 napon belül bocsát ki új CRL-t. A KGYHSZ nem nyújt OCSP szolgáltatást, igaz, a tanúsítványában – nagyon bölcsen – elhelyezték egy később bekapcsolható OCSP szolgáltatás elérhetőségét. Ha a KGYHSZ tanúsítványára vezetünk vissza egy aláírást, akkor esetleg csak 35 nap után tudjuk igazolni, hogy valóban gondosan ellenőriztük az aláíró tanúsítványának érvényességét.

7. Ki írta alá a dokumentumot?

Az aláíró személyéről az aláírásban szereplő aláírói tanúsítvány révén kaphatunk információt. Sajnos – ahogy erre [11] is rámutat – a tanúsítványban szereplő adatok alapján nem feltétlenül könnyű feladat megállapítani, hogy ezen adatok pontosan mely személyt jelentik. A magyar hitelesítés szolgáltatók adatvédelmi okok miatt csak akkor írhatnák bele a tanúsítványba az aláíró valamely igazolványának számát, ha az aláíró ebbe beleegyezik. Ez azt jelenti, hogy az aláírást befogadó fél nem várhatja el, hogy az aláíró tanúsítványában igazolványszám szerepeljen. Így az elektronikus aláírásból – a kézzel írott aláíráshoz hasonlóan – egyetlen egy információ derül ki biztosan az aláíróról: a neve.

Tovább bonyolítják a problémát az úgy nevezett *álneves tanúsítványok*. A [2] lehetővé teszi, hogy a tanúsítványban ne az aláíró valódi neve, hanem „álnév” szerepeljen. A magyar hitelesítés szolgáltatóknak kötelező lehetővé tenni, hogy ügyfeleik álneves tanúsítványt is igényelhessenek, így ha elektronikusan aláírt dokumentumot kapunk, célszerű megvizsgálni, hogy nem álneves-e a tanúsítvány. Ha a tanúsítvány álneves, a benne lévő név nem az aláíró neve. A [2] szerint az álneves tanúsítványban jelölni kell, ha a tanúsítvány álnevet tartalmaz. Sajnos a magyar hitelesítés szolgáltatók ezt többféle módon jelölik, így ahány szolgáltató, annyi módon lehet ellenőrizni, hogy a tanúsítvány álneves-e. Az álneves tanúsítvánnyal aláírt minősített aláírás azonos bizonyító erővel bír, mint az, amelyiket nem álneves tanúsítvánnyal hoztak létre, csupán az nem derül ki belőle, hogy pontosan ki hozta létre az aláírást, ki vállalt az aláírással kötelezettséget. Bíróság elkérheti a hitelesítés szolgáltatótól az álnevet viselő személy adatait, de senki nem tudja megállapítani az aláíró személyazonosságát, amíg eddig nem fajul az ügy. Így az álneves aláírás hiába érvényes, hiába bír jogilag bizonyító erővel, általában senki nem hajlandó elfogadni azt.

A közigazgatási keretrendszer meghatározza, hogy milyen formátumú tanúsítványokat használhatnak a közigazgatást képviselő személyek, és azt is, hogy a közigazgatáshoz milyen formátumú tanúsítványokkal fordulhatunk. E keretrendszer határozott útmutatást ad arra, hogy mely mezőbe pontosan milyen értékek kerülhetnek, láthatóan a szolgáltatók hajlandóak tanúsítványprofiljaikat a közigazgatás által megadott, a szabványoknak és nemzetközi ajánlásoknak is megfelelő profilra alakítani. Így e keretrendszer rendet tesz a hazai kusza tanúsítványprofilok között. A keretrendszer – nagyon helyesen – kizárja az álneves tanúsítványok használatát is.

Adatvédelmi okok miatt csak az aláíró neve derülhet ki az elektronikus aláírásából, így az aláírást befogadó fél két módon járhat el: Feltételezheti, hogy az aláíró az, akinek mondja magát. Amennyiben e feltételezés hamisnak bizonyul, akkor bíróság előtt felelősségre vonhatja, és a bíróság már jogosult utánajárni, hogy pontosan ki készítette az aláírást. Másik lehető-

ség, hogy különböző trükkös módszerekkel igyekszik meggyőződni az aláíró kilétéről. Ma többen használják azt a módszert, hogy egy szervezettől papíron kiállított, cégszerűen aláírt igazolást kérnek arról, hogy az adott (kiállítójú és sorozatszámú stb.) tanúsítvány birtokosa jogosult elektronikusan eljárni a szervezet nevében. A közigazgatás egy „viszontazonosítás" névre hallgató protokollt dolgozott ki, amelyben az aláírást befogadó közigazgatási szerv XML alapú kérdéseket tehet fel a hitelesítés szolgáltatónak. Az aláíró természetes azonosító adataira (neve, anyja neve, születési ideje) kérdezhet rá, amelyre a hitelesítés szolgáltató csak „igen" vagy „nem" választ adhat. A világon egyedi PKI-megoldásról van szó, amely a „barkohba" nevű játékra emlékezetet.

8. Jogosan írta-e alá az aláíró a dokumentumot?

Ha tudjuk, hogy az aláíró kicsoda, akkor is felmerül a kérdés, hogy jogosan írta-e alá a dokumentumot. Ugyanez a probléma merül fel a kézzel írott aláírások esetében is, csak a papíron történő ügyintézés esetén sokkal lassabb, valamint a kézzel írott aláírásokat ellenőrző emberek sokkal kevésbé hajlamosak szarvashibákat elkövetni, mint az automaták. Így e probléma nem az elektronikus aláírás technológiából, hanem az automatizált elektronikus ügyintézésből, a gyors elektronikus kommunikáció támasztotta követelményekből ered.

A tanúsítványban fel lehet tüntetni, hogy az aláíró mely szervezethez tartozik. Az jelenti a nehéz kérdést, hogy az aláíró jogosult-e az adott szervezet nevében aláírni és amennyiben igen, akkor pontosan milyen feltételek mellett. Sok hitelesítés szolgáltató feltünteti a tanúsítványban az aláírási jogosultságot, de itt sem beszélhetünk egységes jelölésrendszerről. Ennek megfelelően a közigazgatás – logikusan – megtiltja, hogy kizárólag a tanúsítvány alapján állapítsák meg a közigazgatás egy ügyfelének képviseleti jogosultságát. A keretrendszer külön hitelesítési rendeket definiál a közigazgatás szereplői és a közigazgatás ügyfelei számára, így a hitelesítési rend alapján számítógép is el tudja dönteni például azt, hogy egy adott aláírást valóban köztisztviselő hozott-e létre. Praktikus megoldásról van szó, amely reális, elérhető célt tűzött ki, és a szabványoknak megfelelő módon éri el azt.

9. Mi a biztosíték arra, hogy bízhatunk a tanúsítványban?

Ha elfogadunk egy elektronikus aláírást, akkor – a hitelesítés szolgáltató által kibocsátott tanúsítvány alapján – elhisszük, hogy az aláírás létrehozásához használt aláírókulcs az aláírás pillanatában az aláíró birtokában volt. A hitelesítés szolgáltató garantálja, hogy a tanúsítvány kibocsátásakor az aláírókulcs az aláíró birtokában volt, és amint tudomást szerez róla, hogy ez már nem így van (például az aláíró bejelenti, hogy elveszítette az intelligens kártyáját), a hitelesítés szolgáltató haladéktalanul visszavonja a tanúsítványt. Természetesen előfordulhat, hogy a hitelesítés szolgáltató hibázik. Bármilyen erős biztonsági intézkedések is működnek nála, előfordulhat, hogy valaki megtéveszti a szolgáltatót, és valaki egy másik ember nevében igényel tanúsítványt, de az is lehet, hogy a szolgáltató nem elég gyorsan von vissza egy kibocsátott tanúsítványt. Ekkor előfordulhat, hogy az aláírást nem a tanúsítványban szereplő aláíró hozza létre, és ez nem derül ki a tanúsítványra vonatkozó visszavonási információkból.

Ha a hitelesítés szolgáltató hibájából valakinek kára keletkezik, a szolgáltató köteles megtéríteni a kárt – az adott tanúsítványra vonatkozó feltételek szerint. A [2] szerint *a hitelesítés szolgáltató korlátozhatja* az egyes tanúsítványokkal egy alkalommal vállalható kötelezettség mértékét. Ezzel a szolgáltató az egy aláírással okozható kár mértékét, és így *saját kártérítési kötelezettségét* korlátozza. Az [2] szerint e korlátozást fel kell tüntetni a tanúsítványban. Ez az érték az úgynevezett *tranzakciós limit*.

A kézzel írott aláírást tipikusan személyi igazolvány vagy aláírási címpéldány alapján szokás ellenőrizni. Általában attól függ, hogy milyen alaposan ellenőrzünk egy kézzel írott aláírást, hogy mekkora kárunk származhat abból, ha az aláírás nem érvényes. Ha elektronikus aláírással kötjük a szerződést, akkor is mérlegelnünk kell, hogy mennyire fontos, hogy a másik fél aláírása érvényes legyen. Elektronikus aláírás esetében is léteznek különféle szintek (például a teljes tanúsítványlánc megkövetelése, az aláírás időpontja után kibocsátott visszavonási információk megkövetelése), de bármilyen gondosan is járunk el az aláírás ellenőrzésekor, a hitelesítés szolgáltató hibája ellen egyedül a szolgáltató kártérítési kötelezettsége véd bennünket. Ez biztosít bennünket arról, hogy nem lesz anyagi kárunk abban az (egyébként rendkívül kis valószínűségű) esetben sem, amikor a hitelesítés szolgáltató hibát követ el. A kártérítési kötelezettség egyúttal arra ösztönzi a szolgáltatót, hogy saját informatikai rendszerét és belső szabályzatait úgy alakítsa ki, hogy a hiba valószínűsége a lehető legkisebb legyen.

Például, a 2. ábrán szereplő tanúsítványlánc a következőket jelentheti: A „HSZ2" tanúsítványról tudom, hogy a HSZ2 szolgáltatóé, és HSZ2-ben megbízom. HSZ2 azt állítja, hogy a „HSZ1" tanúsítvány a HSZ1 szolgáltatóé. HSZ1 pedig azt állítja, hogy az „Aladár" tanúsítvány Aladáré. Nemcsak az számít, hogy Aladár mennyire megbízható; az „Aladár" tanúsítvánnyal történő aláírás semmit sem ér, ha HSZ1 vagy HSZ2 állítása hamis. *Mennyire bízhatunk HSZ1 vagy HSZ2 állításában?* Mekkora kárt hajlandóak megtéríteni, ha kiderül, hogy hibáztak? *Mennyire biztonságosan kezelik a tanúsítványaikat: mit mondanak, milyen értékű tranzakciókhoz használhatóak?*

Aláírás befogadásakor meg kell vizsgálnunk, hogy az aláíráshoz tartozó tanúsítványért mely hitelesítés szolgáltató vállal felelősséget. Ha a tanúsítványláncban több szolgáltató is szerepel, akkor célszerű egészen a gyökér tanúsítványig ellenőrizni, hogy mely ta-

núsítványért mekkora kártérítési kötelezettséget vállal a tanúsítványt kibocsátó szolgáltató. Emellett mérlegelnünk kell, hogy az aláíró az aláírt dokumentumban mekkora kötelezettséget vállal (nem pénzügyi jellegű dokumentum esetén ez nem könnyű feladat), és ennek függvényében kell döntenünk, hogy az adott esetben az elfogadjuk-e az adott aláírást. Sok szolgáltató 0 Ft tranzakciós limittel is bocsát ki tanúsítványt. Az ilyen tanúsítványokkal elvileg semekkora anyagi kötelezettség nem vállalható, így elképzelhető, hogy a szolgáltató egyáltalán nem vállal anyagi felelősséget a tanúsítvány helyes kezeléséért. Nehéz megítélni, hogy ez pontosan mit jelent, várhatóan az ezzel kapcsolatos bírói ítéletek alakítják majd ki, hogy a tranzakciós limit hogyan viszonyul a szolgáltatók kártérítési kötelezettségéhez.

A KGYHSZ minden felelősséget elhárít a hitelesítési rendjében az általa kibocsátott szolgáltatói tanúsítványokkal kapcsolatban [8]. Ez igen bizarr esetet eredményez: a közigazgatás egy olyan szolgáltatóra vezette vissza minden elektronikus aláírás biztonságát, amely semmilyen pénzügyi garanciát nem tud vállalni a saját biztonságos működéséért. Így gazdasági erő nem ösztönzi a KGYHSZ-t biztonságos működésre. Míg a közigazgatási szervek számára előírás a KGYHSZ gyökértanúsítványának használata, a magánszférára ez nem vonatkozik. Ha egy vállalat a KGYHSZ tanúsítványára visszavezetett aláírást kap, akkor azt látja, hogy ha a KGYHSZ hibát követ el, abból olyan kára származhat, amelyet senki nem köteles neki megtéríteni. Így nem várható, hogy egy gazdasági szempontból racionálisan gondolkodó szervezet elfogadja a KGYHSZ tanúsítványára visszavezetett aláírásokat.

10. Összefoglalás

Egy elektronikus aláírás elfogadásához számos feladatot kell elvégeznünk. Ezek jelentős részét szoftver is megoldhatja helyettünk, egy másik részéhez azonban emberi döntés szükséges – akárcsak a papíron történő aláírások esetében. A közigazgatás által kidolgozott keretrendszer ezen lényeges problémákra ad egyfajta választ. E keretrendszer szükséges: aki úgy dönt, hogy elektronikus aláírást is elfogad, az jól teszi, ha meghatározza, hogy pontosan milyen aláírást fogad el. Ugyanakkor, ha később egy másik közösség – például a bankok, vagy az EU – hasonló szintű elvárásokat – például saját gyökeret – határoz meg, az nagyon könnyen előidézheti, hogy a két követelményrendszer kölcsönösen kizárja egymást és az egyik rendszerben használt tanúsítványokat, aláírásokat a másik rendszer nem fogadja el.

