

Forum:[Hack Forums](#)

Thread: [DLL Injector](#) | [FASAM/C++](#) | [x86](#) | [Bypass AC](#) | [ManualMap](#) | [Thread Hijacking](#)

RCE - January 12, 2020, 7:58 am

This FASAM injector used to inject DLL files into the game process using 2 loading methods, manual map, and load library.

this will be a reference for me and for other people who are interesting with game cheats.

```
Code          format PE GUI 4.0
              entry main
```

```
include 'FS/INCLUDE/win32ax.inc'
```

```
struct PROCESSENTRY32
    dwSize          dd ?
    cntUsage         dd ?
    th32ProcessID    dd ?
    th32DefaultHeapID dd ?
    th32ModuleID     dd ?
    cntThreads       dd ?
    th32ParentProcessID dd ?
    pcPriClassBase   dd ?
    dwFlags          dd ?
    szExeFile        dw MAX_PATH dup (?)
ends
```

```
struct LARGE_INTEGER
    LowPart          dd ?
    HighPart         dd ?
ends
```

```
struct IMAGE_DOS_HEADER
    e_magic          dw ?
    e_cblp           dw ?
    e_cp             dw ?
    e_crlc           dw ?
    e_cparhdr        dw ?
    e_minalloc       dw ?
    e_maxalloc       dw ?
    e_ss            dw ?
    e_sp            dw ?
    e_csum           dw ?
    e_ip            dw ?
    e_cs            dw ?
    e_lfarlc         dw ?
    e_ovno           dw ?
    e_res            rw 4
    e_oemid          dw ?
```

```

    e_oeminfo          dw ?
    e_res2             rw 10
    e_lfanew           dd ?
ends

```

```

struct IMAGE_FILE_HEADER
    Machine            dw ?
    NumberOfSections   dw ?
    TimeDateStamp      dd ?
    PointerToSymbolTable dd ?
    NumberOfSymbols     dd ?
    SizeOfOptionalHeader dw ?
    Characteristics    dw ?
ends

```

```

struct IMAGE_DATA_DIRECTORY
    VirtualAddress     dd ?
    Size               dd ?
ends

```

```

struct IMAGE_OPTIONAL_HEADER
    Magic              dw ?
    MajorLinkerVersion db ?
    MinorLinkerVersion db ?
    SizeOfCode         dd ?
    SizeOfInitializedData dd ?
    SizeOfUninitializedData dd ?
    AddressOfEntryPoint dd ?
    BaseOfCode         dd ?
    BaseOfData         dd ?
    ImageBase          dd ?
    SectionAlignment   dd ?
    FileAlignment      dd ?
    MajorOperatingSystemVersion dw ?
    MinorOperatingSystemVersion dw ?
    MajorImageVersion  dw ?
    MinorImageVersion  dw ?
    MajorSubsystemVersion dw ?
    MinorSubsystemVersion dw ?
    Win32VersionValue  dd ?
    SizeOfImage        dd ?
    SizeOfHeaders       dd ?
    CheckSum           dd ?
    Subsystem          dw ?
    DllCharacteristics  dw ?
    SizeOfStackReserve dd ?
    SizeOfStackCommit  dd ?
    SizeOfHeapReserve  dd ?
    SizeOfHeapCommit   dd ?
    LoaderFlags        dd ?
    NumberOfRvaAndSizes dd ?
    DataDirectory       db 16 * sizeof.IMAGE_DATA_DIRECTORY dup ?
ends

```

```

struct IMAGE_NT_HEADERS
    Signature      dd ?
    FileHeader     IMAGE_FILE_HEADER
    OptionalHeader IMAGE_OPTIONAL_HEADER
ends

```

```

struct IMAGE_SECTION_HEADER
    Name          rb 8
    VirtualSize   dd ?
    VirtualAddress dd ?
    SizeOfRawData dd ?
    PointerToRawData dd ?
    PointerToRelocations dd ?
    PointerToLinenumbers dd ?
    NumberOfRelocations dw ?
    NumberOfLinenumbers dw ?
    Characteristics dd ?
ends

```

```

struct IMAGE_BASE_RELOCATION
    VirtualAddress dd ?
    SizeOfBlock    dd ?
ends

