Breaking Payloads with Runtime Code Stripping and Image Freezing

Collin Mulliner
Matthias Neugschwandtner

Black Hat USA, August 6th, 2015, Las Vegas

Who We Are (postdocs!)

Collin Mulliner

Postdoc @ Northeastern University

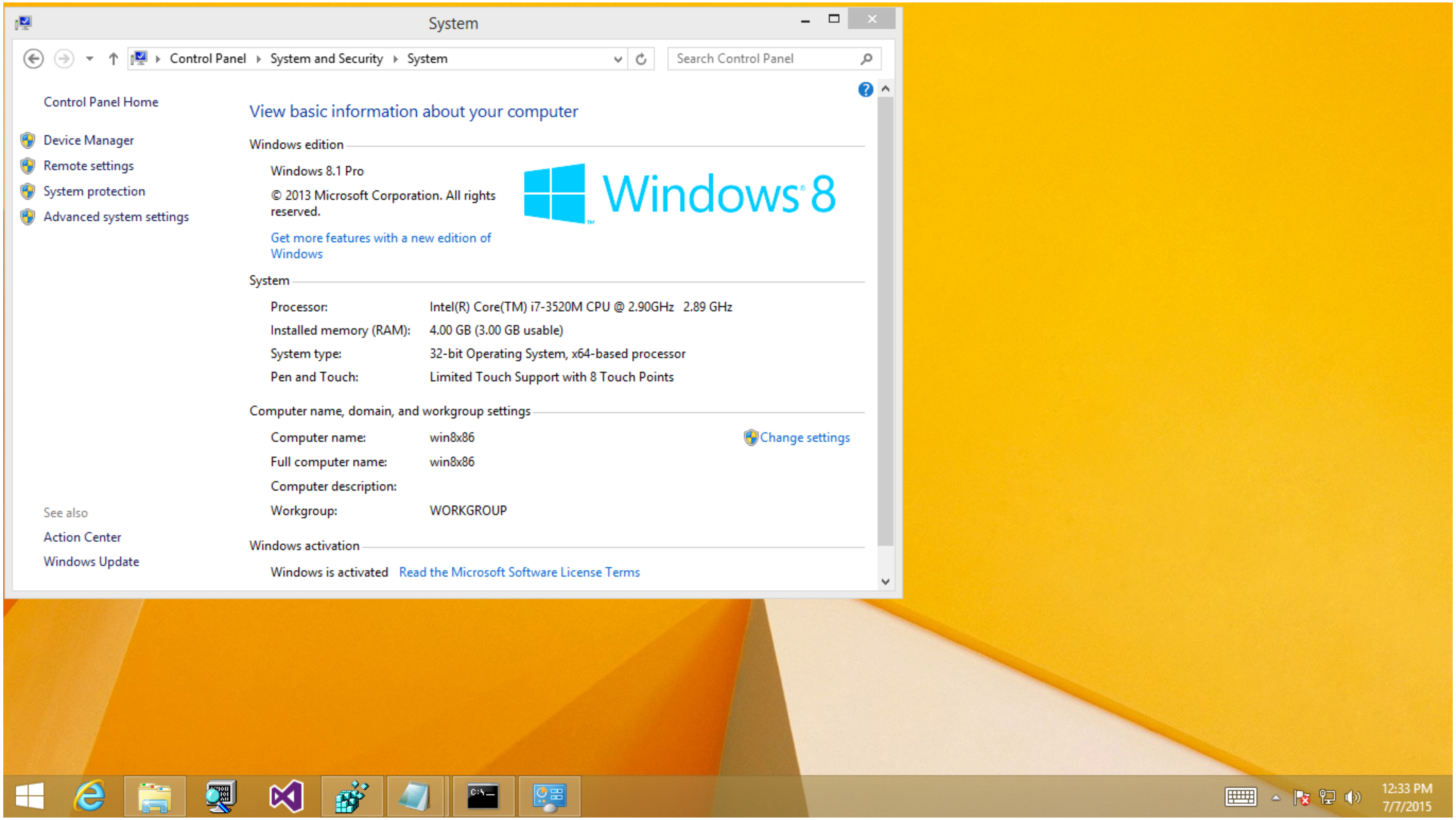
http://www.mulliner.org/collin/

🐦 @collinrm

Matthias Neugschwandtner

Postdoc @ IBM Research Zurich

This is about Windows Security!



View basic information about your computer

Windows edition

Windows 8.1 Pro

© 2013 Microsoft Corporation. All rights reserved.

Get more features with a new edition of Windows

System

Processor:	Intel(R) Core(TM) i7-3520M CPU @ 2.90GHz 2.89 GHz
Installed memory (RAM):	4.00 GB (3.00 GB usable)
System type:	32-bit Operating System, x64-based processor
Pen and Touch:	Limited Touch Support with 8 Touch Points

Computer name, domain, and workgroup settings

Computer name:	win8x86
Full computer name:	win8x86
Computer description:	
Workgroup:	WORKGROUP

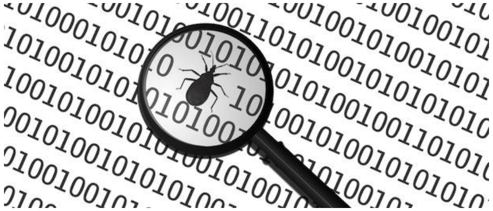
Change settings

Windows activation

Windows is activated Read the Microsoft Software License Terms

Securing Software (is hard!)

- Software has bug(s)
  - Very hard to fix, even harder without source



- Bugs get exploited
  - Preventing initial exploitation (PC ctrl) is hard

- Exploit has a payload
  - Payload is complex code (or it is just a PoC)

⇒ **Let’s break the payload!**

Exploits and Payloads

- Initial stage gains program counter control
  - PC can be directed anywhere
    - Code on stack, heap, or ROP chain

- Attacker has to prepare and execute payload
  - Unless the payload is ROP only!

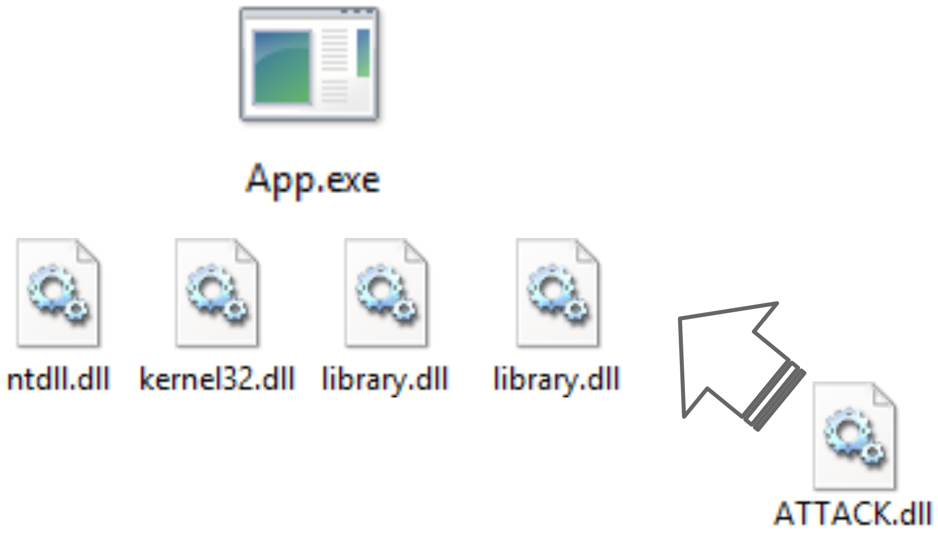
- Preparing the payload
  - Make memory containing payload executable
  - Load DLL (automatically sets memory executable)

Payload

- Uses system functionality
  - Code present in DLLs loaded by target application
  - Payload can load additional DLLs if needed

- Windows shellcode
  - System interaction via DLL ⇒ no direct syscalls
  - Syscall interface not stable across Windows versions (or even Service Packs)

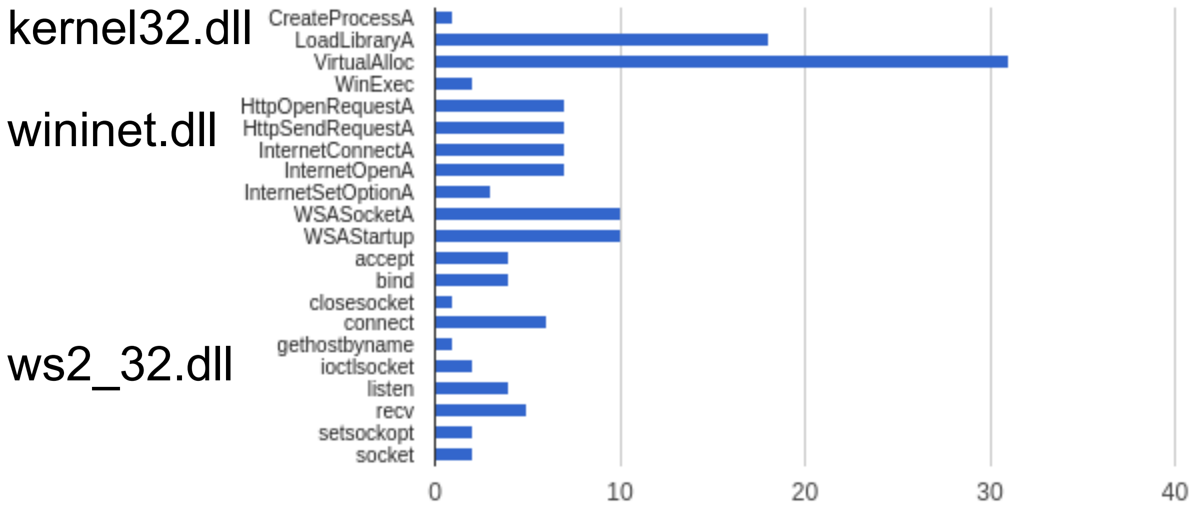
Loading the Payload



'ATTACK.dll' can be loaded from anywhere

- Local
- Network

Payloads and DLLs



Windows API functions used by 34 Metasploit payloads

Previous / Related Work

- EMET (Microsoft)
  - Prevents setting stack and heap executable
  - Prevents loading DLLs over UNC
    - Easy bypass… *Aaron Portnoy's "Bypass all Of the Things"* talk
  - A lot of other stuff …

- EMET is a DLL
  - Injected into process
  - Hooks critical APIs (in user space)

- We follow up on EMET and try to improve it!

Breaking the Payload

This is what this project is about!

Breaking the Payload

- Payload needs specific functionality
  - Functionality == API calls

- Functionality
  - Might NOT be used by the target application
  - Still is available since DLL is loaded by process

Breaking the Payload

- Payload needs specific functionality
  - Functionality == API calls

- Functionality
  - Might NOT be used by the target application
  - Still is available since DLL is loaded by process

⇒ Our idea: remove functionality that is not used by the application

Remove unused Functionality

- Example: server application
  - Uses: `socket(),bind(),listen(),accept()`
  - Does NOT use: `connect()`

- Removing `connect()` means
  - Payload cannot connect back home

Remove unused Functionality

- Example: server application
  - Uses: `socket(),bind(),listen(),accept()`
  - Does NOT use: `connect()`

- Removing `connect()` means
  - Payload cannot connect back home

- **Imagine getting rid of critical functions**
  - **Process creation, library loading, networking, ...**

Remove unused Functionality

- Example: server application
  - Uses: `socket(),bind(),list` `pt()`
  - Does NOT use: `connect()`

- Removing `con` eans
  - Payload c back home

- **I etting rid of critical functions**
  - **ess creation, library loading, networking, ...**

