CROWDSTRIKE

Visualizing Page Tables

… for Local Exploitation: Hacking Like in the Movies

Alexandru Radocea

- Developer at CrowdStrike, Inc.
  - iOS internals fan
  - Recovering software security assessor
  - Likes bringing pain to the adversary
- @defendtheworld on Twitter

Alexandru Ionescu

- Reverse Engineered Windows kernel since 1999
  - Lead kernel developer at ReactOS
- Coauthor of Windows Internals
- Founded Winsider Seminars & Solutions Inc. to provide services and training to various organizations
- Interned at Apple for a bunch of years
- Now chief architect at CrowdStrike

Georg Wicherski

- Researcher at CrowdStrike, Inc.
  - x86 & ARM low-level stuff
  - Reverse Engineering, Malware analysis
  - Exploitation and Mitigation research
- @ochsff on Twitter
- http://blog.oxff.net/

CROWDSTRIKE

Introduction

Paging 101

- Translation from virtual addresses to physical
  - Virtual address: the pointers the CPU works with
  - Physical address: the actual address of a memory cell in the physical RAM chip, only used by MMU and DMA
- Virtual address unique per virtual memory space
  - Usually means per process for usermode, one shared kernel space for all processes

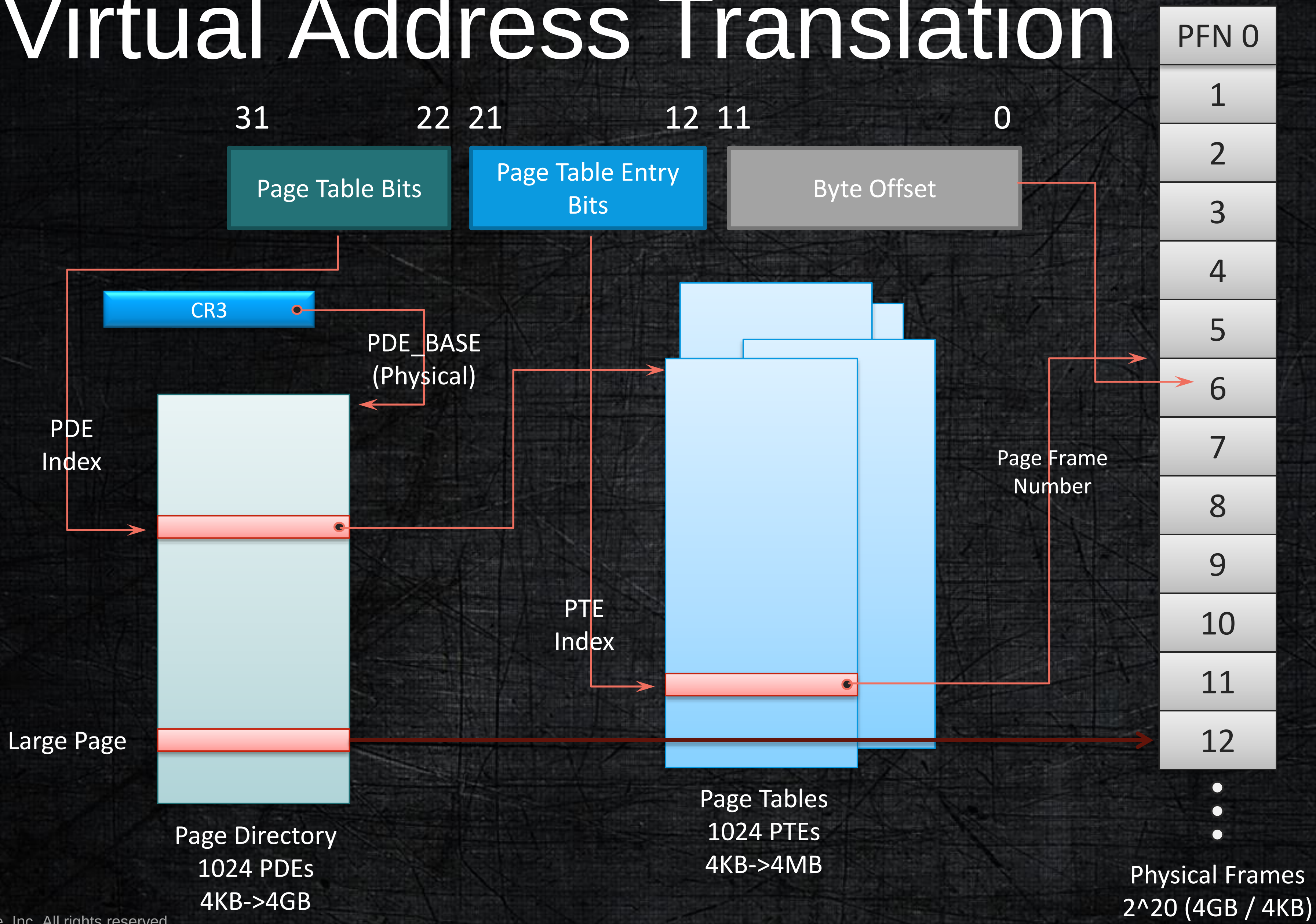
Efficient Hardware Implementation

- Group addresses into pages: block of addresses that are translated in the same way
- Cache translation results: TLB
- Hierarchical translation tables (trees) to conserve memory
  - Two levels on x86, three levels on PAE, four on x86_64
  - Two levels on ARMv7-A, three levels with LPAE

