BIZTONSÁGOS PROGRAMOZÁS, FOGALMAK AMIT TUDNI KELL

Felelősséget nem vállalok, helyenként nagyon nem értem mit másoltam :D - Alexa

- sebezhetőség

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

- rosszindulatú támadás véghezvitelére;
- érzékeny információk megváltoztatására, törlésére vagy megszerezésére;
- a számítógép rendszere felett az irányítás átvételére.

- 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

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 rendszerek 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.

- CIA

Biztonsági elvárások:

- Confidential - bizalmasság.
- Integrity - sértetlenség (integritás).
- Availability - 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

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,
- ellenőrzőösszeg,
- hibajavító kódolás,
- védett metaadatok bevezetése
    - tulajdonos,
    - létrehozási, módosítási dátumok,
    - jogok (írás, olvasás, végrehajtás).

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:

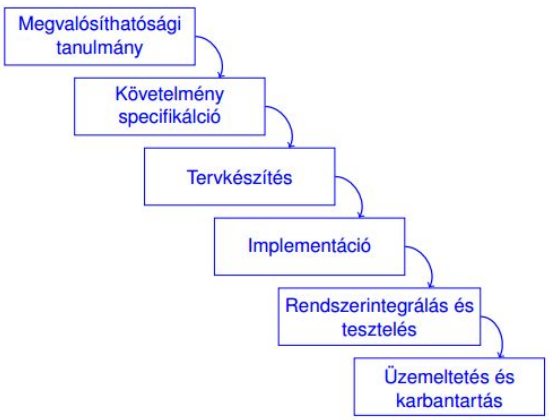
A felhasználó a szükséges adatokat és erőforrásokat minden pillanatban el tudja érni.

Biztosítása

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

- fejlesztés alatt elkövetett biztonsági hibák osztályozása

SDCL – szoftver életciklus (vízesés modell)



1. Hibás tervezés miatt létrejövő sebezhetőségek – SDLC 1,2,3 fázis.
Alapvető hiba amely az összes fázisra rányomja a bélyegét.
Költséges javítani.
Egyéb elnevezések:
   - magasszintű sebezhetőség;
   - architekturális repedés;
   - hiányos program követelmény vagy kikötés.

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.
2. Hibás implementációból adódó sebezhetőségek - SDLC 4,5 fázis.

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 mikrotervezést kell folytatni.
- A specifikáció túl bonyolult, és nincs lehetőség a felmerült kérdés tisztázására.

Implementációs biztonsági hibákat sokszor nehéz megkülönböztetni a tervezési hibáktól.

Tipikus implementációs hibák: SQL injection, buffer overflow

3. Rossz működtetésből származó sebezhetőségek – SDLC 6. fázis.

   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.

- bizalmi viszony

- 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 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.

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.

- szoftverfejlesztés alapelvek

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.

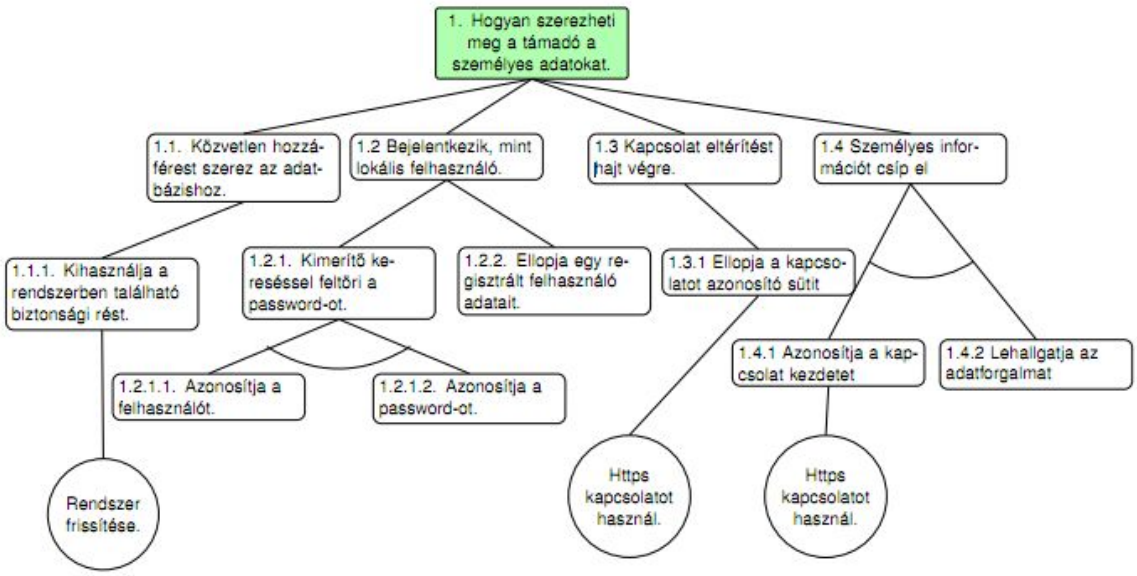
világosság: A szoftver terve lehet nagyon komplex is, de a jó terv a feladatokat szétbontja könnyen kezelhető magától értetődő részekre.

