Báo cáo Project 1

DLL Injection

(Memory access control attack)

1. Định nghĩa:

DLL Injection là một kỹ thuật được sử dụng để running code trong không gian địa chỉ của một process khác đang chạy, code ở đây là ở dạng thư viện liên kết động (Dynamic Link Libraries - DLL). **DLL Injection** thường được các program bên ngoài sử dụng để gây ảnh hưởng đến hoạt động của một program khác mà người viết program không muốn.

2. Phương thức hoạt động, phân tích các nguy cơ:

a. Phương thức hoạt động:

Code injection không tác động vào các ứng dụng cơ bản được lưu trên ổ đĩa. Thay vào đó, nó chờ đến khi ứng dụng đó được khởi chạy sau đó chèn thêm mã vào tiến trình đang chạy để thay đổi cách thức mà tiến trình đó hoạt động.

Như chúng ta đã biết, Windows có chứa một loạt các giao diện lập trình ứng dụng (API) có thể được sử dụng để tiêm mã. Một tiến trình có thể tự gắn nó vào một tiến trình mục tiêu, cấp phát bộ nhớ, sau đó viết một DLL hoặc mã khác vào bộ nhớ đó, và hướng dẫn cho tiến trình đích thực thi các mã. Windows hoàn toàn không ngăn chặn việc các tiến trình trên máy tính của bạn can thiệp vào nhau như thế này.

b. Phân tích các nguy cơ:

- Có thể chiếm quyền sử dụng máy tính.
- Hack game, các phần mềm trong máy.
- Gây ra các lỗ hổng, các vấn đề về đồng bộ hóa về tài nguyên mà trước đây chưa từng gặp hoặc các file này sẽ làm trầm trọng thêm các vấn đề hiện hữu trên ứng dụng.

3. Phương thức thực hiện:

Có nhiều cách thực hiện DLL Injection:

- Injtecting a DLL Using the Registry.

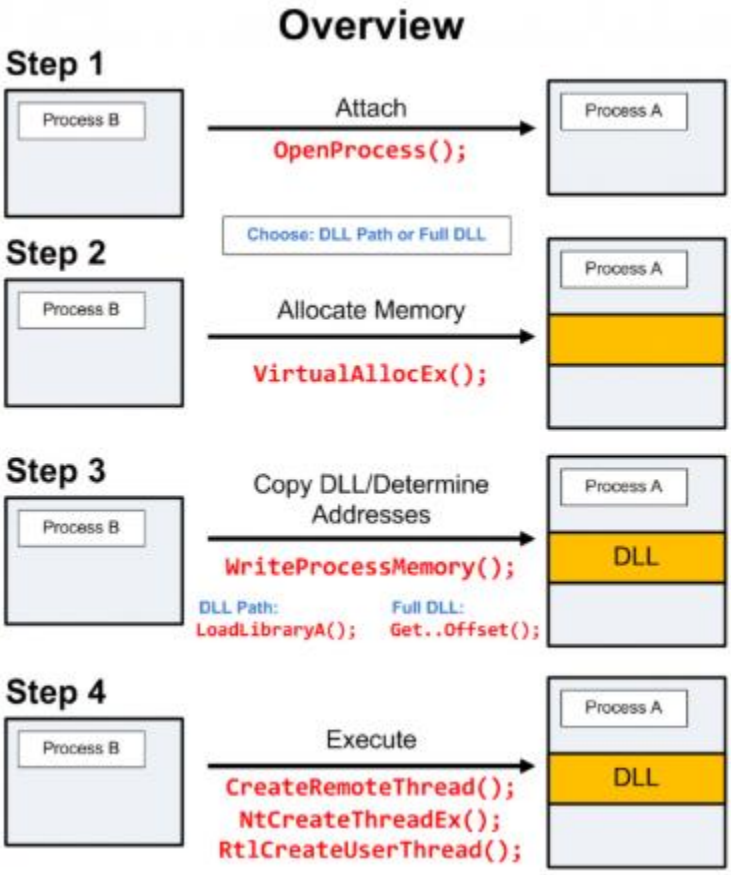
- Injtecting a DLL Using Windows Hooks.

- Injtecting a DLL Using Remote Threads.
- Injecting a DLL with Trojan DLL.
- Injtecting a DLL as a Debugger.
- Injecting Code with CreateProcess.

Dưới đây là demo Injtecting a DLL Using Remote Threads.

