FUZZING THE WINDOWS KERNEL

Yong Chuan Koh

26th August 2016

#whoami

Yong Chuan, Koh (@yongchuank)

- Security Consultant and Researcher
- @ MWR Infosecurity (SG) since 2014
- Interests:
  - Reverse Engineering
  - Vulnerability Research
  - Malware Analysis
- Previous Research
  - "Understanding the Microsoft Office 2013 Protected-View Sandbox"

++

OUTLINE

- Introduction
- Framework Architecture And Components
- Framework Algorithms
- Framework Setup And Configuration
- Results And Case Study
- Conclusion And Future Work

++

FUZZING THE WINDOWS KERNEL

INTRODUCTION

++

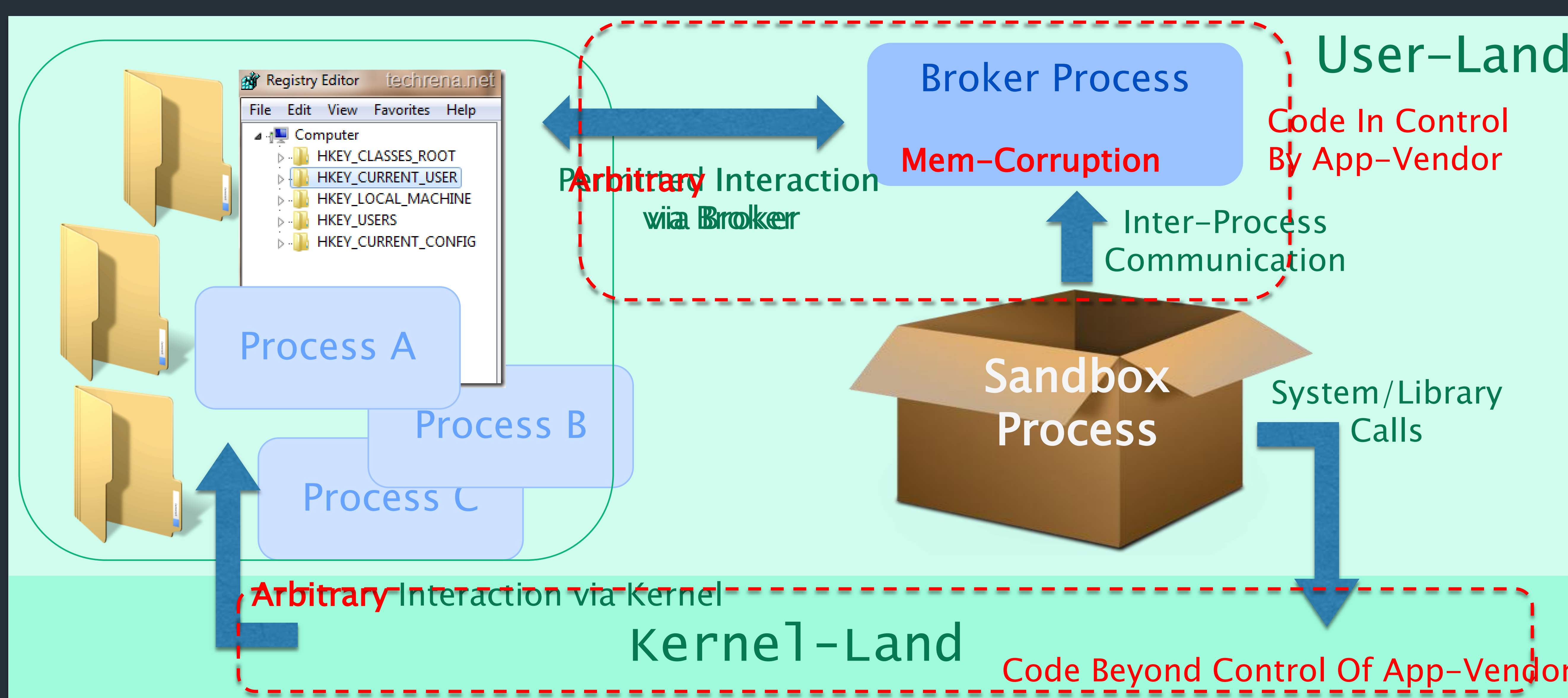
Sandbox

- Sandboxing 101
  - Wikipedia: *"…a sandbox is a security mechanism for separating running programs…A sandbox typically provides a tightly controlled set of resources for guest programs to run in, …A sandbox is implemented by executing the software in a restricted operating system environment, thus controlling the resources (…) that a process may use…"*

- Sandbox aims to contain exploits by limiting damage to system

++
Sandbox Escapes

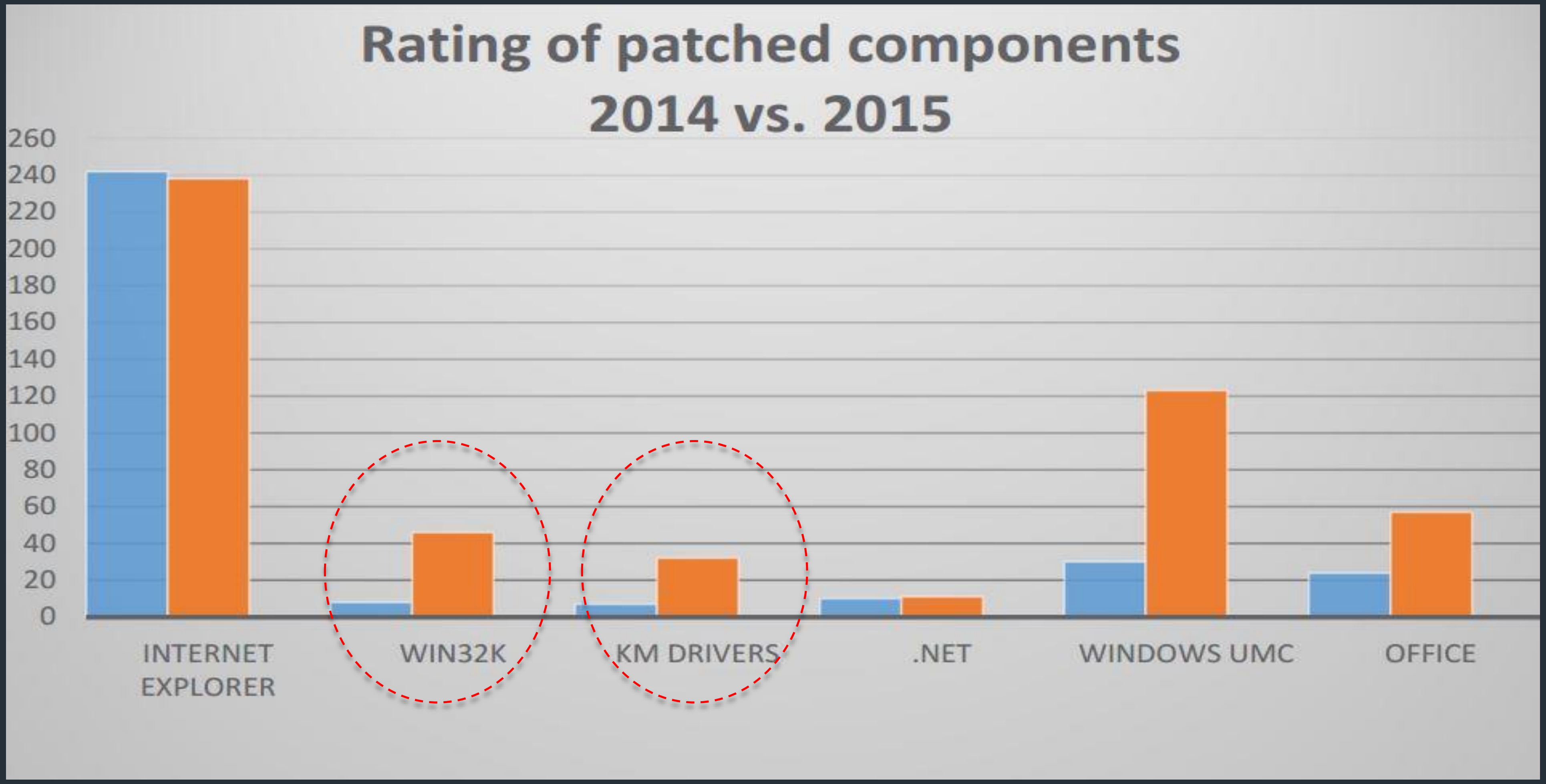
- Maturity of sandbox adoption in popular applications…
  - 2006: Internet Explorer 7 Protected-Mode
  - 2010: Chrome Browser Sandbox
  - 2010: Adobe Reader X Protected Mode
  - 2012: Internet Explorer 10 Enhanced Protected-Mode



LABS

++
Kernel An Easier Target (Really?)

- Pwn2Own Winning Entries
  - 2016: 6 new Kernel vulnerabilities / 7 attempts on Windows targets
  - 2015: 4 new Kernel vulnerabilities / 7 attempts on Windows targets
  - 2014: 1 new Kernel vulnerabilities / 8 attempts on Windows targets
  - 2013: 1 new Kernel vulnerabilities / 8 attempts on Windows targets
- Increased Kernel patches from 2014-2015 (~4X)

