RUHR-UNIVERSITÄT BOCHUM

RUB

Using Memory Management to Detect and Extract Illegitimate Code for Malware Analysis

ACSAC 28 | December 3-7, 2012

***Carsten Willems*[1], Felix C. Freiling[2], Thorsten Holz[1]**

[1]**Horst Görtz Institute for IT-Security,** Chair for Systems Security
[2]**Friedrich-Alexander-Universität Erlangen-Nürnberg,** Department Informatik

hgi
Horst Görtz Institut
für IT-Sicherheit

```
[21.9.2012 12:11:24] [  19]  from  0x77c22667  msvcrt.type_info::name+0x97
[21.9.2012 12:11:24]                ROP-RET  ####################
[21.9.2012 12:11:24] [  19]    to  0x77c3ed6e  msvcrt._flsbuf+0x111
[21.9.2012 12:11:24] [  18]  from  0x77c3ed77  msvcrt._flsbuf+0x11a
[21.9.2012 12:11:24]                ROP-RET  ####################
[21.9.2012 12:11:24] [  18]    to  0x77c244c6  msvcrt.UnDecorator::getVCallThunkType+0x37
[21.9.2012 12:11:24] [  17]  from  0x80541fc7  ntkrnlpa.Kei386EoiHelper+0xab
[21.9.2012 12:11:24] [  17]    to  0x77c244c6  msvcrt.UnDecorator::getVCallThunkType+0x37
[21.9.2012 12:11:24] [  16]  from  0x77c244c7  msvcrt.UnDecorator::getVCallThunkType+0x38
[21.9.2012 12:11:24]                     RET  --------------------
[21.9.2012 12:11:24] [  16]    to  0x77c244c3  msvcrt.UnDecorator::getVCallThunkType+0x34
[21.9.2012 12:11:24] [  15]  from  0x77c244c7  msvcrt.UnDecorator::getVCallThunkType+0x38
```

Motivation

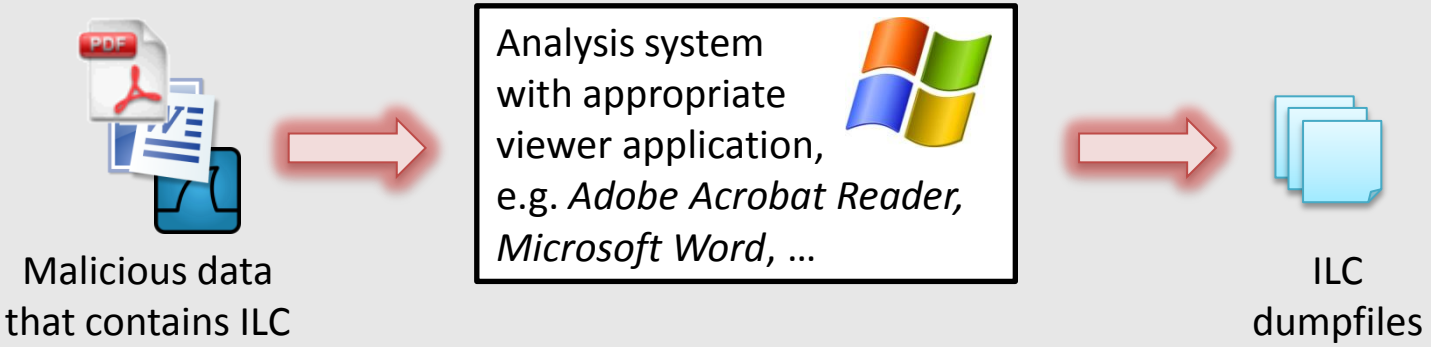
- Attackers use **illegitimate code** (ILC) when exploiting systems
  - e.g. shellcode in network packets, malicious documents, ..
- NX+ASLR is a hurdle, but not a barrier
  - implementation flaws, information leakage, unrandomized modules, legacy systems, …
- Insight into shellcode helps to protect systems
- Amount of malware demands automation

Overview of the Talk

1. Motivation
2. General Approach
3. Prototype Implementation
4. Evaluation
5. Discussion

Approach
General Idea

- Build a *generic* tool that
  - hooks into a system
  - detects the execution of ILC
  - automatically dumps ILC for later analysis
  - continues operation until all ILC has been dumped
- Not meant for *protection*, but only for analysis

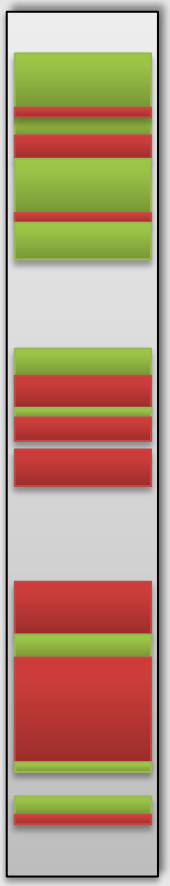


Approach
Implementation Idea

- Partition memory into regions that contain
  - legitimate code (LC)
  - and (possibly) illegitimate code (ILC)
- Instrument memory related system calls
  - force ILC memory to be always non-executable
- Instrument page fault handler
  - attempt to execute NX memory ➔ page-fault ➔ ILC detected

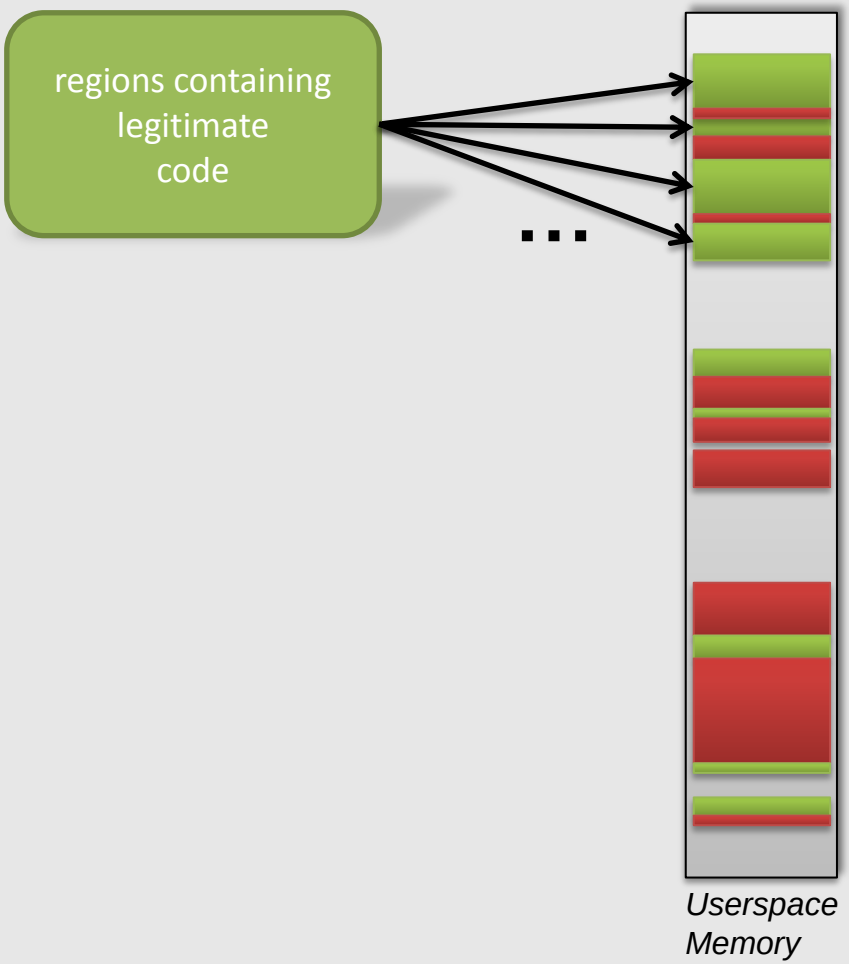
- *How to decide which code is legitimate?*