A közigazgatás alaposan kidolgozott, a nemzetközi szabványoknak megfelelő specifikációt készített, amely – egy-két szerencsétlen megoldástól (például a kötelezően 4 órás kivárási idő, és az OCSP nélküli, minden felelősséget kizáró KGYHSZ) eltekintve – hasznos, mert várhatóan összefogja, egységesíti a hazai elektronikus aláírás piacon használt technológiák sokaságát.

Nem véletlen, hogy az elektronikus aláírás csak lassan kezdett terjedni. A gyakorlati alkalmazás számos olyan problémát vetett fel, amelyre sem a technológia, sem a jogszabályok nem adnak közvetlen választ. Ha pusztán a rendszer egyes elemeit nézzük – például a hitelesítés szolgáltató, vagy egy PKI-ra épülő alkalmazás fejlesztőjének szemszögéből – e problémák gyakran nem látszanak. Egy hitelesítés szolgáltatónak sokkal egyszerűbb dolga van, ha nem nyújt OCSP szolgáltatást, és az alkalmazásfejlesztőnek is egyszerűbb a dolga, ha nem foglalkozik a kivárási idővel, vagy esetleg a felhasználóra bízza a szolgáltatói tanúsítványok visszavonási állapotának ellenőrzését.

Talán az a helyes, ha egy PKI-ra épülő rendszert a legfontosabb résztvevője, a *felhasználó* szemszögéből nézzük. Fontos, hogy a felhasználó értelmes, elfogadható választ kapjon az itt felsorolt kérdésekre, kizárólag ekkor jelenthet számára hasznos, költséghatékony megoldást az elektronikus aláírás technológiája.

Irodalom

[1] Buttyán L., Vajda I.,
Kriptográfia és alkalmazásai, Typotex Kiadó, 2004.

[2] 2001. évi XXXV. törvény az elektronikus aláírásról.

[3] A közigazgatás elektronikus aláírással kapcsolatos ajánlásai:
www.ihm.gov.hu/jogszabalyok/kapcsolodo_ajanlasok?ayear=2006&amonth=0

[4] ETSI TS 101 903 XML
Advanced Electronic Signatures, V1.2.2 2004.

[5] CWA 14171, General guidelines for electronic signature verification, 2004.

[6] Certificate and Certificate Revocation List (CRL) Profile, 2002.

[7] 2004. évi CXL. törvény a közigazgatási hatósági eljárás és szolgáltatás általános szabályairól.

[8] A Közigazgatási Gyökér Hitelesítés Szolgáltató (KGYHSZ) honlapja, hitelesítési rendje, 2006.
http://www.kgyhsz.gov.hu/

[9] Berta I.,
A CRL és az OCSP összehasonlítása,
Microsec Kft., 2005.
http://www.e-szigno.hu/wp_crl_vs_ocsp.html

[10] Rivest, R.,
Can we eliminate Certificate Revocation Lists?,
Financial Cryptography, 1988.
http://citeseer.ist.psu.edu/rivest98can.html

[11] Ellison, C., Schneier, B.,
Ten Risks of PKI: What You're not Being Told about Public Key Infrastructure,
Computer Security Journal, Vol.16, Nr. 1, 2000.

Anonymizer hálózatok elleni támadások

SZENTPÁL ZOLTÁN

Budapesti Műszaki és Gazdaságtudományi Egyetem, Távközlési és Médiainformatikai Tanszék
zoltan.szentpal@gmail.com

ZÖMBIK LÁSZLÓ

BME Távközlési és Médiainformatikai Tanszék, Ericsson Magyarország, R&D
laszlo.zombik@ericsson.com

Lektorált

Kulcsszavak: anonim kommunikáció, anonymizer hálózatok, privacy

Az anonymizer hálózatok két fő feladata egyrészt a felhasználók forgalmának és kommunikációs partnereinek elrejtése a helyi hálózatok elől, másrészt annak biztosítása, hogy a felhasználók kiléte és helyzete ismeretlen maradjon a távoli hálózatokban. Cikkünkben az anonymizer hálózatok korlátai kerülnek bemutatásra, ismertetünk egy támadási eljárást ezen hálózatok felhasználóinak anonimitása ellen. A támadás lényege, hogy a felhasználó kommunikációs partnerénél a forgalmat torzítjuk. Ezután a partnertől kiindulva, a forgalom torzítást detektálva a felhasználó helyzetét visszakövetjük.

1. Anonymizer hálózatok

A végpontok és a hálózati forgalom hitelesítése, valamint a forgalmazott adatok titkosítása nem képes védeni a felhasználókat olyan támadások ellen, melyek a felhasználók anonimitása ellen irányulnak. A csomagok IP fejlécének forrás cím – cél cím mezőpárja elegendő információt árul el a felhasználóról, ezért a felhasználó IP címét módosítani szükséges. Ezt többféleképpen lehet elérni, például hálózati címfordítással (Network Address Translation), vagy proxy szerverek használatával. Címfordítás esetén az IP címek, proxy szerverek esetén az IP címek és az IP csomagok forrásra jellemző tartalma lecserélődik. Proxy szerverek hálózatából anonymizer hálózatok épülnek fel, melyekben a forgalmat a proxy szerverek láncolatán keresztül továbbítják, esetlegesen titkosítva is azt *(1. ábra)*.

A proxy szerverekből felépült hálózatok lehetnek nyilvánosak, vagy zártak. Nyilvános proxy szervereket általában önkéntesek üzemeltetnek, ezért ezek a hálózatok gyakran kevésbé megbízható szolgáltatást nyújtanak.

A privát proxyk alkalmazása kedvezőbb a teljesítmény és sebesség szempontjából. Használatuk hátránya, hogy ez a megoldás lehetővé teszi az anonymizer hálózatot üzemeltető vállalat számára, hogy megfigyelje a felhasználók forgalmát.

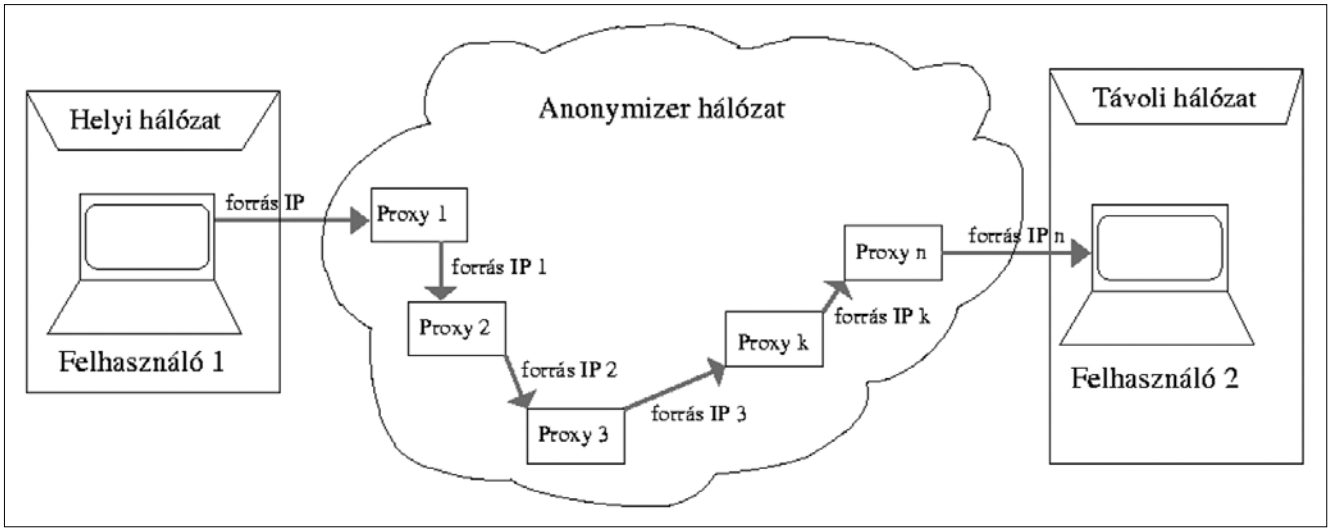
Számos kliens alkalmazás áll a felhasználók rendelkezésére, amelyek különböző anonymizer hálózatokat használnak, különböző mértékű anonimitást nyújtva a felhasználóknak. Sok közülük egyéb funkciókat is kínál (pl. a GhostSurf rendszer még anti-spyware alkalmazást is). Többségük Windows operációs rendszer alapú.

A legnépszerűbb anonymizer hálózatok:

– Steganos Internet Anonym Pro 7 [1]
– Bypass Proxy Client [2]
– Anonymizer 2004 [3]
– MorphMix [4]
– GhostSurf 2005 Platinum [5]
– Tor [6]

Cikkünkben az utolsó két anonymizer hálózaton mutatjuk be az általunk megvalósított támadásokat. Ezeket a hálózatokat a következőkben röviden ismertetjük.

1. ábra Anonymizer hálózatok elvi felépítése



1.1. GhostSurf 2005 Platinum

A GhostSurf 2005 Platinum egy olyan szoftver-gyűjtemény, amely a felhasználók személyes adatainak védelmét és anonimitását segíti elő. A program zárt forráskódú (a vizsgálat során a program ingyenes próbaverzióját használtuk), és csak Microsoft Windows operációs rendszeren fut. A kliens szoftver zárt anonymizer hálózatot használ. A GhostSurf rendszer TCP feletti alkalmazásokat támogat. A kliens alkalmazás és a proxy szerver közötti kommunikáció során lehetőség van SSL/TLS titkosítás használatára is.

1.2. Tor

A Tor anonymizer rendszer a legtöbb, manapság népszerű operációs rendszert támogatja. Nyílt forráskódú, ingyenes szoftver. A Tor egy önkéntesek által karbantartott, elosztott, anonim szerverhálózatot használ. Ehhez a hálózathoz bárki csatlakozhat szerver üzemeltetőként, természetesen csak akkor, ha elegendő sávszélesség áll a rendelkezésére. A hálózatba bekerülő csomagok véletlenszerű útvonalon haladnak a hálózat szerverei között, ezért egy adott megfigyelési pontban lehetetlen megállapítani egy csomag forrás és cél címét. A csomagok küldése előtt a felhasználó kliensprogramja kiválaszt egy útvonalat a hálózat véletlenszerűen kiválasztott szerverei között, az üzeneteket Onion Routing használatával továbbítja *(2. ábra)*.

A kliensnek az aktuális szerverlánc összes szerverével létezik közös kulcsa (*K_ai,* ahol *„a”* a kliens, *„i”* a szerverlánc i-edik szervere). A kliens *„m”* csomag küldése előtt azt mindegyik kulccsal titkosítja, kezdve a legutolsó szerver kulcsával. A legkülsőbb titkosítást a legközelebb lévő szerver kulcsával végzi a kliens, ahogy az az ábrán is látható. Így egy adott szerver csak a csomag visszafejtése után fogja megtudni, hogy hova továbbítsa a csomagot, vagyis hogy melyik a következő szerver a láncban. Az útvonal összes szervere csak az őt megelőző (neki továbbító), és az őt követő szerverrel kommunikál, vagyis egyetlen egy szerver sem ismeri a csomagok teljes útvonalát.

A Tor rendszer csak TCP kapcsolatokat kezel, azonban bármely SOCKS-ot támogató alkalmazással képes együttműködni. A Tor hálózatot, a magánszemélyeken kívül, számos szervezet használja világszerte, például az Amerikai Egyesült Államok Haditengerészete is. Sok vállalat a hagyományos VPN (Virtual Private Network) hálózatuk biztonságosabbá tételére alkalmazza. Törvényszéki nyomozók a kormányzati IP címek elrejtésére használják a Tor rendszert, így a nyomozások során látogatott és megfigyelt honlapok weblogjában nem a saját IP címeik fognak szerepelni.

A forgalom analízisen alapuló támadások hatásosak az anonymizer hálózatok felhasználóinak anonimitása ellen. Tehát ezzel meghatározható, hogy ki kivel folytat kommunikációt egy nyilvános hálózaton keresztül, akkor is, ha a csomagokat titkosították és a fejléceket álcázták. A cikkben egy ilyen támadás megvalósítását mutatjuk be, élő anonymizer hálózatokon.

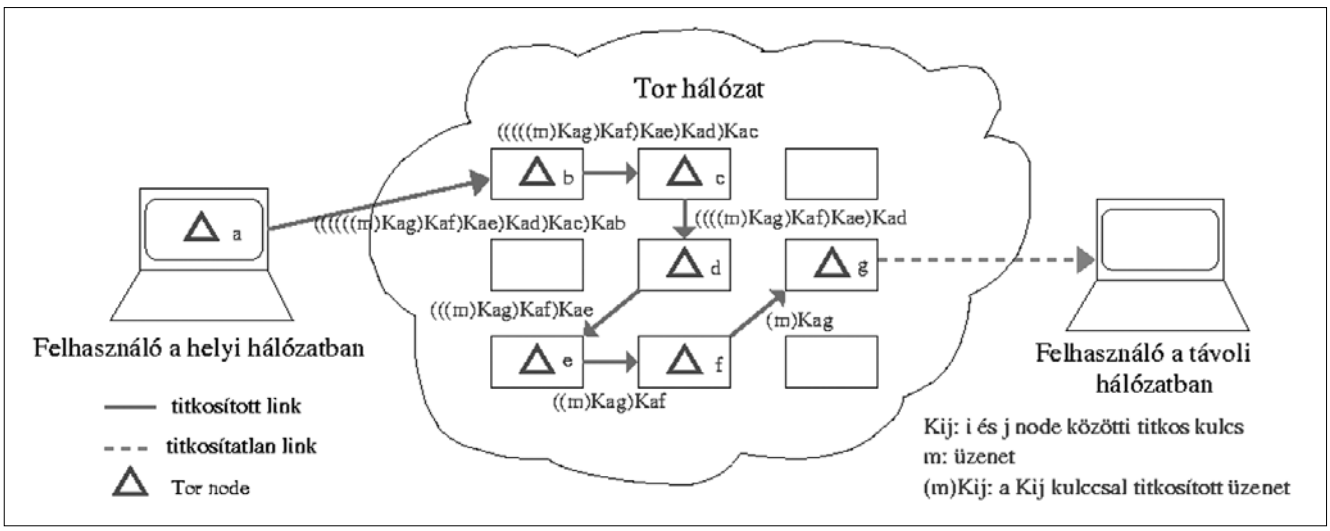
A támadás elméleti hátterét a második, a támadás megvalósításának részleteit a harmadik fejezetben ismertetjük. A negyedik fejezetben az anonymizer hálózatokban mért eredményeinket mutatjuk be.

