



# DLL Injection

---

CONTACT@ADAMFURMANEK.PL

[HTTP://BLOG.ADAMFURMANEK.PL](http://blog.adamfurmanek.pl)

[FURMANEKADAM](https://twitter.com/furmanekadam)

# About me

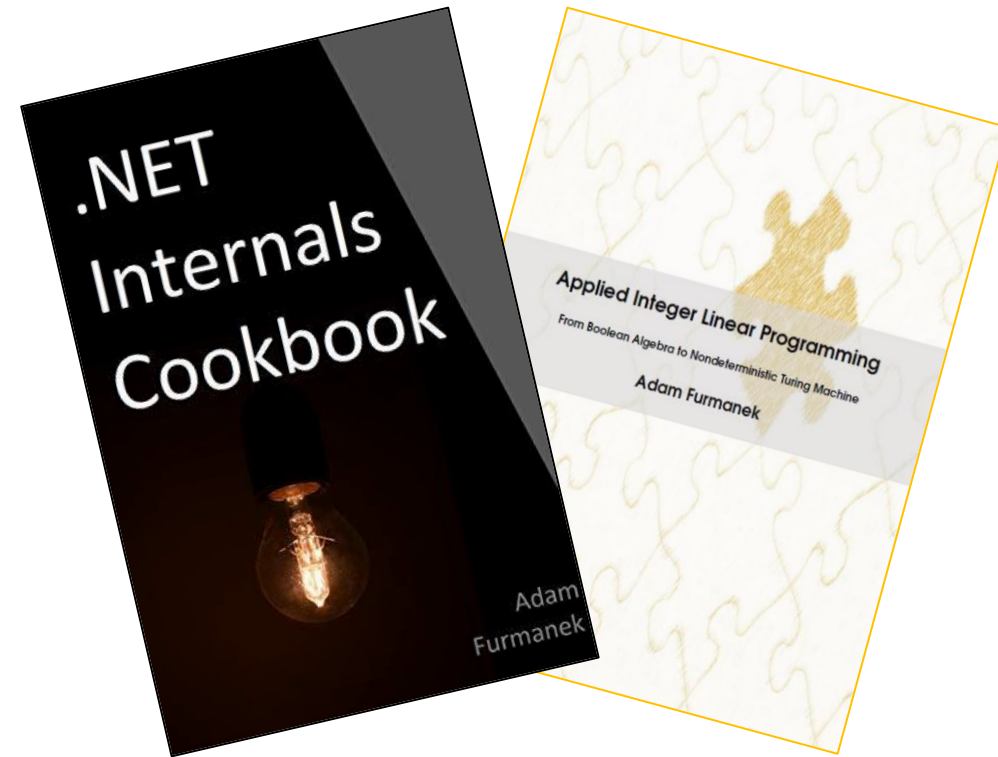
---

Software Engineer, Blogger, Book Writer, Public Speaker.  
Author of *Applied Integer Linear Programming* and *.NET Internals Cookbook*.

<http://blog.adamfurmanek.pl>

[contact@adamfurmanek.pl](mailto:contact@adamfurmanek.pl)

[✈ furmanekadam](https://twitter.com/furmanekadam)



Random IT Utensils

IT, operating systems, maths, and more.

