WOW64 Callback Table (FinFisher)

By odzhan ⋮⋮ 4/19/2023

```
.text:00007FF94E5F31C2                  lea     r15, Wow64CallbackTable
.text:00007FF94E5F31C9                  mov     ebx, r14d
.text:00007FF94E5F31CC                  mov     rdi, r15
.text:00007FF94E5F31CF
.text:00007FF94E5F31CF loc_7FF94E5F31CF:                        ; CODE XREF: LdrpLoadWow64+F7↓j
.text:00007FF94E5F31CF                  mov     rax, [rsp+310h+var_2C0]
.text:00007FF94E5F31D4                  xor     r8d, r8d
.text:00007FF94E5F31D7                  mov     r9, [rdi+8]
.text:00007FF94E5F31DB                  mov     rdx, [rdi]
.text:00007FF94E5F31DE                  mov     rcx, [rax+30h]
.text:00007FF94E5F31E2                  mov     rax, [rbp+218h]
.text:00007FF94E5F31E9                  mov     [rsp+310h+var_2E8], rax
.text:00007FF94E5F31EE                  mov     dword ptr [rsp+310h+var_2F0], r14d
.text:00007FF94E5F31F3                  call    LdrGetProcedureAddressForCaller
.text:00007FF94E5F31F8                  mov     esi, eax
.text:00007FF94E5F31FA                  test    eax, eax
.text:00007FF94E5F31FC                  js      loc_7FF94E6395C7
.text:00007FF94E5F3202                  inc     ebx
.text:00007FF94E5F3204                  add     rdi, 10h
.text:00007FF94E5F3208                  cmp     ebx, 6
.text:00007FF94E5F320B                  jb      short loc_7FF94E5F31CF
```

Introduction

Ken Johnson (otherwise known as Skywing) first talked about the KiUserExceptionDispatcher back in 2007 . Since then, scattered around the internet are various posts talking about it, but for some reason nobody demonstrating how to use it. It's been documented that FinFisher misuses the function pointers as part of its virtual machine functionality, so let's take a look at how to find the table before doing anything creative with it…The code to locate the table didn't take long and didn't require looking at FinFisher internals or existing code. It's a simple heuristic based search.

References

Observations

If you take a look at ***ntdll!LdrpLoadWow64***, that's called during initialization of a WOW64 process, you'll see it loading wow64.dll and resolving the address of six exports. This process has been better documented in the posts mentioned above.

- Wow64LdrpInitialize
- Wow64PrepareForException

- Wow64ApcRoutine
- Wow64PrepareForDebuggerAttach
- Wow64SuspendLocalThread
- Wow64SuspendLocalProcess

A closer look at how this works will provide you with an array of function names stored in STRING format and a pointer to a variable that holds each address resolved. The following is my attempt at recreating the same structure.

```
1
2
3
4
5  typedef union _W64_T {
6  LPVOID p;
7  DWORD64 q;
8  LPVOID *pp;
9  } W64_T;
10
11 typedef struct _WOW64_CALLBACK {
12 STRING Name;
13 W64_T  Function;
14 } WOW64_CALLBACK, *PWOW64_CALLBACK;
15 typedef struct _WOW64_CALLBACK_TABLE {
16 WOW64_CALLBACK  Wow64LdrpInitialize;
17 WOW64_CALLBACK  Wow64PrepareForException;
18 WOW64_CALLBACK  Wow64ApcRoutine;
19 WOW64_CALLBACK  Wow64PrepareForDebuggerAttach;
20 WOW64_CALLBACK  Wow64SuspendLocalThread;
21 WOW64_CALLBACK  Wow64SuspendLocalProcess;
22 } WOW64_CALLBACK_TABLE, *PWOW64_CALLBACK_TABLE;
23 WOW64_CALLBACK_TABLE Wow64Table = {
24 {RTL_CONSTANT_STRING("Wow64LdrpInitialize"), NULL},
25 {RTL_CONSTANT_STRING("Wow64PrepareForException"), NULL},
26 {RTL_CONSTANT_STRING("Wow64ApcRoutine"), NULL},
27 {RTL_CONSTANT_STRING("Wow64PrepareForDebuggerAttach"), NULL},
28 {RTL_CONSTANT_STRING("Wow64SuspendLocalThread"), NULL},
29 {RTL_CONSTANT_STRING("Wow64SuspendLocalProcess"), NULL}
30 };
31
```

Locating Table

There could be a number of ways to do this. In the following example, we search the .rdata section for STRING structures that equal the function pointer we wish to find. Since these strings are constant and unlikely to change, it works reasonably well.

```
1  BOOL
2  IsReadOnlyPtr(LPVOID ptr) {
   MEMORY_BASIC_INFORMATION mbi;
```

```
3
4  if (!ptr) return FALSE;
5
6  DWORD res = VirtualQuery(ptr, &mbi, sizeof(mbi));
7  if (res != sizeof(mbi)) return FALSE;
8  return ((mbi.State   == MEM_COMMIT    ) &&
9  (mbi.Type    == MEM_IMAGE     ) &&
10 (mbi.Protect == PAGE_READONLY));
11 }
12 BOOL
13 GetWow64FunctionPointer(PWOW64_CALLBACK Callback) {
14 auto m = (PBYTE)GetModuleHandleW(L"ntdll");
15 auto nt = (PIMAGE_NT_HEADERS)(m + ((PIMAGE_DOS_HEADER)m)->e_lfanew);
16 auto sh = IMAGE_FIRST_SECTION(nt);
17
18 for (DWORD i=0; i<nt->FileHeader.NumberOfSections; i++) {
19 if (*(PDWORD)sh[i].Name == *(PDWORD)".rdata") {
20 auto rva = sh[i].VirtualAddress;
21 auto cnt = (sh[i].Misc.VirtualSize - sizeof(STRING)) / sizeof(ULONG_PTR);
22 auto ptr = (PULONG_PTR)(m + rva);
23
24 for (DWORD j=0; j<cnt; j++) {
25 if (!IsReadOnlyPtr((LPVOID)ptr[j])) continue;
26
27 auto api = (PSTRING)ptr[j];
28 if (api->Length == Callback->Name.Length &&
29 api->MaximumLength == Callback->Name.MaximumLength)
30 {
31 if (!strncmp(api->Buffer, Callback->Name.Buffer, Callback->Name.Length)) {
32 Callback->Function.p = (PVOID)ptr[j + 1];
33 return TRUE;
34 }
35 }
36 }
37 break;
38 }
39 }
40 return FALSE;
41 }
42 void
43 GetWow64CallbackTable(PWOW64_CALLBACK_TABLE Table) {
44 GetWow64FunctionPointer(&Table->Wow64LdrpInitialize);
45 GetWow64FunctionPointer(&Table->Wow64PrepareForException);
46 GetWow64FunctionPointer(&Table->Wow64ApcRoutine);
47 GetWow64FunctionPointer(&Table->Wow64PrepareForDebuggerAttach);
48 GetWow64FunctionPointer(&Table->Wow64SuspendLocalThread);
49 GetWow64FunctionPointer(&Table->Wow64SuspendLocalProcess);
50 }
51
52
53
54
```

55

Summary

This type of code isn’t useful to a 32-Bit WOW process without jumping to 64-Bit since the function pointers are stored in the 64-Bit version of NTDLL. There are potentially other uses though like intercepting APCs, anti-debugging and processing exceptions before VEH or SEH, which FinFisher did successfully for many many years….

PoC here