Lecture 19 – Finding Vulnerabilities

Stephen Checkoway

Oberlin College

Slides based on Bailey's ECE 422

Finding Vulns

- Specification testing
- Automated white box tools
- Fuzzing
- Reverse engineering

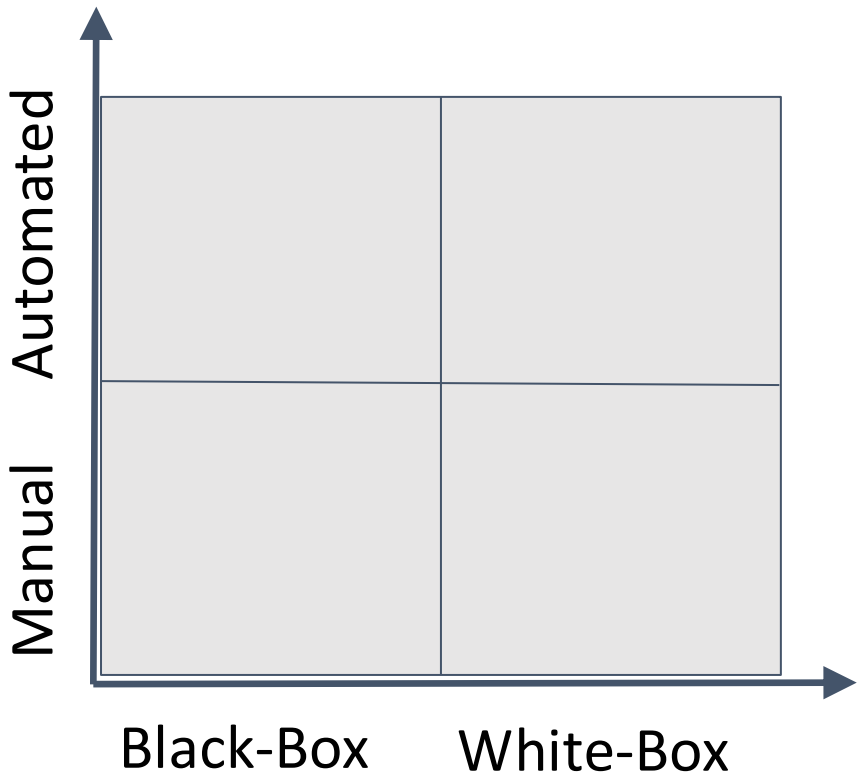
The Need for Specifications

- Testing checks whether program implementation agrees with program specification
- Without a specification, there is nothing to test!
- Testing is a form of consistency checking between implementation and specification
  - Recurring theme for software quality checking approaches
  - What if both implementation and specification are wrong?

Developer != Tester

- Developer writes implementation, tester writes specification
- Unlikely that both will independently make the same mistake
- Specifications useful even if written by developer itself
  - Much simpler than implementation
  - Specification unlikely to have same mistake as implementation

Classification of Testing Approaches



Automated vs. Manual Testing

- Automated Testing:
  - Find bugs more quickly
  - No need to write tests
  - If software changes, no need to maintain tests
- Manual Testing:
  - Efficient test suite
  - Potentially better coverage

Black-Box vs. White-Box Testing

- Black-Box Testing:
  - Can work with code that cannot be modified
  - Does not need to analyze or study code
  - Code can be in any format (managed, binary, obfuscated)
- White-Box Testing:
  - Efficient test suite
  - Potentially better coverage

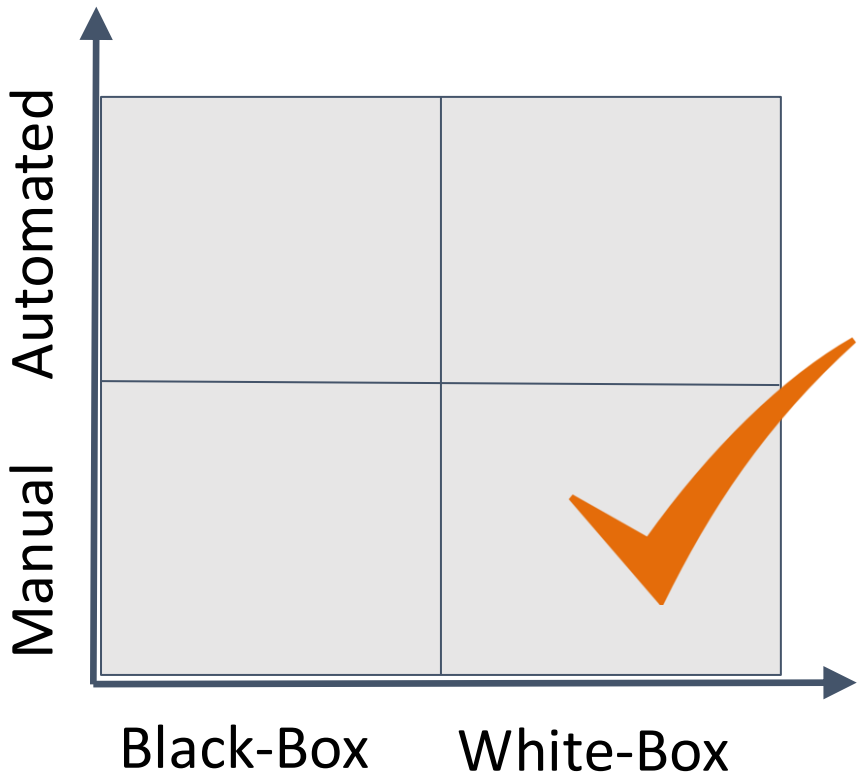
How Good Is Your Test Suite?

- How do we know that our test suite is good?
  - Too few tests: may miss bugs
  - Too many tests: costly to run, bloat and redundancy, harder to maintain
- Example: SQLite
  “As of version 3.42.0 (2023-05-16), the SQLite library consists of approximately 155.8 KSLOC of C code. (KSLOC means thousands of "Source Lines Of Code" or, in other words, lines of code excluding blank lines and comments.) By comparison, the project has 590 times as much test code and test scripts - 92053.1 KSLOC.”
- Nevertheless, 45 CVEs listed in their recent SQLite CVEs page (as checked 2026-03-13)

Code Coverage

- Metric to quantify extent to which a program's code is tested by a given test suite
  - Function coverage: which functions were called?
  - Statement coverage: which statements were executed?
  - Branch coverage: which branches were taken?
- Given as percentage of some aspect of the program executed in the tests
- 100% coverage rare in practice: e.g., inaccessible code
  - Often required for safety-critical applications
  - Example: SQLite has 100% branch coverage

