

BÁO CÁO PROJECT 1

DLL INJECTION

(Memory access control attacks)
















Giảng viên hướng dẫn: **Nguyễn Ngọc Tự**

Mã số sinh viên: 18521371
Họ tên: Phạm Ngọc Tâm
Lớp:

I. Tổng quan về DLL

DLL là gì?

- DLL là **Dynamic Link Library** (thư viện liên kết động), là tổ hợp và dữ liệu mà nhiều ứng dụng khác nhau có thể gọi đến để thực hiện tác vụ nào đó ở cùng thời điểm, vì thế giúp giải quyết được vấn đề trên. Ví dụ như: user32.dll hoặc kernel32.dll, bất kì ứng dụng Windows đều phải dùng đến.

	kdusb.dll	3/19/2019 11:45 AM	Application extens...	44 KB
	keepaliveprovider.dll	8/13/2020 1:32 PM	Application extens...	69 KB
	KerbClientShared.dll	8/13/2020 1:32 PM	Application extens...	149 KB
	kerberos.dll	8/13/2020 1:32 PM	Application extens...	985 KB
	kernel.appcore.dll	3/19/2019 11:44 AM	Application extens...	58 KB
	kernel32.dll	9/14/2020 10:54 AM	Application extens...	706 KB
	KernelBase.dll	9/14/2020 10:55 AM	Application extens...	2,708 KB
	KeyboardSystemToastIcon.contrast-white...	3/19/2019 11:43 AM	PNG File	1 KB
	KeyboardSystemToastIcon.png	3/19/2019 11:43 AM	PNG File	1 KB
	KeyCredMgr.dll	3/19/2019 11:44 AM	Application extens...	53 KB
	keyiso.dll	7/17/2020 12:47 AM	Application extens...	89 KB
	keymgr.dll	3/19/2019 11:45 AM	Application extens...	55 KB
	KeywordDetectorMsftSidAdapter.dll	3/19/2019 11:44 AM	Application extens...	124 KB
	klist.exe	3/19/2019 11:45 AM	Application	37 KB
	kmddsp.tsp	3/19/2019 11:45 AM	TSP File	47 KB

Tại sao lại cần có DLL?

Giả sử khi ta thao tác với máy in, in một nội dung ta cần gọi hàm `print()` và ta đã có mã nguồn của hàm `print()` này. Bất cứ ứng dụng nào muốn sử dụng hàm `print()` sẽ bao gồm cả hàm này vào trong mã nguồn: ví dụ như Word, Excel,... như vậy trong hệ thống sẽ có rất nhiều bản sao của hàm `print()`, làm cho dung lượng để chứa các chương trình tăng lên. Ngoài ra khi phần cứng thay đổi ta lại phải thay đổi mã nguồn của `print()` để nó hoạt động đúng, dẫn đến phải sửa đổi theo dây chuyền, gây ra rất nhiều khó khăn. DLL sinh ra để giải quyết vấn đề này.

Ưu điểm của DLL?

- Mở rộng chức năng của một ứng dụng
- Đơn giản hoá việc quản lý dự án
- Giúp tiết kiệm bộ nhớ
- Chia sẻ tài nguyên giữa các ứng dụng
- Giúp giải quyết được vấn đề đa nền tảng
- Phục vụ cho những mục đích đặc biệt

II. DLL injection

DLL injection là gì?

DLL injection là một kỹ thuật được sử dụng để chạy code trong không gian địa chỉ của một tiến trình nào đó thông qua việc tải lên nó một file DLL. DLL injection được sử dụng để thay đổi hành vi của một chương trình nào đó theo cách mà người tạo ra chương trình đó cũng không biết.

Các cách inject một DLL vào một tiến trình

- Injecting a DLL using the Registry
- Injecting a DLL using Windows Hooks
- Injecting a DLL using Remote Threads
- Injecting a DLL with a Trojan DLL
- Injecting a DLL as a Debugger
- Injecting a DLL with `CreateProcess`

Cơ chế hoạt động của DLL injection

**Ví dụ một process A inject DLL vào process B:*

- Process A mở một luồng điều khiển process B sử dụng `OpenProcess()`
- Sau đó Process A lấy một vùng bộ nhớ trong process B sử dụng `VirtualAllocEx()`

- Process A ghi vào vùng nhớ vừa tạo địa chỉ nơi DLL's pathname được tạo sử dụng WriteProcessMemory()
- Process A sử dụng CreateRemoteThread() tạo một luồng khác trong process B, luồng này chạy một hàm loadLibrary() tải một DLL file lên process B.
- Loadlibrary() nằm trong file kernel32.dll, bất cứ process nào được tạo ra cũng đều chứa kernel32.dll trong không gian vùng nhớ, nên process A có thể xác định địa chỉ của kernel32.dll trong process B bằng GetModuleHandle(), và xác định địa chỉ của Loadlibrary trong kernel32.dll bằng GetProcAddress().
- Lúc này process A đã có thể thực hiện Loadlibrary() trong process B, process A đặt vào hàm Loadlibrary() địa chỉ của DLL's pathname đã lưu vào không gian bộ nhớ của process B trước đó.
- Cuối cùng process A thực hiện thành công inject DLL file vào process B.