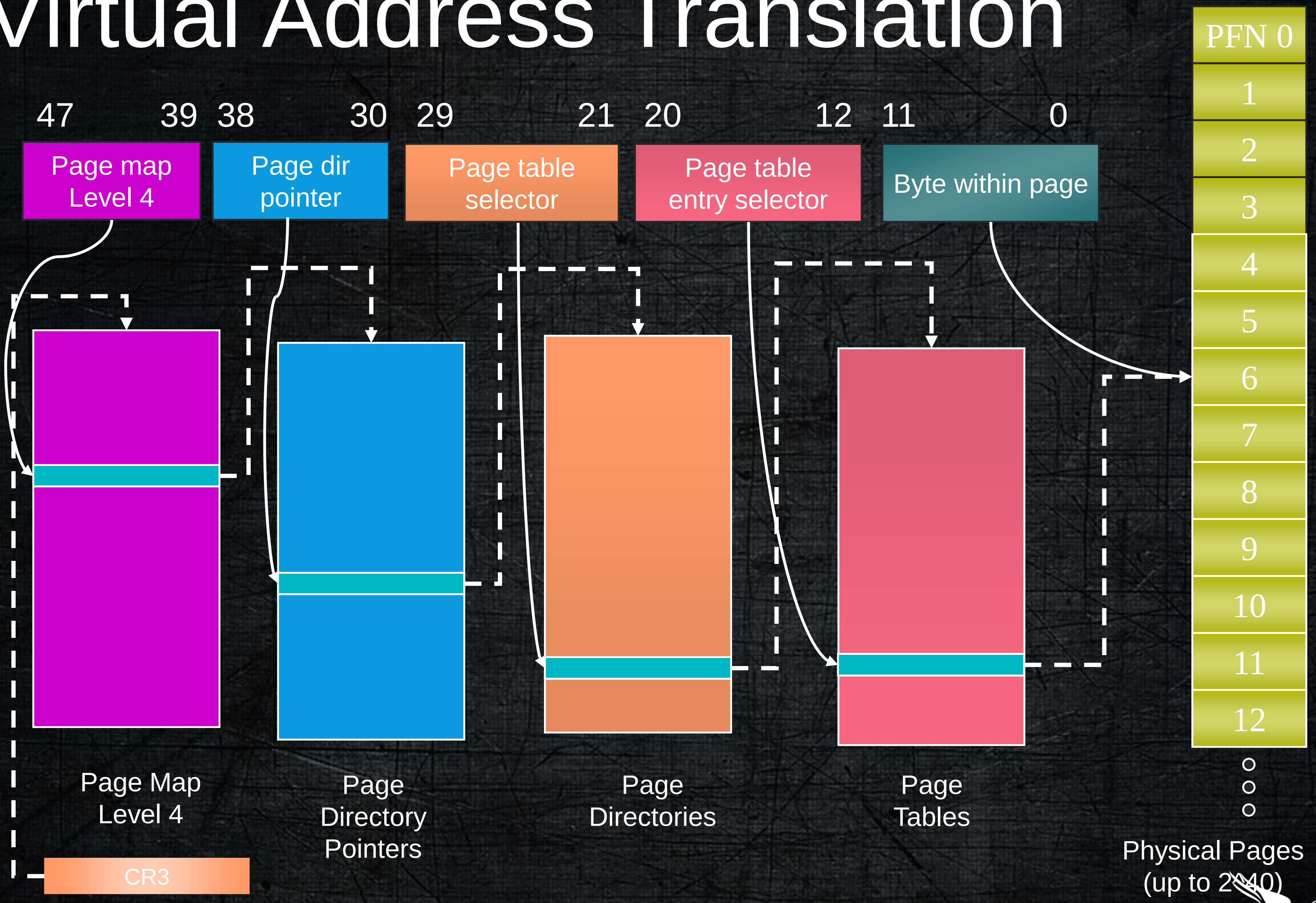
Memory Protections

- Memory protections implemented on top of paging
  - Read-only vs. read-write memory areas
  - Executable vs. data-only memory areas
    - x86_64: NX (No-eXecute) bit per page; SMEP
    - ARM: XN (eXecute-Never) bit per page; PXN
  - Privilege level to access page
    - ARM: Supervisor bit, Domains, different table sets
    - x86: Supervisor bit (CPL, SMAP)

x86 Virtual Address Translation



x64 Virtual Address Translation



What a Movie Hacker Looks for

- Mappings at repeatedly constant addresses
  - Constant physical address: Subject to reliable external physical attacks
  - Constant virtual address: ASLR bypass
- Mappings with unexpected protections
  - Read-write but not NX/XN: Classical copy shellcode and execute scenario
  - Driver specific weirdness (DMA memory, …)

CROWDSTRIKE

Background and Methodology

Data Collection

- Android: Both custom kernel and local exploit
- iOS: Custom driver for jailbroken device
- x86_64 Linux: Custom kernel module
- x86_64 OS X: Custom kernel extension
- Windows RT (ARM): Crash dumps
- Windows 8 x86_64: Crash dumps & custom driver
- Windows 8.1, x86_64: Crash dumps

Hilbert Curve

- Space Filling Curve
  - Well known from visualizing IP address space
  - Adapted to visualize memory address space
- Adapted scurve implementation
  - Aldo Cortesi http://corte.si/
  - Simple glue code to parse data collection output

Hilbert Curve Legend

- ■ User RO
- ■ Kernel RO
- ■ User RW
- ■ Kernel RW
- ■ User RX
- ■ Kernel RX
- ■ User RWX
- ■ Kernel RWX