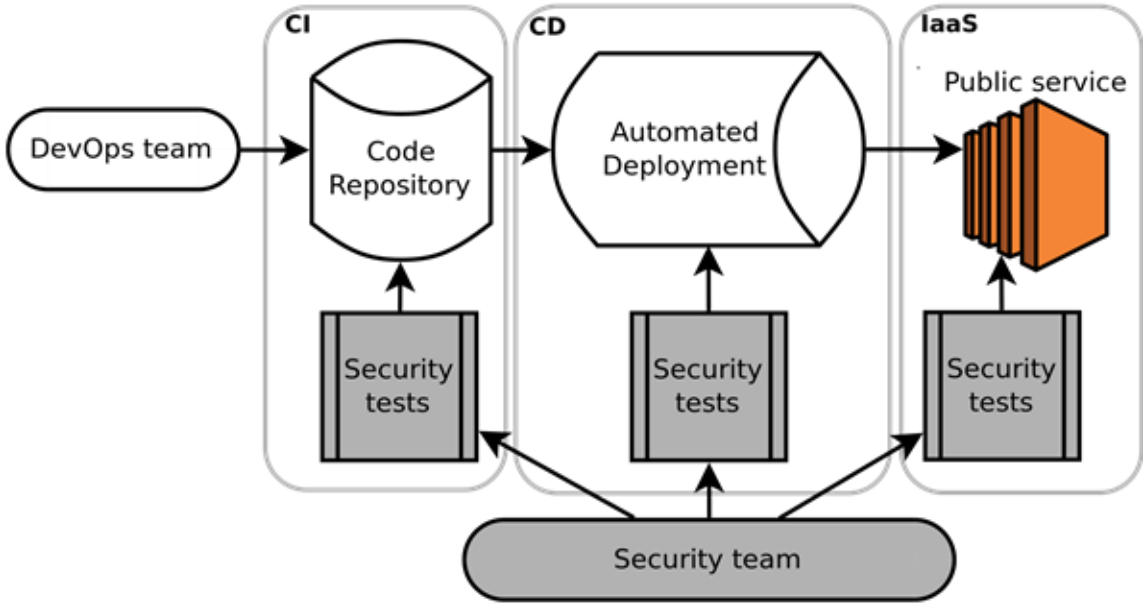
Classification of Testing Approaches

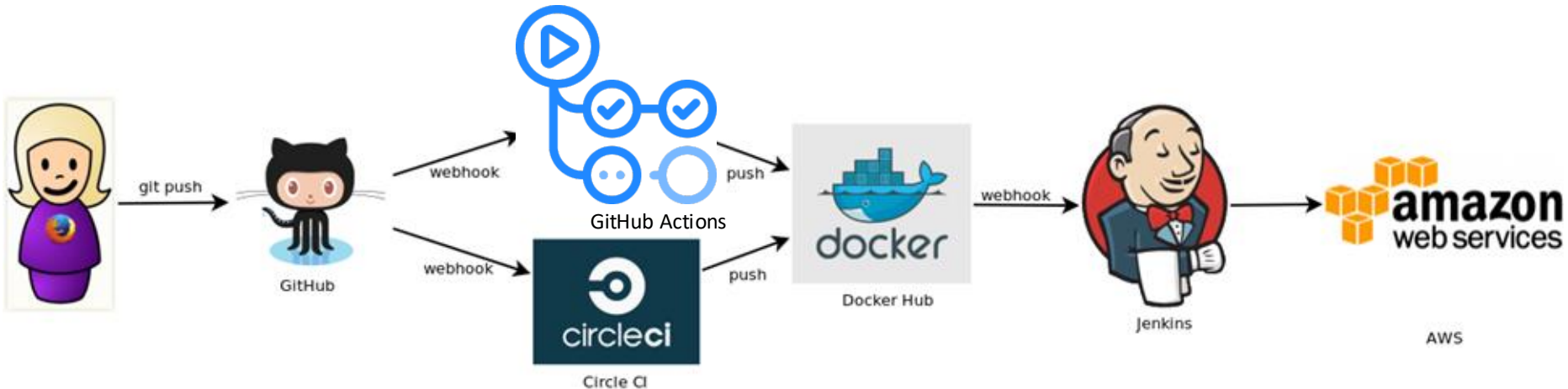


Manual white-box testing

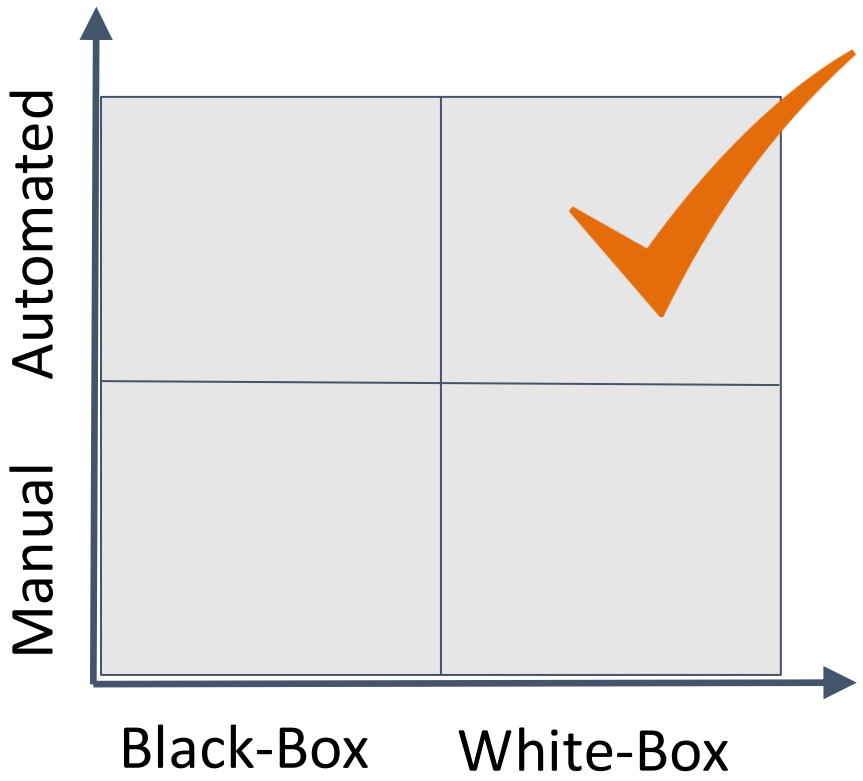
- Tests written by hand
- Full knowledge of source code/deployment/infrastructure
- Can test all parts
- Test *running* can be automated (e.g., on commits/deployment)

Test Driven Security





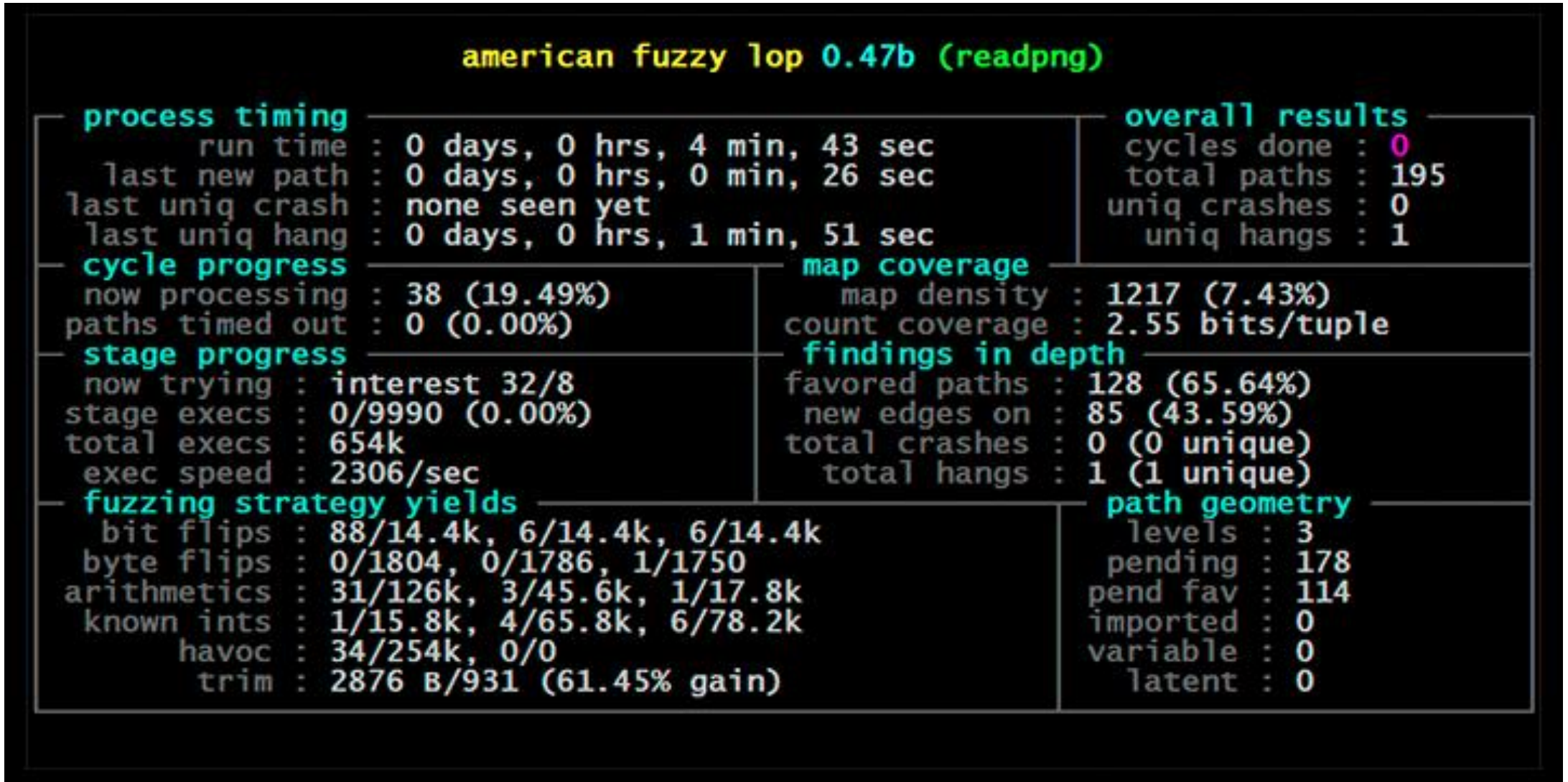
Classification of Testing Approaches



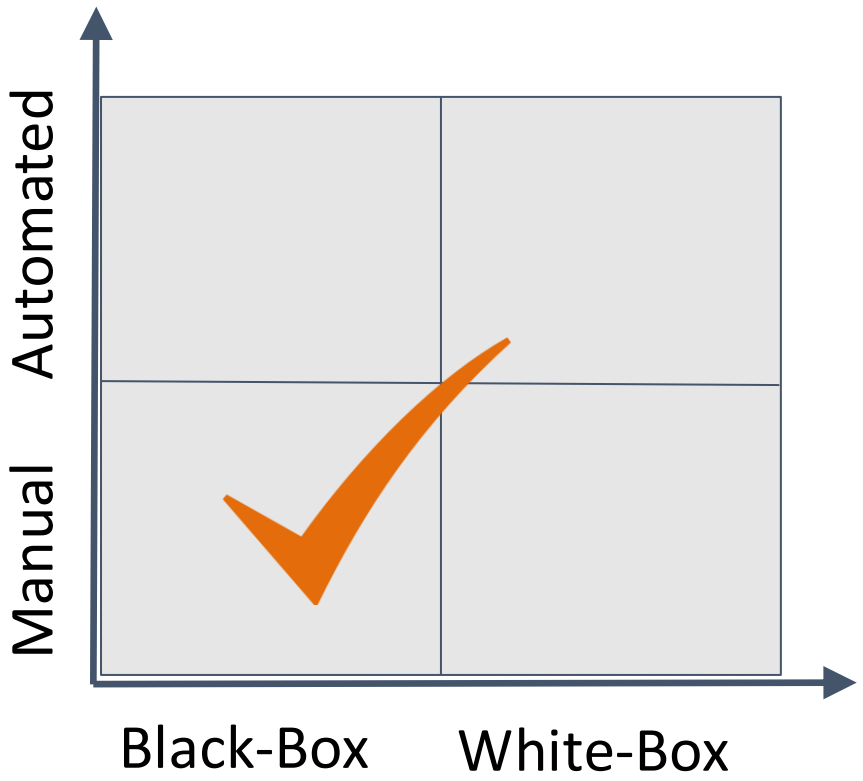
Automated white-box testing

- Tests created automatically/dynamically
- Godefroid et al. “Automated Whitebox Fuzz Testing”
  - Record trace of program on well-formed inputs
  - Symbolic execution to capture constraints on input
  - Negate a constraint, use a constraint solver to derive new input, run on that input to test different control paths
- American fuzzy lop
  - Compile-time instrumentation
  - Genetic algorithms guided by the instrumentation

