Dynamic-Link Libraries

Thư viện liên kết động (DLL) là một mô-đun chứa các chức năng và dữ liệu có thể được sử dụng bởi một mô-đun khác (ứng dụng hoặc DLL).

Một DLL có thể xác định hai loại chức năng: exported và internal. Các chức năng được xuất ra nhằm mục đích được gọi bởi các mô-đun khác, cũng như từ bên trong DLL nơi chúng được xác định. Các hàm bên trong thường chỉ được gọi từ bên trong DLL nơi chúng được định nghĩa. Mặc dù DLL có thể xuất dữ liệu, nhưng dữ liệu của nó thường chỉ được sử dụng bởi các chức năng của nó. Tuy nhiên, không có gì ngăn cản một mô-đun khác đọc hoặc ghi địa chỉ đó.

DLL cung cấp một cách để mô-đun hóa các ứng dụng để chức năng của chúng có thể được cập nhật và tái sử dụng dễ dàng hơn. DLL cũng giúp giảm chi phí bộ nhớ khi một số ứng dụng sử dụng cùng một chức năng cùng một lúc, bởi vì mặc dù mỗi ứng dụng nhận được bản sao dữ liệu DLL của riêng mình, các ứng dụng chia sẻ mã DLL.

Giao diện lập trình ứng dụng Windows (API) được triển khai dưới dạng một tập hợp các DLL, vì vậy bất kỳ quy trình nào sử dụng Windows API đều sử dụng liên kết động.

Liên kết động cho phép một mô-đun chỉ bao gồm thông tin cần thiết để định vị một hàm DLL đã xuất tại thời gian tải hoặc thời gian chạy. Liên kết động khác với liên kết tĩnh quen thuộc hơn, trong đó trình liên kết sao chép mã của chức năng thư viện vào mỗi mô-đun gọi nó.

1)Các loại liên kết động

Có hai phương pháp để gọi một hàm trong DLL:

Trong liên kết động thời gian tải, một mô-đun thực hiện các lệnh gọi rõ ràng đến các hàm DLL đã xuất như thể chúng là các hàm cục bộ. Điều này yêu cầu bạn liên kết mô-đun với thư viện nhập cho DLL có chứa các chức năng. Thư viện nhập cung cấp cho hệ thống thông tin cần thiết để tải DLL và định vị các chức năng DLL đã xuất khi ứng dụng được tải.

Trong liên kết động thời gian chạy, một mô-đun sử dụng chức năng LoadLibrary hoặc LoadLibraryEx để tải DLL tại thời điểm chạy. Sau khi DLL được tải, mô-đun gọi hàm GetProcAddress để lấy địa chỉ của các hàm DLL đã xuất. Mô-đun gọi các hàm DLL đã xuất bằng cách sử dụng các con trỏ hàm do GetProcAddress trả về. Điều này giúp loại bỏ sự cần thiết của một thư viện nhập.

2)DLL và quản lý bộ nhớ

Mọi quy trình tải DLL sẽ ánh xạ nó vào không gian địa chỉ ảo của nó. Sau khi quá trình tải DLL vào địa chỉ ảo của nó, nó có thể gọi các hàm DLL đã xuất.

Hệ thống duy trì số lượng tham chiếu trên mỗi quá trình cho mỗi DLL. Khi một luồng tải DLL, số lượng tham chiếu sẽ tăng lên một. Khi quá trình kết thúc hoặc khi số lượng tham chiếu trở thành 0 (chỉ liên kết động trong thời gian chạy), DLL sẽ được dỡ bỏ khỏi không gian địa chỉ ảo của quá trình.

Giống như bất kỳ hàm nào khác, một hàm DLL được xuất chạy trong ngữ cảnh của luồng gọi nó. Do đó, các điều kiện sau được áp dụng:

Các luồng của quá trình được gọi là DLL có thể sử dụng các chốt được mở bởi một hàm DLL. Tương tự, các chốt được mở bởi bất kỳ luồng nào của quá trình gọi có thể được sử dụng trong hàm DLL.

DLL sử dụng ngăn xếp của luồng đang gọi và không gian địa chỉ ảo của tiến trình gọi.

DLL cấp phát bộ nhớ từ không gian địa chỉ ảo của quá trình gọi.