Approach
LC vs ILC memory

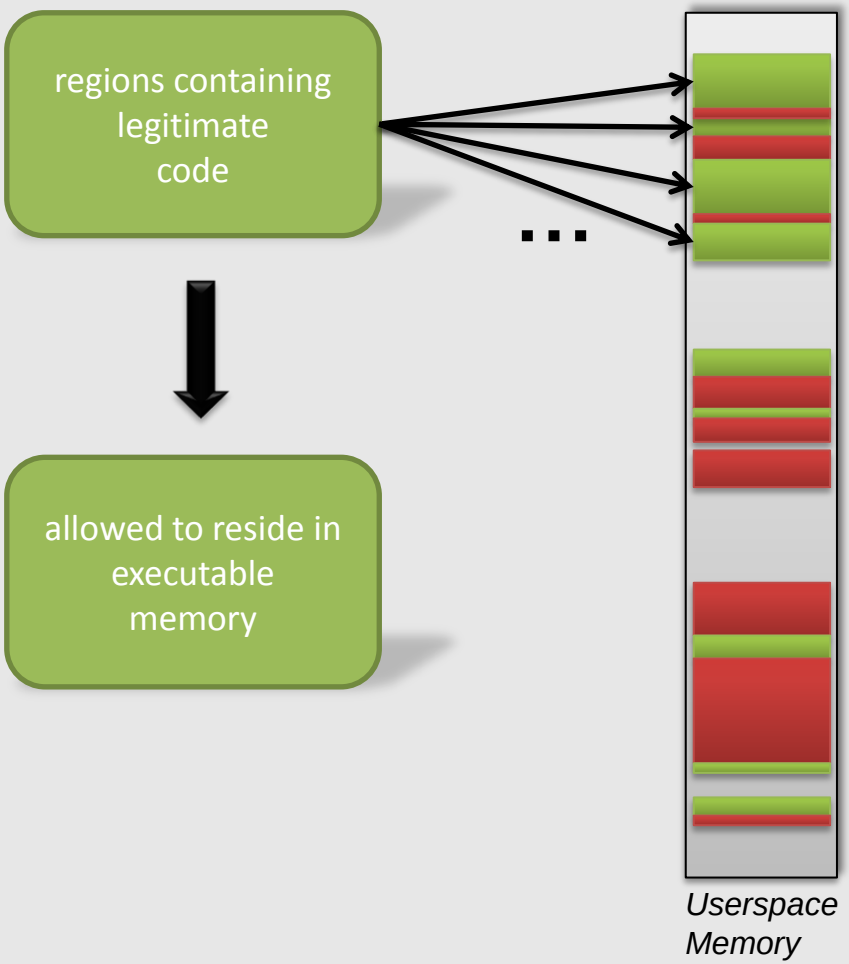


Userspace Memory

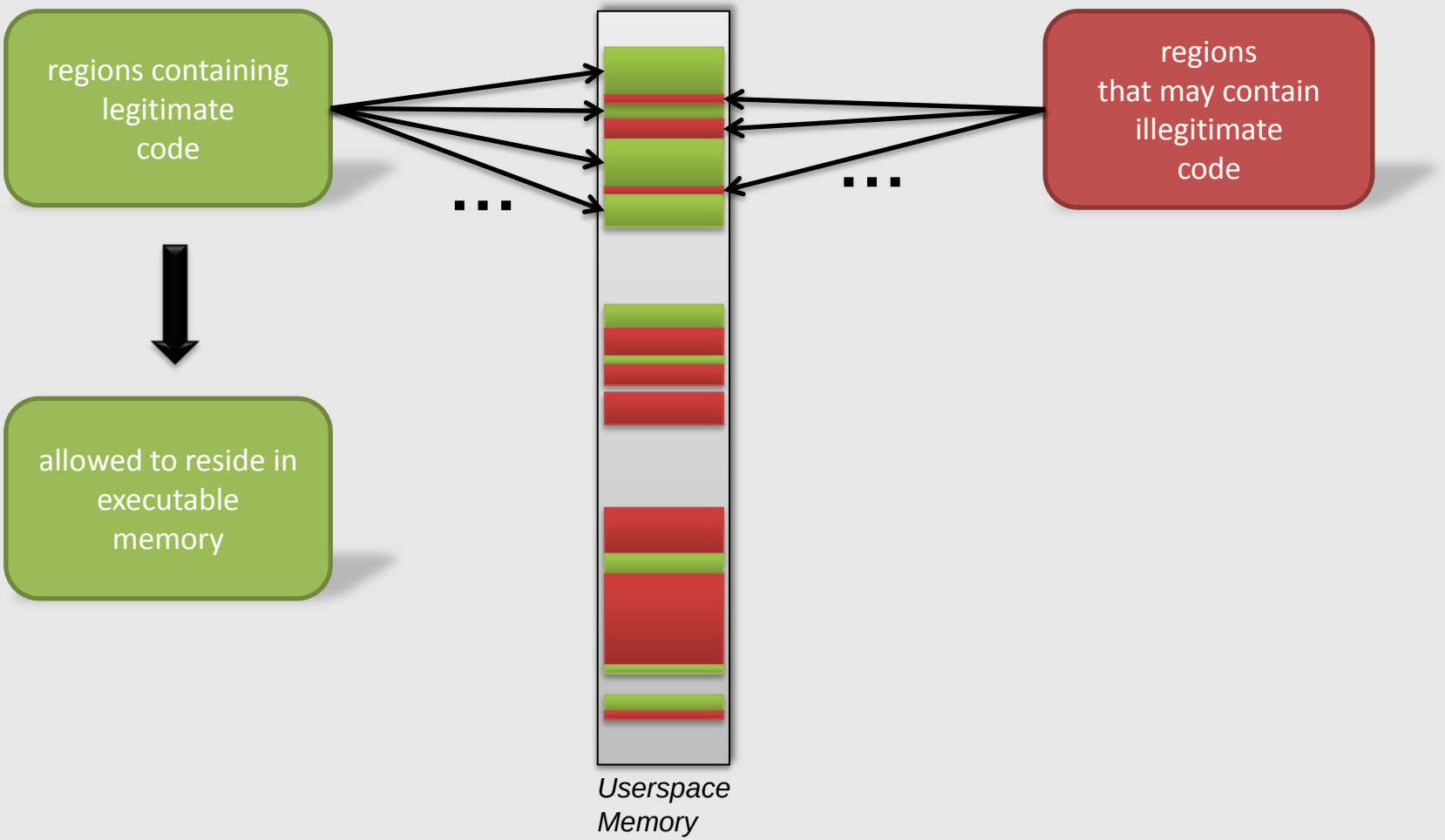
Approach
LC vs ILC memory



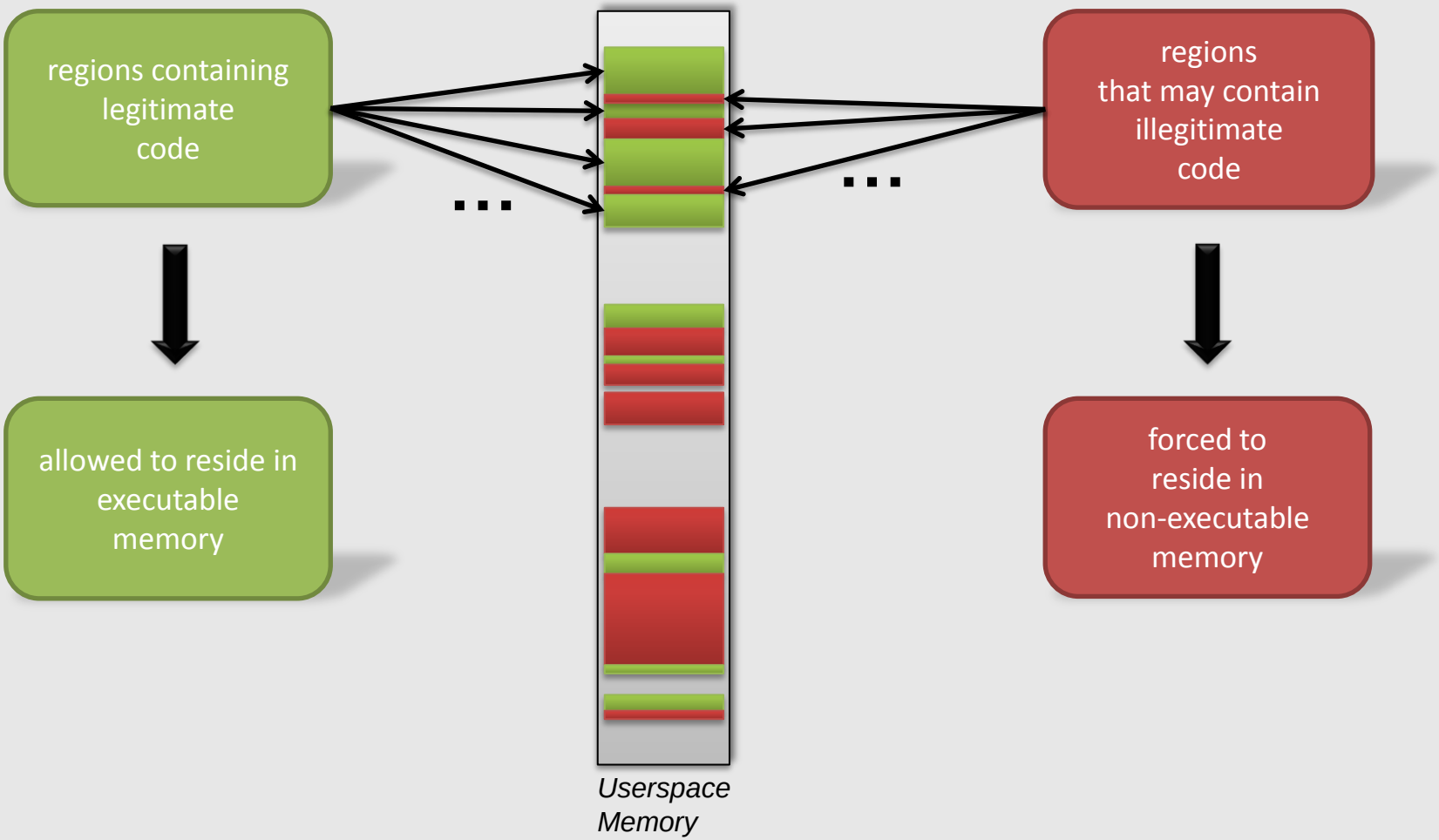
Approach
LC vs ILC memory



Approach
LC vs ILC memory

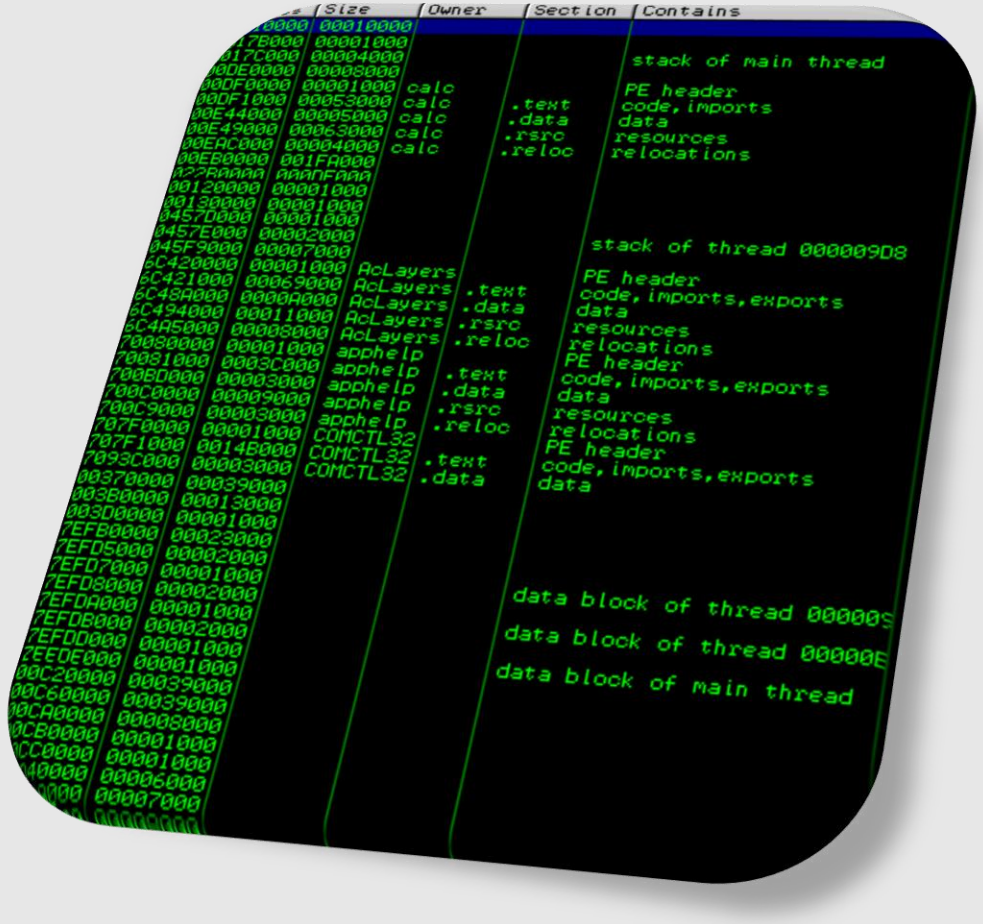


Approach
LC vs ILC memory



Approach
Memory Regions

- Memory regions are either
  - Mapped files, e.g.
    - applications
    - shared libraries
    - data files
  - or dynamically allocated, e.g.
    - heaps
    - thread stacks
    - control blocks
    - JIT code



How to decide if code is illegitimate
Memory Mapped Files

- Divide memory-mapped files into
  - Trusted files
    - belong to the OS or the analyzed benign application
    - results in LC memory
  - Untrusted files
    - unknown source
    - results in ILC memory

- Use simple heuristic: trust only files that
  - already existed before the analysis
  - **and** have not been modified since then

How to decide if code is illegitimate

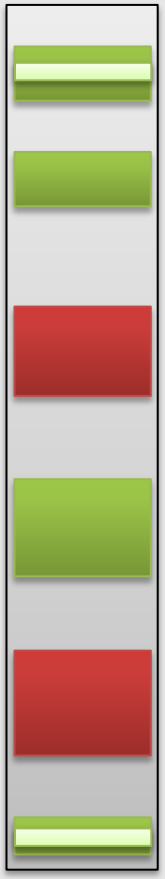
Dynamically Allocated Memory

- Is dynamically allocated memory LC or ILC?
  - initial approach:
    only memory allocated by trusted files is LC

- But: programmers make mistakes
  - only very few functions from all trusted files really need privileges to create executable memory
    - e.g. loader functions or JIT compiler
  - identify those functions and name them *trusted callers*
  - better approach:
    only memory allocated by a trusted caller is LC