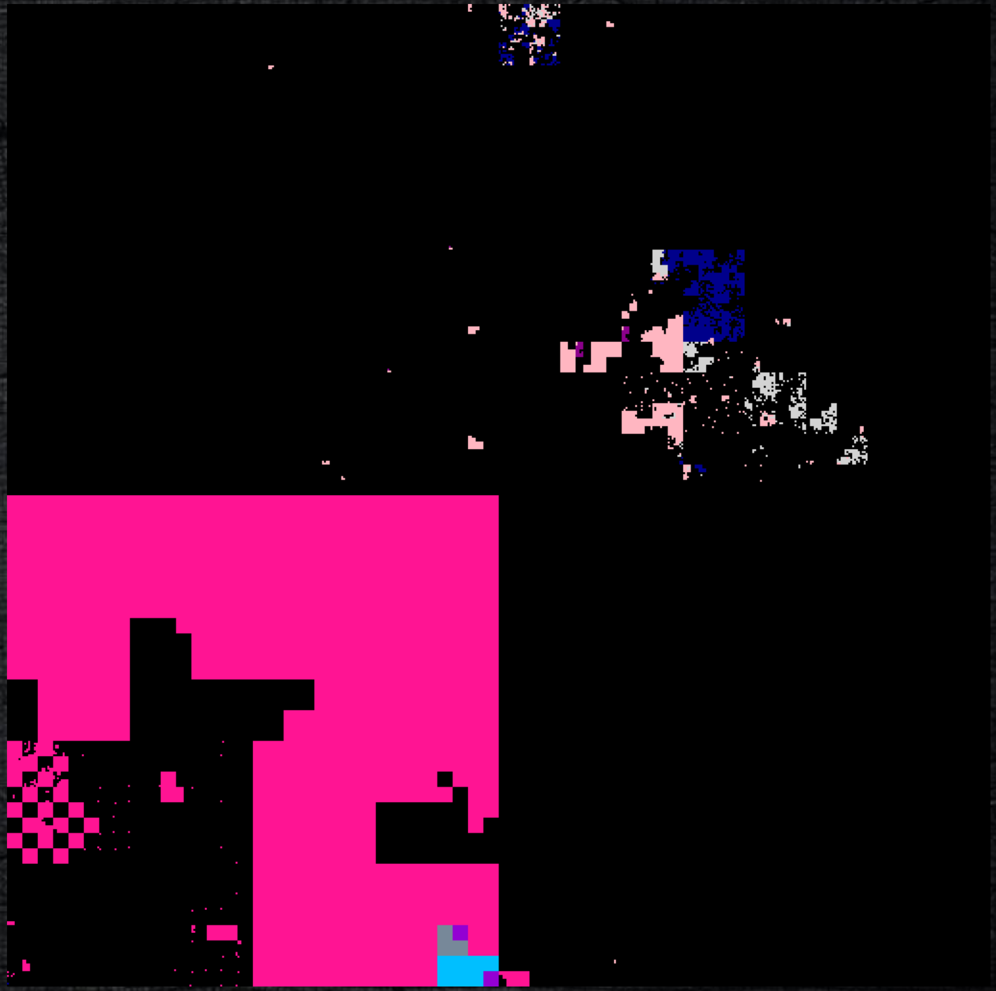
CROWDSTRIKE

Case Studies

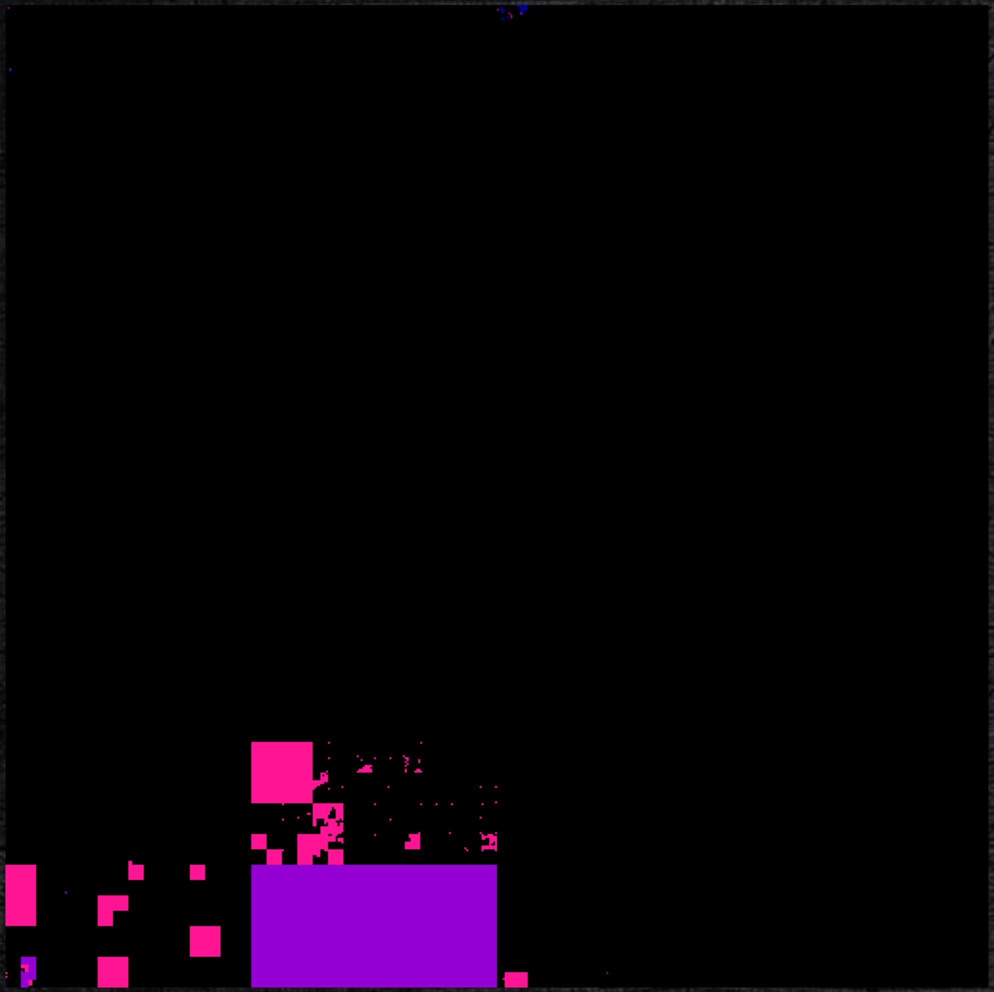
Android

Android Process Comparison