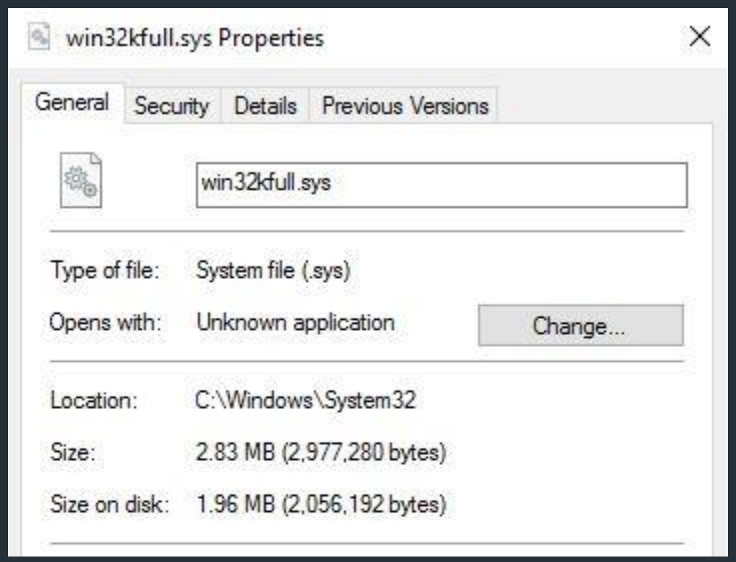


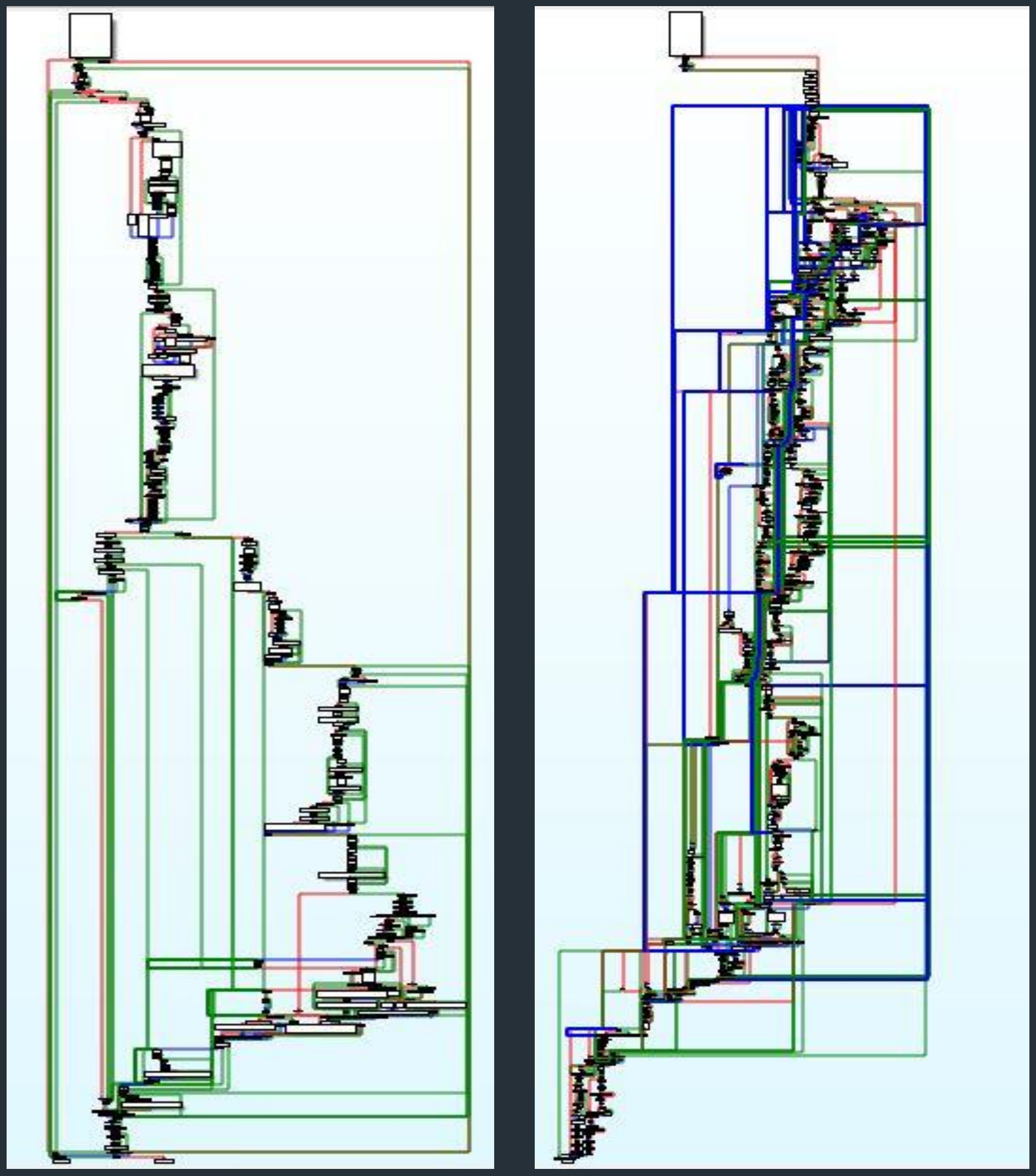
http://www.welivesecurity.com/wp-content/uploads/2016/01/Windows_Exploitation_in_2015.pdf

++

Ok, Kernel is pretty huge…

- Which Kernel component?
  - ~600+ drivers in %WINDIR%\System32
  - Loaded by default, reachable in sandbox
  - Complicated code
  - “Spidey sense”…
- WIN32K.SYS driver
  - 2997280 bytes
  - Complicated
  - Lots of disclosed vulnerabilities already
  - “How bad design decisions created the least secure driver on Windows” by Thomas Garnier

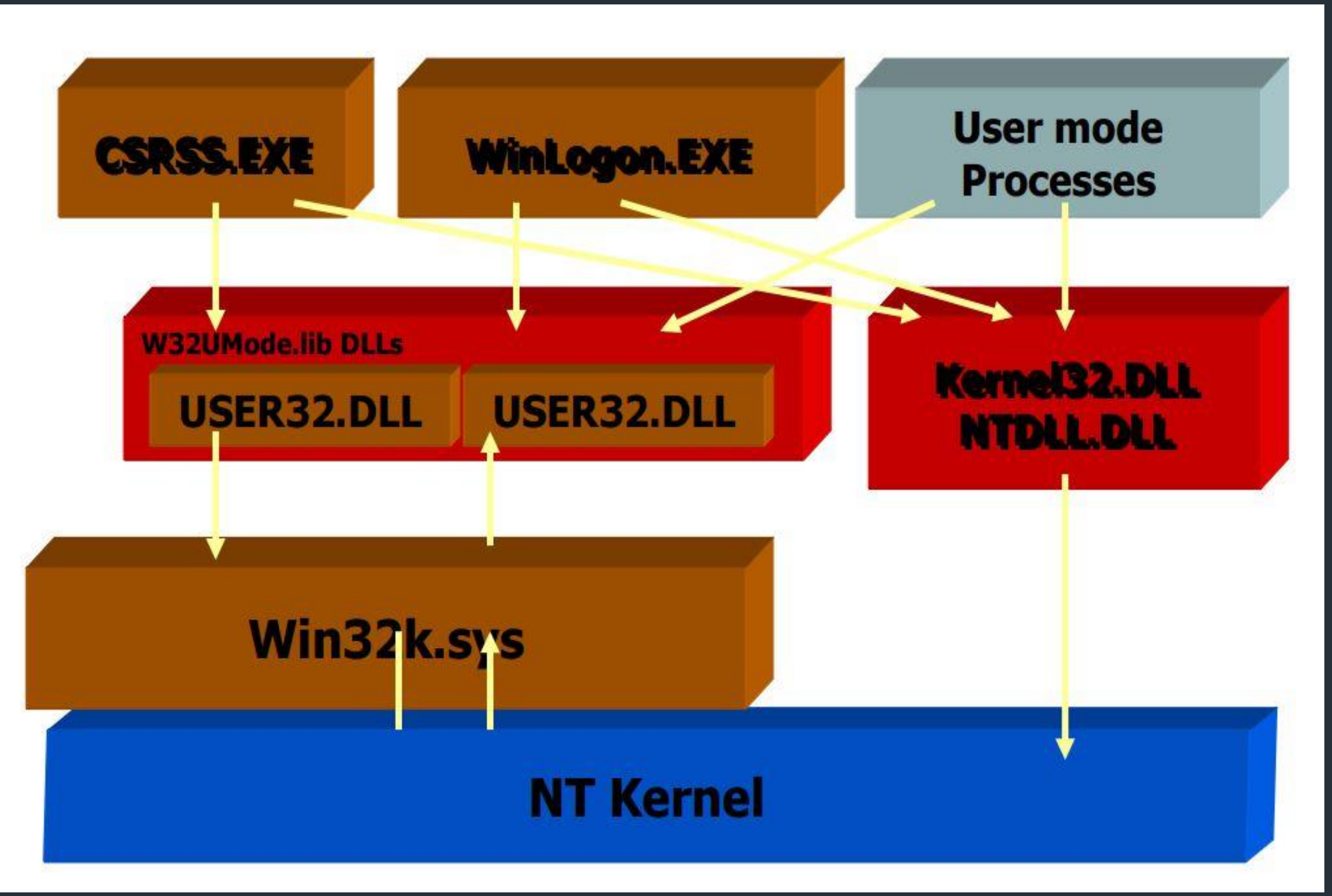




++

WIN32K.SYS Kernel-Mode Driver

- "Windows Kernel Internals: Win32k.sys" by Dave Probert
- Graphical User Interface (GUI) infrastructure of Windows
  - Window Manager (USER)
  - Graphic Device Interface (GDI)
  - Dx thunks to dxg.sys (DirectX)
- W32UMode.lib DLLs
  - USER32.DLL, IMM32.DLL
  - GDI32.DLL, MSIMG32.DLL
  - D3D8THK.DLL



Dave Probert: http://pasotech.altervista.org/windows_internals/Win32KSYS.pdf

++
Goals

- Windows Kernel Fuzzing Framework
  - Easily scalable
  - Reproducible BSOD
  - Modular and adaptable

- Friendly internal competition
  - “Windows Kernel Fuzzing” by Nils
  - “Platform Agnostic Kernel Fuzzing” by James Loureiro and Georgi Geshev
  - Different implementation find different vulnerabilities

