Bizalom, biztonság és a nyílt forráskód kapcsolatának vizsgálata

Készítette a Közigazgatási és Igazságügyi minisztérium E-közigazgatási
Szabad Szoftver Kompetencia Központja
Budapest, 2013

Kódszám: EKOP–1.2.15

Tartalomjegyzék

Bizalom, biztonság és a nyílt forráskód kapcsolatának vizsgálata

A zárt forráskód veszélyei: ismeretlenek

2012 márciusában látott napvilágot a hír[1], hogy az ausztrál kormány a nemzeti hírszerzési hivatal[2] tanácsára, biztonsági okból nem engedélyezi a kínai Huawei Technologies Co Ltd indulását az NBN Co által kiírt tendereken, így az állami digitális fejlesztési programban. A kormányzat szűkszavú nyilatkozata szerint alapvető fontosságú, hogy ez a kritikus infrastruktúra biztonságos legyen. A Huawei vállalattal és annak termékeinek biztonságával kapcsolatban számos kérdőjel merült fel az elmúlt években[3]. Ezeknek a biztonsággal kapcsolatos aggályoknak a forrása rávilágít a zárt forráskódú szoftverekkel kapcsolatos egyik alapvető biztonsági sajátosságra. Mivel a forráskód zárt, így az ilyen rendszerek felhasználóinak nincs lehetőségük megvizsgálni, hogy valójában mit is csinál az adott szoftver. Számos országban egy ilyen vizsgálat kivitelezése pusztán a szállított, bináris rendszeren nem egyszerűen nehézkes, hanem törvények tiltják. A felhasználóknak meg kell bízniuk a fejlesztők szavában, hogy a termék azt és csak azt csinálja, ami a leírásban áll.

2011 augusztusában Tavis Ormandy nyilvánosságra hozott egy rendkívül érdekes elemzést[4], mely a Sophos Antivirus 9.5 szoftver védelmi funkcióinak vizsgálatáról szólt. A rendszer biztonsági funkcióival kapcsolatban számos súlyos problémát talált, a leginkább érdekes, hogy a benne lévő emulátor NT kód támogatása gyenge, egy súlyos hiba miatt 8-9 éve nem is működhetett. Ez az elemzés rámutat, hogy még az ezzel professzionálisan foglalkozó cégek is súlyos biztonsági hiányosságokat hagyhatnak a szoftvereikben. Ez sokkal kisebb valószínűséggel maradhatott volna évekig rejtve, ha a forráskód nyilvános lett volna. A zárt forráskódú rendszerek biztonságára úgy kell tekinteni, hogy a prospektusok által ígért biztonsági funkciók épp annyira biztosak, mint ha valaki tanúsítás nélkül kijelenti magáról, hogy követi az ISO 9001 minőség ellenőrzési szabvány előírásait. A zárt forráskódú szoftvereket fejlesztő cégek kormányoknak és nagyvállalatoknak sokszor – megfelelő titoktartási szabályok mellett – betekintést engednek termékeik forráskódjába, ugyanakkor

[1] http://www.reuters.com/article/2012/03/26/us-australia-huawei-nbn-idUSBRE82P0GA20120326
[2] Australian Security Intelligence Organization (ASIO)
[3] http://en.wikipedia.org/wiki/Huawei#Security_concerns
[4] Sophail: A Critical Analysis of Sophos Antivirus

MAGYARORSZÁG MEGÚJUL
A projekt az Európai Unió támogatásával, az Európai Regionális Fejlesztési Alap társfinanszírozásával valósul meg.

semmi nem bizonyítja, hogy a szállított termék pontosan abból a forráskódból készül, amit átadnak betekintésre. Erről csak akkor tud a felhasználó meggyőződni, ha ő maga fordítja le forráskódból az adott szoftvert. Erre komoly biztonsági igény esetén általában szükség is van (lásd A szoftverek moduláritása fejezetet).