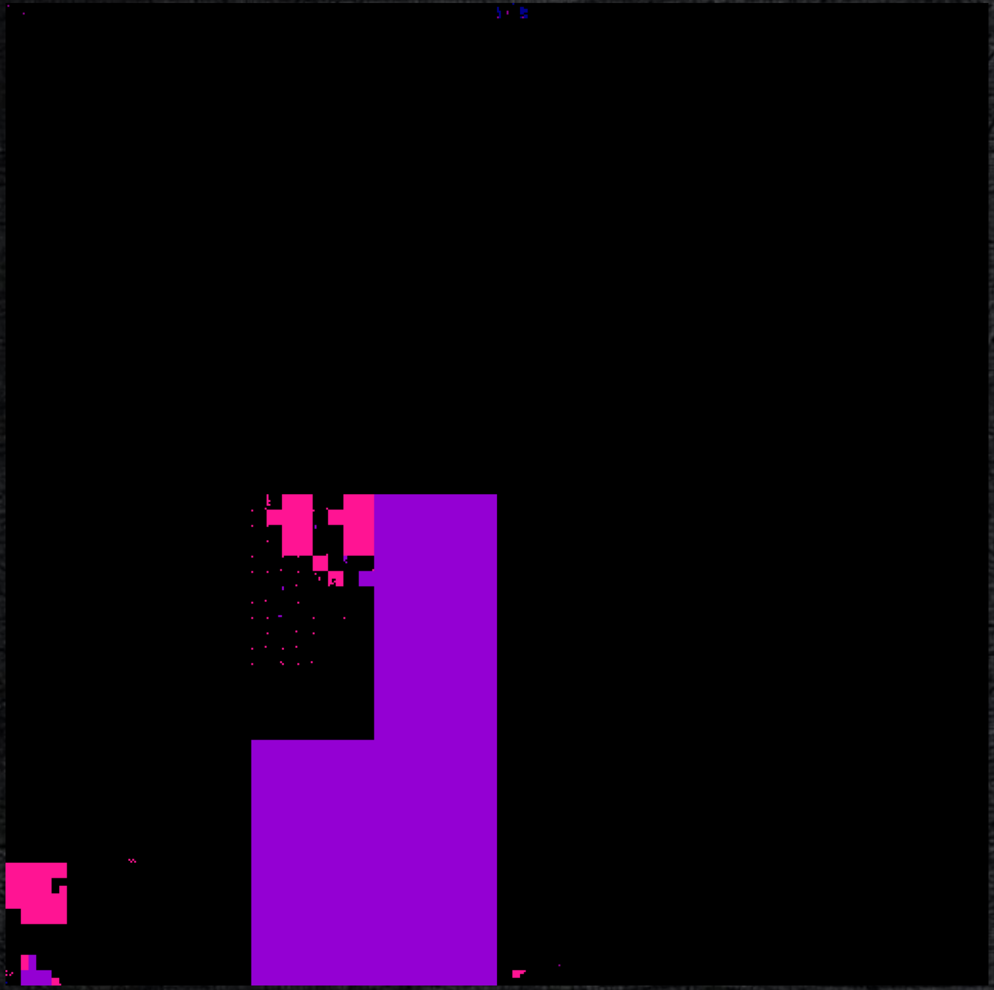
1. init
2. dhcpd
3. zygote
4. com.android.email
5. sandboxed_process0 (Chrome)