# Agenda

---

What and why

Preliminaries

How + Demos

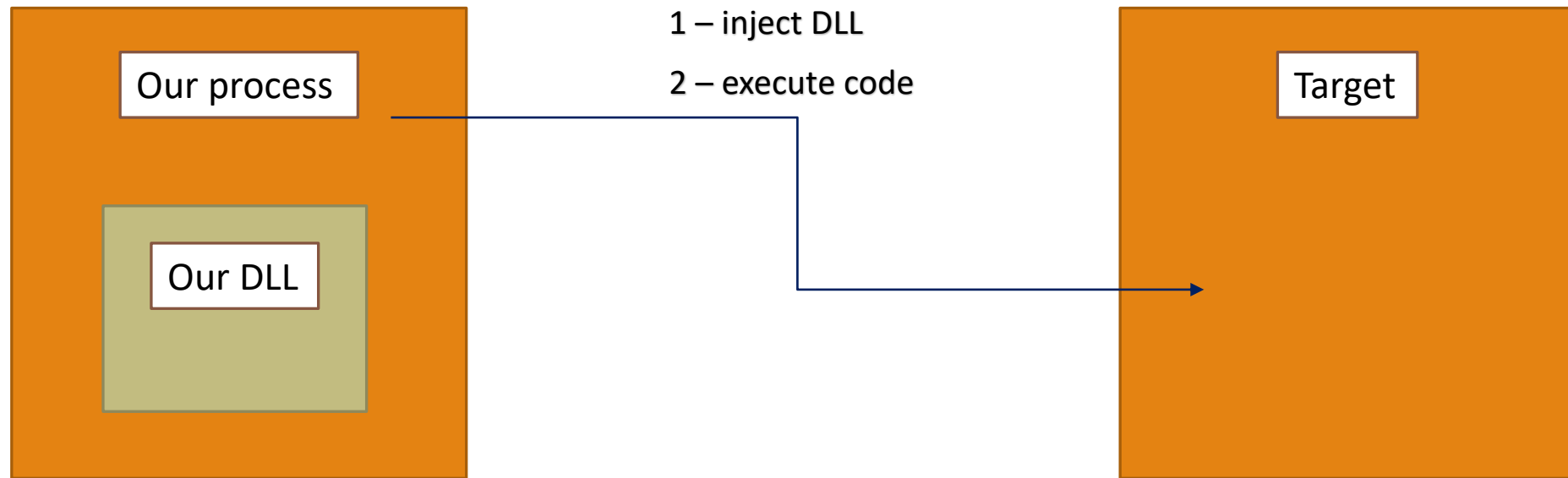
Summary

# What and why

---

# What we are going to do

---



# 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

---

REAL LIFE USAGES

# Preliminaries

---



# Virtual Address Space

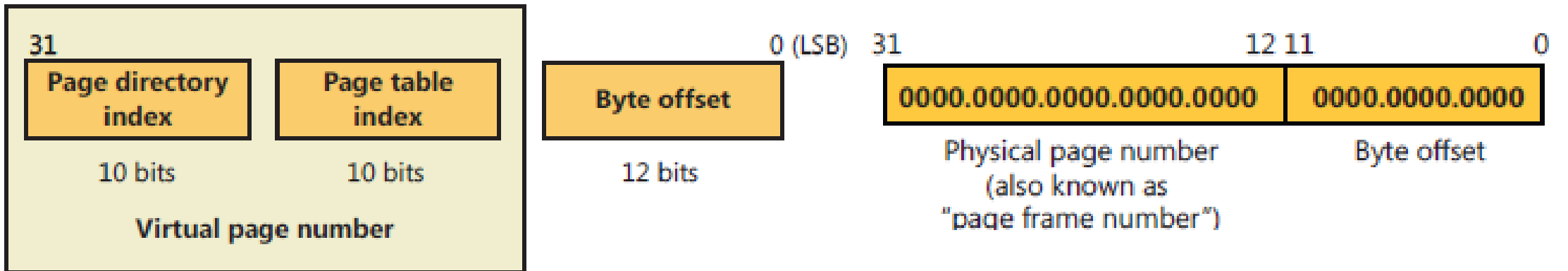
Every proces has its own address space.

Partition	x86 32-Bit Windows	x86 32-Bit Windows with 3 GB User-Mode	x64 64-Bit Windows	IA-64 64-Bit Windows
NULL-Pointer Assignment	0x00000000	0x00000000	0x00000000'00000000	0x00000000'00000000
	0x0000FFFF	0x0000FFFF	0x00000000'0000FFFF	0x00000000'0000FFFF
User-Mode	0x00010000	0x00010000	0x00000000'00010000	0x00000000'00010000
	0x7FFEFFFF	0xBFFEFFFF	0x000007FF'FFFEFFFF	0x000006FB'FFFEFFFF
64-KB Off-Limits	0x7FFF0000	0xBFFF0000	0x000007FF'FFFF0000	0x000006FB'FFFF0000
	0x7FFFFFFF	0xBFFFFFFF	0x000007FF'FFFFFFFF	0x000006FB'FFFFFFFF
Kernel-Mode	0x80000000	0xC0000000	0x00000800'00000000	0x000006FC'00000000
	0xFFFFFFFF	0xFFFFFFFF	0xFFFFFFFF'FFFFFFFF	