Ken Thompson, a számítástechnika egyik atyja, akinek a munkái (C programozási nyelv, Unix rendszerek) alapvetőek a modern számítástechnikában, 1984-ben megjelent klasszikusában[5] elmélkedik arról, hogy ha a fordítóprogram rosszindulatú részeket helyez el a binárisba, akkor hiába a szoftverfejlesztők minden igyekezete, az onnan semmilyen alapos tervezés és implementáció mellett sem távolítható el, hisz nem a program forráskódja a hibás. Ha egy fordító környezet zárt forrású, működése nem megismerhető, akkor semmi nem garantálja, hogy az olyan binárist fog készíteni, mely pontosan azt csinálja, amit a forráskód alapján kellene. Ha valakinek ráhatása lenne egy ilyen fordítórendszer fejlesztésére, terjesztésére, az képes lenne az összes ezzel az eszközzel készített program fölött átvenni a felügyeletet. A zárt forrású szoftver fejlesztői környezeteknél ki garantálja, hogy nincs ilyen befolyás akár érdekelt üzleti körök, akár valamely állam nemzetbiztonsági intézményei részéről? Ilyen esetben egy szoftvernek hiába lenne makulátlan a tervezési-fejlesztési-tesztelési-terjesztési-üzemeltetési ciklusa, akkor is biztonsági hibás lenne.

[5] Ken Thompson: Reflections on Trusting Trust; *Communication of the ACM*, Vol. 27, No. 8, August 1984, pp. 761-763.

Titkolózás és biztonság kapcsolata

Az Egyesült Államok Kereskedelmi Minisztériumának Nemzeti Szabvány és Technológiai Intézete (National Institute of Standards and Technology – NIST; a továbbiakban NIST) több dokumentumban is kifejezetten javasolja a titkolózáson alapuló biztonság módszerének kerülését. Az általános szerver biztonsági útmutató például a következőt fogalmazza meg: „A rendszer biztonságának nem szabad a megvalósítás vagy annak részeinek titokban tartásától függnie."[6]

Mikor a jól képzett kriptográfusok még szinte kizárólag a nemzetek titkosszolgálatainak dolgoztak, a titkolózáson alapuló biztonság többé-kevésbé elfogadott volt. Manapság, mikor a kriptográfusok jelentős része egyetemeken és kutatóintézetekben dolgozik, és munkáikat természetesen nyilvánosságra hozzák, másokkal is ellenőriztetik, az algoritmus titkokként kezelése veszt korábbi népszerűségéből. Egy bizonyítottan erős algoritmusnak nem szabad arra építenie, hogy titkos. Erre jó példa a PGP[7] amelynél a forráskód nyilvános, mégis általánosan katonai szintű rejtjelezési rendszernek fogadják el. Egy másik jó példa erről a területről az Egyesült Államok új szimmetrikus rejtjelezési szabványának kidolgozásának a története. 2000-ben a NIST három éves nyilvános kiválasztási folyamatot követően jelentette be, hogy a következő általuk elfogadott szimmetrikus rejtjelezési szabvány az addig Rijndael-nek nevezett, belga kriptográfusok által kidolgozott algoritmus, mely nem csak megfelel a kor elvárásainak, hanem a hosszú kiválasztási eljárás során számos professzionális, kriptoanalízissel foglalkozó kutató megvizsgálta, és nem sikerült gyengeséget találni rajta.[8] Később ez lett a FIPS szabvány-csomag 197.-ként bejelentett szabványa.[9] Elmondható tehát, hogy a kriptográfiában általánosan elfogadottá vált az az elv, hogy az algoritmus eltitkolása önmagában nem alkalmas annak biztonságának növelésére.

Az informatika biztonsági szakemberek között némi vita van a titkolózáson alapuló biztonság[10] hasznosságán, de az általánosan elfogadott álláspont, hogy sem a szoftver fejlesztésben, sem a rendszer üzemeltetésben nem szabad erre építeni a védelmet. A titkolózás lelassíthatja a támadót, de nem fogja megállítani.

