MRLAB

İcra edilə bilən (portable executable) fayl içərisinə shellcode yerləşdirilməsi

Azərbaycan Respublikası Xüsusi Rabitə və İnformasiya Təhlükəsizliyi Dövlət Xidməti – Kompüter İnsidentlərinə qarşı Mübarizə Mərkəzi - Malware Research Lab – S. Abasov - 17 Aprel 2022

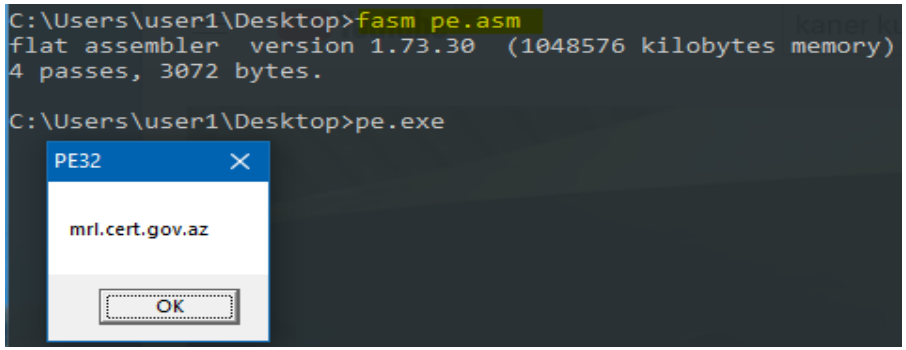
Bu məqalədə göstərilən metod tamamı ilə məlumat xarakterlidir və məqalədə bəhs edilən nümunə fayl üzərində test edilmişdir!

PE fayl formatı windows əməliyyat sisteminin ən kritik hissəsidir desək yəgin ki, çox şişirdilmiş bir fikir olmaz. Çün ki, əməliyyat sistemində icra edilə bilən fayllar (proqram təminatları, kernel sürücüləri, servislər, DLL kitabxana faylları) PE (Portable Executable) fayl formatındadır. Format haqqında daha ətraflı məlumat üçün https://docs.microsoft.com/en-us/windows/win32/debug/pe-format. Məqalədə sizlərə PE fayl içərisində shellkodu necə yerləşdirə bilərik bu haqda məlumat veriləcək. Kodumuzu yerləşdirməzdən öncə hədəf haqqında məlumat almalıyıq. Burada bizim hədəfimiz PE formatına malik icra edilə bilən fayllardır. Məqalədə kod yerləşdirmək üçün 32 bitlik PE fayldan istifadə ediləcək. Lakin oxşar metod ilə 64bit formatda olan pe fayllara da müdaxilə edə bilərsiniz. Sözü gedən əməliyyatı daha yaxşı anlamağınız üçün assembly dilində kiçik bir kod yazıb, compile edərək nümunə olaraq bu fayl üzərində işləyəcəyik. Compiler olaraq fasm (flat assembler) istifadə ediləcəkdir. Ekrana MessageBoxA funksiyası ilə kiçik bir mesaj verən bir kod.

```
format pe console
entry main
section '.text' code readable executable
main:
    push 0
    push _title
    push mesaj
    push 0
    call [MessageBoxA]
    push 0
    call [ExitProcess]
section '.data' data readable writeable

    mesaj db 'mrl.cert.gov.az', 0
    _title db 'PE32', 0

section '.idata' import data readable writeable
```

```
    dd 0,0,0,RVA kernel_name,RVA kernel_table
      dd 0,0,0,RVA user_name,RVA user_table
      dd 0,0,0,0,0

      kernel_table:
        ExitProcess dd RVA _ExitProcess
        dd 0
      user_table:
        MessageBoxA dd RVA _MessageBoxA
        dd 0

      kernel_name db 'KERNEL32.DLL',0
      user_name db 'USER32.DLL',0

      _ExitProcess dw 0
        db 'ExitProcess',0
      _MessageBoxA dw 0
        db 'MessageBoxA',0

section '.reloc' fixups data readable discardable
```

Kodumuzu yazdıqdan sonra compile edib işə salırıq.