3)Tạo thư viện liên kết động

Để tạo Thư viện liên kết động (DLL), bạn phải tạo một hoặc nhiều tệp mã nguồn và có thể là tệp trình liên kết để xuất các chức năng. Nếu bạn định cho phép các ứng dụng sử dụng DLL của mình sử dụng liên kết động trong thời gian tải, bạn cũng phải tạo một thư viện nhập.

Tạo tệp nguồn

Các tệp nguồn cho một DLL chứa các chức năng và dữ liệu đã xuất, các chức năng và dữ liệu bên trong cũng như một chức năng điểm nhập tùy chọn cho DLL. Bạn có thể sử dụng bất kỳ công cụ phát triển nào hỗ trợ việc tạo DLL dựa trên Windows.

Nếu DLL của bạn có thể được ứng dụng đa luồng sử dụng, bạn nên đặt DLL của mình "an toàn theo luồng". Bạn phải đồng bộ hóa quyền truy cập vào tất cả dữ liệu toàn cầu của DLL để tránh hỏng dữ liệu. Bạn cũng phải đảm bảo rằng bạn chỉ liên kết với các thư viện an toàn theo chuỗi. Ví dụ: Microsoft Visual C ++ chứa nhiều phiên bản của Thư viện thời gian chạy C, một phiên bản không an toàn theo luồng và hai phiên bản.

Các chức năng xuất khẩu

Cách bạn chỉ định hàm nào trong DLL sẽ được xuất tùy thuộc vào công cụ bạn đang sử dụng để phát triển. Một số trình biên dịch cho phép bạn xuất một hàm trực tiếp trong mã nguồn bằng cách sử dụng công cụ sửa đổi trong khai báo hàm. Những lần khác, bạn phải chỉ định xuất trong tệp mà bạn chuyển cho trình liên kết.

Ví dụ: sử dụng Visual C ++, có hai cách khả thi để xuất các hàm DLL: với công cụ sửa đổi \_\_declspec (dllexport) hoặc với tệp định nghĩa mô-đun (.def). Nếu bạn sử dụng công cụ sửa đổi \_\_declspec (dllexport), thì không cần thiết phải sử dụng tệp .def. Để biết thêm thông tin, hãy xem Xuất từ ​​một DLL.

Tạo Thư viện Nhập

Tệp thư viện nhập (.lib) chứa thông tin mà trình liên kết cần để giải quyết các tham chiếu bên ngoài đến các hàm DLL đã xuất, do đó hệ thống có thể xác định vị trí DLL được chỉ định và các hàm DLL đã xuất tại thời điểm chạy. Bạn có thể tạo một thư viện nhập cho DLL của mình khi bạn xây dựng DLL của mình.

Để biết thêm thông tin, hãy xem Xây dựng Thư viện Nhập và Xuất Tệp.

Sử dụng Thư viện Nhập

Ví dụ: để gọi hàm CreateWindow, bạn phải liên kết mã của mình với thư viện nhập User32.lib. Lý do là CreateWindow nằm trong một DLL hệ thống có tên là User32.dll và User32.lib là thư viện nhập được sử dụng để giải quyết các lệnh gọi đến các hàm đã xuất trong User32.dll trong mã của bạn. Trình liên kết tạo một bảng chứa địa chỉ của mỗi lệnh gọi hàm. Các lệnh gọi đến các hàm trong DLL sẽ được sửa khi DLL được tải. Trong khi hệ thống đang khởi tạo quy trình, nó sẽ tải User32.dll vì quy trình phụ thuộc vào các hàm được xuất trong DLL đó và nó cập nhật các mục nhập trong bảng địa chỉ hàm. Tất cả các lệnh gọi tới CreateWindow đều gọi hàm được xuất từ ​​User32.dll.

Tạo một Thư viện liên kết động đơn giản



4)Các chức năng của thư viện liên kết động

AddDllDirectory :Thêm một thư mục vào đường dẫn tìm kiếm DLL của quy trình.

DisableThreadLibraryCalls Tắt thông báo đính kèm và tách chuỗi cho DLL được chỉ định.

DllMain : Một điểm vào tùy chọn vào một DLL.

FreeLibrary : Giảm số lượng tham chiếu của DLL đã tải. Khi số lượng tham chiếu bằng không, mô-đun sẽ không được ánh xạ khỏi không gian địa chỉ của quá trình gọi.

FreeLibraryAndExitThread : Giảm số lượng tham chiếu của một DLL đã tải, và sau đó gọi ExitThread để kết thúc chuỗi gọi.

