**Thứ tự tìm kiếm DLL trong Windows và các vấn đề liên quan**

**1. Cơ chế hoạt động của việc tải DLL trong Windows**

Trong Windows, khi một chương trình cần sử dụng một thư viện động (DLL), hệ điều hành sẽ phải **tìm kiếm và nạp DLL đó** vào bộ nhớ quá trình. Việc này có thể xảy ra theo hai cách chính: **liên kết tĩnh (load-time linking)** hoặc **liên kết động (runtime linking)**. Với liên kết tĩnh, ngay khi chương trình khởi chạy, Windows Loader sẽ tự động tìm và tải tất cả các DLL được khai báo trong import table của file thực thi. Còn với liên kết động, ứng dụng có thể gọi hàm LoadLibrary hoặc LoadLibraryEx vào lúc chạy để nạp một DLL theo tên hoặc đường dẫn.

Khi chỉ cung cấp **tên DLL (không kèm đường dẫn đầy đủ)**, Windows phải lần lượt tìm DLL đó ở nhiều vị trí khác nhau theo một thứ tự xác định sẵn (xem mục 2 bên dưới). Ngược lại, nếu ứng dụng chỉ rõ **đường dẫn đầy đủ** tới DLL (ví dụ C:\Program Files\MyApp\abc.dll), Windows sẽ tải trực tiếp từ đường dẫn đó mà không tìm ở nơi khác. Ngoài ra, trước khi tìm kiếm trên đĩa, Windows còn kiểm tra một số yếu tố đặc biệt:

* **Danh sách module đã nạp của quá trình**: Nếu DLL đã được tải trước đó vào quá trình (dù từ thư mục nào) thì Windows không tải lại nữa mà sử dụng ngay module đã có sẵn (tăng reference count)​

[learn.microsoft.com](https://learn.microsoft.com/en-us/windows/win32/dlls/dynamic-link-library-search-order#:~:text=%2A%20Loaded,KnownDLLs)

.

* **Known DLLs**: Windows duy trì một **danh sách các DLL hệ thống nổi tiếng** trong registry (KnownDLLs, ví dụ như kernel32.dll, user32.dll, v.v.). Nếu yêu cầu load một DLL trùng tên với KnownDLL, Windows sẽ luôn tải bản DLL từ thư mục hệ thống thay vì bất kỳ bản nào khác tìm thấy ở nơi khác​

[learn.microsoft.com](https://learn.microsoft.com/en-us/windows/win32/dlls/dynamic-link-library-search-order#:~:text=,KnownDLLs)

. Cơ chế này ngăn chặn việc *giả mạo* các DLL lõi của Windows bởi các file cùng tên nằm trong thư mục ứng dụng. (Danh sách KnownDLLs nằm ở khóa registry: HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\KnownDLLs​

[learn.microsoft.com](https://learn.microsoft.com/en-us/windows/win32/dlls/dynamic-link-library-search-order#:~:text=,KnownDLLs)

).

Ngoài ra, Windows còn có các cơ chế **chuyển hướng DLL** (DLL redirection), **API Set**, và **Side-by-Side (SxS) manifest** để thay đổi cách tìm và tải DLL trong những tình huống đặc biệt. Ví dụ, ứng dụng có thể đi kèm một file manifest để yêu cầu hệ thống nạp đúng phiên bản DLL phù hợp (tránh “DLL Hell”). Các cơ chế đặc biệt này được xử lý *trước* khi Windows thực hiện tìm kiếm theo thứ tự tiêu chuẩn​

[unit42.paloaltonetworks.com](https://unit42.paloaltonetworks.com/dll-hijacking-techniques/#:~:text=the%20search%20order%20comprising%20both,parts%20in%20Figure%201)

. Sau khi xét hết các yếu tố trên, hệ điều hành mới tiến hành tìm trên đĩa theo *thứ tự tìm kiếm DLL chuẩn* như dưới đây.

**2. Thứ tự tìm kiếm DLL trong Windows**

A diagram of a software process

AI-generated content may be incorrect.