```

section '.text' code executable

```

proc main
    local pid:DWORD, path:rb MAX_PATH

    cinvoke __getmainargs, argc, argv, env, 0
    cmp [argc], 4
    jne .wrongArgumentCount
    mov esi, [argv]
    lea eax, [path]
    invoke GetFullPathNameA, dword [esi + 8], MAX_PATH, eax, 0
    mov esi, [argv]
    stdcall findProcessId, dword [esi + 12]
    mov [pid], eax

    cinvoke strcmp, dword [esi + 4], <'-loadlibrary', 0>
    test eax, eax
    jnz @f
    lea eax, [path]
    stdcall loadlibrary, eax, [pid]
    @@:
    mov esi, [argv]
    cinvoke strcmp, dword [esi + 4], <'-manual-map', 0>
    test eax, eax
    jnz .wrongInjectionMethod
    lea eax, [path]
    stdcall manualmap, eax, [pid]
.wrongInjectionMethod:

```

```
    stdcall criticalError, <'Wrong injection method!', 0>
.wrongArgumentCount:
    stdcall criticalError, <'Wrong amount of command line arguments!', 0>
```

endp

```
proc findProcessId, name
    local snapshot:DWORD, processEntry:PROCESSENTRY32
```

```
    invoke CreateToolhelp32Snapshot, 0x2, 0
    mov [snapshot], eax
    mov [processEntry.dwSize], sizeof.PROCESSENTRY32
    lea eax, [processEntry]
    invoke Process32First, [snapshot], eax
    test eax, eax
    jz .error
```

```
@@:
    lea eax, [processEntry.szExeFile]
    cinvoke strcmp, eax, [name]
    test eax, eax
    jz .return
    lea eax, [processEntry]
    invoke Process32Next, [snapshot], eax
    test eax, eax
    jnz @b
```

```
.error:
    xor eax, eax
    ret
```

```
.return:
    mov eax, [processEntry.th32ProcessID]
    ret
```