Automated white-box testing tools



Classification of Testing Approaches

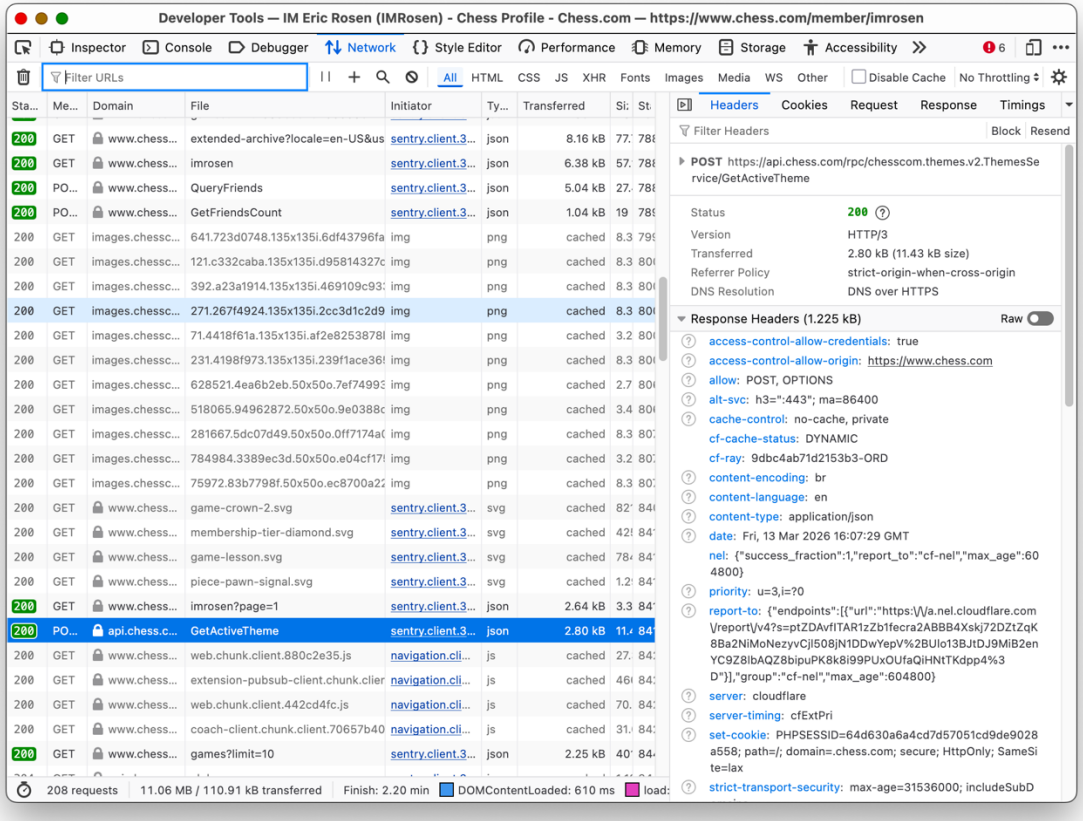


Manual black-box testing

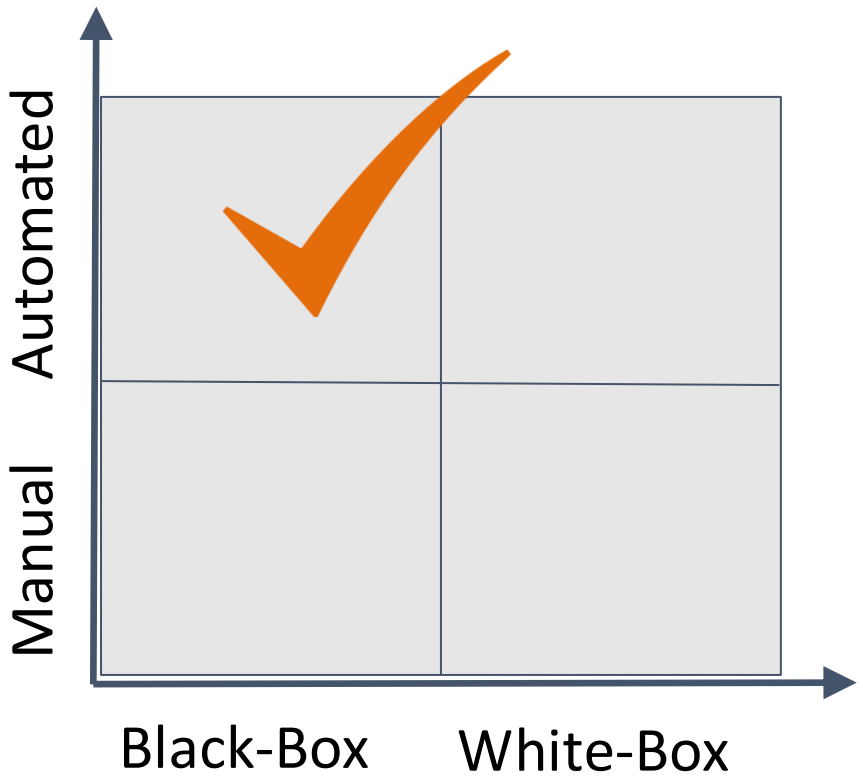
- Tester interacts with the system in a black-box fashion (i.e., with no insight into the inner workings of the system)
- Crafts ill-formed inputs, tests them, and records how the system reacts

Black-box testing example: A website

- Open the website in a browser with developer tools open
- Examine URLs loaded, looking for API calls
- Send requests to the same endpoints using different/invalid arguments
- Look at the responses (error messages sometimes report database errors, e.g., authentication failures and bad table names)
- Modify arguments (e.g., sql injection) and try again



Classification of Testing Approaches



Automated black-box testing

- Fuzzing components
  - Test case generation
  - Application execution
  - Exception detection and logging

- Same basic approach as the manual black-box testing except automated and less targeted

Test Case Generation

- Random Fuzzing
- “Dumb” (mutation-based) Fuzzing
  - Mutate an existing input
- “Smart” (generation-based) Fuzzing
  - Generate an input based on a model (grammar)

Mutation Fuzzer

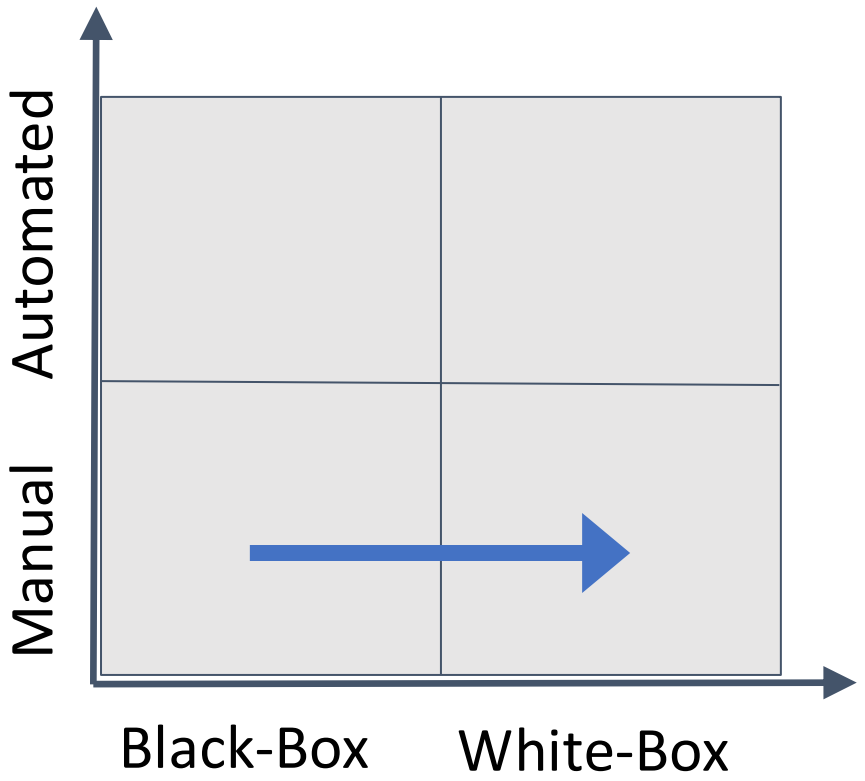
- Charlie Miller's "5 lines of Python" fuzzer
- Found bugs in PDF and PowerPoint readers

```
numwrites=random.randrange(
                    math.ceil((float(len(buf)) / FuzzFactor)))+1
for j in range(numwrites):
   rbyte = random.randrange(256)
   rn = random.randrange(len(buf))
   buf[rn] = "%c"%(rbyte);
```

Generation-based fuzzing

- File formats like images, videos, fonts, and even html have a structure
- Any input that doesn’t adhere at least somewhat to the structure will get rejected early before any bugs can be found (e.g., VLC won’t just try to play random data as video)
- Describe the input as a grammar (in the sense of CSCI 383)
- Fuzzer will generate inputs based on the grammar and feed them to the program

Moving from black-box to white-box



Reverse Engineering

- Reverse Engineering (RE) -- process of discovering the technological principles of a [insert noun] through analysis of its structure, function, and operation.
- The development cycle ... backwards

Why Reverse Engineer?

- Malware analysis
- Vulnerability or exploit research
- Check for copyright/patent violations
- Interoperability (e.g., understanding a file/protocol format)
- Copy protection removal

Legality

- Gray Area (a common theme)
- Usually breaches the EULA contract of software
- Additionally -- DMCA law governs reversing in U.S. (17 U.S.C. § 1201)
  - (a)(1)(A) No person shall circumvent a technological measure that effectively controls access to a work protected under this title […]
  - (f) Reverse Engineering
    - (1) […] a person who has lawfully obtained the right to use a copy of a computer program may circumvent a technological measure that effectively controls access to a particular portion of that program for the sole purpose of […] achiev[ing] interoperability […] with other programs […]
  - (g)(2) Permissible acts of encryption research
  - (j)(2) Permissible acts of security testing

Two Techniques

- Static Code Analysis (structure)
  - Disassemblers
- Dynamic Code Analysis (operation)
  - Tracing / Hooking
  - Debuggers
- Combination of the two works best in my experience

Disassembly (CSCI 210 all over again!)