2. A támadás elméleti háttere

Az ismertetésre kerülő támadás a támadó által készített mérőjel frekvenciatartománybeli vizsgálatán alapul. A hálózatban a bufferelés, a csomagok sorrendjének felcserélése, az üzenetküldési késleltetés változása, valamint a csomagvesztés nemlineáris torzítást eredményez. Azonban megfelelő mérőjel esetén ezek a torzítások nem jelentősek, és a keresendő jel elég ideig történő küldése a jel teljesítmény spektrumát is kiemeli a háttérzajéból.

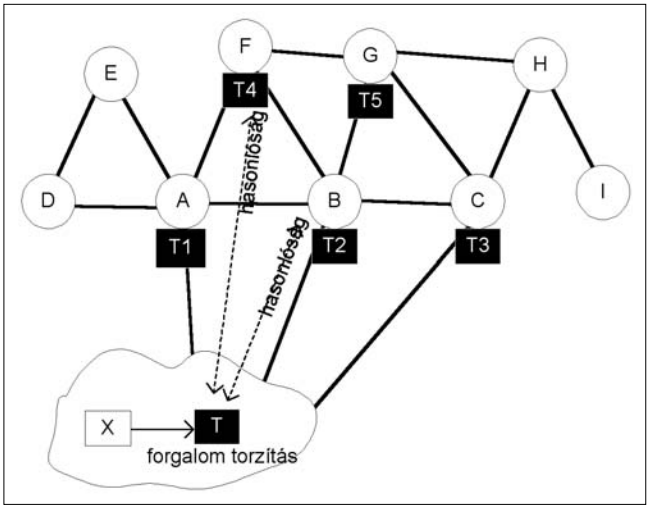
Mérőjelként szinuszos jelet használtunk, ennek egy vonalból áll a spektruma. Ezt a mérőjelet TCP alapú kommunikáció esetén úgy értük el, hogy egy közbeékelődő router segítségével módosítottuk a forgalom alakját, oly módon, hogy a kiküldött csomagok által lét-

2. ábra A Tor anonymizer hálózat



rehozott hálózati forgalom sávszélessége egy szinuszjelet közelítsen az időtartományban. Ezt úgy valósítottuk meg, hogy a TCP kommunikáció felfutási szakaszát (slow start) követően buffereltük a csomagokat. A buffernél a hálózat sávszélességét leszűkítettük úgy, hogy a bufferben összegyűljenek a csomagok. Ezután a bufferből az adott csomagméret és az ennek megfelelő szinuszos sávszélesség érték hányadosa által meghatározott időközönként olvastuk ki és küldtük el a csomagokat.

A támadás elvét a *3. ábra* segítségével mutatjuk be. A támadó célja, hogy egy adott felhasználó forgalmának végpontjait meghatározza, annak ellenére, hogy a felhasználó éppen ezt kívánta eltitkolni. Feltételezzük, hogy a felhasználó el akar érni egy meghatározott szervert (X). Az ebből a szerverből kiinduló forgalmakat a T támadó módosítja. Első lépésben a támadó a szervertől közvetlenül elérhető hálózatok forgalmának spektrumát figyeli, keresve bennük az általa módosított, szinuszos sávszélesség forgalom-mintáit (az ábrán ezek az *A, B, C* hálózatokban a T1,T2, T3 támadók). A támadó az első lépésben megfigyelt hálózatok közül az egyikben talál egyezést az általa keresett mintával (az ábrán a T2 a *B* hálózatban), ezért következő lépésként az innen közvetlenül elérhető hálózatokban hasonlítja össze a forgalmat *T* forgalmi alakjával (az ábrán az *F* és *G* hálózatokban T4,T5 által). A megfigyelést egészen addig folytatja, amíg az éppen megfigyelt hálózatból már nem érhető el közvetlenül egyetlen hálózat sem, és a keresett minta megtalálható a vizsgált hálózat forgalmának spektrumában. Az ábrán ez a hálózat az *F* jelű, vagyis a felhasználó forgalmának végpontja itt található.



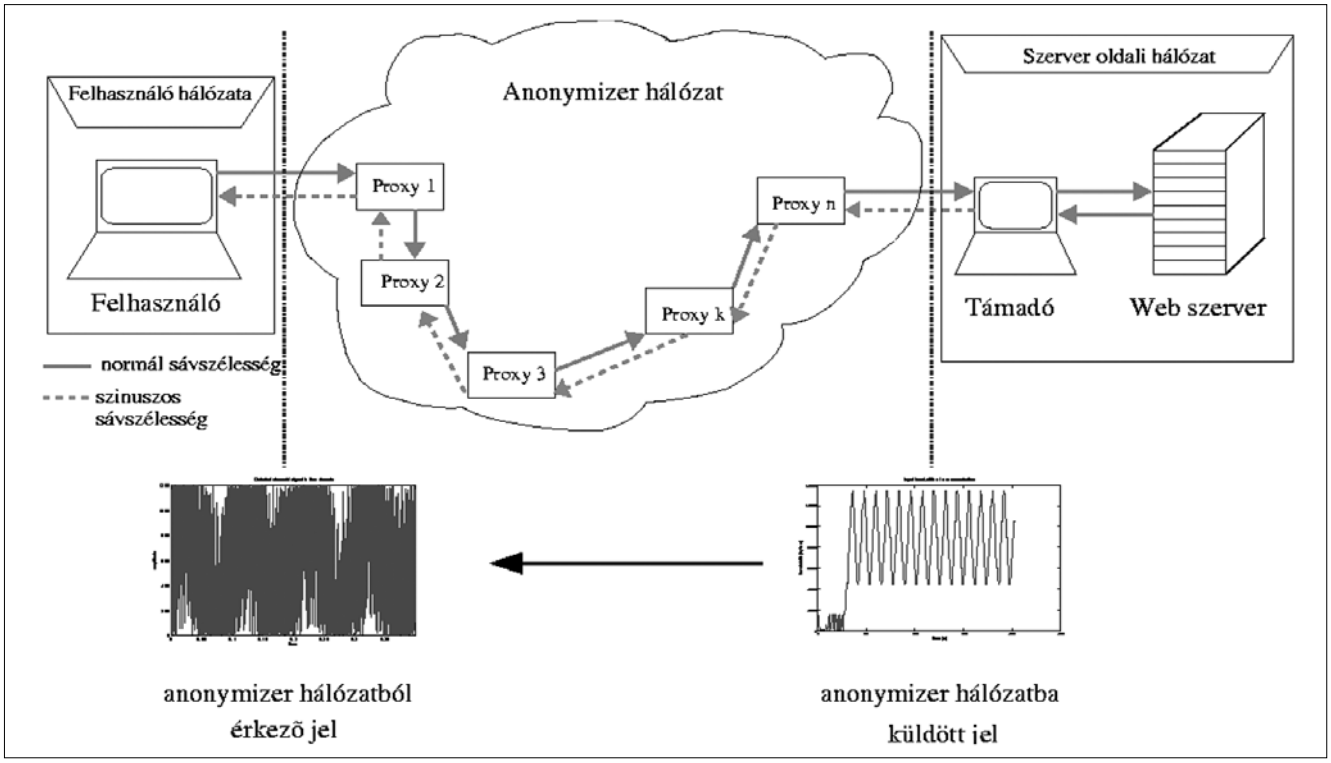
3. ábra A támadás elvének vázlata

3. A támadás megvalósítása

A támadást egy HTTP szerverről történő fájl letöltésén keresztül mutatjuk be. Abban a hálózatban torzítjuk a forgalmat, ahol a szerver elhelyezkedik, így a HTTP kérést küldő felhasználóhoz már a módosított sávszélességgel fog megérkezni a válasz *(4. ábra)*.

A támadás sikeres végrehajtásához az anonymizer hálózat mindkét végpontján, tehát a szerver és a felhasználó hálózatában is szükséges a forgalom vizsgálata. Ezután a két oldal forgalmának frekvenciatartománybeli alakját összehasonlítjuk, és elemezzük az eredményt. Amennyiben a két minta egyezik, akkor fény derült a felhasználó kilétére. Ha a minták nem egyez-

4. ábra A támadás megvalósítása



nek, akkor a felhasználó valószínűleg nem abban a helyi hálózatban tartózkodik, ezért más hálózatokat is vizsgálni kell, egészen addig, amíg valamelyikben a forgalom spektruma hasonlóságot nem mutat a szerver oldali forgalom spektrumával. Ezzel a módszerrel kideríthetjük, hogy merre haladnak a szervertől jövő csomagok, akkor is, ha azok egy anonymizer hálózaton is áthaladnak, mielőtt a felhasználóhoz érnének.

A szerver oldali hosztok megvalósításának blokkvázlata az *5. ábrán* látható. A hardver eszközök számának minimalizálása érdekében a Xen virtualizációs technológiát [7] alkalmaztuk, így emulálva egy hálózatot és hosztokat egyetlen fizikai számítógépen. A Xen technológia lehetővé teszi virtuális gépek, hosztok létrehozását egy számítógépen belül, ahol mindegyik virtuális gép a saját operációs rendszerét használja.

A mérések során a fizikai hoszt *(domain 0)* és a web szerver két különböző hálózatban volt, tehát egymással csak a router hoszton keresztül tudtak kommunikálni. Mindegyik hoszton Debian GNU/Linux operációs rendszer futott. A sávszélesség módosítást a router hoszt végezte, az NTMF (Network Traffic Manipulation Framework) [8] hálózati forgalommódosító keretrendszer alkalmazásával.

Az NTMF C++ nyelven íródott, alapvetően hálózati protokollok tesztelésére szánt keretrendszer. Alacsony szintű hozzáférést enged a linux kernel által továbbított csomagokhoz, moduláris jellege miatt kiválóan alkalmas továbbfejlesztésre, ezért remekül meg tudtuk valósítani vele a kimenő forgalom sávszélesség módosítását.

A router hoszt változtatás nélkül továbbította a szerver felé haladó forgalmat, vagyis a router hoszton *eth0*-tól *eth1*-ig nem történt sávszélesség módosítás (az 5. ábrán, a router hoszton belül folytonos vonallal jelölve). A másik irányban viszont a csomagok kibocsátási ideje módosítva lett, oly módon, hogy a sávszélesség időben szinuszosan változzon (szaggatott vonallal jelölve).

A sávszélesség-idő függvény a következő alakú volt:

$$A + C \cdot \sin((2\pi/N) \cdot f \cdot n),$$

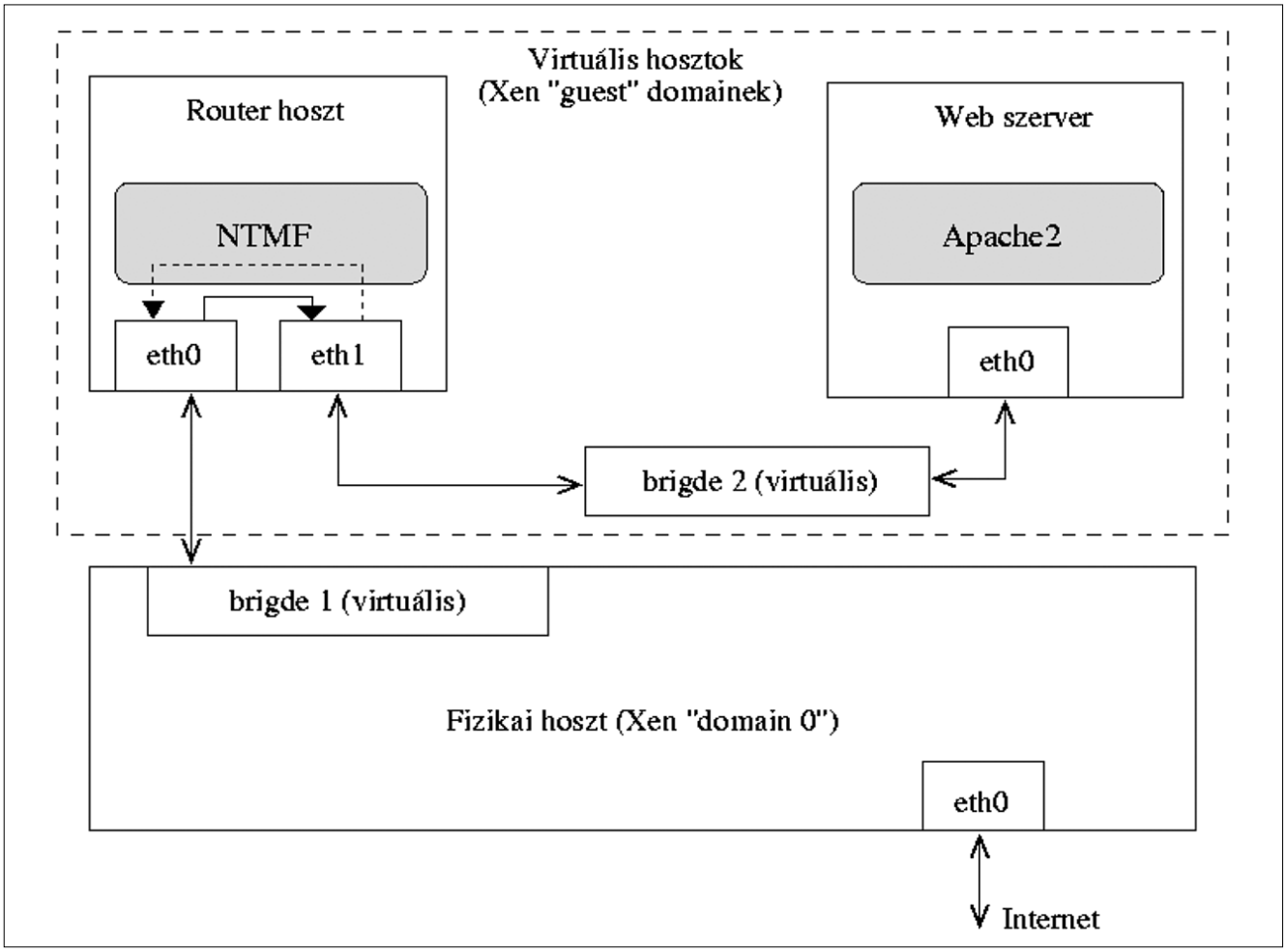
ahol

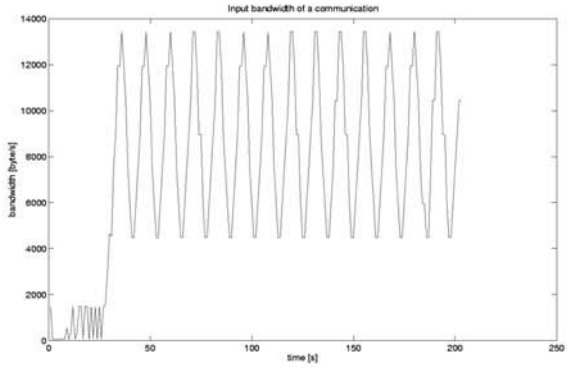
A a sávszélesség DC komponense,
C a szinusz amplitudója (Kbyte/s),
$1/N$ az időkvantum,
f a szinuszos jel frekvenciája (Hz) és
n/N az előző csomag kiküldése óta eltelt idő (sec).

