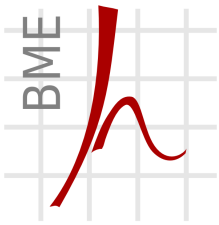
Biztonságos programozás – Alapfogalmak



Híradástechnikai Tanszék

Izsó Tamás

2015. október 1.

Bemutatkozás

Izsó Tamás

- BME Villamosmérnöki és Informatikai Kar
  - Hálózati Rendszerek és Szolgáltatások Tanszék
- honlap: www.hit.bme.hu/∼izso
- email: izso@hit.bme.hu
- szoba: IB124
- Tel: 06 1 463 3277

Milyen tantárgyakat érint

1. Programozás alapjai I. II.
2. Számítógép-architektúrák
3. Gépi nyelvek
4. Operációs rendszerek
5. Assembly programozás
6. stb.

Biztonsági kockázat

A kockázat csökkentése érdekében:

- meg kell tanulnunk milyen biztonsági *fenyegetések* vannak;
- meg kell tanulnunk, hogy fejlesztés alatt milyen *sebezhetőségeket* hozhatunk létre;
- ismernünk kell, hogy hogyan csökkenthetjük, vagy blokkolhatjuk a *fenyegetéseket*.

Program sebezhetőség

„Any sufficiently advanced technology is indistinguishable from magic.”

— Artur C. Clarke

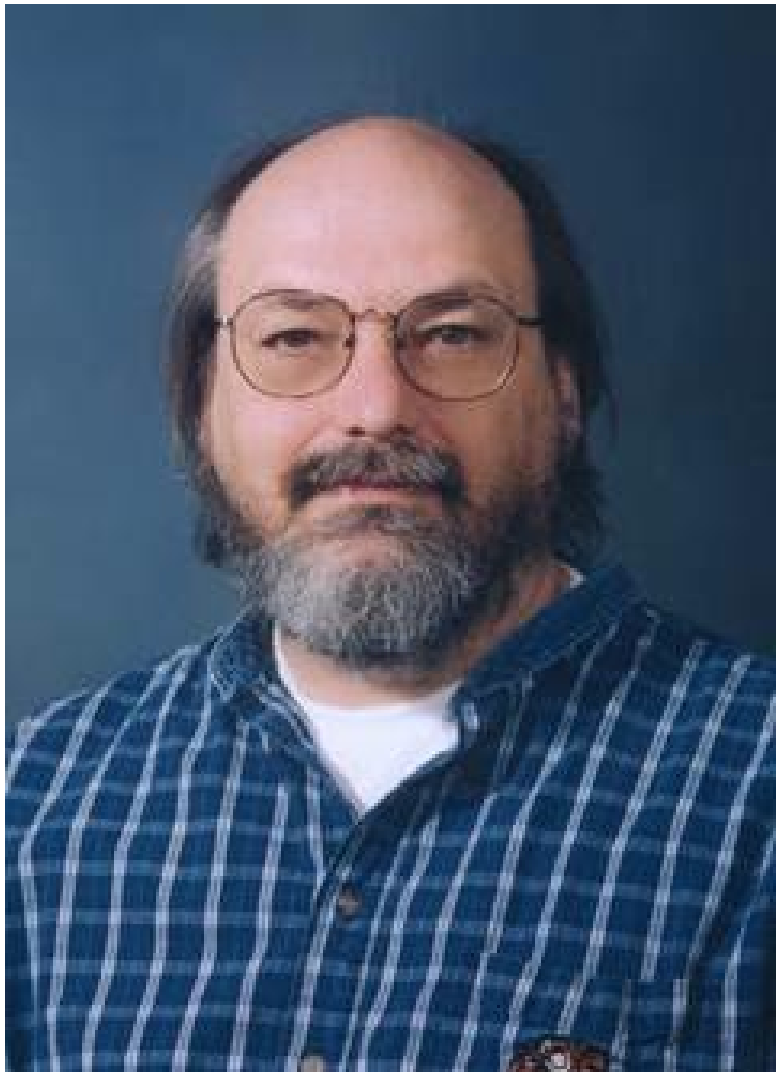
Definíció (Sebezhetőség)

A *biztonsági rések* lehetőséget adnak a hekkereknek, hogy a programok ne a tervezett módon viselkedjenek, így lehetőségük nyílik

- rosszindulatú támadás véghezvitelére;
- érzékeny információt megváltoztatására, törlésére vagy megszerezésére;
- a számítógéprendszer felett az irányítás átvételére.

Első "hacker"

Illusztráció!



Ken Thompson

Illusztráció!

- Főbb munkássága:
  - B Programming Language (Később Dennis Ritchie ez alapján készítette a C-t)
  - UNIX
  - Plan 9
  - reguláris kifejezések elterjesztése
  - Számítógépes sakk algoritmus (végjáték)
  - Rob Pike-vel UTF-8 kódolás (változó hosszúságú)
- 1983: Turing Award (Dennis Ritchie-vel) a UNIX operációs rendszerért
- 1999: US National Medal of Technology
- 1999: First IEEE Tsutomu Kanai Award

Analógia

Illusztráció!

```
char*f="char*f=%c%s%c;main() {printf(f,34,f,34,10);}%c";
main(){printf(f,34,f,34,10);}
```

Output:

Analógia

Illusztráció!

```
char*f="char*f=%c%s%c;main() {printf(f,34,f,34,10);}%c";
main(){printf(f,34,f,34,10);}
```

Output:

```
char*f=
```

Analógia

Illusztráció!

```
char*f="char*f=%c%s%c;main() {printf(f,34,f,34,10);}%c";
main(){printf(f,34,f,34,10);}
```

Output:

```
char*f="
```

Analógia

Illusztráció!

```
char*f="char*f=%c%s%c;main() {printf(f,34,f,34,10);}%c";
main(){printf(f,34,f,34,10);}
```

Output:

```
char*f="char*f=%c%s%c;main() {printf(f,34,f,34,10);}%c
```

Analógia

Illusztráció!

```
char*f="char*f=%c%s%c;main() {printf(f,34,f,34,10);}%c";
main(){printf(f,34,f,34,10);}
```

Output:

```
char*f="char*f=%c%s%c;main() {printf(f,34,f,34,10);}%c"
```

Analógia

Illusztráció!

```
char*f="char*f=%c%s%c;main() {printf(f,34,f,34,10);}%c";
main(){printf(f,34,f,34,10);}
```

Output:

```
char*f="char*f=%c%s%c;main() {printf(f,34,f,34,10);}%c";
main(){ printf (f,34,f,34,10);}
```

Analógia

Illusztráció!

```
char*f="char*f=%c%s%c;main() {printf(f,34,f,34,10);}%c";
main(){printf(f,34,f,34,10);}
```

Output:

```
char*f="char*f=%c%s%c;main() {printf(f,34,f,34,10);}%c";
main(){ printf (f,34,f,34,10);}
```

C fordító

Illusztráció!

- A C (cc, gcc, cl) fordítót C-ben írták.
- Speciális karaktereket csak úgynevezett *escape sequence* segítségével írhatunk le. Például '\n', '\\' .

```
c = next();
if(c != '\\')
    return c;
c = next();
if(c == '\\')
    return '\\';
if(c == 'n')
    return '\n'
...
```

C fordító

Illusztráció!

Adjunk hozzá új *escape sequence*-t.

- A C (cc, gcc, cl) fordítót C-ben írták.
- A '\v'-t még nem ismeri a régi verziójú fordító, ezért fordítása hiba lesz.

```
c = next();
if(c != '\\')
    return c;
c = next();
if(c == '\\')
    return '\\';
if(c == 'n')
    return '\n'
. . .
if(c == 'v')
    return '\v' //hiba
```

C fordító

Illusztráció!