Hibák feltárása és a forráskód nyíltsága

Általánosan elmondható, hogy a szoftverek biztonsági tesztelése azok forráskódjának elérhetőségétől függetlenül ún. fekete doboz teszteléssel kezdődik. Ez a leggyakrabban azt jelenti, hogy a támadó automatizáltan, speciális támadó szoftverek segítségével (fuzzing eszközök) megpróbál olyan (általában véletlenszerű) mennyiségű, formátumú információt bevinni a szoftvernek, melytől az a tervezett, helyes működési folyamatait elhagyja. Ennek az eléréséhez a támadó olyan csatornákon keresztül is megpróbál a szoftverbe adatot bevinni, amit a fejlesztő nem ellenőriz megfelelően. (Például környezeti változók értékén, vagy előre létrehozott átmeneti fájlokon keresztül.) Ha ezeken a teszteken keresztül a támadó sikeresen rátalál egy potenciális biztonsági hibára, akkor a forráskód megléte csak egy könnyítő tényező, megfelelő felkészültségű támadó a zárt forráskódú szoftverekben is képes a hibák kihasználására. Számos olyan szoftver létezik, mely lehetővé teszi a bináris

[6] Eredeti nyelven: "System security should not depend on the secrecy of the implementation or its components." Forrás: Guide to General Server Security. National Institute of Standards and Technology. July 2008. Retrieved 2 October 2011.

[7] http://en.wikipedia.org/wiki/Pretty_Good_Privacy

[8] http://csrc.nist.gov/archive/aes/round2/r2report.pdf

[9] Federal Information Processing Standards Publication 197 November 26, 2001 Announcing the ADVANCED ENCRYPTION STANDARD (AES) http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[10] Security through obscurity

programok visszafejtését, működésének feltérképezését.[11] Noha a támadót a nyílt forráskód felgyorsíthatja, de semmi sem garantálja, hogy valóban fel is gyorsítja a hiba kihasználásában.

Auguste Kerckhoffs a 19. században írt cikkeiben számos, a rejtjelezéssel kapcsolatos megállapítást tesz. Talán a legfontosabb, mely később Kerckhoffs axiómájaként vált ismertté a következő. „A rejtjelező rendszernek akkor is biztonságosnak kell lennie, ha a kulcson kívül minden részlete ismert."[12] Az ismert amerikai matematikus és kriptográfus, az információelmélet atyja, Claude Shannon[13], így adaptálta Kerckhoffs axiómáját: "Az ellenség ismeri a rendszert." Valódi biztonsági igény esetén nem építhető arra a védelem, hogy a támadó nem ismeri a forráskódot.

Ugyanakkor a nyílt forráskódú szoftverek esetében a támadónak lehetősége van forráskód elemző szoftverekkel potenciális biztonsági hibákat keresni. Azt is el kell azonban mondani, hogy akár jelentős zárt forrású termékek forráskódja is számos esetben kiszivárgott[14], így ez a lehetőség sokszor akkor is adott, ha a szoftver alapvetően nem nyílt forráskódú. Ezt a lehetőséget az elemző szoftverek gyártói arra használják[15], hogy a jelentősebb nyílt forráskódú szoftvereket elemzik, és az eredményeket ajándékként a közösség rendelkezésére bocsátják, ezzel népszerűsítve magukat és a terméküket.

2004-ben a Carnegie Mellon egyetem négy kutatója négy éves kutatás után azt a megállapítást tette, hogy a Linux kernel forráskódja extrém biztonságos a zárt versenytársakhoz képest[16]. Megállapításuk szerint míg egy átlagos kereskedelmi szoftver termék esetében 20-30 hiba jut 1000 forráskód sorra, addig a Linux kernelben ez az érték 0,17. Ezt a gigantikus eltérést számos tényező okozza, az egyik ezek közül a forráskód nyíltsága lehet.