How to decide if code is illegitimate

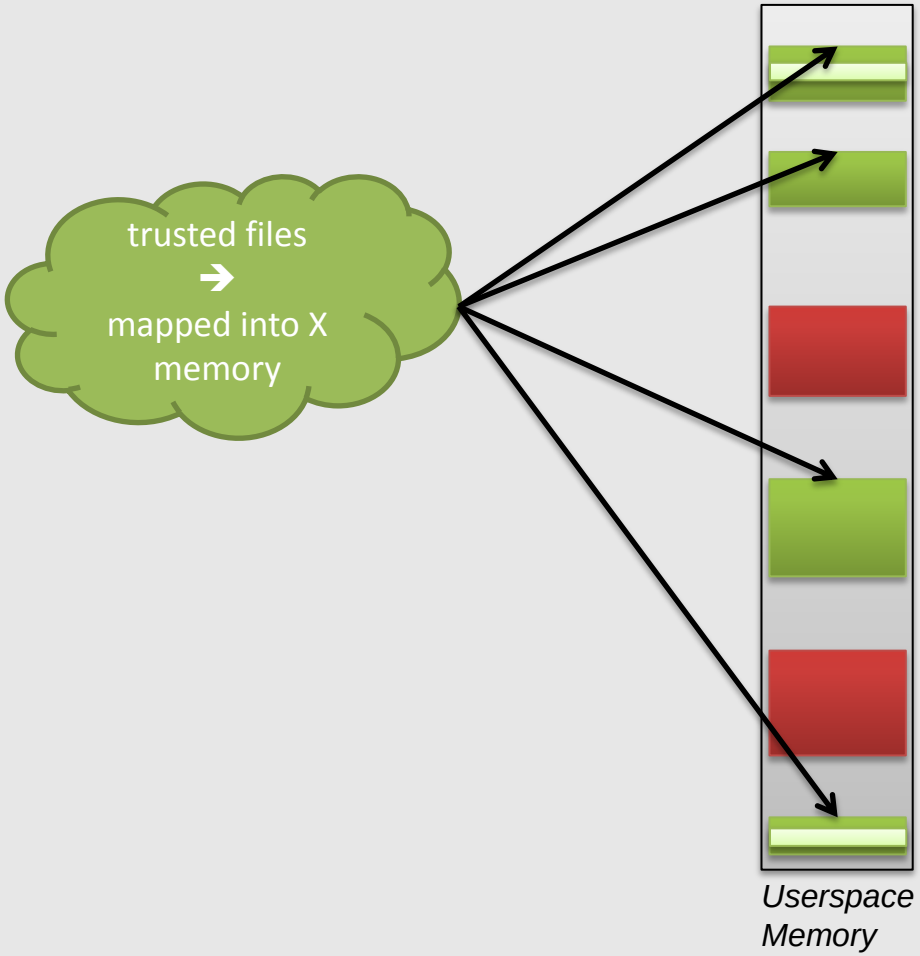
Dynamically Allocated Memory Example



Userspace Memory

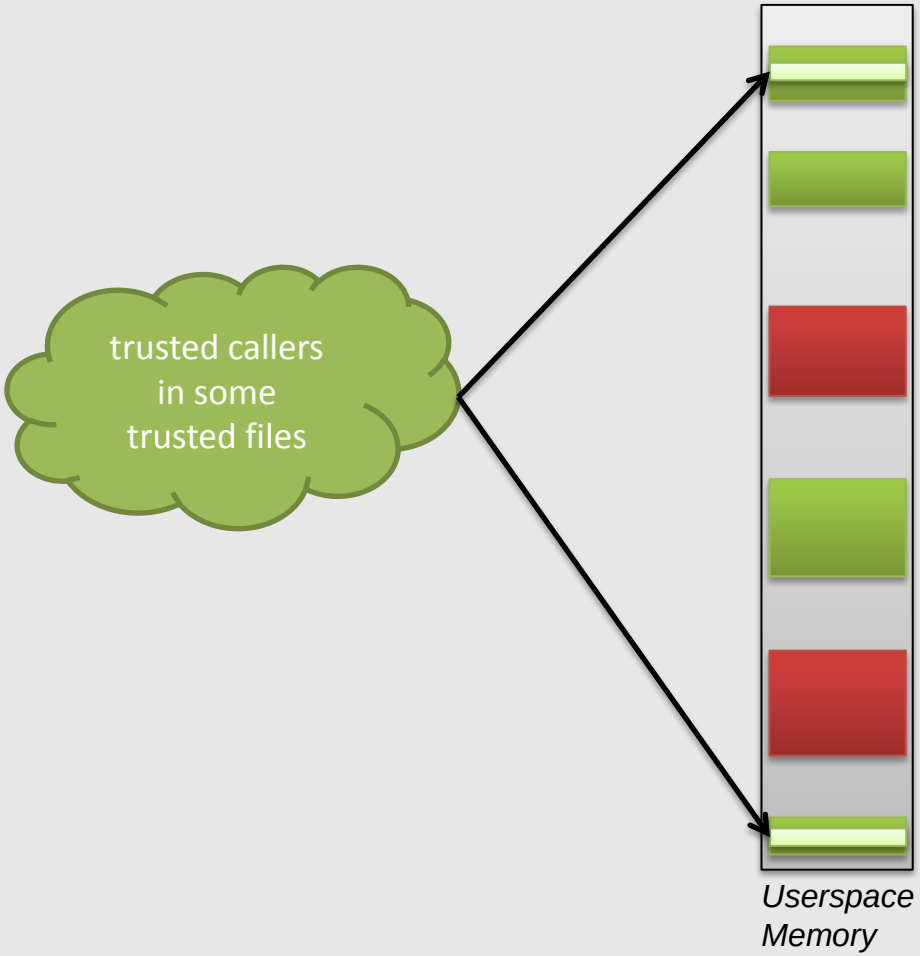
How to decide if code is illegitimate

Dynamically Allocated Memory Example

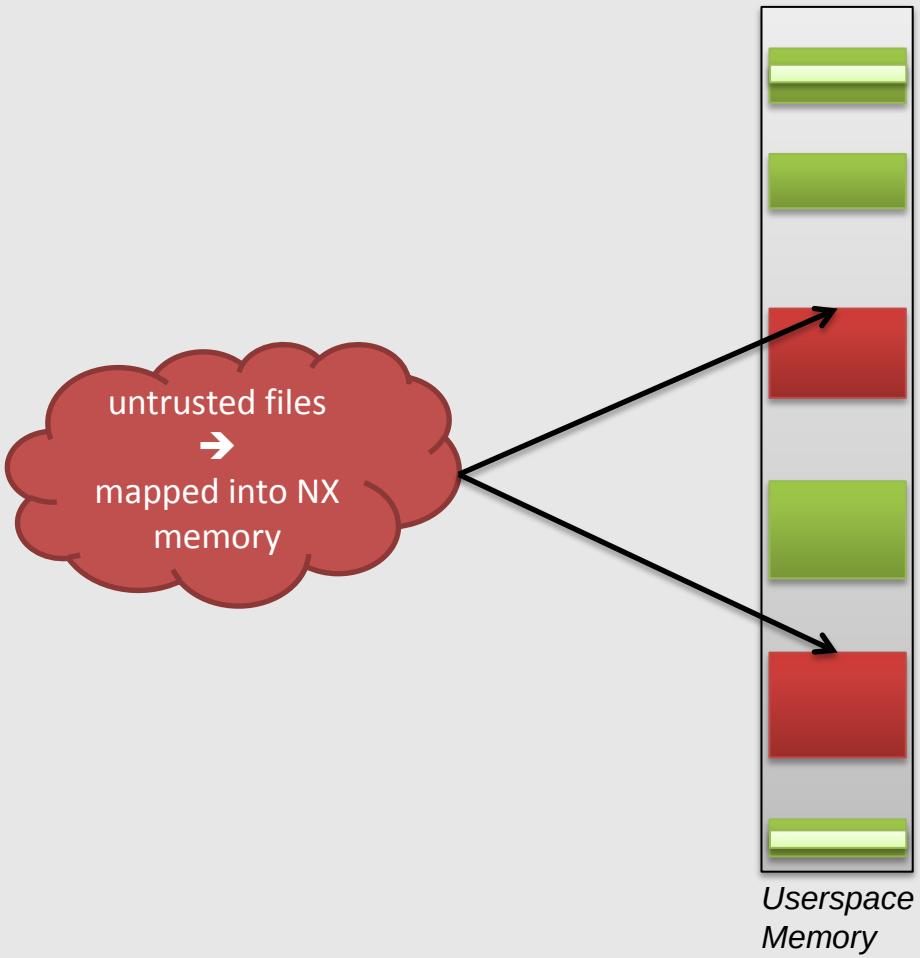


How to decide if code is illegitimate

Dynamically Allocated Memory Example

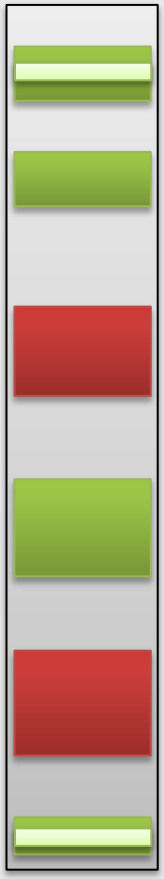


How to decide if code is illegitimate
Dynamically Allocated Memory Example



How to decide if code is illegitimate

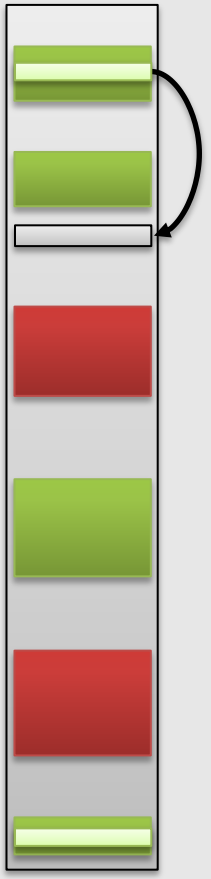
Dynamically Allocated Memory Example



Userspace Memory

How to decide if code is illegitimate

Dynamically Allocated Memory Example



trusted caller
tries to allocate
X memory

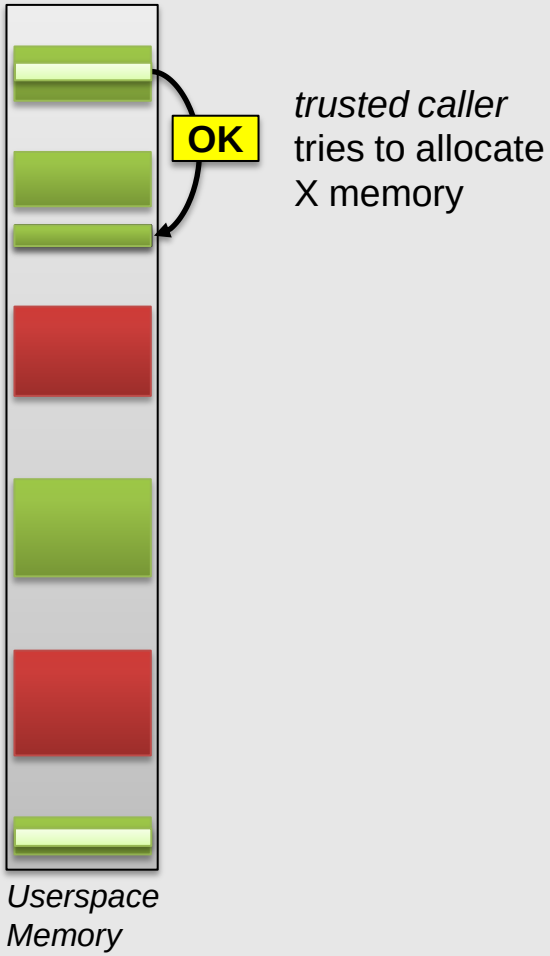
Userspace Memory

How to decide if code is illegitimate

Dynamically Allocated Memory Example

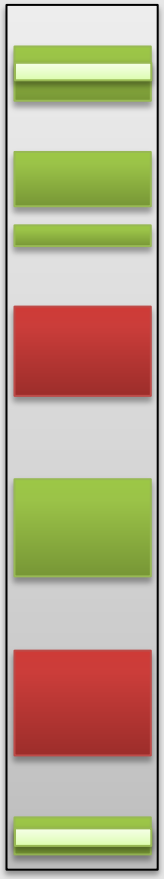
How to decide if code is illegitimate

Dynamically Allocated Memory Example



How to decide if code is illegitimate

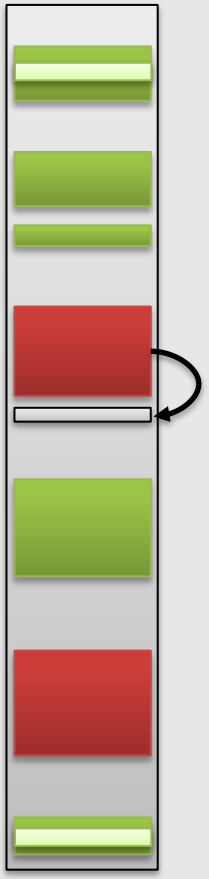
Dynamically Allocated Memory Example



Userspace Memory

How to decide if code is illegitimate

Dynamically Allocated Memory Example

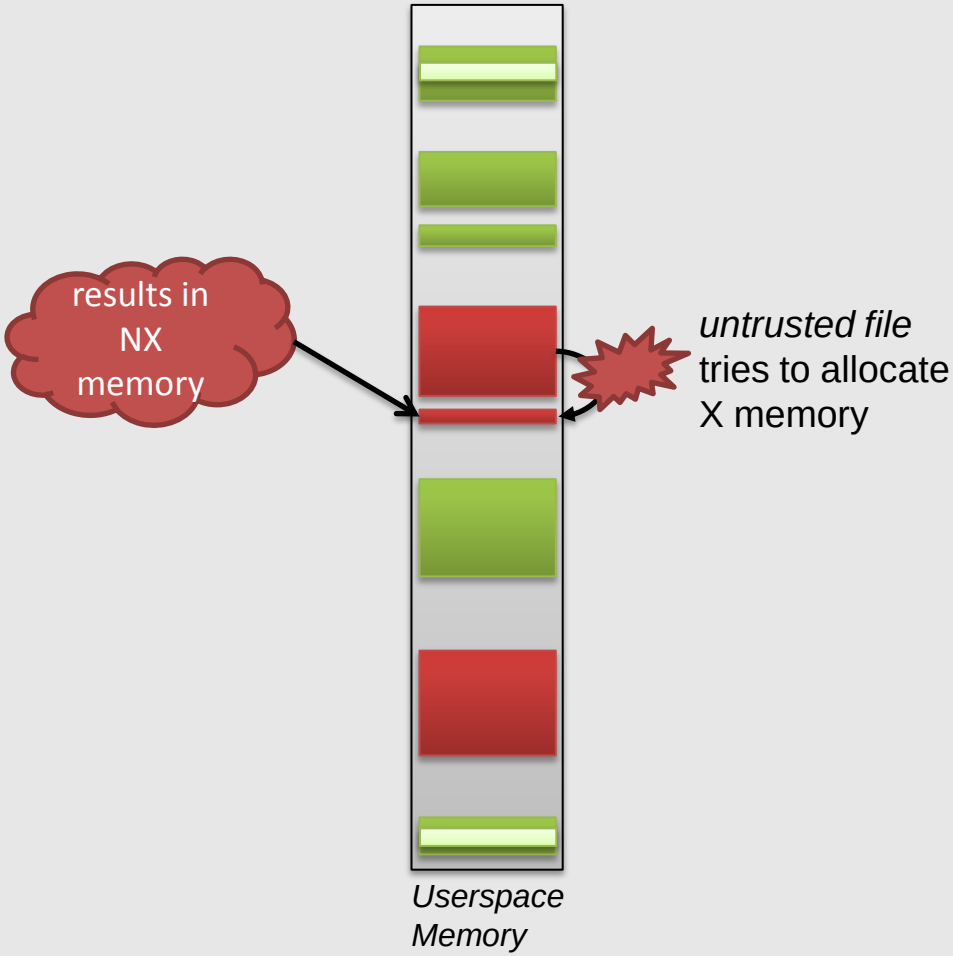


untrusted file tries to allocate X memory

Userspace Memory

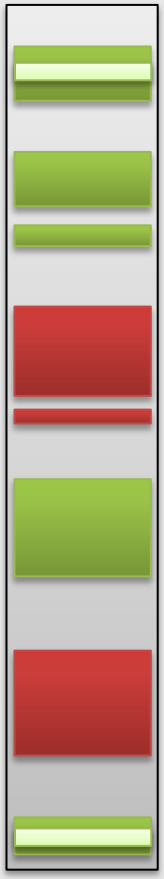
How to decide if code is illegitimate

Dynamically Allocated Memory Example



How to decide if code is illegitimate

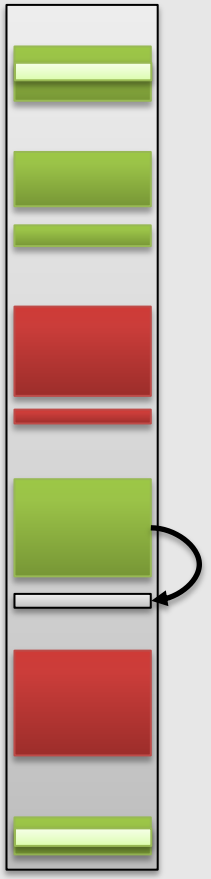
Dynamically Allocated Memory Example



Userspace Memory

How to decide if code is illegitimate

Dynamically Allocated Memory Example

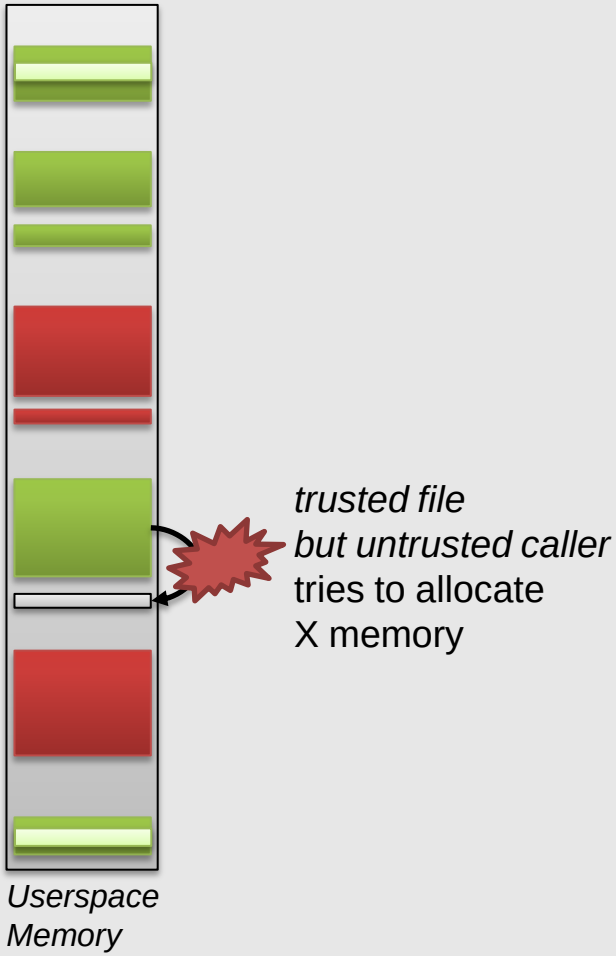


Userspace Memory

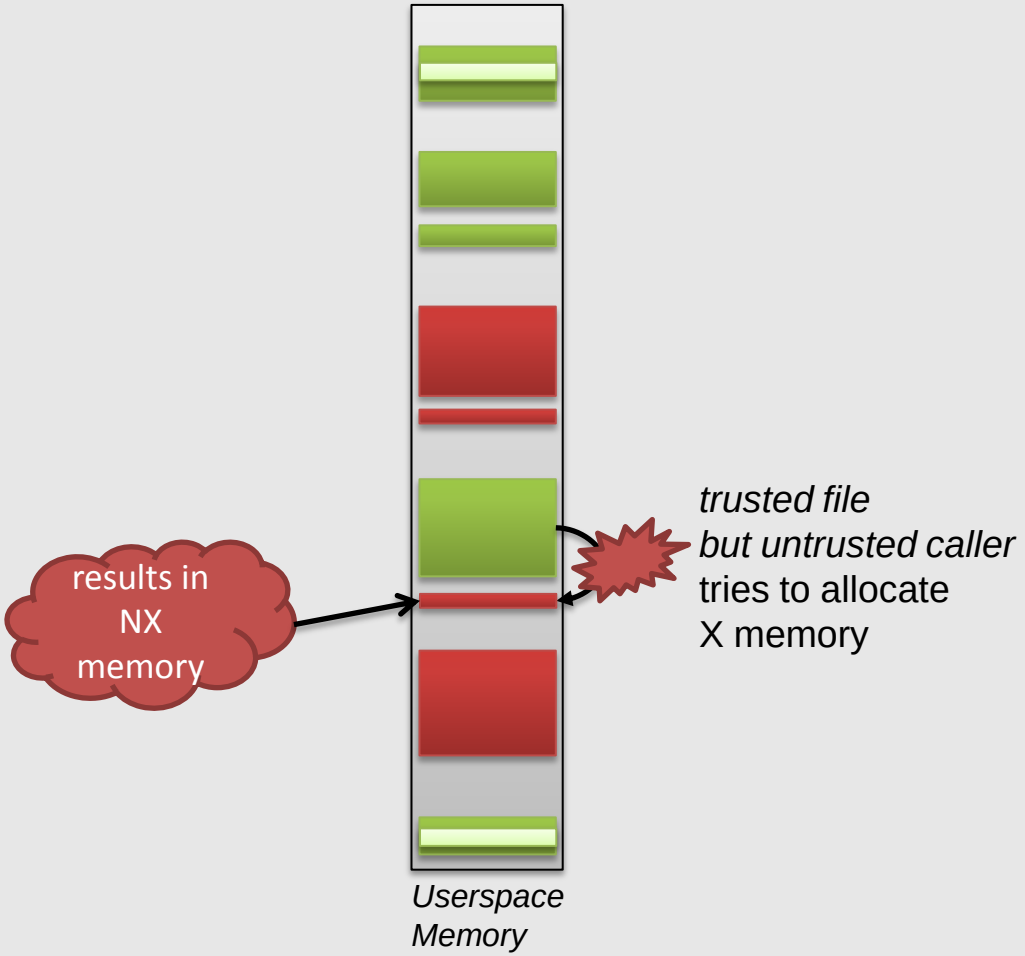
trusted file
but untrusted caller
tries to allocate
X memory

How to decide if code is illegitimate

Dynamically Allocated Memory Example



How to decide if code is illegitimate
Dynamically Allocated Memory Example



How to decide if code is illegitimate

Dynamically Allocated Memory Example

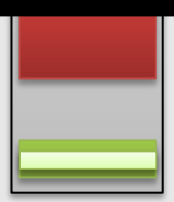
```
TARGET_APPLICATION=C:\Programme\Adobe\Reader 9.0\Reader\AcroRd32.exe