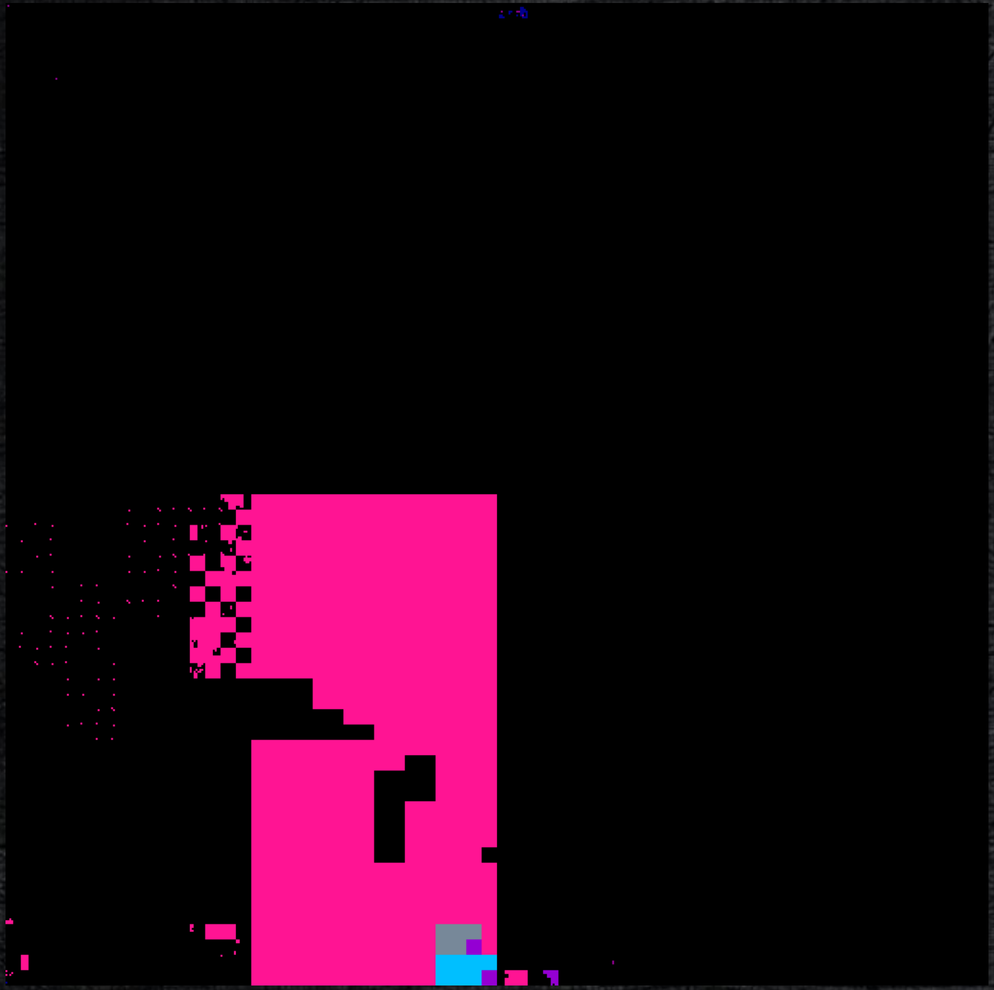
Galaxy Nexus, Android 4.2.2



Nexus 7, Android 4.2.2



Galaxy S4, Android 4.2.2 (MSM)



Android Observations

- Fixed r-x mapping at 0xffff0000 in all processes
  - 0xffff0000 is the ARM exception vectors base address
  - Abused in a *vsyscall* like manner by Linux on ARM
- Kernel `.text` is rwx on almost all kernels
  - `CONFIG_DEBUG_RODATA` not set in kernel configs
  - 3.4.x MSM kernel has RO .text
    - `CONFIG_STRICT_MEMORY_RWX` (Qualcomm)
    - Still has two rwx supervisor sections (1Mb pages)