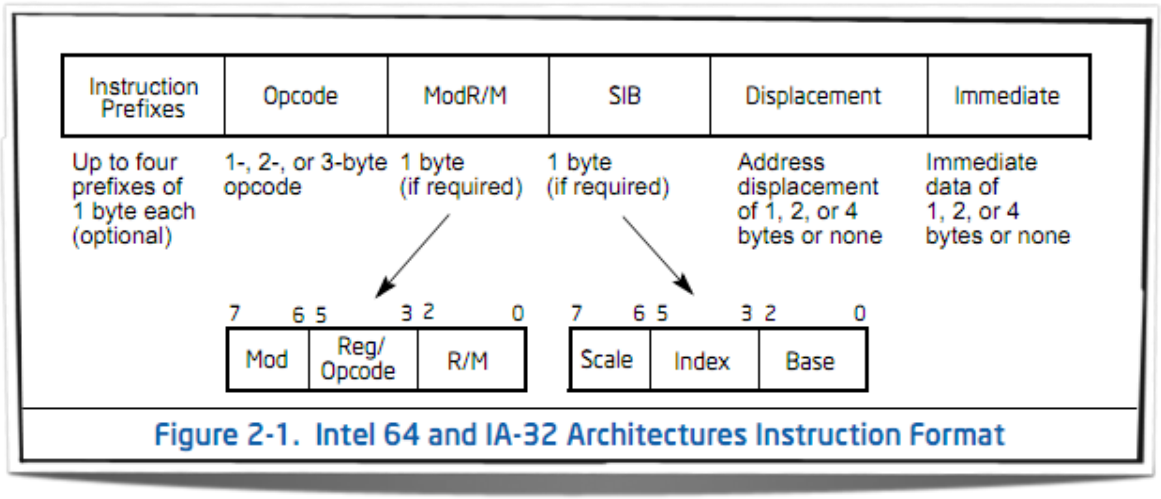
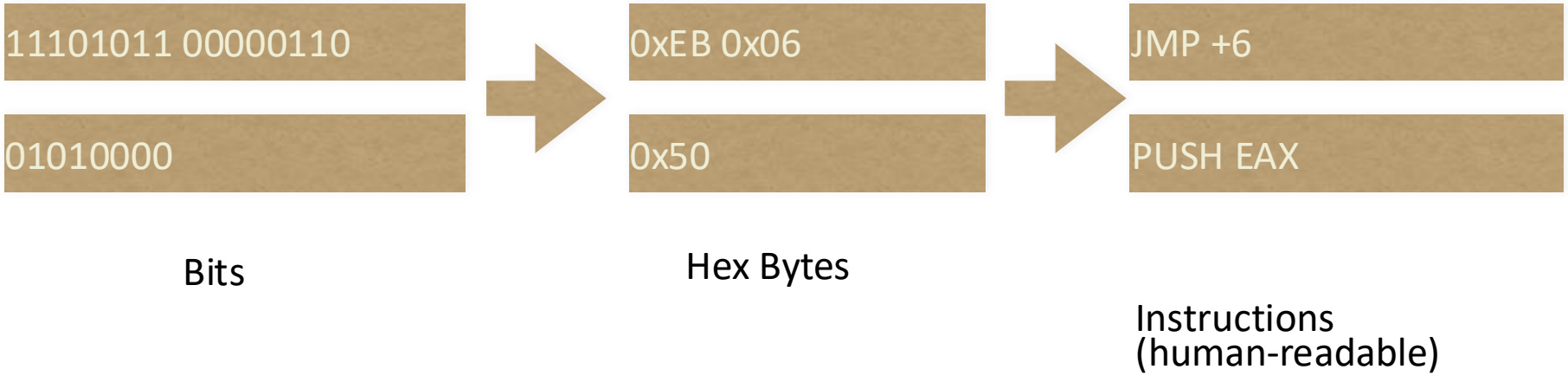
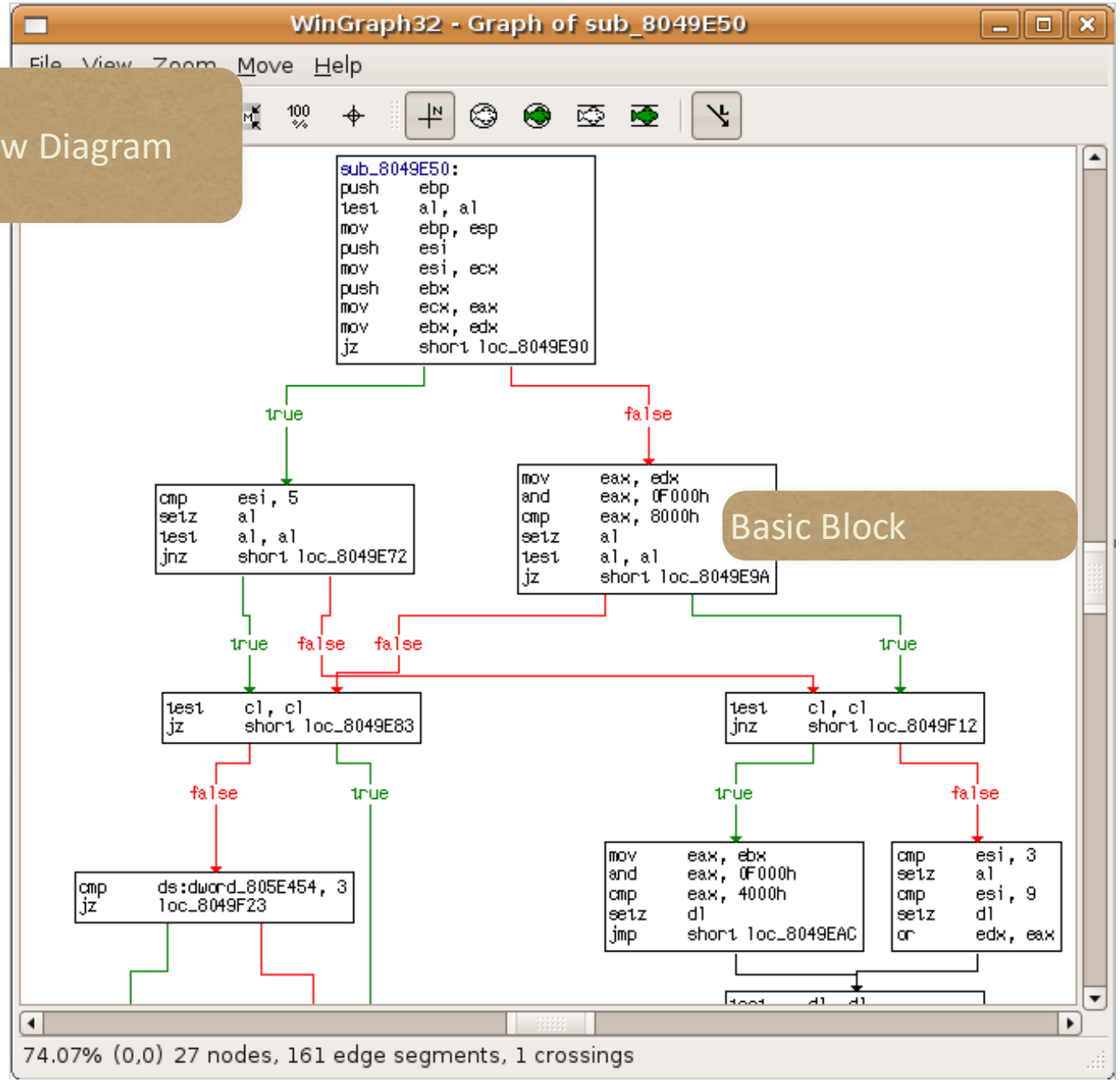
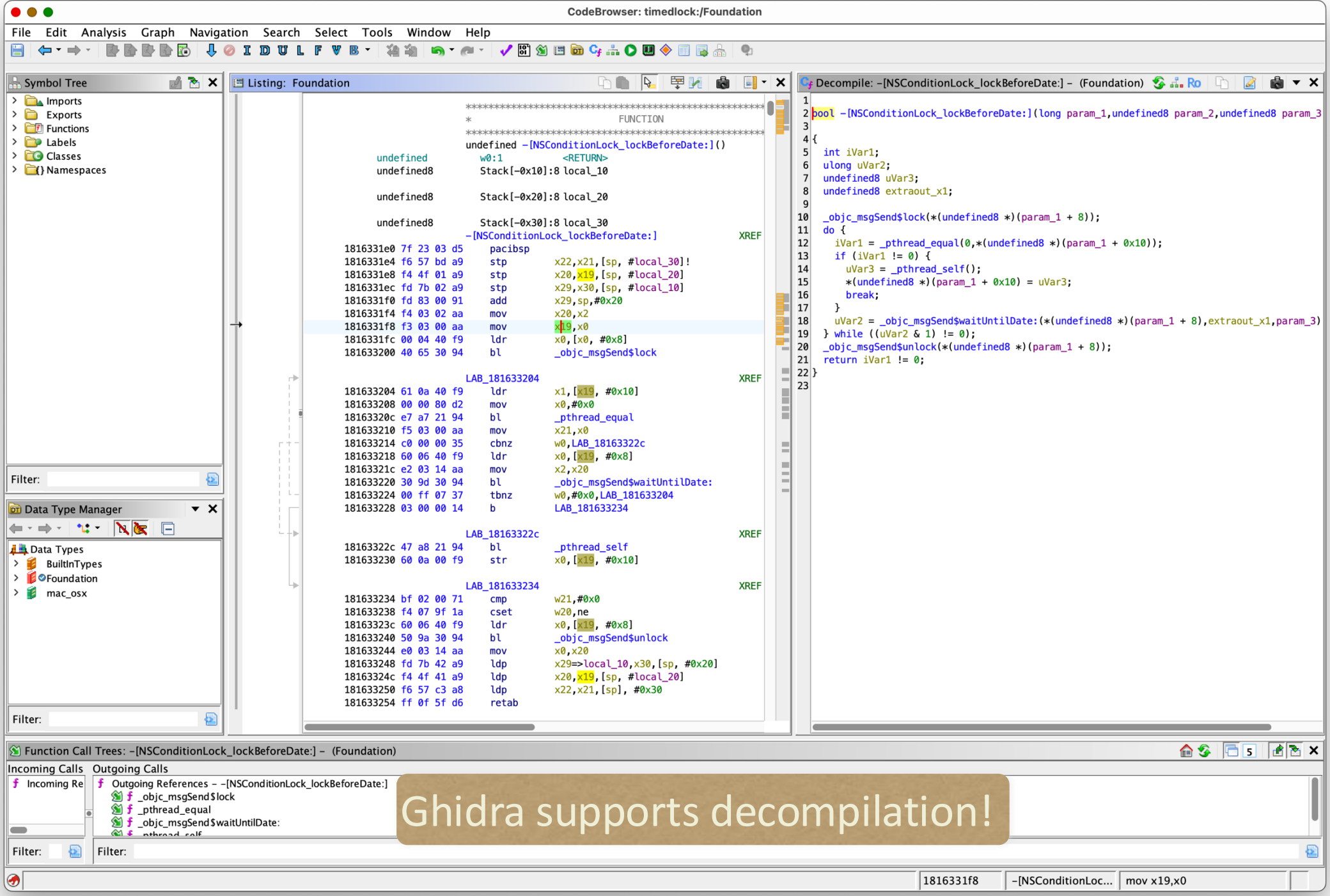