- Learning about Windows Kernel security

LABS

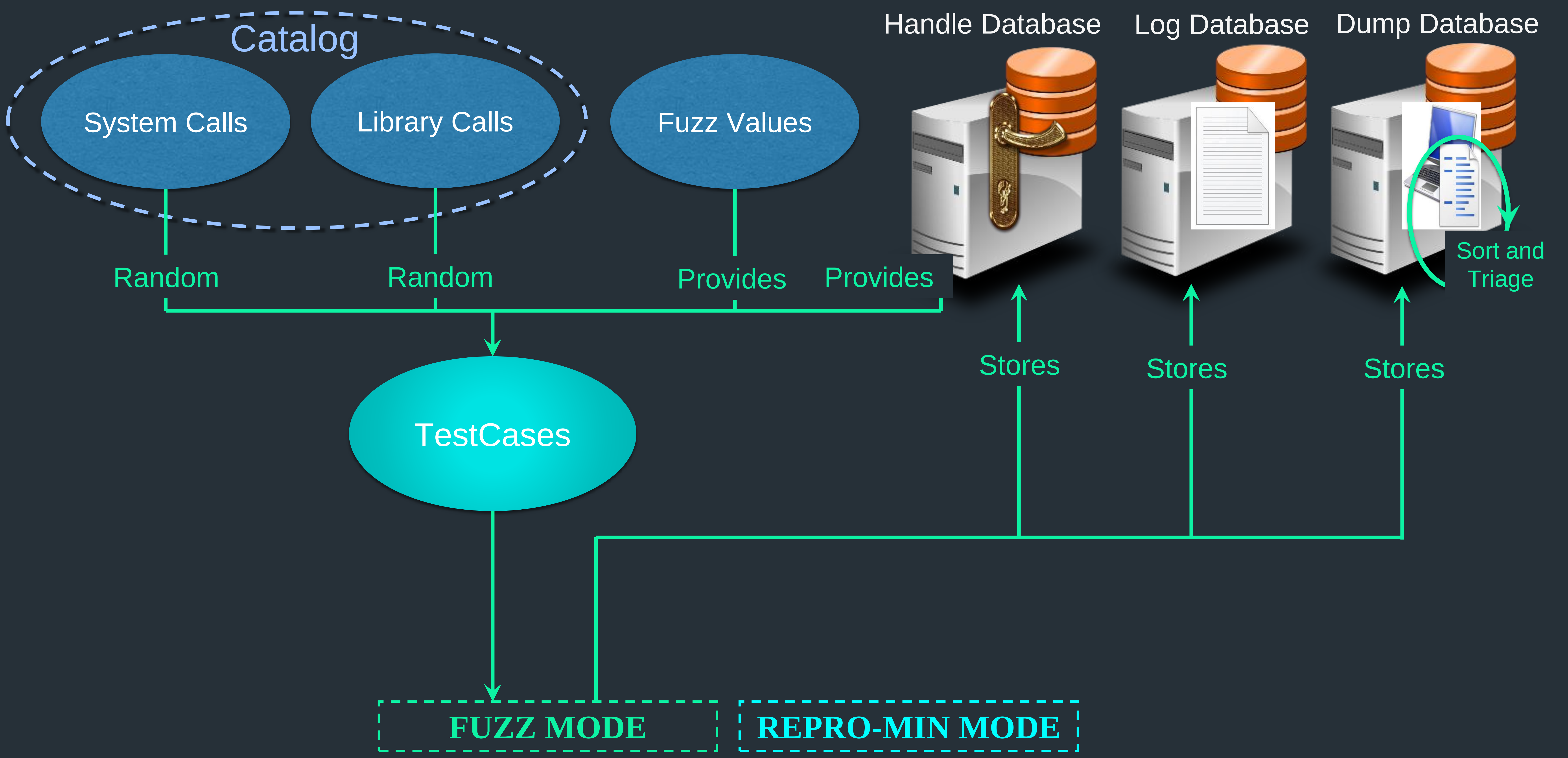
++

Fuzzing the Windows Kernel

- FRAMEWORK ARCHITECTURE AND COMPONENTS

++

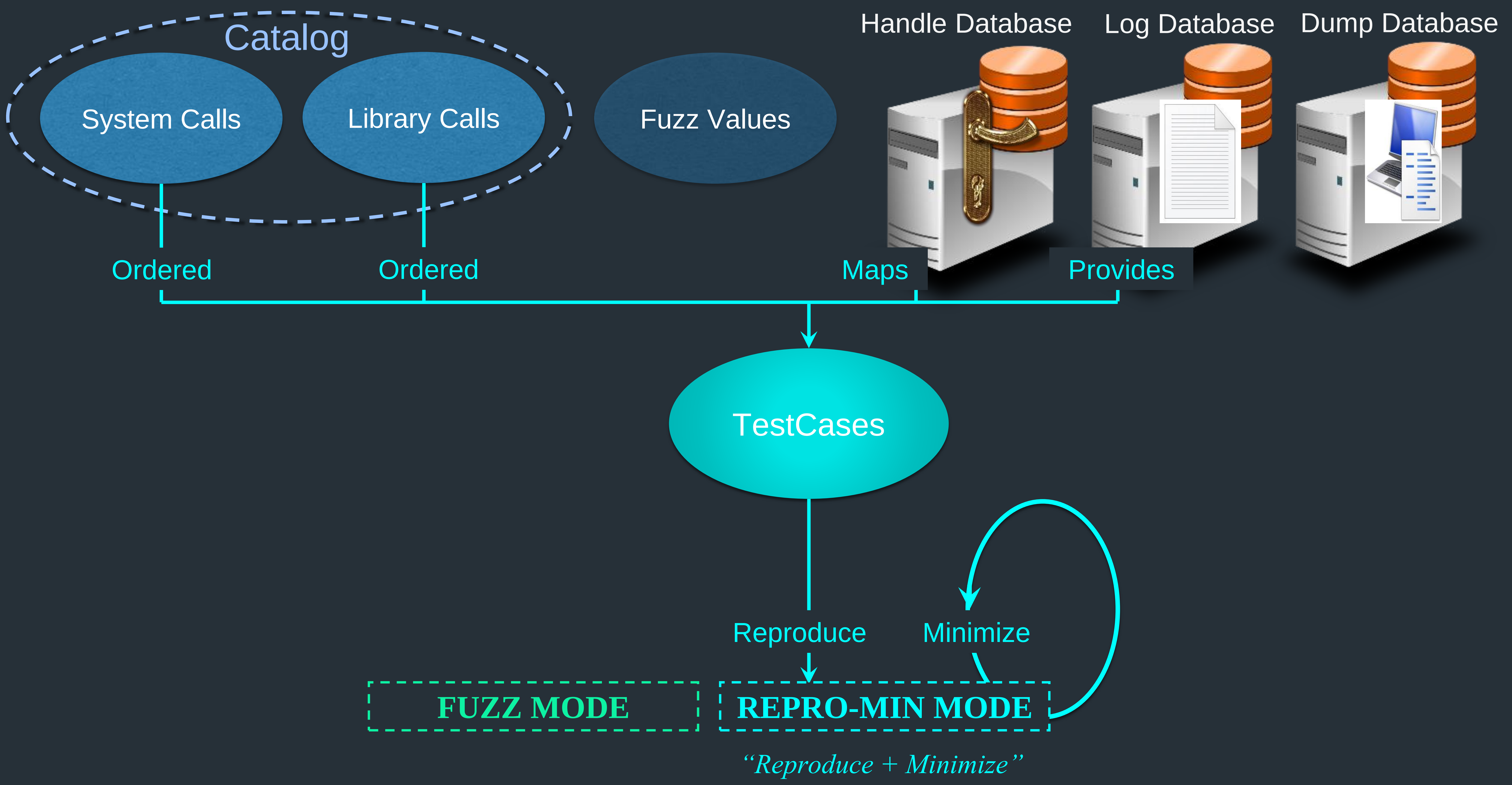
Architecture



LABS

++

Architecture



++

Architecture - Implementation

- Implemented in Python
  - Familiarity and ease ✓
  - Extensive usage of ctypes library for C-compatibility ✓
  - Re-define numerous C function prototypes and structures ✗
- Alternative: C/C++
  - Development and debugging ✗
  - Existing C function prototypes and structures ✓
  - Efficient fuzzing performance ✓

LABS

++

Components – Catalog

- Determine interaction with target Kernel component
  - In this case, fuzzing Win32k.sys with relevant library and system calls
  - Easily repurposed for different Kernel components
- Quality of catalog determines
  - Type of vulnerability class
  - Code coverage

LABS

++

Components – Catalog

- Collection of Library and System call definitions
  - Argument types and values
  - Return values
  - Custom logging syntax rules to bridge Fuzz Mode and Repro-Min Mode
- Purpose of Library calls
  - Wrapper for System calls
  - Introduce more randomness
- Sources for Library and System call definitions
  - MSDN, Headers, ReactOS (thanks!), Google-fu, reverse-engineering

++

Components – Catalog Syntax Rules

- Categorize argument and return types
  - HEX, STRING, HANDLE, STRUCTURE
- Syntax Rule: HEX
  - Integers represented in hexadecimals
  - Signed vs unsigned
  - Byte vs Word vs Dword vs Qword
- Syntax Rule: STRING
  - Pointers to sequence of bytes
  - Arrays, Strings, Pointers to integers, etc

LABS

++

Components – Catalog Syntax Rules

- Syntax Rule: HANDLE
  - Special User-land references to Kernel-land objects
  - Different values between Fuzz Mode and Repro-Min Mode runs
  - Database to store handles to types (Fuzz Mode)
  - Database to provide handles to types (Fuzz Mode)
  - Database to map handle values to creation (Repro-Min Mode)

++

Components – Catalog Syntax Rules

- Syntax Rule: STRUCTURE
  - Combination of HEX, STRING and HANDLE
  - Represented as STRING in itself
  - Can also contain HEX, STRING and HANDLE in fields

LABS

++

Components – Catalog Example 1

Reference from MSDN

```
HBITMAP CreateCompatibleBitmap (
      _In_ HDC hdc,
      _In_ int nWidth,
      _In_ int nHeight
);
```

Catalog Definition

```
class GDI32_CreateCompatibleBitmap (TestCase):
    def generateArguments (self):
        self.hdc          = self.handlearg ("HDC")
        self.nWidth       = self.hexarg (self.GetFuzzValue (“Hex”))
        self.nHeight      = self.hexarg (self.GetFuzzValue (“Hex”))

        self.args.append (self.hdc)
        self.args.append (self.nWidth)
        self.args.append (self.nHeight)
    def runTestCase (self):
        self.handle       = gdi32.CreateCompatibleBitmap (self.args[0], self.args[1], self.args[2])
        self.addhandle ("HBITMAP", self.handle)
```

- Get HDC from HANDLE Database
- Get fuzz HEX values
- Add HITMAP to HANDLE Database

LABS

++

Components – Catalog Example 2

Structure Definition

```
class ACCEL(ctypes.Structure, TestCase):
   _fields_ = [ ("fVirt", BYTE), ("key", WORD), ("cmd", WORD) ]
   def __init__(self, *args, **kwargs):
         setattr(self,   "fVirt",     self.GetFuzzValue (“Hex”))
         setattr(self,   "key",       self.GetFuzzValue (“Hex”))
         setattr(self,   "cmd",       self.GetFuzzValue (“Hex”))
```

Catalog Definition

```
class USER32_CreateAcceleratorTableA (TestCase):
   def generateArguments (self):
      self.cEntries     = self.hexarg (self.GetFuzzValue (“Hex”))   } Get fuzz HEX values
      self.lpaccel      = self.structarg (ACCEL)                     } Get STRUCTURE pointer

      self.args.append (self.lpaccel)
      self.args.append (self.cEntries)
   def runTestCase (self):
      self.handle       = user32.CreateAcceleratorTableW (self.args[0], self.args[1])
      self.addhandle ("HACCEL", self.handle)                         } Add HACCEL to HANDLE Database
```

++

Components – TestCases

- Instances of Library or System calls
  - Catalog definition + Fuzz values + Valid handles
  - Fuzz Mode: randomly selected
  - Repro-Min Mode: ordered according to logs

LABS

++

Components - Databases

- Handle Database
  - Stores valid handles created during run
  - Provides valid handles created during run
  - Maps handle values to creation conditions
- Log Database
  - Stores ordered sequence of testcases, fuzz values and handle values
- Dump Database
  - Stores, sorts and triages BSOD.dmps
  - FAILURE_ID_HASH_STRING and TIMESTAMP

LABS

++

Components – Logging (Fuzz Mode)

- Ordered sequence of testcases
- Arguments (fuzz values and handle values) of testcases
- Return values of testcases
- Log format
  - [thread_name] [module_name] [function_name] [function_arguments]

- Pitfall: Excessive logging!
  - 8MB to 2GB
  - Log offsets on binary template for suitable STRING type
  - Log only handle values on library/system call return

++

Components – Logging (Fuzz Mode) Example

```
......
t0:runTestCase:USER32_SetUserObjectInformationW(…,S[template_bin(0x0:0x40)],H[0x10])
t0:runTestCase:SC_NtGdiSetFontEnumeration(H[0x6])
t0:runTestCase:SC_NtGdiEndDoc(HANDLE[0x1011051])
t0:runTestCase:SC_NtGdiExtCreatePen(…,H[0xD7],H[0x3],S['\xac\x1b\xfag'],…)
t0:runTestCase:handle => 0x1B00016
t0:runTestCase:USER32_OpenInputDesktop(H[0x1],H[0x1],H[0x1FF])
t0:runTestCase:handle => 0x0
t0:runTestCase:GDI32_CancelDC(HANDLE[0x121184C])
t0:runTestCase:SC_NtGdiSelectPen(HANDLE[0x2401073E],HANDLE[0x1B00016])
t0:runTestCase:handle => 0x1B00017
t0:runTestCase:USER32_CreateWindowStationW(H[0x0],H[0x0],H[0x37F],H[0x0])
t0:runTestCase:handle => 0x195C
......
```

Offset into binary template

Actual BYTE values

++

Components – Logging (Repro-Min Mode)

- Tokenize log according to format and catalog syntax
  - [thread_name] [module_name] [function_name] [function_arguments]
  - HEX, STRING, HANDLE, STRUCTURE

++

Components - Logging (Repro-Min Mode) Example

- Assign testcase to corresponding thread…

Fuzz Mode Log

```
……
t0:runTestCase:SC_NtGdiExtCreatePen(…,H[0xFFFFFFEF],H[0xD7],H[0x3],S['\xac\x1b\xfag'],…)
t0:runTestCase:handle => 0x1B00016
t0:runTestCase:SC_NtGdiSelectPen(HANDLE[0x2401073E],HANDLE[0x1B00016])
t0:runTestCase:handle => 0x1B00017
……
```

```
……
t0:
```

Repro-Min Mode Log

LABS

++

Components – Logging (Repro-Min Mode) Example

- Assign testcase context…

Fuzz Mode Log

```
……
t0:runTestCase:SC_NtGdiExtCreatePen(…,H[0xFFFFFFEF],H[0xD7],H[0x3],S['\xac\x1b\xfag'],…)
t0:runTestCase:handle => 0x1B00016
t0:runTestCase:SC_NtGdiSelectPen(HANDLE[0x2401073E],HANDLE[0x1B00016])
t0:runTestCase:handle => 0x1B00017
……
```

```
……
t0:runTestCase:
```

Repro-Min Mode Log

++

Components – Logging (Repro-Min Mode) Example

- Get catalog testcase in ordered sequence...

Fuzz Mode Log

```
......
t0:runTestCase:SC_NtGdiExtCreatePen(…,H[0xFFFFFFEF],H[0xD7],H[0x3],S['\xac\x1b\xfag'],…)
t0:runTestCase:handle => 0x1B00016
t0:runTestCase:SC_NtGdiSelectPen(HANDLE[0x2401073E],HANDLE[0x1B00016])
t0:runTestCase:handle => 0x1B00017
......
```

```
......
t0:runTestCase:SC_NtGdiExtCreatePen
```

Repro-Min Mode Log

++

Components – Logging (Repro-Min Mode) Example

- Run testcase with corresponding arguments...