Các nguy cơ của DLL injection

- Vì Window cung cấp những API cho phép một tiến trình có thể điều khiển tiến trình khác và làm biến đổi chức năng ứng dụng → dẫn đến nhiều người đã lợi dụng để thực hiện gian lận trong các game bản quyền (crack)
- Các phần mềm độc hại có thể sử dụng DLL injection để điều khiển một tiến trình khác thực hiện các hành động tấn công thay nó, như tải xuống phần mềm độc hại khác hoặc lấy cắp thông tin từ hệ thống.
- Sau khi được inject, các tiến trình độc hại có thể có quyền truy cập vào bộ nhớ, các đặc quyền và tài nguyên của tiến trình bị tấn công mà không bị phát hiện.

III. Thực hiện DLL injection

Kỹ thuật DLL injection yêu cầu một thread trong target process (tiến trình muốn chèn DLL vào) gọi là Load-Library để load DLL lên. Bởi vì không thể điều khiển các threads trong một process khác hơn là process tự tạo ra nên sẽ thực hiện tạo một thread mới trong target process, từ đó dễ dàng điều khiển những mã code thực hiện. Windows cung cấp một function gọi là CreateRemoteThread, giúp hỗ trợ việc tạo ra một thread trong một process dễ dàng hơn.

```
HANDLE CreateRemoteThread(  
    HANDLE hProcess,  
    PSECURITY_ATTRIBUTES psa,  
    DWORD dwStackSize,  
    PTHREAD_START_ROUTINE pfnStartAddr,  
    PVOID pvParam,  
    DWORD fdwCreate,  
    PDWORD pdwThreadId);
```

hProcess: tham số đại diện, dùng để nhận dạng process tạo ra

pfnStartAddr: tham số dùng để nhận dạng địa chỉ của thread function

Sau khi tạo một thread mới trong target process, sử dụng thread này để gọi hàm LoadLibraryW.

```
HANDLE hThread = CreateRemoteThread(hProcessRemote, NULL, 0,
    LoadLibraryW, L "C:\\MyLib.dll", 0, NULL);
```

Khi thread mới được tạo trong target process, thread này lập tức gọi LoadLibraryW, đặt địa chỉ của DLL's pathname. Nhưng sử dụng như vậy sẽ gây ra hai 2 vấn đề:

Vấn đề 1: nếu gọi loadLibrary trực tiếp có thể gây ra vấn đề access violation

Vấn đề 2: nếu DLL's pathname không nằm trong không gian địa chỉ của target process khi loadLibrary tham chiếu đến sẽ gây ra access violation

Để giải quyết vấn đề 1 nên lấy chính xác địa chỉ của **LoadLibraryW** bằng cách gọi **GetProcAddress**.

```
// Get the real address of LoadLibraryW in Kernel32.dll.
PTHREAD_START_ROUTINE pfnThreadRtn = (PTHREAD_START_ROUTINE)
    GetProcAddress(GetModuleHandle(TEXT( "Kernel32 ")), "LoadLibraryW ");

HANDLE hThread = CreateRemoteThread(hProcessRemote, NULL, 0,
    pfnThreadRtn, L "C:\\MyLib.dll", 0, NULL);
```

Để fix vấn đề 2, cần lấy DLL's pathname string đưa vào không gian địa chỉ của target process. Sau đó, khi CreateRemoteThread được gọi, chúng ta cần đặt vào địa chỉ nơi đặt pathname string. Windows cung cấp một function, VirtualAllocEx, nó cho phép một process phân vùng bộ nhớ trong một không gian địa chỉ của process khác.

```
PVOID VirtualAllocEx(
    HANDLE hProcess,
    PVOID pvAddress,
    SIZE_T dwSize,
    DWORD flAllocationType,
    DWORD flProtect);
```

Khi đã phân vùng bộ nhớ cho DLL's pathname string, cần có cách để copy chuỗi này từ không gian địa chỉ process của chúng ta vào không gian địa chỉ của target process. Sử dụng hàm **WriteProcessMemory**.