Figure 2-1. Intel 64 and IA-32 Architectures Instruction Format



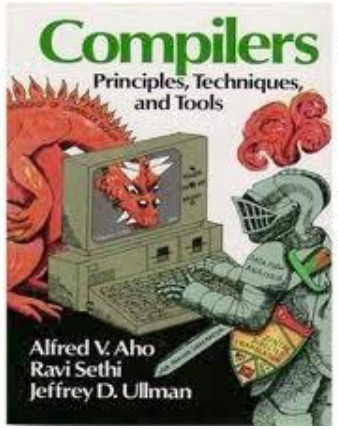
Control Flow Diagram




CodeBrowser: timedlock:/Foundation
File Edit Analysis Graph Navigation Search Select Tools Window Help
Symbol Tree
Imports
Exports
Functions
Labels
Classes
Namespaces
Filter:
Data Type Manager
Data Types
BuiltInTypes
Foundation
mac_osx
Filter:
Listing: Foundation
FUNCTION
undefined -[NSConditionLock_lockBeforeDate:]()
undefined w0:1 <RETURN>
undefined8 Stack[-0x10]:8 local_10
undefined8 Stack[-0x20]:8 local_20
undefined8 Stack[-0x30]:8 local_30
-[NSConditionLock_lockBeforeDate:] XREF
1816331e0 7f 23 03 d5 pacibsp
1816331e4 f6 57 bd a9 stp x22,x21,[sp, #local_30]!
1816331e8 f4 4f 01 a9 stp x20,x19,[sp, #local_20]
1816331ec fd 7b 02 a9 stp x29,x30,[sp, #local_10]
1816331f0 fd 83 00 91 add x29,sp,#0x20
1816331f4 f4 03 02 aa mov x20,x2
1816331f8 f3 03 00 aa mov x19,x0
1816331fc 00 04 40 f9 ldr x0,[x0, #0x8]
181633200 40 65 30 94 bl _objc_msgSend$lock
LAB_181633204 XREF
181633204 61 0a 40 f9 ldr x1,[x19, #0x10]
181633208 00 00 80 d2 mov x0,#0x0
18163320c e7 a7 21 94 bl _pthread_equal
181633210 f5 03 00 aa mov x21,x0
181633214 c0 00 00 35 cbnz w0,LAB_18163322c
181633218 60 06 40 f9 ldr x0,[x19, #0x8]
18163321c e2 03 14 aa mov x2,x20
181633220 30 9d 30 94 bl _objc_msgSend$waitUntilDate:
181633224 00 ff 07 37 tbnz w0,#0x0,LAB_181633204
181633228 03 00 00 14 b LAB_181633234
LAB_18163322c XREF
18163322c 47 a8 21 94 bl _pthread_self
181633230 60 0a 00 f9 str x0,[x19, #0x10]
LAB_181633234 XREF
181633234 bf 02 00 71 cmp w21,#0x0
181633238 f4 07 9f 1a cset w20,ne
18163323c 60 06 40 f9 ldr x0,[x19, #0x8]
181633240 50 9a 30 94 bl _objc_msgSend$unlock
181633244 e0 03 14 aa mov x0,x20
181633248 fd 7b 42 a9 ldp x29=>local_10,x30,[sp, #0x20]
18163324c f4 4f 41 a9 ldp x20,x19,[sp, #local_20]
181633250 f6 57 c3 a8 ldp x22,x21,[sp], #0x30
181633254 ff 0f 5f d6 retab
Decompile: -[NSConditionLock_lockBeforeDate:] - (Foundation)
bool -[NSConditionLock_lockBeforeDate:](long param_1,undefined8 param_2,undefined8 param_3
{
int iVar1;
ulong uVar2;
undefined8 uVar3;
undefined8 extraout_x1;
_objc_msgSend$lock(*(undefined8 *)(param_1 + 8));
do {
iVar1 = _pthread_equal(0,*(undefined8 *)(param_1 + 0x10));
if (iVar1 != 0) {
uVar3 = _pthread_self();
*(undefined8 *)(param_1 + 0x10) = uVar3;
break;
}
uVar2 = _objc_msgSend$waitUntilDate:(*(undefined8 *)(param_1 + 8),extraout_x1,param_3)
} while ((uVar2 & 1) != 0);
_objc_msgSend$unlock(*(undefined8 *)(param_1 + 8));
return iVar1 != 0;
}
Function Call Trees: -[NSConditionLock_lockBeforeDate:] - (Foundation)
Incoming Calls
Outgoing Calls
Incoming Re
Outgoing References - -[NSConditionLock_lockBeforeDate:]
_objc_msgSend$lock
_pthread_equal
_objc_msgSend$waitUntilDate:
Filter:
Filter:
Ghidra supports decompilation!
1816331f8
-[NSConditionLoc...
mov x19,x0

Difficulties

- Imperfect disassembly (and especially decompilation!)
- Benign Optimizations
  - Constant folding
  - Dead code elimination
  - Inline expansion
  - Loop unrolling
  - etc...
- Intentional Obfuscation
  - Packing
  - No-op instructions
  - Unused function calls can fool disassembly tools into treating a region of memory as code when the actual code that is being run might start in the middle of some of the instructions

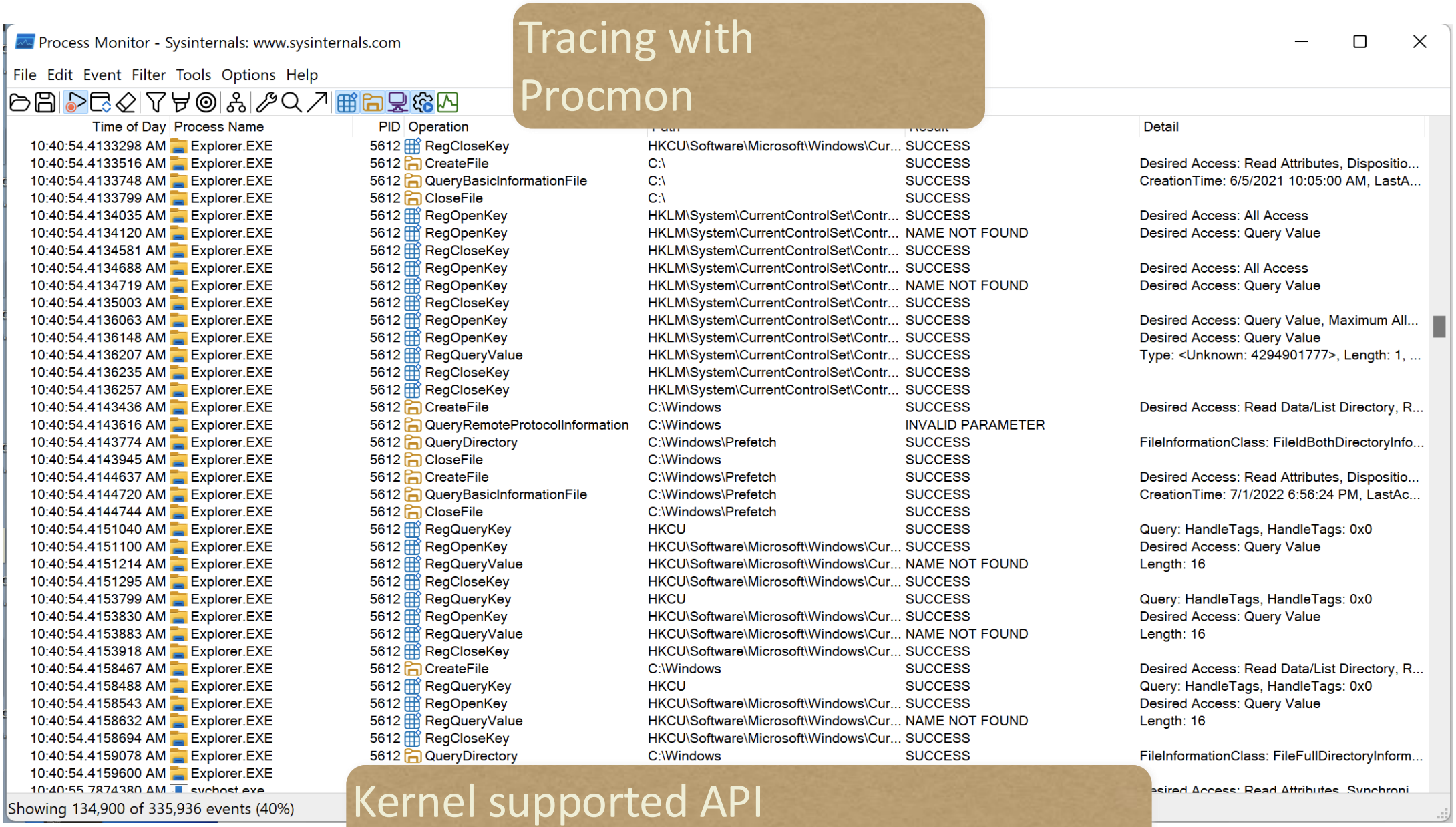


Packers and Crypters

- Packers produce executables that decompresses themselves at runtime
  - UPX is a freely available, cross-platform packer
- Crypters use some form of encryption to hide the code, decrypted at runtime
- Malware might use both