Hátsó ajtók, beépített felhasználók és jelszavak

A fejlesztők szándékosan vagy csak kényelemből a rendszerbe építhetnek, esetleg benne felejthetnek olyan hátsó ajtókat, melyek megkönnyítik a hibák elhárítását, kényelmesebbé teszik a fejlesztést, tesztelést, de a produktív környezetben elfogadhatatlanok. Miért tennének ilyet a fejlesztők? Ennek több oka lehet. Az egyik ilyen, ha a fejlesztés alatt nem szeretnének minden alkalommal újra és újra a valós jelszavukkal bejelentkezni, ekkor kikapcsolhatják a felhasználóhitelesítést, például a jelszó ellenőrző függvényt úgy módosítják, hogy minden megadott jelszóra azt adja vissza, hogy a hitelesítés sikeres. Minden bizonnyal ez lehet az oka a következő példának. A Dropbox nevű felhő adattároló szolgáltatásban 2011 júniusában, feltehetőleg egy fejlesztői figyelmetlenségből adódóan minden felhasználó tetszőleges jelszóval beléphetett a rendszerbe.[17] A hibát néhány órán belül elhárították, de a közben eltelt idő alatt bárki bármilyen adathoz hozzáférhetett, csak a megtámadott felhasználó nevére volt szüksége.

Elképzelhető, hogy a fejlesztők a tervezők utasítására hagynak szándékosan hátsó ajtót a rendszerben. Erre jó példa az Allied Telesis cég esete. A cég 2011 májusában véletlenül nyilvánosságra hozta[18] azokat a hátsó ajtó céljából beégetett jelszavakat, melyeket arra az esetre épített a rendszerbe, ha a felhasználók kizárnák magukat a rendszerből. Természetesen az ilyen lehetőségek szűk körű nyilvánosságra kerülés esetén veszélyeztetik az adott termék minden felhasználóját, de a széles nyilvánosság esetén is tipikus, hogy a fejlesztő cég csak napok alatt képes a probléma elhárítására. Ebben a támadási ablakban a felhasználó gépe védtelen.

[11] Visszafejtés (angolul: reverse engineering)
[12] Kerckhoffs axiómája
[13] Claude Shannon
[14] Ismert forráskód szivárgások
[15] Coverity Scan projekt a nyílt forráskódú szoftverek biztonságának javítására
[16] Linux: Fewer Bugs Than Rivals
[17] http://www.hwsw.hu/hirek/46904/dropbox-cloud-tarhely-biztonsagi-hiba.html
[18] http://www.h-online.com/security/news/item/Allied-Telesis-divulges-secret-backdoor-1251556.html

Nagyon érdekes és veszélyes hibára hívta fel a figyelmet Charlie Miller[19], az ismert számítógép biztonsággal foglalkozó szakértő 2011 júliusában. Az Apple cég Mac típusú számítógépeiben lévő akkumulátorok és a számítógép olyan gyengén, beégetett jelszavakkal védett protokollon kommunikált, melyet Miller sikeresen használt ki az akkumulátorok mikrokontrollerének átprogramozására. Ez a hiba a támadó számára akár azt is lehetővé teszi, hogy az hibás töltési parancsokkal távolról felrobbantsa az akkumulátort. Ugyanakkor olyan támadó kód helyezhető el az akkumulátor vezérlő számítógépébe, melyet a hagyományos módszerekkel lényegében lehetetlen detektálni és eltávolítani.

És ilyen példákat rendkívül hosszan lehetne sorolni. Véletlenül a nyílt forrású programok fejlesztői is elkövethetnek hibákat, de amennyiben szándékosan próbálnak ilyet elhelyezni, tisztában vannak vele, hogy a szakmai renoméjukat kockáztatják. Mivel a nyílt forrású szoftverek forráskódja nyilvános, így az ilyen, titkolózáson alapuló problémák nem léteznek, ebben a kontextusban értelmezni sem lehet a „titkos hátsó ajtó” kifejezést, hiszen a forráskód nyíltsága a felhasználók számára minden esetben lehetővé teszi a rendszer működésének teljes megismerését.