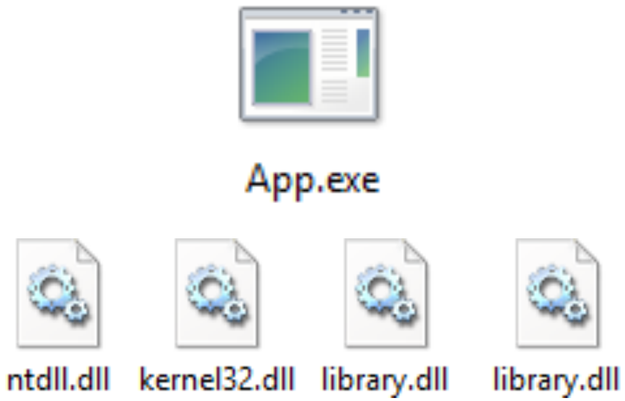
Attack Surface Reduction

“Modern” Software

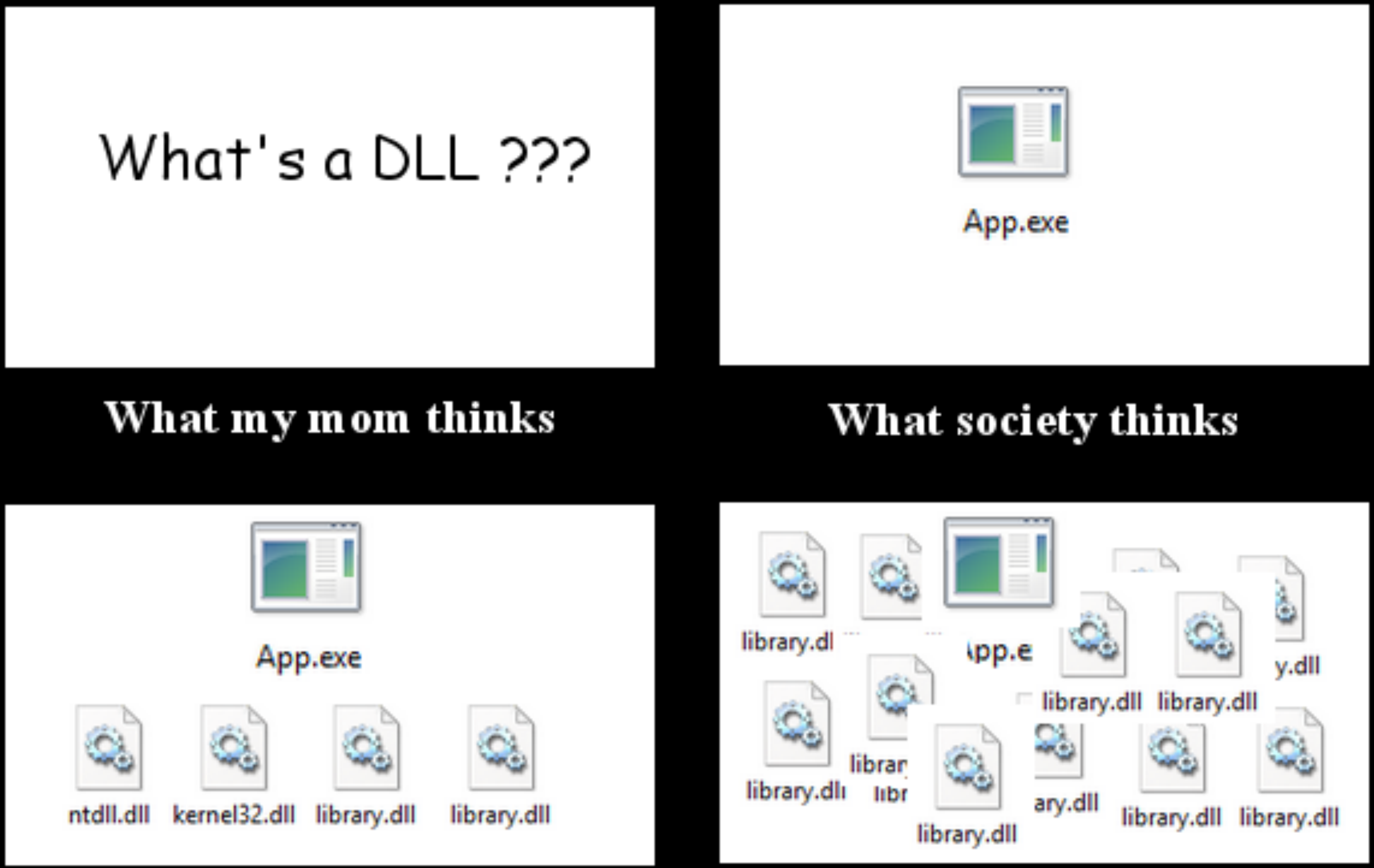
- Applications rely on shared code
  - You do not want to reinvent the wheel
  - File I/O, GUI, networking, ...

- Dynamic Link Library (DLL)
  - Shared library that is linked/loaded at runtime
- OS provides “basic” DLLs
  - DLLs can be updated independent from the app

“Modern” Software



of DLLs used by App



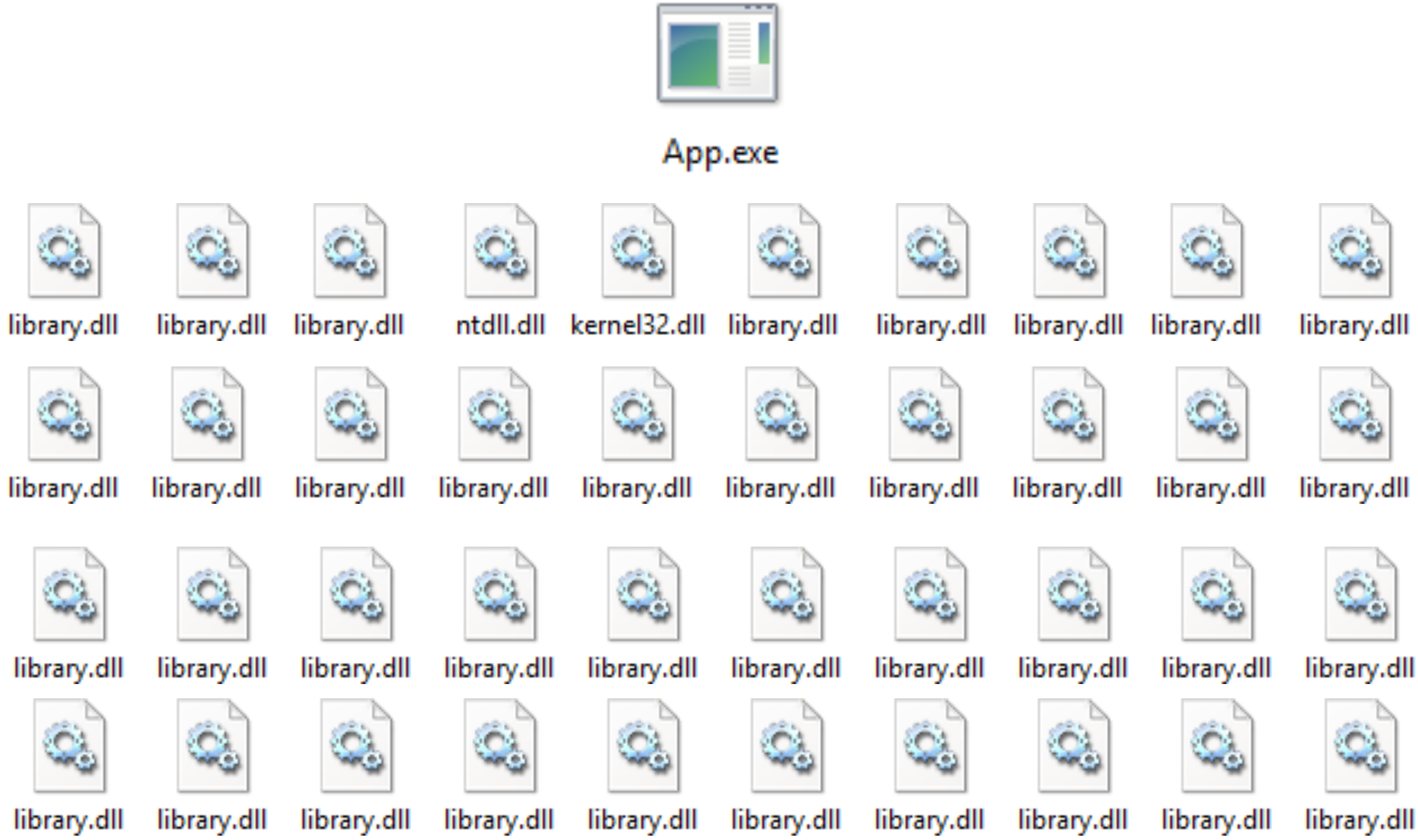
What my mom thinks

What society thinks

What I think

What actually happens

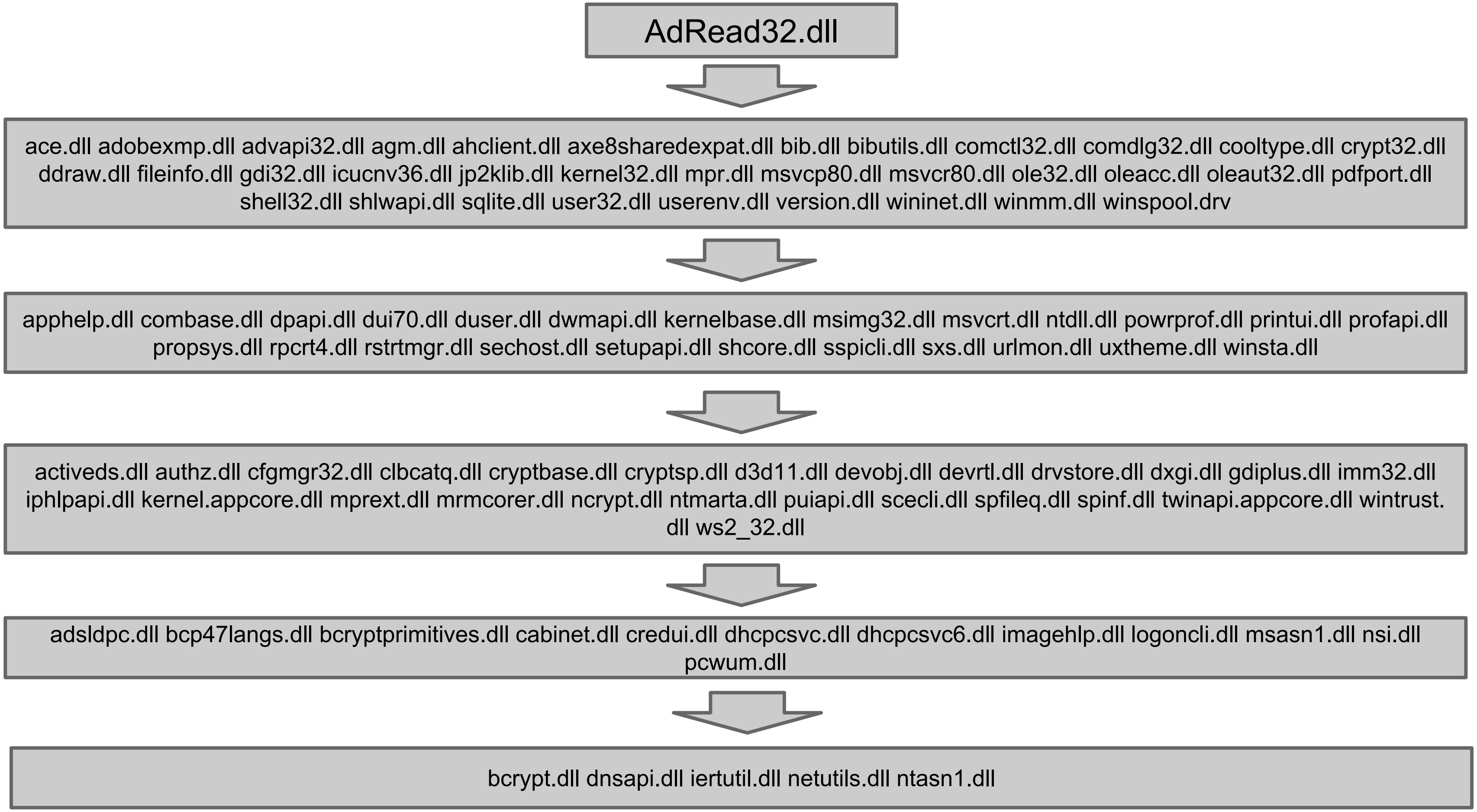
“Modern” Software



DLL Usage

- Applications use many DLLs
  - +100 DLLs is not unusual!
- DLL provides specific functionality
  - Specific still has a pretty broad meaning here!
- Applications only use a subset of each DLL
  - DLL is loaded if **one** symbol is used by the app
- Process has access to all code of a DLL

