DLL Injection

CONTACT@ADAMFURMANEK.PL
HTTP://BLOG.ADAMFURMANEK.PL
@FURMANEKADAM
About me

.NET developer for 5 years.
Scala Developer at Amazon.
Blogger, public speaker.
Author of .NET Internals Cookbook.

http://blog.adamfurmanek.pl
contact@adamfurmanek.pl
@furmanekadam
Agenda

What and Why
Preliminaries
How + Demos
Summary
What and Why
What we are going to do

1 – inject DLL
2 – execute code

Our process

Our DLL

Target
What we are going to do

We want to execute **our** code in different (**target**) process. This means:

◦ Our code should be able to access target process’ descriptors (memory, security tokens etc.)
◦ Our code should be able to create, modify, and remove handlers, pointers, and resources in target process
◦ In other words, our code should pretend to be normal part of target process

We want to do it by injecting DLL

We **are not modifying** the target process’ source code (especially, we are not recompiling the target)

We control the machine (however, we might not be administrators)

We want the whole process to be clean, safe, and reliable
Demos
Preliminaries
Preliminaries

Virtual Address Space, Pagination

Dynamic Linked Libraries
  ◦ What is it
  ◦ Process’ Address Space
  ◦ Rebasing Modules, Binding Modules
  ◦ Address Space Layout Randomization

Global Loader Lock
## Virtual Address Space

Every process has its own address space.

<table>
<thead>
<tr>
<th>Partition</th>
<th>x86 32-Bit Windows</th>
<th>x86 32-Bit Windows with 3 GB User-Mode</th>
<th>x64 64-Bit Windows</th>
<th>IA-64 64-Bit Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL-Pointer Assignment</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td>0x0000000000000000</td>
<td>0x0000000000000000</td>
</tr>
<tr>
<td></td>
<td>0x0000FFFF</td>
<td>0x0000FFFF</td>
<td>0x000000000000FFFF</td>
<td>0x000000000000FFFF</td>
</tr>
<tr>
<td>User-Mode</td>
<td>0x00010000</td>
<td>0x00010000</td>
<td>0x0000000000010000</td>
<td>0x0000000000010000</td>
</tr>
<tr>
<td></td>
<td>0x7FFEFFFF</td>
<td>0xBFFEFFFF</td>
<td>0x0000007FFF000000</td>
<td>0x0000006FBF000000</td>
</tr>
<tr>
<td>64-KB Off-Limits</td>
<td>0x7FFFF0000</td>
<td>0xBFFFF0000</td>
<td>0x0000007FF000000</td>
<td>0x0000006FBF000000</td>
</tr>
<tr>
<td></td>
<td>0x7FFFFFFFF</td>
<td>0xBFFFFFFFF</td>
<td>0x0000007FFFFFFFF</td>
<td>0x0000006FBFFFFFFFF</td>
</tr>
<tr>
<td>Kernel-Mode</td>
<td>0x80000000</td>
<td>0xC0000000</td>
<td>0x0000008000000000</td>
<td>0x0000006FC0000000</td>
</tr>
<tr>
<td></td>
<td>0xFFFFFFFF</td>
<td>0xFFFFFFFF</td>
<td>0xFFFFFFFF00000000</td>
<td>0x0000006FC00000000</td>
</tr>
</tbody>
</table>
Memory Page Table

Every memory address is translated by CPU.

Every process has its own memory page table.

![Diagram of Memory Page Table](image)
Translation

Diagram showing the translation process in a computer system, including components like KPROCESS, CR3, Virtual address, Page directory index, Page table index, Byte offset, CR3, Physical address, Index, PDE, PFN, PTE, Desired page, Desired byte, and Page directory, Page tables, and Physical address space.
DLLs
DLLs

Cornerstone of Microsoft Windows

All functions in the API are contained in DLLs

Three most important:

- Kernel32.dll – managing memory, processes, and threads
- User32.dll – user-interface tasks (window creation, message sending etc.)
- GDI32.dll – drawing graphical images and displaying text

How many DLLs does notepad have?
How many threads does a notepad have?
DLLs and a Process’ Address Space

Before application can call functions in a DLL, the DLL’s file image must be mapped into the calling process’ address space.