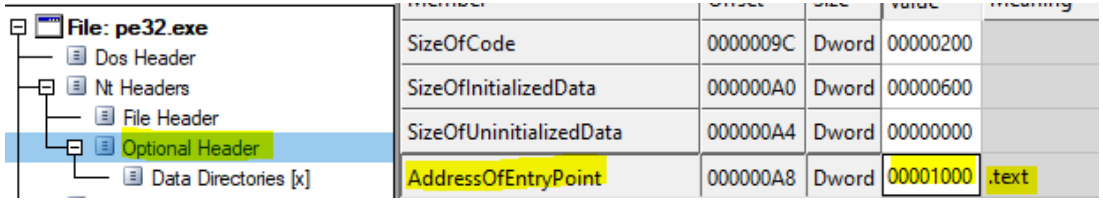


Burada .text bölməsində icra edilən kodlar yazılıb. PELoader pe faylını işə salarkən **AddressOfEntryPoint** dəyərinə baxaraq **ImageBase** üzərinə **AddressOfEntrPoint** dəyərini qoyur və EIP (Extended Instruction Pointer) registerinə bu adresi yazır və kodlar bu adresdən başlayaraq icra edilir. **AdressOfEntryPoint** isə .text bölməsində yerləşən kodların başlanğıc adresinə işarə edir. MessageBoxA və ExitProcess funksiyalarının çağrıla bilməsi üçün isə .idata bölməsi lazımı kitabxana funksiyalarını import edir. Kodları disasm edərək daha yaxşı anlaya bilərik.

```
public start
start proc near
push    0                   ; uType
push    offset Caption      ; "PE32"
push    offset Text         ; "mrl.cert.gov.az"
push    0                   ; hWnd
call    ds:MessageBoxA
push    0                   ; uExitCode
call    ds:ExitProcess
start endp
```

```
000000000040303C        ExitProcess     KERNEL32
0000000000403044        MessageBoxA     USER32
```

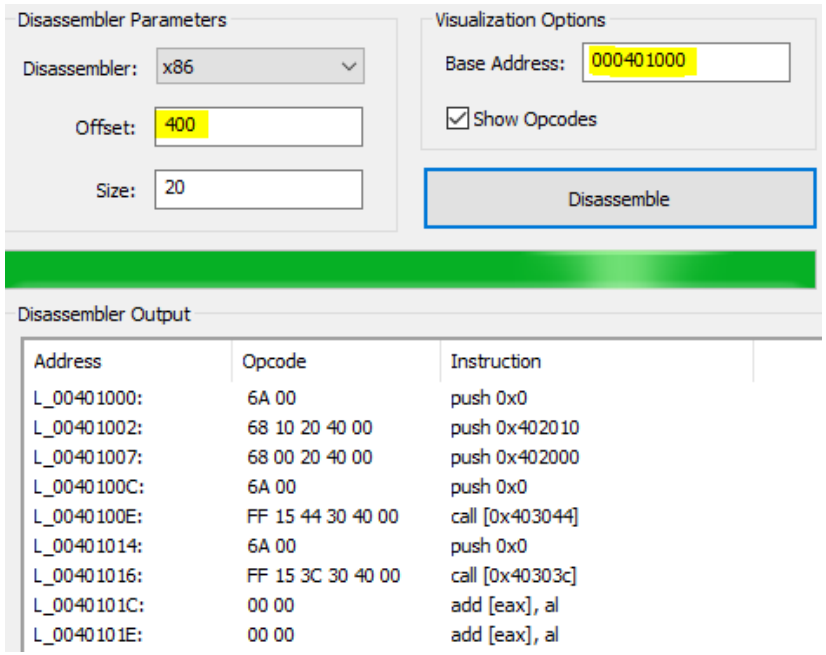
Flat assembleri seçməyimin əsas məqsədi digər compilerlərdən fərqli olaraq faylın içərisinə əlavə heç bir kod yazmamasıdır. Bu shellkod yerləşdirmə zamanı bizim əlimizi rahatlaşdıracaq. Kodun işlədiyindən əmin olduqdan sonra gəlin PE formatına diqqət yetirək. PE formatında bizə ilkin olaraq lazım olan dəyər AddressOfEntryPoint dəyəridir. Bu field NT Header-OptionalHeader bölməsində yerləşir.