Injtecting a DLL Using Remote Threads gồm 4 bước sau:

- Can thiệp vào process.
- Cấp phát vùng nhớ trong process.
- Copy toàn bộ DLL hoặc đường dẫn đến DLL vào vùng nhớ đó và xác định vị trí của vùng nhớ.
- Process thực thi DLL.



Để process thực thi DLL có một vài lựa chọn CreateRemoteThread(), NtCreateThreadEx(), … Chúng ta thực hiện các bước cấp phát và copy để có không gian bộ nhớ của process và chuẩn bị để bắt đầu thực thi DLL.

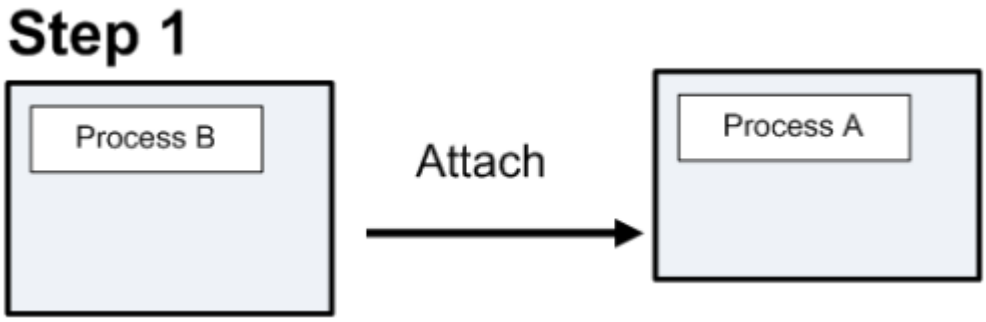
Có 2 cách phổ biến là LoadLibraryA() và nhảy đến DLLMain.

LoadLibraryA() là hàm trong Kernel32.dll để nạp DLL, file thực thi hoặc các loại thư viện khác (User32.dll, GDI32.dll). Tham số truyền vào của hàm là tên DLL. Tức là chỉ cần cấp phát một vùng nhớ, ghi đường dẫn đến DLL và chọn nơi bắt đầu thực thi là địa chỉ của hàm *LoadLibraryA()* với tham số truyền vào là địa chỉ vùng nhớ chứa đường dẫn đến DLL. Tuy nhiên, nhược điểm của hàm *LoadLibraryA()* là nó sẽ đăng kí DLL với program nên sẽ dễ bị phát hiện. Thêm vào đó, DLL chỉ được nạp lên chứ không được thực thi.

Nhảy đến DLLMain (hoặc một entry point khác) là nạp toàn bộ DLL vào một vùng nhớ và xác định offset tới DLL entry point. Sử dụng cách này sẽ tránh được việc đăng kí DLL với program và thực thi DLL trong process.

Ở bài báo cáo này sử dụng *LoadLibraryA()*.

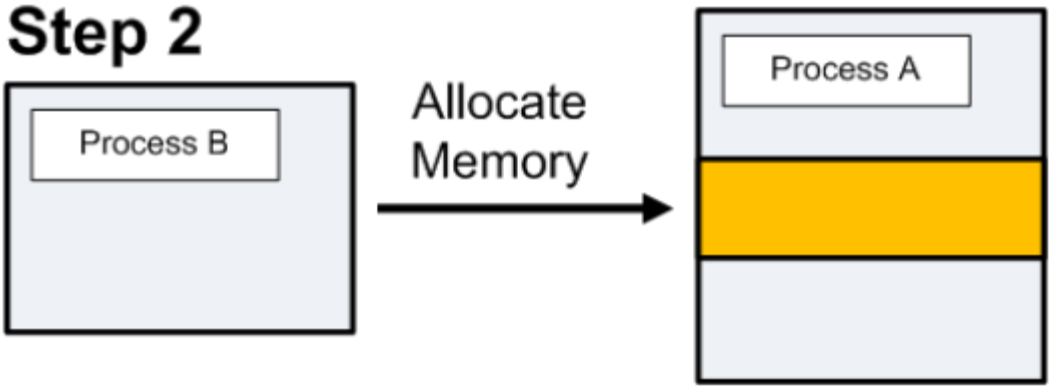
a. Can thiệp vào process:



Chúng ta cần lấy được handle của process để có thể thao tác với nó. Có thể thực hiện việc này bằng cách spawn ra process cần tìm hoặc bằng cách tìm nó dựa trên một dấu vết mà chắn chắn có tồn tại khi process đó chạy (Sử dụng hàm *OpenProcess()*).

```
//Lấy handle
HANDLE hHandle = OpenProcess(PROCESS_ALL_ACCESS, FALSE, PID);
```

b. Cấp phát vùng nhớ:



Sử dụng hàm *VirtualAllocEx()* để lấy dung lượng vùng nhớ cần cấp phát làm tham số truyền vào. Nếu sử dụng hàm *LoadLibraryA()*, cần cấp phát vùng nhớ để ghi đường dẫn đến DLL, còn nếu sử dụng phương thức nhảy đến DLLMain thì cần cấp phát vùng nhớ đủ lớn để ghi toàn bộ DLL vào.

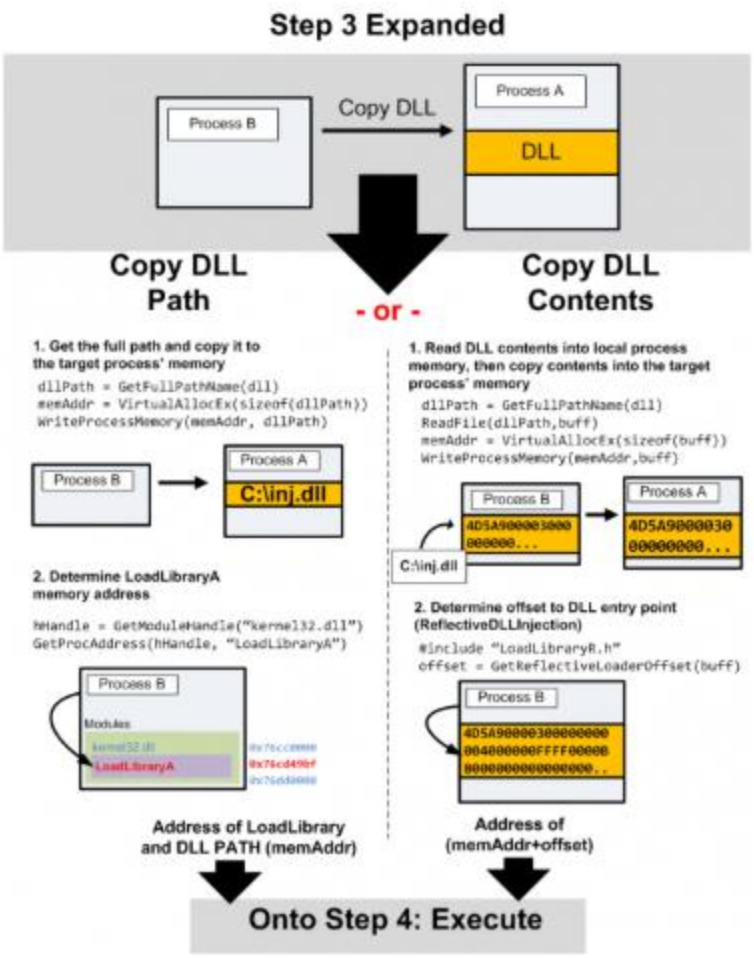
Sử dụng đường dẫn đến DLL sẽ phải sử dụng hàm *LoadLibraryA()* – một phương pháp rất phổ biến.

Cấp phát đủ vùng nhớ để ghi đường dẫn đến DLL vào:

```
//Lấy đương dẫn tới file dll
GetFullPathName(TEXT("testlib.dll"), _MAX_PATH, dllPath, NULL);

//Cấp phát vùng nhớ
LPVOID dllPathAddress = VirtualAllocEx(hHandle, 0, strlen(dllPath), MEM_RESERVE | MEM_COMMIT, PAGE_EXECUTE_READWRITE);
```

c. Copy DLL và xác định địa chỉ:



Khi đã có vùng nhớ cần thiết, chúng ta sẽ thực hiện công việc ghi (hàm *WriteProcessMemory()*).

```
//Viết đường dẫn dll vào
WriteProcessMemory(hHandle, dllPathAddress, dllPath, strlen(dllPath), NULL);
```