DEBUGGER_CMD=C:\Programme\Immunity Inc\Immunity Debugger\ImmunityDebugger.exe -p
DISASSEMBLE_MAX_LINES=5
SHORT_LOG=0
USE_COLORS=1
LOG_TO_CONSOLE=1
CLOSE_DIALOGS=1
MULTI_VERSION_DUMP=1

# SnapIAT+0x29c
TRUSTED_CALLER_1=ntdll.dll + 0x1C0E9

# LdrpSetProtection
TRUSTED_CALLER_2=ntdll.dll + 0x1CC27

# authplay 10.0.42.34
TRUSTED_CALLER_3=authplay.dll + 0x9f213
```

ller



Userspace Memory

How to decide if code is illegitimate

Dynamically Allocated Memory Example

```
TARGET_APPLICATION=C:\Program Files\Internet Explorer\iexplore.exe
DEBUGGER_CMD="C:\Program Files\Immunity Inc\Immunity Debugger\ImmunityDebugger.exe" –p
ALLOW_ALL_PROCESSES=1

## NtProtectVirtualMemory Callers:
TRUSTED_CALLER_1=ntdll.dll + 0x1c0e9
TRUSTED_CALLER_2=ntdll.dll + 0x1cc27
TRUSTED_CALLER_3=IEFRAME.dll + 0xa4dcd,
TRUSTED_CALLER_4=IEFRAME.dll + 0xa34e9
TRUSTED_CALLER_5=IEFRAME.dll + 0xa3594
TRUSTED_CALLER_6=RPCRT4.dll + 0x8b5bf
TRUSTED_CALLER_7=IEFRAME.dll + 0x9434c
TRUSTED_CALLER_8=IEFRAME.dll + 0x943f3
TRUSTED_CALLER_9=ShimEng.dll + 0x6a78
TRUSTED_CALLER_10=xpshims.dll + 0x1960
TRUSTED_CALLER_11=xpshims.dll + 0x1975
TRUSTED_CALLER_12=Flash32_11_4_402_278.ocx + 0x4ace5c

## NtAllocateVirtualMemory Callers:
TRUSTED_CALLER_13=IEFRAME.dll + 0xa4efc
TRUSTED_CALLER_14=RPCRT4.dll + 0x8b4f6
TRUSTED_CALLER_15=IEUI.dll + 0xd430
TRUSTED_CALLER_16=Flash32_11_4_402_278.ocx + 0x68844d
```

Memory

Prototype Implementation

CWXDetector

Windows Prototype

- Windows XP 32 Bit, but easy to migrate
- Kernel driver
  - hooks some system calls
  - instruments page fault handler
- Usermode application
  - to control the driver
  - and log the data
- Modes of operation
  - fully automated
  - interactive

Difficulties

- Windows is not open source
  - reverse page fault handler
  - reverse memory related system calls
- Modifying the paging structures is not sufficient
  - reverse memory management objects and consider *virtual address descriptors (VADs), PrototypePTEs, Segments, Subsegments, Sections, …*
- Results published in technical report
  - *Internals of Windows Memory Management (not only) for Malware Analysis,* TR-2011-1, University of Mannheim