BOOL WriteProcessMemory(
HANDLE hProcess,
PVOID pvAddressRemote,
LPCVOID pvBufferLocal,
SIZE_T dwSize,
SIZE_T pdwNumBytesWritten);

hProcess: tham chiếu đến target process
pvAddressRemote: địa chỉ của Target process
pvBufferLocal: địa chỉ của bộ nhớ trong local process
dwSize: số lượng bytes cần chuyển
pdwNumBytesRead và **pdwNumBytesWritten:** chỉ ra số byte thực sự chuyển

Tóm tắt lại DLL injection chung gồm các bước sau:

1. Sử dụng **OpenProcess** để can thiệp vào target process
2. Sử dụng **VirtualAllocEx** function để chỉ định bộ nhớ trong target process's address space.
3. Sử dụng **WriteProcessMemory** function để copy **DLL's pathname** vào bộ nhớ được chỉ định ở bước 1
4. Sử dụng **GetProcAddress** function để lấy địa chỉ thực của LoadLibraryW function
5. Sử dụng **CreateRemoteThread** function để tạo ra một thread trong target process mà gọi LoadLibrary function, đặt vào nó địa chỉ được chỉ định ở bước 1. Cùng lúc này, the DLL đã được chèn vào không gian địa chỉ của target process, và DLL's dllmain function nhận một thông báo DLL_PROCESS_ATTACH và có thể thực hiện code mình muốn. Khi Dllmain kết thúc, thread được tạo ra để gọi Load-LibraryW trở về **BaseThreadStart** function. BaseThreadStart sau đó gọi **ExitThread**, làm cho thread tạo ra biến mất.
6. Sử dụng **VirtualFreeEx** function để giải phóng bộ nhớ ở bước 1.
7. Sử dụng **GetProcAddress** function lấy địa chỉ thực của FreeLibrary function
8. Sử dụng **createRemoteThread** function để tạo ra thread trong target process gọi FreeLibrary function, đặt vào remote DLL's HMODULE.

Demo:


**DLL injection sử dụng kỹ thuật tạo Remote Thread.*

Target process (Remote Thread): là một chương trình có chức năng in ra thông tin các adapter mà máy host hiện có.

DLL file: chứa các mã nguồn có chức năng fake thông tin adapter đầu tiên theo thông tin cho trước.

Injector: Thực hiện việc inject DLL file để thay đổi thông tin adapter đầu tiên

- **Đầu tiên thực hiện run code để tạo ra target process:**

 C:\Users\ASUS\Downloads\hooks_example-master\Debug\target.exe

```

ComboIndex:      26
Adapter Name:    {DC2DDF74-4A38-49D2-A6DD-A52E5C530AA2}
Adapter Desc:    Realtek PCIe GbE Family Controller
Adapter Addr:    04-D4-C4-7A-8A-47
Index:           26

ComboIndex:      7
Adapter Name:    {24388892-7B80-45E5-A27E-00C46117C0A1}
Adapter Desc:    Bluetooth Device (Personal Area Network)
Adapter Addr:    D4-D2-52-71-87-BF
Index:           7

ComboIndex:      9
Adapter Name:    {3E75E187-5B23-46D7-9D5D-FB7C323D1D2C}
Adapter Desc:    Npcap Loopback Adapter
Adapter Addr:    02-00-4C-4F-4F-50
Index:           9

ComboIndex:      4
Adapter Name:    {0BA2B1B1-7822-4C6B-BEC4-A180B26C6598}
Adapter Desc:    VMware Virtual Ethernet Adapter for VMnet1
Adapter Addr:    00-50-56-C0-00-01
Index:           4

```

Hiện thông tin các Adapter mà máy host có. Tên process là target.exe

- **Tiếp theo sử dụng GetProcAddress để lấy địa chỉ của LoadLibraryW function:**

```

lpLoadLibraryW = GetProcAddress(GetModuleHandle(L"KERNEL32.DLL"), "LoadLibraryW");

if (!lpLoadLibraryW)
{
    PrintError(TEXT("GetProcAddress"));
    // close process handle
    CloseHandle( processInformation.hProcess);
    return FALSE;
}

```

- **Sử dụng VirtualAllocEx để phân vùng nhớ trong target process để chèn địa chỉ thread mới vào**

```

// allocate mem for dll name
lpRemoteString = VirtualAllocEx(processInformation.hProcess, NULL, nLength + 1, MEM_COMMIT, PAGE_READWRITE);
if (!lpRemoteString)
{
    PrintError(TEXT("VirtualAllocEx"));

    // close process handle
    CloseHandle(processInformation.hProcess);

    return FALSE;
}