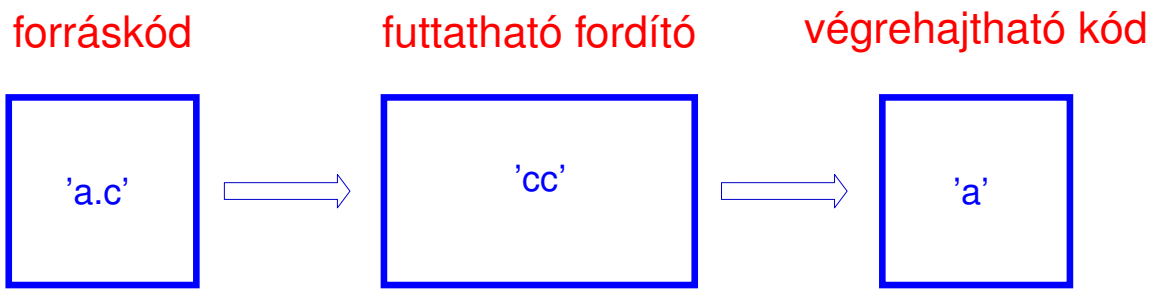
Adjunk hozzá új *escape sequence*-t.

- A C (cc, gcc, cl) fordítót C-ben írták.
- A ’\v’-t még nem ismeri a régi verziójú fordító, ezért fordítása hiba lesz.

```
c = next();
if(c != '\\')
    return c;
c = next();
if(c == '\\')
    return '\\';
if(c == 'n')
    return '\n'
...
if(c == 'v')
    return 11; // jo
```

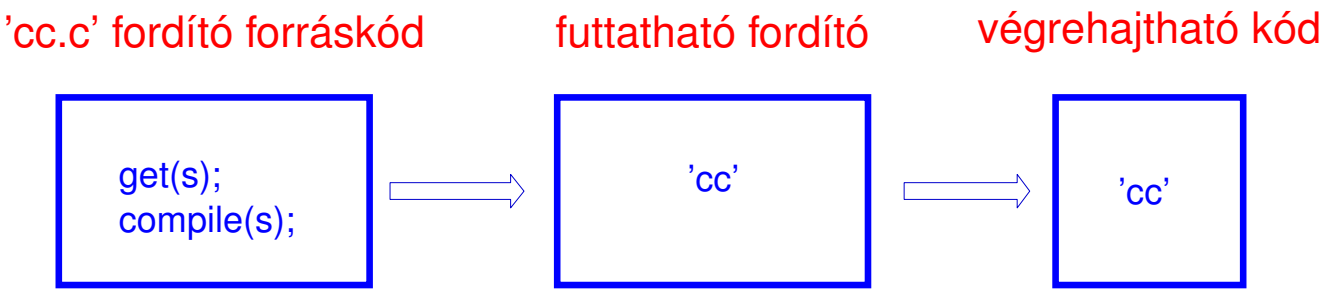
Illusztráció!

Fordító működése



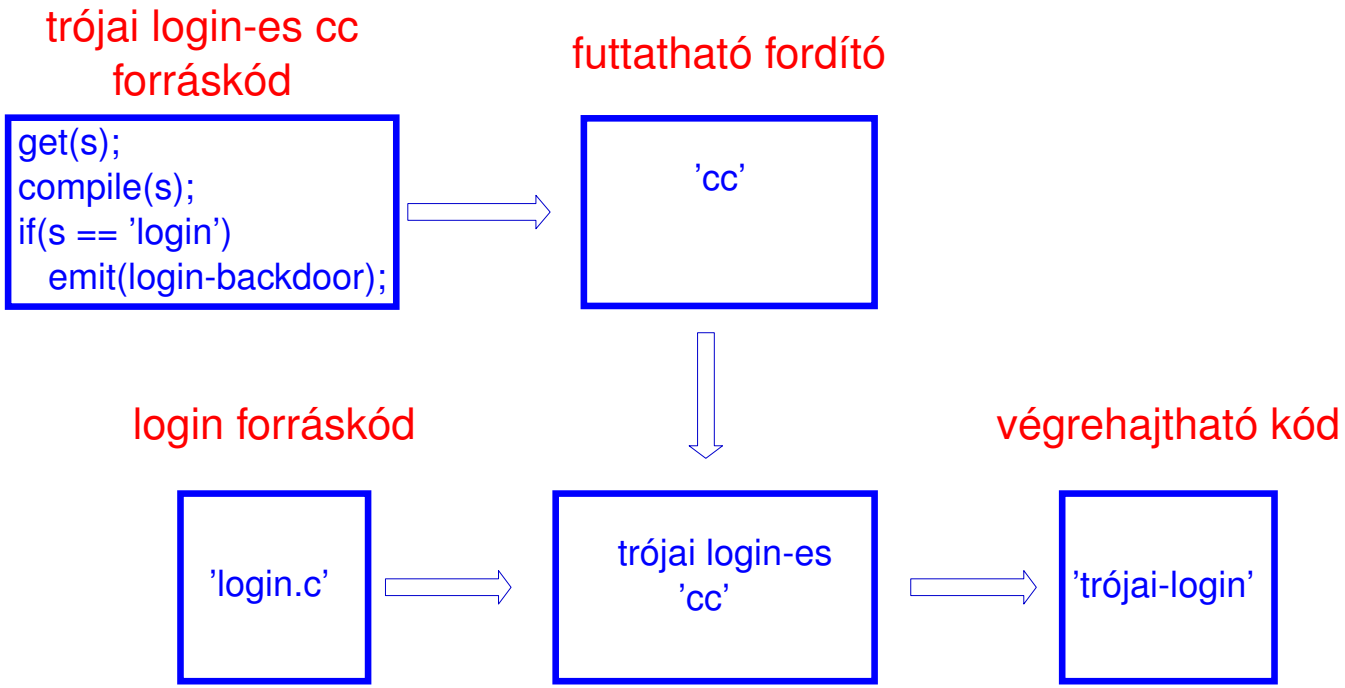
Illusztráció!