4. Eredmények

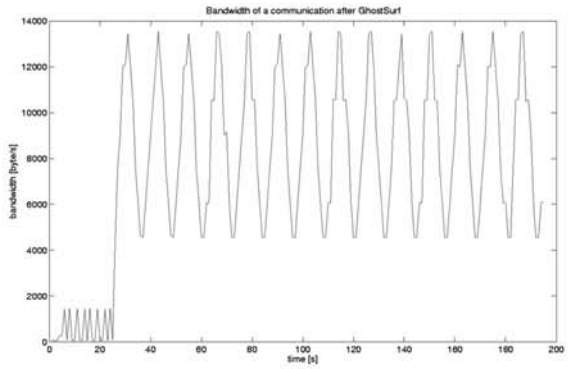
Mindkét anonymizer hálózat esetében a Windows platformra írt kliens szoftvereket teszteltük. A kliens és szerver egymástól fizikailag is távol helyezkedett el, a köztük lévő forgalom az anonymizer hálózaton keresztül haladt.

5. ábra A szerver oldali hosztok megvalósításának blokkvázlata

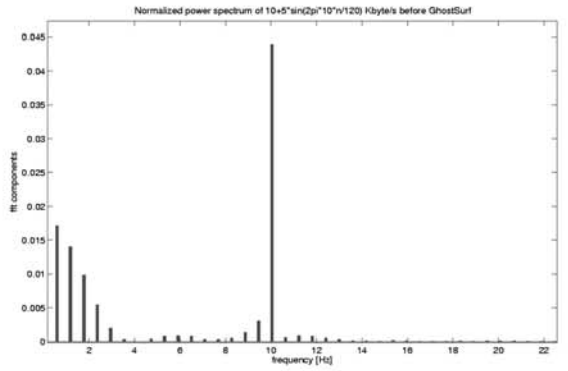




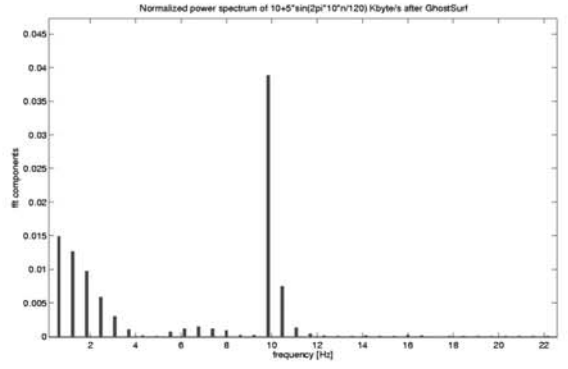
A bemeneti sávszélesség



A sávszélesség a GhostSurf hálózat után

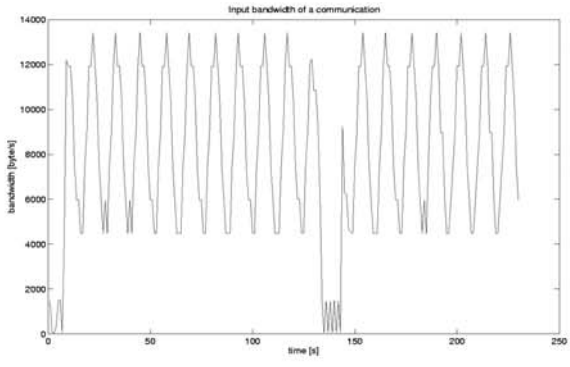


A bemeneti sávszélesség spektruma

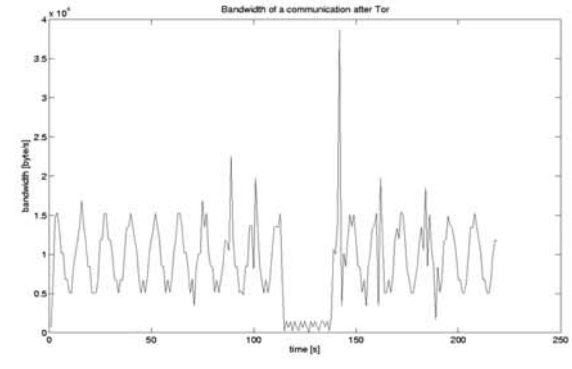


A sávszélesség spektruma a GhostSurf hálózat után

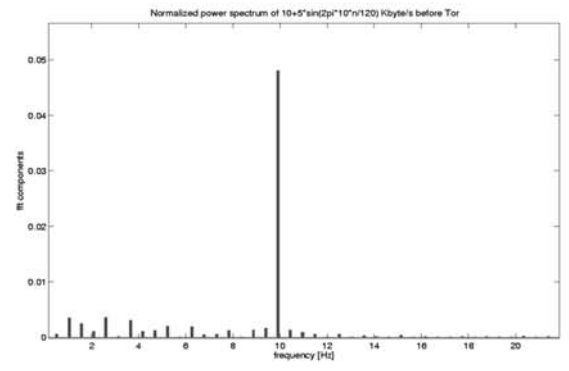
6. ábra



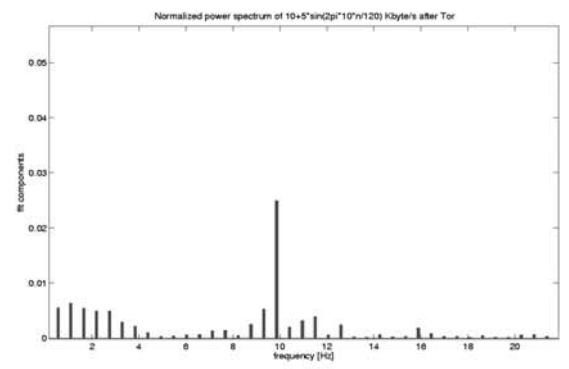
A bemeneti sávszélesség



A sávszélesség a Tor hálózat után



A bemeneti sávszélesség spektruma



A sávszélesség spektruma a Tor hálózat után

7. ábra

A mérések eredményeit bemutató ábrákon az anonymizer hálózat bemeneti sávszélesség-idő függvénye, ennek spektruma a frekvenciatartományban, az anonymizer hálózat utáni ismeretlen kimeneti forgalom sávszélesség-idő függvénye és ennek spektruma a frekvenciatartományban látható.

A bufferből a csomagokat mindkét vizsgálat során

$$8.5 + 4.5 \cdot \sin((2\pi/120) \cdot 10\ [Hz] \cdot n)\ \text{[Kbyte/s]}$$

változó sávszélességgel engedtük ki.

Tehát a spektrumokban 10 Hz-nél csúcs várható.

4.1. GhostSurf 2005 Platinum

A GhostSurf anonymizer hálózatban végzett vizsgálat eredményei a *6. ábrán* találhatók.

A mérés alapján megállapítható, hogy a bemeneti forgalom spektruma jól illeszkedik a GhostSurf hálózat utáni ismeretlen forgalom spektrumához. Ez azt jelenti, hogy a GhostSurf anonymizer hálózat csak csekély mértékben módosította a forgalom sávszélességét, és megmaradt annak szinuszos jellege.

A támadás tehát sikeresnek bizonyult, a felhasználó azonossága nem maradt rejtve.

4.2. Tor

A *7. ábrán* a Tor anonymizer hálózatban mért eredményeket ismertetjük.

A mérés kimeneti forgalom-mintájának spektrumában az eredeti minta 10 Hz-es komponense a legnagyobb energiájú, a bemeneti forgalom spektruma tehát jól illeszkedik a Tor hálózat utáni forgalom spektrumára.

Ez azt jelenti, hogy a támadás ez esetben is sikeres volt, még a Tor hálózat sem védi felhasználóit az ilyen típusú támadások ellen.

5. Összefoglalás

Cikkünkben egy olyan forgalom analízisen alapuló támadási módszer került bemutatásra, amely a TCP protokollra épülve valósítja meg az anonymizer hálózatok felhasználóinak nyomkövetését. A mérések alapján megállapítható, hogy ezek a támadási módszerek követhetővé teszik a célpontokat, vagyis a bemutatott anonymizer hálózatokban a felhasználók anonimitása nem megfelelően biztosított.

Ez azt jelenti, hogy Internet szolgáltatók, vagy a felett álló kormányzati szervek használhatnak ilyen eljárásokat, hogy az anonymizer hálózatok mögé bújó személyek kilétére fényt derítsenek. A hálózati forgalmak szisztematikus megfigyelésével, és a forgalmakban egy előre definiált minta keresésével meghatározható egy adott felhasználó forgalmának mindkét végpontja, ezzel a felhasználó anonimnak vélt kommunikációja nem marad rejtve.

Irodalom

[1] Steganos Internet Anonym Pro 7, http://www.steganos.com
[2] Bypass Proxy Client, http://www.bypass.cc
[3] Anonymizer 2004, http://www.anonymizer.com
[4] MorphMix, http://www.tik.ee.ethz.ch/~morphmix
[5] GhostSurf 2005 Platinum, http://www.tenebril.com
[6] A Tor anonymizer hálózat, http://tor.eff.org
[7] The Xen virtual machine monitor, http://www.cl.cam.ac.uk/Research/SRG/netos/xen
[8] Real-time Network Traffic Manipulation Framework for Protocol Testing, http://ntmf.sourceforge.net

Másolásvédelem szoftver vízjelezés és obfuszkálás segítségével

EBERHARDT GERGELY, NAGY ZOLTÁN
SEARCH-LAB Kft., {gergely.eberhardt, zoltan.nagy}@search-lab.hu
JEGES ERNŐ, HORNÁK ZOLTÁN
BME Méréstechnikai és Információs Rendszerek Tanszék, SEARCH Laboratórium
{jeges, hornak}@mit.bme.hu

Lektorált

Kulcsszavak: szoftver vízjel, obfuszkálás, kódvisszafejtés, megbízható operációs rendszer, mobiltelefonos szoftverek

A szerzői jogok megsértését a szoftverfejlesztő cégek manapság elsősorban jogi úton próbálják orvosolni, mivel a jelenleg létező technológiák nem teszik lehetővé a szoftverek illegális használatának és terjesztésének a megakadályozását. Mindezidáig általános tapasztalat volt, hogy szinte minden piacra dobott másolásvédelmi megoldást rövid időn belül feltörtek. A másolásvédelemi megoldások hiányossága eredményezte néhány a közelmúltban szárnyra kapott üzleti modell bukását is, ami bizonyos értelemben a „dotcom" világ válságához is hozzájárult. Cikkünkben egy olyan megoldást mutatunk be, amely az obfuszkálás és a vízjelezés technikáinak ötvözésével nyújt megfelelő erősségű védelmet a kereskedelmi forgalomba került szoftverek másolása ellen. Az általunk javasolt megoldás elsősorban mobiltelefonon futó alkalmazások – tipikusan játékprogramok – védelmére használható, mivel ezen eszközök esetében a védelmi megoldást az operációs rendszer és a hardver által szavatolt biztonságos működésre alapozhatjuk.

1. Bevezetés

A Business Software Alliance (BSA) adatai szerint az illegális szoftverhasználatból eredő kár éves szinten 30 milliárd dollárra volt becsülhető világviszonylatban 2004-ben [3]. Az Európai Uniót tekintve a használt szoftvereknek csak közel a fele legális, és ezen a helyzeten sem technikai sem jogi úton eddig nem sikerült javítani. Sajnos a köztudatban is az a nézet uralkodik, hogy a szoftverkalózkodást szinte lehetetlen megakadályozni. A mobiltelefonok világában azonban, ahol, tekintettel arra, hogy az azokban futó operációs rendszer sértetlenségében meg lehet bízni, még van arra lehetőség, hogy az illegális szoftvermásolásból eredő máshol tapasztalt károk elkerülhetőek, vagy legalább csökkenthetőek legyenek.

Meggyőződésünk szerint a mobil szoftverek piacán a további növekedés legfontosabb előfeltétele egy erős másolásvédelemi technika megléte, ezért kutatásfejlesztési projektünk ezen szoftverek másolásvédelmét célozta meg. Ezen a területen azonban némileg más peremfeltételekkel kell számolnunk, mint az általában tárgyalt személyi számítógépek esetében.