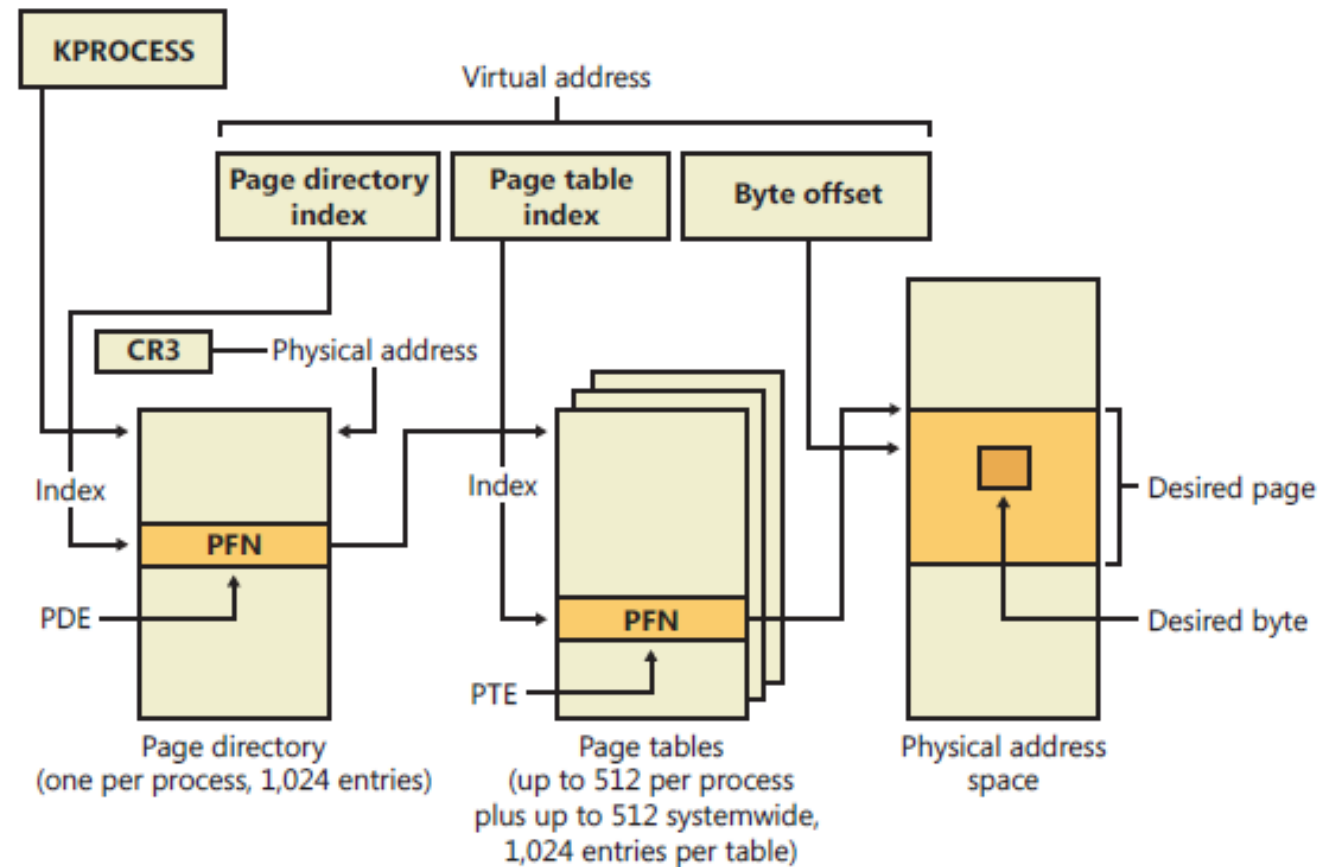
# Memory Page Table

Every memory address is translated by CPU.

Every proces has its own memory page table.



# Translation



# How many threads does a notepad have?

The screenshot displays two windows from a Windows operating system. The primary window is Process Explorer, showing the details for the 'notepad.exe:18940' process. A sub-window titled 'notepad.exe:18940 Properties' is open, showing the 'Threads' tab with a list of 22 threads. The first thread, with TID 7784, is highlighted. Below the thread list, detailed information for this thread is shown, including its start time, state, and cycles. A second window, 'Untitled - Notepad', is open in the background, showing a file explorer view of the 'Documents' folder. The taskbar at the bottom indicates system performance metrics: CPU Usage 40.41%, Commit Charge 67.99%, Processes: 257, and Physical Usage 52.25%.