Fuzz Mode Log

```
……
t0:runTestCase:SC_NtGdiExtCreatePen(…,H[0xFFFFFFEF],H[0xD7],H[0x3],S['\xac\x1b\xfag'],…)
t0:runTestCase:handle => 0x1B00016
t0:runTestCase:SC_NtGdiSelectPen(HANDLE[0x2401073E],HANDLE[0x1B00016])
t0:runTestCase:handle => 0x1B00017
……
```

```
……
t0:runTestCase:SC_NtGdiExtCreatePen(…,H[0xFFFFFFEF],H[0xD7],H[0x3],S['\xac\x1b\xfag'],…)
```

Repro-Min Mode Log

LABS

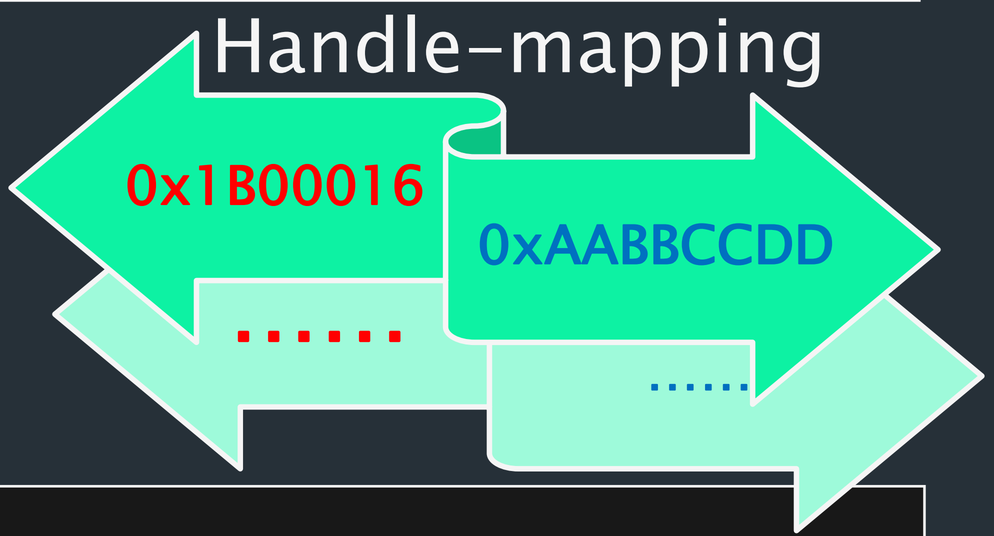
++

Components – Logging (Repro-Min Mode) Example

- Map handles in database…

Fuzz Mode Log

```
……
t0:runTestCase:SC_NtGdiExtCreatePen(…,H[0xFFFFFFEF],H[0xD7],H[0x3],S['\xac\x1b\xfag'],…)
t0:runTestCase:handle => 0x1B00016
t0:runTestCase:SC_NtGdiSelectPen(HANDLE[0x2401073E],HANDLE[0x1B00016])
t0:runTestCase:handle => 0x1B00017
……
```




```
……
t0:runTestCase:SC_NtGdiExtCreatePen(…,H[0xFFFFFFEF],H[0xD7],H[0x3],S['\xac\x1b\xfag'],…)
t0:runTestCase:handle => 0xAABBCCDD
```

Repro-Min Mode Log

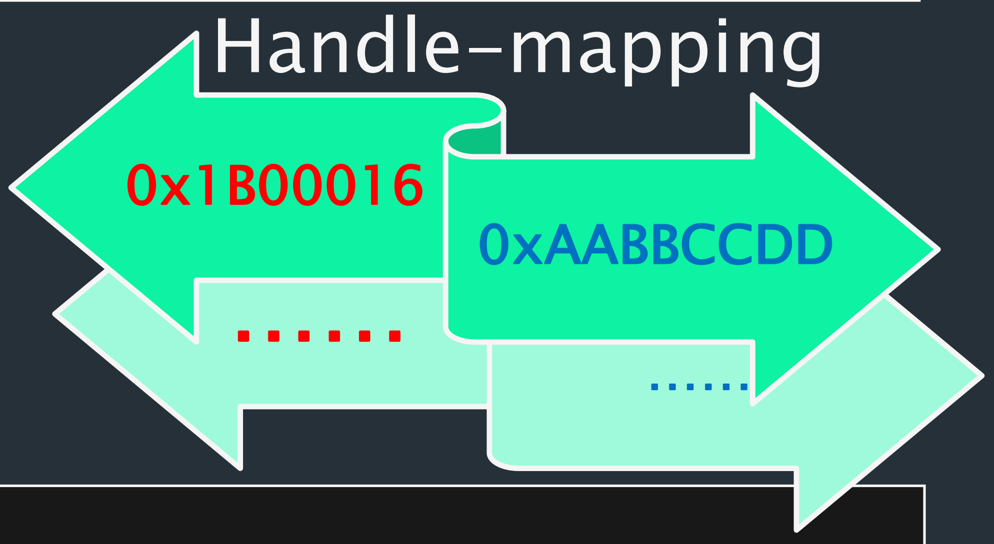
++

Components – Logging (Repro-Min Mode) Example

- Map handles in database…

Fuzz Mode Log

```
……
t0:runTestCase:SC_NtGdiExtCreatePen(…,H[0xFFFFFFEF],H[0xD7],H[0x3],S['\xac\x1b\xfag'],…)
t0:runTestCase:handle => 0x1B00016
t0:runTestCase:SC_NtGdiSelectPen(HANDLE[0x2401073E],HANDLE[0x1B00016])
t0:runTestCase:handle => 0x1B00017
……
```




```
……
t0:runTestCase:SC_NtGdiExtCreatePen(…,H[0xFFFFFFEF],H[0xD7],H[0x3],S['\xac\x1b\xfag'],…)
t0:runTestCase:handle => 0xAABBCCDD
t0:runTestCase:SC_NtGdiSelectPen(HANDLE[0x2401073E],HANDLE[0xAABBCCDD])
```

Repro-Min Mode Log

LABS

++

Fuzzing the Windows Kernel

- FRAMEWORK ALGORITHMS

LABS

++

Fuzz Mode

1. Select library/system call from catalog
   a. Specific selection of testcases that create handles (“Trinity fuzzer”)
   b. Random selection of testcases
2. Generate testcase arguments
3. Log testcase arguments
4. Run testcase
5. Log result
6. Repeat step 1

```
def runTestCase(self, testcase):
        f = testcase
        f.generateArguments()
        arguments = f.serializearguments()
        testcases_log.info("%s(%s)"%(test_name, arguments))
        f.runTest()
        if hasattr(f, "handle") : testcases_log.info("handle => 0x%X"%(f.handle))
```

LABS

++

Repro-Min Mode

- No. of lines in typical logs: 15000 to 250000
- "setup group of testcases" vs "fuzzing group of testcases"

1. Generate set of "fuzzing group of testcases" (N)
2. Divide N into blocks (N/M)
3. Remove one block of testcases
4. Remove unreferenced "setup group of testcases"
5. Run all remaining blocks and check BSOD
6. Repeat step 2 until N/M=1

++

Fuzzing the Windows Kernel

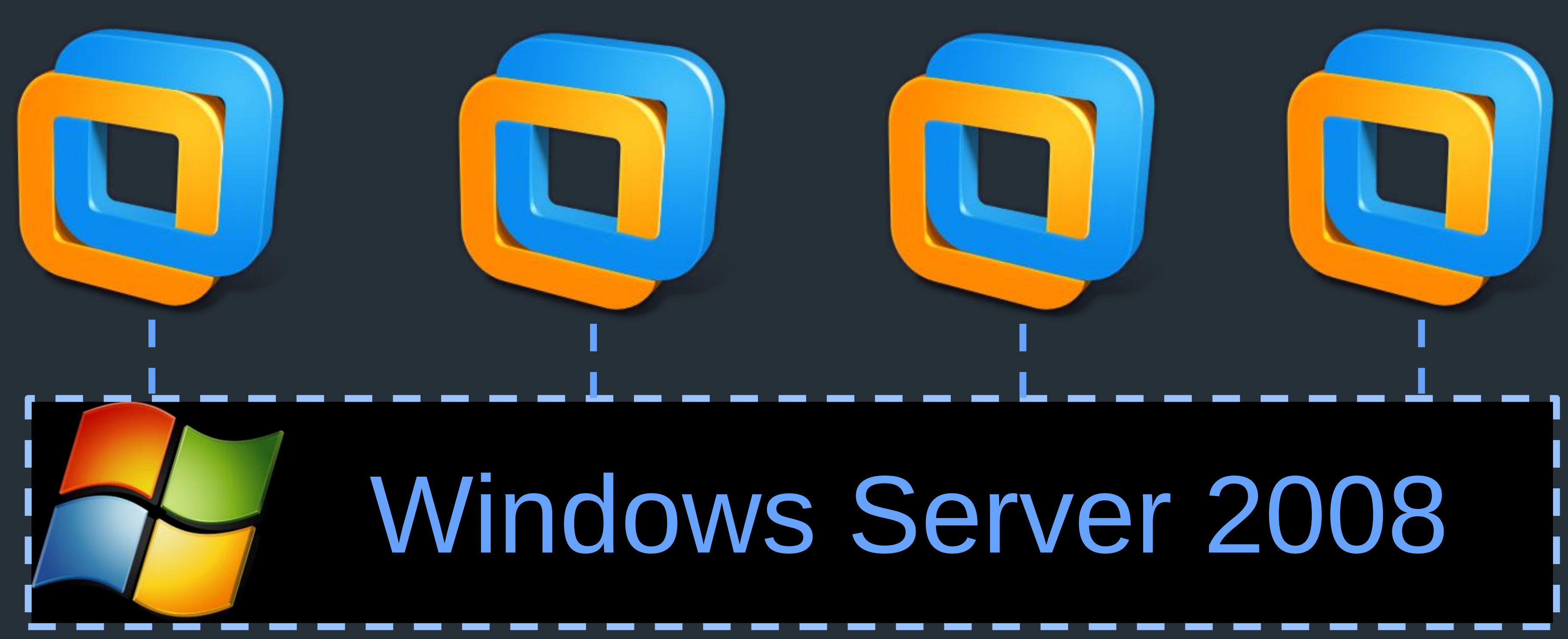
- FRAMEWORK SETUP AND CONFIGURATION

++

Setup And Configuration - Host

- Most basic hardware!
  - Spare machine laying around. Definitely can do better……☺

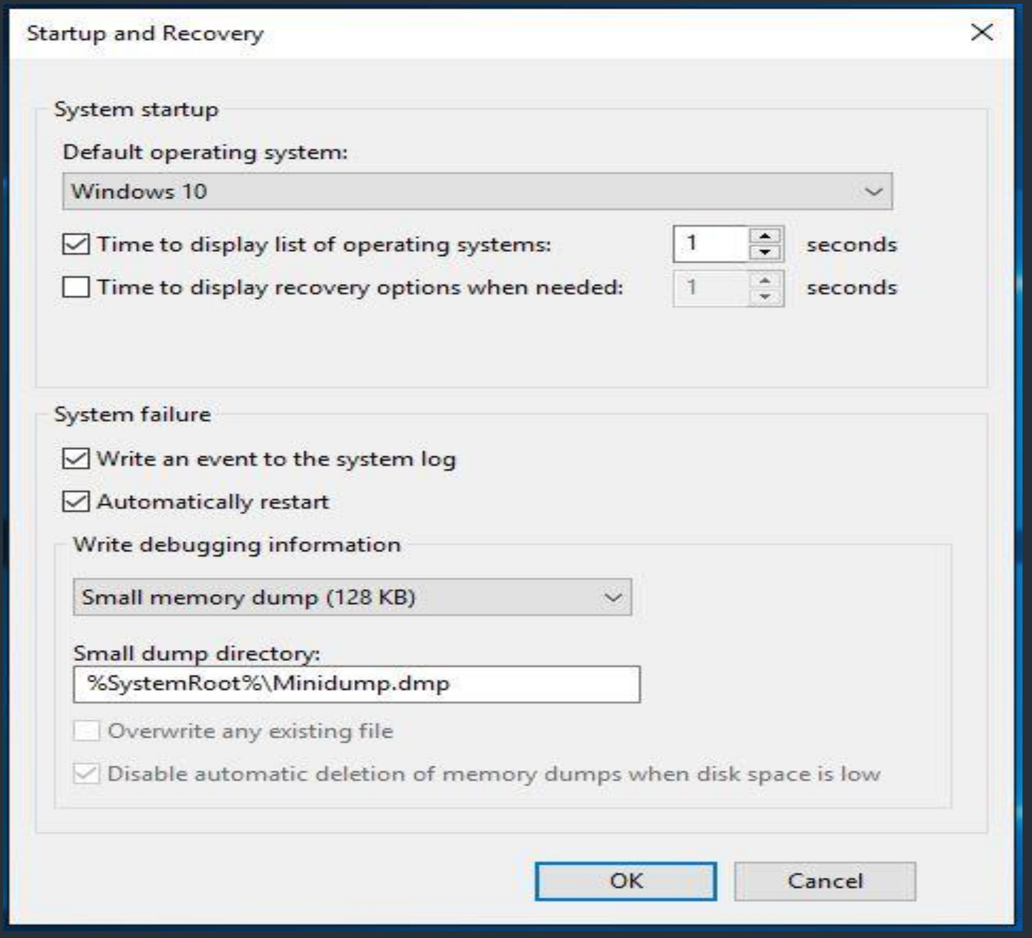
- Intel Xeon X3450, QuadCore @2.67 GHz
- 16 GB RAM
- Windows Server 2008 (x64) Standard