A megbízható operációs rendszer *(trusted OS)* elengedhetetlen alapja egy megfelelően erős másolásvédelemnek, hiszen amennyiben az operációs rendszer maga is megváltoztatható, akkor bármilyen rá épülő szoftver alapú védelem elvi problémák miatt feltörhető. Elképzelhetőek olyan megoldások is, ahol a fejlesztők egyáltalán nem bíznak meg az operációs rendszerben. Ezen megoldások zöme a ködösítés elvét *(security by obscurity)* kihasználó biztonságra alapoz, azaz egyszerűen elrejti a kód azon részeit, amelyek a szoftver sértetlenségét és érvényességét ellenőrzik. Ezen megoldások kifejlesztői általában azt feltételezik, hogy az elrejtett ellenőrzésnek a visszafejtése vagy a védelem megkerülése kellően sok időt vesz igénybe ahhoz, hogy ne befolyásolja számottevően a bevételek mértékét. Mindannak ellenére, hogy egy ilyen módszer valóban megnöveli a védelem feltöréséhez és megkerüléséhez szükséges időt, a tapasztalatok azt mutatják, hogy a titokban tartott módszeren alapuló védelmeket minden esetben előbb-utóbb feltörték.

A fentiekkel szemben az általunk javasolt szoftver másolásvédelmi megoldás ötvözi a Nyilvános Kulcsú Infrastruktúrát (PKI), az obfuszkálási módszereket valamint a szoftver vízjelezést, miközben egy megbízható, sértetlenségét biztosító operációs rendszert feltételezünk. A pusztán licenszeket és digitális aláírást alkalmazó módszerekkel szemben a sémánk legfontosabb előnye az, hogy egyaránt megengedi szabadon terjeszthető és másolásvédett szoftverek futtatását is.

2. Elméleti háttér

A szoftver másolásvédelemi megoldásoknak két fő kategóriája létezik: az önálló rendszerek, valamint azok, amelyek külső eszközöket használnak működésük során [10].

Az önálló rendszerek védelmét önmagába a szoftverbe építik bele, így annak biztonsága csak a felhasznált programozási technikáktól függ. Ilyen technikák lehetnek különböző integritásvédelmek, program obfuszkálás, ellenőrző összegek, titkosítás, a kódvisszafejtés a megfigyelt futtatás (debug) megakadályozása, illetve egyéb, a cracker-ek feladatát megnehezítő megoldások [8]. Ezen módszerek alapja szükségképpen az, hogy a program *önmagát* ellenőrizze. Mivel az ellenőrzés a program részét képezi, ezért annak visszafejtésével az ellenőrzést végző programrészletek lokalizálhatóak és a kód módosításával a védelem kiiktatható

[10]. Ezek a technikák tehát elméleti szempontból és a tapasztalatok alapján gyakorlati szempontból sem tekinthetőek elegendően biztonságosnak [2].

A védelmi mechanizmusok másik kategóriáját a külső segítséget igénybe vevő megoldások alkotják. Ezek a másolásvédett programok általában valamilyen megbízható processzort, operációs rendszert, vagy más biztonságos hardver, esetleg szoftver megoldást használnak. A külső támogatás lehet on-line vagy off-line [10]. Az on-line támogatás esetében néhány ellenőrző függvény olyan távoli számítógépen fut le, amelyekhez a támadónak nincs hozzáférése. Ezzel ellentétben az off-line együttműködés során a védelem valamilyen biztonságos hardvert vagy szoftvert igényel. A biztonságos hardver általában egy smart card használatát jelenti, míg a megbízható szoftver komponenst általában egy sérthetetlen operációs rendszer biztosítja.

A jogosult felhasználók hozzákapcsolását a program egy példányához általában Nyílt Kulcsú Infrastruktúrán (Public Key Infrastructure, PKI) [9] alapuló digitálisan aláírt licenszek használatával valósítják meg. Másolásvédett szoftverek esetében ilyenkor a programhoz csatolni kell egy úgynevezett licensz fájlt, ami információkat tartalmaz a felhasználóról, a gyártóról, a forgalmazóról és magáról a termékről (például az adott szoftver hash kódja). Mivel a licensz sértetlenséget digitális aláírás védi, az operációs rendszer már biztonsággal ellenőrizheti a jogosultságot, és a megfelelő licensz nélkül az alkalmazás futtatását megtagadhatja.

Egy programvédelmi megoldásnak támogatnia kell a különböző üzleti modelleket, a védett alkalmazás felhasználásának különböző eseteit, valamint a kényelmes szoftverfejlesztést is, de – ami a legnagyobb technikai kihívást jelenti – az ezzel felvértezett operációs rendszernek futtatnia kell tudni mind a másolásvédett, mind pedig a szabad terjesztésű programokat. Elengedhetetlen feltétel tehát, hogy az operációs rendszer minden olyan esetben felismerje, hogy a program eredetileg védett volt, amikor a szoftver egy részlete megváltozott, vagy ha a licensz fájlt eltávolították mellőle.

A hagyományos hang, kép, illetve video fájlok védelmi megoldásaival ellentétben, amelyek a vízjelet a tartalom eredetének a megállapítására használják, a szoftver esetében annak a jelzésére is használhatjuk a vízjelet, hogy a program másolásvédett-e vagy sem. A média fájlokhoz képest sokkal könnyebben megoldható az, hogy a program kódját úgy módosítsuk, hogy közben a felhasználó által tapasztalt működés ne változzon érezhető módon.

A *szoftver vízjelek* két típusát különböztethetjük meg: a statikust és a dinamikust. A statikus vízjelek esetében az információt a végrehajtható állomány hordozza. Az ilyen vízjeleket tipikusan az inicializált *data, code és text* szakaszokban szokták elhelyezni [1,7,8,11,13,14]. Ezzel ellentétben a dinamikus vízjelek nem a végrehajtható program kódjából, hanem a program végrehajtási állapotából nyerhetők ki. Ez azt jelenti, hogy a programot le kell futtatni, hogy a programállapot valamely tulajdonsága jelezze a vízjel jelenlétét [5,6,12].

A vízjel egyszerű eltávolíthatóságának a megakadályozására *szoftver obfuszkálást* alkalmazhatunk. Ez a módszer különböző kód-transzformációs eljárások gyűjtőneve, amelyeket azzal a közös céllal hajtunk végre, hogy a program visszafejtését és megértését nehezebbé tegyük. Az obfuszkálás mind az automatikus eszközökkel történő, mind az emberi megértés által végzett visszafejtési támadást számottevően nehezíti [4,15].

3. Követelmények és minőségi célok

A továbbiakban meghatározzuk a másolásvédelmi megoldásunkhoz szükséges követelményeket és a minőségi célok eléréséhez szükséges további feltételeket.

Megbízható operációs rendszer – sértetlenség és bizalmasság

Egy megbízható operációs rendszer esetében feltételezhetjük a futó folyamatok *sértetlenségét*, esetünkben azonban nem feltételezzük az operációs rendszer, vagy a rajta keresztül áramló információk bizalmasságát. Ez azzal a feltétellel függ össze, miszerint a vízjel ellenőrző módszert nem tarthatjuk titokban, tehát a támadó azt megismerheti, ennek ellenére azonban a vízjel eltávolításának a védett programból nehéz feladatnak kell lennie.

A fentieken túl az általunk használt megbízható operációs rendszernek képesnek kell lennie arra, hogy az alkalmazáson lévő digitális aláírást ellenőrizze, és támogatnia kell a PKI különböző egyéb elemeit is, mint például a tanúsítvány lánc kezelését vagy a tanúsítvány (kulcspár) visszavonását.

Azonos megfigyelhető működés

Esetünkben a *megfigyelő* a program felhasználója, akinek a szemszögéből nézve a transzformált programnak funkcionálisan ugyanúgy kell működnie, mint az eredetinek. Az alkalmazásnak például ugyanazokat az ablakokat kell tartalmaznia, ugyanazokat a fájlokat és kapcsolatokat kell fenntartania a külvilággal, és mindezeknek ugyanolyan módon kell működniük. A sebességnek, a memória használatnak, a program végrehajtása közbeni belső állapotoknak, valamint a program kódjának azonban kis, vagy akár nagyobb mértékű változása is megengedett.

Nehezebb visszafejthetőség

Az obfuszkálási módszerek alkalmazásával a programnak egyrészt az *automatikus* visszafordítását kell tudnunk meggátolni, illetve megnehezíteni, másrészt pedig a transzformációk hatására a kód *emberi megértését* is nehezíteni kell [8]. Célunk tehát az, hogy a visszafejtést, és így a védelem megszüntetését annyira időigényessé tegyük, hogy már ne érje meg a hasonló törések végrehajtásához szükséges időráfordítást.

A *nehezítés*, illetve a nehéz esetünkben azt jelenti, hogy egy védett program visszafordítása olyan komplex feladat megoldását kívánja meg, amelynek nehézsége visszavezethető a kriptográfiában elfogadottan nehéznek tartott problémára, tipikusan legyen azzal egyenértékű, mintha egy kriptográfiai algoritmust kellene feltörni.

Vízjel nehezebb eltüntethetősége

Ez a minőségi cél szorosan összefügg az előzővel, hiszen az erős obfuszkálási megoldások használata nem csak a visszafejtést nehezíti, de a vízjel könnyű eltávolíthatóságát is megakadályozza.

Mivel esetünkben a megbízható operációs rendszernél csak a sértetlenséget feltételezzük, de a megbízhatóságot nem, ezért feltételeznünk kell azt is, hogy a vízjel felismerő algoritmust sem lehet titokban tartani. Ebből adódóan a vízjel eltávolításának még abban az esetben is kellően nehéznek kell lennie, ha a vízjel detektáló algoritmus nyilvánosan ismert. A vízjelet ezért az eredeti kódba olyan mélyen kell elhelyezni, hogy azt ne lehessen eltávolítani a *teljes* kód megértése nélkül. A teljes megértés követelménye azt jelenti, hogy mindennek mindennel össze kell függnie ahhoz, hogy a támadó ne legyen képes dekomponálni a megértés feladatát.

A transzformációk költségeinek skálázhatósága

Mivel mindegyik transzformációnak van valamekkora költsége, amely általában az alkalmazás végrehajtási idejére és memóriahasználatára vonatkozik, ezért fontos, hogy a védelem ebből a szempontból is hatékony legyen. A különböző követelmények, és a kód különböző területein felmerülő transzformációs költségek miatt a megvalósított transzformációknak tehát *skálázhatóaknak* kell lenniük, és a rendszernek elsősorban a transzformált program futási sebessége és a memória használat tekintetében kell támogatnia a megfelelő munkapont beállítását.

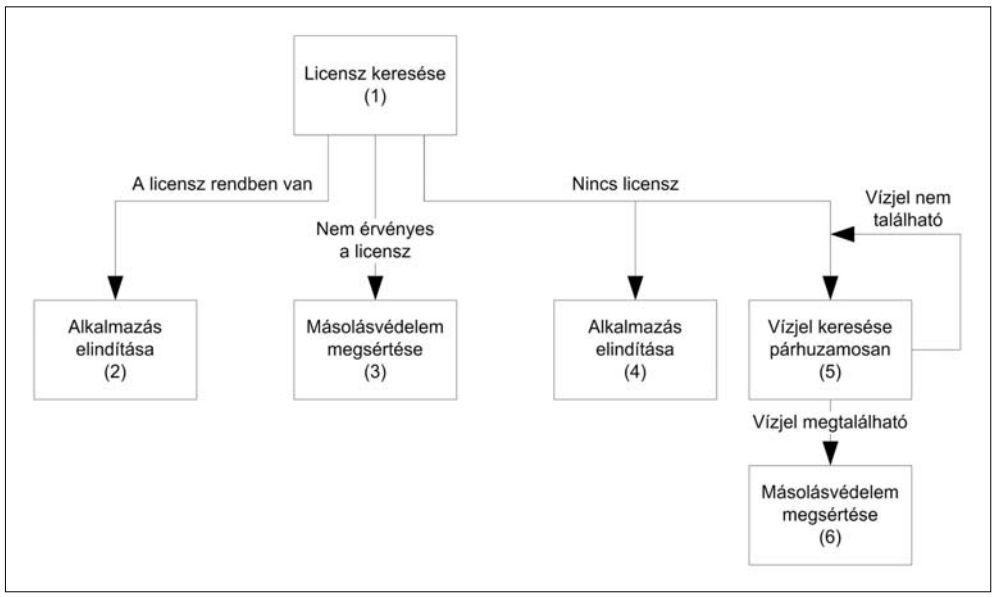
4. A javasolt másolásvédelmi megoldás

A másolásvédelmi megoldásunk tehát a fentebb ismertetett követelmények figyelembevételével, az említett építőkövek felhasználásával valósul meg. A módszer lényege, hogy a rendszer egy digitálisan aláírt licensz segítségével ellenőrzi a védett program sértetlenségét, és amennyiben a digitális aláírása vagy a licensz nem érvényes, megakadályozza a futtatást. Abban az esetben, ha nincs licensz csatolva egy alkalmazáshoz, akkor a rendszer a program futása alatt folyamatosan keresi a dinamikus vízjel jelenlétének a nyomait, amely jelzi az operációs rendszer számára, hogy másolásvédett szoftverről van szó, tehát a licensz fájl megléte kötelező lenne. A vízjel eltávolítása pedig megfelelő obfuszkálási technikák alkalmazása miatt nehéz, ezért a támadó nem tudja olyan módon változtatni, feltörni a védett alkalmazást, hogy az a licensz nélkül is működőképes legyen.

A másolásvédelmi megoldás operációs rendszeren belüli ellenőrző része az *1. ábrán* látható algoritmussal írható le.

1. Digitálisan aláírt licensz keresése a programhoz.
2. Ha van ilyen licensz, és mind a licensz (például az abban található hash), mind pedig a digitális aláírás megfelelő, akkor az alkalmazás minden további vízjel-ellenőrzés nélkül futtatható.
3. Ha programhoz kapcsolódó licensz vagy annak aláírása nem megfelelő, a végrehajtás azonnal leáll, valamint a rendszer megteheti az egyéb szükséges lépéseket (például naplózás vagy jelentés készítése), mivel a másolásvédelem, vagy az alkalmazás integritása sérült.
4. Amennyiben a programhoz nincs licensz mellékelve, akkor az lehet egy szabad felhasználású, vagy egy védett, de manipulált program is, ezért a program elindul.
5. Ezzel párhuzamosan az operációs rendszernek el kell kezdenie a vízjel folyamatos keresését.
6. Ha a vízjel megtalálható a programban, akkor a végrehajtást azonnal fel kell függesztenie, és ismételten a szükséges lépéseket kell megtennie, hiszen a vízjel jelenléte azt jelzi, hogy a program másolásvédett, így érvényes licensszel kellene rendelkeznie, és enélkül illegális másolatnak számít.