GetDllDirectory : Truy xuất phần dành riêng cho ứng dụng của đường dẫn tìm kiếm được sử dụng để định vị các tệp DLL cho ứng dụng.

GetModuleFileName : Truy xuất đường dẫn đủ điều kiện cho tệp chứa mô-đun được chỉ định.

GetModuleFileNameEx : Truy xuất đường dẫn đủ điều kiện cho tệp chứa mô-đun được chỉ định.

GetModuleHandle : Truy xuất một tay cầm mô-đun cho mô-đun được chỉ định.

GetModuleHandleEx : Truy xuất một tay cầm mô-đun cho mô-đun được chỉ định.

GetProcAddress : Lấy địa chỉ của một hàm hoặc biến được xuất từ ​​DLL được chỉ định.

LoadLibrary : Ánh xạ mô-đun thực thi được chỉ định vào không gian địa chỉ của quá trình gọi.

LoadLibraryEx : Ánh xạ mô-đun thực thi được chỉ định vào không gian địa chỉ của quá trình gọi.

LoadPackagedLibrary : Bản đồ mô-đun đóng gói được chỉ định và các phụ thuộc của nó vào không gian địa chỉ của quá trình gọi. Chỉ các ứng dụng Windows Store mới có thể gọi chức năng này.

RemoveDllDirectory : Loại bỏ một thư mục đã được thêm vào đường dẫn tìm kiếm DLL của quy trình bằng cách sử dụng AddDllDirectory.

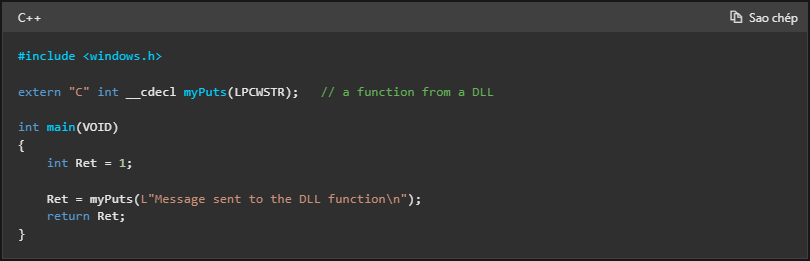
SetDefaultDllDirectories : Chỉ định một bộ thư mục mặc định để tìm kiếm khi quá trình gọi tải một DLL.

SetDllDirectory : Sửa đổi đường dẫn tìm kiếm được sử dụng để định vị các tệp DLL cho ứng dụng.

Sử dụng liên kết động trong thời gian tải

Sau khi bạn đã tạo một DLL, bạn có thể sử dụng các chức năng mà nó định nghĩa trong một ứng dụng. Sau đây là một ứng dụng bảng điều khiển đơn giản sử dụng chức năng myPuts được xuất từ Myputs.dll (xem Tạo Thư viện Liên kết Động Đơn giản).

Vì ví dụ này gọi hàm DLL một cách rõ ràng, mô-đun cho ứng dụng phải được liên kết với thư viện nhập Myputs.lib. Để biết thêm thông tin về việc xây dựng DLL, hãy xem tài liệu đi kèm với các công cụ phát triển của bạn.

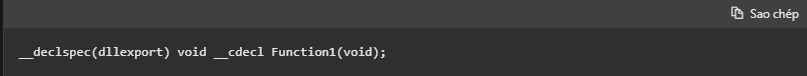


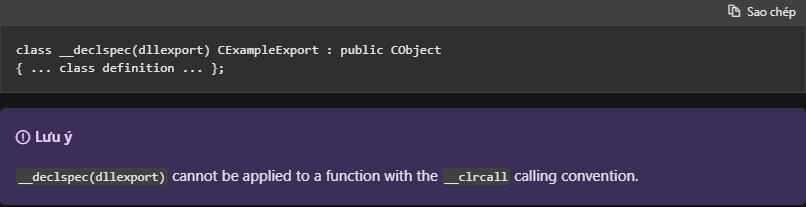
Exporting from a DLL Using \_\_declspec(dllexport)

Bạn có thể xuất dữ liệu, hàm, lớp hoặc hàm thành viên lớp từ DLL bằng cách sử dụng từ khóa \_\_declspec (dllexport). \_\_declspec (dllexport) thêm chỉ thị xuất vào tệp đối tượng để bạn không cần sử dụng tệp .def.