Hex editor ilə burada nə olduğunu görmək üçün 0x00001000 adresinə gətməyimiz lazımdır. P.S Bu adress pe içərisində raw offset olaraq deyil RVA (relative virtual address) olaraq saxlanılır. Yəni fayl açıb bu offsetə getmək istədiyiniz zaman bu sizi kodların olduğu offsetə aparmayacaqdır. Bunun üçün rva adresi file offset -ə convert edərək hex editor ilə baxırıq.

```
6A 00 68 10 20 40 00 68 00 20 40 00 6A 00 FF 15   j.h. @.h. @.j.ÿ.
44 30 40 00 6A 00 FF 15 3C 30 40 00 00 00 00 00   D0@.j.ÿ.<0@.....
```

Şəkildə gördüyünüz baytlar əslində bizim yazdığımız .text bölməsində olan kodlardır.



Burada sırası ilə **0x402010** "PE32" mətninin olduğu adresə pointerdir. **0x402000** "mrl.cert.gov.az" mətninin olduğu adresə pointerdir. **0x403044 user32.MessageBoxA** funksiyasına, **0x40303c** isə **kernel32.ExitProcess** funksiyasına pointerdir. Kodların necə icra edildiyi haqqında məlumat aldıqdan sonra keçirik öz shell kodumuzu hədəf faylının içərisinə yerləşdirilməsi prosesinə. Burada önəmli olan məqam yerləşdirilən shell kodumuzun icrasından sonra proqramın normal fəaliyyətinə davam etməsidir. Yəni bizim shellkodumuz icra edildikdən sonra proqram orijinal kodlarını icra etməlidir. Bunun üçün shellkodumuza diqqət ilə yazmalıyıq.

Hədəf fayl içərisinə yerləşdiriləcək shell kodun hazırlanması.

Nomral flow zamanı proqramın ekrana MessageBoxA funksiyası ilə mesaj verdiyini gördünüz. Bizim yazdığımız shellkodu eyni şəkildə ekrana MessageBoxA funksiyası ilə başqa bir mesaj göstərdikdən sonra proqram orijinal variantda olduğu kimi "mrl.cert.gov.az" mesajını verməlidir. Bunu qarşı tərəfin diqqətini çəkməmək üçün mütləqdir. Bunun üçün shellkodumuz içərisində orijinal kodların olduğu adresi saxlamalıyıq. Yəni bizim shellkodumuz icra edildikdən sonra EIP -i orijinal kodların olduğu adresə yönləndirməlidir. Bunu test etmək üçün pe faylı içərisində yeni bir bölmə yaradaraq içərisinə bizi orijinal kodların olduğu bölməyə yönləndirəcək kodu

yazırıq və test edirik. Yeni yaradılacaq bölməyə lazımı bölmə bayraqlarını(executable, read, write, code) set etdikdən sonra yazığımız shellkodu bu bölməyə copy-paste edirik.

```
use32

call tmp
tmp:
    pop  eax
    sub  eax, 0x5000
    sub  eax, 0x5
    add  eax,  0x1000
    jmp  eax
```

```
E8 00 00 00 00 58 2D 00 50 00 00 83 E8 05 05 00   è....X-.P..▮è▯▯.
10 00 00 FF E0 00 00 00 00 00 00 00 00 00 00 00   ▯..ÿà...........
```

Aşağıda ki, məlumatlar bizə lazım olacaqdır.