endp

```
proc loadlibrary, path, pid
    local handle:DWORD, allocatedMemory:DWORD, pathLength:DWORD
```

```
    invoke OpenProcess, PROCESS_VM_WRITE + PROCESS_VM_OPERATION +
PROCESS_CREATE_THREAD, FALSE, [pid]
    mov [handle], eax
    cinvoke strlen, [path]
    inc eax
    mov [pathLength], eax
    invoke VirtualAllocEx, [handle], NULL, eax, MEM_COMMIT + MEM_RESERVE,
PAGE_READWRITE
    mov [allocatedMemory], eax
    invoke WriteProcessMemory, [handle], [allocatedMemory], [path], [pathLength], NULL
    invoke CreateRemoteThread, [handle], NULL, 0, [LoadLibraryA], [allocatedMemory], 0, NULL
    invoke WaitForSingleObject, eax, 0xFFFFFFFF
    invoke VirtualFreeEx, [handle], [allocatedMemory], [pathLength], MEM_RELEASE
    invoke CloseHandle, [handle]
```

```
    invoke ExitProcess, 0
endp
```

```
proc manualmap, path, pid
    cinvoke manualMap, [path], [pid]
    invoke ExitProcess, 0
endp
```

```
proc criticalError, message
    cinvoke printf, <'Critical Error: %s', 0>, [message]
    invoke ExitProcess, 0
endp
```

```
struct LoaderData
    allocationBase      dd ?
    loadLibraryA        dd ?
    getProcAddress      dd ?
    rtlZeroMemory       dd ?
    imageBase           dd ?
    relocVirtualAddress dd ?
    importVirtualAddress dd ?
    addressOfEntryPoint dd ?
ends
```

```
proc loadImage, data
    local currentReloc:DWORD, delta:DWORD, relocInfo:DWORD

    mov eax, [data]
    stdcall [eax + LoaderData.loadLibraryA], <'user32.dll', 0>
    mov ebx, [data]
    stdcall [ebx + LoaderData.getProcAddress], eax, <'MessageBoxA', 0>
    stdcall eax, 0, <'Demo', 0>, <'It works!', 0>, MB_OK

    mov eax, [data]
    mov ebx, [eax + LoaderData.relocVirtualAddress]
    add ebx, [eax + LoaderData.allocationBase]
    mov [currentReloc], ebx
    mov ebx, [eax + LoaderData.allocationBase]
    sub ebx, [eax + LoaderData.imageBase]
    mov [delta], ebx

    ret
endp
loadImageSize dd $ - loadImage
```

```
proc manualmap_2, path, pid
    local handle:DWORD, fileSize:LARGE_INTEGER, imageMemory:DWORD,
heapHandle:DWORD, ntHeaders:DWORD, loaderData:LoaderData, loaderMemory:DWORD

    invoke CreateFileA, [path], GENERIC_READ, 0, NULL, OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL, NULL
    mov [fileHandle], eax
    lea ebx, [fileSize]
```

```
invoke GetFileSizeEx, eax, ebx
cinvoke printf, <'File size: %d', 10, 0>, [fileSize.LowPart]
```

```
invoke GetProcessHeap
test eax, eax
jz heapFail
mov [heapHandle], eax
cinvoke printf, <'Heap handle: %p', 10, 0>, [heapHandle]
```

```
invoke HeapAlloc, [heapHandle], 0, [fileSize.LowPart]
test eax, eax
jz heapAllocFail
mov [heapMemory], eax
cinvoke printf, <'Heap memory: %p', 10, 0>, eax
```

```
invoke ReadFile, [fileHandle], [heapMemory], [fileSize.LowPart], NULL, NULL
test eax, eax
jz readFileFail
cinvoke printf, <'ReadFile: %d', 10, 0>, eax
```

```
mov eax, [heapMemory]
xor ebx, ebx
mov bx, [eax + IMAGE_DOS_HEADER.e_magic]
cinvoke printf, <'DOS SIGNATURE: 0x%X', 10, 0>, ebx
```

```
mov eax, [heapMemory]
add eax, [eax + IMAGE_DOS_HEADER.e_lfanew]
mov [ntHeaders], eax
cinvoke printf, <'Size of Image: %d', 10, 0>, [eax +
IMAGE_NT_HEADERS.OptionalHeader.SizeOfImage]
```

```
invoke OpenProcess, PROCESS_VM_WRITE + PROCESS_VM_OPERATION +
PROCESS_CREATE_THREAD, FALSE, [pid]
test eax, eax
jz openProcessFail
mov [handle], eax
cinvoke printf, <'Process handle: %p', 10, 0>, eax
```

```
mov eax, [ntHeaders]
invoke VirtualAllocEx, [handle], NULL, [eax +
IMAGE_NT_HEADERS.OptionalHeader.SizeOfImage], MEM_COMMIT + MEM_RESERVE,
PAGE_EXECUTE_READWRITE
mov [imageMemory], eax
cinvoke printf, <'Image memory: %p', 10, 0>, eax
```

```
mov eax, [ntHeaders]
xor ebx, ebx
mov bx, [eax + IMAGE_NT_HEADERS.FileHeader.NumberOfSections]
```

```
mov ecx, ebx
@@:
push ecx
dec ecx
```

```

mov ebx, [ntHeaders]
add ebx, sizeof.IMAGE_NT_HEADERS
mov eax, sizeof.IMAGE_SECTION_HEADER
mul ecx
add ebx, eax

mov edx, [imageMemory]
add edx, [ebx + IMAGE_SECTION_HEADER.VirtualAddress]
mov eax, [heapMemory]
add eax, [ebx + IMAGE_SECTION_HEADER.PointerToRawData]

invoke WriteProcessMemory, [handle], edx, eax, [ebx +
IMAGE_SECTION_HEADER.SizeOfRawData], NULL

pop ecx
dec ecx
jnz @b

mov eax, [LoadLibraryA]
mov [loaderData.loadLibraryA], eax
mov eax, [GetProcAddress]
mov [loaderData.getProcAddress], eax
mov eax, [ntHeaders]
mov eax, [eax + IMAGE_NT_HEADERS.OptionalHeader.ImageBase]
mov [loaderData.imageBase], eax
mov eax, [ntHeaders]
mov eax, dword [eax + IMAGE_NT_HEADERS.OptionalHeader.DataDirectory + 5 *
sizeof.IMAGE_DATA_DIRECTORY + IMAGE_DATA_DIRECTORY.VirtualAddress]
mov [loaderData.relocVirtualAddress], eax
mov eax, [imageMemory]
mov [loaderData.allocationBase], eax

invoke VirtualAllocEx, [handle], NULL, 4096, MEM_COMMIT + MEM_RESERVE,
PAGE_EXECUTE_READ
mov [loaderMemory], eax
cinvoke printf, <'Loader memory: %p', 10, 0>, eax

lea eax, [loaderData]
invoke WriteProcessMemory, [handle], [loaderMemory], eax, sizeof.LoaderData, NULL

mov eax, [loaderMemory]
add eax, sizeof.LoaderData
invoke WriteProcessMemory, [handle], eax, loadImage, [loadImageSize], NULL

mov eax, [loaderMemory]
add eax, sizeof.LoaderData
invoke CreateRemoteThread, [handle], NULL, 0, eax, [loaderMemory], 0, NULL

invoke WaitForSingleObject, eax, 0xFFFFFFFF
invoke VirtualFreeEx, [handle], [loaderMemory], 0, MEM_RELEASE
invoke CloseHandle, [handle]