*Hình: Sơ đồ thứ tự tìm kiếm DLL trong Windows, gồm hai nhóm “Vị trí tìm kiếm đặc biệt” (màu xanh lá) và “Vị trí tìm kiếm tiêu chuẩn” (màu xanh dương). Các vị trí đặc biệt (DLL redirection, API sets, SxS manifest, module đã nạp, Known DLLs, dependency graph) được xét trước, sau đó mới đến các vị trí tiêu chuẩn như thư mục ứng dụng, System32, System, Windows, thư mục hiện tại và PATH​*

[*unit42.paloaltonetworks.com*](https://unit42.paloaltonetworks.com/dll-hijacking-techniques/#:~:text=the%20search%20order%20comprising%20both,parts%20in%20Figure%201)

*.*

Khi tìm DLL trên đĩa, Windows lần lượt kiểm tra các **thư mục theo thứ tự cố định**. Thứ tự này mặc định như sau (áp dụng khi **Safe DLL Search Mode** được kích hoạt – xem giải thích thêm bên dưới)​

[support.microsoft.com](https://support.microsoft.com/en-us/topic/secure-loading-of-libraries-to-prevent-dll-preloading-attacks-d41303ec-0748-9211-f317-2edc819682e1#:~:text=1,the%20application%20loaded)

​

[learn.microsoft.com](https://learn.microsoft.com/en-us/windows/win32/dlls/dynamic-link-library-search-order#:~:text=7,by%20the%20App%20Paths%20registry)

:

1. **Thư mục của chương trình đang chạy** – thư mục chứa file .exe của ứng dụng hiện tại​

[support.microsoft.com](https://support.microsoft.com/en-us/topic/secure-loading-of-libraries-to-prevent-dll-preloading-attacks-d41303ec-0748-9211-f317-2edc819682e1#:~:text=1,the%20application%20loaded)

. (Đây thường là nơi ứng dụng đặt các DLL riêng đi kèm, nếu có. Windows luôn ưu tiên tìm ở đây trước tiên.)

1. **Thư mục Hệ thống (System32)** – thư mục hệ thống 32-bit của Windows, thường là C:\Windows\System32​

[support.microsoft.com](https://support.microsoft.com/en-us/topic/secure-loading-of-libraries-to-prevent-dll-preloading-attacks-d41303ec-0748-9211-f317-2edc819682e1#:~:text=1,the%20application%20loaded)

. Đây là nơi chứa hầu hết các DLL hệ điều hành.

1. **Thư mục Hệ thống 16-bit (System)** – thường là C:\Windows\System​

[support.microsoft.com](https://support.microsoft.com/en-us/topic/secure-loading-of-libraries-to-prevent-dll-preloading-attacks-d41303ec-0748-9211-f317-2edc819682e1#:~:text=2)

. Thư mục này tồn tại vì lý do tương thích (trên các hệ thống Windows 32-bit) để chứa các thư viện 16-bit hoặc dùng cho một số thành phần cũ.

1. **Thư mục Windows** – thư mục gốc cài đặt Windows, thường là C:\Windows​

[support.microsoft.com](https://support.microsoft.com/en-us/topic/secure-loading-of-libraries-to-prevent-dll-preloading-attacks-d41303ec-0748-9211-f317-2edc819682e1#:~:text=3.%20The%2016)

.

1. **Thư mục làm việc hiện tại (Current Working Directory - CWD)** – thư mục hiện thời của quá trình tại thời điểm gọi load DLL​

[support.microsoft.com](https://support.microsoft.com/en-us/topic/secure-loading-of-libraries-to-prevent-dll-preloading-attacks-d41303ec-0748-9211-f317-2edc819682e1#:~:text=4)

. Lưu ý thư mục này có thể *khác* với thư mục chương trình đang chạy (ví dụ nếu ứng dụng đổi thư mục làm việc trong khi chạy).