TID	CPU	Cycles Delta	Start Address
7784	0.09	21,320,180	notepad.exe+0x19450
8488	< 0.01	525,289	ntdll.dll!TpAllocWork+0x1a0
10012			ntdll.dll!TpAllocWork+0x1a0
21236			ntdll.dll!TpAllocWork+0x1a0
19804			combase.dll!WindowsDeleteStringBuffer+0x1950
14288			msvcrt.dll!endhreadex+0x30
9656			shcore.dll!SHReleaseThreadPlet+0x400
18536			combase.dll!WindowsDeleteStringBuffer+0x1950
21452			PROPSYS.dll!PSFormatForDisplayAlloc+0x2c0
10036			ntdll.dll!TpAllocWork+0x1a0
10428			ntdll.dll!TpAllocWork+0x1a0
18652			ntdll.dll!TpAllocWork+0x1a0
15724			ntdll.dll!TpAllocWork+0x1a0
18908			clr.dll!SetRuntimeInfo+0x4540
9684			clr.dll!InstallCustomModule+0x1240
16404			ntdll.dll!TpAllocWork+0x1a0
10780			ntdll.dll!TpAllocWork+0x1a0
1020			ntdll.dll!TpAllocWork+0x1a0
21140			clr.dll!InstallCustomModule+0x1240
352			clr.dll!InstallCustomModule+0x1240
17620			clr.dll!InstallCustomModule+0x1240
7728			clr.dll!InstallCustomModule+0x1240

Name	Description	Company Name	Path
Amazon\WorkDocs\Drive\WinClient\Common.dll	WinClientCommon	Amazon Web Services, Inc	C:\Program Files\Amazon\AWS\WorkDocs\DriveClient\Amazon\WorkDocs\...
AWSSDK.Core.dll	AWSSDK.Core	Amazon.com, Inc	C:\Program Files\Amazon\AWS\WorkDocs\DriveClient\AWSSDK.Core.dll
AWSWorkDocs\Drive\Common.dll	Amazon WorkDocs Drive Common	Amazon Web Services, Inc	C:\Program Files\Amazon\AWS\WorkDocs\DriveClient\AWSWorkDocs\...
AWSWorkDocs\Drive\Shell.dll	AWSWorkDocs Drive Shell	Amazon Web Services, Inc	C:\Program Files\Amazon\AWS\WorkDocs\DriveClient\AWSWorkDocs\...
Newtonsoft.Json.dll	Json.NET	Newtonsoft	C:\Program Files\Amazon\AWS\WorkDocs\DriveClient\Newtonsoft.Json.dll
SharpShell.dll	SharpShell	Dave Keer	C:\Program Files\Amazon\AWS\WorkDocs\DriveClient\SharpShell.dll
tpst.dll	Touch Keyboard and Handwriting ...	Microsoft Corporation	C:\Program Files\Common Files\microsoft shared\ink\tpst.dll
OFFICE.ODF	Microsoft Office culture data dll	Microsoft Corporation	C:\Program Files\Common Files\microsoft shared\OFFICE16\Cultures\O...
DpFeedb.dll	DigitalPersona OTS Feedback	DigitalPersona, Inc.	C:\Program Files\Hewlett-Packard\HP Protect Tools Security Manager\...

# 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?

# 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)
4. The Windows directory returned by *GetSystemDirectory*
5. The process' current directory
6. The directories listed in the PATH environment variable

Can be changed!