Fordító működése saját forrására



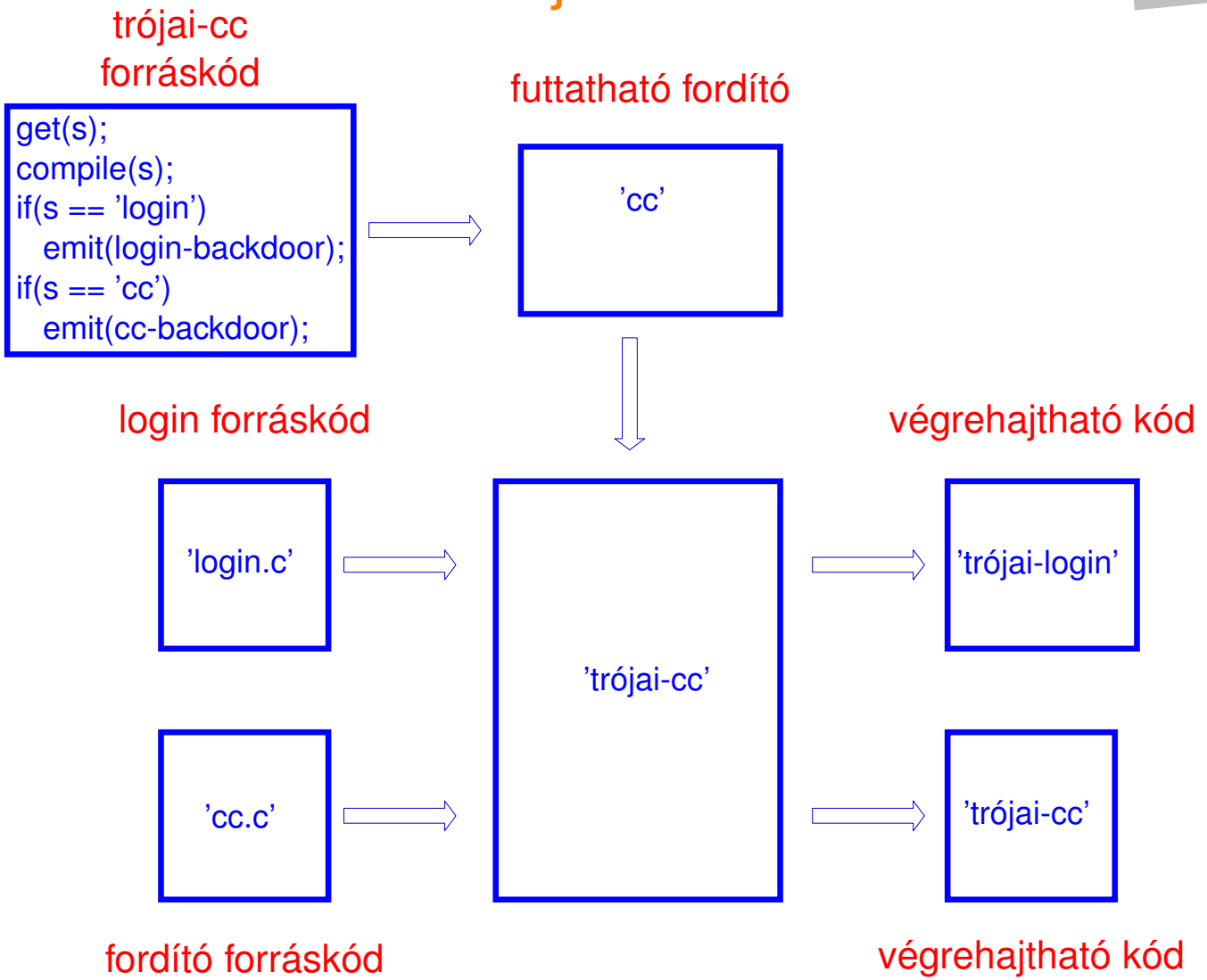
Illusztráció!

Trójai login



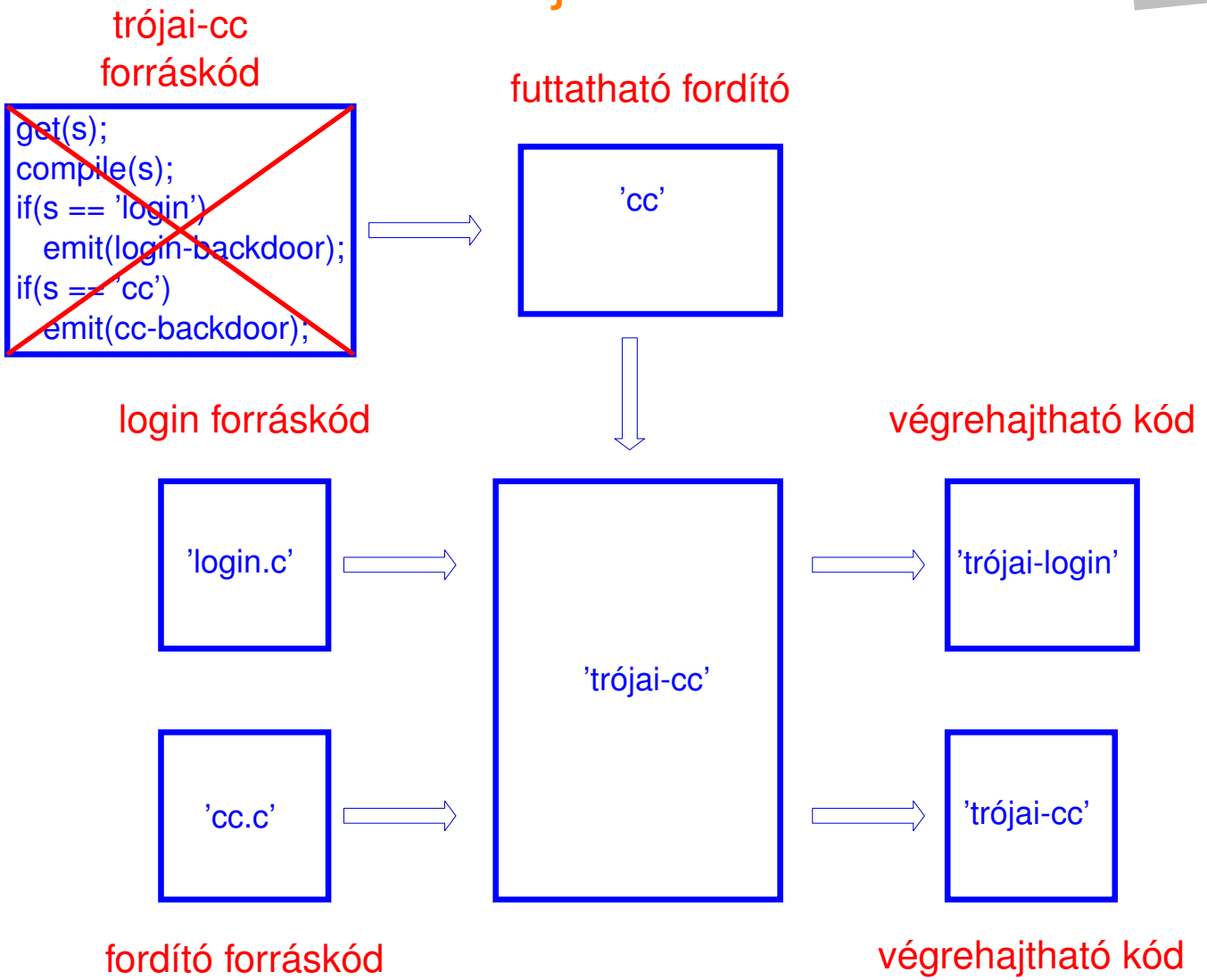
Illusztráció!

Trójai fordító



Illusztráció!

Trójai fordító



Biztonsági rés

- Általában[1] a biztonsági rés egy programhiba eredménye.
- Nem minden programhiba vezet kockázatos működéshez, azaz nem minden hibát tud a támadó kihasználni.
- A szoftverbiztonság részhalmaza a szoftver megbízhatóságnak.

[1] Bizonyos esetekben a program nem követeli meg a biztonságos működést pl.: Telnet, ilyenkor nem hibáról, hanem kockázatos program használatról beszélünk.

Fenyegetés

Definíció (Fenyegetés)

A *fenyegetés* olyan körülmény, melyek káros lehet a működésre, és potenciálisan sérti a biztonságot.

A fenyegetésbe beletartozik, hogy:

- ki milyen értékek megszerzése érdekében támad;
- milyen erőforrásokat használ;
- milyen célból gondolta ki;
- milyen valószínűséggel sikerül kivitelezni a támadását

Biztonsági szabályzat (security policy)

Biztonsági szabályzat meghatározza, hogy a rendszer résztvevőinek mit szabad tenni és mit nem.