Two methods:
- Implicit load-time linking
- Explicit run-time linking

Once an image is mapped into the address space, it is in fact no longer library:
- During call to a DLL function it looks at the thread’s stack
- Object created by code in the DLL’s functions are owned by the calling thread
- DLL’s global and static variables are created in a process’ address space
Linking

**Implicit loading**

When application’s source code reference symbols contained in the DLL

Loader implicitly loads and links the required library during startup

**Explicit loading**

Application can load library in runtime

Requires call to `LoadLibrary` or `LoadLibraryEx`

Flexible – allows to load library as a datafile or change search path
Search order

1. The directory containing the executable image file
2. The Windows system directory returned by `GetWindowsDirectory` function
3. The 16-bit system directory (System subfolder under the Windows directory)
5. The process’ current directory
6. The directories listed in the PATH environment variable

Can be changed!
Rebasing Modules

Every executable and DLL module has a preferred base address

This address identifies the ideal memory address where the module should get mapped into a process’ address space.

- Executable has address 0x00400000
- DLL has address 0x10000000

Why is this so important?
Rebasing Modules

DLL can have a relocation section
- It contains a list of byte offsets
- Each byte offset identifies a memory address used by a machine code instruction

When a DLL cannot be loaded at its preferred address loader can modify relocation section and adjust offsets

We can do it using Rebase + Bind utilities
Address Space Layout Randomization

Security technique involved in protection from buffer overflow attacks

ASLR randomly arranges the address space positions of key data areas of a process:
- Position of stack
- Position of heap
- Positions of libraries
- Base of the executable
Entry-Point function

DLL can have a single entry-point function

The system calls this function at various times

These calls are informational – DLL is notified when it’s attached to process or thread

```c
BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved) {
    switch (fdwReason) {
    case DLL_THREAD_DETACH:
        EnterCriticalSection(&g_csGlobal);
    }
}
```
Loader Lock

Windows holds a loader lock during DLL initialization

This is required to block other threads from calling DLL’s functions before the library is initialized

This often causes deadlock

```c
BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved) {
    switch (fdwReason) {
    case DLL_THREAD_DETACH:
        EnterCriticalSection(&g_csGlobal);
        break;
    }
}
```
Demos

Most popular techniques:
- Registry
- Windows Hooks
- Remote Threads
Loading DLL on demand
Using the Registry
Using the Registry

1 – process starts
2 – process loads user32.dll
3 – user32.dll loads our dll
Using Windows Hooks
Using Windows Hooks

1 – process starts
2 – we press some key
3 – windows loads our dll and executes hook function
Using Remote Threads
Using Remote Threads

1 – target starts
2 – our process starts
3 – our process allocates memory in target
4 – our process writes memory in target
5 – our process creates thread in target
6 – thread loads our dll

Target

C:\...

Our process

Our DLL (on disk)
Injecting Managed DLL
Injecting Managed DLL

1 – target starts
2 – our process starts
3 – our process allocates memory in target
4 – our process writes memory in target
5 – our process creates thread in target
6 – thread loads our dll
7 – our process creates another thread to run function inside native dll
8 – our function loads and starts .NET
Other methods

◦ Trojan library
  ◦ Just replace the library on the drive with custom one having the same methods
◦ Injecting using debugger
  ◦ Attach debugger and explicitly load the library
◦ Injecting into child
  ◦ When starting a process inject the library
◦ Injecting using Asynchronous Procedure Call (APC)
  ◦ Send some code to load the library
◦ LD_PRELOAD
  ◦ Linux equivalent of registry injection on Windows
◦ DOTNET_STARTUP_HOOKS environment variable
  ◦ For .NET Core
◦ ptrace
  ◦ Can be used to implement CreateRemoteThread equivalent in Linux
◦ Replacing classes in jars
  ◦ To inject code into java process
Q&A
References

Jeffrey Richter, Christophe Nasarre - „Windows via C/C++”
Mark Russinovich, David A. Solomon, Alex Ionescu - „Windows Internals”
Penny Orwick – „Developing drivers with the Microsoft Windows Driver Foundation”
Jeffrey Richter - „CLR via C#”
Mario Hewardt, Daniel Pravat - „Advanced Windows Debugging”
Mario Hewardt - „Advanced .NET Debugging”
Adam Furmanek – „.NET Internals Cookbook”
Bonus
Thanks!

CONTACT@ADAMFURMANEK.PL
HTTP://BLOG.ADAMFURMANEK.PL
@FURMANEKADAM