++

Setup And Configuration - Guest

- 1 CPU, 2 GB RAM
- Windows 10 (x86) Pro
- Enable special pool for WIN32K.SYS
  - *"verifier.exe /flags 0x1 /driver win32k.sys"*
- Set BSOD MiniDump for disk-space saving
- Mapped drive to Host for MiniDumps and Logs
- Set normal Windows reboot
  - *"bcdedit /set bootstatuspolicy IgnoreAllFailures"*



++

Setup And Configuration - Scaling Up

- Guest is designed to be as self-contained as possible
- Effectively scaling up means spinning more Virtual-Machines
  - Use cloud
  - Buy more hardware ($$$)
- Need a Server-Client model to store MiniDumps and Logs centrally

++

Fuzzing the Windows Kernel

- RESULTS AND CASE STUDY

++
Results

Test Period	Jan 2016 – Mar 2016	
Total BSOD	10	
DRIVER_PAGE_FAULT_IN_FREED_SPECIAL_POOL (D5)	Use-After-Free	3
PAGE_FAULT_IN_NONPAGED_AREA (50)	Invalid Read	1
KMODE_EXCEPTION_NOT_HANDLED (1E)	Null Dereference	4
IRQL_NOT_LESS_OR_EQUAL (0A)	Miscellaneous	1
APC_INDEX_MISMATCH (01)	Miscellaneous	1

LABS

++

Case Study – MiniDump.dmp

```
3: kd> !analyze -v
*******************************************************************************
*                                                                             *
*                        Bugcheck Analysis                                    *
*                                                                             *
*******************************************************************************

DRIVER_PAGE_FAULT_IN_FREED_SPECIAL_POOL (d5)
Memory was referenced after it was freed.
This cannot be protected by try-except.
When possible, the guilty driver's name (Unicode string) is printed on
the bugcheck screen and saved in KiBugCheckDriver.
Arguments:
Arg1: b853ad9c, memory referenced
Arg2: 00000000, value 0 = read operation, 1 = write operation
Arg3: 9262db7b, if non-zero, the address which referenced memory.
Arg4: 00000000, (reserved)

Debugging Details:
------------------


READ_ADDRESS: GetPointerFromAddress: unable to read from 00000000
GetPointerFromAddress: unable to read from 00000000
unable to get nt!MmSpecialPoolStart
unable to get nt!MmSpecialPoolEnd
unable to get nt!MmPagedPoolEnd
unable to get nt!MmNonPagedPoolStart
unable to get nt!MmSizeOfNonPagedPoolInBytes
 b853ad9c

FAULTING_IP:
win32kfull!DEVLOCKBLTOBJ::~DEVLOCKBLTOBJ+13b
9262db7b 8b4014          mov     eax,dword ptr [eax+14h]

MM_INTERNAL_CODE:  0

IMAGE_NAME:  win32kfull.sys

DEBUG_FLR_IMAGE_TIMESTAMP:  5699d1c7

MODULE_NAME: win32kfull

FAULTING_MODULE: 92600000 win32kfull

DEFAULT_BUCKET_ID:  WIN8_DRIVER_FAULT

BUGCHECK_STR:  0xD5

PROCESS_NAME:  python.exe

CURRENT_IRQL:  2

ANALYSIS_VERSION: 6.3.9600.17237 (debuggers(dbg).14071
```

Use-After-Free BSOD…

… reading from freed pool
in DEVLOCKBLTOBJ destructor

```
TRAP_FRAME:  b6fd294c -- (.trap 0xffffffffb6fd294c)
ErrCode = 00000000
eax=b853ad88 ebx=b6fd2a38 ecx=b80f6718 edx=00000000 esi=b6fd2a4c edi=916f78f0
eip=9262db7b esp=b6fd29c0 ebp=b6fd29f0 iopl=0         nv up ei ng nz na pe nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00010386
win32kfull!DEVLOCKBLTOBJ::~DEVLOCKBLTOBJ+0x13b:
9262db7b 8b4014          mov     eax,dword ptr [eax+14h] ds:0023:b853ad9c=????????
Resetting default scope

LAST_CONTROL_TRANSFER:  from 819f2b0d to 81978f04

STACK_TEXT:
b6fd234c 819f2b0d 00000003 f1a638a6 00000065 nt!RtlpBreakWithStatusInstruction
b6fd23a0 819f25ed 8794a340 b6fd27a4 b6fd2810 nt!KiBugCheckDebugBreak+0x1f
b6fd2778 81977d42 00000050 b853ad9c 00000000 nt!KeBugCheck2+0x742
b6fd279c 81977c79 00000050 b853ad9c 00000000 nt!KiBugCheck2+0xc6
b6fd27bc 819bfbb6 00000050 b853ad9c 00000000 nt!KeBugCheckEx+0x19
b6fd2810 81913956 b853ad9c 81913956 b6fd294c nt! ?? ::FNODOBFM::`string'+0x31544
b6fd28a8 81989aec 00000000 b853ad9c 00000000 nt!MmAccessFault+0x4e6
b6fd28a8 9262db7b 00000000 b853ad9c 00000000 nt!KiTrap0E+0xec
b6fd29f0 92601376 232106b2 0b6ef58c 92901b44 win32kfull!DEVLOCKBLTOBJ::~DEVLOCKBLTOBJ+0x13b
b6fd2b5c 9260109b 170106a6 fffffe68 7fffffae win32kfull!GrePlgBlt+0x2c4
b6fd2be0 819864e7 232106b2 08d3f558 170106a6 win32kfull!NtGdiPlgBlt+0x89
b6fd2be0 77031230 232106b2 08d3f558 170106a6 nt!KiSystemServicePostCall
0b6ef554 74c3b50a 74c72bc0 232106b2 08d3f558 ntdll!KiFastSystemCallRet
0b6ef558 74c72bc0 232106b2 08d3f558 170106a6 GDI32!NtGdiPlgBlt+0xa
0b6ef598 1d1addda 232106b2 08d3f558 170106a6 GDI32!PlgBlt+0xe0
WARNING: Stack unwind information not available. Following frames may be wrong.
0b6ef5cc 1d1acae6 1d1ac930 0b6ef5ec 00000
0b6ef5fc 1d1a8de8 74c72ae0 0b6ef730 27cfd
0b6ef6ac 1d1a95ce 00001100 74c72ae0 0b6ef
0b6ef818 1d1a54d8 74c72ae0 0547edf8 00000
0b6ef870 1e07cdec 00000000 0547edf8 00000000 _ctypes+0x54d8
00000000 00000000 00000000 00000000 00000000 python27!PyObject_Call+0x4c


STACK_COMMAND:  kb

FOLLOWUP_IP:
win32kfull!DEVLOCKBLTOBJ::~DEVLOCKBLTOBJ+13b
9262db7b 8b4014          mov     eax,dword ptr [eax+14h]

SYMBOL_STACK_INDEX:  8

SYMBOL_NAME:  win32kfull!DEVLOCKBLTOBJ::~DEVLOCKBLTOBJ+13b
```

For now, rem B853AD88 addr

BSOD due to library call gdi32.PlgBlt()

"..Bit-block transfer of bits of color data from specified rectangle in hdcSrc to specified parallelogram in hdcDest.."

```
BOOL PlgBlt(
   _In_    HDC     hdcDest,          //Handle to destination DC
   …,
   _In_    HDC     hdcSrc,           //Handle to source DC
   …,
   _In_    HBITMAP hbmMask,          //(Optional) Handle to monochrome bitmap for color mask
   …);
```

Reference from MSDN

LABS

++
Case Study – Repro-Min

- Patched in MS16-062 (May 2016)
  - Bug Collision with one of these…
    - CVE-2016-0171 (Nils [bytegeist])
    - CVE-2016-0173 (Nils [bytegeist])
    - CVE-2016-0174 (Liang Yin [Tencent])
    - CVE-2016-0196 (Dhanesh [FireEye]; Vulcan Team [Qihoo 360])

- Reproduced and minimized after ~120 iterations
  - 14888 lines to 9 lines

- Analysis for this case study is performed on
  - win32kfull.sys v10.0.10586.71
  - win32kbase.sys v10.0.10586.20

