Szoftvervédelem

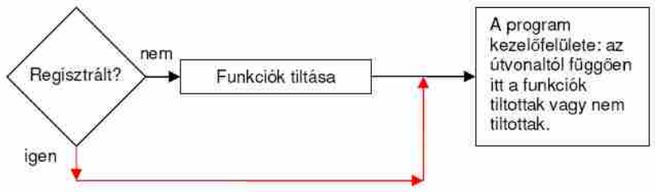
A kreatív programozó

Hol van az a határ, melyről már állítható, hogy egy szoftver védett? Tényezők, amik fölöttébb relatívvá teszik ezt a kérdést: az idő és a viszonyítási alap. Az idő teltével és a kutatások felhasználásával fejlődnek a védelmi mechanizmusok. Mivel egy lefordított programkódban, érdemben turkálni, már nem alapszintű tudást igénylő feladat, ezért egy nem védett szoftver is „védett”. Lássuk be, hogy kizárólag erre támaszkodni bőven nem elegendő.

A programvédelem alapmechanizmusa

Adott egy kereskedelmi szoftver, ahol n fizikailag beépített funkció a felhasználó számára közvetlenül nem elérhető. Ahhoz, hogy ezen *n* funkció közül, legalább egy aktivizálódni tudjon, meg kell vásárolni a terméket. A vásárlás után a felhasználó kap egy ún. regisztrációs kulcsot, amivel aktiválhatja a terméket. Sikeres aktiválás után elérhető lesz az összes olyan tiltott funkció, amiért a vásárló fizetett. Az, hogy az aktiválás milyen módon történik, (pl. regisztrációs szám, kulcsfájl, hardverkulcs…) az a szoftverfejlesztő cég döntése.

Leggyakoribb a regisztrációs szám, amely teljesen egyedi, bonyolult algoritmussal előállított kódsor és általában tartalmaz érvényességi időt, engedélyezett funkciókat és egyéb opcionális adatokat. Ez a **vásárlás-aktiválás** folyamat sok esetben nem így működik, azzal a különbséggel, hogy a vásárlás elmarad. Ez természetesen anyagi mínusz a vállalatnak, amit egyik cégtulajdonos sem szeret. Tény, hogy a vásárlás mindig elkerülhető úgy, hogy az aktiválás sikeres legyen. Tehát **bármely program kódja módosítható úgy, hogy azzal a program funkcionálisan teljes értékűvé váljon. Feltétel, hogy tartalmazzon n fizikailag beépített, de alapesetben nem elérhető funkciót, továbbá tartalmazza valamennyi funkcióhoz vezető feltételes útvonalat.**



A fenti sematikus ábra az útvonalakat illusztrálja. A programot a processzor **alacsony szintű utasítások sorozata**ként hajtja végre. Ezek az **utasítások szabadon módosíthatók** (jogilag nem feltétlen) mielőtt a processzor végrehajtana. Tehát a program útvonali (konstansai, de bármely bit-e) megváltoztathatók. Amit a program feltörők ellen lehet tenni védekezésképp, az a kreatív ötletek: több ellenőrzéspont, ciklikus ellenőrzések, integritás ellenőrzések (…). A védelem kialakítása sokszor a strukturált programozást felborítja. Természetesen a szoftverfejlesztők számára minden ugyanolyan áttekinthető maradhat. A cél mindig: a cracker-ek összezavarása. Igen, még egy fontos közölnivaló: a „cracker” szóról megoszlanak a vélemények; tulajdonképpen én mindig program feltörőt értek alatta. Olyan személyt, aki saját tudását alkalmazva képes egy **szoftver terméket funkcionálisan befolyásolni (kód átírása, kód hozzácsatolása, kulcs generálása) úgy, hogy azzal saját magának vagy/és környezetének kedvezzen.**

Licenc menedzselők lustaság, pénz, tudás és időhiány

Mint szinte minden problémára, a programvédelemre is léteznek **univerzális módszerek**. Gondolom senkinek nem kell bemutatni az univerzális festéket, ragasztót. Noha a tendencia görbéje szigorúan emelkedik; „ezek” mindig elmaradnak a speciális megoldásoktól. De legyen **más nézőpontja** a programozónak! Mikor egy licenc generátor alá kerül a program valójában a következőt teszi a programozó, mint felhasználó:

rábízza egy programra saját alkalmazása védelmi rendszerét,

valójában **fogalma sincs, hogy mi történik** a háttérben, milyen mechanizmussal,

a program feltörők szeretnek „nagy halakra” vadászni, ezért a **cracker-eket fokozottan ingerli** ez a fajta univerzális rendszer,

onnan kezdve, hogy egy alkalmazást sikerült feltörni az adott verziójú, licenc menedzselt programok halmazából elméletileg az **összes többi program feltörhető** ugyanazon (de max. paraméterekben különböző) módszerrel,

az univerzális védelmi mechanizmusok leírásának (cracking tutorials) **megjelenése több olvasótábort** tud magáénak, ezért több ember ismeri meg a védelmi rendszer működését rendszerközeli szinten.