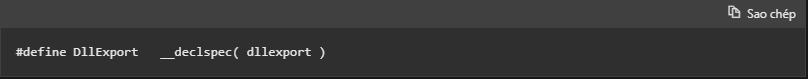
Sự thuận tiện này rõ ràng nhất khi cố gắng xuất các tên hàm C ++ được trang trí. Bởi vì không có đặc điểm kỹ thuật tiêu chuẩn cho trang trí tên, tên của một hàm được xuất có thể thay đổi giữa các phiên bản trình biên dịch. Nếu bạn sử dụng \_\_declspec (dllexport), chỉ cần biên dịch lại tệp DLL và tệp .exe phụ thuộc để giải thích cho bất kỳ thay đổi quy ước đặt tên nào.

Nhiều chỉ thị xuất, chẳng hạn như thứ tự, NONAME và RIÊNG TƯ, chỉ có thể được tạo trong tệp .def và không có cách nào để chỉ định các thuộc tính này mà không có tệp .def. Tuy nhiên, việc sử dụng \_\_declspec (dllexport) ngoài việc sử dụng tệp .def không gây ra lỗi xây dựng.

Để xuất các hàm, từ khóa \_\_declspec (dllexport) phải xuất hiện ở bên trái của từ khóa quy ước gọi, nếu một từ khóa được chỉ định. Ví dụ: 



Khi xây dựng DLL, bạn thường tạo một tệp tiêu đề chứa các nguyên mẫu hàm và / hoặc các lớp mà bạn đang xuất và thêm \_\_declspec (dllexport) vào các khai báo trong tệp tiêu đề. Để làm cho mã của bạn dễ đọc hơn, hãy xác định macro cho \_\_declspec (dllexport) và sử dụng macro với mỗi ký hiệu bạn đang xuất:



DLL injection

**DLL injection** (dịch: Tiêm chích DLL) là một kỹ thuật được sử dụng để chạy [mã code](https://vi.wikipedia.org/wiki/Ng%C3%B4n_ng%E1%BB%AF_m%C3%A1y) trong [không gian địa chỉ](https://vi.wikipedia.org/w/index.php?title=Kh%C3%B4ng_gian_%C4%91%E1%BB%8Ba_ch%E1%BB%89&action=edit&redlink=1) )của một '[tiến trình](https://vi.wikipedia.org/wiki/Ti%E1%BA%BFn_tr%C3%ACnh) khác' thông qua cách ép nó tải một [thư viện liên kết động](https://vi.wikipedia.org/wiki/Dynamic_Link_Library)  DLL injection hay được các 'chương trình bên ngoài' sử dụng để gây ảnh hưởng đến hành vi của một 'chương trình khác' theo cách mà [tác giả](https://vi.wikipedia.org/wiki/L%E1%BA%ADp_tr%C3%ACnh_vi%C3%AAn) của nó không hề tiên liệu hoặc dự định.Ví dụ, 'mã được tiêm' có thể [móc](https://vi.wikipedia.org/w/index.php?title=Hooking&action=edit&redlink=1) vào các lời gọi hàm hệ thống, hoặc đọc nội dung của các hộp văn bản [mật khẩu](https://vi.wikipedia.org/wiki/M%E1%BA%ADt_kh%E1%BA%A9u), những điều vậy không thể được thực hiện theo cách thông thường.Một chương trình mà được sử dụng để tiêm 'mã tùy ý' vào các 'tiến trình tùy ý' thì được gọi là một **DLL injector**.

Trên [Microsoft Windows](https://vi.wikipedia.org/wiki/Microsoft_Windows), có nhiều cách để ép một [tiến trình](https://vi.wikipedia.org/wiki/Ti%E1%BA%BFn_tr%C3%ACnh) phải nạp và thực thi code của một DLL, nằm ngoài ý định của tác giả phần mềm đó:

* Các DLL được liệt kê trong mục [registry](https://vi.wikipedia.org/wiki/Windows_Registry) HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows\AppInit\_DLLs đều được tải vào mọi tiến trình có tải [User32.dll](https://vi.wikipedia.org/w/index.php?title=User32.dll&action=edit&redlink=1) ([en](https://en.wikipedia.org/wiki/User32.dll)) trong lời gọi ban đầu tới DLL đó. Bắt đầu với [Windows Vista](https://vi.wikipedia.org/wiki/Windows_Vista), AppInit\_DLL bị tắt theo mặc định.[]](https://vi.wikipedia.org/wiki/DLL_injection#cite_note-10)Bắt đầu với Windows 7, cơ sở hạ tầng của AppInit\_DLL hỗ trợ [code signing](https://vi.wikipedia.org/w/index.php?title=Code_signing&action=edit&redlink=1) ([en](https://en.wikipedia.org/wiki/code_signing)). Bắt đầu với [Windows 8](https://vi.wikipedia.org/wiki/Windows_8), toàn bộ tính chức năng của AppInit\_DLL đều bị tắt nếu có bật [Secure Boot](https://vi.wikipedia.org/wiki/UEFI), bất chấp code signing hoặc thiết lập registry.
* Các tệp DLL được liệt kê trong registry key HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\AppCertDLLs đều được nạp vào mọi tiến trình có gọi các hàm Win32 API là CreateProcess, CreateProcessAsUser, CreateProcessWithLogonW, CreateProcessWithTokenW và WinExec. Đấy là cách đúng để sử dụng DLL injection một cách hợp lẽ trên phiên bản Windows hiện tại – Windows 10. DLL phải được ký bằng một chứng chỉ hợp lệ.
* Các hàm để thao túng tiến trình như CreatRemoteThread hoặc các kỹ thuật code injection như AtomBombing, có thể được sử dụng để tiêm DLL vào một chương trình sau khi nó đã bắt đầu.
  1. Mở một [handle](https://vi.wikipedia.org/w/index.php?title=Handle_(%C4%91i%E1%BB%87n_to%C3%A1n)&action=edit&redlink=1) ([en](https://en.wikipedia.org/wiki/Handle_(computing))) tới tiến trình mục tiêu. Điều này có thể được thực hiện bằng cách [spawn](https://vi.wikipedia.org/w/index.php?title=Spawn_(%C4%91i%E1%BB%87n_to%C3%A1n)&action=edit&redlink=1) ([en](https://en.wikipedia.org/wiki/Spawn_(computing))) ra tiến trình đấy hoặc bằng cách lần nó dựa trên một đầu mối mà chắc chắn có tồn tại khi tiến trình đó chạy – ví dụ, một [cửa sổ](https://vi.wikipedia.org/w/index.php?title=C%E1%BB%ADa_s%E1%BB%95_(%C4%91i%E1%BB%87n_to%C3%A1n)&action=edit&redlink=1) ([en](https://en.wikipedia.org/wiki/Window_(computing))) có tiêu đề đễ dự đoán, hoặc bằng cách lấy danh sách các 'tiến trình đang chạy'và quét tìm 'tên tệp' của 'tệp thực thi mục tiêu'.
  2. Cấp phát bộ nhớ trong 'tiến trình mục tiêu',] rồi ghi tên của 'DLL cần tiêm' vào bộ nhớ đó.[

Bước này có thể được bỏ qua nếu một 'tên DLL phù hợp' đã có sẵn trong tiến trình mục tiêu. Ví dụ: nếu một tiến trình mà có liên kết đến '[User32.dll](https://vi.wikipedia.org/w/index.php?title=User32.dll&action=edit&redlink=1) ([en](https://en.wikipedia.org/wiki/User32.dll))', '[GDI32.dll](https://vi.wikipedia.org/w/index.php?title=GDI32.dll&action=edit&redlink=1) ([en](https://en.wikipedia.org/wiki/GDI32.dll))', '[Kernel32.dll](https://vi.wikipedia.org/w/index.php?title=Kernel32.dll&action=edit&redlink=1) ([en](https://en.wikipedia.org/wiki/Kernel32.dll))' hoặc bất kỳ thư viện nào khác có tên kết thúc bằng ‘32.dll’, thì có thể tải thư viện có tên '32.dll'. Kỹ thuật này trong quá khứ đã được chứng minh là có hiệu quả để vượt qua hàng rào ngăn ngừa DLL injection của một tiến trình.

* 1. Tạo một [thread](https://vi.wikipedia.org/wiki/Lu%E1%BB%93ng_(%C4%91i%E1%BB%87n_to%C3%A1n)) mới trong 'tiến trình mục tiêu'[[26]](https://vi.wikipedia.org/wiki/DLL_injection#cite_note-26) với 'địa chỉ bắt đầu' của thread được đặt thành địa chỉ của LoadLibrary và đối số được đặt thành địa chỉ của chuỗi văn bản vừa được ghi vào mục tiêu ở bước 2.[[27]](https://vi.wikipedia.org/wiki/DLL_injection#cite_note-27)

Thay vì ghi tên của một 'DLL cần tải' vào mục tiêu và bắt đầu thread mới tại LoadLibrary, người ta có thể viết hẳn một 'mã-để-thực-thi' vào mục tiêu và bắt đầu thread tại mã đó.