LABS

++

Case Study – Repro-Min

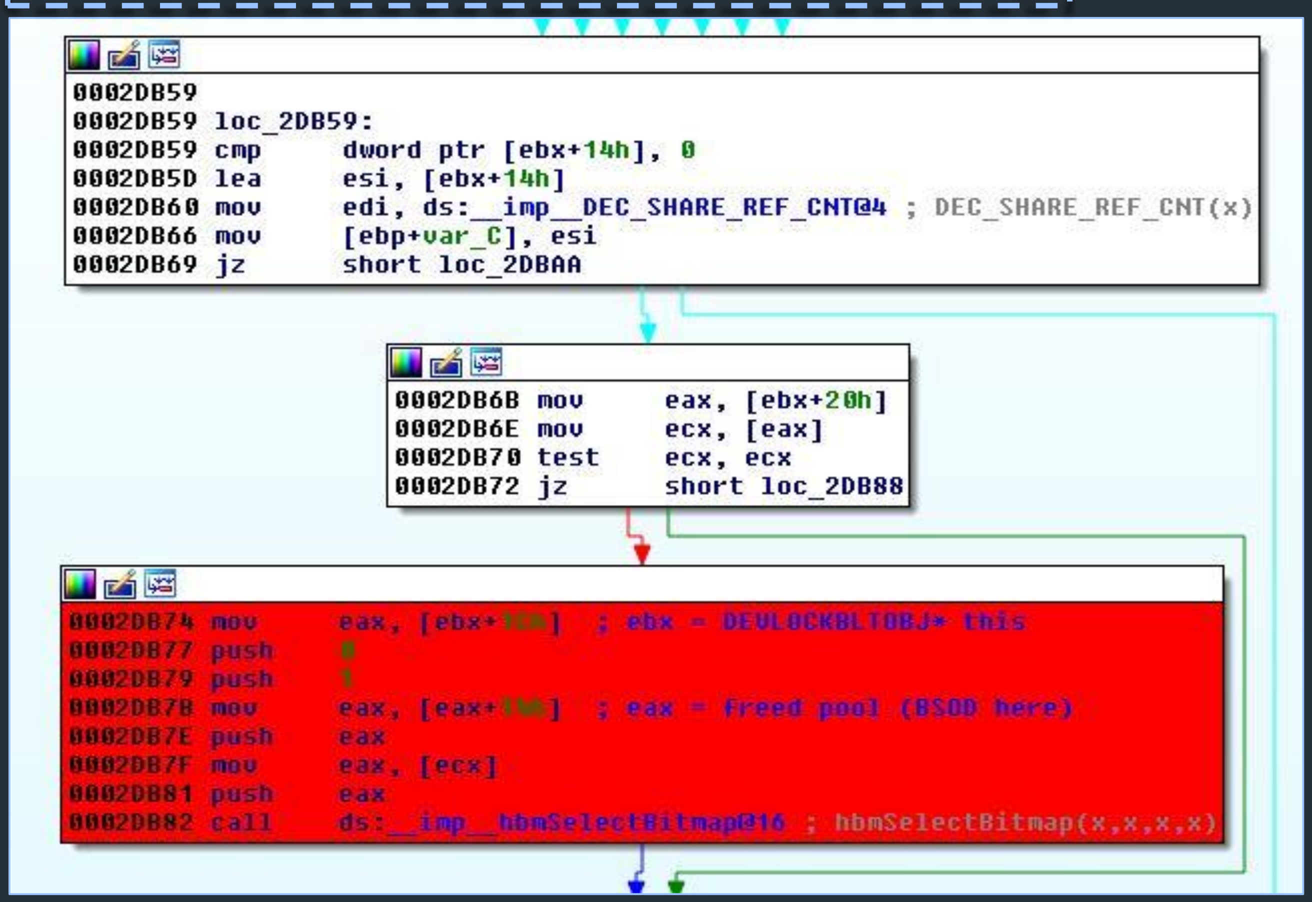
```
t0:runTestCase:GDI32_CreateICA(…)
t0:runTestCase:handle => 0x52109D0
t0:runTestCase:SC_NtGdiCreateMetafileDC(HANDLE[0x52109D0])
t0:runTestCase:handle => 0x22109D3
t0:runTestCase:SC_NtGdiCreateCompatibleBitmap(HANDLE[0x22109D3],…)
t0:runTestCase:handle => 0x60509D5
t0:runTestCase:GDI32_CreateICA(…)
t0:runTestCase:handle => 0x60109D4
t0:runTestCase:GDI32_CreateCompatibleDC(HANDLE[0x60109D4])
t0:runTestCase:handle => 0x40109DB
t0:runTestCase:GDI32_CreateICA(…)
t0:runTestCase:handle => 0x22109E7
t0:runTestCase:SC_NtGdiSelectBitmap(HANDLE[0x40109DB],HANDLE[0x60509D5])
t0:runTestCase:handle => 0x185000F
t0:runTestCase:SC_NtGdiDeleteObjectApp(HANDLE[0x60509D5])
t0:runTestCase:GDI32_PlgBlt(HANDLE[0x22109E7],…,HANDLE[0x40109DB],…,HANDLE[0x185000F],…)
```

Green: Handles related to hdcDest
Red : Handles related to hdcSrc
Blue : Handles related to hbmMask

++

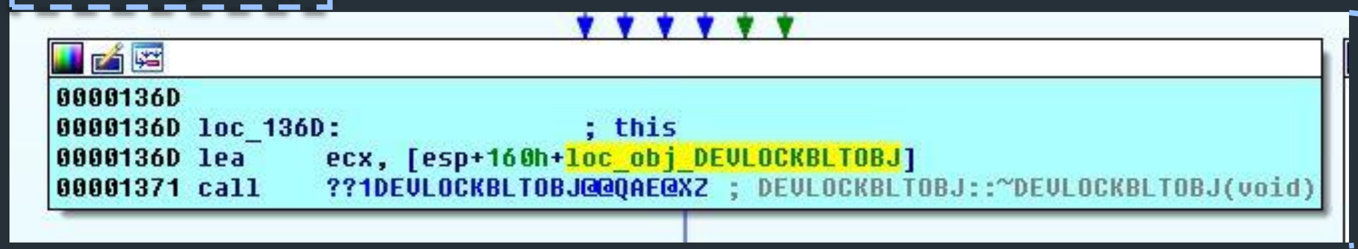
Case Study – Analysis

DEVLOCKBLTOBJ::~DEVLOCKBLTOBJ()



DEVLOCKBLTOBJ::~DEVLOCKBLTOBJ() is called from GrePlgBlt()…

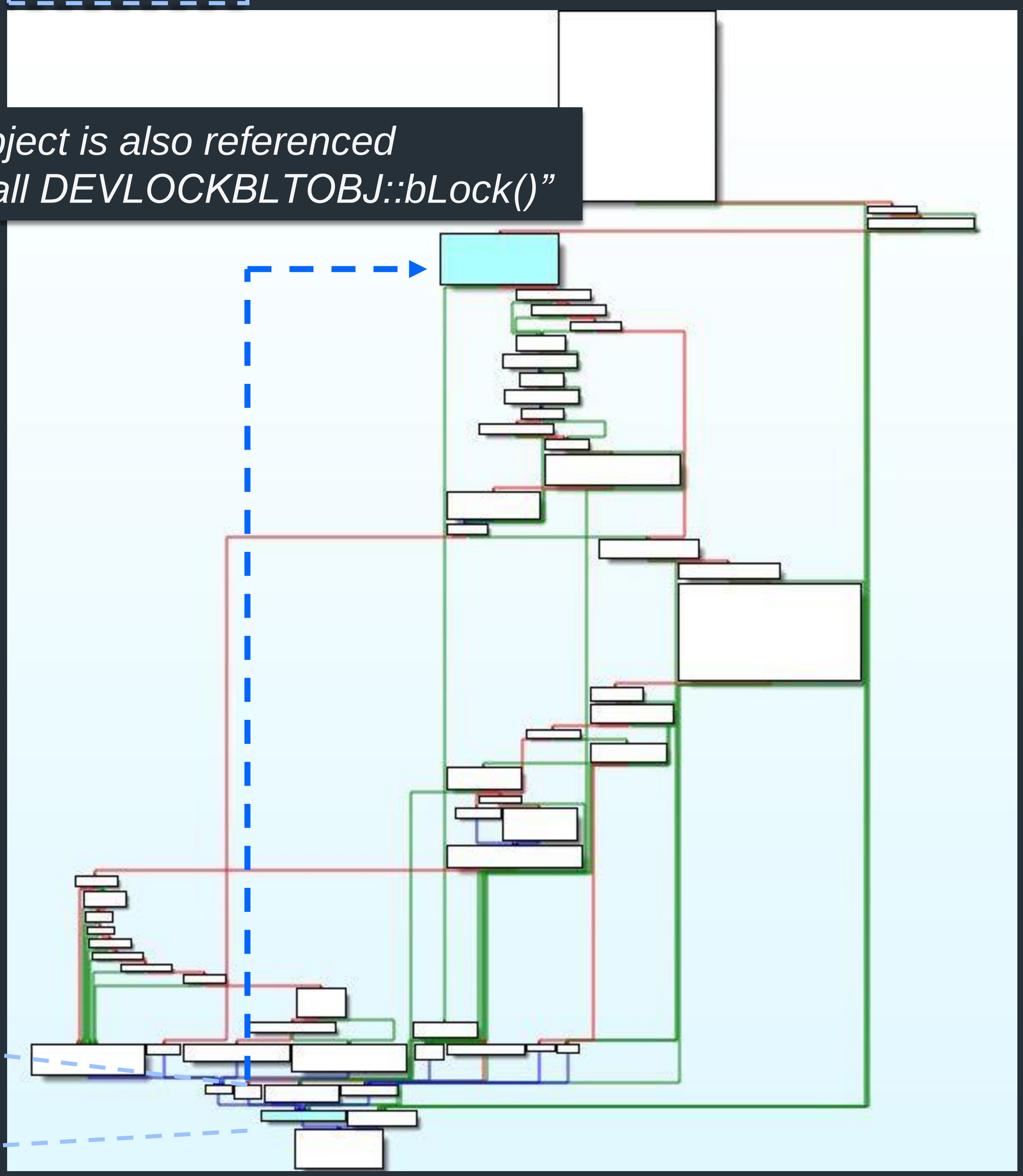
GrePlgBlt()



Enlarge of code-block…

GrePlgBlt()

DEVLOCKBLTOBJ object is also referenced in this code block…"call DEVLOCKBLTOBJ::bLock()"