gyenge függőség : A szoftverrészek jól definiált felületen kapcsolódnak egymáshoz, az egyes részek a többi rész megváltoztatása nélkül lecserélhetőek. 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.

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.

- 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.



- hibakezelési szempontok

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 feltételezés, 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.

- hozzáférési szabályok DAC MAC

DAC: Discretionary Access Control — tetszés szerinti hozzáférési jog biztosítása:

- Minden egyes felhasználó számára külön rendelkezhetünk a hozzáférési jogokról.
- Az információ birtokosa adja a jogokat.
- A DAC-ot rugalmassága miatt az operációs rendszerek előszeretettel használják

DAC hátránya

- Nehéz globális szabályokat megkövetelni.
- Nem biztos forrásból származó program a felhasználó jogosultsága alapján megváltoztathatja a hozzáférési jogokat.
- Nem biztonságosan megírt program gyengeségeit kihasználva lehetőséget kap a támadó, a DAC szabályainak a módosítására.

MAC: Mandatory Access Control előre maghatározott hozzáférési szabályok alapján történik az információk elérése.

- Rendszer szintű adathozzáférési szabályokat, a felhasználó nem tudja megváltoztatni.
- A rendszerre van bízva a jogok hozzárendelése.

Ezt használják a hardverek.

- X86-ban a védelem részei

- **védelem tárgya**: Memória, regiszterek, I/O eszközök
- **védelem alanya:** Az azonosításhoz a felhasználó user ID vagy processz ID lenne a legalkalmasabb, de ezeket az operációs rendszer definiálja, és az alatta lévő hardver ezekről s emmit sem tud. Ezért itt a futó processz kódszegmensére fogjuk a védelmet alkalmazni.
- **tevékenység**: gépi utasítások sorozata.
- **hozzáférési szabályok:** CPU-ban előre meghatározott. (Mi a játéktere akkor az operációs rendszernek?)

Védelmi szintek (gyűrűk)
MAC terminológiában címkéket jelent.

0. szint a legmagasabb szint, bár a legkisebb érték tartozik hozzá. Általában az operációs rendszer kernel programja használja, például memória lapozás, taszkok ütemezése stb.

1. szint a nagy prioritású készülék meghajtó programok (device driver) futnak ezen a szinten.
2. szint az alacsonyabb prioritású meghajtó programok részére van fenntartva.

3. szint a felhasználói programok futtatásához.

Mandatory védelem biztosítása

- A védelem tárgyát, azaz a memóriát, regisztereket, I/O eszközöket felcímkézzük.
- A védelem alanyát azaz a kódot is felcímkézzük.
- Meg kell határozni, hogy az egyes címkével rendelkező alanyok milyen műveleteket végezhetnek a felcímkézett objektummal (angol irodalomban ez a protection state). A gyártó határozta meg, nem lehet megváltoztatni.
- Meg kell határozni, hogy milyen elvek alapján oszthatjuk ki a címkéket (angol irodalomban ez a labeling state).
- Meg kell határozni, hogy a védelem alanya vagy tárgya milyen szabályok alapján válthat címkét (angol irodalomban ez a transition state). A gyártó határozta meg, nem lehet megváltoztatni.