A valódi biztonság záloga: a tervezés

A szoftverek forráskódjának eltitkolása semmilyen garanciát nem nyújt azok biztonsági funkcióinak erejére. A védelmi funkciók megfelelő erejét kizárólag úgy lehet megalapozni, ha a biztonságot mint elvárást a fejlesztés első pillanatától, a tervezéstől kezdve szem előtt tartják. Általánosan elfogadott, hogy a valódi biztonság alapjait a tervezés során kell letenni. Számos példa bizonyítja, hogy a rosszul tervezett rendszerek biztonsága később rendkívül nehezen vagy nem növelhető.

A korábbi TCSEC[20], ITSEC[21] és CTCPEC[22] információ biztonsági ajánlások alapján nemzetközi, informatika biztonsági szakértőkből álló csoport kidolgozta a Common Criteria[23] nevű, biztonságos szoftver fejlesztéssel kapcsolatos iránymutatásokat tartalmazó módszertant. A dokumentum 2.1-es változata 2009-ben 15408 lajstromszám alatt ISO szabvánnyá vált. Ennek magyar megfelelője a MIBÉTS (Magyar Informatika Biztonsági Értékelési és Tanúsítási Séma). Mindkét módszertan azt az iránymutatást fogalmazza meg, hogy a biztonságos szoftver fejlesztésének alapvető lépései:

– A fenyegetettségek azonosítása
– Védelmi funkciók tervezése, melyek képesek a feltárt fenyegetettségek elhárítására
– A szoftver kívánt mélységű megtervezése, az elvárt védelmi funkciók figyelembe vételével
– Implementáció a tervek precíz betartásával, a tervezett fejlesztési eljárások mentén
– Szükséges alaposságú tesztelés és forráskód audit, javítás
– A szállítási, üzemeltetési eljárások biztonsági kívánalmaknak megfelelő tervezése
– Érthető, betartható felhasználói dokumentáció

Noha megfelelő garancia szinten extrém biztonsági szintre szánt szoftverek fejlesztésére is tartalmaz iránymutatásokat a két dokumentum csomag, ezek semmilyen követelményt nem fogalmaznak meg a forráskód hozzáférhetőségével kapcsolatban. Ennek az az oka, hogy a világ vezető informatika biztonsági szakemberei szerint nem ettől függ a szoftverek valódi biztonsági szintje.

[19] http://hup.hu/cikkek/20110723/biztonsagi_szempontbol_kockazatosak_lehetnek_a_laptop_akkumulatorok
[20] http://en.wikipedia.org/wiki/TCSEC
[21] http://en.wikipedia.org/wiki/ITSEC
[22] http://en.wikipedia.org/wiki/CTCPEC
[23] http://www.commoncriteriaportal.org

A nyílt forráskódú szoftverek fejlesztői

Általánosan elterjedt tévhit, hogy a nyílt forráskódú programok fejlesztői hobbi programozók, akik csak otthon, a szabadidejükben fejlesztenek szoftvereket. Noha vannak ilyenek is, a tények mégis azt mutatják, hogy a jelentősebb nyílt forrású projektek legfontosabb fejlesztői olyan cégeknek dolgoznak, ahol az adott szoftvert önmagában vagy egy megoldás részeként használják. 2012 márciusában a Linux Foundation által kiadott tanulmány szerint[24] a Linux kernel fejlesztőinek 75%-a valamilyen profitorientált cég fizetett, főállású fejlesztője. A legtöbb esetben ezek a profik tervezik az új funkciókat és vezetik a fejlesztést, a hobbi fejlesztők csak akkor adhatnak forráskódot a szoftverhez, ha az megfelel a vezetők rendkívül magas minőségi követelményeinek. A forráskódokba való írást a legtöbb esetben csak több éve a projektben dolgozó, bizonyítottan jó fejlesztők kaphatják meg, a többiek forráskódjai felülvizsgálva kerülnek be a szoftverbe.