Xác định điểm bắt đầu thực thi: Đường dẫn DLL và *LoadLibraryA()*: Xác định địa chỉ của hàm *LoadLibraryA()*, chuyển nó đến hàm thực thi cùng với tham số truyền vào là địa chỉ vùng nhớ chứa đường dẫn đến DLL (hàm *GetModuleHandle()* và *GetProcessAddress()*).

```
//Lấy địa chỉ LoadLibraryA và chuyển đến thực thi
LPVOID loadLibraryAddress = GetProcAddress(GetModuleHandleA("Kernel32.dll"), "LoadLibraryA");
```

d. Thực thi DLL:

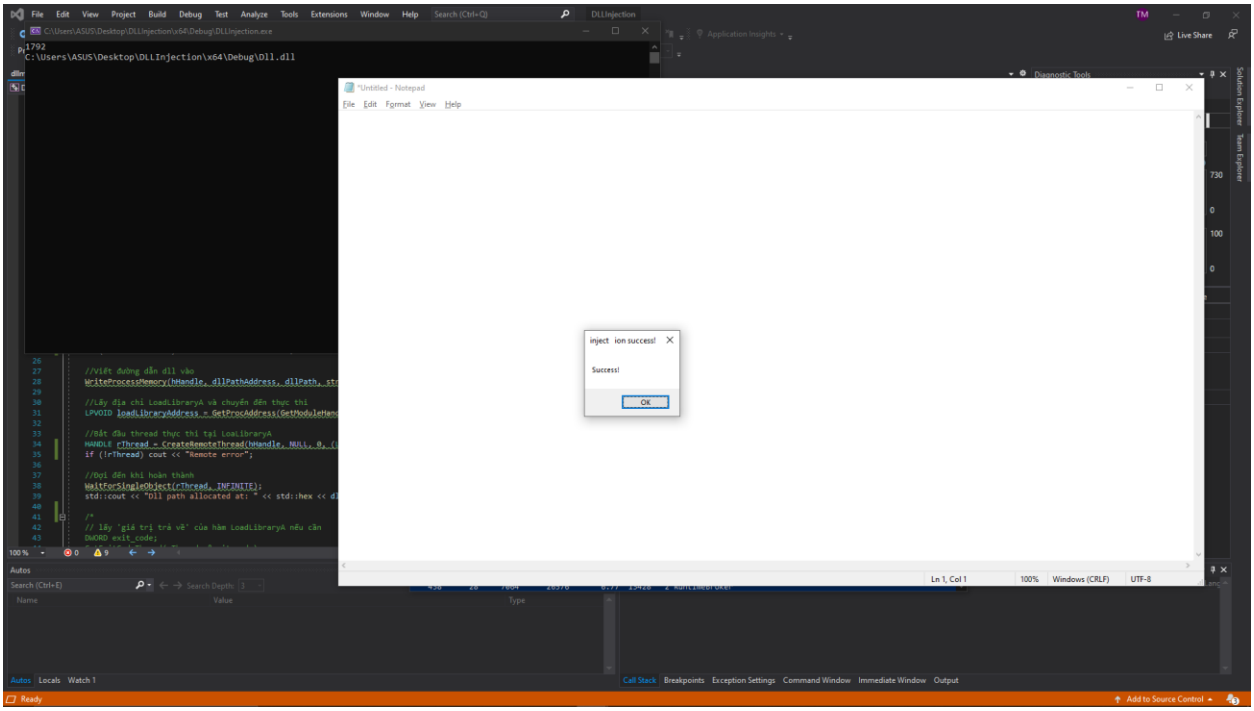
Có nhiều cách để process thực thi DLL nhưng CreateRemoteThread() là cách sử dụng phổ biến nhất.

```
//Bắt đầu thread thực thi tại LoaLibraryA
HANDLE rThread = CreateRemoteThread(hHandle, NULL, 0, (LPTHREAD_START_ROUTINE)loadLibraryAddress, dllPathAddress, 0, NULL);
```

Hàm WaitForSingleObject() để chắc chắn rằng DLL đã được thực thi trước khi Windows thực thi các công việc tiếp theo của process.

```
//Đợi đến khi hoàn thành
WaitForSingleObject(rThread, INFINITE);
std::cout << "Dll path allocated at: " << std::hex << dllPath<< std::endl;
std::cin.get();
```

Kết quả demo:



4. Đề xuất cách phòng chống:

- Thường xuyên kiểm tra danh sách các module import và tìm các vùng nhớ có quyền đọc ghi của process.

- Sử dụng SpyDLLRemover để phát hiện và loại bỏ các mã độc trong tập tin có đuôi .dll.