Rendszer résztvevői:

- felhasználók;
- programok;
- védett objektumok (erőforrások);

Kérdések:

- Ki határozza meg a szabályokat?
- Hol tároljuk a szabályok betartásához szükséges információkat.
- Hogyan kényszerítjük ki azokat?

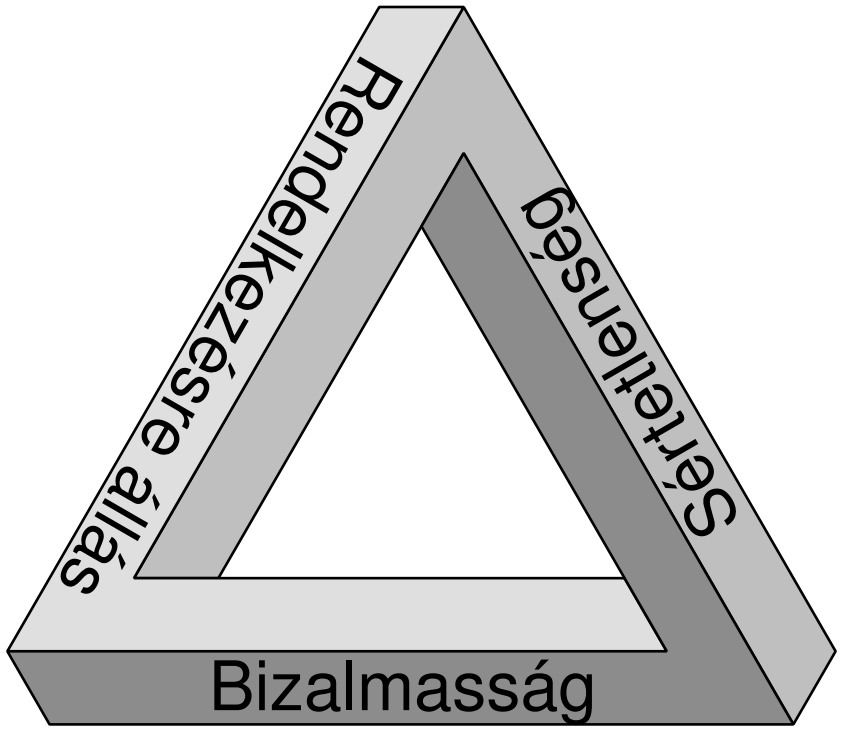
Biztonsági szabályzat betartatása

- Kritikus beágyazott rendszere esetén (robotpilóta, szoftverrel ellátott egészségügyi berendezések, forgalomirányítás, stb.) matematikai módszerekkel kell ellenőrizni az előírások teljesülését.
- Sok esetben csak egy dokumentum tartalmazza a szabályzatot. Például a bankkártya adatait nem szabad felfedni harmadik fél számára. Ezek a szabályok már a szoftver tervezés folyamán elkészülnek, amelyek foglalkoznak a
  - szoftver használatával,
  - hálózat elérésével (az ott lévő programokkal),
  - az operációs rendszerrel,
  - adatbázis hozzáférésével,

  szemben támasztott követelményekkel.

Biztonsági elvárások

- **C**onfidential - bizalmasság.
- **I**ntegrity - sértetlenség (integritás).
- **A**vailability - rendelkezésre állás.



Bizalmasság

Az adatokat a szoftver kezelje magánjellegűként. Bizonyos esetben a program ne adja ki az információkat, illetve annak még a létezéséről se szolgáltasson adatokat.
Bizalmas információk:

- üzleti,
- pénzügyi,
- egészségügyi,
- személyes adatok (születési idő, vallás, lakcím, munkahely, telefon, email, stb.)

Eszközök a bizalmasság megvalósítására:

- jogosultságkezelés,
- kriptográfia.

Sértetlenség I

A sértetlenség magában foglalja a adatforrás szavahihetőségét és a szolgáltatott adatok helyességét.

- Az adatokat illetéktelen ne tudja megváltoztatni.
- Adatváltozás visszakövethető legyen (ki, mikor, mit).

Sértetlenség megőrzése:

- adatok mentése,
- ellenőrző összeg,
- hibajavító kódolás,
- védett metaadatok bevezetése
  - tulajdonos,
  - létrehozási, módosítási dátumok,
  - jogok (írás, olvasás, végrehajtás).

Sértetlenség II

Sértetlenség megtartása:

- adatok sérülésének a meggátlása,
- sérülés detektálása,
- sérülés kijavítása.

Sértetlenség kiértékelése:

- ki volt az adat forrása,
- ki védte az adatot mielőtt hozzánk került,
- mennyire volt védve az adattovábbítás alatt,
- mennyire védett a saját gépen.

Rendelkezésre állás I

A felhasználó a szükséges adatokat és erőforrásokat minden pillanatban el tudja érni.

Biztosítása

- Fizikai:
  - külön energiaforrás,
  - bombabiztos szoba,
  - földrengésbiztos épület.
- Számítógépes redundancia:
  - RAID (redundant array of inexpensive disks),
  - szerverfarm.

Rendelkezésre állás II

1. példa: Flooder (elárasztó) segítségével a hekkerek olyan nagy adatforgalmat tudnak generálni, hogy ezzel lebénítják a szolgáltatást (DoS Denial of Service). Fontos, hogy a DoS támadás után a rendszer önmagától fel tudjon állni.

2. példa: OTP átutalások leállása tavasszal. Az OTP ügyfelű boltokban nem lehetett bankkártyával fizetni 4 órán keresztül. (Nem DoS támadásról volt szó, a leállás oka ismeretlen.)

3. példa:

- Bob rengeteg kreditkártyát tulajdonít el;
- blacklist.visa.com-on ezek a kártyák fekete listára kerülnek;
- Bob támadást intéz a blacklist.visa.com, így a kereskedők nem tudják lekérni a fekete listára került kártyák számát.

Szoftver felülvizsgálat szükségessége

Hogyan szerezhetünk bizonyosságot, hogy a vásárolt program a CIA alapelveknek megfelel?

Mit vállal a gyártó az End User License Agreements-be?