A nyílt forrású projektek legnagyobb része teljesen nyílt, abba bárki bekapcsolódhat. Így a rosszindulatú fejlesztők is könnyen, általában szinte anonim módon tudnak kódot bejuttatni a szoftverekbe. Ha a támadó elég felkészült, akkor képes olyan módon álcázni a bejuttatott hibás kódot, hogy azt komoly tapasztalattal rendelkező fejlesztők sem tudják észrevenni, elutasítani. El kell azonban mondani, hogy ez a lehetőség minden szoftver fejlesztésénél adott, csak nehezebben kivitelezhető. Noha nem anonim módon, de egy jó képességű támadó szoftverfejlesztőként bármelyik zárt forrású fejlesztést folytató cégbe beépülhet. Van szabad szoftver ellenpélda is, például a Google mobil operációs rendszere, az Android[25], és a szintén általuk fejlesztett Chromium webböngésző, ahol a tervezésben és a fejlesztésben nem vehet részt akárki, ezeket a tevékenységeket a cég belül tartja.

A szoftverek modularitása

Mivel a programokat emberek írják és nem robotok, a forráskód tartalmaz hibákat. Ezeknek a hibáknak a száma a forráskód sorok számával közel egyenes arányban van. Az arány konkrét értéke függ a fejlesztői környezettől, a fejlesztő felkészültségétől, lelki állapotától, a projekt méretétől és bonyolultságától és még számos más tényezőtől. Elmondható azonban, hogy ha egy bizonyos szoftver kódjának a méretét csökkentjük, ésszerű módon kihagyunk részeket, akkor a potenciális biztonsági hibák száma is csökkenni fog.

A legtöbb szoftver lényegesen több funkciót tartalmaz, mint amit a felhasználók az adott szituációban használni szeretnének. Ezért a szoftverek bizonyos funkciói beállításokon, konfigurációs állományon keresztül kikapcsolhatók. Egy-egy funkció kikapcsolásával az adott funkcionalitást tartalmazó kódrészleteket virtuálisan eltávolítja a felhasználó a kódból. Ez nagy valószínűséggel csökkenti a potenciális biztonsági hibák számát. Előfordulhat azonban, hogy a támadó valamilyen módon vissza tud kapcsolni egy olyan funkciót, melynek segítségével már képes megemelni a privilégium szintjét.

A nyílt forráskódú szoftverek esetében a felhasználónak lehetősége van arra, hogy igényei szerint, a biztonság vagy egyéb cél érdekében ténylegesen eltávolítsa a nem kívánt funkcionalitást, és így fordítsa le magának a programot. Ekkor a támadó a futó rendszeren nem képes újra bekapcsolni az adott funkciót, hisz az nincs belefordítva a szoftverbe. Ennek elvégzésére a legtöbb nagy kódbázisú, általánosan használt nyílt forrású szoftver beépített funkciót tartalmaz, melynek segítségével fordítás előtt beállítható a fordítási környezetben, hogy mely funkciókra van szüksége az adott helyzetben a felhasználónak (configure scriptek).

[24] Linux Kernel Development – How Fast it is Going, Who is Doing It, What They are Doing, and Who is Sponsoring It; Jonathan Corbet, LWN.net; Greg Kroah-Hartman, The Linux Foundation; Amanda McPherson, The Linux Foundation; March 2012