Android ~~4.2.2~~ 4.3 ASLR Bypass

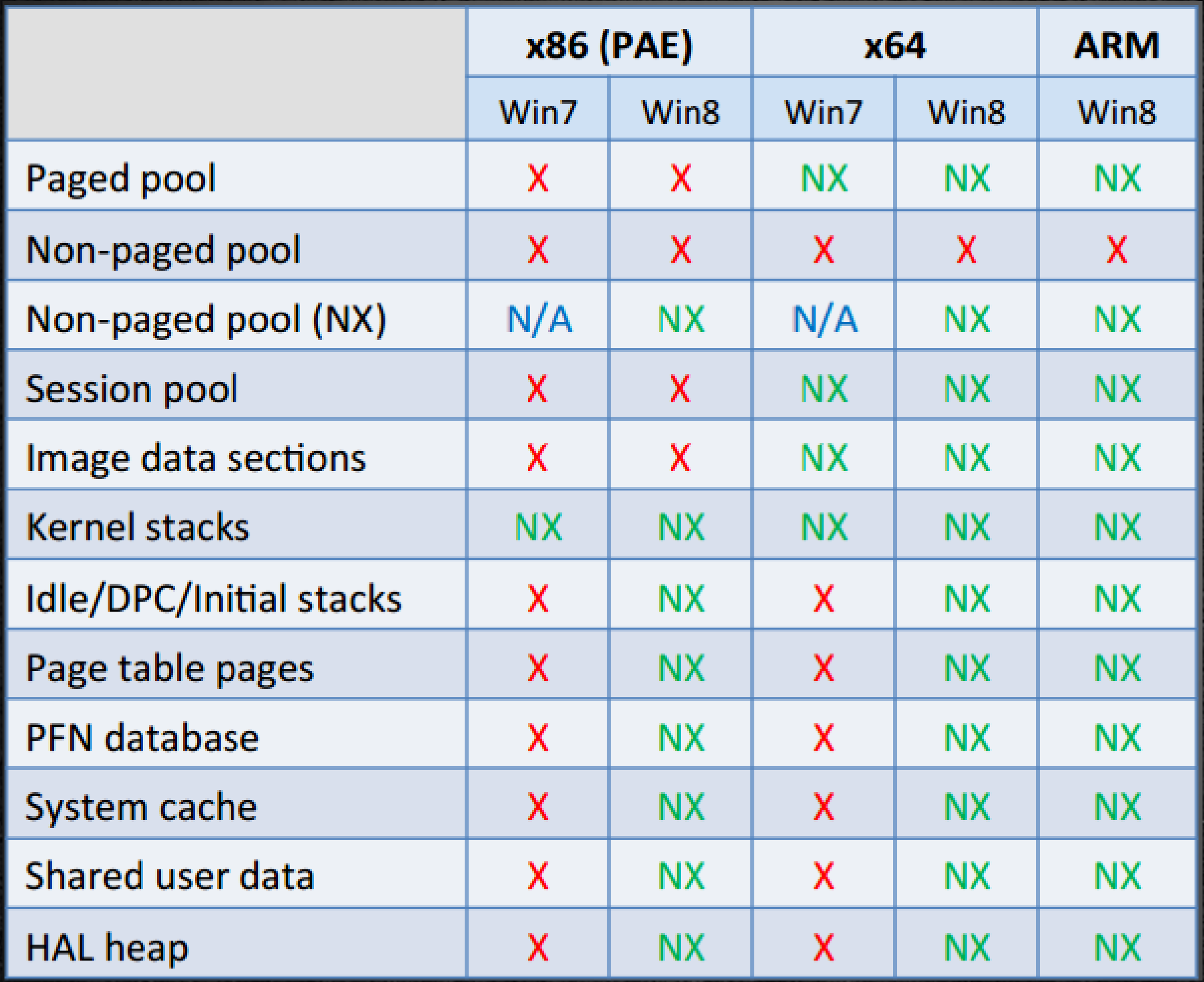
- `__kuser_cmpxchg: @ 0xffff0fc0`
  - `arch/arm/kernel/entry-armv.S`
  - `iff *r2 == r0: *r2 := r1`
  - Bruteforce addresses by invoking a loop, `r0`-`r2` are legitimate register parameters
  - Jump past equality check for arbitrary write gadget
- `__kuser_cmpxchg64: @ 0xffff0f60`
- `ffff0008:   ldr pc, [pc, #1072]   ; 0xffff0440`
  - This leaks the kernel's system call handler address to user-space

Responsible Disclosure for ARM/Linux

- Initially disclosed to Google (Android)
- Quick vectors patch from Russel King, ARM
  - Randomize some code locations in page
  - Only partial solution for user-space helpers
  - Actual vector handlers are branch into adjacent page
  - Fill page with undefined instruction (ARM and Thumb)
- Randomization lacks available entropy at boot
  - Issue being discussed with security@kernel.org

Surface RT

Evolution of Windows Kernel RWX



	x86 (PAE)		x64		ARM
	Win7	Win8	Win7	Win8	Win8
Paged pool	X	X	NX	NX	NX
Non-paged pool	X	X	X	X	X
Non-paged pool (NX)	N/A	NX	N/A	NX	NX
Session pool	X	X	NX	NX	NX
Image data sections	X	X	NX	NX	NX
Kernel stacks	NX	NX	NX	NX	NX
Idle/DPC/Initial stacks	X	NX	X	NX	NX
Page table pages	X	NX	X	NX	NX
PFN database	X	NX	X	NX	NX
System cache	X	NX	X	NX	NX
Shared user data	X	NX	X	NX	NX
HAL heap	X	NX	X	NX	NX