Multi Version Dumping

- Redump memory, is modified after initial dumping
- Compare dumps to detect self-modifying shellcode
  - encryption, obfuscation or multi-staging

```
0000:00000000     pop     edx
0000:00000001     nop
0000:00000002     push    esp
0000:00000003     nop
0000:00000004     pop     edx
0000:00000005     jmp     short loc_1C
0000:00000007 ; ---------------------------------------
0000:00000007 loc_7:  ; CODE XREF: seg000:loc_1Cp
0000:00000007     pop     eax
0000:00000008 loc_8:  ; CODE XREF: seg000:00000018j
0000:00000008     mov     ebx, [edx]
0000:0000000A     mov     [eax], ebx
0000:0000000C     add     eax, 4
0000:0000000F     add     edx, 4
0000:00000012     cmp     ebx, 0C0C0C0Ch
0000:00000018     jnz     short loc_8
0000:0000001A     jmp     short loc_21
0000:0000001C ; ---------------------------------------
0000:0000001C loc_1C: ; CODE XREF: seg000:00000005j
0000:0000001C     call    loc_7
0000:00000021 loc_21: ; CODE XREF: seg000:0000001Aj
0000:00000021     db 0
0000:00000022     db 0
0000:00000023     db 0
0000:00000024     db 0
0000:00000025     db 0
```

```
0000:00000000     pop     edx
0000:00000001     nop
0000:00000002     push    esp
0000:00000003     nop
0000:00000004     pop     edx
0000:00000005     jmp     short loc_1C
0000:00000007 ; ---------------------------------------
0000:00000007 loc_7:  ; CODE XREF: seg000:loc_1Cp
0000:00000007     pop     eax
0000:00000008 loc_8:  ; CODE XREF: seg000:00000018j
0000:00000008     mov     ebx, [edx]
0000:0000000A     mov     [eax], ebx
0000:0000000C     add     eax, 4
0000:0000000F     add     edx, 4
0000:00000012     cmp     ebx, 0C0C0C0Ch
0000:00000018     jnz     short loc_8
0000:0000001A     jmp     short loc_21
0000:0000001C ; ---------------------------------------
0000:0000001C loc_1C: ; CODE XREF: seg000:00000005j
0000:0000001C     call    loc_7
0000:00000021 loc_21: ; CODE XREF: seg000:0000001Aj
0000:00000021     mov     eax, 42C363EFh
0000:00000026     sub     ecx, ecx
0000:00000028     fcmovbe st, st
0000:0000002A     fnstenv byte ptr [esp-0Ch]
0000:0000002E     mov     cl, 56h ; 'V'
```