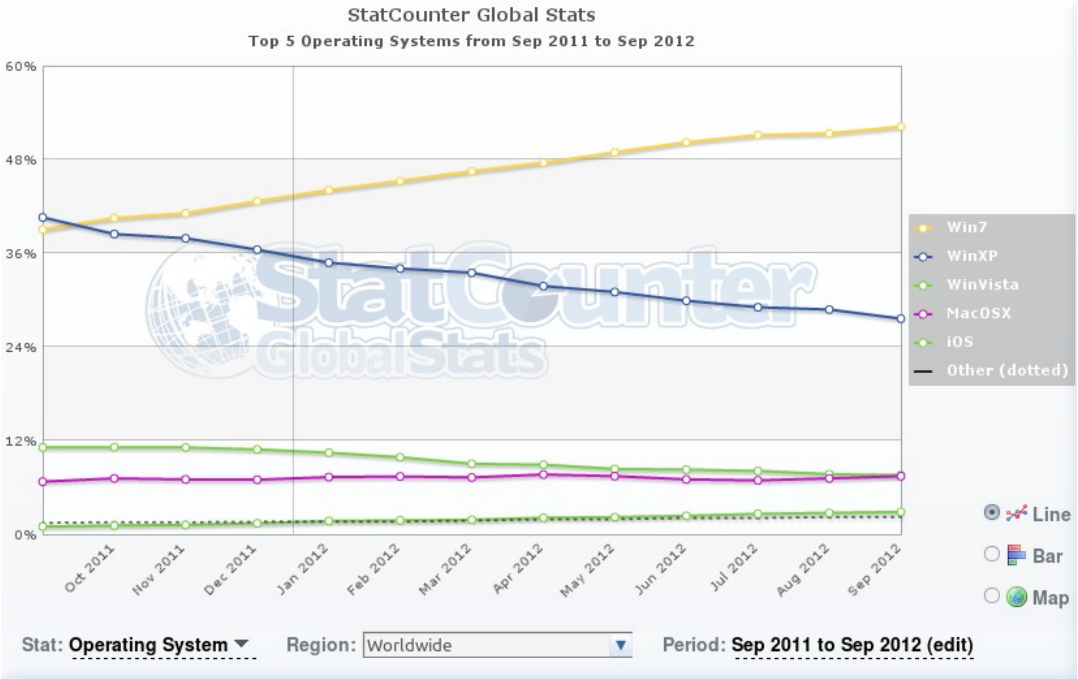
[25] Study: Android Is Least Open of Open Source Mobile Platforms

Extrém biztonsági igény esetén nyílt forráskódú szoftvert használva a felhasználó kisebb modulokra bonthatja a programot, gondoskodva róla, hogy minden részegység csak azokhoz az erőforrásokhoz és fájlokhoz férhessen hozzá, amelyhez ténylegesen szükséges. Ez a lehetőség zárt forráskódú programoknál csak akkor van meg, ha azt a fejlesztő beépítette.

Biztonsági frissítések kezelése

A zárt rendszerek sajátosságai

A szoftverek folyamatos frissítésének lehetősége a legális felhasználók rendelkezésére áll, azokat a modern szoftverek rendszeresen ellenőrzik és felajánlják azok telepítését. A BSA 2010-ben közzétett tanulmánya szerint[26] világszerte nagyon magas a kalóz szoftverek aránya. Ha valaki illegális forrásból szerezte be a szoftverét, így az általában internet alapú regisztrációt nem tudja megtenni, akkor nagyon gyakran nem jut hozzá a biztonsági frissítésekhez. Vagy ha nem is lenne szükség regisztrációra, az illegális szoftverek felhasználói nem frissítik a rendszerüket annak életciklusának lejártakor és sokszor nem merik bekapcsolni a frissítéseket, nehogy a szoftverük egy frissítés miatt használhatatlanná váljon. 2012 szeptemberében a világon a második leggyakrabban használt operációs rendszer változat a Windows XP 27,64%-kal.



1. ábra: Operációs rendszerek részesedésének változása 2012 szeptemberig

Azt a Windows XP rendszert használja a világ felhasználóinak több, mint negyede, melynek a biztonsági támogatása 2009. április 14-e óta kizárólag kereskedelmi ügyfelek számára érhető el. Ez azért probléma, mert az intézményi, vállalati felhasználók jelentős része otthon számos illegális, akár elavult szoftvert használ, és ha ott sikeresen bejut egy támadó a gépre, akkor a támadás viszonylag egyszerűen tovább terjedhet a munkahelyi gépre is. Például USB pendrive vagy a mobiltelefonok otthoni és munkahelyi gép közötti fájl cseréje közben.