Adobe Reader DLL Dependencies



AdRead32.dll

ace.dll adobexmp.dll advapi32.dll agm.dll ahclient.dll axe8sharedexpat.dll bib.dll bibutils.dll comctl32.dll comdlg32.dll cooltype.dll crypt32.dll ddraw.dll fileinfo.dll gdi32.dll icucnv36.dll jp2klib.dll kernel32.dll mpr.dll msvcp80.dll msvcr80.dll ole32.dll oleacc.dll oleaut32.dll pdfport.dll shell32.dll shlwapi.dll sqlite.dll user32.dll userenv.dll version.dll wininet.dll winmm.dll winspool.drv

apphelp.dll combase.dll dpapi.dll dui70.dll duser.dll dwmapi.dll kernelbase.dll msimg32.dll msvcrt.dll ntdll.dll powrprof.dll printui.dll profapi.dll propsys.dll rpcrt4.dll rstrtmgr.dll sechost.dll setupapi.dll shcore.dll sspicli.dll sxs.dll urlmon.dll uxtheme.dll winsta.dll

activeds.dll authz.dll cfgmgr32.dll clbcatq.dll cryptbase.dll cryptsp.dll d3d11.dll devobj.dll devrtl.dll drvstore.dll dxgi.dll gdiplus.dll imm32.dll iphlpapi.dll kernel.appcore.dll mprext.dll mrmcorer.dll ncrypt.dll ntmarta.dll puiapi.dll scecli.dll spfileq.dll spinf.dll twinapi.appcore.dll wintrust.dll ws2_32.dll

adsldpc.dll bcp47langs.dll bcryptprimitives.dll cabinet.dll credui.dll dhcpcsvc.dll dhcpcsvc6.dll imagehlp.dll logoncli.dll msasn1.dll nsi.dll pcwum.dll

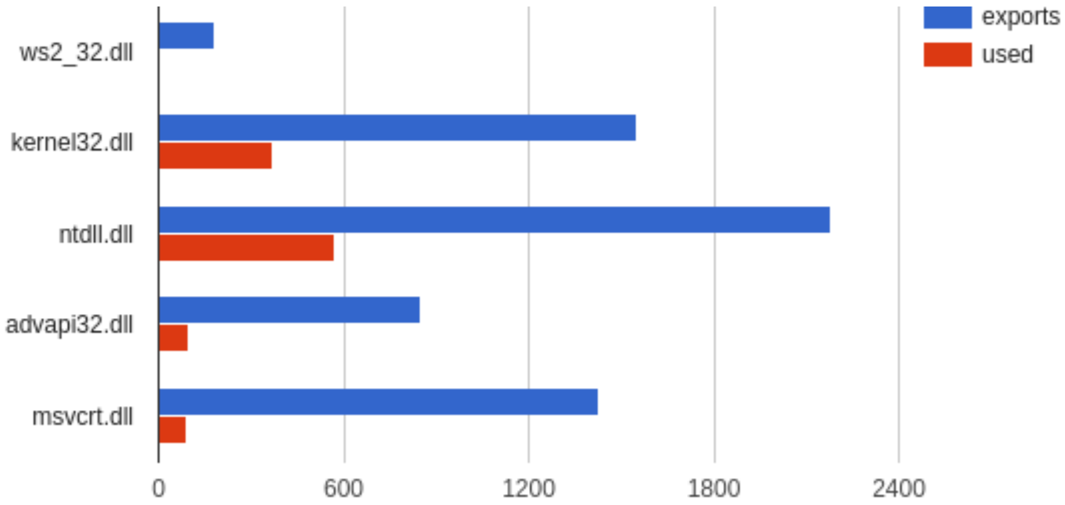
bcrypt.dll dnsapi.dll iertutil.dll netutils.dll ntasn1.dll

= 101 DLLs

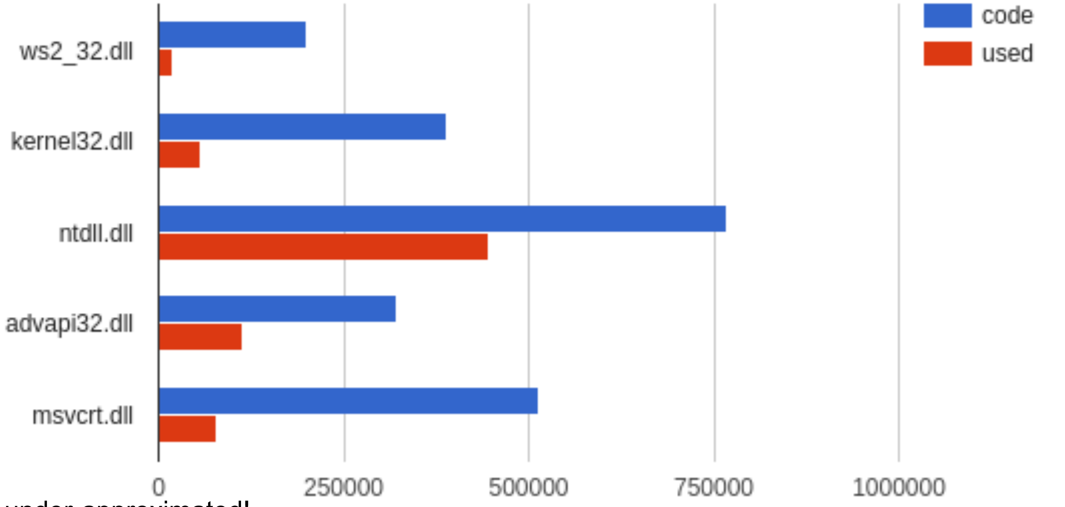
disclaimer: data solely based on static analysis and under-approximated!

Adobe Reader DLL Usage

exported function usage



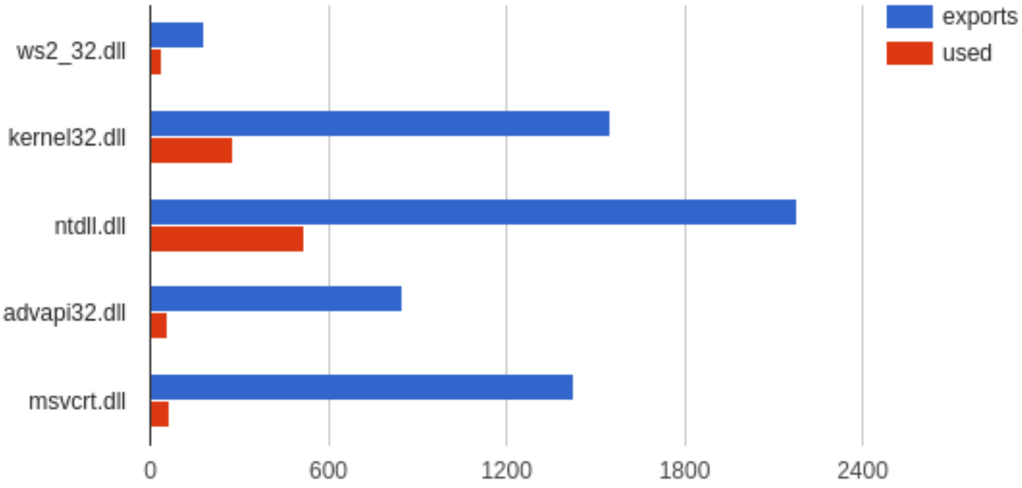
code usage (bytes)



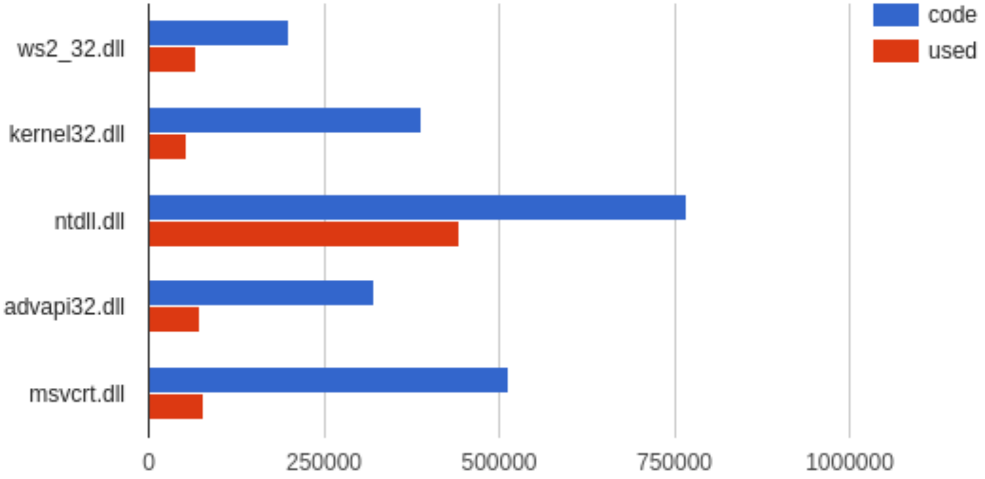
disclaimer: data solely based on static analysis and under-approximated!

Viber DLL Usage

exported function usage



code usage (bytes)



disclaimer: data solely based on static analysis and under-approximated!

Introducing: CodeFreeze

- Code Stripping

- Image Freezing

CodeFreeze

- **Code Stripping**
  - Remove unused code from process
  - Specifically remove unused DLL code

- **Image Freezing**

CodeFreeze

- **Code Stripping**
  - Remove unused code from process
  - Specifically remove unused DLL code

- **Image Freezing**
  - Image == process memory image
  - Prevent adding new code
  - Specifically prevent creating new executable pages
  - Prevent code injection (incl. loading libraries)

CodeFreeze

- **Code Stripping**
  - Remove unused code from process
  - Specifically remove unused DLL code

- **Image Freezing**
  - Image == process memory image
  - Prevent adding new code
  - Specifically prevent creating new executable pages
  - Prevent code injection (incl. loading libraries)

⇒ Do all of this at runtime!

Code Stripping

- What code is not used?
    - Application (PE file) has import table
    - DLL name and function (symbol) name
        - e.g. `ntdll!ExitProcess`

- DLL has an export table
    - e.g. ntdll.dll `ExitProcess @ 0x13374223`

⇒ Unused code == func that are not imported!?

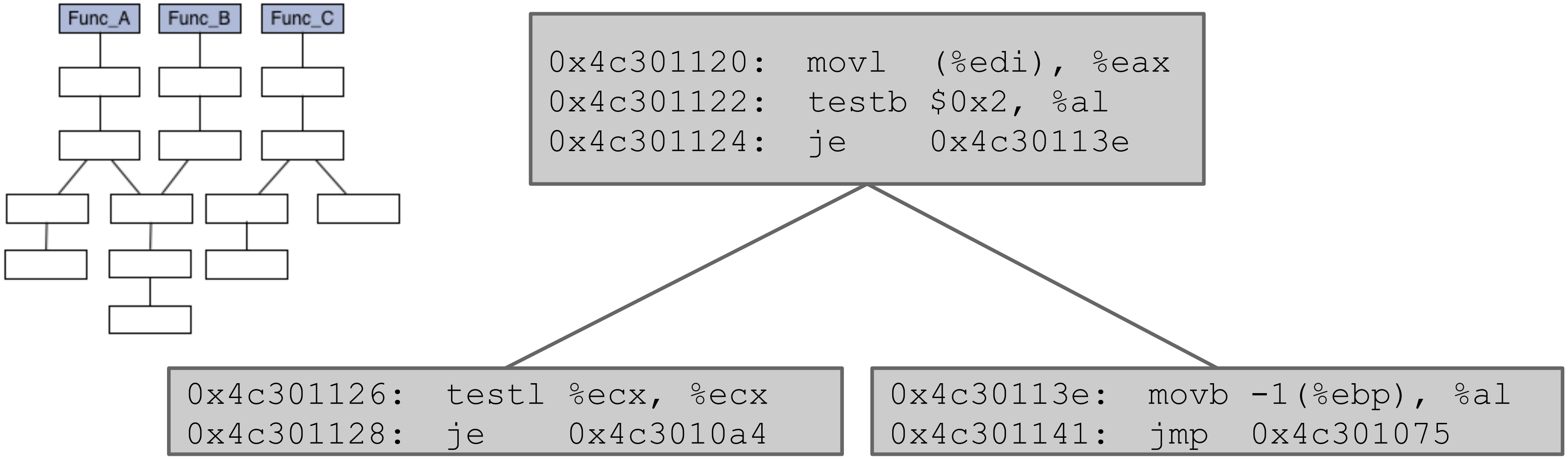
- But of course dynamic lookup `GetProcAddress()`
    - More details on this later in this talk

Control Flow Graph (CFG)

Split program code into basic blocks at control flow transitions (branch, jump, call, etc.)

- Basic blocks = nodes in the CFG
- Control flow transfers = edges in the CFG

⇒ We know which code regions are used by a function!