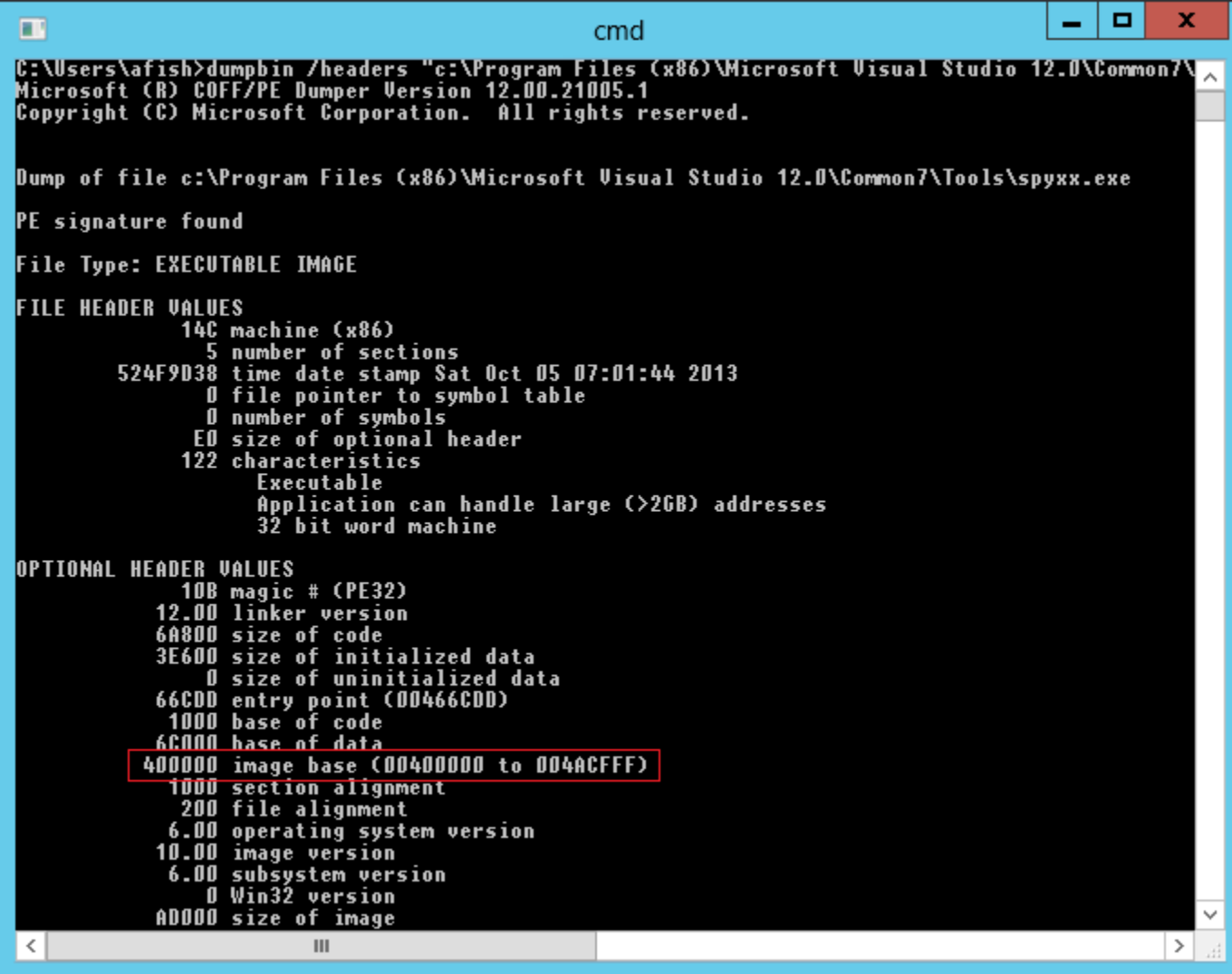
# Rebasing Modules

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?



```
cmd
C:\Users\afish>dumpbin /headers "c:\Program Files (x86)\Microsoft Visual Studio 12.0\Common7\
Microsoft (R) COFF/PE Dumper Version 12.00.21005.1
Copyright (C) Microsoft Corporation. All rights reserved.

Dump of file c:\Program Files (x86)\Microsoft Visual Studio 12.0\Common7\Tools\spyxx.exe
PE signature found
File Type: EXECUTABLE IMAGE
FILE HEADER VALUES
    14C machine (x86)
     5 number of sections
524F9D38 time date stamp Sat Oct 05 07:01:44 2013
     0 file pointer to symbol table
     0 number of symbols
     E0 size of optional header
    122 characteristics
        Executable
        Application can handle large (>2GB) addresses
        32 bit word machine
OPTIONAL HEADER VALUES
    10B magic # (PE32)
    12.00 linker version
    6A800 size of code
    3E600 size of initialized data
     0 size of uninitialized data
    66CDD entry point (00466CDD)
    1000 base of code
    6C000 base of data
    400000 image base (00400000 to 004ACFFF)
    1000 section alignment
     200 file alignment
     6.00 operating system version
    10.00 image version
     6.00 subsystem version
     0 Win32 version
    AD000 size of image
```