- Biba modell

(Információ iránya):

- **Write** a védelem alanya felől halad az objektum felé.
- **Read** a védelem tárgya felől halad a védelem alanya felé

(Biba modell) : adatok integritásának (CIA) védelmére szolgáló modellt.

- **No Write Up** A futó kód a nála magasabb privilégiummal rendelkező erőforrásokat nem módosíthatja.
- **No Read Down** A futó kód a nála alacsonyabb privilégiummal rendelkező erőforrásokat nem olvashatja.

Adatok sértetlenségének a biztosítása
Megjegyzések:

- Az első pont triviális.
- A második meggátolja, hogy a kisebb privilégiummal rendelkező erőforrás, amit a kisebb privilégiummal rendelkező kód (user) módosított, befolyásolja a nagyobb privilégiummal működő kód működését.
- Rendszerhívás esetén biztosítani kell, hogy a nagyobb prioritással futó kód olvassa a kisebb prioritású kód által átadott paramétereket, ezért ott ellenőrizni kell azok tartalmát.

- Bell-LaPadula modell

(Bell-LaPadula modell): az adatok bizalmasságának (CIA modell) a védelmére, azaz a titok kiszivárogtatásának a meggátlására szolgáló modellt.

- **No Write Down** A futó kód nála alacsonyabb privilégiummal rendelkező erőforrásokat nem módosíthatja.
- **No Read Up** A futó kód nála magasabb privilégiummal rendelkező erőforrásokat nem olvashatja.

A No Write Down meggátolja, hogy a nagyobb privilégiummal rendelkező de vírusos kód ne szivárogtathasson ki titkos információt a kisebb privilégiummal rendelkező támadó felé.
Az eredeti Bell-LaPadula modellben a kiosztott címkéket nem lehet megváltoztatni.

- X86 adathozzáférés ellenőrzése

- Kisebb privilégium szintű kód (CPL) nem fér hozzá nagyobb privilégium szinttel rendelkező adathoz (DPL). (Bell-LaPadula elv).
- Csak a DS, ES, FS, GS vagy SS regiszterek megváltoztatásánál kell erre figyelni.
- Ezeket a regisztereket csak a MOV, POP, LDS, LES, LFS, LGS, LSS utasításokkal lehet módosítani.
- Szabály: max(CPL;RPL) <=DPL.

- X86 kódhozzáférés

Miért nem hívunk meg kisebb privilégiummal rendelkező kódot?

- Könnyű átmenni az alacsonyabb privilégiummal rendelkező kódba, de nehéz onnan visszatérni.
- Áttérés után a kód nem érheti el a saját adatait, mivel a megváltozott CPL numerikus értéke nagyobb, mind a kód DPL értéke.
- Valóban szükség van erre?

Miért nem ugorhatunk nagyobb privilégiummal rendelkező kódba?

- Biztonsági okokból nem tehetjük meg.
- Valóban szükség van rá? Igen, a device drivereknél, például hogy az USB eszközt lássa a felhasználói program.
- Megoldás call gate-tel, amit a következő fejezetben tárgyalunk.

- conform / nem conform adatszegmens

Nem conform(nem alkalmazkodó) kódszegmenst csak akkor lehet meghívni, ha CPL = DPL. Ebben az esetben RPL-nek kisebb szerepe van, csak arra kell figyelni, hogy az értéke kisebb vagy egyenlő legyen a CPL-lel.

Conform tulajdonságú kódszegmens CPL >= DPL feltétel mellett meghívható. Ilyenkor az RPL nem számít. A függvény meghívása után a CPL értéke nem változik, ez az egyetlen olyan eset, hogy a CPL és az utasításokat tartalmazó szegmens DPL-je nem biztos, hogy azonos érték˝u. Nem jön létre stack váltás. Felhasználási lehetőség például a matematikai könyvtári függvények céljára.

- call-gate

Nagyobb privilégiummal rendelkező kódszegmenst, egyszerű call utasítással biztonsági okokból nem hívhatunk meg. Különböző interface-k léteznek, hogy a hívások minél ellenőrzöttebben történjenek.