1. ábra Licensz ellenőrzés algoritmusa



A védelmi sémánkban a vízjelet a program másolásvédettségének jelzésére használjuk, ezért a vízjelnek a következő tulajdonságokkal kell rendelkeznie:

– csak egy „bitnyi” információ tárolására kell alkalmasnak lennie, amelynek a jelentése az, hogy a program másolásvédett,
– nem használhat titkos módszert a vízjel detektálására, mivel a támadónak lehetősége van arra, hogy megismerje a vízjelet felismerő algoritmust,
– ne lehessen felismerni az összes vízjelet a programban statikus elemzéssel, még abban az esetben se, ha a támadó ismeri a felismerő algoritmust.

Ezen követelmények mindegyikének a kielégítéséhez a legjobban a dinamikus vízjel felel meg, mivel esetében a vízjel bináris reprezentációja a program végrehajtása során jelenik csak meg. A dinamikus vízjel detektálásához azonban a programot egy ideig futtatni kell, amiből az következik, hogy a program állapotát folyamatosan kell figyelni a végrehajtás alatt. Emiatt a vízjel keresés ugyan lelassíthatja az alkalmazások végrehajtását, de csak abban az esetben, ha nincs licensz fájl mellékelve a programhoz. A fejlesztőknek tehát különösen érdekük licenszet biztosítani a termékeikhez még akkor is, ha az szabad felhasználású, mivel csak ily módon tudják az eszköz teljesítményét teljesen kihasználni.

A program és az operációs rendszer közötti kapcsolatot a vízjel felismerése közben a *2. ábra* illusztrálja.

Az általunk használt dinamikus vízjel csak egy bit információt tárol egy speciális szám formájában, ami egy véletlen számból és ennek a véletlen számnak a transzformált értékéből áll elő úgy, hogy a két értéket egyszerűen egymás mögé írjuk. Az *f* függvénnyel végrehajtott transzformáció lehet digitális aláírás, hash érték, CRC vagy egyéb, például egy egyszerű konstanssal való XOR művelet is; szerepe abban van, hogy egy nem védett programban a két egymást követő érték előfordulási valószínűsége elegendően kicsi legyen, így a védett programban való előfordulása egyértelműen jelezze a vízjel meglétét. A vízjelet tehát a következő módon definiálhatjuk:

$$WM = (RND; f(RND))$$

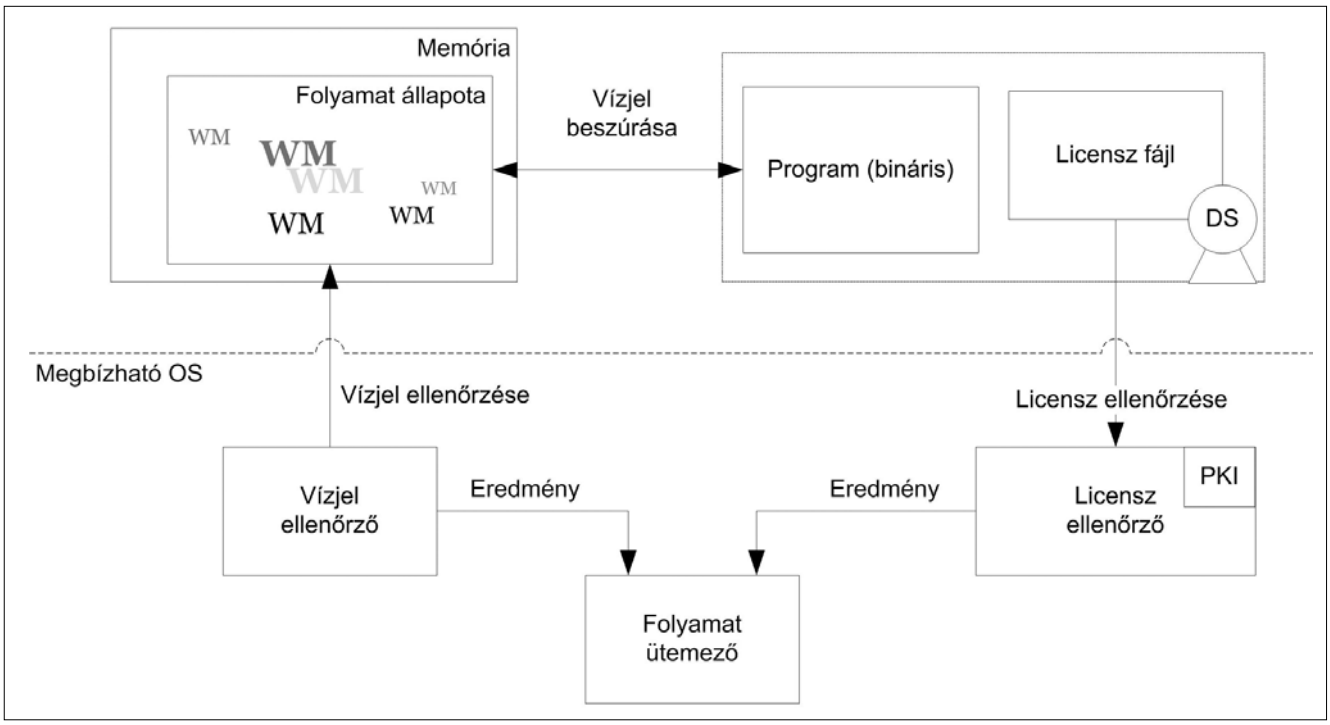
A véletlen számtól függő különböző *WM* vízjel értékpárokat tehát olyan sokszor kell elrejteni az alkalmazásban – azaz a védett programnak a futása során annyiszor kell előállítania ilyen érték párost – amennyiszer csak lehetséges. Ezen cél elérése érdekében többféle, elsősorban az adat obfuszkálással összekapcsolható technikát alkalmazhatunk; egy ciklusváltozót például olyan módon transzformálhatunk, hogy annak egy bizonyos értékére, amelyet egyszer biztosan felvesz, egy a fenti módon definiált *WM* szám álljon elő.

Vegyük észre, hogy sem az operációs rendszer, sem a támadó nem tudja a vízjel összes lehetséges értékét, csak felismerni tudja azt, amikor a megfelelő bemenetek hatására előáll a program állapotában. Így ha a támadó az összes vízjelet el akarja tüntetni, akkor végre kell hajtania a program összes elágazását az összes lehetséges bementettel, hogy megbizonyosodjon róla, hogy az eltávolítás maradéktalanul sikeres.

5. A rendszer felépítése

A következő, *3. ábrán* látható a másolásvédelmi megoldásunk keretrendszere, annak főbb elemei (moduljai) és azok kapcsolata. Az obfuszkálási és vízjel elhelyező

2. ábra A program és az operációs rendszer közötti kapcsolat a vízjel felismerése közben



transzformációk a szokványos C/C++ fordítási folyamatba integrálhatóak. A rendszer bemenetét a védendő alkalmazás C/C++ forrás fájljai képezik. A forráskódban elhelyezett direktívák szabályozhatják az obfuszkálás és a vízjel elhelyezés folyamatát azáltal, hogy behatárolják az egyes kódrészletek transzformálása során megengedett költségeket (futásidő, memória használat). Ezen direktívákat az előfeldolgozás során összegyűjtjük, majd a fordító az eredeti forrásból előállítja az assemblyre lefordított kódot tartalmazó kimeneti fájlt (LST) és az egyéb debug információkat tartalmazó fájlokat.

Az összegyűjtött információk egy fordító-független absztrakt reprezentáció *(Virtual Machine Code, VMC)* előállításának az alapjául szolgálnak, amin az obfuszkáló és vízjelező transzformációk végrehajtása történik. Az absztrakt reprezentáció létrehozásához azonban nem csak a fentebb említett forrásfájlok használhatók, hanem esetleg valamely dissasembler eszközzel nyert LST fájl, a Map fájl vagy akár a különböző Profile információk is.

A direktívák összegyűjtése és az előkészítés után a rendszer elemzi az LST fájlban és a többi forrásként használt fájlban található kódot, illetve egyéb információt a kóddal kapcsolatban. Ezen elemzés részét képezi a vezérlési folyam és az adatfüggőségi gráfok megállapítása is, melyek létrehozásával teljessé válik a program belső, platform független absztrakt reprezentációja. Ez a belső reprezentáció, amelyet *Code Model*-nek nevezünk, tartalmazza az összes olyan információt, amelyre szükség van a különböző transzformációk megtervezéséhez és végrehajtásához.

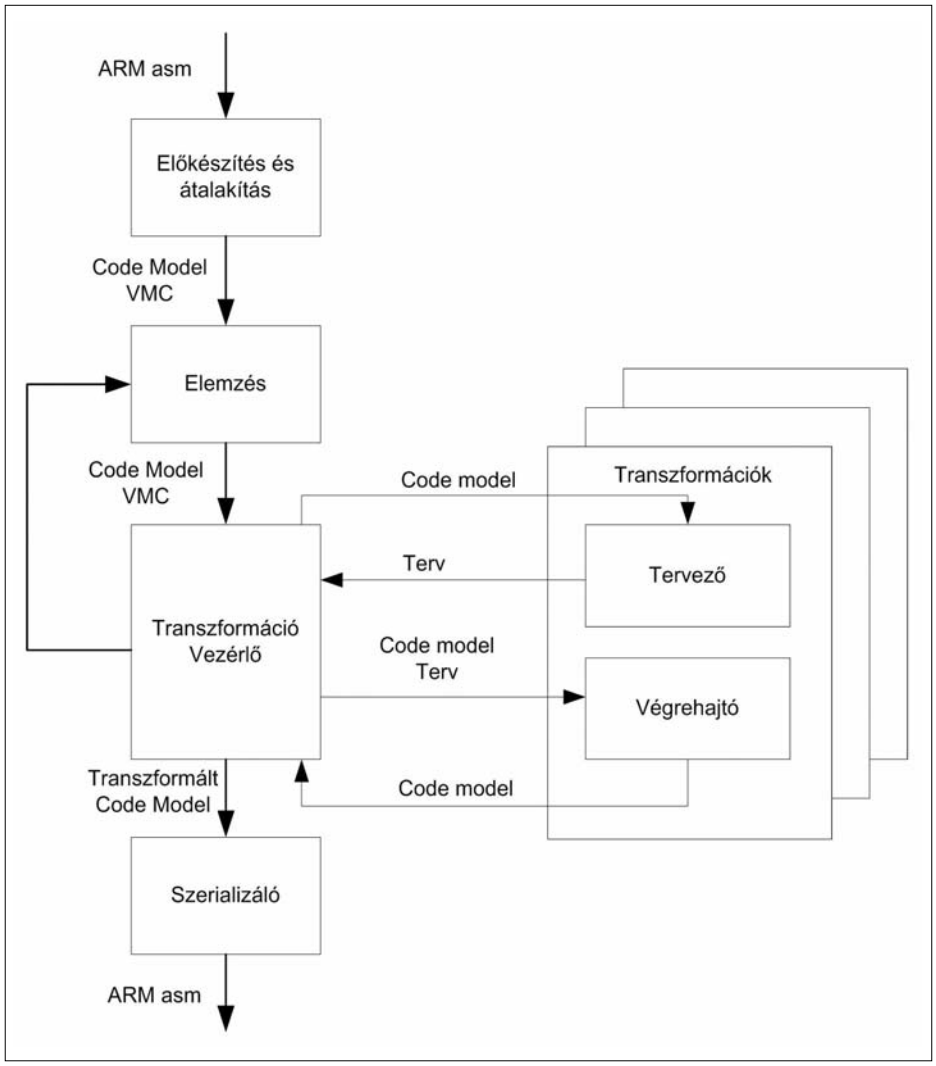
A transzformációk végrehajtása több lépésben történik. A *Transzformáció Vezérlő* minden lépés előtt egy részletes tervet készít az elvégzendő műveletekről és azok sorrendjéről. Minden transzformációs lépést követően a belső absztrakt reprezentációnak konzisztensnek kell maradnia, ami azt jelenti, hogy minden iterációs lépésnél a programnak funkcionálisan azonosnak kell lennie az eredeti transzformálatlan kóddal.

A hatékonyság növelés és a funkcionális azonosság biztosítása érdekében minden transzformációs lépés két fázisban zajlik. Az első fázisban tervek készülnek a lehetséges végrehajtási módokról és a végrehajtás esetleges paramétereiről, továbbá az egyes lépések végrehajtásának hatásairól.

Az egyes transzformációk *Tervezője* által felajánlott paraméterek alapján elkészül a terv, amely tartalmazza a kiválasztásra került transzformációk sorrendjét és paraméterezését, majd a második fázisban a transzformációk *Végrehajtója* végrehajtja azokat. Az egyes transzformációkhoz tartozó *Tervező* feladata az első fázisban a paraméterek meghatározása, a *Végrehajtó* pedig elvégzi a tényleges transzformációt és biztosítja a funkcionális azonosságot. Így a helyes működés bizonyításához elég csak a *Végrehajtó* eljárások helyességét bizonyítani.

A transzformációs lépések addig követik egymást, amíg a meghatározott minőségi célok meg nem valósulnak, azaz elegendő vízjel nem kerül elhelyezésre és a kód bonyolultsága el nem éri a kívánt szintet. A transzformációs lépések sorozata után az absztrakt reprezentáció szerializálásával ismét előáll a közvetlenül futtathatóvá fordítható assembly kód. A teljes folyamat eredménye tehát egy lefordított tárgy kód, amely egyrészt obfuszkált, másrészt tartalmazza a vízjelet is.

