DLL Injection and x86 Hooking Demystified

Giorgio Gori

Sources:

What is a DLL?
https://support.microsoft.com/en-ca/kb/815065

Windows DLL Injection Basics by Brad Antoniewicz
http://blog.opensecurityresearch.com/2013/01/windows-dll-injection-basics.html

x86 API Hooking Demystified by Jurriaan Bremer
http://jbremer.org/x86-api-hooking-demystified/
What is a DLL?

A DLL - Dynamic Link Library - is a library that contains code and data that can be used by more than one program at the same time.

- Uses fewer resources
- Promotes modular architecture
- Eases deployment and installation
Creating a DLL

BOOL APIENTRY DllMain(HANDLE hModule,
            DWORD ul_reason_for_call, LPVOID lpReserved ) {

    switch ( ul_reason_for_call ) {
        case DLL_PROCESS_ATTACHED: // A process is loading the DLL.
        case DLL_THREAD_ATTACHED:  // A process is creating a new thread.
        case DLL_THREAD_DETACH:    // A thread exits normally.
        case DLL_PROCESS_DETACH:   // A process unloads the DLL.
            break;
    }

    return TRUE;
}

extern __declspec(dllexport) void HelloWorld() {
    MessageBox( NULL, TEXT("Hello World"), TEXT("In a DLL"), MB_OK);
}
Using a DLL

• Load-time dynamic linking
  Provide a header (.h) and library (.lib) at compile and link time. Linker will provide information to resolve the DLL functions at load time.

```c
#include "MyDLL.h"

int main() {
    HelloWorld();
    return 0;
}
```
Using a DLL

- Run-time dynamic linking

Call LoadLibrary(...) and GetProcAddress(...) at run time, then call the function by address.

```c
int main() {
    HMODULE dll = LoadLibrary("MyDLL.dll");
    if (dll != NULL) {
        FARPROC HelloWorld = GetProcAddress(dll, "HelloWorld");
        if (HelloWorld != NULL)
            HelloWorld();

        FreeLibrary(dll);
    }
    return 0;
}
```
DLL Injection

Invoke `LoadLibrary` from the target process

Create a Thread, use `LoadLibrary` as entry point, and the dll path as argument
DLL Injection

1. Attach to the target process.
2. Allocate memory within the process.
3. Copy DLL path into the process memory and find LoadLibrary address.
4. Execute your DLL.
1. Attach
   OpenProcess();

2. Allocate Memory
   VirtualAllocEx();

3. Copy DLL / Determine Addr
   WriteProcessMemory();
   GetProcAddress(..., "LoadLibrary")

4. Execute
   CreateRemoteThread(process_handle, ...
   LoadLibraryPtr, PathPtr, ...);
DLL Proxying, DLL Hijacking

• Both work by impersonating the legitimate DLL and (typically) relaying functionality to it. They can be used both to extend functionality and as a malicious attack vector.

• **Proxying:** Rename the legitimate DLL, replace with your own.

• **Hijacking:** Abuse Windows' DLL Search order to load your DLL before the legitimate one.
DLL Injection: Why?

- Read and write process memory
- Execute custom code, invoke existing functions
- Patch binary code, add hooks
x86 Hooking

Change the byte code to alter the execution. Common uses include:

- Debugging.
- Profiling.
- Extending functionality.
- Execute general "on event" code.
function_A:
0x401000: push ebp
0x401001: mov ebp, esp
0x401003: sub esp, 0x40
0x401006: push ebx
0x401007: mov ebx, dword [esp+0x0c]
...
function_A:
0x401000: push ebp
0x401001: mov ebp, esp
0x401003: sub esp, 0x40
0x401006: push ebx
0x401007: mov ebx, dword [esp+0x0c]
...