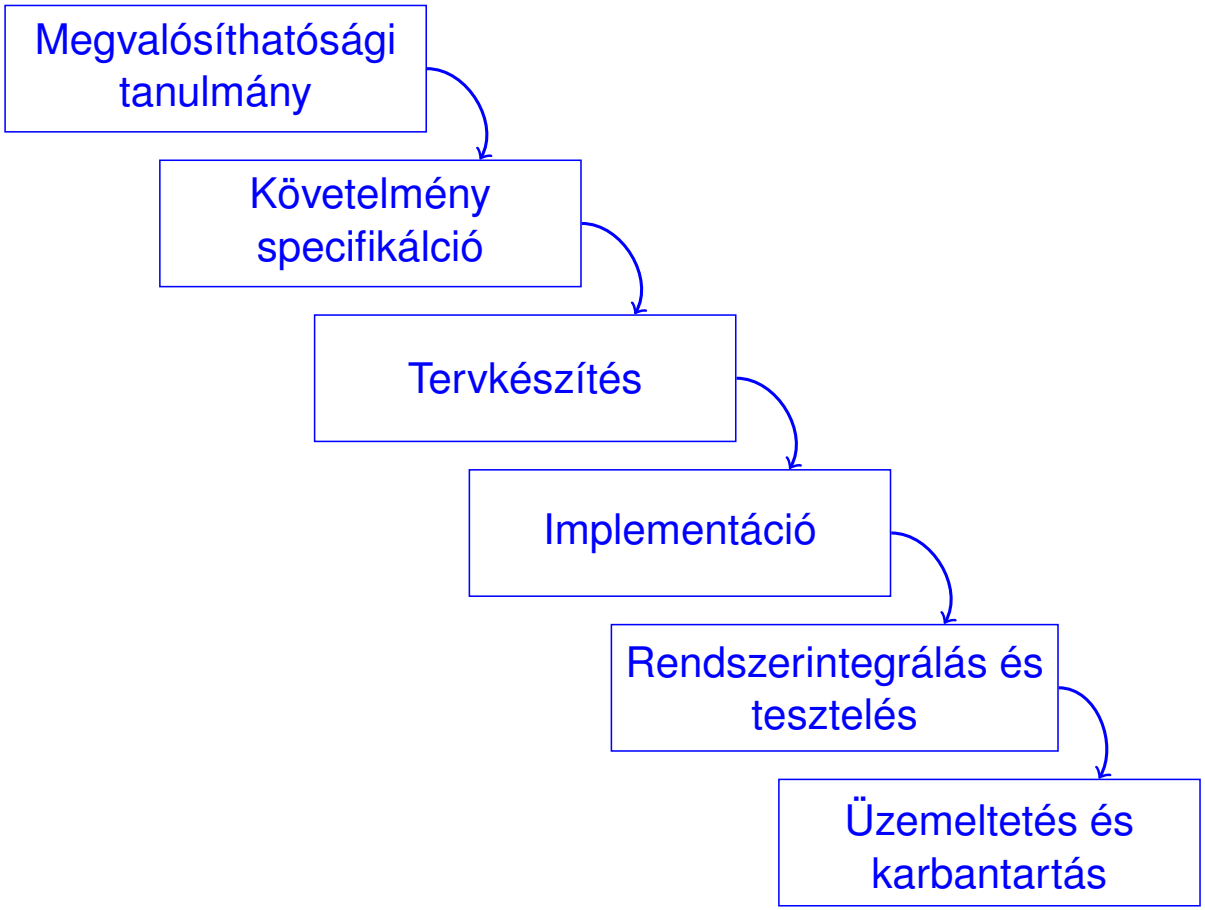
- Linux - nem vállal felelősséget semmiért.
- A fizetős Windows sem.

”Felelősségkorlátozás - Az alkalmazandó jog által megengedett legnagyobb mértékben és a Microsoft Garanciában foglalt kivételekkel a Microsoft és beszállítói kizárják felelősségüket valamennyi kárért (ideértve többek között az üzleti haszon elmaradásából, az üzleti tevékenység félbeszakadásából, az üzleti információ elvesztéséből származó károkat és egyéb vagyoni hátrányt), amely a Szoftver használatából vagy használhatatlanságából ered, még akkor is, ha a Microsoft-ot az ilyen károk bekövetkezésének lehetőségéről tájékoztatták. A Microsoft-nak a jelen Szerződés bármely rendelkezése alapján fennálló teljes felelőssége minden esetben az Ön által a Szoftverért ténylegesen kifizetett összeg erejéig korlátozott. Ezek a korlátozások nem vonatkoznak azokra az esetekre, amikor az alkalmazandó jogszabályok nem teszik lehetővé a felelősség korlátozását illetve kizárását.”

— Microsoft XP, EULA

Szoftver életciklus - Systems Development Life Cycle (SDLC).

Vízesés modell



Szoftverfejlesztési modell

1. Megvalósíthatósági tanulmány megvizsgálja, hogy a felhasználó kívánalmai kivitelezhetőek-e adott technológiai és pénzügyi feltételek mellett.
2. Követelmény specifikáció tartalmazza, hogy a megvalósítandó rendszernek milyen szolgáltatásokat kell tartalmazni, és milyen környezetben kell az elkészítendő programnak futni.
3. Tervkészítés foglakozik az elkészítendő szoftver architektúrájával, az alrendszerek leírásával adat és algoritmus szinten, és az alrendszerek közti interfészekkel.
4. Implementáció a tényleges szoftver megvalósítását jelenti.
5. Rendszerintegrálás és tesztelés fázisban az elkészült alrendszerek összeépítése és vizsgálata zajlik.
6. Üzemeltetés és karbantartás magában foglalja a szoftver telepítését, módosítását, javítását, és a felhasználó által adott visszajelzéseket.

Létrejövő sebezhetőségek kategorizálása

- Hibás tervezés miatt létrejövő sebezhetőségek - SDLC 1,2,3 fázis.
- Hibás implementációból adódó sebezhetőségek - SDLC 4, 5 fázis.
- Rossz működtetésből származó sebezhetőségek - SDLC 6. fázis.

Hibás tervezés miatt létrejövő sebezhetőség

Alapvető hiba amely az összes fázisra rányomja a bélyegét. Költséges javítani.

Egyéb elnevezések:

- magasszintű sebezhetőség;
- architekturális repedés;
- hiányos program követelmény vagy kikötés.

Követelmény és terv fázis

Neptun program követelménye:

- A hallgatók tantárgy felvételének a lebonyolítása.
- Egyidejűleg legyen képes 400 hallgató kiszolgálására.
- stb.

Telnet protokoll követelmény

- Távoli gép elérése a lokális terminálon keresztül.
- Az összeköttetés nem titkos.

Tervezési biztonsági hibák

- Alaptévedés amely kapcsolatos a program szerkezetével, kulcs algoritmusával, vagy főbb interfészekkel.
- Túl sokat tételezünk fel más programrészekről, vagy operációs rendszerről.
- Előre nem vagy nehezen látható támadási felületet figyelmen kívül hagyjuk.
- Támadó lenézése - olyan bonyolult a biztonsági rést kihasználni, hogy nem kell vele foglalkozni.

Példa tervezési hibára:

- Telnet nem használ titkosított csatornát $\rightarrow$ lehallgatható.

Implementációs biztonsági hibák

Ebben az esetben a programozó a kódkészítés fázisában,

- osztály tagfüggvény,
- másodlagos szerepet játszó osztályok,
- bonyolult ciklusok,

megalkotásánál visz be gyengeségeket.

Okok:

- A specifikáció nem határoz meg mindent, így a programozónak is *mikrótervezést* kell folytatni.
- A specifikáció túl bonyolult, és nincs lehetőség a felmerült kérdés tisztázására.

Implementációs biztonsági hibák

Implementációs biztonsági hibákat sokszor nehéz megkülönböztetni a tervezési hibáktól.
Például ha valami fontos, miért nem szerepel a követelmények között.

Tipikus implementációs hibák: SQL injection, buffer overflow

Példa:

Telnet demon kilépésnél nem törli a beállításokat, így a következő user kihasználhatja azokat (közrejátszik a dinamikus linkelhetőség). Bizonyos Telnet demon esetén buffer overflow támadás még az hitelesítés (bejelentkezés előtt), így root jogot szerezhet meg a támadó.

Üzemeltetési biztonsági rések

Tervezés és implementálás során lehetőséget biztosítottak, hogy csak jogosultak férjenek hozzá bizonyos műveletekhez, de a telepítés során a jogosultságokat rosszul állítják be.

Ebbe a kategóriába tartozik a social engineering vagy a tolvajlás is.

Példa:

A megfelelően megírt rendszerbe a jogokat vagy érzékeny adatokat valaki otthonról Telnet segítségével viszi be.

Az adat útja a rendszerben I

Esettanulmány : Vállalati értekezlet előrejelző rendszer.

Követelmény:

- Az adatokat hálózaton keresztül lehet megadni.
- Adatok tárolása SQL adatbázisban történik.
- Max 1000 karakteres leírás a megbeszélés céljáról.
- Érintettek körének a megadása.
- Értekezlet időpontjának a megadás.
- Minden reggel a következő 7 nap eseményének a kijelzése.