Evaluation

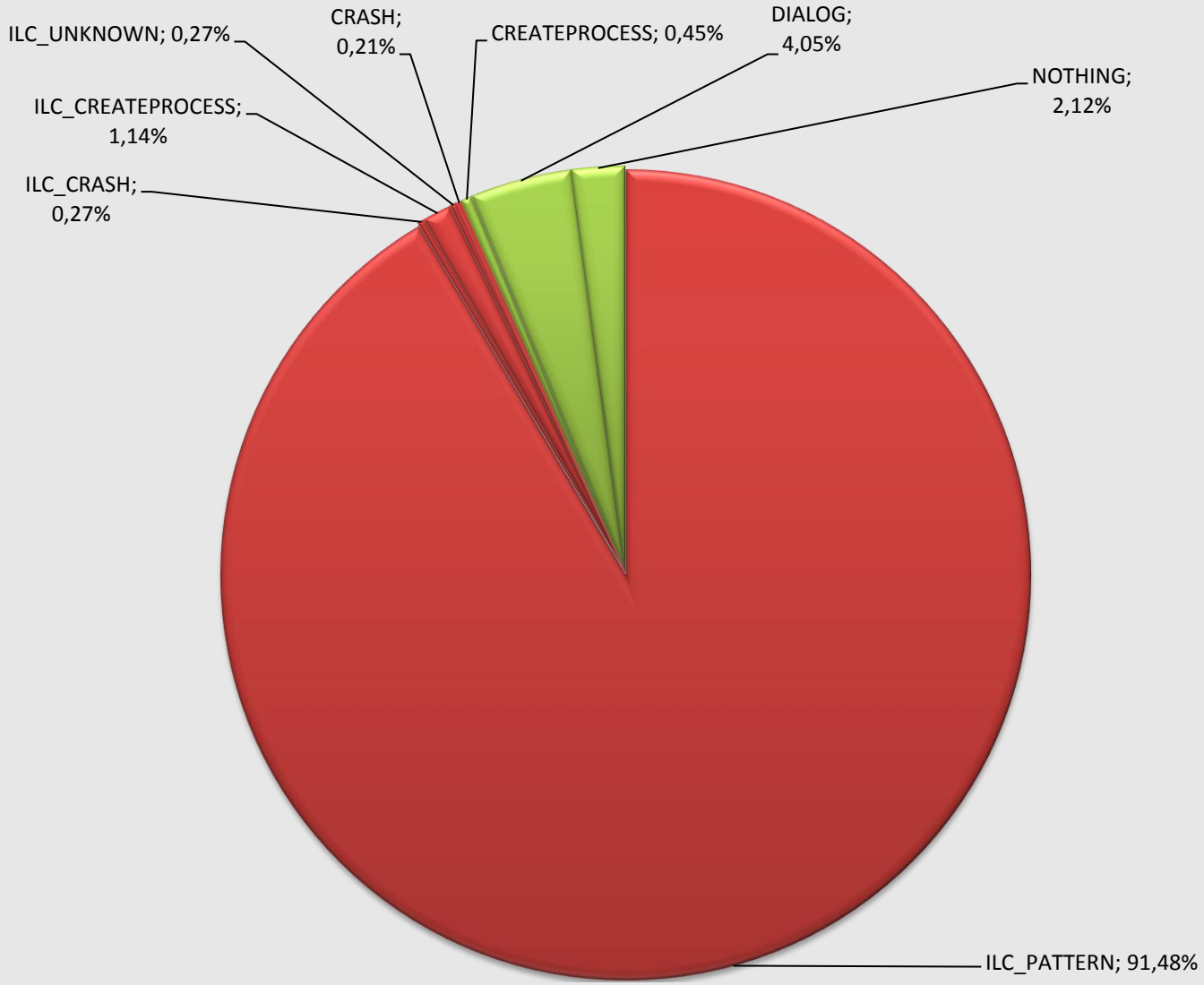
Evaluation of CWXDetector

- Analysis of PDF documents
  - Tested with different applications and combined results
    - Acrobat Reader 6.0.0, 7.0.0, 7.0.7, 8.1.1, 8.1.2, 8.1.6, 9.0.0, 9.2.0, 9.3.0
    - Foxit Reader 3.0.0
  - Set of 7,278 benign documents
    - downloaded from the Alexa‘s Top 2000 sites and AV checked
  - Set of 7,278 malicious documents
    - collected by an AV vendor from different sources
      - sample sharing (70,0%)
      - found in the wild (24,0%)
      - multi-scanner projects, e.g. Virus Total (4,8%)
      - intercepted botnet traffic (1,2%)

Malicious PDF documents

Detection Details

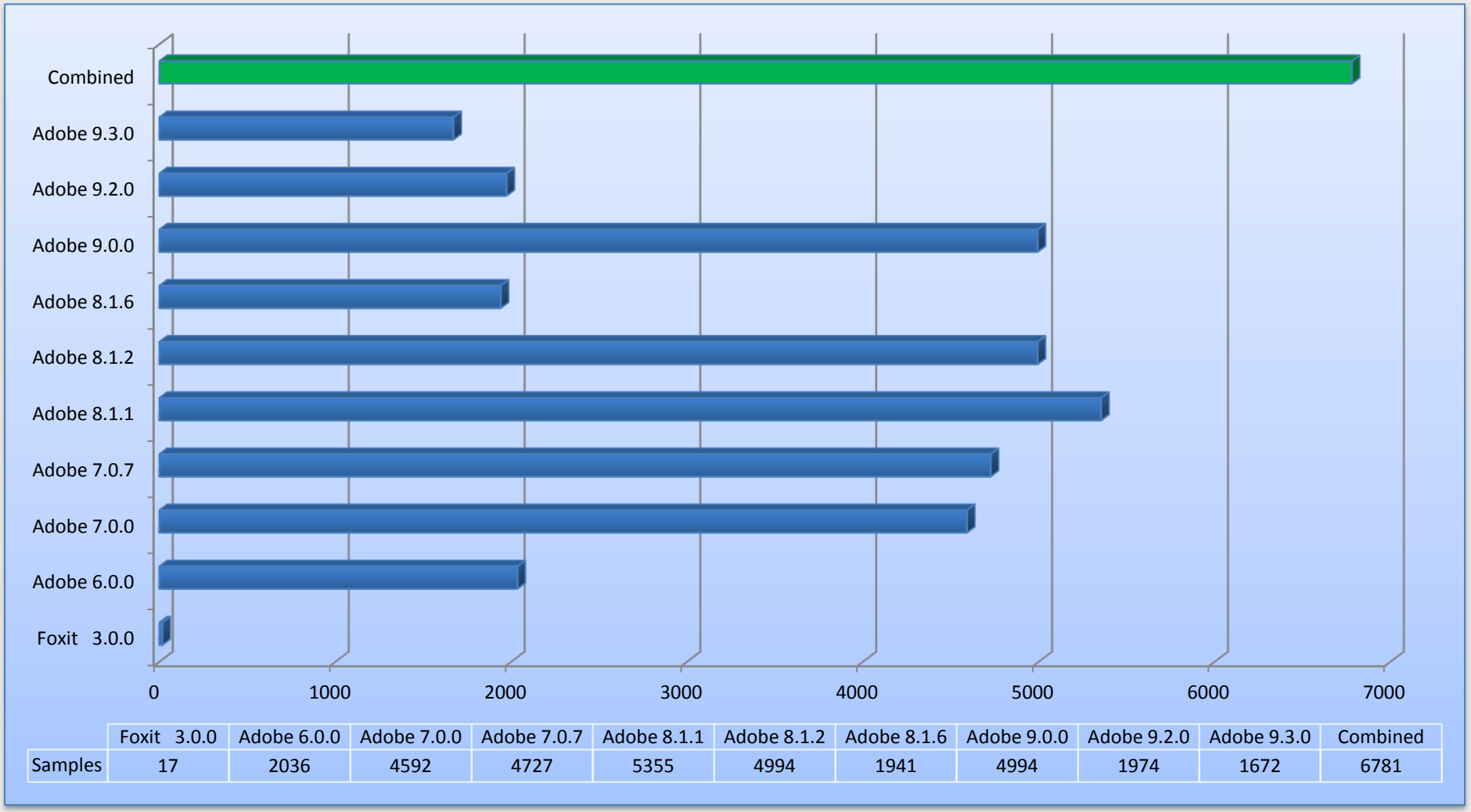
Result	Percent	Samples	
ILC_PATTERN	91,5%	6658	
ILC_CRASH	0,3%	20	
ILC_CREATEPROCESS	1,1%	83	6781
ILC_UNKNOWN	0,3%	20	
CRASH	0,2%	15	
CREATEPROCESS	0,4%	33	
DIALOG	4,1%	295	497
NOTHING	2,1%	154	
Total sum	100,0%	7278	