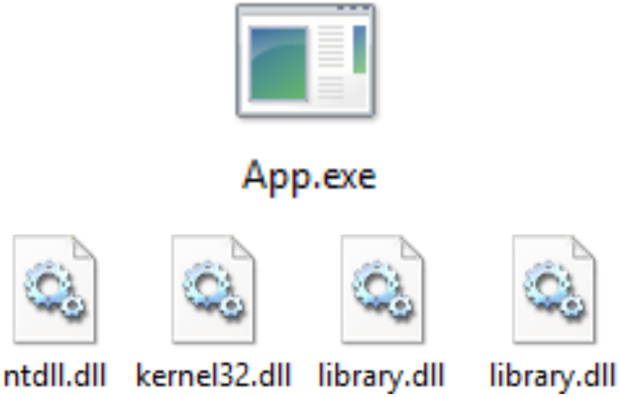


```
0x4c301120:  movl   (%edi), %eax
0x4c301122:  testb  $0x2, %al
0x4c301124:  je     0x4c30113e
```

```
0x4c301126:  testl  %ecx, %ecx
0x4c301128:  je     0x4c3010a4
```

```
0x4c30113e:  movb  -1(%ebp), %al
0x4c301141:  jmp   0x4c301075
```

Code Stripping

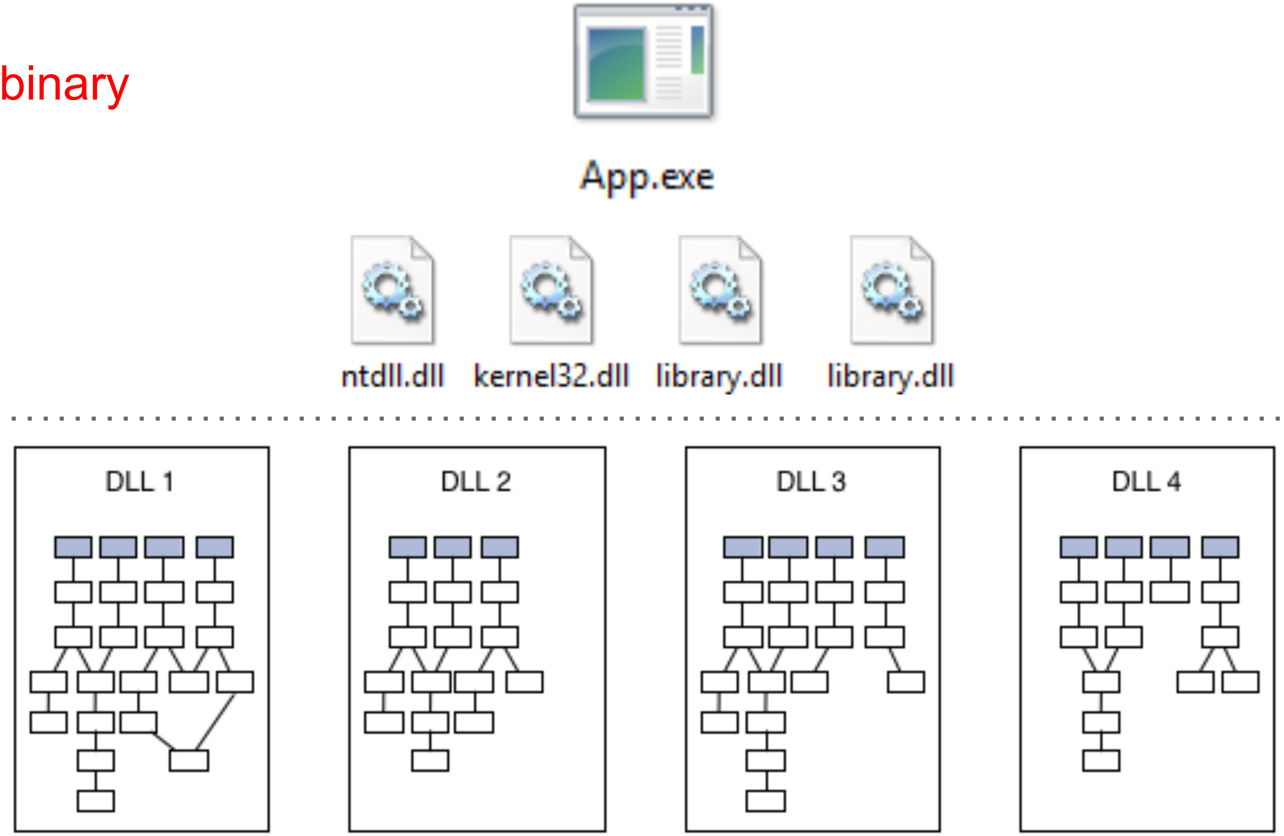


Code Stripping : DLL CFGs

application binary

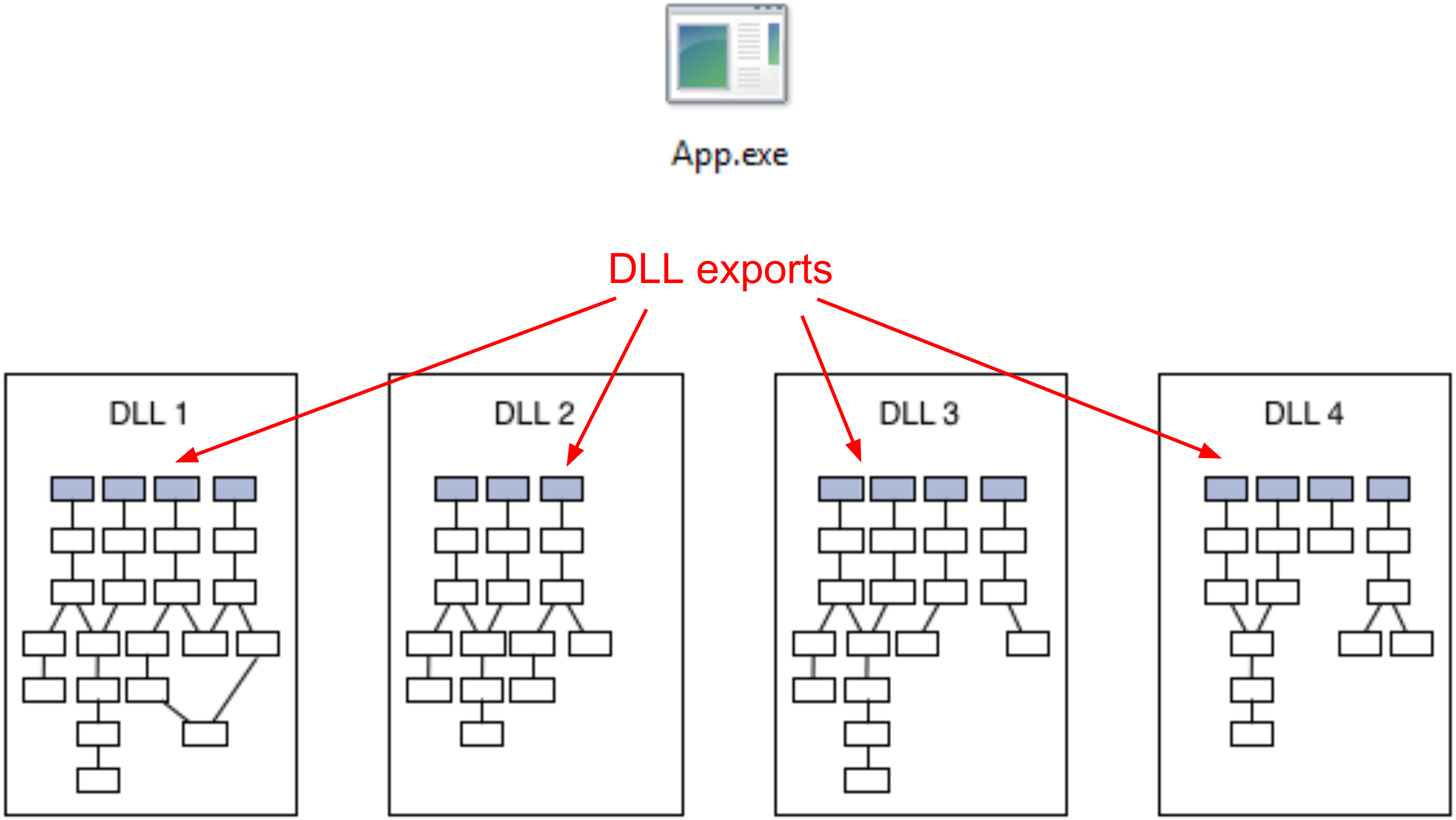
DLLs

DLL CFGs

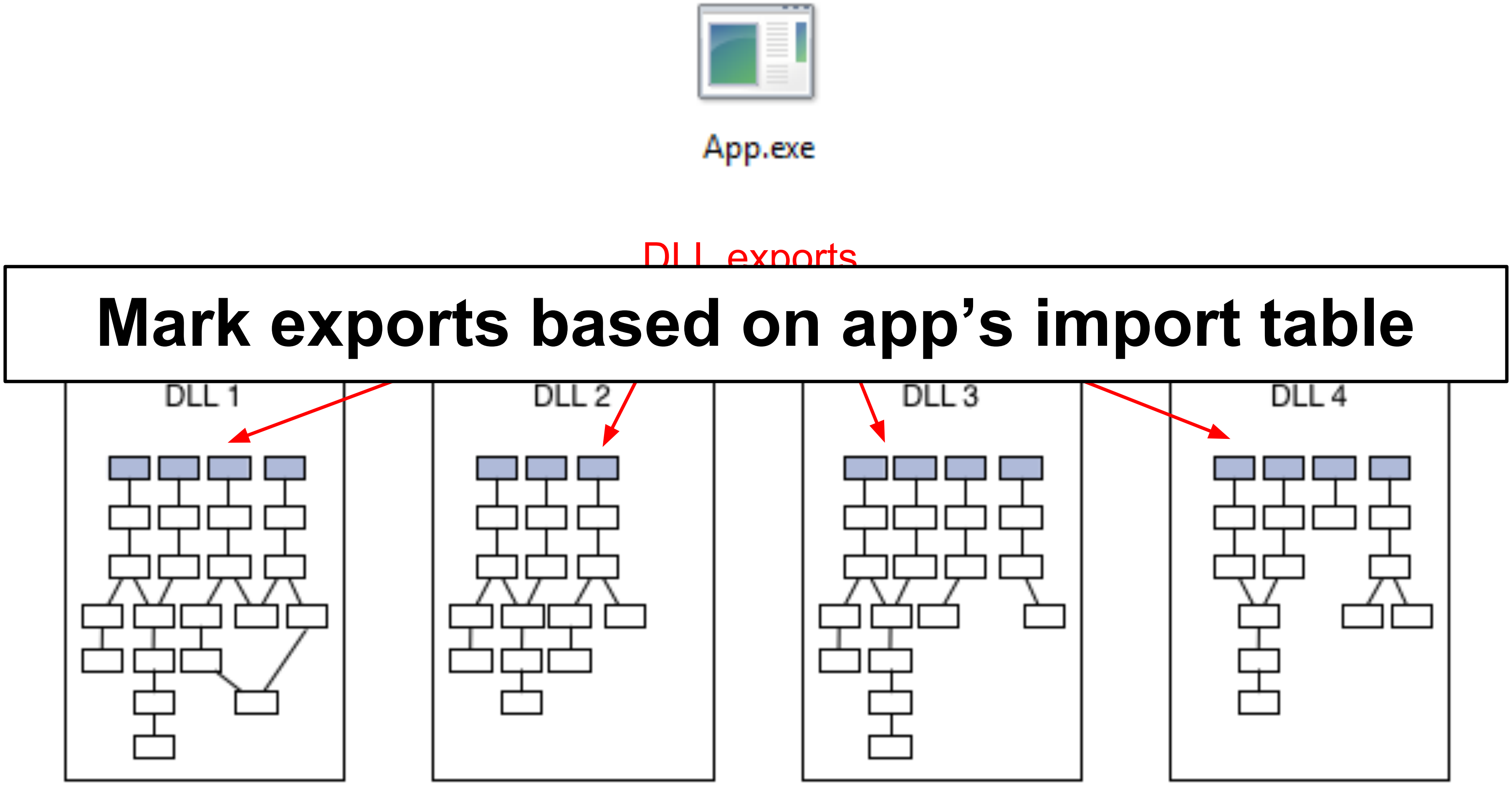


DLL CFGs are independent from the application!