Az adat útja a rendszerben II

Megvalósítás:

A riport készítő program a megbeszélés célját fix méretű 1000 karakteres tömbbe tölti. A többi részt a programozók tetszőleges hosszú üzenet kezelésére is felkészítették.
A rendszerbe hónapokig tartózkodhat 1000 karakternél hosszabb rosszindulatú üzenet, míg problémát nem okoz.

Ellenőrzési lehetőség:

- adatbevitel idején;
- SQL adatbázisba mentésnél;
- események riport generálásánál.

Felhasználó által szabadon formázott adat komoly fenyegetést jelenthet.

Bizalmi kapcsolatok

- Bizalmi kapcsolat meghatározza a bizalmi szintek között átmenő adatok érvényességének az ellenőrzését.
- Különböző programrészek különböző bizalmi kapcsolatba vannak egymással.
- A tervező és a fejlesztő sokszor úgy gondolja, hogy a megbízható komponens áthatolhatatlan a rosszindulatú támadó számára, így feltételezi, hogy biztonságos annak a használata.
- A bizalmi kapcsolatok tranzitív tulajdonsággal rendelkeznek.

A bizalmi viszony tárgya

A bizalmi viszony az

- adatok érvényességére;
- támogató szoftverek biztonságosságára;
- a program és környezetérére potenciálisan rosszindulatú támadás hiányára;
- a felhasználó és a támadó képességére;
- felhasználói program interfész (API) viselkedésére;
- felhasznált programozási nyelvre terjed ki.

Esettanulmány input adatok érvényességére

A tervező és programozó feltételezései:

- 5000 karakter elegendő a lakcím tárolásához;
- a lakcím csak nyomtatható karaktereket tartalmaz;
- adatok hossza nem lehet negatív;
- a bemenő adat formátuma helyes;
- a hálózaton csak a megengedett adatmennyiség érkezik.

A támadó szemszögéből

- a tervező által ésszerű feltételezéseket meg kell sérteni;
- nem várt módon kell a programot használni;
- extrém adatokat kell a programoknak megadni;
- előre nem látott módon kell a futási környezetet megváltoztatani (pl. rengeteg memóriát foglalni).

Esettanulmány az interfész érvényességére

A tervező és programozó feltételezései:

- csak megbízható partner küld adatokat;
- a támadó csak a program által biztosított felületen vihet be veszélyes adatokat;
- komplex saját titkosított hálózati protokollt használ, ezért az biztonságos.

A támadó szemszögéből:

- rendelkezik program visszafejtő eszközökkel, mellyel megismerheti a hálózati protokollt;
- a támadó létrehozhat olyan klienst, amely hasonlatos a meglévőhöz;
- saját kliensen keresztül manipulálhatja az adatokat.

Környezeti támadások

Szoftver nem légüres térben fut. A környezet részei:

- hardver architektúra;
- operációs rendszer;
- hálózat;
- fájlrendszer;
- adatbázis;
- felhasználók.

Sok esetben a környezeti hatások olyan összetevők, amely ellen csak a nagy tapasztalattal rendelkező tervező tud védekezni.

Versenyhelyzet kihasználás

- Unix operációs rendszeren a védett program a \tmp-be létrehoz egy fájlt.
- A támadó operációs rendszer feladatot ellátó programot indít el. Esetleg DoS támadással lassítja a rendszert.
- Szimbolikus linket készít a szervizprogram által létrehozott fájlra, mielőtt a program a fájl hozzáférési jogokat korlátozná.
- Ezen keresztül olvassa és módosítja a fájlt.
- Ez erős támadásnak számít, ha a sérülékeny program root jogokkal fut.

Követelmény és üzleti logika ellentmondás mentessége

Követelmény:

- Tranzakciók számát minimális érték alatt kell tartani.

Üzleti logika:

- Az ügyfél át tudjon utalni pénzt a személyes bankszámlájáról, minden érvényes bankszámlára.
- Adott ügyfél át tudjon utalni pénzt a vállalkozása bankszámlájáról, minden érvényes bankszámlára.
- Adott ügyfél havonként automatikusan kapjon pénzt a vállalkozása bankszámlájáról a személyes bankszámlájára.
- Ha az ügyfél személyes bankszámlája lemerül, de a vállalkozása bankszámláján több pénz van, akkor automatikusan egyenlítse ki a hiányt a rendszer.

Figyelni kell az üzleti logika ellentmondás mentességére.
Ebben az esetben az ügyfél szándékosan lemerítheti a keretét, így havonta nem csak egyszer utal át a bank a vállalkozásáról pénzt.

Bizalmi kapcsolat

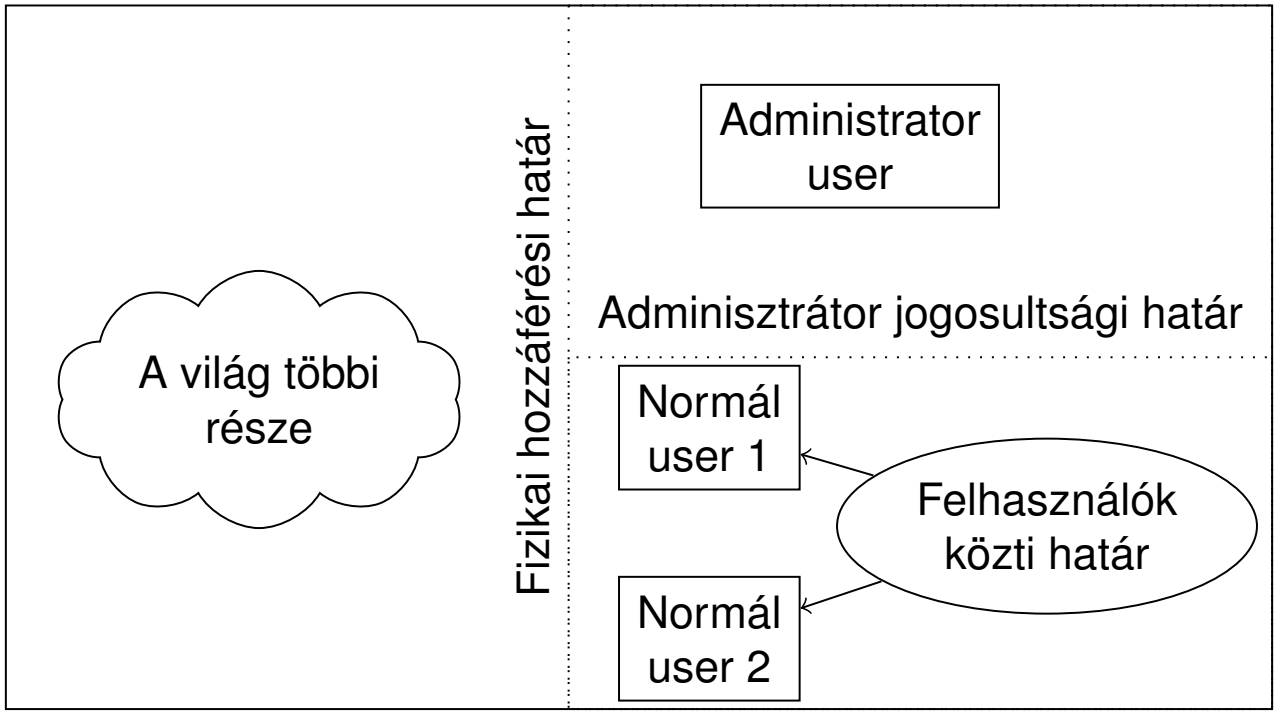
A szoftver alkotóelemei egymással kommunikálnak. A kommunikációs partnerek a bizalom alapján korlátozzák az átadott információk tartalmát, vagy az egymás által nyújtható funkciók halmazát. A felek közötti korlátozott kommunikáció kijelöli a bizalmi határvonalat. A határvonalak bizalmi tartományra osztják a programot.
A bizalom értéke nem bináris, hanem több értéket is felvehet, ezt nevezzük jogosultságnak.