1. **Các thư mục trong biến môi trường %PATH%** – lần lượt từng thư mục được liệt kê trong biến môi trường PATH của người dùng hoặc hệ thống​

[support.microsoft.com](https://support.microsoft.com/en-us/topic/secure-loading-of-libraries-to-prevent-dll-preloading-attacks-d41303ec-0748-9211-f317-2edc819682e1#:~:text=5,CWD)

. Đây thường bao gồm các thư mục cài đặt chương trình, thư mục hệ thống, v.v. (ngoại trừ những thư mục đã kể trên nếu đã có trong PATH, thì không lặp lại tìm nữa).

Windows sẽ lần lượt tìm file DLL theo đúng thứ tự trên. Nếu tìm thấy file phù hợp ở bước nào thì sẽ dừng lại và tải DLL đó. Ngược lại, nếu không tìm thấy ở tất cả các vị trí, hàm LoadLibrary sẽ trả về lỗi.

**Safe DLL Search Mode**: Trên các phiên bản Windows hiện đại, **chế độ tìm kiếm DLL an toàn** được **kích hoạt mặc định** (bắt đầu từ Windows XP SP2 trở lên)​

[learn.microsoft.com](https://learn.microsoft.com/en-us/windows/win32/dlls/dynamic-link-library-security#:~:text=,Link%20Library%20Search%20Order)

. Chế độ này chính là thứ tự chuẩn được liệt kê ở trên, trong đó **thư mục hiện tại (CWD) được đẩy xuống gần cuối** danh sách (chỉ trước PATH). Việc này nhằm giảm rủi ro bảo mật (vì thư mục hiện tại có thể là nơi không tin cậy, xem mục 3). Ngược lại, nếu tắt Safe DLL Search Mode, thứ tự tìm kiếm sẽ thay đổi: **thư mục hiện tại sẽ được tìm *sớm hơn***. Cụ thể, khi Safe Mode bị vô hiệu, thứ tự sẽ là: thư mục ứng dụng, **sau đó đến thư mục hiện tại**, rồi mới đến System32, System, Windows, và PATH​

[learn.microsoft.com](https://learn.microsoft.com/en-us/windows/win32/dlls/dynamic-link-library-search-order#:~:text=If%20safe%20DLL%20search%20mode,from%20which%20the%20application%20loaded)

. Nói cách khác, CWD sẽ nhảy từ vị trí thứ 5 lên vị trí thứ 2. Việc tắt Safe DLL Search Mode không được khuyến nghị trừ khi có lý do đặc biệt, bởi nó làm tăng nguy cơ nạp nhầm DLL độc hại.

**Lưu ý**: Ở cấp lập trình, Windows còn cung cấp các biến thể hàm như LoadLibraryEx với cờ LOAD\_WITH\_ALTERED\_SEARCH\_PATH để thay đổi thứ tự tìm kiếm cho một lần load nhất định, hoặc các hàm như AddDllDirectory/SetDllDirectory để bổ sung/điều chỉnh đường dẫn tìm kiếm DLL cho *quá trình hiện tại*. Những kỹ thuật này cho phép nhà phát triển kiểm soát chi tiết hơn cách ứng dụng tìm DLL, thay vì hoàn toàn dựa vào thứ tự mặc định của hệ thống.

**3. Tác động bảo mật của thứ tự tìm kiếm DLL**

Thứ tự tìm kiếm DLL tưởng như chỉ là chi tiết kỹ thuật, nhưng thực tế nó **ảnh hưởng lớn đến bảo mật** của ứng dụng Windows. Lý do là nếu kẻ tấn công có thể đặt được một **file DLL độc hại** ở **một trong các vị trí được tìm trước** bản DLL hợp lệ, ứng dụng sẽ vô tình nạp và thực thi mã độc đó. Dưới đây là hai khía cạnh quan trọng về rủi ro và lỗ hổng liên quan:

* **Rủi ro “nạp nhầm” DLL do thứ tự tìm kiếm**: Giả sử một ứng dụng dự kiến sử dụng ABC.dll (một thư viện hợp lệ nằm trong System32). Nếu ứng dụng gọi LoadLibrary("ABC.dll") mà không chỉ rõ đường dẫn, Windows sẽ tìm theo thứ tự đã nêu. Kẻ tấn công có thể lợi dụng bằng cách đặt một file ABC.dll giả mạo vào thư mục mà Windows sẽ tìm **trước System32** – ví dụ đặt ngay trong thư mục chương trình hoặc thư mục hiện tại của ứng dụng. Khi đó, ứng dụng sẽ nạp phải DLL giả mạo này thay vì DLL thật từ System32​

[support.microsoft.com](https://support.microsoft.com/en-us/topic/secure-loading-of-libraries-to-prevent-dll-preloading-attacks-d41303ec-0748-9211-f317-2edc819682e1#:~:text=,application%20and%20controls%20the%20CWD)

. Kết quả là mã độc trong DLL sẽ chạy bên trong quá trình ứng dụng, với cùng **quyền hạn của người dùng hiện tại**​

[support.microsoft.com](https://support.microsoft.com/en-us/topic/secure-loading-of-libraries-to-prevent-dll-preloading-attacks-d41303ec-0748-9211-f317-2edc819682e1#:~:text=CWD,has%20permission%20to%20do%20this)

(hoặc tệ hơn, nếu ứng dụng chạy với quyền admin/system thì DLL độc cũng sẽ chạy với quyền đó). Rủi ro đặc biệt cao khi **thư mục hiện tại (CWD)** của ứng dụng trỏ đến một thư mục không tin cậy. Chẳng hạn, có trường hợp ứng dụng mở một file dữ liệu từ ổ mạng hoặc USB, làm CWD trỏ đến thư mục chứa file đó – nếu kẻ tấn công đã đặt sẵn DLL độc tên trùng với DLL mà ứng dụng cần trong cùng thư mục, ứng dụng sẽ nạp nhầm​

[support.microsoft.com](https://support.microsoft.com/en-us/topic/secure-loading-of-libraries-to-prevent-dll-preloading-attacks-d41303ec-0748-9211-f317-2edc819682e1#:~:text=,application%20and%20controls%20the%20CWD)

. Lỗ hổng này thường được gọi là **“DLL preloading”** hay **“binary planting”**, từng được công bố rộng rãi vào năm 2010, ảnh hưởng đến nhiều phần mềm phổ biến.

* **Lỗ hổng DLL Hijacking (DLL Search Order Hijacking)**: Đây là thuật ngữ chung chỉ việc **chiếm quyền điều khiển quá trình load DLL** của ứng dụng theo cách bất hợp pháp. Hacker khai thác chính thứ tự tìm kiếm DLL để **chèn một DLL độc hại thay thế** cho DLL hợp lệ mà ứng dụng mục tiêu sử dụng​

[upguard.com](https://www.upguard.com/blog/dll-hijacking#:~:text=By%20replacing%20a%20required%20DLL,loads%2C%20activating%20its%20malicious%20operations)

​

[upguard.com](https://www.upguard.com/blog/dll-hijacking#:~:text=exploiting%20the%20way%20some%20Windows,DLL)

. Có nhiều biến thể tấn công DLL hijacking, nhưng phổ biến nhất là **Search Order Hijacking** – đặt một DLL cùng tên ở thư mục ưu tiên cao hơn. Kỹ thuật này cho phép mã độc **chạy ẩn dưới tiến trình hợp pháp**: chương trình chính vẫn là file exe hợp lệ của hãng phát triển, nhưng nó đang “ôm” theo DLL độc mà người dùng khó phát hiện​

[crowdstrike.com](https://www.crowdstrike.com/en-us/blog/4-ways-adversaries-hijack-dlls/#:~:text=What%20is%20DLL%20Hijacking%3F)

. DLL hijacking có thể được sử dụng cho *nhiều mục đích tấn công*, từ **thực thi mã tùy ý**, **leo thang đặc quyền**, cho đến **duy trì sự hiện diện (persistency)** trên hệ thống​