Orijinal EP = 0x00001000

Shell section virtual address = 0x00005000

Bu məlumatları götürdükdən sonra orijinal EP yə jump edəcək kiçik bir shell kod yazaraq və test edirik. Lakin burada kiçik bir problemimiz var. Orijinal kodlar ilə bizə yeni yaratdığımız shellkodlar başqa bölmədə olduğu üçün bu bölməyə jump etməyimiz üçün kiçik bir metod istifadə etməliyik. Birbaşa EIP registerinə müdaxilə edə bilməyəcəyimiz üçün bizə ilk öncə cari bölməmizin base adresi lazımdır. Bu adresi call instruction ilə öyrənirik. Call tmp ilə tmp sətrinə call edirik.

Daha sonra pop eax instruction ilə stack üzərindən cari bölmənin base adresini + 5(call instruction ölçüsü) bayt olaraq əldə edirik. Daha sonra shell bölməsinin virtual adresini və call instruction ölçüsünü əldə etdiyimiz adres dəyərindən çıxırıq. Əldə etdiyimiz dəyər bizə faylın load adresini (imagebase) verir. Bundan sonra isə bu dəyərin üzərinə orijinal kodların olduğu EP (entrypoint) dəyərini əlavə edirik və jmp instruction ilə bu adresə jump edərək orijinal kodların icra olunmasını təmin edirik.

```
call <pe32.sub_5E5005>
pop eax
sub eax,5000
sub eax,5
add eax,1000
jmp eax
```

```
EAX   005E1000   pe32.005E1000
EBX   01160000
ECX   005E5000   <pe32.EntryPoint>
EDX   005E5000   <pe32.EntryPoint>
```

```
push 0
push pe32.2F2010
push pe32.2F2000
push 0
call dword ptr ds:[<&MessageBoxA>]
push 0
call dword ptr ds:[<&ExitProcess>]
```

Kodların işlədiyininə əmin olduqdan sonra keçirik əsas shellkodumuzu yazmağa. Məqalənin başında qeyd edildiyi kimi testlər yalnız nümunə faylı üzərində həyata keçirilmişdir. Buna görə sistemə, hədəf faylına görə shellkod üzərində dəyişiklik etmək vacibdir. Lakin məqalədə əsas məqsədimiz sizi bu tip metodlar ilə tanış etmək olduğu üçün ümumi(generic shellcoding) metodlardan istifadə edilməyəcəkdir. İlk öncə nələr etməliyik ona baxaq. Bizə lazım olan orijinal kodların bizə göstərdiyi (MessageBoxA funksiyası üzərindən) mesajdan öncə **kernel32.Beep** funksiyası ilə kiçik bir alert(səs) verərək daha sonra isə orijinal EP -ə tullanmaq. Əlbətdə Windows XP əməliyyat sistemində statik adreslərdən istifadə edərək daha rahat shellkod yaza bilərdik. Lakin ASLR xüsusiyyəti ilə birlikdə artıq bu metodun qarşısı alındığına görə bizə dinamik olaraq funksiya adreslərini öyrənib shellkodumuzu icra etmək lazımdır. Bunun üçün PEB (Process Environment Block) müraciət etmək məcburiyyətindəyik. Daha öncəki məqalələrimizdən bildiyiniz kimi prosesin load etdiyi modullar PEB stukturuna saxlanılır. Bu struktur üzərindən load edilən modullar arasında kernel32 modulunu axtarıb tapdıqdan sonra kitabxananın export tablosunda Beep funksiyasının adresini öyrənərək shellkodumuzu icra etməliyik.

İlk öncə debuggər ilə hədəf faylımız load etdiyi modulların siyahısına baxırıq.

```
ModLoad: 008e0000 008e6000   image008e0000
ModLoad: 77360000 77503000   ntdll.dll
ModLoad: 77100000 771f0000   C:\WINDOWS\SysWOW64\KERNEL32.DLL
```

```
0:000> dt _LDR_DATA_TABLE_ENTRY 0x14f3a68
ntdll!_LDR_DATA_TABLE_ENTRY
   +0x000 InLoadOrderLinks : _LIST_ENTRY [ 0x14f3e48 - 0x14
   +0x008 InMemoryOrderLinks : _LIST_ENTRY [ 0x77485d9c - 0x
   +0x010 InInitializationOrderLinks : _LIST_ENTRY [ 0x7710
   +0x018 DllBase          : 0x000f0000 Void
   +0x01c EntryPoint       : 0x00420040 Void
   +0x020 SizeOfImage      : 0x14f3b70
   +0x024 FullDllName      : _UNICODE_STRING "KERNEL32.DLL"
```