* 1. Hệ điều hành sau đó gọi vào thường trình sơ khởi ([initialization](https://vi.wikipedia.org/w/index.php?title=S%C6%A1_kh%E1%BB%9Fi_(l%E1%BA%ADp_tr%C3%ACnh)&action=edit&redlink=1) ([en](https://en.wikipedia.org/wiki/Initialization_(programming))) routine) của 'DLL được tiêm'.

Lưu ý rằng nếu không có sự đề phòng, lối tiếp cận này có thể bị 'tiến trình mục tiêu' phát hiện do các tin báo DLL\_THREAD\_ATTACH được gửi đến mọi 'mô-đun đang chạy' ngay khi một thread bắt đầu.

* Các lời gọi [hook](https://vi.wikipedia.org/w/index.php?title=Hooking&action=edit&redlink=1) ([en](https://en.wikipedia.org/wiki/Hooking)) của Windows như SetWindowsHookEx.
* Sử dụng hàm SuspendThread hoặc NtSuspendThread để tạm đình chỉ tất cả các thread, sau đó sử dụng hàm SetThreadContext hoặc NtSetContextThread để sửa đổi ngữ cảnh của một thread đang tồn tại trong ứng dụng đó để thực thi 'mã được tiêm', từ đó có thể nạp DLL.
* Khai thác các 'hạn chế về thiết kế' trong Windows và các ứng dụng có gọi hàm LoadLibrary hoặc LoadLibraryEx mà không chỉ định 'đường dẫn hợp cách đầy đủ' ([full-qualified path](https://vi.wikipedia.org/w/index.php?title=%C4%90%C6%B0%E1%BB%9Dng_d%E1%BA%ABn_h%E1%BB%A3p_c%C3%A1ch_%C4%91%E1%BA%A7y_%C4%91%E1%BB%A7&action=edit&redlink=1) ([en](https://en.wikipedia.org/wiki/Fully_qualified_name))) đến DLL được tải
* [Shim](https://vi.wikipedia.org/w/index.php?title=Shim_(%C4%91i%E1%BB%87n_to%C3%A1n)&action=edit&redlink=1) ([en](https://en.wikipedia.org/wiki/Shim_(computing))) ở mức hệ điều hành.
* Thay thế một 'DLL đặc thù ứng dụng' bằng một 'bản thay thế giả mạo' có bản [thực hiện](https://vi.wikipedia.org/wiki/Th%E1%BB%B1c_hi%E1%BB%87n) của các hàm được xuất (export) tương tự như bản gốc

### Sử dụng hàm API LoadLibrary

Trong hệ điều hành Windows, có một cách đơn giản để tiêm DLL vào tiến trình khác là sử dụng hàm API CreateRemoteThread. Phương pháp là tạo một thread vào tiến trình đấy từ xa, thread đó sẽ nạp 'DLL mong muốn' vào. Hàm CreateRemoteThread yêu cầu một '[con trỏ hàm](https://vi.wikipedia.org/w/index.php?title=Con_tr%E1%BB%8F_h%C3%A0m&action=edit&redlink=1) ([en](https://en.wikipedia.org/wiki/Function_pointer))' trỏ tới một hàm nằm ở trong 'không gian địa chỉ' của tiến trình mục tiêu – tức là "phải có sẵn" một hàm có chức năng nạp 'DLL mong muốn' ở ngay tiến trình mục tiêu. Có một sự thật có thể được khai thác, đó là *kernel32.dll được ánh xạ tới cùng một địa chỉ trong hầu hết tất cả các tiến trình*. Do đó hàm LoadLibrary của kernel32.dll (có chức năng nạp DLL bất kỳ) cũng sẽ có cùng địa chỉ trong hầu hết tiến trình. Cùng với việc hầu hết các ứng dụng đều liên kết với kernel32.dll nên hàm LoadLibrary luôn có sẵn ở hầu hết các tiến trình. Vì vậy có thể dùng CreateRemoteThread tạo một thread bắt đầu tại hàm LoadLibrary với đối số là đường dẫn tới DLL cần nạp. Xem ví dụ bên dưới chocụ thể: 