3. ábra A rendszer moduljai és köztük lévő kapcsolatok



6. Eredmények

Az általunk tervezett másolásvédelmi rendszer értékeléséhez implementáltuk a keretrendszert, illetve megvalósítottunk számos transzformációt. A transzformációk közül egy egyszerű példával, a rendszerhívások elrejtését megvalósító obfuszkálási technika gyakorlatba ültetésével illusztráljuk az általunk javasolt séma és a megvalósított keretrendszer lehetőségeit.

A legtöbb program intenzíven használja a különböző szabványos programozói könyvtárakat, illetve az API-kat az operációs rendszer szolgáltatásainak eléréséhez. Ezek a függvényhívások jól dokumentáltak és a legtöbb programozó által jól ismertek, így nagy segítséget jelentenek a program megértésében és visszafejtésében. Ennek elkerülésére alkalmazható a rendszerhívások elrejtését megvalósító obfuszkálási technika, amely eltünteti a hasonló nyomokat a programból.

Különböző eljárások léteznek az ilyen ismert függvények hívásainak elrejtésére. Az alapvető ötlet ezek mögött az, hogy az eredeti rendszerhívás lecserélésre kerül egy belső úgynevezett fedő *(wrapper)* függvény meghívásával, amely tovább hívja az eredeti függvényt. Az obfuszkáló tetszőleges számú ilyen interfész függvényt hozhat létre, de akár elhelyezheti valamennyi API hívást egyetlen ilyen interfész függvényben is. Ilyenkor általában egy változó értéke dönti el, hogy az interfész függvénynek ténylegesen melyik API függvényt kell meghívnia.

Ezen obfuszkálási transzformáció esetében a *Tervező* meglehetősen egyszerű, mivel csak az ismert függvények hívási helyeit kell megkeresnie, majd ezekből kiválasztania az elrejtendőket (akár mindet). Így az elkészült terv ezen hívások listáját fogja majd tartalmazni.

A hívások elrejtésének algoritmusa a terv ismeretében a következő:

1. Egy új függvény létrehozása, ami interfész függvényként fog szolgálni az API hívásokhoz.
2. Az API hívásokhoz azonosítók rendelése, és az interfész függvény feltöltése a megfelelő blokkokkal és utasításokkal a lehetséges API hívásoknak megfelelően.
3. Az API függvények hívási helyeinek módosítása úgy, hogy azok az új interfész függvényre mutassanak.
   A hívandó API függvények azonosítójának beállítása az interfész függvény meghívása előtt a megfelelő változóban.

7. Összefoglalás

A fentiekben egy olyan megoldást mutattunk be, amelyik ötvözi a kriptográfiát, a szoftver vízjelezést és az obfuszkálást annak érdekében, hogy egy megalapozott és megbízható technikai megoldást eredményezzen a szoftver másolásvédelem területén, elsősorban a mobiltelefon alkalmazási területét célozva meg. A bemutatott módszerre alapozva terveztük meg egy olyan másolásvédelmi eszköz architektúráját, amely integrálható bármely fejlesztői környezetbe annak érdekében, hogy megfelelő másolásvédelmi szolgáltatásokat biztosítson.

A rendszer felépítése robusztus és nyílt abban az értelemben, hogy az az alrendszer, ami mind a vízjelezéssel, mind pedig az obfuszkálással járó átalakításokért felelős teljes mértékben független a processzortól, az operációs rendszertől és a fejlesztői környezettől, hiszen a forráskód egy absztrakt reprezentációján működik. Ily módon, lecserélve az előfeldolgozó, fordító és szerializáló modulokat, számos platformra és fejlesztői környezetbe is integrálhatjuk az általunk kifejlesztett rendszert.

A rendszer megbízható működése érdekében bármely kód-transzformáció esetében az elvégzett műveletek helyességének a formális bizonyítása elengedhetetlen. Ennek megfelelően minden transzformációt két lépésben valósítunk meg: az egyes transzformációk megtervezése, azaz a transzformációk céljainknak legjobban megfelelő sorozatának a létrehozása után a különálló és sokkal egyszerűbb transzformációs lépéseket úgy kell végrehajtanunk, hogy az általuk végrehajtott műveletek helyessége már formálisan is bizonyítható legyen.

A keretrendszer elkészülte után a kutatás-fejlesztési projektünk következő lépéseként további transzfor-

A példa egy API hívást szemléltet a rendszerhívások elrejtését megvalósító transzformáció végrehajtása után:

```
    ldr lr, LI11            @ Visszatérési cím elmentése
    mov ip, #1              @ Ip beállítása a hívandó függvény azonosítójára
    ldr r5, LI12            @ Globális változó címének betöltése
    str ip, [ r5, #0]       @ Azonosító elmentése a globális változóba
    b HideCalls_2           @ Interfész függvény meghívása
LI11:
    .align 0
    .word .L12              @ Következô blokk címe
LI12:
    .align 0
    .word .LD110            @ Globális változó címe
.L12:
```

mációk megvalósítása következik. Célunk egyrészt különböző kontroll és adat obfuszkáló eljárások kifejlesztése és hatékonyságának a tesztelése, másrészt pedig a transzformációk révén a dinamikus vízjelek elrejtése a kódban, majd ezek detektálhatóságával kapcsolatos mérések elvégzése.

Köszönetnyilvánítás

A kutatási projektet a Gazdasági Versenyképesség Operatív Programja (GVOP 3.1.1/AKF) támogatta.

Irodalom

[1] G. Arboit:
A Method for Watermarking Java Programs via Opaque Predicates,
In The Fifth International Conference on Electronic Commerce Research (ICECR-5), 2002.

[2] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan and K. Yang:
On the (im)possibility of obfuscating programs,
In: Proc. CRYPTO'01.
Lecture notes in computer science, Vol .2139.
Springer, pp.1–18, 2001.

[3] First Annual BSA and IDC
Global Software Privacy Study, 2004.
Business Software Alliance and IDC Global Software

[4] C. Collberg, C. Thomborson, and D. Low:
A Taxonomy of Obfuscating Transformations,
Technical Report 148, Dept. of Computer Science,
The Univ. of Auckland, 1997.

[5] C. Collberg, and C. Thomborson:
On the Limits of Software Watermarking,
Technical Report 164, Dept. of Computer Science,
The Univ. of Auckland, 1998.

[6] C. Collberg, C. Thomborson, and G. M. Townsend:
Dynamic Graph-Based Software Watermarking,
Technical Report TR04-08, 2004.

[7] R. Davidson, and N. Myhrvold:
Method and system for generating and auditing a signature for a computer program,
US Patent 5,559,884, Microsoft Corporation, 1996.

[8] G. Hachez:
A Comparative Study of Software Protection Tools Suited for E-Commerce with Contributions to Software Watermarking and Smart Cards,
Ph.D. thesis,
Universite Catholique de Louvain, 2003.

[9] International Telegraph and Telephone Consultative Committee (CCITT):
The Directory – Authentication Framework,
Recommendation X.509, 1988.

[10] A. Mana, J. Lopez, J. J. Ortega, E. Pimentel and J. M. Troya:
A framework for secure execution of software,
International Journal of Information Security,
Vol. 2, Issue 4, pp.99–112, Springer,
November 2004.

[11] A. Monden., H. Iida, and K. Matsumoto:
A Practical Method for Watermarking Java Programs,
The 24th Computer Software and Applications Conference (compsac2000),
Taipei, Taiwan, October 2000.

[12] J. Palsberg, S. Krishnaswamy, M. Kwon, D. Ma, Q. Shao, and Y. Zhang:
Expreience with Software Watermarking
In Proc. of the 16th Annual Computer Security Applications Conference, ACSAC'00,
pp.308–316, 2000.

[13] J. P. Stern, G. Hachez, F. Koeune, and J.-J. Quisquater:
Robust Object Watermarking: Application to Code,
In A. Pfitzmann, editor, Information Hiding '99,
Vol. 1768 of Lectures Notes in Computer Science,
pp.368–378, Dresden, Germany, 2000.

[14] R. Venkatesan, V. Vazirani, and S. Sinha:
A Graph Theoretic Approach to Software Watermarking,
In Proceedings of the 4th International Workshop on Information Hiding table of contents,
pp.157–168, 2001.

[15] G. Wroblewski:
General Method of Program Code Obfuscation,
Ph.D. thesis, Wroclaw University of Technology,
Institute of Engineering Cybernetics, 2002.

Ma már Magyarországon is óránként készülnek új „Fürdő utca 10”-ek

HORVÁTH ÁKOS, DR. HORVÁTH LÁSZLÓ
Budapesti Műszaki Egyetem, Puskás Tivadar Távközlési Technikum
lacibacsi@puskas.hu

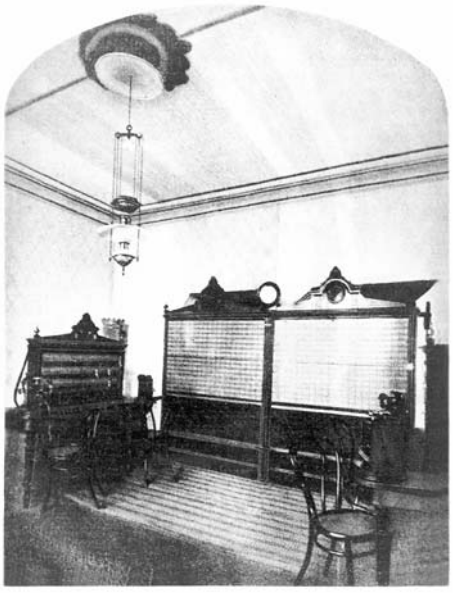
A korabeli krónikákból olvashatjuk, hogy 1881. május 1-jén a Fürdő u. 10. számú ház III. emeletén, Matkovitséktól bérelt szobában megkezdte működését az első magyar telefonközpont, a „Budapesti Telefonhálózat” nevű cég jóvoltából. A cégtulajdonosok a Puskás fivérek: Tivadar és Ferenc voltak. Érdekesség, hogy az első előfizetők összekapcsolását a lakástulajdonos unokája Matkovits Júlia (akit barátnői csak Ilonkának hívtak) végezte, vagyis Ő volt az első magyar telefonos kisasszony.

125 év távlatából most már minden sallangtól letisztulva próbáljuk felidézni a legendás éveket. A párizsi telefonközpont megnyitásának hetében 1879. július 4-én Puskás Tivadar öccse, a szabadságolt huszárfőhadnagy saját, Gyöngytyúk utcai lakásán a II. emelet és a földszint között nyilvános telefonbemutatót tartott a „főváros előkelő köreinek” [4]. Ez az előfizetőtoborzó volt a telefon első pesti bemutatkozása. Mindenki el volt ragadtatva a „szellemes és mulattató játékszertől”. Senki nem hitte, hogy a szórakoztatáson kívül másra is jó lenne. Néhány nap múlva, 1879. július 28-án Puskás Ferenc igazgató beadta kérelmét a Budapesti Telefonhálózat engedélyeztetésére a Közmunka- és Közlekedési Minisztériumba. Ugyanakkor Puskás Tivadar megrendelte a Bell Companytól a telefonközpontot és a telefonkészülékeket. 1880. május 20-án végre megérkezett az illetékes, (nem a megcímzett) báró Kemény Gábor Földművelés-, Ipar- és kereskedelmi miniszter 4767. számú rendelete, melyben Puskás Ferenc 20 évre kizárólagos jogot kapott arra, hogy Budapest főváros és Újpest község területén távbeszélő összeköttetéseket építsen [2].

A pesti üzleti negyed centrumában a Fürdő utca és a Percz köz sarkán, a III. emeleten bérelt sarokszobában megkezdődött az első 200 vonal kapacitású, LB-típusú telefonközpont szerelése. 1881. februárjában Puskás Ferenc előfizetői felhívást tett közzé. A jelentkezőknél azonnal felszerelték a készüléket és megkezdték a légvezetékek kiépítését. Matkovits Júlia betanítását maga Puskás végezte. „Licencének” dátuma 1881. április 1-je. (Későbbiekben az új jelentkezőket Matkovits Júlia tanította be, vagyis tekinthetjük az első magyar távközlési szaktanárnak, postaforgalmi ágazaton.)

1881. május 1-jén 25 előfizetővel megkezdte hivatalos működését a Fürdő u. 10-ben a Budapesti Telefonhálózat Társaság első telefonközpontja [1]. (Egy másik forrás szerint 1881. május 15-én [3], 54 előfizetővel indult meg a budapesti telefon szolgáltatás). A bécsi központ csak egy hónap múlva, június 3-án nyílt meg. Mi sem jellemzi jobban a Puskás fivérek dinamizmusát, mint – ahogy már említettük –, hogy a következő központot a Lövész utca 7-ben már 1881. augusztus 1-jén átadták. (Itt volt a cég irodája, tehát kísérleti hívásokat már hónapokkal előbb is kapcsolhattak, akár a Fürdő utcai központ üzembe helyezése előtt is. Innen is adódhatnak a téves információk az első központ pontos helyéről.) Még abban az évben, 1881. december 1-jén megnyílt az első vidéki, a temesvári telefonközpont. Az Osztrák-Magyar Monarchia négy telefonközpontja közül akkor három Magyarország területén működött.

A sors fintora, hogy közvetve Puskás Tivadar okozta a Fürdő utcai központ megszüntetését is: 1887. november 28-án indult meg nagykörúti, keskenyvágányú villamos próbavasút, melynek ünnepi első járatát Puskás vezette a Nyugati pu. és a Király utca között [5]. Az új tömegközlekedési eszköz hihetetlen gyorsan elterjedt és 1890-re a villamos vontatás okozta földzárlatok, és áthallások miatt lehetetlenné tette a biztonságos telefonálást. (Akkoriban a telefon egyvezetékes volt és az áramkör a földön keresztül zárult.) A ‘90-es évektől már csak kétvezetékes vagy kábeles előfizető áramköröket szereltek és a Szerecsen (ma Paulay Ede) utcai 4800 előfizetős új központ üzembe helyezése után 1901. szeptember 21-én a Fürdő utcai központ megszűnt.