# 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

```
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

```
BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved) {  
    switch (fdwReason) {  
        case DLL_THREAD_DETACH:  
            EnterCriticalSection(&g_csGlobal); ← BAD IDEA!  
        }  
    }  
}
```

# Demos

---

REGISTRY, HOOKS, REMOTE THREADS

# Loading DLL on demand

---

# Using the Registry

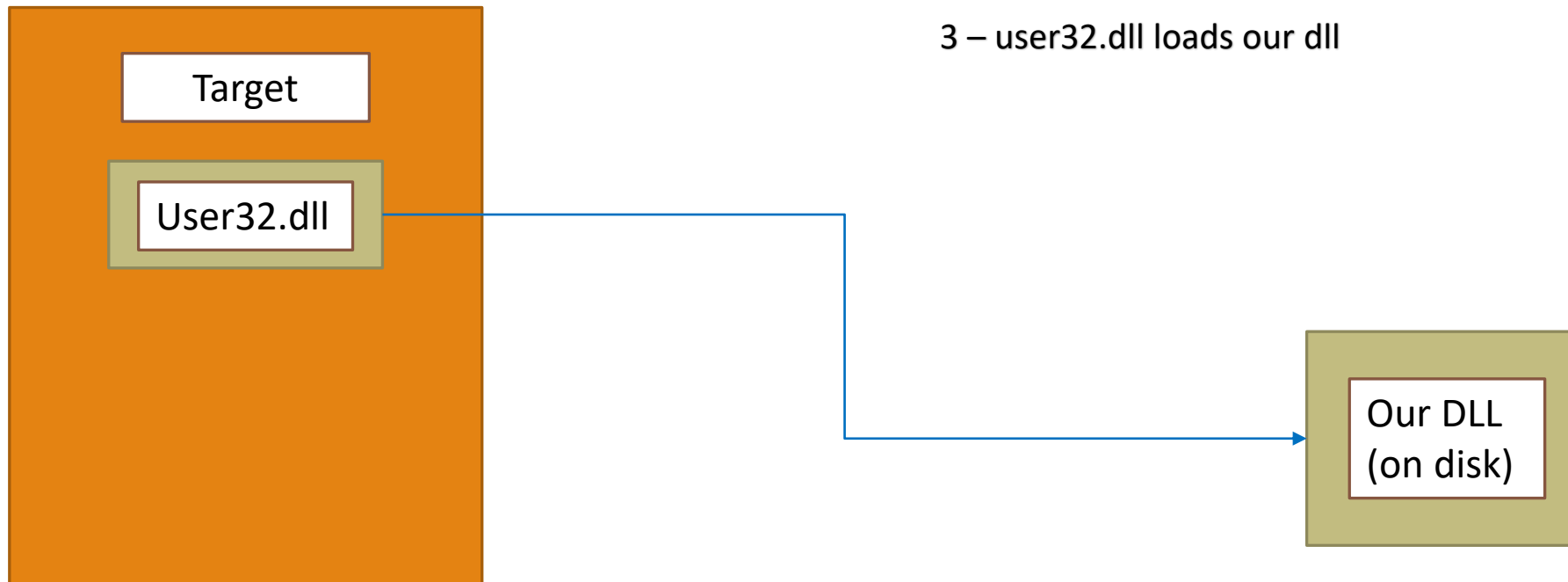
---



# Using the Registry

---

- 1 – process starts
- 2 – process loads user32.dll
- 3 – user32.dll loads our dll

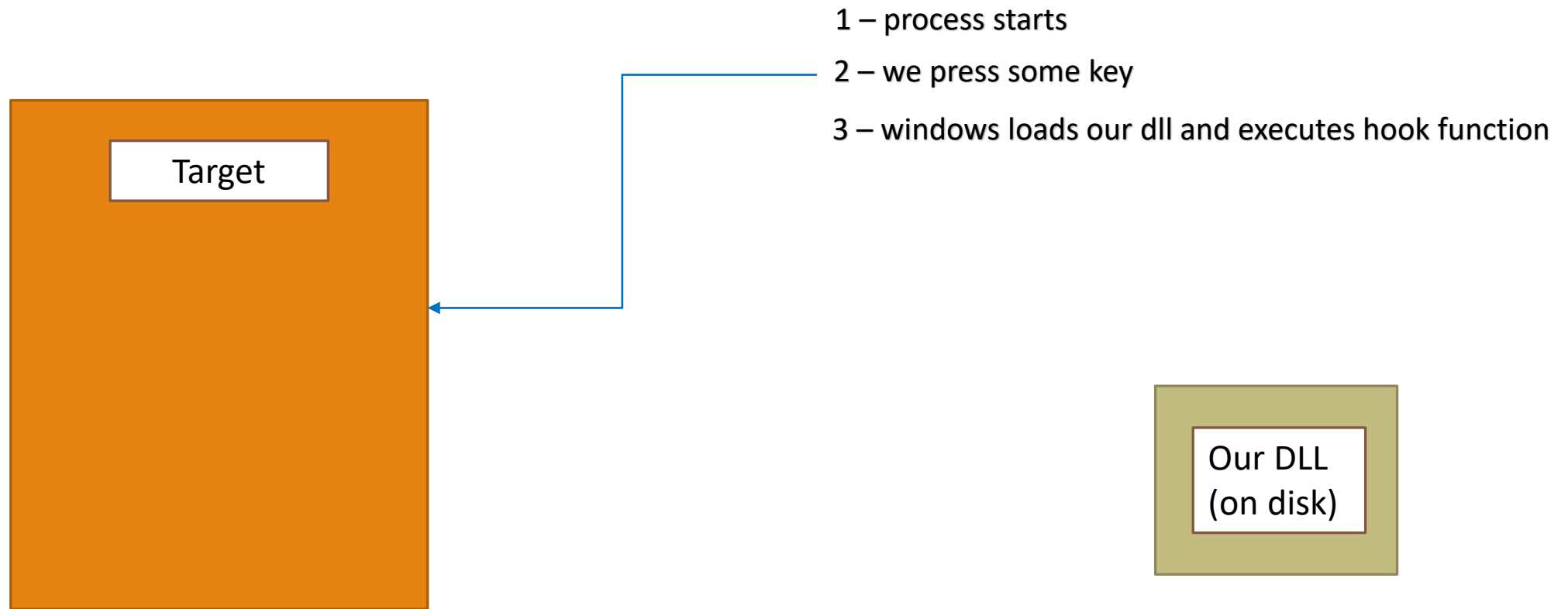