function_A:
0x401000: jmp function_B
0x401005: nop
0x401006: push ebx
0x401007: mov ebx, dword [esp+0x0c]
...
<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x401000</td>
<td>jmp function_B</td>
</tr>
<tr>
<td>0x401005</td>
<td>nop</td>
</tr>
<tr>
<td>0x401006</td>
<td>push ebx</td>
</tr>
<tr>
<td>0x401007</td>
<td>mov ebx, dword [esp+0x0c]</td>
</tr>
</tbody>
</table>

function_A:
<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x401800</td>
<td>push ebp</td>
</tr>
<tr>
<td>0x401801</td>
<td>mov ebp, esp</td>
</tr>
<tr>
<td>0x401800</td>
<td>sub esp, 0x40</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x401820</td>
<td>call function_A_gate</td>
</tr>
<tr>
<td>0x401825</td>
<td>... snip ...</td>
</tr>
<tr>
<td>0x401836</td>
<td>retn</td>
</tr>
<tr>
<td>0x401006</td>
<td>push ebx</td>
</tr>
</tbody>
</table>

function_A_gate:
<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x402000</td>
<td>push ebp</td>
</tr>
<tr>
<td>0x402001</td>
<td>mov ebp, esp</td>
</tr>
<tr>
<td>0x402003</td>
<td>sub esp, 0x40</td>
</tr>
<tr>
<td>0x402006</td>
<td>jmp function_A + 6</td>
</tr>
</tbody>
</table>
Hooking example

- Game does not support clickable links. Players have to click, select, copy, paste in web browser.
- We follow the call from the input handler to the UI creation.
- Hook the function that creates the UI element.
- Open in web browser if the name is a URL.
Detour End

```
0A32F544 .73 EF JNB SHORT GWToolbo.0A32F535
0A32F546 > 6A 01 PUSH 1
0A32F548 6A 00 PUSH 0
0A32F54A 57 PUSH EDI
0A32F54C 68 F474430A PUSH GWToolbo.0A4374F4
0A32F54D 6A 00 PUSH 0
0A32F554 FF15 38D43D0A CALL DWORD PTR DS:[<&SHELL32.ShellEx] ShellExecuteW
0A32F555 5F 5F POP EDI
0A32F557 5B 5F POP EDI
0A32F559 5B 5B POP EBX
0A32F55B 8BE5 MOV ESP,EBP
0A32F55D 5D 5D POP EBX
0A32F55F 5C C3 RETN
0A32F561 > E8 FAE8FFFF CALL GWToolbo.GWCA::GWCAManager::GWCA
0A32F562 8B4D FC MOV ECX, DWORD PTR SS:[EBP-4]
0A32F564 8BD3 MOV EDX,EBX
0A32F566 8B40 44 MOV EAX, DWORD PTR DS:[EAX+44]
0A32F568 FF00 CALL EAX
0A32F570 5F 5F POP EDI
0A32F572 5B 5F POP EDI
0A32F574 5B 5B POP EBX
0A32F576 8BE5 MOV ESP,EBP
0A32F578 5D 5D POP EBX
0A32F57A 5C C3 RET

EAX=1916DD20

Breakpoint at GWToolbo.0A32F56E (GWCA::ChatMgr::det_openTemplate+8E)
```
Stolen Bytes
DirectX EndScene Hooking

Game Mods

Steam Overlay

Performance Monitors

FPS Counters
DLL injection and x86 hooking demystified

Other topics include:

• Advanced / Stealth injection techniques
• Integrity of execution during hook installation
• Hook restoration / cleanup
• Hooking detection (anti-cheat) and advanced hooking methods
• Multiple layers of hooks
• Prevent hook recursion
• Hooking different calling conventions and class methods

Sources:

What is a DLL?
https://support.microsoft.com/en-ca/kb/815065

Windows DLL Injection Basics by Brad Antoniewicz
http://blog.opensecurityresearch.com/2013/01/windows-dll-injection-basics.html

x86 API Hooking Demystified by Jurriaan Bremer
http://jbremer.org/x86-api-hooking-demystified/