[crowdstrike.com](https://www.crowdstrike.com/en-us/blog/4-ways-adversaries-hijack-dlls/#:~:text=What%20is%20DLL%20Hijacking%3F)

. Ví dụ, malware có thể đặt một DLL độc với tên trùng tên thư viện mà một dịch vụ hệ thống chạy khi khởi động cần, trong thư mục dịch vụ đó – kết quả là mỗi lần máy khởi động, dịch vụ nạp DLL độc và cho phép attacker duy trì quyền kiểm soát. Đã có rất nhiều CVE liên quan đến DLL hijacking được báo cáo. Chẳng hạn, lỗ hổng CVE-2024-7061 cho phép leo thang quyền hạn trong phần mềm Okta Verify bằng cách chèn một DLL tùy ý vào thư mục mà ứng dụng sẽ tìm đến trước khi nạp DLL hợp lệ của nó​

[github.com](https://github.com/advisories/GHSA-jh6g-rh4q-hcw2#:~:text=CVE,2)

. Kỹ thuật DLL hijacking không hề mới – nó **xuất hiện từ thời Windows 2000** và đến nay vẫn là phương thức tấn công hiệu quả nếu ứng dụng không được bảo vệ đúng mức​

[upguard.com](https://www.upguard.com/blog/dll-hijacking#:~:text=DLL%20hijacking%20is%20not%20an,cybercriminals%20since%20Windows%202000%20launched)

.

Tóm lại, nếu thứ tự tìm kiếm DLL không được kiểm soát chặt chẽ, kẻ xấu có thể lợi dụng để **“đi đường tắt” trong trình tự load DLL**, khiến ứng dụng **chạy nhầm mã độc** thay vì mã hợp lệ. Lỗ hổng này nguy hiểm vì **khó phát hiện** – phần mềm vẫn khởi chạy bình thường, *chỉ có điều bên trong nó đã thực thi thêm mã của kẻ tấn công*.

**4. Phòng chống và biện pháp bảo vệ**

Để bảo vệ hệ thống và ứng dụng khỏi các rủi ro liên quan đến thứ tự tìm kiếm DLL và DLL hijacking, chúng ta có thể áp dụng nhiều biện pháp kết hợp. Dưới đây là một số phương pháp quan trọng:

* **Ký số DLL để xác thực nguồn gốc**: Việc **ký số** (digital signature) cho các DLL bởi nhà phát triển uy tín giúp xác thực rằng DLL chưa bị sửa đổi kể từ khi phát hành. Hệ điều hành Windows có cơ chế Authenticode cho phép kiểm tra chữ ký số của file thực thi/DLL. Mặc dù Windows không tự động chặn nạp DLL chưa được ký, nhưng việc tất cả DLL hợp lệ đều có chữ ký số sẽ giúp các giải pháp bảo mật (SmartScreen, antivirus, AppLocker, v.v.) hoặc chính ứng dụng kiểm tra trước khi load. Doanh nghiệp cũng có thể thiết lập chính sách **cho phép chạy code đã ký bởi nhà phát hành tin cậy** (Code Signing Policy), nhờ đó ngăn chặn DLL lạ (không có chữ ký hợp lệ) được tải vào quá trình. Tóm lại, ký số DLL không trực tiếp thay đổi thứ tự tìm kiếm nhưng là lớp bảo vệ bổ sung để phát hiện và ngăn chặn DLL giả mạo.
* **Sử dụng hàm SetDllDirectory() để kiểm soát đường dẫn tìm kiếm**: Lập trình viên có thể chủ động kiểm soát việc tìm kiếm DLL trong ứng dụng của mình bằng cách sử dụng hàm Windows API SetDllDirectory. Hàm này cho phép **thêm một thư mục tùy ý vào danh sách tìm kiếm DLL của quá trình**, đồng thời **loại bỏ thư mục hiện tại (CWD)** khỏi danh sách mặc định​

[support.microsoft.com](https://support.microsoft.com/en-us/topic/secure-loading-of-libraries-to-prevent-dll-preloading-attacks-d41303ec-0748-9211-f317-2edc819682e1#:~:text=Recommendation)