Burada load edilən 3. modul bizə lazım olan **kernel32.dll** moduludur. **InLoadOrderLinks** listi üzərində gəzərək 3. modulu (kernel32) parse edərək export tablosundan **Beep** funksiyasının adresini götürüb call etməliyik.

İlk öncə **PEB.Ldr** üzərindən kernel32.dll adresini götürməliyik.

InLoadOrderLinks baxsaq ardıcıl olaraq hədəf fayl , **ntdll**, **kernel32** modullalarının siyahısının saxlandığını görə bilərik.

PEB + 0xC bizə Ldr adresini verir. Bu adresin üzərinə 0xC əlavə etdiyimiz zaman isə bizə InLoadOrderLinks adresini verəcəkdir.

```
0:000> dx -r1 ((ntdll!_LIST_ENTRY *)0xf43560)
((ntdll!_LIST_ENTRY *)0xf43560)                 : 0xf43560 [Type: _LIST_ENTRY *]
    [+0x000] Flink            : 0xf43a60 [Type: _LIST_ENTRY *]
    [+0x004] Blink            : 0xf43670 [Type: _LIST_ENTRY *]
```

Burada LIST_ENTRY struktunun 3. elementi kernel32.dll haqqında məlumatların olduğu **LDR_DATA_TABLE_ENTRY** adresdir (**0xf43a60**).

```
0:000> dt _LDR_DATA_TABLE_ENTRY 0xf43a60
ntdll!_LDR_DATA_TABLE_ENTRY
   +0x000 InLoadOrderLinks : _LIST_ENTRY [ 0xf43e40 - 0xf43560 ]
   +0x008 InMemoryOrderLinks : _LIST_ENTRY [ 0xf43e48 - 0xf43568 ]
   +0x010 InInitializationOrderLinks : _LIST_ENTRY [ 0x77485d9c - 0xf43e50 ]
   +0x018 DllBase          : 0x77100000 Void
   +0x01c EntryPoint       : 0x7711f640 Void
   +0x020 SizeOfImage      : 0xf0000
   +0x024 FullDllName      : _UNICODE_STRING "C:\WINDOWS\System32\KERNEL32.DLL"
   +0x02c BaseDllName      : _UNICODE_STRING "KERNEL32.DLL"
```

Bu adresin üzərinə 0x18 gəldiyimiz zaman isə bizə kernel32.dll load adresini verir. Geriyə qalan kernel32.dll export tablosunu parse edib Beep funksiyasını tapmaq qalır.

Kernel32.dll load adresini götürən asm kodlar.

```
xor eax, eax
mov eax, [fs:0x30] ; PEB adresi
mov eax, [eax + 0xc] ;Ldr
mov eax, [eax + 0xc] ; InLoadOrderModuleList
mov eax, [eax]
mov eax, [eax]
mov eax, [eax + 0x018] ;kernel32.dll load adresi
```

```
  660000 62651224 Apr 24 13:02:28 2022 C:\Users\user1\Desktop\beep.exe
77360000 C:\WINDOWS\SYSTEM32\ntdll.dll
77100000 C:\WINDOWS\System32\KERNEL32.DLL
```

```
0:000> !dlls -c 0x77100000

0x00f43a60: C:\WINDOWS\System32\KERNEL32.DLL
      Base   0x77100000  EntryPoint  0x7711f640  Size        0x000f0000    DdagNode     0x00f43b28
      Flags  0x000ca2cc  TlsIndex    0x00000000  LoadCount   0xffffffff    NodeRefCount 0x00000000
             <unknown>
             LDRP_LOAD_NOTIFICATIONS_SENT
             LDRP_IMAGE_DLL
             LDRP_DONT_CALL_FOR_THREADS
             LDRP_PROCESS_ATTACH_CALLED
```