```

- Sử dụng `WriteProcessMemory` để ghi thông tin địa chỉ process vừa tạo vào vùng nhớ đã cấp ở target process

```
// write dll name
if (!WriteProcessMemory(processInformation.hProcess, lpRemoteString, lpCwszDll, nLength, NULL)) {

    PrintError(TEXT("WriteProcessMemory"));
    // free allocated memory
    VirtualFreeEx(processInformation.hProcess, lpRemoteString, 0, MEM_RELEASE);

    // close process handle
    CloseHandle(processInformation.hProcess);

    return FALSE;
}
```

- Tiếp theo là sử dụng `CreateRemoteThread` để tạo thread mới chạy `LoadLibraryW` để tải file DLL lên

```
// call loadlibraryw
HANDLE hThread = CreateRemoteThread(processInformation.hProcess, NULL, NULL, (LPTHREAD_START_ROUTINE)lpLoadLibraryW, lpRemoteString, NULL, NULL);

if (!hThread) {
    PrintError(TEXT("CreateRemoteThread"));
}
else {
    WaitForSingleObject(hThread, 4000);

    //resume suspended process
    ResumeThread(processInformation.hThread);
}
```

- Cuối cùng là giải phóng vùng nhớ đã cấp phát ở target process

```
// free allocated memory
VirtualFreeEx(processInformation.hProcess, lpRemoteString, 0, MEM_RELEASE);

// close process handle
CloseHandle(processInformation.hProcess);
```

- Ở file DLL, nếu inject thành công sẽ hiện messagebox như bên dưới

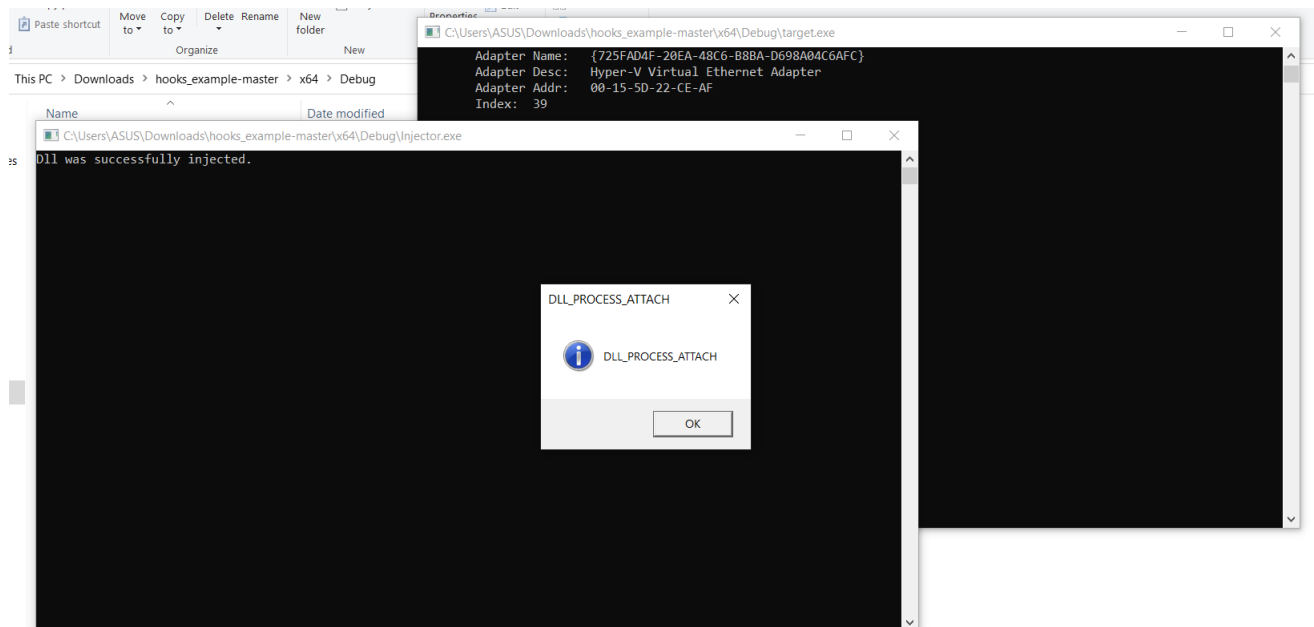
```
static HooksManager* manager = new HooksManager();
MessageBox(NULL, L"DLL_PROCESS_ATTACH", L"DLL_PROCESS_ATTACH", MB_ICONINFORMATION);
switch (ul_reason_for_call)
{
case DLL_PROCESS_ATTACH:
    MessageBox(NULL, L"DLL_PROCESS_ATTACH", L"DLL_PROCESS_ATTACH", MB_ICONINFORMATION);
    break;
case DLL_THREAD_ATTACH:
    MessageBox(NULL, L"DLL_THREAD_ATTACH", L"DLL_THREAD_ATTACH", MB_ICONINFORMATION);
    break;
case DLL_THREAD_DETACH:
    MessageBox(NULL, L"DLL_THREAD_DETACH", L"DLL_THREAD_DETACH", MB_ICONINFORMATION);
    break;
case DLL_PROCESS_DETACH:
    MessageBox(NULL, L"DLL_PROCESS_DETACH", L"DLL_PROCESS_DETACH", MB_ICONINFORMATION);
    break;
}

return TRUE;
```