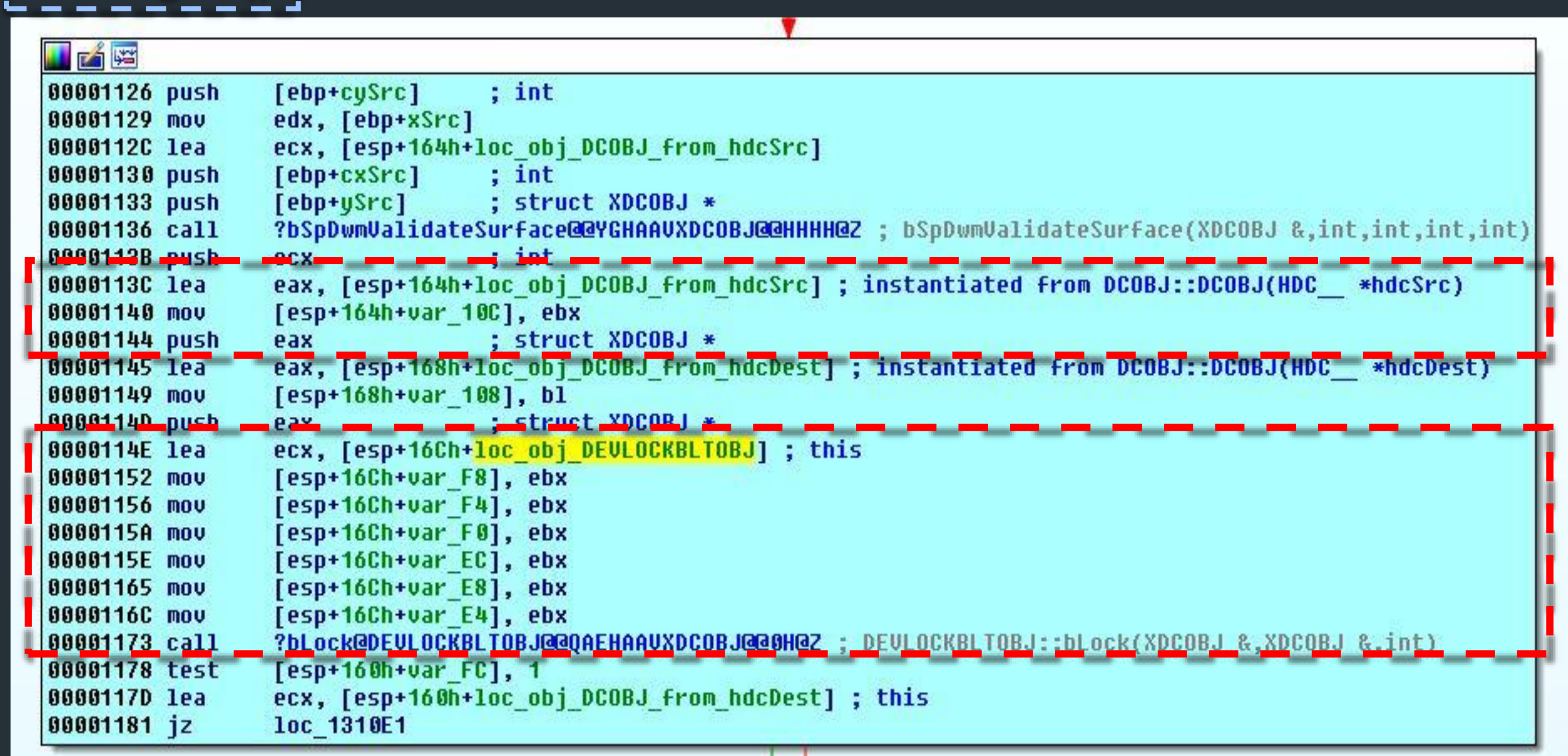


LABS

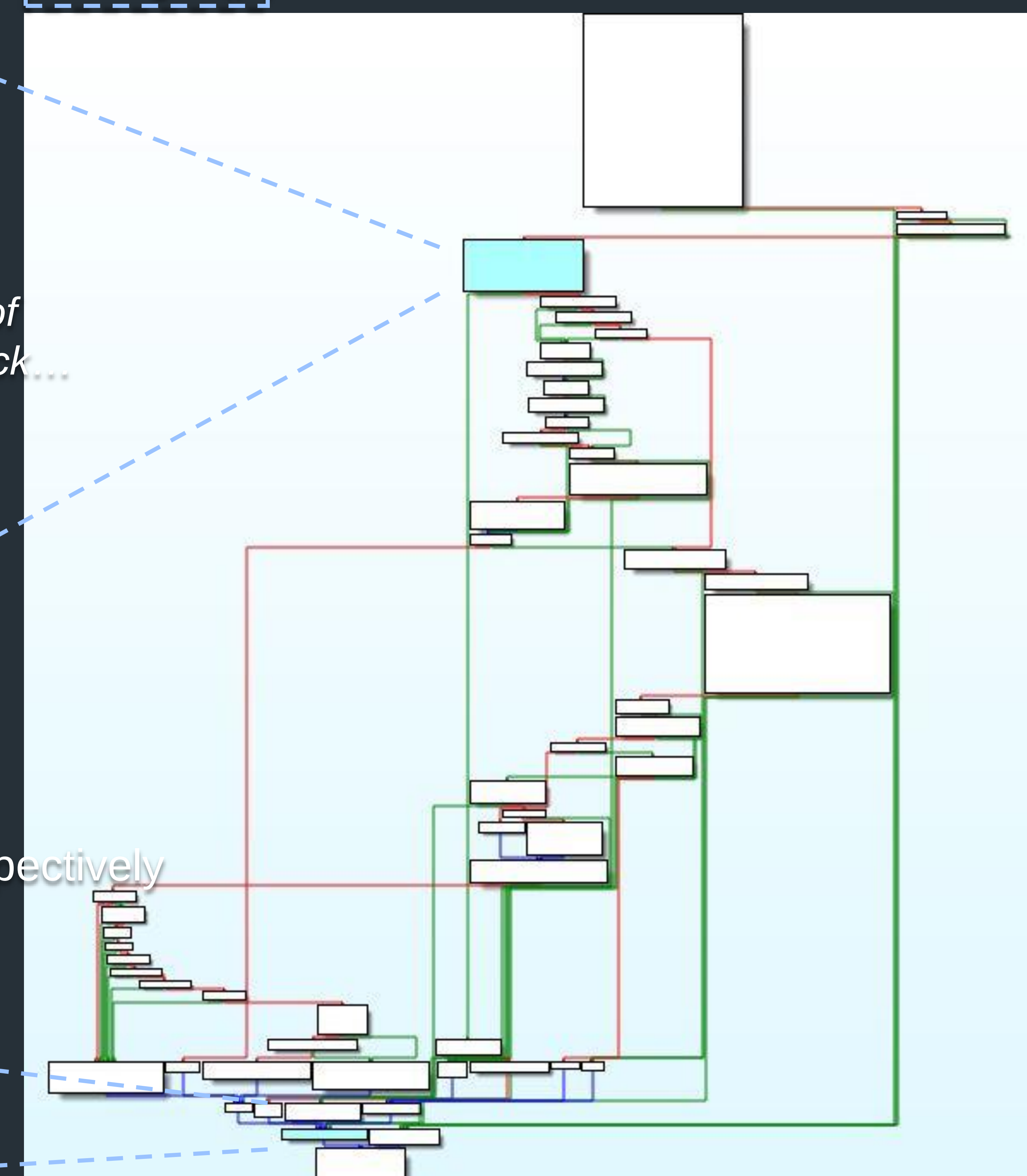
++

Case Study – Analysis

GrePlgBlt()



GrePlgBlt()

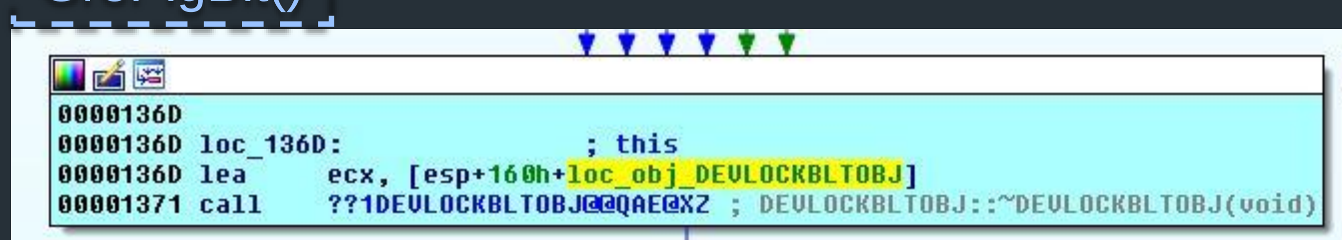


Enlarge of code-block…

Observations:

1. DEVLOCKBLTOBJ is a local variable
2. Referenced in DEVLOCKBLTOBJ::bLock() without prior initialization
3. DCOBJ of hdcDest and hdcSrc are passed as 1st and 2nd arguments respectively

GrePlgBlt()



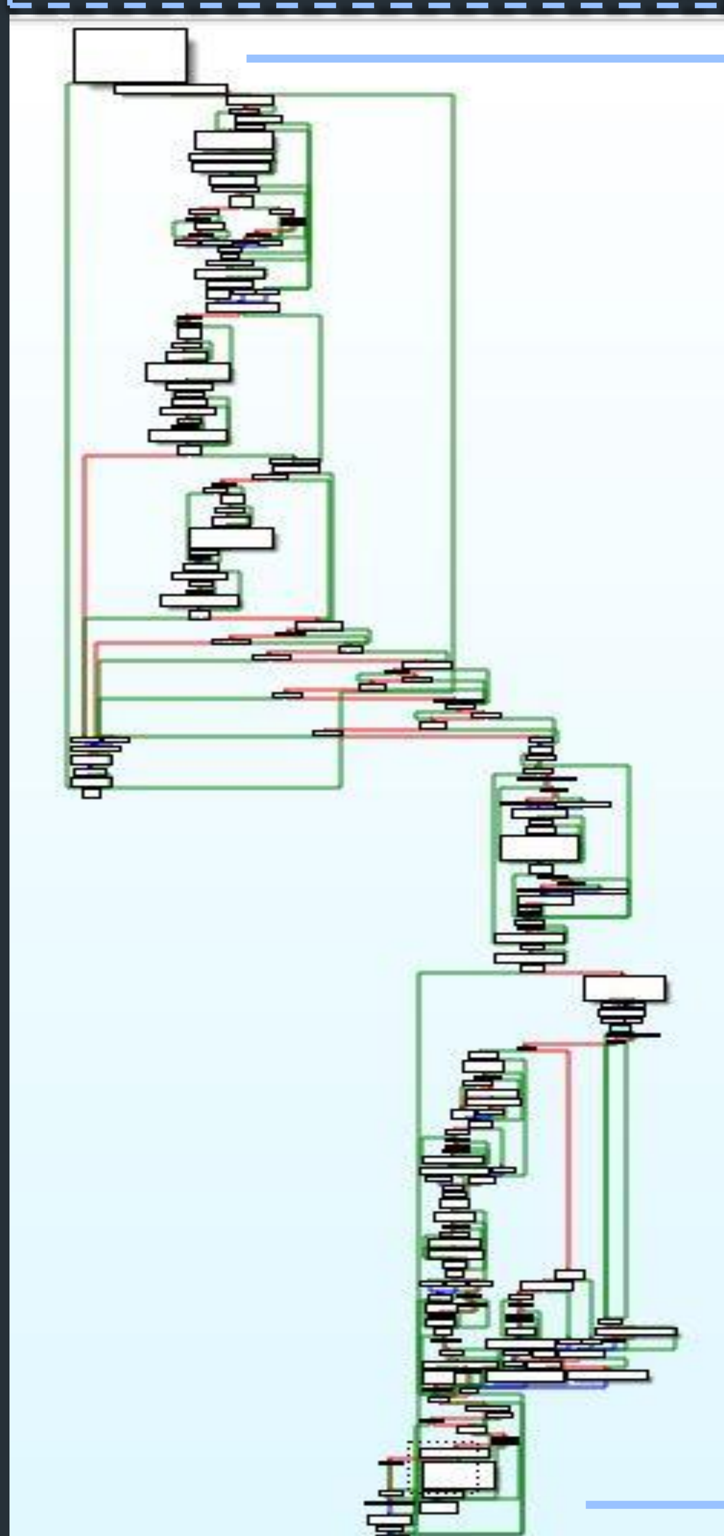
Enlarge of code-block…

LABS

++

Case Study – Analysis

DEVLOCKBLTOBJ::bLock()



[Set logging breakpoint] :

1: kd> bp nt!ExFreePoolWithTag ".printf |"[ExFreePoolWithTag] P %08X, Tag %08X---|",poi(esp+4),......"

```
[ExFreePoolWithTag] P B853AD88, Tag 00000000----------------------------------------------
PROCESS b5e3d040  SessionId: 1  Cid: 14dc    Peb: 002da000  ParentCid: 1720
    DirBase: 7fff0740  ObjectTable: b5c86f00  HandleCount: <Data Not Accessible>
    Image: python.exe

ChildEBP RetAddr  Args to Child
b6fd2808 81d4f4f2 b853ad88 00000000 93961100 nt!ExFreePoolWithTag
b6fd281c 926ba54e b853ad88 00000000 b6fd2840 nt!VerifierExFreePoolWithTag+0x3e
b6fd282c 926ba533 b853ad88 00000000 93961100 win32kfull!NSInstrumentation::PlatformFree+0x10
b6fd2840 916e19ed 93961100 b853ad88 00000000 win32kfull!Win32FreeToPagedLookasideListImpl+0x43
b6fd2930 916eb274 00000000 b853ad88 00000000 win32kbase!SURFACE::bDeleteSurface+0x8fd
b6fd2944 916d63fc 00000000 b6fd2a08 b6fd2a14 win32kbase!SURFREF::bDeleteSurface+0x14
b6fd29b0 9262ec8b 170106a6 ab050107 00000001 win32kbase!hbmSelectBitmap+0xb5c
b6fd29e4 92601178 b6fd2a08 b6fd2a14 07ff05e6 win32kfull!DEVLOCKBLTOBJ::bLock+0xb4b
```

Observations from call-stack:

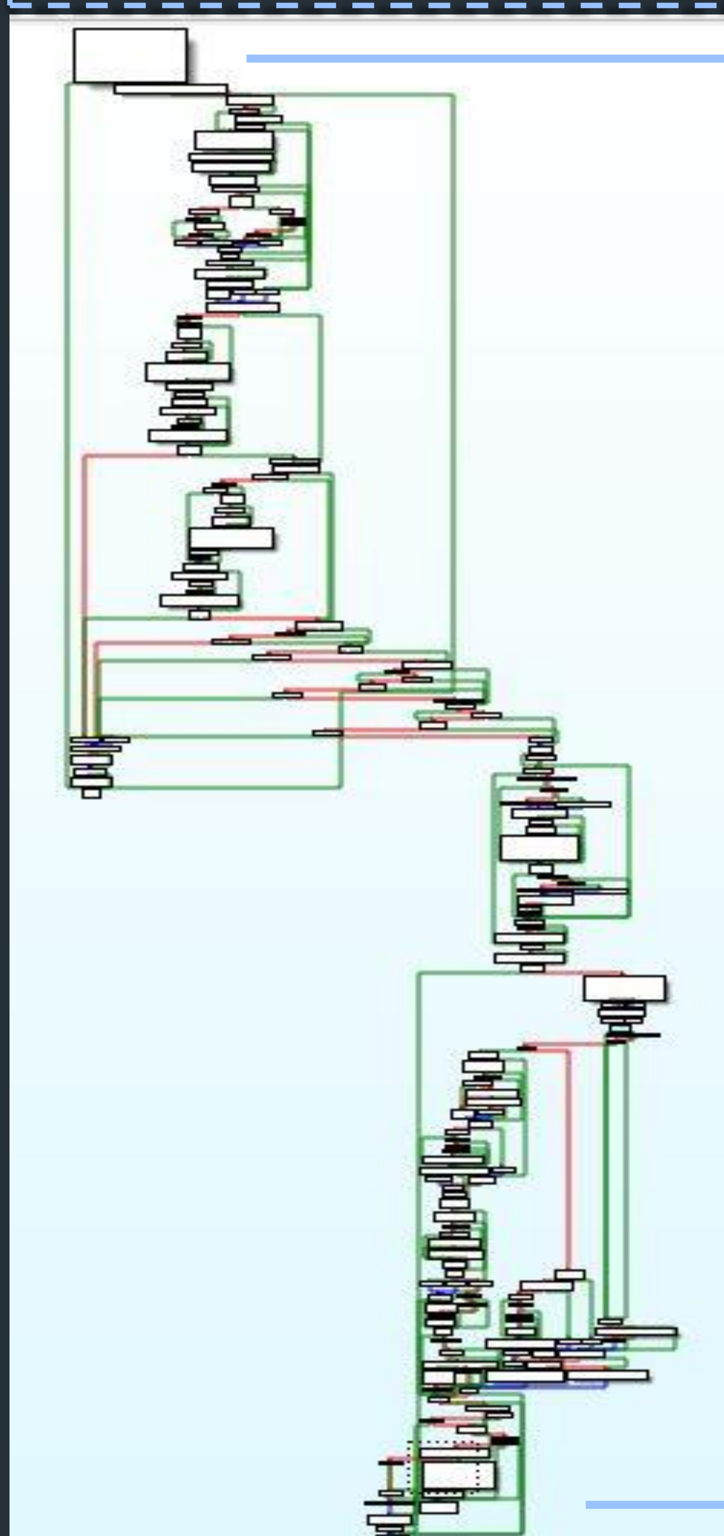
1. Pool B853AD88 is freed during deletion of SURFACE object (recall: this is the addr that was read and caused the BSOD)
2. SURFACE object is referenced from SURFREF object during hbmSelectBitmap()
3. We now know where in DEVLOCKBLTOBJ::bLock() would lead to ExFreePoolWithTag()

[Remove logging breakpoint] :

*1: kd> bc **

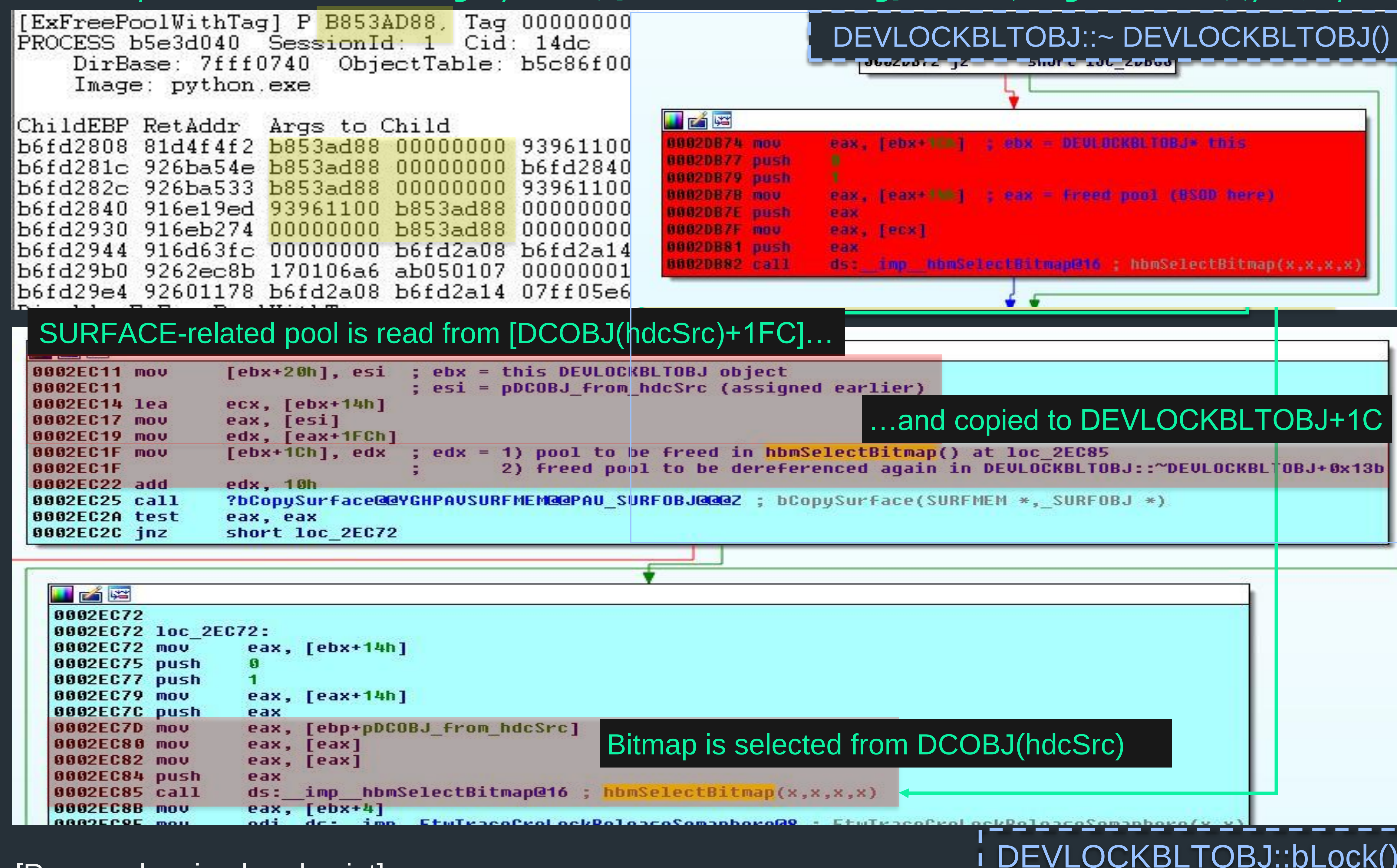
++

Case Study – Analysis



[Set logging breakpoint] :

1: kd> bp nt!ExFreePoolWithTag ".printf |"[ExFreePoolWithTag] P %08X, Tag %08X---|",poi(esp+4),......"



[Remove logging breakpoint] :

*1: kd> bc **

++

Case Study – Analysis Summary

- A DCOBJ object is instantiated from PlgBlt (…, hdcSrc, …)
- The DCOBJ object is passed as 2nd argument in DEVLOCKBLTOBJ::bLock (…, DCOBJ_hdcSrc, …)
- At BSOD faulting address, a de-reference is read from a freed pool; *[B853AD88h+14h]
- Freed pool is used in 2 ways in DEVLOCKBLTOBJ::bLock()
  1. Copied from [DCOBJ+1FCh] to [DEVLOCKBLTOBJ+1Ch]
  2. Freed in SURFACE::bSelectSurface (…, B853AD88h, …) during hbmSelectBitmap()
- Eventually, this SURFACE-related freed pool is referenced in DEVLOCKBLTOBJ::~DEVLOCKBLTOBJ() destructor, resulting in a Use-After-Free vulnerability
- Misc: DEVLOCKBLTOBJ is used in a ::bLock() -> ::~DEVLOCKBLTOBJ() manner
  – ::bLock() initializes and locks DEVLOCKBLTOBJ at the same time

++

Fuzzing the Windows Kernel

- CONCLUSION AND FUTURE WORK

++
Conclusion

- WIN32K.SYS as an attractive target for sandbox escapes

- Discussed about framework…
  - Architecture and Components
  - Algorithms
  - Setup and Configuration

- Effectiveness
  - Results from ~8 weeks of fuzzing
  - Demonstrated how this fuzzing could create a source HDC that would free a SURFACE-related pool during hbmSelectBitmap()

++

Future Work

- Server-Client (Distributed) Model
- WIN32k.SYS User-Mode Callbacks
  - “Kernel Attacks through User-Mode Callbacks” by Tarjei Mandt
  - “Analyzing local privilege escalations in win32k ” by Thomas Garnier

++

Future Work

- Expand catalog for other .sys (then again WIN32K.SYS for sandbox escapes may not last long…)
  - Chrome's DisallowWin32kSystemCalls

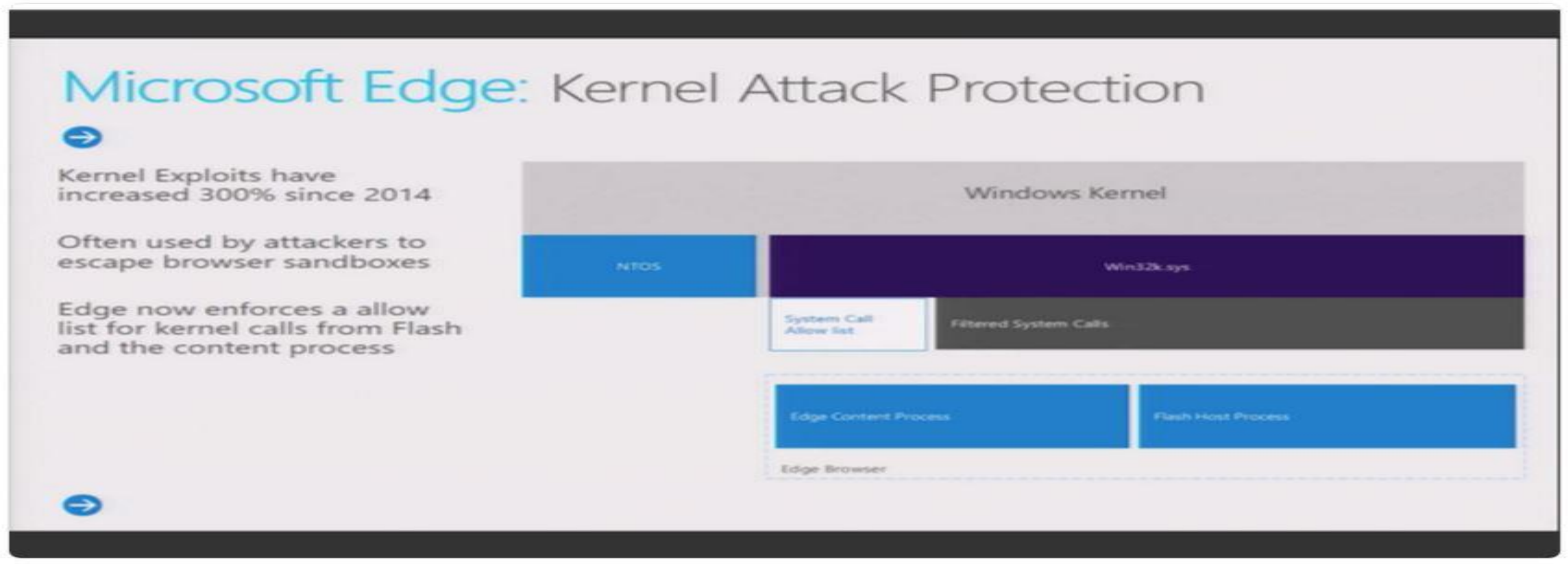
Win32k.sys lockdown:

- >= Win8
- ProcessSystemCallDisablePolicy, which allows selective disabling of system calls available from the target process.
- Renderer processes now have this set to DisallowWin32kSystemCalls which means that calls from user mode that are serviced by win32k.sys are no longer permitted. This significantly reduces the kernel attack surface available from a renderer. See here for more details.

  - WIN32K.SYS syscall filter in Edge

James Forshaw @tiraniddo — Follow

Win10 getting a syscall filter! Shame only for win32k, no matter what Dave said in the talk to imply otherwise :-(



++ Acknowledgements

- Nils
- James Loureiro
- Georgi Geshev

++

Thank You!

- Questions?