Code Stripping : mark used code

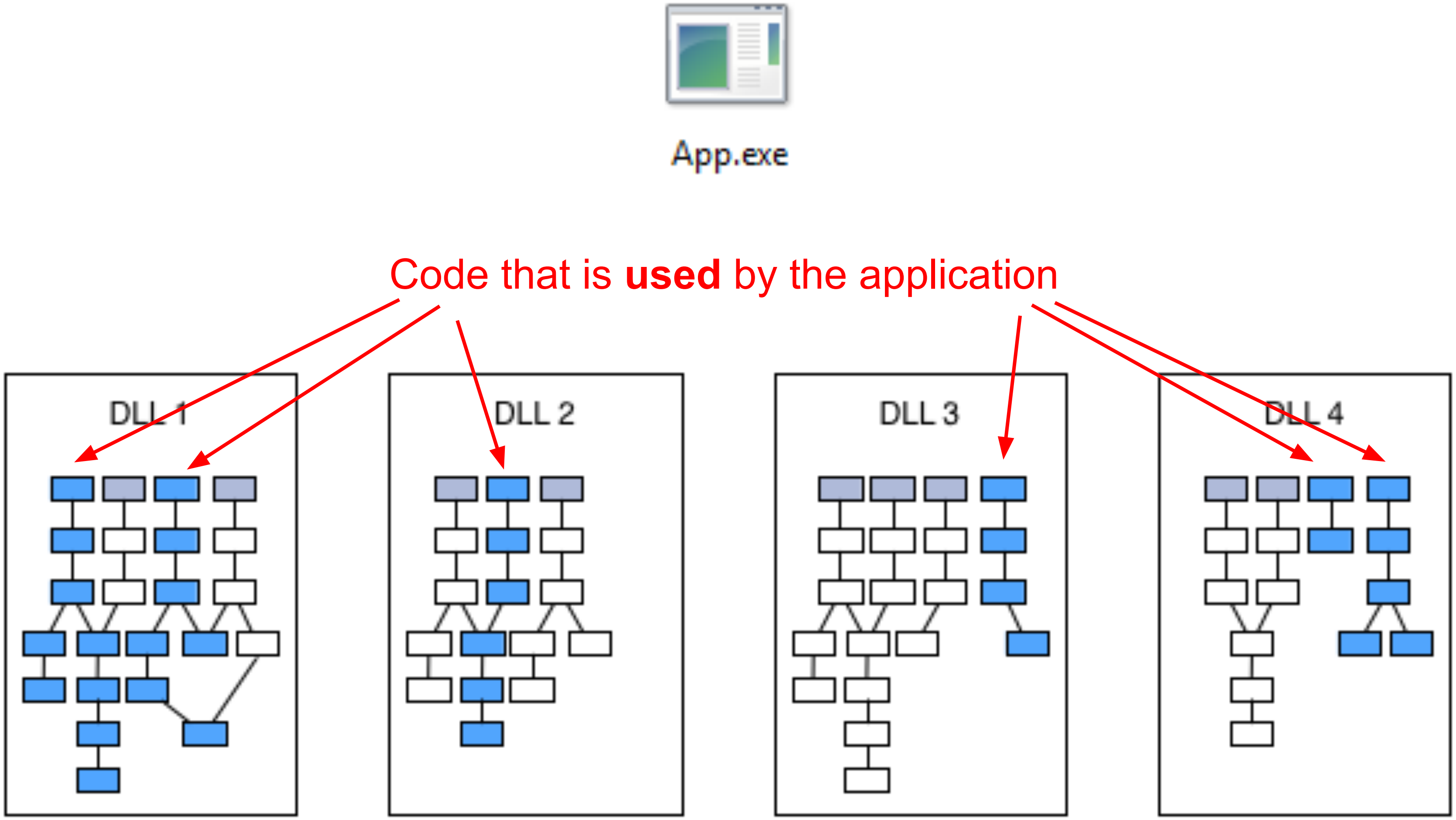


Code Stripping : mark used code

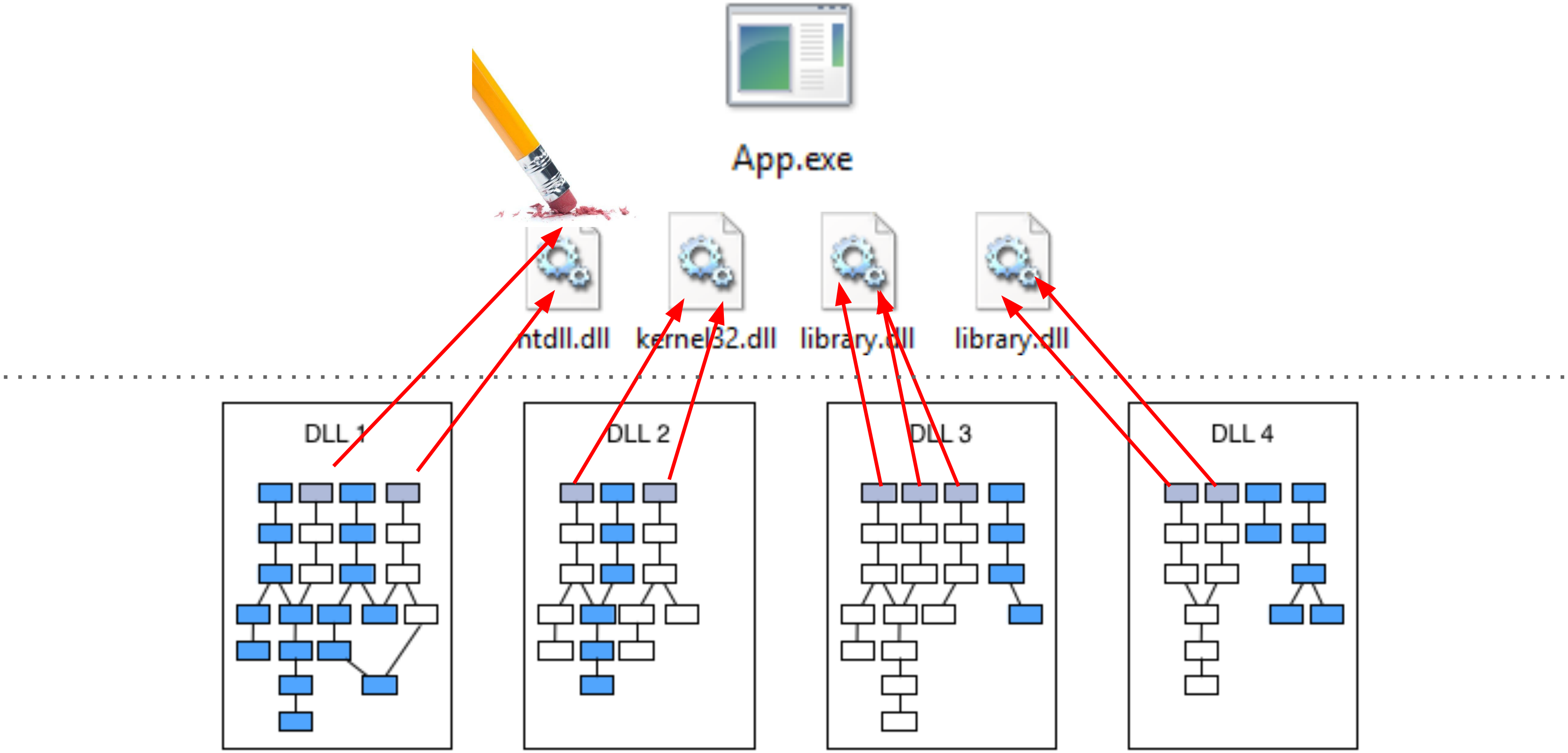


Mark exports based on app’s import table

Code Stripping : mark used code



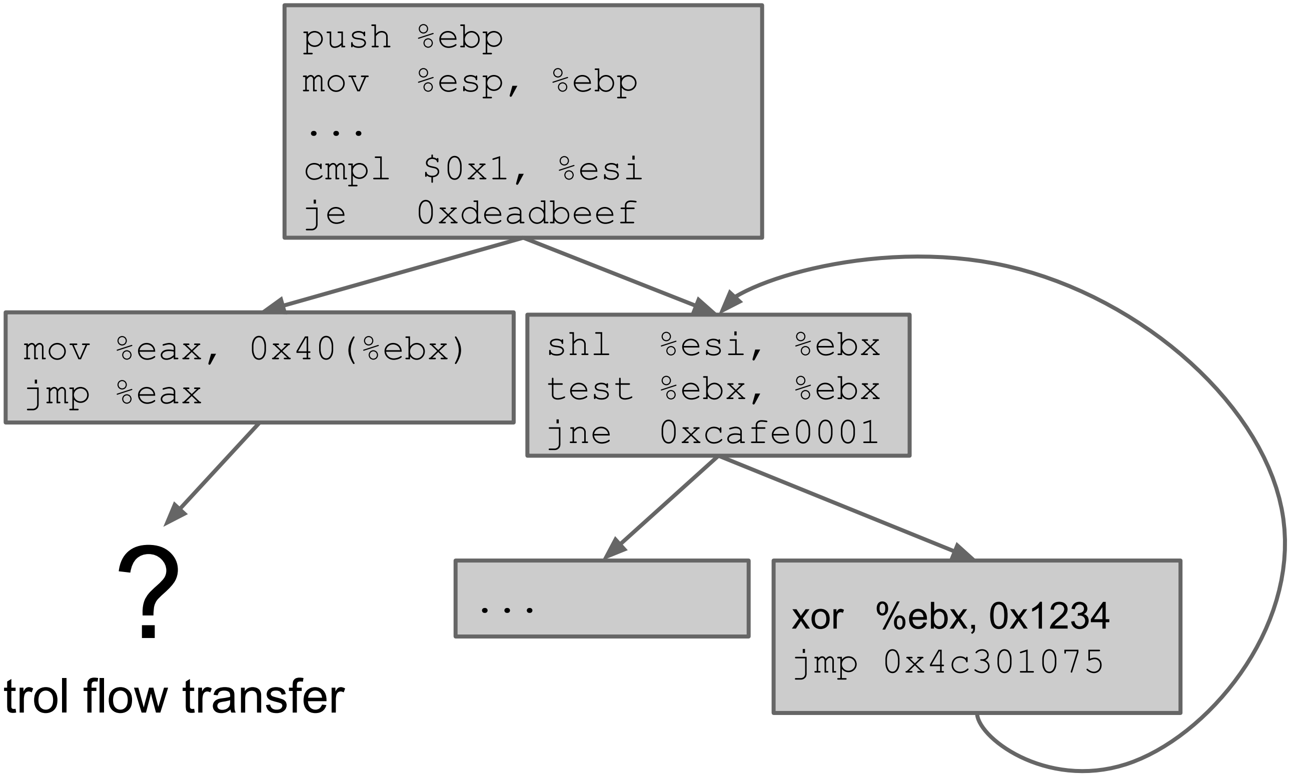
Code Stripping : remove unused code



Control Flow Graph Recovery

Recursive disassembly:

- start disassembly at entry points
- follow all control flow transfers



```
push %ebp
mov  %esp, %ebp
...
cmpl  $0x1, %esi
je   0xdeadbeef
```

```
mov %eax, 0x40(%ebx)
jmp %eax
```

```
shl   %esi, %ebx
test  %ebx, %ebx
jne   0xcafe0001
```

```
...
```

```
xor   %ebx, 0x1234
jmp 0x4c301075
```

Indirect control flow transfer

Control Flow Recovery

- Reasons for indirect control flow transfers:
  - Jump tables (switch statements)
  - Callbacks
  - Virtual function calls via v-tables
  - Dynamic code loading: `GetProcAddress()` et al.
- Abstract interpretation can help!
  - Nifty static program analysis technique
  - Bounded address tracking
    - Try to determine possible values of variables based on a low-level memory model

Control Flow Graph Recovery

- industry-grade recursive disassembly
- basic control flow recovery

+

Jakstab

- academic abstract interpretation framework
- resolve indirect control flow transfers

+

- magic glue code
- pieces CFG together
- processes import/export, API sets, ...

⇒ generates kill files!