- call gate (többszegmens modell esetén, rugalmas)

call gate leíró adatai:

- Meghívandó kódszegmens szelektora.
- Szegmensen belül az eljárás belépési címe.
- A hívó kódtól megkövetelt DPL privilégium szint.
- Stack váltás esetén:

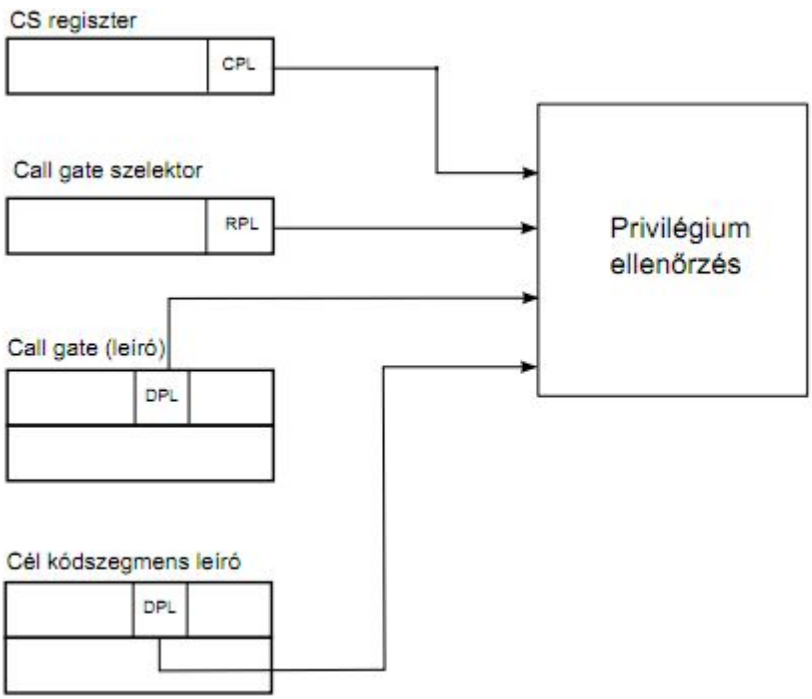
- átadandó paraméterek száma,
- stack adatának a mérete. Lehetővé teszi, hogy 32 bites alkalmazás 16 bites programot hívjon meg.

Call gate hozzáférési szabályok

- CPL <= DPL_{GATE}
- RPL <= DPL_{GATE}
- CALL utasítás végrehajtása csak a $DPL_{cél}$ <= CPL szabály esetén lehetséges,
- Nem conform szegmensbe ugrás JMP utasítással csak akkor lehetséges, ha $DPL_{cél}$ = CPL.
- Conform szegmensbe ugrás JMP utasítással csak akkor lehetséges, ha $DPL_{cél}$ <= CPL.

Miért nem megengedett a kisebb privilégiummal rendelkező függvény hívása? Azért mert akkor a visszatérésnél lennének problémák.

Call gate ellenőrzése



- Lapszintű hozzáférési szabályok

A 4 privilegium szint a következőképpen képződik le a lap U/S bitjére:

- DPL = 0,1,2 érték supervisor mód.
- DPL = 3 felhasználó mód

Típus védelem lap esetén csak írásra/olvasásra terjed ki. PAE és 64 bites lapozásnál végrehajtás letíltás (execute disable NX) is lehetséges.

- Csak olvasás van megengedve, ha R/W = 0
- Írás és olvasás is engedélyezett, ha R/W = 1
- Inicializálás miatt a supervisor minden lapot írhat és olvashat, de csak akkor, ha a CR0 regiszter WP bitje 0. Különben neki is be kell tartani a játékszabályokat.

- Supervisor módú lapokat a user módú program sem írni sem olvasni nem tudja.
- Page Directory Table, és Page Table esetén az R/W és a U/S bitek használata azonos a fent leírtakkal.

----------2.diasor----------

- általános támadási modell

Támadás megtervezése
A támadó célja a számítógép felett átvenni a felügyeletet.

Megvalósítás:

1 A támadást végrehajtó kód bejuttatása a számítógépre.

Kivitelezés:

- A támadó input ablakba beírja a gépi kódú programot.
- A támadó preparált film, kép, mp3 fájl segítségével juttatja be a kártékony programot.
- stb.