# Using Windows Hooks

---

# Using Windows Hooks

---

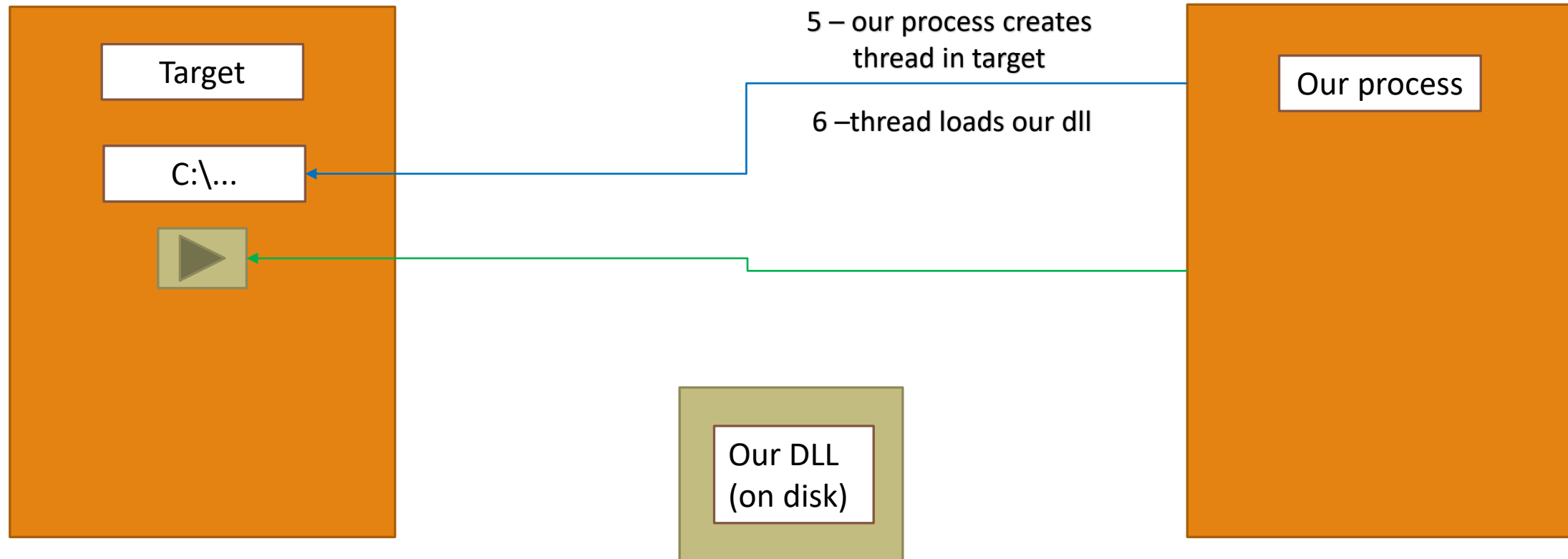


# Using Remote Threads

---

# 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

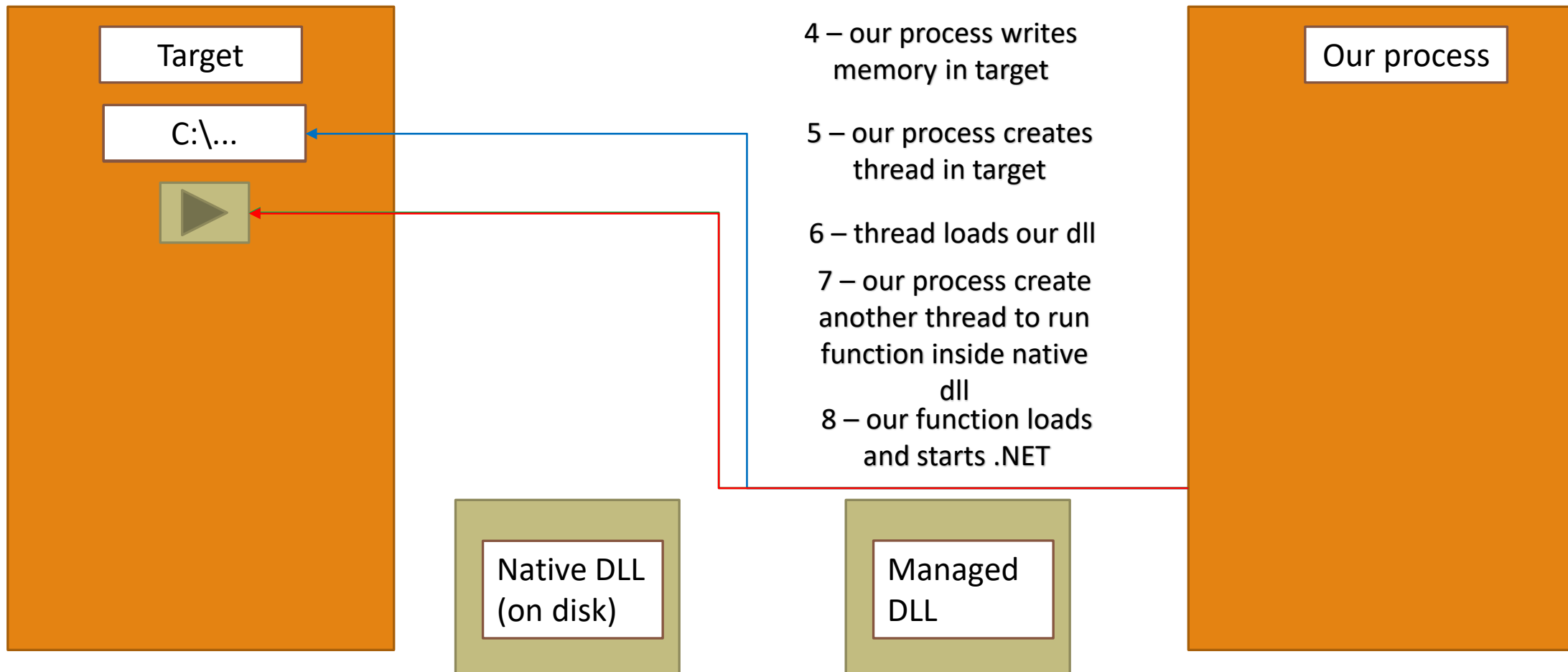


# Injecting Managed DLL

---

# 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 create 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 - „CLR via C#”

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”

Mario Hewardt, Daniel Pravat - „Advanced Windows Debugging”

Mario Hewardt - „Advanced .NET Debugging”

Steven Pratschner - „Customizing the Microsoft .NET Framework Common Language Runtime”