Kill Files

- Contain the input used for code stripping
- Callgraph of a DLL
- Nodes = functions +
    - Internal dependencies (code regions)
    - External dependencies (functions imported from other DLLs)

Kill Node (example 1)

```
Process32NextW
0x0000000068997d57 - 0x0000000068997d63
0x0000000068983d59 - 0x0000000068983d69
0x0000000068919842 - 0x000000006891984f
0x000000006891963f - 0x00000000689196d1

NtMapViewOfSection ntdll.dll
NtUnmapViewOfSection ntdll.dll
BaseSetLastNTError kernel32.dll
RtlSetLastWin32Error ntdll.dll
```

Function/Export Name

Code Regions

Dependencies

`(kernel32.dll 0x68900000)`

Kill Node (example 2)

```
WSADuplicateSocketA
0x000000004f7a5c09 - 0x000000004f7a5c75

sub_4F7B1C99 ws2_32.dll
@__security_check_cookie@4 ws2_32.dll
SetLastError kernelbase.dll
WSADuplicateSocketW ws2_32.dll
```

Function/Export Name

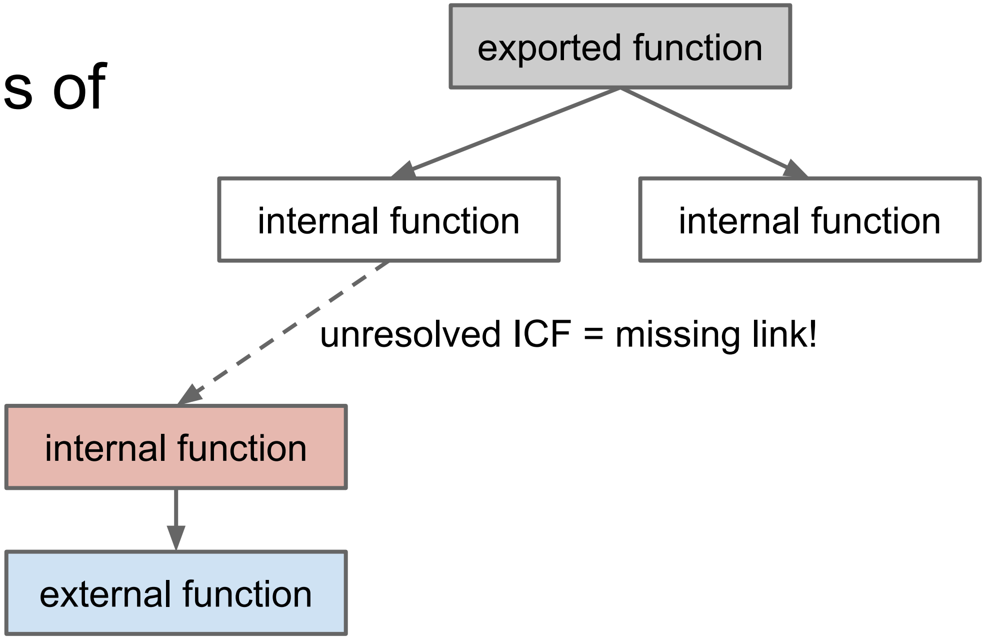
Code Regions

Dependencies

```
(ws2_32.dll 0x4f780000)
```

Orphans

- CFG recovery is far from perfect ⇒ unresolved indirect control flow transfers
- Orphan = function of a DLL that has no parent and is not exported
- External dependencies of orphans need to be whitelisted



CodeFreeze : Implementation

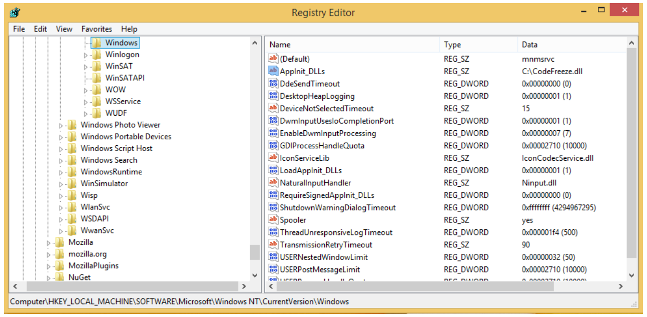
- DLL that is injected into process
  - Similar to EMET

- Executes before application starts
  - We hook the program’s entry point

- Implemented for Windows 8.1 (32bit)
  - 64bit should just work
    - Implementing on 32bit was easier (for me)

DLL Injection

- AppInit_DLL
  - Configure via Windows Registry
  - Feature of User32 (requires that app uses User32)



Name	Type	Data
(Default)	REG_SZ	mnmsrvc
AppInit_DLLs	REG_SZ	C:\CodeFreeze.dll
DdeSendTimeout	REG_DWORD	0x00000000 (0)
DesktopHeapLogging	REG_DWORD	0x00000001 (1)
DeviceNotSelectedTimeout	REG_SZ	15
DwmInputUsesIoCompletionPort	REG_DWORD	0x00000001 (1)
EnableDwmInputProcessing	REG_DWORD	0x00000007 (7)
GDIProcessHandleQuota	REG_DWORD	0x00002710 (10000)
IconServiceLib	REG_SZ	IconCodecService.dll
LoadAppInit_DLLs	REG_DWORD	0x00000001 (1)
NaturalInputHandler	REG_SZ	Ninput.dll
RequireSignedAppInit_DLLs	REG_DWORD	0x00000000 (0)
ShutdownWarningDialogTimeout	REG_DWORD	0xffffffff (4294967295)
Spooler	REG_SZ	yes
ThreadUnresponsiveLogTimeout	REG_DWORD	0x000001f4 (500)
TransmissionRetryTimeout	REG_SZ	90
USERNestedWindowLimit	REG_DWORD	0x00000032 (50)
USERPostMessageLimit	REG_DWORD	0x00002710 (10000)

Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows

- QueueUserAPC (for apps without User32)
  - Similar to CRT but works on suspended processes
  - Technique borrowed from Cuckoo Sandbox

Running an App with CodeFreeze

- Run app with **CodeFreezeMonitor**
  - Collect DLLs that loaded by the process
  - Collect names of dynamic resolved functions
    - via `GetProcAddress()`
  - Determine if app alloces pages with EXEC bit

- Generate CodeFreeze config for app
  - List of DLLs that need to be preloaded
  - List of functions that need to be whitelisted
  - Settings for memory permissions

DLL Preloading

- **Strip code from dynamically loaded DLLs**

- We use `LoadLibrary()` to load DLLs before application starts
  - This will map the DLL into the app’s memory space

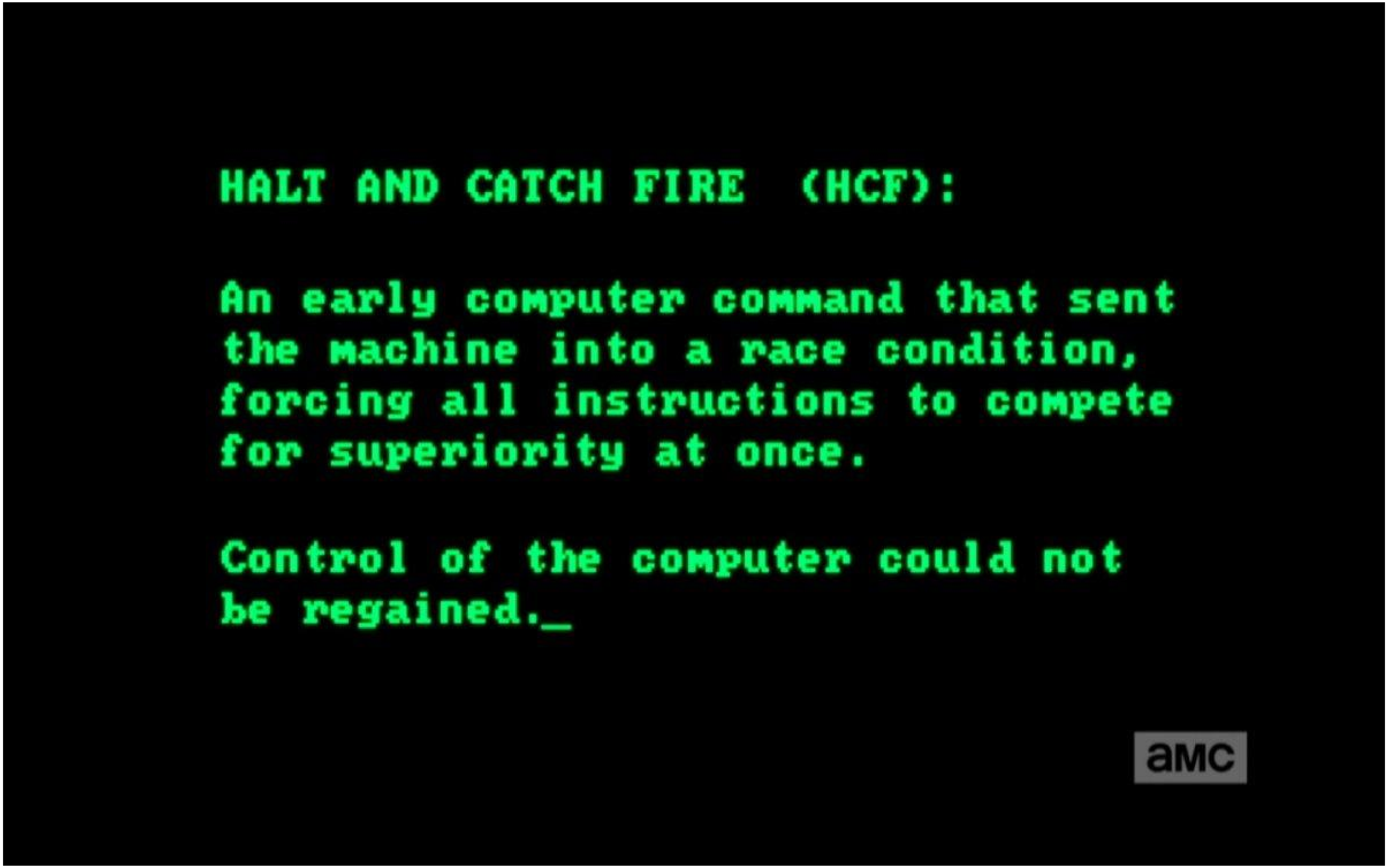
- Call to `LoadLibrary()` by application will just return handle to the loaded DLL
  - Every DLL is loaded only once!

How to Actually Strip Code?

- “Remove” code from process memory
  - Remove == make unavailable

- Remove ⇒ overwrite code in memory

  (will discuss other options later)
  - What do we overwrite with?
  - Multiple bytes
    - jump to some handler (show popup?)
  - Single byte
    - `0x00 (zero)`
    - `0x90 (nop)`
    - `0xF4 (hlt)`



HALT AND CATCH FIRE (HCF):

An early computer command that sent the machine into a race condition, forcing all instructions to compete for superiority at once.

Control of the computer could not be regained._

check out: http://en.wikipedia.org/wiki/Halt_and_Catch_Fire

Overwrite with HLT (0xF4)

- Privileged instruction
  - Causes an exception if called from user space
  - Will kill process or trap debugger

- Single byte instruction
  - Can wipe any number of bytes
  - We can just use `memset()`

Image Freezing

- **Prevent process from adding code to it’s virtual address space**

- Essentially prevent adding PAGE_EXEC
  - Also prevent modification of executable pages

⇒ Prevent code injection / library loading

Source of PAGE_EXEC

Map:

`ZwCreateSection()`

(used by dynamic loader and linker to load DLLs)

Allocate:

`ZwAllocateVirtualMemory()`

`VirtualAllocEx()`

Change Protection:

`VirtualProtectEx()`

`NtProtectVirtualMemory()`

Hook Mem API in User Space

- Allow per app page exec policy
  - Selectively allow/disallow calls
  - Support JIT and other “funky” stuff

- Good enough for prototype
  - Can determine if applications will crash due to this

- Real world ⇒ kernel based implementation
  - One-time switch that will disable process’s power to create/change pages with PAGE_EXEC