Export tablosuna getmək üçün dinamik olaraqda kod yaza bilerik. Lakin məqalənin uzanmaması üçün statik olaraq davam edirik. Kernel32.dll (Windows 10 x64) 0x170 offsetində export qovluğunun(**IMAGE_EXPORT_DIRECTORY**) adresi saxlanılır. Bu qovluqda isə 9. field export edilən funksiyaların adreslərinin saxlanıldığı bölgəyə pointerdir.

```
0:000> dd 0x77100000 + 0x170
77100170  00092d30 0000dc14 000a0944 00000780
77100180  000d0000 00000520 00000000 00000000
77100190  00099000 00003b38 000e0000 00004894
771001a0  00085340 00000070 00000000 00000000
771001b0  00000000 00000000 00000000 00000000
771001c0  00080138 000000ac 00000000 00000000
771001d0  00080b50 000014dc 00092b5c 00000060
771001e0  00000000 00000000 00000000 00000000
0:000> dd 0x77100000 + 00092d30
77192d30  00000000 bf2d9a47 00000000 00096c1e
77192d40  00000001 00000647 00000647 00092d58
77192d50  00094674 00095f90 0001fa10 00096c58
77192d60  00082034 00096ca2 00096cd8 00020ac0
77192d70  00020400 000195a0 0001b8d0 00023c10
77192d80  00023c20 00096d5e 00035fc0 00052e00
77192d90  00052e60 00033710 00020f40 00033720
77192da0  00019830 00033740 00031fd0 00096e97
```

Məqaləni qısa tutmaq üçün bu adresi deyil export edilən funksiya adlarının olduğu bölgəyə pointer verən 0x20 offsetində olan **AddressOfNames** fieldinə müraciət edəcəyik. Burada tək-tək funksiya adlarını yoxlayaraq Beep stringinə rast gəldiyimiz index ilə

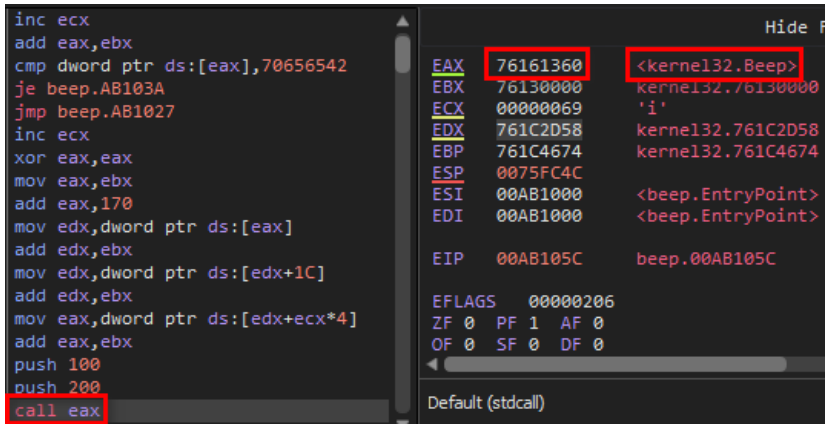
AddressOfFunctions bölgəsində index-ə qarşılıq gələn funksiyanı çağıracağıq. Ilk olaraq aşağıda ki, kodlar ilə Beep funksiyasının AddressOfFunctions içərisində yerini (index) alırıq.

```
main:
    xor eax, eax
    mov eax, [fs:0x30] ; PEB adresi
    mov eax, [eax + 0x00c] ;Ldr
    mov eax, [eax + 0xc] ; InLoadOrderModuleList
    mov eax, [eax]
    mov eax, [eax]
    mov eax, [eax + 0x018] ;kernel32.dll load adresi
    mov ebx, eax
    ;ebx =  kernel32.dll load adresi
    add eax, 0x170
    mov edx, ebx
    add edx, [eax] ; edx = kernel32.ExportDir adres
    mov edx, [edx + 0x20]
    add edx, ebx ;edx = pointer-> NameOfFunctions
    xor ecx, ecx
find_beep:
    mov ebp, edx
    mov eax, [ebp + ecx * 4]
    inc ecx
    add eax, ebx
    cmp dword [eax], 0x70656542 ;0x70656542 = 'Beep'
    jz  founded
    jmp find_beep
```