Serge Lidin - „Expert .NET 2.0 IL Assembler”

Joel Pobar, Ted Neward — „Shared Source CLI 2.0 Internals”

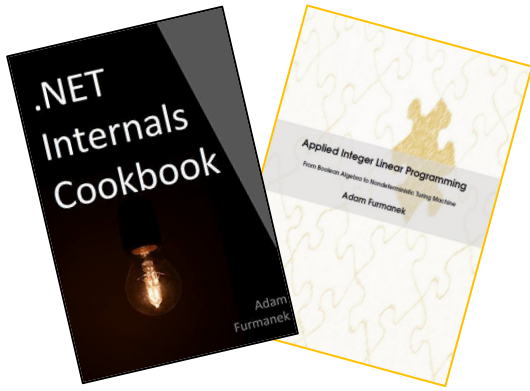
Adam Furmanek – „.NET Internals Cookbook”

<https://github.com/dotnet/coreclr/blob/master/Documentation/botr/README.md> — „Book of the Runtime”

<https://blogs.msdn.microsoft.com/oldnewthing/> — Raymond Chen „The Old New Thing”

# Bonus

---



## Random IT Utensils

IT, operating systems, maths, and more.

# Thanks!

---

[CONTACT@ADAMFURMANEK.PL](mailto:CONTACT@ADAMFURMANEK.PL)

[HTTP://BLOG.ADAMFURMANEK.PL](http://BLOG.ADAMFURMANEK.PL)

[FURMANEKADAM](https://twitter.com/FURMANEKADAM)