Dynamic Analysis

- A couple techniques available:
  - Tracing / Hooking
  - Debugging


Tracing with Procmon
Process Monitor - Sysinternals: www.sysinternals.com
File Edit Event Filter Tools Options Help
Time of Day | Process Name | PID | Operation | Path | Result | Detail
10:40:54.4133298 AM | Explorer.EXE | 5612 | RegCloseKey | HKCU\Software\Microsoft\Windows\Cur... | SUCCESS
10:40:54.4133516 AM | Explorer.EXE | 5612 | CreateFile | C:\ | SUCCESS | Desired Access: Read Attributes, Dispositio...
10:40:54.4133748 AM | Explorer.EXE | 5612 | QueryBasicInformationFile | C:\ | SUCCESS | CreationTime: 6/5/2021 10:05:00 AM, LastA...
10:40:54.4133799 AM | Explorer.EXE | 5612 | CloseFile | C:\ | SUCCESS
10:40:54.4134035 AM | Explorer.EXE | 5612 | RegOpenKey | HKLM\System\CurrentControlSet\Contr... | SUCCESS | Desired Access: All Access
10:40:54.4134120 AM | Explorer.EXE | 5612 | RegOpenKey | HKLM\System\CurrentControlSet\Contr... | NAME NOT FOUND | Desired Access: Query Value
10:40:54.4134581 AM | Explorer.EXE | 5612 | RegCloseKey | HKLM\System\CurrentControlSet\Contr... | SUCCESS
10:40:54.4134688 AM | Explorer.EXE | 5612 | RegOpenKey | HKLM\System\CurrentControlSet\Contr... | SUCCESS | Desired Access: All Access
10:40:54.4134719 AM | Explorer.EXE | 5612 | RegOpenKey | HKLM\System\CurrentControlSet\Contr... | NAME NOT FOUND | Desired Access: Query Value
10:40:54.4135003 AM | Explorer.EXE | 5612 | RegCloseKey | HKLM\System\CurrentControlSet\Contr... | SUCCESS
10:40:54.4136063 AM | Explorer.EXE | 5612 | RegOpenKey | HKLM\System\CurrentControlSet\Contr... | SUCCESS | Desired Access: Query Value, Maximum All...
10:40:54.4136148 AM | Explorer.EXE | 5612 | RegOpenKey | HKLM\System\CurrentControlSet\Contr... | SUCCESS | Desired Access: Query Value
10:40:54.4136207 AM | Explorer.EXE | 5612 | RegQueryValue | HKLM\System\CurrentControlSet\Contr... | SUCCESS | Type: <Unknown: 4294901777>, Length: 1, ...
10:40:54.4136235 AM | Explorer.EXE | 5612 | RegCloseKey | HKLM\System\CurrentControlSet\Contr... | SUCCESS
10:40:54.4136257 AM | Explorer.EXE | 5612 | RegCloseKey | HKLM\System\CurrentControlSet\Contr... | SUCCESS
10:40:54.4143436 AM | Explorer.EXE | 5612 | CreateFile | C:\Windows | SUCCESS | Desired Access: Read Data/List Directory, R...
10:40:54.4143616 AM | Explorer.EXE | 5612 | QueryRemoteProtocolInformation | C:\Windows | INVALID PARAMETER
10:40:54.4143774 AM | Explorer.EXE | 5612 | QueryDirectory | C:\Windows\Prefetch | SUCCESS | FileInformationClass: FileIdBothDirectoryInfo...
10:40:54.4143945 AM | Explorer.EXE | 5612 | CloseFile | C:\Windows | SUCCESS
10:40:54.4144637 AM | Explorer.EXE | 5612 | CreateFile | C:\Windows\Prefetch | SUCCESS | Desired Access: Read Attributes, Dispositio...
10:40:54.4144720 AM | Explorer.EXE | 5612 | QueryBasicInformationFile | C:\Windows\Prefetch | SUCCESS | CreationTime: 7/1/2022 6:56:24 PM, LastAc...
10:40:54.4144744 AM | Explorer.EXE | 5612 | CloseFile | C:\Windows\Prefetch | SUCCESS
10:40:54.4151040 AM | Explorer.EXE | 5612 | RegQueryKey | HKCU | SUCCESS | Query: HandleTags, HandleTags: 0x0
10:40:54.4151100 AM | Explorer.EXE | 5612 | RegOpenKey | HKCU\Software\Microsoft\Windows\Cur... | SUCCESS | Desired Access: Query Value
10:40:54.4151214 AM | Explorer.EXE | 5612 | RegQueryValue | HKCU\Software\Microsoft\Windows\Cur... | NAME NOT FOUND | Length: 16
10:40:54.4151295 AM | Explorer.EXE | 5612 | RegCloseKey | HKCU\Software\Microsoft\Windows\Cur... | SUCCESS
10:40:54.4153799 AM | Explorer.EXE | 5612 | RegQueryKey | HKCU | SUCCESS | Query: HandleTags, HandleTags: 0x0
10:40:54.4153830 AM | Explorer.EXE | 5612 | RegOpenKey | HKCU\Software\Microsoft\Windows\Cur... | SUCCESS | Desired Access: Query Value
10:40:54.4153883 AM | Explorer.EXE | 5612 | RegQueryValue | HKCU\Software\Microsoft\Windows\Cur... | NAME NOT FOUND | Length: 16
10:40:54.4153918 AM | Explorer.EXE | 5612 | RegCloseKey | HKCU\Software\Microsoft\Windows\Cur... | SUCCESS
10:40:54.4158467 AM | Explorer.EXE | 5612 | CreateFile | C:\Windows | SUCCESS | Desired Access: Read Data/List Directory, R...
10:40:54.4158488 AM | Explorer.EXE | 5612 | RegQueryKey | HKCU | SUCCESS | Query: HandleTags, HandleTags: 0x0
10:40:54.4158543 AM | Explorer.EXE | 5612 | RegOpenKey | HKCU\Software\Microsoft\Windows\Cur... | SUCCESS | Desired Access: Query Value
10:40:54.4158632 AM | Explorer.EXE | 5612 | RegQueryValue | HKCU\Software\Microsoft\Windows\Cur... | NAME NOT FOUND | Length: 16
10:40:54.4158694 AM | Explorer.EXE | 5612 | RegCloseKey | HKCU\Software\Microsoft\Windows\Cur... | SUCCESS
10:40:54.4159078 AM | Explorer.EXE | 5612 | QueryDirectory | C:\Windows | SUCCESS | FileInformationClass: FileFullDirectoryInform...
10:40:54.4159600 AM | Explorer.EXE
10:40:55.7874380 AM | svchost.exe
Showing 134,900 of 335,936 events (40%)
Kernel supported API
Event Tracing for Windows (ETW)

strace and ltrace

- strace traces system calls (you'll use this in project 2) a process makes
- ltrace traces dynamic library function calls (by intercepting calls made via the PLT)

- These, especially strace, are invaluable debugging/analysis tools

Hooking

- Intercept function calls and run additional code at that point
- Basic idea is to inject code into the process (easiest to do at run time but can be done via binary rewriting)
- Replace the few few bytes of a function body with a jmp to the injected code
- Injected code can perform logging and return to the legitimate code (after performing whatever action the replaced code should have been doing like setting up the stack frame) or it can replace the functionality entirely (e.g., bypass authentication/authorization checks)

Microsoft Research's Detours

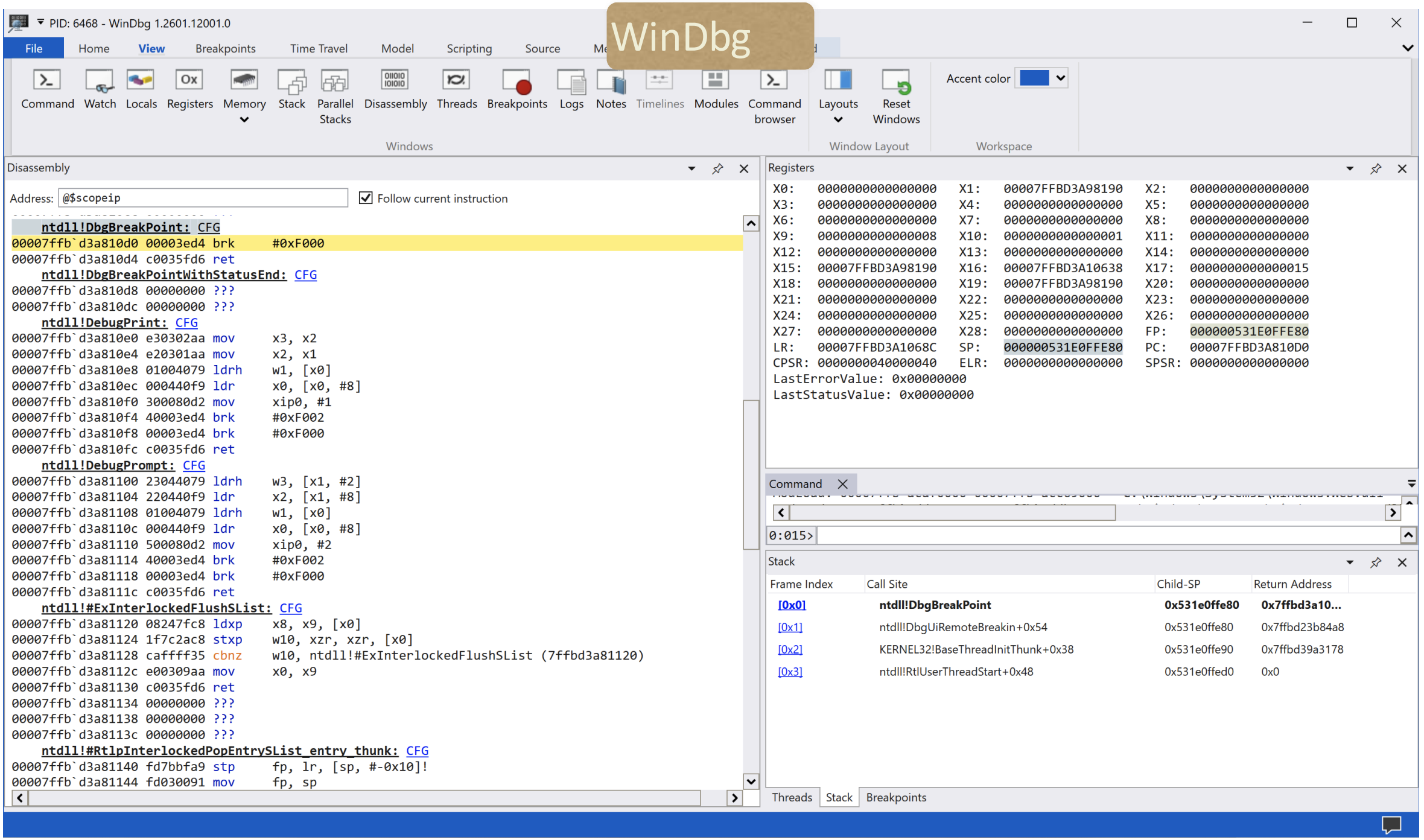
- Supported mechanism for hooking functions at runtime for Windows applications

ELF LD_PRELOAD

- LD_PRELOAD environment variable specifies a dynamic library to load before all of the others
- Global functions in the preloaded library will be called via the PLT rather than the functions with the same name in other libraries
- Allows hooking functions in dynamic libraries