Indexi tapdıqdan sonra isə AdressOfFunctions siyahısı içərisində həmin index -ə qarşılıq gələn funksiya adresini götürüb call edirik.

```
founded:
    inc ecx
    xor eax, eax
    mov eax, ebx
    add eax, 0x170
    mov edx, [eax]
    add edx, ebx
    mov edx, [edx + 0x1c]
    add edx, ebx
    mov eax, [edx + ecx * 4]
    add eax, ebx
```

```
    push 0x100
    push 0x200
    call eax
```



Daha sonra shell kodumuzu .shell bölməsinə yazırıq ve test edirik. Nəticə müsbətdir.

```
E8 00 00 00 00 31 C0 64 A1 30 00 00 00 8B 40 0C
8B 40 0C 8B 00 8B 00 8B 40 18 89 C3 05 70 01 00
00 89 DA 03 10 8B 52 20 01 DA 31 C9 89 D5 8B 44
8D 00 41 01 D8 81 38 42 65 65 70 74 02 EB ED 41
31 C0 89 D8 05 70 01 00 00 8B 10 01 DA 8B 52 1C
01 DA 8B 04 8A 01 D8 68 00 01 00 00 68 00 02 00
00 FF D0 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Geriyə qalır shell kodun icrasından sonra orijinal kodların olduğu adresə tullanmaqdır. Bunu əməliyyatı həyata keçirən assembly kodlar: *P.S Kodlar optimizasiya edilməmişdir və yalnız məqalədə yer alan hədəf fayl üzərində test edilmişdir*.

```
use32
call main
main:
    xor eax, eax
    mov eax, [fs:0x30] ; PEB adresi
    mov eax, [eax + 0x00c] ;Ldr
    mov eax, [eax + 0xc] ; InLoadOrderModuleList
    mov eax, [eax]
```

```
    mov eax, [eax]
    mov eax, [eax + 0x018] ;kernel32.dll load adresi
    mov ebx, eax
    ;ebx =  kernel32.dll load adresi
    add eax, 0x170
    mov edx, ebx
    add edx, [eax] ; edx = kernel32.ExportDir adres
    mov edx, [edx + 0x20]
    add edx, ebx ;edx = pointer-> NameOfFunctions
    xor ecx, ecx
find_beep:
    mov ebp, edx
    mov eax, [ebp + ecx * 4]
    inc ecx
    add eax, ebx
    cmp dword [eax], 0x70656542 ;0x70656542 = 'Beep'
    jz  founded
    jmp find_beep
founded:
    inc ecx
    xor eax, eax
    mov eax, ebx
    add eax, 0x170
    mov edx, [eax]
    add edx, ebx
    mov edx, [edx + 0x1c]
    add edx, ebx
    mov eax, [edx + ecx * 4]
    add eax, ebx
    push 0x100
    push 0x200
    call eax
    call oep
oep:
    pop eax
    sub eax, 0x5000
    sub eax, 0x5
    sub eax, 0x63
    add eax, 0x1000
    jmp eax
```

Fasm compiler ilə shellkodu compile edib hədəf faylın .**shell** bölməsinə yazıb icra etdikdə ilk olaraq **Beep** funksiyası ilə səs daha sonra **MessageBoxA** funksiyası ilə isə mesaj verildi.

İstinadlar

https://flatassembler.net/docs.php
https://docs.microsoft.com/en-us/windows/win32/debug/pe-format
https://0xrick.github.io/win-internals/pe2/
https://docs.microsoft.com/en-us/windows/win32/api/winternl/ns-winternl-peb