Bizalmi tartomány példa I

Windows 98 egyfelhasználós operációs rendszer. Tekintsünk el a hálózattól. Mindenki aki be tud jelentkezni, az azonos jogosultsággal használhatja a rendszert, végezhet fájlműveleteket, vagy konfigurálhatja az eszközöket. Két bizalmi tartomány van, aki hozzáfér a géphez, és akik nem.

Windows 8.1 többfelhasználós operációs rendszer. Az operációs rendszer biztosítja a felhasználók között a bizalmassági és sértetlenségi viszonyt. Ugyanakkor van egy kitüntetett felhasználó az adminisztrátor, amely mindenhez hozzáfér.

Bizalmi tartomány példa II



Szoftverfejlesztés alapelvei I

- pontosság : megköveteljük a követelmények előállításánál, a tervezésnél és az implementálásnál is. Az implementálás visszahathat a tervre, illetve a követelményekre is, de az összhangnak nem szabad sérülni.
- világosság: A szoftver terve lehet nagyon komplex is, de a jó terv a feladatokat szétbontja könnyen kezelhető magától értetődő részekre.
- gyenge függőség : A szoftverrészek jól definiált felületen kapcsolódnak egymáshoz, az egyes részek a többi rész megváltoztatása nélkül lecserélhető. Erős függőség esetén az egyes részek nagy bizalommal vannak egymás iránt, ezért az átadott adatokat nem ellenőrzik. Ez biztonsági réshez vezethet. Biztonsági szemszögből ilyenkor a fejlesztőnek nagyon kell vigyázni a bizalmi tartományok határánál.

Szoftverfejlesztés alapelvei II

- erős kohézió (kohézió = elemek összetartozásának mértéke) : Melyik programmodul (rész, osztály) mely feladatok halmazát látja el. A kohézióval kapcsolatos biztonság akkor sérül, ha a programot nem bontjuk részekre a bizalmi határnál.

Erős függőség sebezhetőségének kihasználása I

A sebezhetőség egyes részei több szerző munkája nyomán vált ismertté, de a teljes működést Chris Paget rakta össze.

- A GUI (grafikus felhasználói interfész) üzeneteit az üzenetsor (message queue) kezeli.
- Egy windows asztalhoz (desktop) egy üzenetsor tartozik
- Egy desktophoz tartozó alkalmazások ezen keresztül üzeneteket küldhetnek egymásnak is.
- A támadó által meghívott SetTimer (NULL, 0, wMsecInterval, TimerProc); függvény hatására a másik processz üzenetkezelő függvénye a WM_TIMER üzenet esetén meghívja a TimeProc eljárást. *Akkor is ha a másik processz nem foglalkozik azzal.*

Erős függőség sebezhetőségének kihasználása II

- Egy probléma van, hogy nagyobb joggal bíró processz a kisebb privilégiummal rendelkező processzhez tartozó függvényt nem tudja meghívni. Ezt például úgy lehet megoldani. hogy a támadó a clipboard-ra teszi a futtatható kódot.
- Ha az a processz, amely végrehajtja a TimeProc eljárást nagyobb joggal rendelkezik, akkor a támadó át tudja lépni a privilégium szintet.

Ahogy a gyengeség kiderült, a Microsoft átírta a WM_TIMER üzenet kezelését. Sajnos ez az alapvető problémát nem oldotta meg.

Erős függőség sebezhetőségének kihasználása III

DEMO

Windows 98 egyfelhasználós rendszerben ez a megoldás nem okoz problémát. Többfelhasználós rendszer esetén különböző privilégiummal rendelkező processzeket erősen összeköti az üzenetsor, ami sérti a helyes tervezési alapelvet.

Bizalmi viszony tranzitív tulajdonságának a kihasználása I

Solaris operációs rendszer tartalmazta az automountd programot. Cél az volt, hogy a kernel megkerülésével automatikusan ki lehessen adni mount, unmount parancsokat. Csak root user adhatja meg azokat a parancsokat, melyeket az automountd használ.

automountd tulajdonságai:

- root joggal fut.
- root joggal fájlrendszer beillesztéssel kapcsolatos parancsokat lehet végrehajtatni vele.
- Nem hallgatózik a hálózaton.
- Csak a root-tól a védett loopback interfészen jövő parancsokat fogadja el.

Bizalmi viszony tranzitív tulajdonságának a kihasználása II

rpc.statd tulajdonságai

- root joggal fut.
- Feladata az NFS szerverek ellenőrzése.
- NFS szerver állapotának a változása esetén üzenetet küld a regisztrált ügyfél programoknak.
- TCP és UDP interfészen a hálózaton hallgatózik.
- Szolgáltatás regisztráláskor a kliensnek kell megadni hogy melyik host-hoz kapcsolódjon, és a host-on belül az értesítendő RPC (Remote Procedure Call) szolgáltatás számát.

Bizalmi viszony tranzitív tulajdonságának a kihasználása III

Támadás végrehajtása:

- a rpc.statd alá beregisztrál egy lokális automountd programot.
- A támadó üzenetet küld az rpc.statd-nak, hogy az NFS szerver elszállt.
- az rpc.statd a loopback csatornán keresztül értesíti az automountd-t, és átadja az üzenetet (amit a támadó határoz meg[2],[3]).

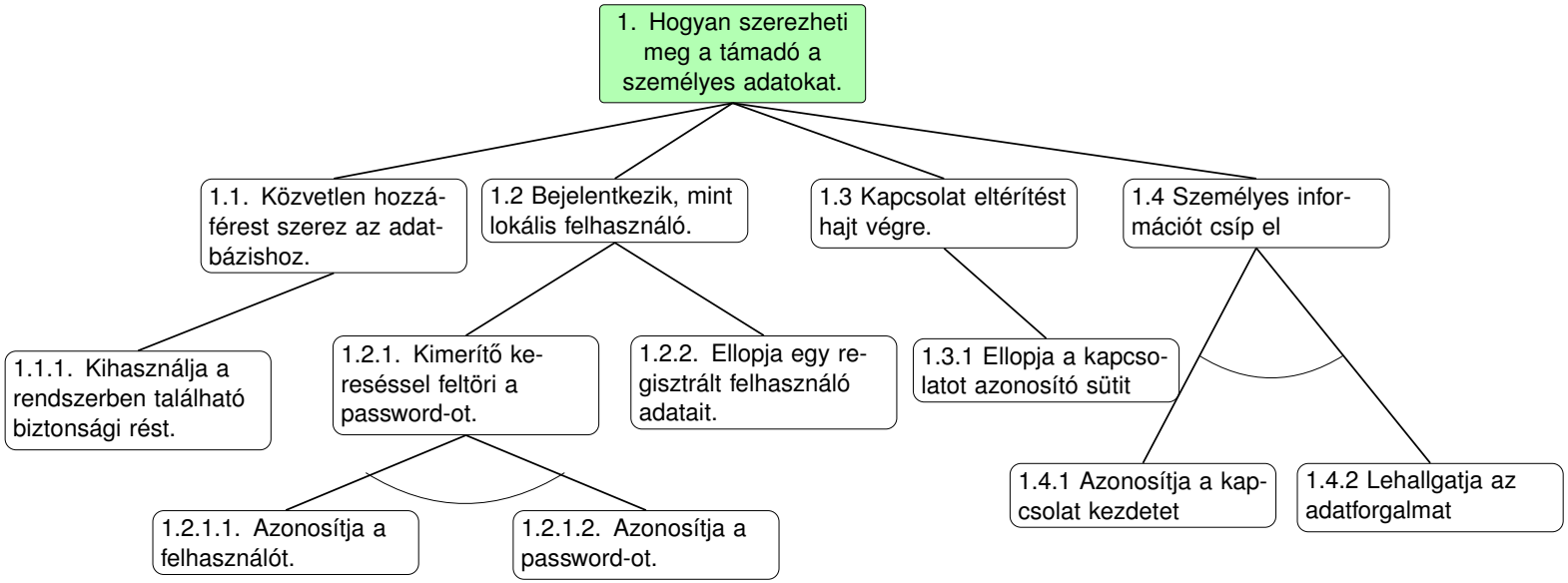
[2] Ehhez még pár trükkre szükség van.