```

```
invoke HeapFree, [heapHandle], 0, [heapMemory]
invoke ExitProcess, 0
```

```
heapFail:
```

```
    stdcall criticalError, <'Failed to get process heap handle!', 0>
```

```
heapAllocFail:
```

```
    stdcall criticalError, <'Failed to allocate heap memory!', 0>
```

```
readFileFail:
```

```
    stdcall criticalError, <'Failed to read dll file!', 0>
```

```
openProcessFail:
```

```
    stdcall criticalError, <'Failed to open process!', 0>
```

```
endp
```

```
section '.bss' data readable writable
```

```
argc    dd ?
```

```
argv    dd ?
```

```
env      dd ?
```

```
heapMemory dd ?
```

```
fileHandle dd ?
```

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

```
library kernel32, 'kernel32.dll', \
```

```
    msvcrt, 'msvcrt.dll', \
```

```
    helper, 'helper.dll'
```

```
import kernel32, \
```

```
    CloseHandle, 'CloseHandle', \
```

```
    CreateFileA, 'CreateFileA', \
```

```
    CreateRemoteThread, 'CreateRemoteThread', \
```

```
    CreateToolhelp32Snapshot, 'CreateToolhelp32Snapshot', \
```

```
    ExitProcess, 'ExitProcess', \
```

```
    GetFileSizeEx, 'GetFileSizeEx', \
```

```
    GetFullPathNameA, 'GetFullPathNameA', \
```

```
    GetProcAddress, 'GetProcAddress', \
```

```
    GetProcessHeap, 'GetProcessHeap', \
```

```
    HeapAlloc, 'HeapAlloc', \
```

```
    HeapFree, 'HeapFree', \
```

```
    LoadLibraryA, 'LoadLibraryA', \
```

```
    OpenProcess, 'OpenProcess', \
```

```
    Process32First, 'Process32First', \
```

```
    Process32Next, 'Process32Next', \
```

```
    ReadFile, 'ReadFile', \
```

```
    VirtualAllocEx, 'VirtualAllocEx', \
```

```
    VirtualFreeEx, 'VirtualFreeEx', \
```

```
    WaitForSingleObject, 'WaitForSingleObject', \
```

```
    WriteProcessMemory, 'WriteProcessMemory'
```

```
import msvcrt, \
```

```
    atoi, 'atoi', \
```

```
    getchar, 'getchar', \
```

```
__getmainargs, '__getmainargs', \  
printf, 'printf', \  
strcmp, 'strcmp', \  
strlen, 'strlen'
```

```
import helper, \  
    manualMap, 'manualMap'Injection methods:  
-loadlibrary  
-manual-map
```

Usage:

Code Injector -injectionMethod -dllFile processName

example:

Code Injector -manual-map -MyCheat.dll pubgLite.exe

Note: helper.dll must be on the same path of the injector, otherwise it will not work, and it will support only x32 processes.

I've made some modification for the dll for bypassing AC:

Before modification:

After modification:

Compiled binaries:

https://anonfile.com/Hcldu2M4nf/InjectorFASM_rar