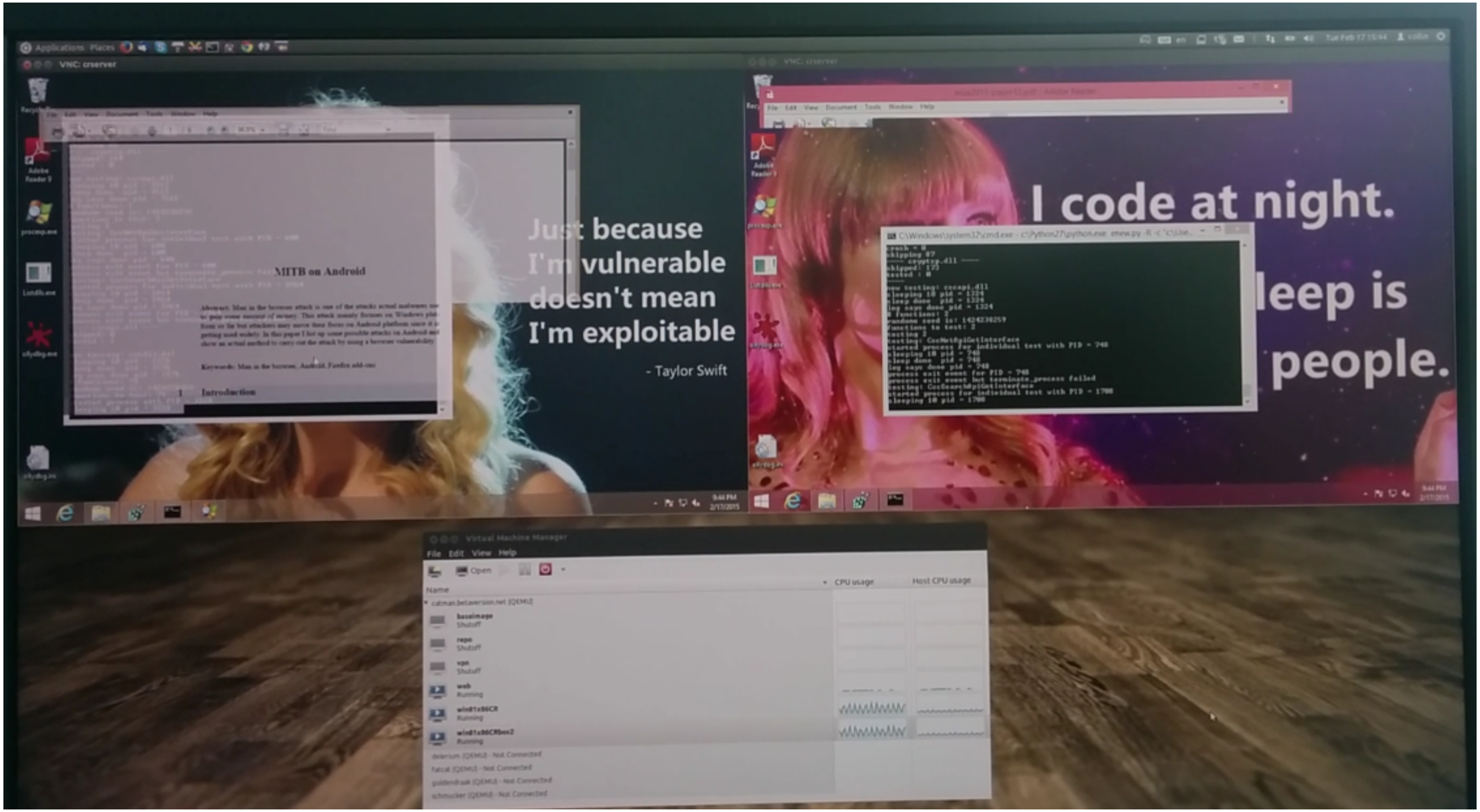
Function Whitelisting

- Static analysis is not sufficient
  - Learn what functions are used
  - Dynamic process, have to execute application

- Run app with CodeFreeze enforcement
  - “Use” app ⇒ cover as much code paths as possible

- App crashes on HLT ⇒ whitelist function
  - Automated tool using PyDbg
    - Lookup crash address in CFG, whitelist resulting function
  - Whitelisting runs on VMs

Whitelisting Functions...



```
C:\Windows\system32\cmd.exe - c:\Python27\python.exe enew.py -R -c "c:\Use...
crash = 0
skipping 87
---- cryptsp.dll ----
skipped: 173
tested : 0
----
new testing: cscapi.dll
sleeping 10 pid = 1324
sleep done  pid = 1324
leg says done pid = 1324
# functions: 2
randome seed is: 1424238259
functions to test: 2
testing 2
testing: CscNetApiGetInterface
started process for individual test with PID = 748
sleeping 10 pid = 748
sleep done  pid = 748
leg says done pid = 748
process exit event for PID = 748
process exit event but terminate_process failed
testing: CscSearchApiGetInterface
started process for individual test with PID = 1708
sleeping 10 pid = 1708
```

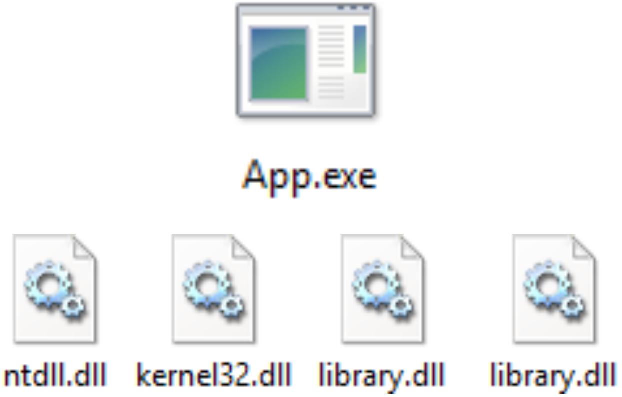
VNC: cserver
MITB on Android
Abstract
Keywords
Introduction
Just because I'm vulnerable doesn't mean I'm exploitable
- Taylor Swift
I code at night. Sleep is for people.
Virtual Machine Manager
File Edit View Help
Open
Name
CPU usage
Host CPU usage
catman.betaversion.net (QEMU)
baseimage
Shutoff
repo
Shutoff
vpn
Shutoff
web
Running
win81x86CR
Running
win81x86CRbox2
Running
delerium (QEMU) - Not Connected
fatcat (QEMU) - Not Connected
goldendraak (QEMU) - Not Connected
schmucker (QEMU) - Not Connected