[3] automountd nem ellenőrzi a parancsot, mivel bízik hogy a root csak megfelelőt adhat.

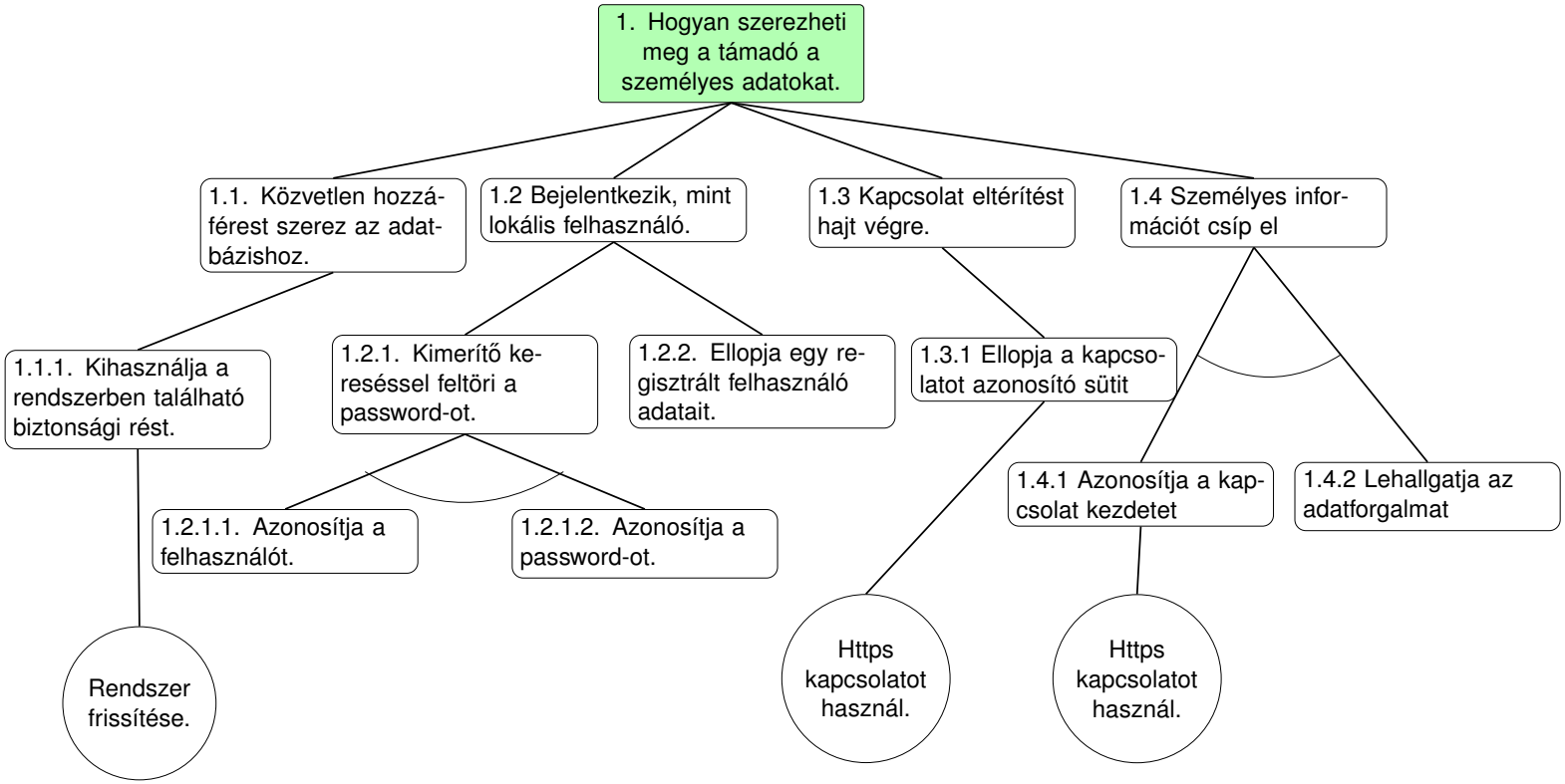
Támadási fa

- Gyökér csomópont jelenti a támadó szándékát
- A többi csomópont a lehetséges támadást ábrázolja.
- Gyerek csomópontok a támadás végrehajtásának a lehetséges módjait ábrázolják.
- Az ívvel összekötött élek esetén a támadás sikeréhez az összes összekötött gyerek csomóponttal ábrázolt tevékenységet végre kell hajtani. Ez az ÉS csomópont. Él nélküli csomópont összeköttetések VAGY kapcsolatban vannak, azaz a gyerek csomópontok közül egyetlen támadás kivitelezése esetén a szülő csomóponttal ábrázolt támadás megvalósítható.

Támadási fa


1. Hogyan szerezheti meg a támadó a személyes adatokat.
1.1. Közvetlen hozzáférest szerez az adatbázishoz.
1.2 Bejelentkezik, mint lokális felhasználó.
1.3 Kapcsolat eltérítést hajt végre.
1.4 Személyes információt csíp el
1.1.1. Kihasználja a rendszerben található biztonsági rést.
1.2.1. Kimerítő kereséssel feltöri a password-ot.
1.2.2. Ellopja egy regisztrált felhasználó adatait.
1.3.1 Ellopja a kapcsolatot azonosító sütit
1.4.1 Azonosítja a kapcsolat kezdetet
1.4.2 Lehallgatja az adatforgalmat
1.2.1.1. Azonosítja a felhasználót.
1.2.1.2. Azonosítja a password-ot.

Támadás enyhítése



Hibakezelés

Szoftvertechnológia szerint: (a feltételezés, hogy a felhasználó véletlenül rossz adatot adott meg, vagy programhiba van).

- Készítsen a program a hiba okáról egyértelmű feljegyzést, és értesítse a felhasználót az esetről.
- Ha lehetséges akkor a program önműködően javítsa ki a hibát, és folytassa a működést.

Szoftver biztonság figyelembe vételével: (a felételezés, hogy a programot extrém adatokkal támadták).

- Csak jelezze a program, hogy hiba volt, de a hiba okáról ne adjon információt. A támadó ebből vissza tud következtetni a program működésére.
- A program a hiba után lehet hogy instabil állapotba marad, ezért a programot terminálni kell.

Példa: BKK jegyadó automata hálózati kapcsolat meghibásodását jelezte ki, közben a hibát a bankkártya rossz kontaktusa okozta. Ez később derült ki egy szupermarketben.