2 A támadó eltéríti a program futását, és ráadja a vezérlést a bejuttatott kódra.

3 A kártékony program célja elérése érdekében használja a feltört számítógép erőforrásához.

----------3.diasor----------

- bug – expolit

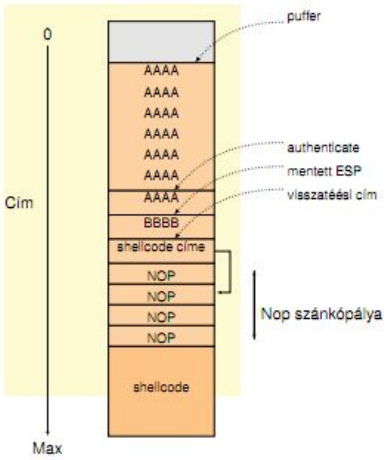
Bug = sebezhetőség. Olyan programhiba, mely megléte esetén néha a program nem a programozó által elvárt viselkedést mutatja.

Exploit = sebezhetőség kihasználása. Olyan input (beolvasott adat, környezeti változó, stb.) amellyel a támadó kihasználja a programban meglévő sebezhetőséget.

- egyszerű puffer-túlcsordulásos támadás

Fajtái:
- stack alapú (stack buffer overflow);
- heap alapú (heap buffer overflow);

Egyik esetben csak a stack keretet vagy azon belül egy területet írunk túl úgy, hogy megváltoztatunk más változók értékét is. A másik esetben a stack hely fogy el, például végtelen rekurzió esetén.



- nop szánkópálya

Címfüggőség enyhítése:

A **Nop** utasítás nem csinál semmit. Mérete 1 byte, de már vannak több byte-os nop-ok is. Eredetileg 0x90 dekódolva az **xchg eax,eax** utasítással azonos.

- jmp esp módszer

1 Keressünk olyan regisztert, amely a shellcode környékére mutat. Az ESP jelenleg pont oda mutat.

2 Keressünk valahol a memóriában lévő DLL-ekben egy jmp reg+displacement utasítást. A mi esetünkben egy jmp ESP-t kell keresni.

3 A visszatérési értéket írjuk át ezzel a címmel.

Figyelmeztetés
Csak azokat a DLL-eket vehetjük számításba, amelyeket a processz is használ.

- puffer túlcsordulás string pufferben

??? tipp

Ha a program a puffer területre stringet vár, akkor a 0x0 érték a string végét jelzi, és a shellcode további része már nem kerül beolvasásra.

Miért nem jó a C program shellcode-nak ?

- A szövegkonstansok külön szegmensben, nem a kód mellett vannak.
- A szöveg 0 értékű byte-okat tartalmaz.
- A kód is tartalmaz 0 értékű byte-ot.
- A MessageBox címét (call dword ptr ds:[00402000h]) a loader oldja fel.

Nullák kiküszöbölése:

xor eax , eax ; eax <□ 0

push eax ; Ha a stackre k e l l 0 .

Szöveg elhelyezés a stacken:

1 A szöveget 4 karakterenként fel kell bontani (32biten 4 karakter fér el).

2 Ha a string hossza nem osztható 4-gyel, akkor ki kell egészíteni például szóközzel. (Kifinomultabb módszerek is lehetnek.)

3 String végét lezáró 0 elhelyezése a stack-en (push eax) +

4-es karakter csoportok elhelyezése. A Little-Endian ábrázolás miatt a byte-okat fordított sorrendben kell letenni. push 20544948h " TIH"

4 Utolsó karakter elmentése után megjegyezzük a string kezdőcímét

mov esi , esp

???

- kanári

Kanári (más néven stack süti)

Tegyünk a stackre egy értéket, és visszatérés előtt ellenőrizzünk, hogy nem írta-e valaki felül. Ezt az értéket nevezzük kanárinak, vagy stack sütinek.

Kanári típusa:

- Konstans. Nem sokat ér.
- Terminátor. Konstans, de olyan értéket tartalmaz, amit nem lehet bevinni. Például string puffer esetén 0-át.
- Véletlen. Egy mestersüti van, de minden program indításkor más az értéke. A függvényenkénti azonos értékek gyengítheti a védelem erősségét.
- Véletlen + xor. Minden hívásnál más az értéke, mivel a véletlen értékkel és a stack címével kizáró vagy műveletet végzünk.