A másik komoly probléma, hogy a zárt forrású rendszereknél a felhasználó a legtöbb esetben kiegészítő szoftvereket is használ az operációs rendszer mellett, melyeket különböző forrásokból szerez be. Ilyen lehet egy dobozos szoftver vásárlása a boltban, letöltés az internet különböző weboldalairól. Ezeknek a külső szoftvereknek a biztonsági frissítései általában nem automatikusak, ha a felhasználó nem olvassa a biztonsággal foglalkozó fórumokat (ami a felhasználók nagy részére nyilván igaz), akkor nem is szereznek tudomást róla, hogy frissíteniük kellene. Ilyen esetben a támadó egy hibás és támadható program által kezelt állományon keresztül a széles körben ismert hiba és tá-

[26] PiracyImpact Study: The economic Benefits of reducing software piracy

madási módszer felhasználásával könnyedén megszerezheti a felhasználó gépe fölötti irányítást. Ennek a problémának az orvoslására néhány külső forrásból származó (ún. third-party software[27]) program saját maga ellenőrzi a frissítései rendelkezésre állását (pl. Firefox – mely egyébként szabad szoftver) és esetenként a háttérben el is végzi a frissítést (pl. Google Chrome – mely szintén szabad), de ez a fejlesztői hozzáállás még egyáltalán nem általános. Gyakran előfordul, hogy a szoftver egy bizonyos verziójában már nem is végzik el a javítást, arra hivatkozva, hogy ha a felhasználó megvásárolja az újabb változatot, akkor a frissítés ezt a hibát is javítja.

Komoly gond az is, hogy mivel nincs, vagy nem általánosan használt a szoftvercsomagok rendszer szintű kezelése, így a programok sokszor a telepítéskor újra és újra telepítik a fejlesztés során felhasznált függvénykönyvtárakat. Ha ezekben hiba van, akkor nem csak egy központi helyen kell azokat frissíteni, hanem minden egyes program saját példánya alatt, ami, ha nem is lehetetlen, de rendkívül nehézkesen kivitelezhető. Ez a probléma még az operációs rendszer gyártója által szállított programok esetén is előfordul.[28]

A nyílt rendszerek egyik alappillére: a csomagkezelő

A nyílt forráskódú rendszereknek van egy több, mint egy évtizede működő, általánosan használt módszere a fenti problémák megoldására, mely segítségével a rendszer összes programja és függvénykönyvtára mindig frissen tartható. Ez a csomagkezelő rendszer. A szabad szoftver alapú operációs rendszerek felhasználói szinte kizárólag szabad szoftvereket futtatnak. Noha a rendszer képes tulajdonosi szoftverek futtatására is, és vannak akik bizonyos feladatokra ezt preferálják, de szinte minden feladatra létezik valamilyen szabad szoftver alternatíva, így ezek használata nem tipikus. A szabad szoftverek esetén semmi akadálya egy olyan központi szoftver-csomag tároló létrehozásának, mely minden szoftvernek azt a verzióját tartalmazza, melyben nincs ismert biztonsági hiba. Minden jelentős Linux alapú terjesztés rendelkezik ezen funkcionalitással. Ha egy programban közismertté válik egy hiba, akkor annak javított változata (a legtöbb terjesztésnél a csomagkezelő csapat digitális aláírásával ellátva, tehát a felhasználó oldalán hitelesíthetően) bekerül a központi csomagtárolóba, melyről ezek után minden számítógép a rendelkezésre álló frissítések automatikus ellenőrzése során értesül, és arról a felhasználót is értesíti. Így tehát a szabad szoftver alapú rendszereken tisztán szabad szoftverek használata esetén a támadásra nyitott rendszerek aránya lényegesen kisebb, mint a tulajdonosi szoftvereket is tartalmazó számítógépek.

A szabad szoftvereken elterjedt szokások miatt a zárt forrású szoftverek gyártói gyakran rendelkezésre bocsátják a szoftvereiket hálózaton elérhető, egyedi csomagtárolókból vagy a disztribútor által felállított, elkülönített zárt forrású csomagokat tartalmazó tárolóból[29]. Ezek Linux terjesztéshez illesztése rendkívül egyszerű, és így a zárt forrású szoftverek felhasználói is élvezhetik az automatikus frissítésből adódó biztonsági előnyöket.

[27] http://en.wikipedia.org/wiki/Third-party_software_component
[28] http://seclists.org/bugtraq/2012/Jul/24
[29] Ubuntu partner csomagforrás