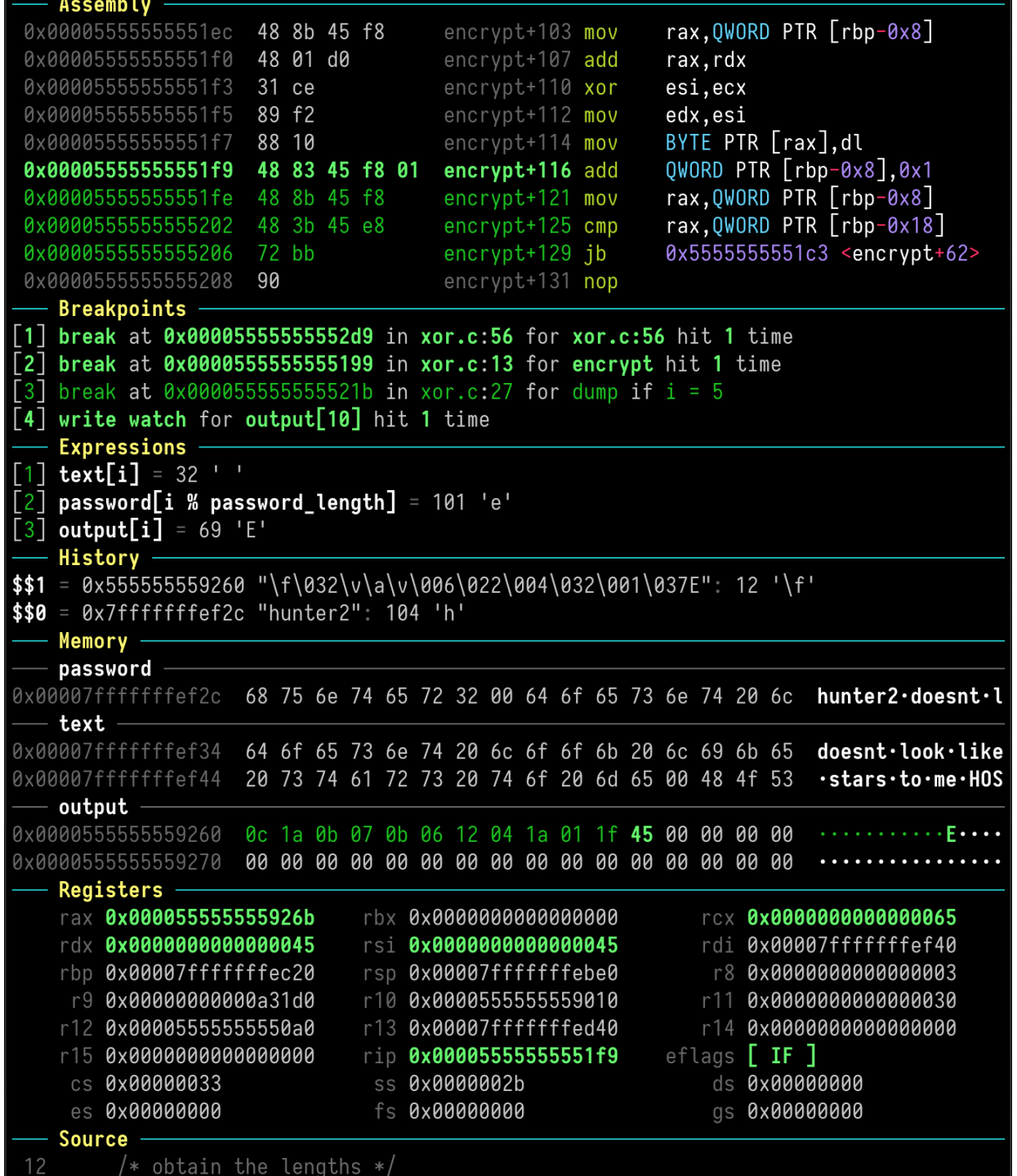
Debugger Features

- Trace every instruction a program executes -- single step
- Or, let program execute normally until an exception
- At every step or exception, can observe / modify:
  - Instructions, stack, heap, and register set
- May inject exceptions at arbitrary code locations
- INT 3 instruction generates a breakpoint exception


PID: 6468 - WinDbg 1.2601.12001.0
WinDbg
File
Home
View
Breakpoints
Time Travel
Model
Scripting
Source
Command
Watch
Locals
Registers
Memory
Stack
Parallel Stacks
Disassembly
Threads
Breakpoints
Logs
Notes
Timelines
Modules
Command browser
Windows
Layouts
Reset Windows
Window Layout
Accent color
Workspace
Disassembly
Address: @$scopeip
Follow current instruction
ntdll!DbgBreakPoint: CFG
00007ffb`d3a810d0 00003ed4 brk #0xF000
00007ffb`d3a810d4 c0035fd6 ret
ntdll!DbgBreakPointWithStatusEnd: CFG
00007ffb`d3a810d8 00000000 ???
00007ffb`d3a810dc 00000000 ???
ntdll!DebugPrint: CFG
00007ffb`d3a810e0 e30302aa mov x3, x2
00007ffb`d3a810e4 e20301aa mov x2, x1
00007ffb`d3a810e8 01004079 ldrh w1, [x0]
00007ffb`d3a810ec 000440f9 ldr x0, [x0, #8]
00007ffb`d3a810f0 300080d2 mov xip0, #1
00007ffb`d3a810f4 40003ed4 brk #0xF002
00007ffb`d3a810f8 00003ed4 brk #0xF000
00007ffb`d3a810fc c0035fd6 ret
ntdll!DebugPrompt: CFG
00007ffb`d3a81100 23044079 ldrh w3, [x1, #2]
00007ffb`d3a81104 220440f9 ldr x2, [x1, #8]
00007ffb`d3a81108 01004079 ldrh w1, [x0]
00007ffb`d3a8110c 000440f9 ldr x0, [x0, #8]
00007ffb`d3a81110 500080d2 mov xip0, #2
00007ffb`d3a81114 40003ed4 brk #0xF002
00007ffb`d3a81118 00003ed4 brk #0xF000
00007ffb`d3a8111c c0035fd6 ret
ntdll!#ExInterlockedFlushSList: CFG
00007ffb`d3a81120 08247fc8 ldxp x8, x9, [x0]
00007ffb`d3a81124 1f7c2ac8 stxp w10, xzr, xzr, [x0]
00007ffb`d3a81128 caffff35 cbnz w10, ntdll!#ExInterlockedFlushSList (7ffbd3a81120)
00007ffb`d3a8112c e00309aa mov x0, x9
00007ffb`d3a81130 c0035fd6 ret
00007ffb`d3a81134 00000000 ???
00007ffb`d3a81138 00000000 ???
00007ffb`d3a8113c 00000000 ???
ntdll!#RtlpInterlockedPopEntrySList_entry_thunk: CFG
00007ffb`d3a81140 fd7bbfa9 stp fp, lr, [sp, #-0x10]!
00007ffb`d3a81144 fd030091 mov fp, sp
Registers
X0: 0000000000000000 X1: 00007FFBD3A98190 X2: 0000000000000000
X3: 0000000000000000 X4: 0000000000000000 X5: 0000000000000000
X6: 0000000000000000 X7: 0000000000000000 X8: 0000000000000000
X9: 0000000000000008 X10: 0000000000000001 X11: 0000000000000000
X12: 0000000000000000 X13: 0000000000000000 X14: 0000000000000000
X15: 00007FFBD3A98190 X16: 00007FFBD3A10638 X17: 0000000000000015
X18: 0000000000000000 X19: 00007FFBD3A98190 X20: 0000000000000000
X21: 0000000000000000 X22: 0000000000000000 X23: 0000000000000000
X24: 0000000000000000 X25: 0000000000000000 X26: 0000000000000000
X27: 0000000000000000 X28: 0000000000000000 FP: 000000531E0FFE80
LR: 00007FFBD3A1068C SP: 000000531E0FFE80 PC: 00007FFBD3A810D0
CPSR: 0000000040000040 ELR: 0000000000000000 SPSR: 0000000000000000
LastErrorValue: 0x00000000
LastStatusValue: 0x00000000
Command
0:015>
Stack
Frame Index Call Site Child-SP Return Address
[0x0] ntdll!DbgBreakPoint 0x531e0ffe80 0x7ffbd3a10...
[0x1] ntdll!DbgUiRemoteBreakin+0x54 0x531e0ffe80 0x7ffbd23b84a8
[0x2] KERNEL32!BaseThreadInitThunk+0x38 0x531e0ffe90 0x7ffbd39a3178
[0x3] ntdll!RtlUserThreadStart+0x48 0x531e0ffed0 0x0
Threads
Stack
Breakpoints

Similar things available for gdb

This is GDB dashboard
(I haven't used it, but it looks great)



Debugging Benefits

- Sometimes easier to just see what code does
- Unpacking
  - just let the code unpack itself and debug as normal
- Most debuggers have in-built disassemblers anyway
  - And sometimes vice versa: IDA Pro and Ghidra support dynamic analysis via debugging (I’ve had limited success with this but that may be due to the unusual targets I’ve been debugging)
- Can always combine static and dynamic analysis

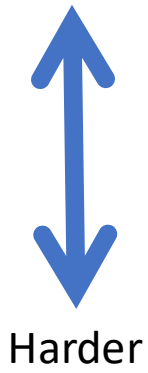
Difficulties

- We are now executing potentially malicous code
  - use an isolated virtual machine
- Anti-Debugging
  - detect debugger and [exit | crash | modify behavior]
  - IsDebuggerPresent(), INT3 scanning, timing, VM-detection, pop ss trick, etc., etc., etc.
  - Anti-Anti-Debugging can be tedious

Commonality of evasion

- Detect evidence of monitoring systems
  - Fingerprint a machine/look for fingerprints
- Hide real malicious intent if necessary
  - IF VM_PRESENT() or DEBUGGER_PRESENT()
    - Terminate() *// hide real intent*
  - ELSE
    - Malicious_Behavior() *//real intent*