- SEH

rendszer szintű kivételkezelés

- Interrupt és kivételdobás események megszakítják a processzor utasítás végrehajtásnak normális folyamatát.
- Az interrupt lehet szinkron, és aszinkron. Aszinkron interrupt esetén a processzor képes az utasítás végrehajtás ismétlésére.
- A megszakítást mindig az operációs rendszer kapja el és továbbítja.

- SEH támadás

https://www.ethicalhacker.net/features/root/tutorial-seh-based-exploits-and-the-development-process

Támadás megvalósítása

Előfeltétel:

- A virtuális függvényt tartalmazó objektum a stack-en legyen.
- Stringgel tudjuk a memóriát felülírni (csak a string vége tartalmaz nullát).

Támadás megvalósítása:

- A shellcode-ot a puffer elejére helyezzük.
- Írjuk felül a virtuális függvénytáblára mutató pointert is. A stack címe Windows alatt 0-val végződik, ezért a stringet lezáró 0 pont a virtuális függvény táblára mutató pointer utolsó byte legyen (Little-Endian)
- A pufferbe hozzunk létre egy ál virtuális függvénytáblát. Hogyan tudjuk a shellcode-ot megcímezni, ha 0-val kezdődő értéket nem tudunk az ál virtuális táblába bevinni?
- Nézzük meg mely regiszterek mutatnak a stack-re. A jmp esp megoldáshoz hasonlóan keressünk megfelelő utasítás sorozatot a betöltött dll-ekben, és ezzel a címmel töltsük fel a virtuális függvénytáblát. (Csak az a cím jó amelyben nincs 0 byte).
- Várjuk, amig a program meg nem hívja a virtuális függvényt. Származtatás esetén a destruktornak illik virtuálisnak lenni.

Stack süti megkerülése

Futtassuk le a shellcode-unkat mielőtt a rendszer ellenőrizné a kanári értéket. Ez SEH exploit estén teljesül. Ez ellen a kanári nem véd. Másik lehetőség a virtuális függvénytábla átírása.

Mi van, ha a függvény nem használ kivételkezelő rutint, és nincs benne virtuális függvényt tartalmazó objektum sem. Nagy valószínűséggel a megelőző stack keretekben van kivételt regisztráló bejegyzés, amit át tudunk írni. Lásd David Litchfield: Defeating the Stack Based Buffer Overflow Prevention Mechanism of Microsoft Windows 2003 Server cikkét.

- Safe SEH

A stack sütit legegyszerűbben SEH exploit segítségével kerülhetjük meg. Akadályozzuk meg, hogy kivételdobás esetén bármely címen lévő utasítást meg lehessen hívni.

A cél, hogy csak a szabályosan regisztrált kivételkezelőket lehessen meghívni.

Visual Studioban a linkernek a /SAFESEH:yes paramétert kell megadni. Ez az alapértelmezés. Letiltása /SAFESEH:no

- DEP

Data Execution Prevention (DEP)

Korlátozzuk le a stack-et úgy, hogy ott ne lehessen kódot futtatni. Ehhez hardware támogatásra van szükség. Szegmens védelemmel ez megoldható, de akkor sérül a flat memória modell. Lap védelemmel lehet megoldani, de ez csak a későbbi Intel processzorokban lett megvalósítva (manapság már alig lehet találni olyan PC-t amelyik nincs erre felkészítve).

Első DEP-es Windows rendszerek

- Windows XP Service Pack 2;
- Windows XP Tablet PC Edition 2005;
- Windows Server 2003 Service Pack 1;
- Újabbak értelemszerűen tudják.

Üzemmódok:

- Optin futtatni lehet a nem DEP kompatibilis programokat, de egy listán meg lehet adni a melyek DEP kompatibilis módba fussanak. Önkicsomagoló exe program például futtat kódot a stacken.
- OptOut Általában a windows szerverekben az az alapértelmezés, hogy minden program DEP kompatibilis. Amelyek nem, azokat kell kivétel listára tenni.

Boot beállítások:

- AlwaysOn minden processz DEP védett, nincs kivétel.
- AlwaysOff egyik processz sem DEP védett.