CodeFreeze at Runtime

CodeFreeze at Runtime

application binary

DLLs

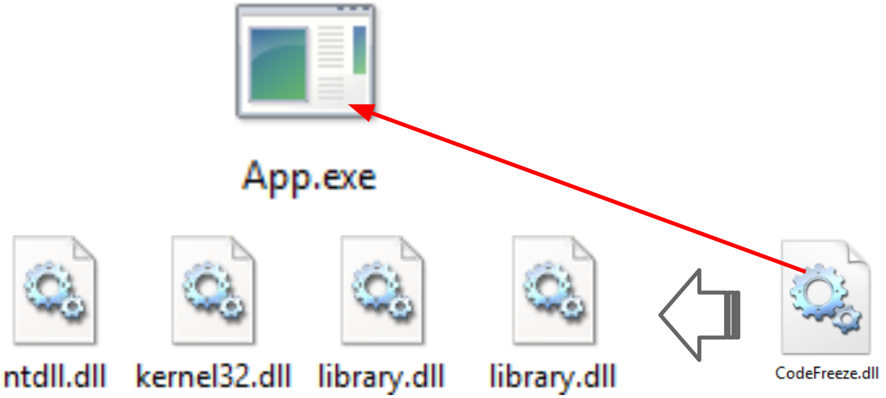


App starts

CodeFreeze at Runtime

application binary

DLLs



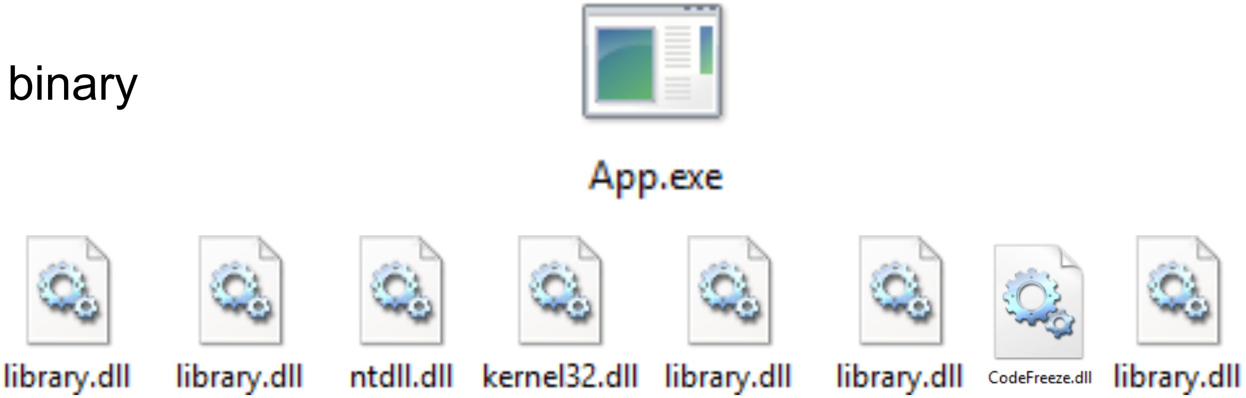
CodeFreeze.DLL gets injected

Hook app entry point

CodeFreeze at Runtime

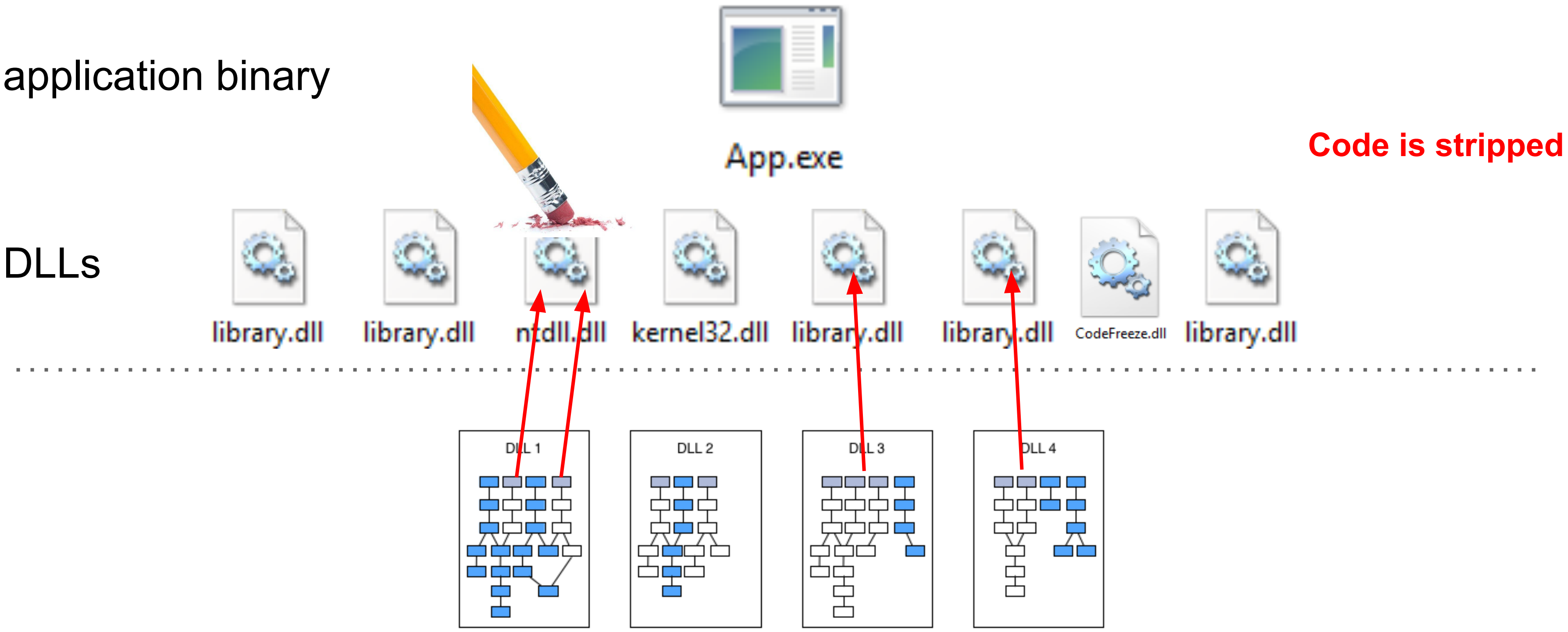
application binary

DLLs



DLLs are preload

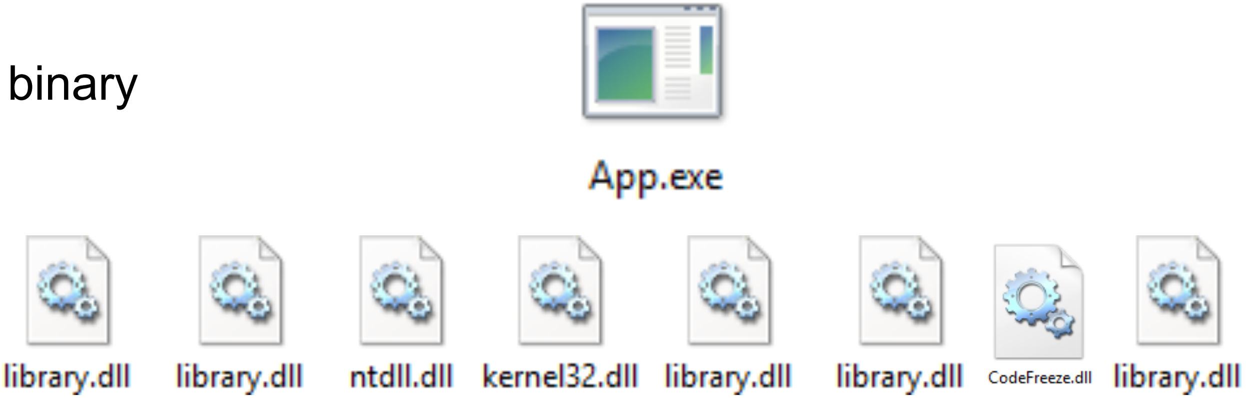
CodeFreeze at Runtime



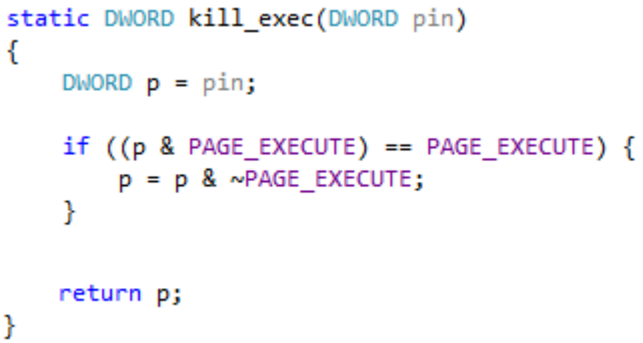
CodeFreeze at Runtime

application binary

DLLs



Freeze Image



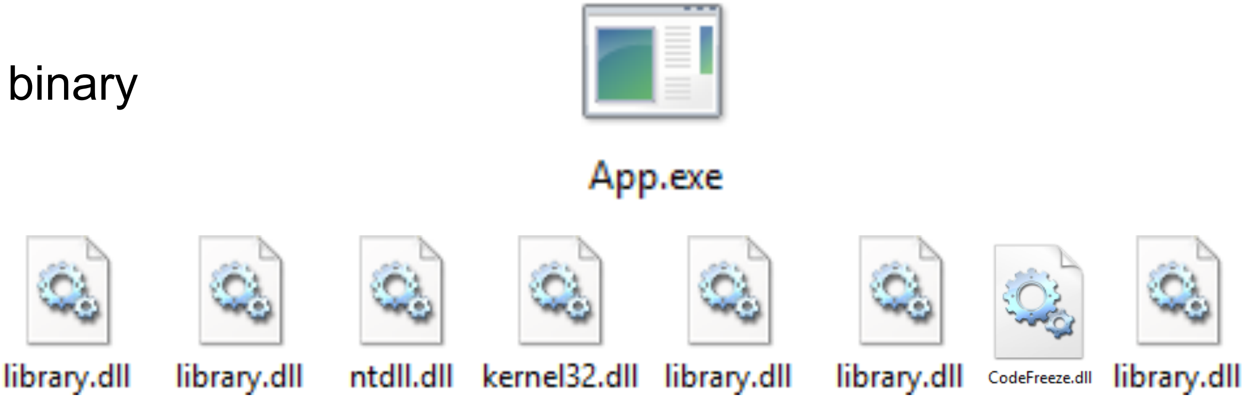
```
static DWORD kill_exec(DWORD pin)
{
    DWORD p = pin;

    if ((p & PAGE_EXECUTE) == PAGE_EXECUTE) {
        p = p & ~PAGE_EXECUTE;
    }

    return p;
}
```

Essentially: API hook that strips EXEC bit

CodeFreeze at Runtime

application binary

DLLs



Execute app code

Application’s entry point is executed

- We could unload CodeFreeze.dll (if fixation was in kernel)

Demo(Server.exe)

- Super very tiny webserver
  - 170 lines of C code!
  - Serves files from C:\WWW

- Simple stack overflow
  - DEP and ASLR are disabled!
    - I did not have the skill or time to bypass those

Demo(Server.exe)

lets go!

DemoServer.exe: Stripped Code

Total

```
15 DLLs
functions: 32040
blocks:    45656
bytes:   6095571 (5M)
```

Stripped

```
functions: 19053
blocks:    25082
bytes:   3530118 (3.4M)
```

DemoServer.exe: Stripped Code

Total

15 DLLs

fun

blo

bytes: 6095571 (5M)

57% code stripped from process memory

Stripped

```
functions: 19053
blocks:    25082
bytes:   3530118 (3.4M)
```

DemoServer.exe: removed functions

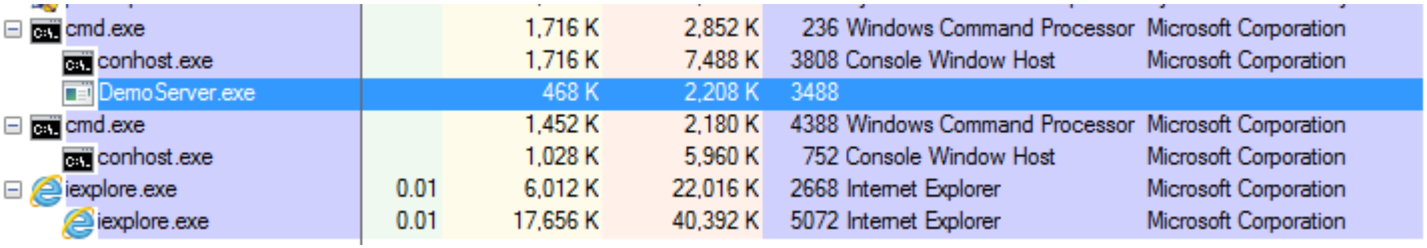
```
kernel32.dll WinExec
kernel32.dll PeekNamedPipe
kernel32.dll GetTempPathA
kernel32.dll GetTempPathW
kernel32.dll CreateFileA
kernel32.dll DeleteFileA
kernel32.dll GetSystemDirectoryA
kernel32.dll GetSystemDirectoryW
kernel32.dll CreateProcessA
kernel32.dll CreateFileMappingA
kernel32.dll CreateFileMappingW
kernel32.dll WaitForSingleObject
```

```
msvcrt.dll fwrite
msvcrt.dll fopen
msvcrt.dll _execv
kernelbase.dll PeekNamedPipe
kernelbase.dll CreateFileA
kernelbase.dll DeleteFileA
kernelbase.dll GetCurrentProcess
kernelbase.dll GetSystemDirectoryA
kernelbase.dll CreateProcessW
kernelbase.dll CreateProcessA
ws2_32.dll ioctlsocket
```

just some of the interesting functions...

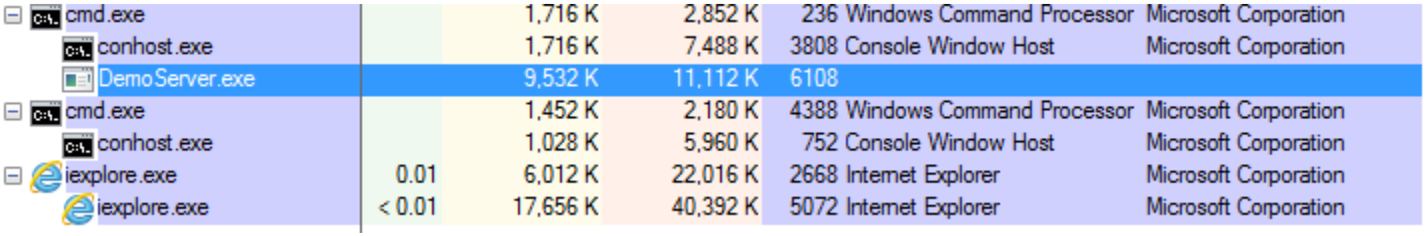
DemoServer.exe: Memory Overhead

Unprotected



cmd.exe		1,716 K	2,852 K	236	Windows Command Processor	Microsoft Corporation
conhost.exe		1,716 K	7,488 K	3808	Console Window Host	Microsoft Corporation
DemoServer.exe		468 K	2,208 K	3488		
cmd.exe		1,452 K	2,180 K	4388	Windows Command Processor	Microsoft Corporation
conhost.exe		1,028 K	5,960 K	752	Console Window Host	Microsoft Corporation
iexplore.exe	0.01	6,012 K	22,016 K	2668	Internet Explorer	Microsoft Corporation
iexplore.exe	0.01	17,656 K	40,392 K	5072	Internet Explorer	Microsoft Corporation

CodeFreeze Protection



cmd.exe		1,716 K	2,852 K	236	Windows Command Processor	Microsoft Corporation
conhost.exe		1,716 K	7,488 K	3808	Console Window Host	Microsoft Corporation
DemoServer.exe		9,532 K	11,112 K	6108		
cmd.exe		1,452 K	2,180 K	4388	Windows Command Processor	Microsoft Corporation
conhost.exe		1,028 K	5,960 K	752	Console Window Host	Microsoft Corporation
iexplore.exe	0.01	6,012 K	22,016 K	2668	Internet Explorer	Microsoft Corporation
iexplore.exe	< 0.01	17,656 K	40,392 K	5072	Internet Explorer	Microsoft Corporation

- We kill on-demand paging
  - Have to load everything in order to overwrite it!

Adobe Reader (AcroRd32.exe)

- Very popular software
  - Known for its “software quality”

- 185 DLLs
  - 20 Adobe DLLs
  - 165 (Windows) system DLLs
  - We only do code stripping on system DLLs

Demo: AcroRd32.exe

let’s CodeFreeze Adobe Reader!

AcroRd32.exe: Stripped Code

Total

165 DLLs

functions:	115892
blocks:	184660
bytes:	22743017 (21M)

Stripped

functions:	35927
blocks:	45132
bytes:	6594960 (6.3M)

AcroRd32.exe: Stripped Code

Total

165 DLLs

fun

blo

byt

28% code stripped from process memory

Stripped

functions:	35927
blocks:	45132
bytes:	6594960 (6.3M)

AcroRd32.exe: removed functions

```
urlmon.dll URLDownloadToFileA
msvcrt.dll fopen
msvcrt.dll _execv
kernelbase.dll CreateProcessA
ws2_32.dll accept
```

just some of the interesting functions...

CodeFreeze Advantages

- Runs before actual application executes
    - Does not add to application code base
    - (Bugs in CodeFreeze are not exploitable)

- Execute with app’s privileges
    - We don’t add code elevated privileges

- **No runtime overhead!**
    - **Performance hit only at application startup**

Current Limitations

- Quality of our Control Flow Graphs
    - We need “perfect” CFGs to do proper stripping

- Windows: no partial unmapping of memory mapped files
    - This would allow us to unmap unused code
    - No memory overhead (not killing on-demand paging)

Future Work: CFG from the Compiler

- CFG recovery sucks!
  - Almost always incomplete

- Have the compiler save the CFG
  - Perfect CFGs

- Requires access to source of Windows DLLs
  - Microsoft could provide JUST the CFGs

Better DLLs for Code Stripping

- Group code/functions inside the DLL
  - Code blocks that belong to the same call chain

- With code on page boundaries, we could:
  - Change page to be NO EXEC
  - Unmap pages
    - Get rid of memory overhead

Conclusions

- Apps contain a large amount of unused code
  - Unused code due to DLLs
  - Code is available to attackers!

- **CodeFreeze** strips unused code
  - Works at runtime, apps don’t need to be modified
  - Disable creation of executable pages
    - Kill malicious library loading and code injection

- CodeFreeze works on real world apps
  - No runtime overhead!

Conclusions

- Apps contain a large amount of unused code
  - Unused code d
  - Cod

Thank you!

- **CodeFreeze** strips unused code
  - Works at runtime, apps don't be modified
  - Disa
    - K jection

Questions?

https://mulliner.org/security/codefreeze/

Thanks

- Cuckoo Sandbox team (various code pieces)
  - specifically: Jurriaan Bremer
- Johannes Kinder (the guy behind Jakstab)
- FX and Joernchen (discussion)
- Various other people (encouragement)