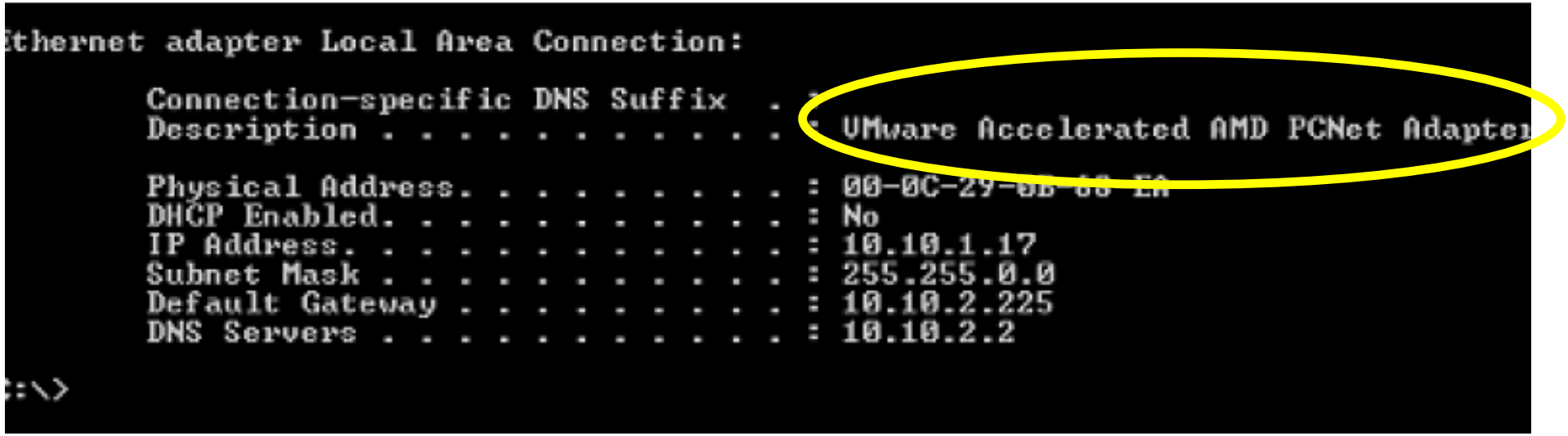
Taxonomy of malware evasion



Layer of abstraction	Examples
Application	Installation, execution
Hardware	Device name, drivers
Environment	Memory and execution artifacts
Behavior	Timing

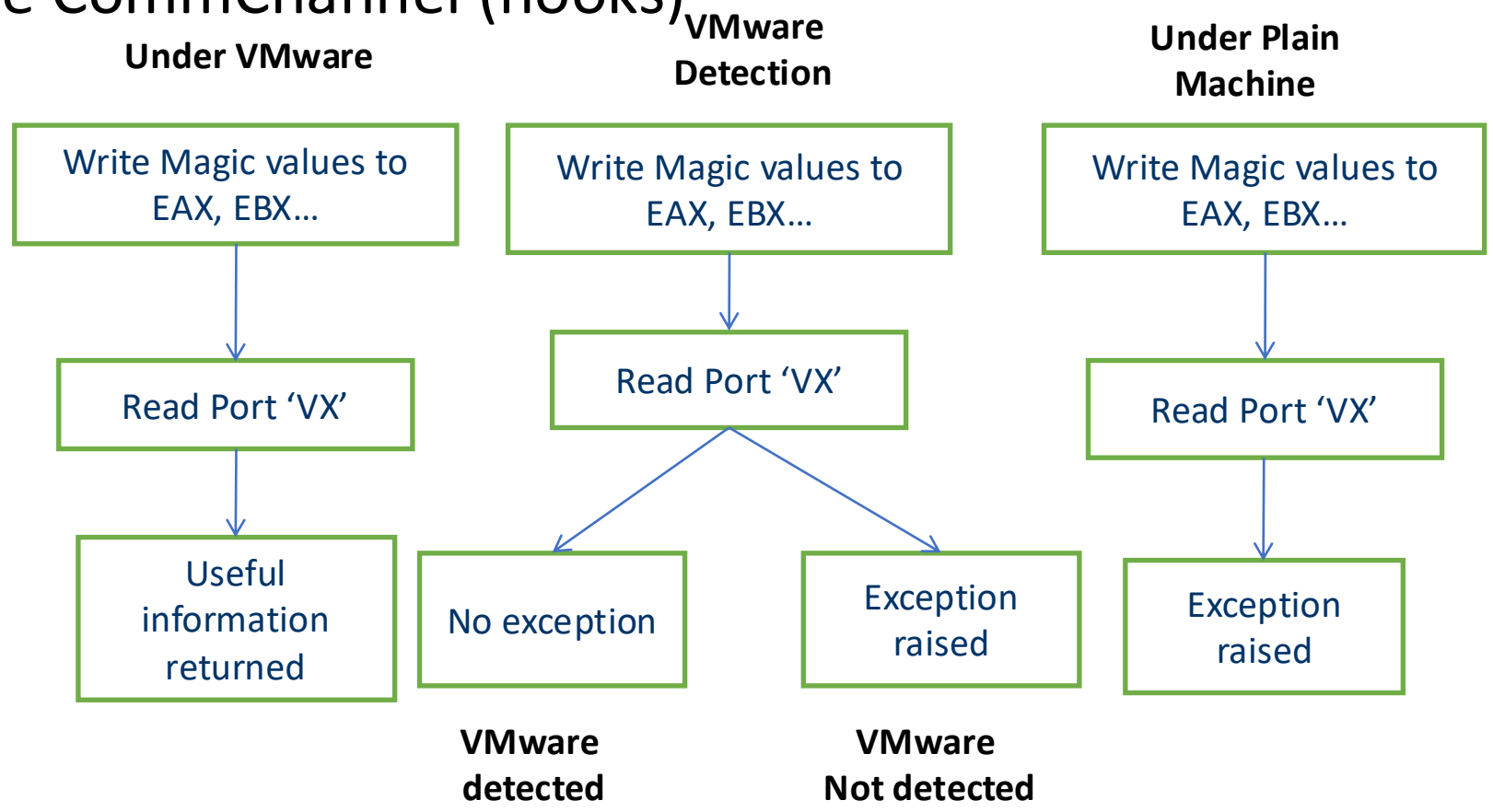
Example 1

- Device driver strings
  - Network cards



Example 2

- VMWare CommChannel (hooks)

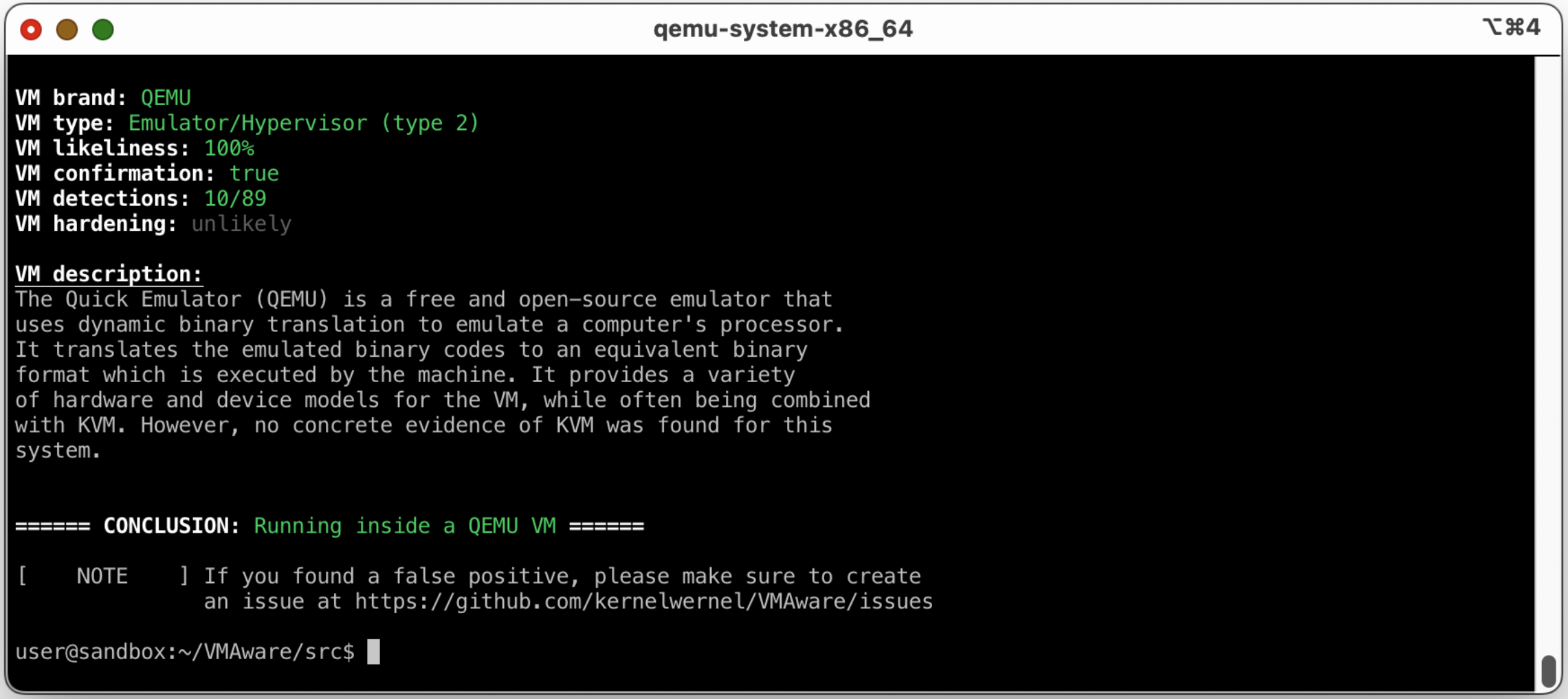


VMware detection code

```
MOV     EAX, 0x564D5868   ; 'VMXh'
MOV     EBX, 0            ; Any value but not the MAGIC VALUE
MOV     ECX, 0x0A         ; Get VMWare version
MOV     EDX, 0x5658       ; 'VX' (port number)
IN      EAX, DX           ; Read port
CMP     EBX, 0x564D5868   ; Is there a reply from VMWare? 'VMXh'
```

https://www.aldeid.com/wiki/VMXh-Magic-Value

VMAware under QEMU



VMAware under Parallels

Prevalence of evasion

- ***40%*** of malware samples exhibit fewer malicious events with debugger attached
- ***4.0%*** exhibit fewer malicious events under VMware execution