- Ret2Clib

A különböző DEP beállítások miatt több függvény is van, amellyel a DEP védelmet ki lehet kapcsolni

A módszer lényege, hogy nem a kódot, hanem függvény címeket és a paramétereket tesszük a stack-re.

- A puffer túlcsordulást tartalmazó függvény visszatérésénél a ret utasítás hatására az oda készített címre kerül a vezérlés, miközben az esp értéke eggyel csökken.
- Ha a meghívott rutinnak paraméterekre van szüksége, akkor a puffer felülírás során gondoskodhatunk ezek megadásáról.
- A Windows API függvények a visszatérés előtt felszabadítják a stack-et (stdcall konvenció). Így az esp regiszter a gondosan előkészített, következő meghívandó függvényre fog mutatni. A ret hatására a következő könyvtári függvény hívódik meg. Vegyük észre, hogy sohasem fut le call utasítás, ami letenné a visszatérési címet.
- Sok esetben nem célszerű a függvény első utasítását meghívni, hanem beleugorhatunk a függvény közepébe. Nem szabad elfeledkezi arról, hogy ilyenkor is a függvény felszabadítja a stack-en használt lokális változókat. Ezért a következő meghívandó függvény címét ennek megfelelően a stacken távolabb kell elhelyezni.

ret2libc módszer korlátai

- Csak olyan standard C vagy operációs rendszer függvényeket lehet meghívni, amelyek:
  - statikusan hozzá vannak linkelve a programkódhoz, vagy
  - betöltött (használt) dll-ben lévő függvények.
- Nem minden cím vagy paraméter érték vihet ˝ o be a pufferbe, például string puffer esetén 0 értékű byte nem szerepelhet a címekben vagy paraméterek értékében.

- ROP

Return Oriented Programing

Gadget olyan gépi utasítássorozat, amely ret utasítással végződik.

- Keressünk a processz címterében gadget-eket. Erre kész alkalmazások vannak.
- A stack tartalmát állítsuk össze úgy, hogy a gadget címeket, és a gadget által pop-pal felvett értékeket tartalmazza.

A támadó nem ijed meg, ha ROP írása közben nem talál pont megfelelő gadget -et. Ha az értékes művelet és a visszatérés között van pár felesleges utasítás, ami nem csinál bajt, akkor azt is felhasználja.

Return-oriented programming (ROP) is a computer security exploit technique that allows an attacker to execute code in the presence of security defenses such as non-executable memory and code signing.[1]

In this technique, an attacker gains control of the call stack to hijack program control flow and then executes carefully chosen machine instruction sequences, called "gadgets".[2] Each gadget typically ends in a return instruction and is located in a subroutine within the existing program and/or shared library code. Chained together, these gadgets allow an attacker to perform arbitrary operations on a machine employing defenses that thwart simpler attacks.

forrás: https://en.wikipedia.org/wiki/Return-oriented_programming

DEP előtt a támadónak könnyű volt a dolga: a stack-be írta saját programkódját és annak futtatására bírta rá az operációs rendszert. A DEP megjelenése után azonban a tack tartalma már nem volt futtatható. A támadó ahhoz, hogy saját kódját futtassa már meglévő kódot kell felülírnia. A metódusok végén lévő ret utasítás visszatérési címét írja át a saját kódjának kezdőcímére. Hova csempészte be a saját kódját? Kénytelen már meglévő másik kódot felülírni, hisz az op. rendszer azokra engedélyezte a futtatást.

- ASLR

ROP program esetén gadget -eket (ret-tel végződő utasításokat) hívunk meg. Ha a kikeresett gadget címe nem fix, akkor ezt nem tehetjük meg.

Megoldás: Address Space Layout Randomization (ASLR)

ASLR esetén minden rendszerindítás után változik a processzek, modulok

- betöltési címe (image base address),
- stack kezdetének a címe,
- heap-en lefoglalt adatok címe.

Visual Studio esetén a /DYNAMICBASE linker opcióval adhatjuk meg.

A program áthelyezhetősége miatt tudnunk kell, hogy hol vannak olyan adatok, amelyek függenek a betöltés helyétől Ezt a relokációs tábla írja le. Sok program fix címre lett linkelve, így ezek áthelyezése lehetetlen.