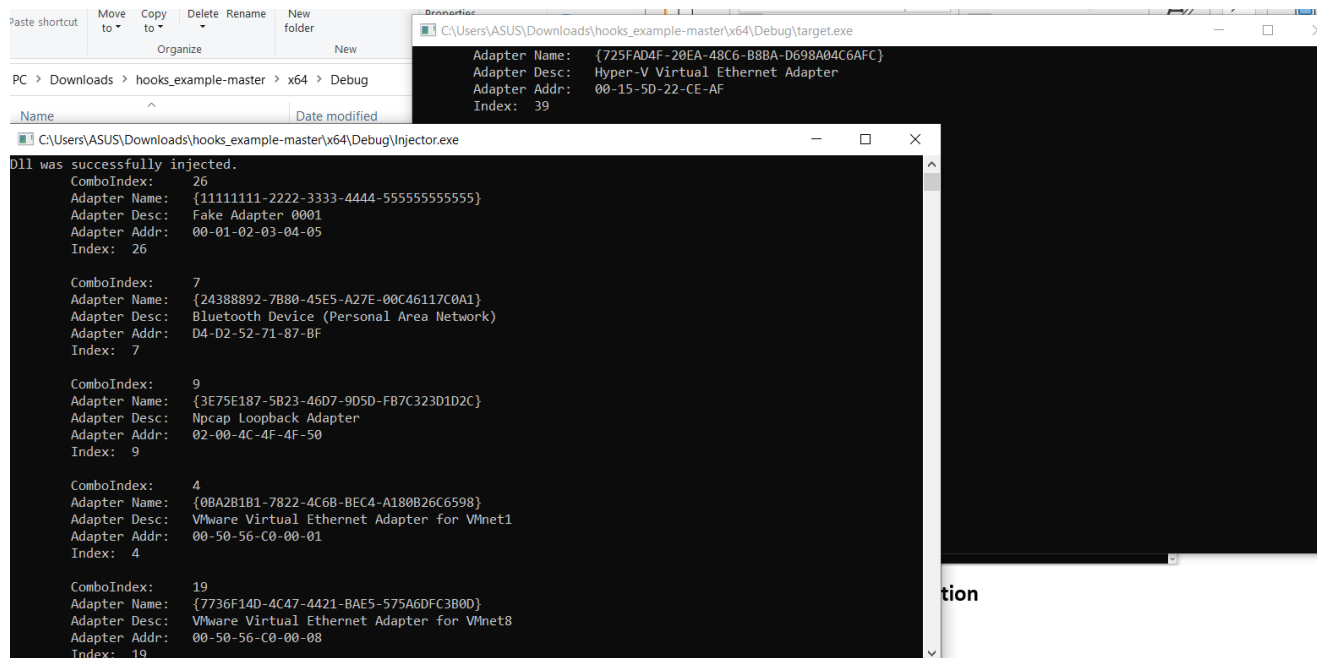

- Thông tin sẽ thay đổi ở đây là adapter đầu tiên và được thay thành

```
std::string fakeAdapterName = "{11111111-2222-3333-4444-555555555555}";
std::string fakeAdapterDescription = "Fake Adapter 0001";
```

- Kết quả:



Thông báo đã inject thành công.



Thông tin adapter đầu tiên đã bị thay đổi thành như ý muốn thông qua DLL injection

VII. Cách phòng chống DLL injection

- Để ngăn chặn DLL injection, chúng ta cần đảm bảo không tin tưởng process nào có quyền Administrator truy cập vào ứng dụng. Một cách khác là nên có một phần mềm diệt virus được cập nhật thường xuyên; tuy nhiên các phần mềm antivirus không hoàn toàn được tin tưởng nó, vì thế nên tải phần mềm từ các trang web chính hãng.
- Khi thực hiện inject DLL, cần tạo một danh sách các DLL cần tải, nếu DLL nào không nằm trong danh sách đó thì không tải lên.