Boston után 4 évvel, Párizs után 2 évvel már működött a telefon Budapesten. (Ez olyan technikai bravúr volt, mintha 1969 után 4 évvel, vagyis 1973-ban magyar ember járt volna a Holdon!) Puskás még rátett egy lapáttal és a telefon felhasználásával egy új médiumot szabadalmaztatott; a Telefonhírmondót.

Aztán jött az I. világháború, Trianon, a II. világháború, a vasfüggöny, a Berlini Fal és lebontották a vasfüggönyt, leomlott a Berlini Fal... Eltelt a Telefonhírmondó megindulása után 100 év, és ismét újabb médium jelent meg a világon, az Internet. Ennek is lett egy kezdetben melléktermékе, manapság már szolgáltatása: a VoIP, vagyis hangátvitel az interneten,

pontosabban beszédkommunikáció a csomagkapcsolt adathálózaton. Megfordult a trend: eddig adatot vittünk át modem segítségével a hanghálózaton, mostantól hangot viszünk át egy program segítségével az adathálózaton.

A Fürdő utca ma már nem létezik. József Attila az utca neve és a számozás is más – ma 14. lenne a helyes házszám. A Hild tér, vagyis egy üres telek van a ház helyén. Az eredetit lebombázták. Tégláit széthordták, beépítették az újjáépítés során a város házaiba. Ott szunnyadtak 60 évig, és most újra élednek. Manapság egyre több lakásban szólal meg telefonprogram felhasználásával a számítógép: „Halló, ott ki beszél?" A szélessávú internet technika, a VoIP technológia elterjedésével ma már Magyarországon is akár 100.000 lakásban a felhasználó tulajdonos (inkább gyermeke, vagy mint akkor 1881-ben unokája) veszi át a kezelő(nő) szerepét [7]. A Fürdő utca 10. elosztott paraméterűvé vált. Ma már mindenki csinálhat magának egy sajátot. Hála a városépítők előrelátásának, az eredeti ház hatalmas, háromemeletes épület volt. Így a széthordott tégláiból akár 3 millió lakásba is juthat, hogy minden család élvezhesse a szélessávú internet, benne a Skype, MSN, Kapcs stb. újabban már képesített áldását is. Az előrejelzések szerint 2006 végére már legalább 200.000-en fognak hazánkban a PC-n otthonról telefonálni.

Köszönjük Mr. Bell, köszönjük Puskás Tivadar, köszönjük Matkovits Ilonka, és köszönjük az összes néven nevezhető és névtelen telefonos, internetes szakembernek; hol dicsérjük, hol inkább szidjuk találmányotokat... Egy biztos; nap mint nap használjuk.

Irodalom

[1] Az 50 éves magyar távbeszélő 1881–1931. Magyar Posta, V. évf. 5. szám, Budapest, 1931.

[2] Rimótzy Mihály (szerk.): Posta Mérnöki Szolgálat 1887–1937. Magyar Királyi Kereskedelmi és Közlekedési Minisztérium, Budapest, 1997.

[3] A Kereskedelmügyi M. Kir. Miniszter: A M. Kir. Posta és Távíró legújabb műszaki berendezései. Hornyánszky Viktor, Cs. és Kir. udvari nyomdája, Budapest, 1908.

[4] Gábor Luca (szerk.): Telefonhírmondó. Magyar Rádió, Budapest, 1993.

[5] Dr. Szabó Dezső: A Budapest Közúti Vasút 100 éve. Közlekedési Dokumentációs Vállalat, Budapest,1966.

[6] Fodor István: Edison magyar úttörői. Magyar Mérnök- és Építészegylet Közlönye, 1927. pp.9–10.

[7] Halló, kivel beszélek? – Internettelefónia Magyarországon Antenna magazin, 2005/4. pp.48–50.

A telefonközpont

Mikor adták át az első telefonközpontot és hol? Amerikában, ez biztos, de mikor? A Postamúzeum birtokában van egy eredeti fénykép, melyre 1912-ben Thomas Alva Edison ráírta Puskás Albert (Puskás Tivadar öccse) özvegye kérésére, hogy „Puskás Tivadar volt az első ember a világon, aki felvetette a telefonközpont gondolatát". Mindez 1877-ben Edison New York közeli Merlo Parkjában történt [6], annyi biztos, hogy az első felében az évnek, hiszen Puskás 1877. július 30-án Londonban Edison európai megbízottjaként jegyezteti be fonográf szabadalmát.

Sokáig a New Havent tekintették az első telefonközpont színhelyének (1878. január 25-i nyitással.) Azóta azonban kiderült, hogy 1877. májusában már üzemelt Bostonban egy betörésjelző rendszer, amelyen éjszakánként az előfizetők jelentkezhettek veszély esetén, nappal pedig telefonkapcsolatokat valósított meg a bekötött állomások között, vagyis valamiféle telefonközpontszerű funkciót is megvalósított.

A fenti információk birtokában próbáljuk meg leírni egy hihető verziót: Puskás olvasott a bostoni betörésjelző rendszerről és beszámolt róla Edisonnak, sőt tovább is fejlesztette azt saját ötleteivel, különösen, felhasználhatóságát hangsúlyozva. Ezen az úton azonnal elindult a fejlesztés, melyek eredményeként az első klasszikus, New Haven-i telefonközpont a következő évben átadásra is került. Ugyan ezen az elven építették meg és adták át Puskás európai megérkezése után, az „Európai Edison Telefontársaság" szabadalmainak értékesítésével: 1879. június 26-án a párizsi, 1879. októberében a londoni, 1880. január 1-jén a zürichi, és még a budapesti előtt közvetlen, 1881. január 12-én a berlini telefonközpontokat.

Mivel a kontinens első „távbeszélő-üzemű távíróhivatala" a Berlin melletti Friedrichbergben nyílt meg 1877. november 12-én, vagyis több mint másfél a párizsi előtt, alig néhány hónappal Puskás Londonba visszaérkezése után, és mivel ez a központ Siemens-telefonokkal volt felszerelve, két telefonközpontos vonal látszik kibontakozni: Az egyik a Bell–Bostoni betörésjelző és Siemens–Friedrichberg távíróhivatal. Ez volt a későn szabadalmaztatott, féllegális ág. A másik a szabadalmakkal védett, legális Edison-féle készülékek – a Puskás féle központ gondolattal, a New Haven-i kezdettel Amerikában és párizsi indulással Európában. Ennek az ágnak hatodik állomása lett a budapesti telefonközpont is.

Azt a felvetést, hogy Puskás tudott a bostoni betörésjelzőkről és annak ismeretében fejlesztette ki a saját elképzelését alátámasztani látszik az 1881. február 1-jén kiadott „Előfizetési felhívás" szövege: „Hirtelen rosszul lett beteg orvosával, lángba borult házat a tűzoltósággal, a meglopott tőkepénzes a rendőrséggel egy perc alatt tudathatja a veszélyt, s megtehetik a szükséges intézkedéseket" [4].

Semmit nem von le Puskás Tivadar zsenialitásából, ha elismerjük, hogy volt egy az övét megelőző, majd elhaló telefonközpont fejlesztési vonal is. Puskás legtávolabbra mutató újítása, a világot közel 20 évvel megelőző új médium-szabadalma az 1893-ban útjára bocsátott Telefonhírmondó volt.

PROTECTION AGAINST DHA WITH CENTRALISED FILTERING

Keywords: Directory Harvest Attack, black list, DNS, centralised protection

Obtaining the e-mail addresses which are handled by the mail servers is the Directory Harvest Attack. The root of the problem in DHA is in the SMTP protocol itself: the e-mail servers, if they got the mail to a proper address, would not respond, simply accept it. In our article we would like to introduce our research, development, and show the results gained from the running of the implemented system. The implemented protection is component based developments, which are strongly coherent and use each other software elements to a high extent. We analyse the data collected by our system. We present which typical DHA attackers exist and whether it is possible to distinguish them unambiguously from each other based on just the attacker statistics.

DISCOVERING FIREWALL RULES

Keywords: firewalk, firewall reconnaissance, discovering rulesets, network reconnaissance, portscan

This publication focuses on a specific field of network reconnaissance: discovering firewall rulesets. After explaining the basic theories, We'll cover in detail the FireWalk technique, it's possible extensions and how to mitigiate the risks.

VIRUSES, WORMS, ROOTKITS, THE LATEST THREATS

Keywords: virus, worm, rootkit, algorithms, kernel mode, user mode, detection tool, hook, DKOM

It's about year and a half since Windows rootkits have started to appear for the public, making users' and security specialists' life more difficult. It turned out how difficult is to prevent, detect and counter this type of malware. What is not clear, is it a possibility to merge this technology with the old malware types? In this article we discuss the most powerful features of worms and rootkits, and some of the detection tools against them.

SECURITY TESTING WITH SOURCE-CODE-BASED FAULT INJECTION

Keywords: security testing, fault injection, source-code-based testing

Exploitable security vulnerabilities cause huge damages year by year. This paper introduces a method, which discovers security flaws relying on fault-injection-based security testing aiming to provide a cost-effective alternative to today's expensive security enhancing techniques, such as formal validation or exhaustive testing. Our test framework, called Flinder, enters test vectors deep into the internal function-calling mechanism of the Target of Evaluation (ToE). The basic idea of our technique is that these 'faulty' test vectors are created by modifying (fuzzing) the original parameters passed to a given function. These modifications contain typical faults that are usually the causes of security flaws. They are produced by generic algorithms, which work automatically, solely relying on data descriptors easily derivable from the source code definitions of the data structures. After injecting these faulty parameters, Flinder analyzes the behavior of the ToE, and assesses whether the ToE handled the fault well, or did the test vector cause a reaction similar to an exploitable vulnerability.

WIFI SECURITY – WEP AND 802.11i

Keywords: WLAN, WEP, 802.11i, WPA2, 802.1x, EAP, TKIP, CCMP, AES, authentication, access control, confidentiality

In this paper, we present a tutorial on WiFi security by giving an overview of the related standards such as WEP and 802.11i.

OUTSOURCING OF SECURITY SERVICES FOR PORTABLE DEVICES

Keywords: WLAN, portable device, security, outsourcing, Security Proxy, security services

Nowadays, the number of portable devices using wireless network technologies is rapidly increasing. These devices are incapable of using secure Internet services, because secure communication often requires computing capacity. In our article we introduce a solution which can be used to offer secure network services for low performance portable devices without data transmission speed degradation. We also show that how this device can utilize a new secure network service which was unavailable to it at this point because the lack of resources.

HOW TO ACCEPT AN ELECTRONIC SIGNATURE?

Keywords: electronic signature, timestamp, certificate revocation list, OCSP, signature policy

The mathematical principles of electronic signatures have been known for several decades. Since then, electronic signatures have been incorporated into the Hungarian legal system: they have become legally equivalent with handwritten ones. Although we can now rely on both their mathematical and on their legal foundations, this technology has began to spread but in the last few years. The practical application of this new technology highlighted certain problems. Many of these also appear in case of handwritten signatures, but have lesser significance in that case.

ATTACKS AGAINST ANONYMIZER NETWORKS

Keywords: anonymizer networks, traffic analysis, traffic distortion

Anonymizer networks are used to hide the communication of the users from local networks and to assure that the location of the users remains unknown in remote networks. In this paper, the limits of these networks will be shown; we introduce our specific attack against the anonymity of the users of these networks. Our attack is based on the idea that we distort the traffic at the remote networks and we search for this distortion in other networks. We show by examples that present anonimizer networks do not provide protection against our attack. Thus, the location of the users can be tracked back started from the remote networks, by detecting the traffic distortion pattern in other networks.

COPY PROTECTION THROUGH SOFTWARE WATERMARKING AND OBFUSCATION

Keywords: software copy protection, software watermarking, obfuscation, reverse engineering, trusted OS, mobile software

Enforcement of copyright laws in the field of software products is primarily managed in legal way by the software developer companies, as the available technological solutions are not strong enough to prevent illegal distribution and use of software. Almost all copy protection techniques were cracked within some weeks after their market launch. The lack of technical copyright enforcement solutions is responsible for the failure of some recently appeared business models, partially also for the recent dotcom crash. The situation is particularly dangerous in case of the growing market of mobile software products. In this paper we aim to propose a scheme which combines obfuscating and software-watermarking techniques in order to provide a solution which is purely technical, but strong enough to overcome the problems concerning software copy protection. The solution we propose is focusing primarily on mobile software products, where we can rely on the hardware based integrity protection of the operating system.

Summaries • of the papers published in this issue

Contents

Cover: Tivadar Puskás, intellectual father of the telephone switching office and inventor of the wire-line Telephonograph

Szerkesztőség
HTE Budapest V., Kossuth L. tér 6-8.
Tel.: 353-1027, Fax: 353-0451, e-mail: info@hte.hu

Hirdetési árak
1/1 (205x290 mm) 4C 120.000 Ft + áfa
Borító 3 (205x290mm) 4 C 180.000 Ft + áfa
Borító 4 (205x290mm) 4 C 240.000 Ft + áfa

Cikkek eljuttathatók az alábbi címre is
Szabó A. Csaba, BME Híradástechnikai Tanszék
Tel.: 463-3261, Fax: 463-3263
e-mail: szabo@hit.bme.hu

Előfizetés
HTE Budapest V., Kossuth L. tér 6-8.
Tel.: 353-1027, Fax: 353-0451
e-mail: info@hte.hu

2006-os előfizetési díjak
Közületi előfizetők részére: bruttó 30.450 Ft/év
Hazai egyéni előfizetők részére: bruttó 6.800 Ft/év
HTE egyén tagok részére: bruttó 3.400 Ft/év

Subscription rates for foreign subscribers:
12 issues 150 USD,
single copies 15 USD

www.hte.hu
Felelős kiadó: NAGY PÉTER
Lapmenedzser: Dankó András

HU ISSN 0018-2028

Layout: MATT DTP Bt. • Printed by: Regiszter Kft.