Minden licenc menedzselő, a szoftvert hivatott védeni. Beépítésük a programba változó. Vannak, amikhez abszolút nem szükséges **semmilyen programozói tudás**, csupán az adott PE (Portable Executable) fájlt kell megadni és beállítani az esetleges paramétereket. Shareware programok egy részhalmaza e technológiával védett. **Programozói tudást igénylő** csoportba tartoznak azok a menedzserek, amelyekhez fájlokat és dokumentációt kap a programozó. A dokumentáció tartalmazza a helyes beépítést a forráskódba, és a menedzselést. A védelmi rendszer motorja előre megírt; a programozó feladata a komponensek beépítése és a szükséges környezet beállítása. Ez a technológia általában előnyösebb az előbbinél, mert több beleszólása van a szoftverfejlesztőnek a védelem kialakításában.

Korlátok, fizikai sebezhetőségek

Ahhoz, hogy egy szoftverfejlesztő cég el tudja adni programját, nem elég beszélni róla a felhasználóknak, hanem be is kell azt mutatni nekik. Tegyük fel, hogy az egyik felhasználó letöltött egy kereskedelmi programot, tetszik neki, és használja is. Miért vásárolja meg, ha így is tudja használni? Azért, mert ő egy shareware (demo…) verziót töltött le, ami korlátozottan használható. Például, **30 napra korlátozódik a használat, vagy nem lehet benne menteni**, de a fejlesztőtől függ bármely limitáció meghatározása. Tegyük fel, hogy az egyik programot öt programindítás választja el, hogy funkcionálisan teljes legyen. Tehát ha a felhasználó már hatodik alkalommal szeretne indítani a programját, nem indul. Ekkor a cracker megkeresi azt a részt a program kódjában, ahol ez a feltételes elágazás van. Ez kb. a következőképp néz ki:

```
cmp     eax,00000005    összehasonlítja az eax regisztert 5-tel

ja      KILEP           ha nagyobb, ugrik a KILEP címkére

jmp     TOVABB          a program tovább fut
```

A cracker általában kétféleképpen közelítheti meg a törést. Visszafordítja a teljes programot assembly nyelvre, vagy nyomkövetővel (hibakeresővel) debuggolja. Az assembly nyelv, API hívások és a rendszerközeli ismeretek elengedhetetlenül szükségesek, mind a töréshez, mind a hatékony programvédelem kialakításához. Visszatérve a fenti utasításssorozathoz, a programkód átírásával megszűnik ez a limit. Ettől a program funkcionálisan nem lehet regisztrált, viszont funkcionálisan teljes értékűvé válik (amennyiben ez volt az egyetlen limitáció és nincs speciális védelem: pl. integritás ellenőrzés).

Regisztráció, aktiválás inaktív korlátok

Adott kereskedelmi szoftver aktiválása történhet regisztrációs szám megadásával, kulcsfájl tallózásával, de ennek módját a szoftverfejlesztő határozza meg. Biztonsági szempontból egy nagyon kritikus pontról van szó. A cracker innen kiindulva tudja megtalálni a regisztrációs szám előállításának algoritmusát, vagy megtalálni azt a pontot, amit átírva regisztráltként indul a program. Miért pont innen indul ki? Mert általában ezen a ponton van a regisztráltság ellenőrzése. A programok több mint 50%-ában ez a hasonlóképp néz ki:

```
call    REG_ELLENORZES      szubrutin, a regisztráció ellenorzése,reg. szám

cmp     eax,00000001        ha eax-ba 1 kerül, akkor regisztrált

je      REGISZTRALT         és regisztráltként folytatódik

jmp     NEM_REGISZTRALT     nem regisztráltként folytatódik.
```

Látható, hogy a REG_ELLENORZES szubrutin-hívásban ellenőrzi a program a regisztrációs számot. Feltételezhető, hogy ez a szubrutin induláskor is meghívódik, hogy ellenőrizzen. Ezen ellenőrzések utáni ugrások átírásával, a program funkcionálisan regisztrált lehet.

***

Remélem, hogy Kedves Olvasó elé tudtam tárni, valójában mi a szoftvervédelem. Ez kinek mennyire fontos, azt mindenki saját maga dönti el. Viszont, ami a cikk folytatására késztet, az a szakirodalom hiánya a hazai és nemzetközi piacon egyaránt. A folytatásban ötleteket írok le, melyek alkalmazásával a szoftver biztonsága növelhető (esetleg tippem újabb tippet szül) az internetes kalózokkal szemben.

Suszter Attila **suszter@freemail.hu**