. Ví dụ, gọi SetDllDirectory(L"") với chuỗi rỗng sẽ **xoá CWD khỏi đường dẫn tìm kiếm** (tức là Windows sẽ *bỏ qua hoàn toàn* thư mục hiện tại khi tìm DLL)​

[support.microsoft.com](https://support.microsoft.com/en-us/topic/secure-loading-of-libraries-to-prevent-dll-preloading-attacks-d41303ec-0748-9211-f317-2edc819682e1#:~:text=Recommendation)

. Đây là khuyến nghị của Microsoft nhằm ngăn chặn kiểu tấn công đặt DLL ở CWD. Ngược lại, nếu ứng dụng có một thư mục chuyên biệt chứa plugin/DLL riêng, ta có thể gọi SetDllDirectory(L"C:\\MyApp\\DLLs") để Windows **chỉ tìm DLL trong thư mục đó (và các vị trí hệ thống)**, không đi lan man các nơi khác. Lưu ý rằng SetDllDirectory ảnh hưởng **toàn cục đến quá trình**, nên nên gọi nó **sớm** (trước khi load bất kỳ DLL nào) và không nên thay đổi qua lại nhiều lần (sẽ ảnh hưởng đến các thread khác)​

[learn.microsoft.com](https://learn.microsoft.com/en-us/windows/win32/dlls/dynamic-link-library-security#:~:text=,carefully%20to%20identify%20any%20incompatibilities)

. Dưới đây là ví dụ minh họa cách sử dụng SetDllDirectory trong code C++:

#include <windows.h>

int main() {

// Loại bỏ thư mục làm việc hiện tại khỏi việc tìm kiếm DLL

SetDllDirectory(L"");

// Thêm thư mục chứa plugin của ứng dụng vào danh sách tìm kiếm

SetDllDirectory(L"C:\\Program Files\\MyApp\\plugins");

// Thử tải một DLL tên là MyPlugin.dll (sẽ chỉ tìm trong thư mục trên, System32,...)

HMODULE h = LoadLibrary(L"MyPlugin.dll");

if (!h) {

wprintf(L"Kh\u00f4ng th\u1ec3 t\u1ea3i \u0111\u01b0\u1ee3c DLL\\n");

} else {

wprintf(L"DLL \u0111\u00e3 \u0111\u01b0\u1ee3c n\u1ea1p th\u00e0nh c\u00f4ng!\\n");

}

// ... (tiếp tục sử dụng DLL)

return 0;

}

Trong ví dụ trên, sau khi gọi SetDllDirectory, thư mục hiện tại sẽ không còn được xét đến khi gọi LoadLibrary. Thay vào đó, Windows chỉ tìm "MyPlugin.dll" trong *thư mục plugin của ứng dụng* (đã chỉ định) và sau đó đến System32, System, Windows,... Nếu MyPlugin.dll không có ở các nơi đó, việc nạp sẽ thất bại (tránh được kịch bản nạp nhầm từ CWD hoặc PATH).

* **Kích hoạt Safe DLL Search Mode**: Đảm bảo rằng **Safe DLL Search Mode luôn được bật** trên hệ thống của bạn (mặc định đã bật trên hầu hết máy, trừ khi ai đó tắt đi). Chế độ này, như đã nói, sẽ đặt thư mục hiện tại xuống sau thư mục Windows trong thứ tự tìm kiếm, giúp **tăng khả năng tìm thấy DLL hợp lệ trước khi quét đến CWD**​

[learn.microsoft.com](https://learn.microsoft.com/en-us/windows/win32/dlls/dynamic-link-library-security#:~:text=,Link%20Library%20Search%20Order)

. Safe mode được kiểm soát bởi giá trị registry: HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Control\Session Manager\SafeDllSearchMode (DWORD=1 là bật, =0 là tắt)​

[learn.microsoft.com](https://learn.microsoft.com/en-us/windows/win32/dlls/dynamic-link-library-security#:~:text=,Link%20Library%20Search%20Order)

. Trừ phi có yêu cầu tương thích đặc biệt, không nên tắt chế độ này. Trong môi trường doanh nghiệp, có thể dùng Group Policy hoặc các công cụ quản lý registry để đảm bảo máy khách không bị ai vô hiệu hóa Safe DLL Search Mode.