Order for combining the results:
PATTERN > CRASH > CREATEPROCESS > DIALOG > NOTHING

Malicious PDF documents

Detection by Viewer Application



	Foxit 3.0.0	Adobe 6.0.0	Adobe 7.0.0	Adobe 7.0.7	Adobe 8.1.1	Adobe 8.1.2	Adobe 8.1.6	Adobe 9.0.0	Adobe 9.2.0	Adobe 9.3.0	Combined
Samples	17	2036	4592	4727	5355	4994	1941	4994	1974	1672	6781

Further Evaluation Results

- Benign PDF sample set
  - No false positives
  - Not really a fair test!
    - Documents were collected randomly, no full code coverage
    - However: tried to get PDFs with fancy features, e.g. JavaScript or AcroForms
      - But it‘s really hard to find *benign* PDFs with embedded Flash ☹
- Additional case studies
  - RealVNC client (CVE-2001-0167)
  - Videolan client (CVE-2010-3275)
  - Flash documents (CVE-2011-0611)
  - Internet Explorer (CVE-2012-4969)

Discussion

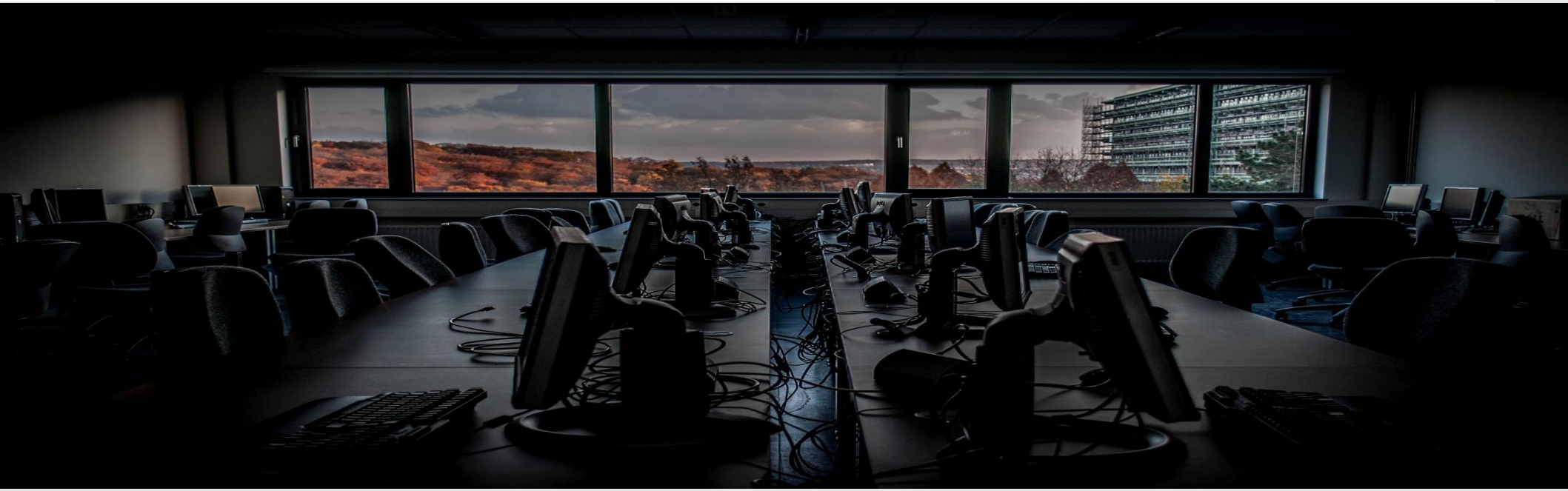
Discussion

- Approach is capable of
  - detecting execution of ILC
  - extracting (different versions of) executed ILC
    - simple form of automatic ILC unpacking
  - working in full-automated manner

- Approach is incapable of
  - detecting ILC that is not executed
  - dealing with full-ROP / JIT-based ILC

- Improvements in next talk *„Down to the bare metal…“*

This is the end …



Thank you for your attention.

Contact at:

carsten.willems@rub.de

Appendix

CVE-2012-4969

ie exec command 0day

```
TARGET_APPLICATION=C:\Program Files\Internet Explorer\iexplore.exe
DEBUGGER_CMD="C:\Program Files\Immunity Inc\Immunity Debugger\ImmunityDebugger.exe" -p
ALLOW_ALL_PROCESSES=1

## NtProtectVirtualMemory Callers:
LEGITIMATE_CALLER_OF_NTPROTECT_1=ntdll.dll+0x1c0e9,1-1
LEGITIMATE_CALLER_OF_NTPROTECT_2=ntdll.dll+0x1cc27,1-1
LEGITIMATE_CALLER_OF_NTPROTECT_3=IEFRAME.dll+0xa4dcd,3-3
LEGITIMATE_CALLER_OF_NTPROTECT_4=IEFRAME.dll+0xa34e9,3-3
LEGITIMATE_CALLER_OF_NTPROTECT_5=IEFRAME.dll+0xa3594,3-3
LEGITIMATE_CALLER_OF_NTPROTECT_6=RPCRT4.dll+0x8b5bf,3-3
LEGITIMATE_CALLER_OF_NTPROTECT_7=IEFRAME.dll+0x9434c,3-3
LEGITIMATE_CALLER_OF_NTPROTECT_8=IEFRAME.dll+0x943f3,3-3
LEGITIMATE_CALLER_OF_NTPROTECT_9=ShimEng.dll+0x6a78,1-1
LEGITIMATE_CALLER_OF_NTPROTECT_10=xpshims.dll+0x1960,3-3
LEGITIMATE_CALLER_OF_NTPROTECT_11=xpshims.dll+0x1975,3-3
LEGITIMATE_CALLER_OF_NTPROTECT_12=Flash32_11_4_402_278.ocx+0x4ace5c,3-3

## NtAllocateVirtualMemory Callers:
LEGITIMATE_CALLER_OF_NTALLOCATE_1=IEFRAME.dll+0xa4efc,3-3
LEGITIMATE_CALLER_OF_NTALLOCATE_2=RPCRT4.dll+0x8b4f6,3-3
LEGITIMATE_CALLER_OF_NTALLOCATE_3=IEUI.dll+0xd430,3-3
LEGITIMATE_CALLER_OF_NTALLOCATE_4=Flash32_11_4_402_278.ocx+0x68844d,3-3
```

CVE-2012-4969

ie exec command 0day

```
[21.9.2012 12:11:24] [   4]    to  0x7c809b42  kernel32.VirtualAllocEx+0x47
[21.9.2012 12:11:24] [   3]  from  0x7c809b54  kernel32.VirtualAllocEx+0x75
[21.9.2012 12:11:24]                   CALL  --------------------
[21.9.2012 12:11:24] [   3]    to  0x7c802511  kernel32._SEH_epilog
[21.9.2012 12:11:24] [   2]  from  0x7c802521  kernel32._SEH_epilog+0x10
[21.9.2012 12:11:24]                    RET  --------------------
[21.9.2012 12:11:24] [   2]    to  0x7c809b59  kernel32.VirtualAllocEx+0x7a
[21.9.2012 12:11:24] [   1]  from  0x7c809b59  kernel32.VirtualAllocEx+0x7a
[21.9.2012 12:11:24]                    RET  --------------------
[21.9.2012 12:11:24] [   1]    to  0x7c809b09  kernel32.VirtualAlloc+0x18
[21.9.2012 12:11:24] [   0]  from  0x7c809b0a  kernel32.VirtualAlloc+0x19
[21.9.2012 12:11:24]                 ROP-RET  ####################
[21.9.2012 12:11:24] [   0]    to  0x0c18fa00
[21.9.2012 12:11:24]
Dissasembly at 0x0c18fa00:
[21.9.2012 12:11:24]      0x0c18fa00   90                      nop
[21.9.2012 12:11:24]      0x0c18fa01   90                      nop
[21.9.2012 12:11:24]      0x0c18fa02   90                      nop
[21.9.2012 12:11:24]      0x0c18fa03   90                      nop
[21.9.2012 12:11:24]      0x0c18fa04   90                      nop
[21.9.2012 12:11:24]      0x0c18fa05   90                      nop
[21.9.2012 12:11:24]      0x0c18fa06   90                      nop
[21.9.2012 12:11:24]      0x0c18fa07   90                      nop
[21.9.2012 12:11:24]      0x0c18fa08   90                      nop
[21.9.2012 12:11:24]      0x0c18fa09   90                      nop
[21.9.2012 12:11:24] >> [c]ontinue, continue [a]ll, [b]reak, break-and-[l]etgo or [t]erminate?
```