* **Sử dụng các công cụ kiểm tra thứ tự tìm kiếm DLL**: Để đánh giá xem ứng dụng có thể bị ảnh hưởng bởi search order hay không, ta nên kiểm tra quá trình load DLL của nó. Một công cụ hữu ích là **Microsoft Process Monitor (ProcMon)** của Sysinternals. ProcMon có thể theo dõi **mọi lần truy cập file/DLL** của một tiến trình. Kỹ thuật thông thường là chạy ứng dụng cần kiểm tra cùng ProcMon, sau đó **lọc sự kiện** cho tiến trình đó với các Operation liên quan đến load DLL (ví dụ filter Operation is CreateFile **hoặc** Operation is LoadImage)​

[learn.microsoft.com](https://learn.microsoft.com/en-us/windows/win32/dlls/dynamic-link-library-security#:~:text=1,Operation%20is%20LoadImage)

. Khi ứng dụng chạy và load các DLL, ProcMon sẽ ghi lại *thứ tự các đường dẫn mà hệ thống thử tìm file DLL*. Nhờ đó, bạn có thể thấy liệu ứng dụng có đang tìm DLL ở những nơi không mong muốn (ví dụ: thấy ProcMon ghi nhận ứng dụng tìm XYZ.dll trong C:\Users\Public\Downloads\... tức thư mục hiện tại chẳng hạn – dấu hiệu không an toàn)​

[learn.microsoft.com](https://learn.microsoft.com/en-us/windows/win32/dlls/dynamic-link-library-security#:~:text=5,a%20vulnerability%20in%20your%20application)

. Nếu phát hiện hành vi đáng ngờ, đó có thể là *điểm yếu để kẻ tấn công lợi dụng*​

[learn.microsoft.com](https://learn.microsoft.com/en-us/windows/win32/dlls/dynamic-link-library-security#:~:text=match%20at%20L170%205,a%20vulnerability%20in%20your%20application)

. Bên cạnh ProcMon, một số công cụ khác cũng hữu dụng:

* + **Process Explorer** (Sysinternals) cho phép liệt kê **toàn bộ DLL đã nạp** trong một quá trình, giúp bạn kiểm tra mỗi DLL được load từ đường dẫn nào.
  + **Dependency Walker** hoặc **dumpbin** (Visual Studio) giúp liệt kê các **phụ thuộc DLL** của một ứng dụng. Từ danh sách này, bạn có thể kiểm tra từng DLL xem nó có nằm ở vị trí an toàn chưa, và dự đoán nếu thiếu DLL thì ứng dụng sẽ tìm ở đâu.
  + Các **công cụ scanner bảo mật**: Nhiều trình quét lỗ hổng có khả năng phát hiện DLL hijacking. Chúng thường kiểm tra ứng dụng bằng cách tạo các file DLL giả tại các vị trí nhạy cảm (như CWD) để xem có được nạp hay không. Ví dụ, **DLL Hijack Auditor** (của Trustwave) là một công cụ từng được dùng để tự động tìm các ứng dụng dễ bị DLL hijacking bằng cách lợi dụng search order.
* **Biện pháp bảo vệ khác**: Ở mức hệ thống, quản trị viên có thể áp dụng thêm một số thiết lập để giảm nguy cơ. Một ví dụ là thiết lập giá trị registry CWDIllegalInDllSearch để ngăn việc tìm kiếm DLL ở thư mục hiện tại trong các tình huống nhất định (như chặn CWD là ổ mạng) theo hướng dẫn của Microsoft​

[support.microsoft.com](https://support.microsoft.com/en-us/topic/secure-loading-of-libraries-to-prevent-dll-preloading-attacks-d41303ec-0748-9211-f317-2edc819682e1#:~:text=%23%20SearchPath)

​

[support.microsoft.com](https://support.microsoft.com/en-us/topic/secure-loading-of-libraries-to-prevent-dll-preloading-attacks-d41303ec-0748-9211-f317-2edc819682e1#:~:text=%2A%20The%2016)

. Ngoài ra, cơ chế **Application Whitelisting** (như AppLocker hoặc Windows Defender Application Control) có thể được cấu hình để chặn không cho tiến trình hợp lệ load các DLL không đáng tin (ví dụ chỉ cho phép load DLL từ C:\Windows\System32 hoặc thư mục cài đặt chính thức của ứng dụng). Việc **cập nhật phần mềm thường xuyên** cũng rất quan trọng – nhiều lỗ hổng DLL hijacking đã được nhà cung cấp phần mềm vá, nên giữ cho ứng dụng và hệ điều hành lên phiên bản mới nhất sẽ giảm nguy cơ bị khai thác.

**Tóm lại**, hiểu rõ thứ tự tìm kiếm DLL và các rủi ro kèm theo giúp chúng ta thiết kế ứng dụng an toàn hơn và cấu hình hệ thống hợp lý hơn. Bằng cách áp dụng các biện pháp như ký số, kiểm soát đường dẫn tìm kiếm (SetDllDirectory, …), bật Safe DLL Search Mode và dùng công cụ giám sát, chúng ta có thể **phòng tránh hiệu quả** các cuộc tấn công DLL hijacking, bảo vệ hệ thống Windows trước những “kẻ xâm nhập ẩn mình trong thư viện”.

**Nguồn tham khảo:** Dynamic-Link Library Search Order​

[support.microsoft.com](https://support.microsoft.com/en-us/topic/secure-loading-of-libraries-to-prevent-dll-preloading-attacks-d41303ec-0748-9211-f317-2edc819682e1#:~:text=1,the%20application%20loaded)

​

[support.microsoft.com](https://support.microsoft.com/en-us/topic/secure-loading-of-libraries-to-prevent-dll-preloading-attacks-d41303ec-0748-9211-f317-2edc819682e1#:~:text=,application%20and%20controls%20the%20CWD)

; Dynamic-Link Library Security​

[learn.microsoft.com](https://learn.microsoft.com/en-us/windows/win32/dlls/dynamic-link-library-security#:~:text=,Link%20Library%20Search%20Order)

​

[learn.microsoft.com](https://learn.microsoft.com/en-us/windows/win32/dlls/dynamic-link-library-security#:~:text=,carefully%20to%20identify%20any%20incompatibilities)

; Microsoft Support (DLL preloading attacks)​

[support.microsoft.com](https://support.microsoft.com/en-us/topic/secure-loading-of-libraries-to-prevent-dll-preloading-attacks-d41303ec-0748-9211-f317-2edc819682e1#:~:text=1,the%20application%20loaded)

​

[support.microsoft.com](https://support.microsoft.com/en-us/topic/secure-loading-of-libraries-to-prevent-dll-preloading-attacks-d41303ec-0748-9211-f317-2edc819682e1#:~:text=,application%20and%20controls%20the%20CWD)

; UpGuard Security Blog​

[upguard.com](https://www.upguard.com/blog/dll-hijacking#:~:text=exploiting%20the%20way%20some%20Windows,DLL)

​

[upguard.com](https://www.upguard.com/blog/dll-hijacking#:~:text=DLL%20hijacking%20is%20not%20an,cybercriminals%20since%20Windows%202000%20launched)

; CrowdStrike Blog​

[crowdstrike.com](https://www.crowdstrike.com/en-us/blog/4-ways-adversaries-hijack-dlls/#:~:text=What%20is%20DLL%20Hijacking%3F)

; Palo Alto Unit42 Blog​

[unit42.paloaltonetworks.com](https://unit42.paloaltonetworks.com/dll-hijacking-techniques/#:~:text=the%20search%20order%20comprising%20both,parts%20in%20Figure%201)

.