Surface RT

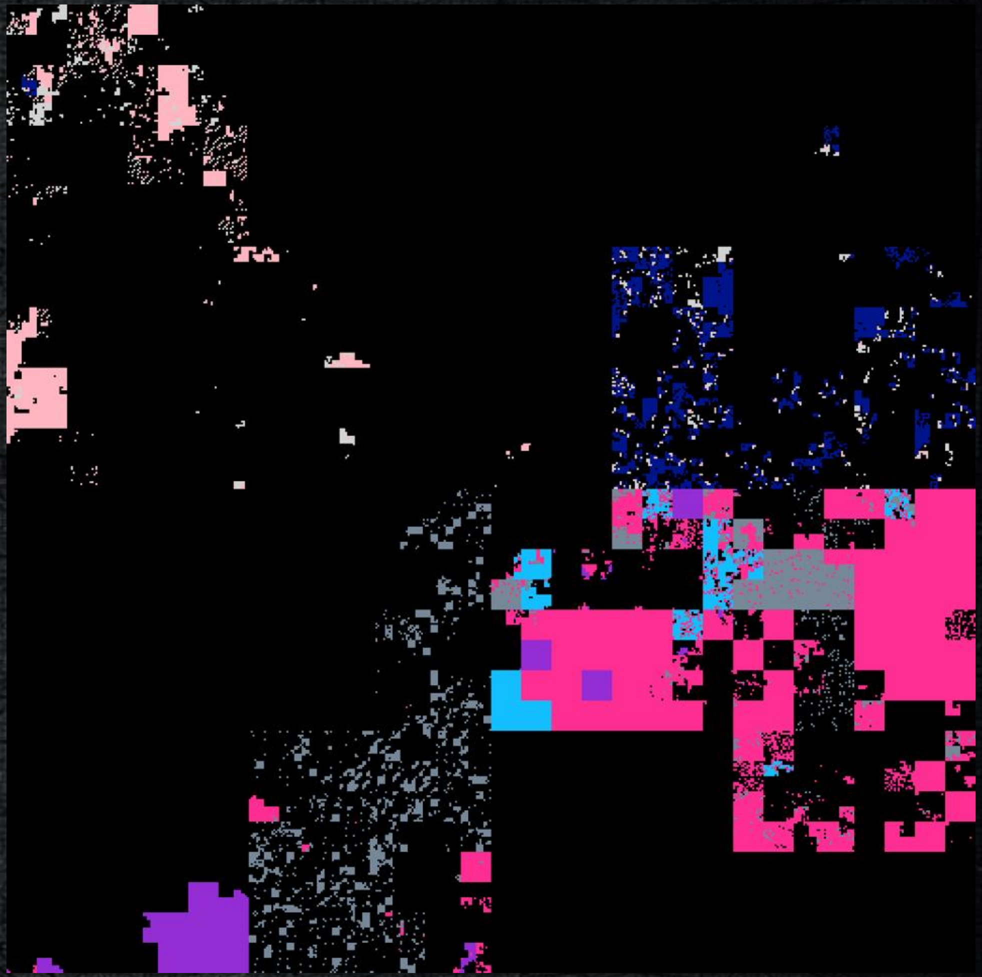
- Runs Windows RT
  - Windows 8 32-bit ARM Kernel
- Locked-down kernel and user-mode signing
  - See “*Windows 8 Mitigations & ARM*” talk at Breakpoint 2012 for more low-level details…
- Suffers from 32-bit address space limitations, but has more x64 mitigations turned on
- Quad-Core Cortex A9, 2GB RAM, 32-128GB Flash

Windows RT & ARM VMSA

- Windows RT uses a VMSA configuration which brings it closest to the x86/x64 model
  - TEX is used to provide 2 software bits
  - AFE is used to provide simplified access model (hardware access bit)
- LPAE and PXN do not appear to be used
- Windows RT PTE format:
  - [0] - XN, [1] - Valid, [2,3] - Caching, [4] - Accessed
  - [5] - Owner, [6] - TEX, [7] - Writable, [8] - Copy-on-Write
  - [9] - !Dirty, [10] - Large, [11] = NonGlobal, [12-31] - Page

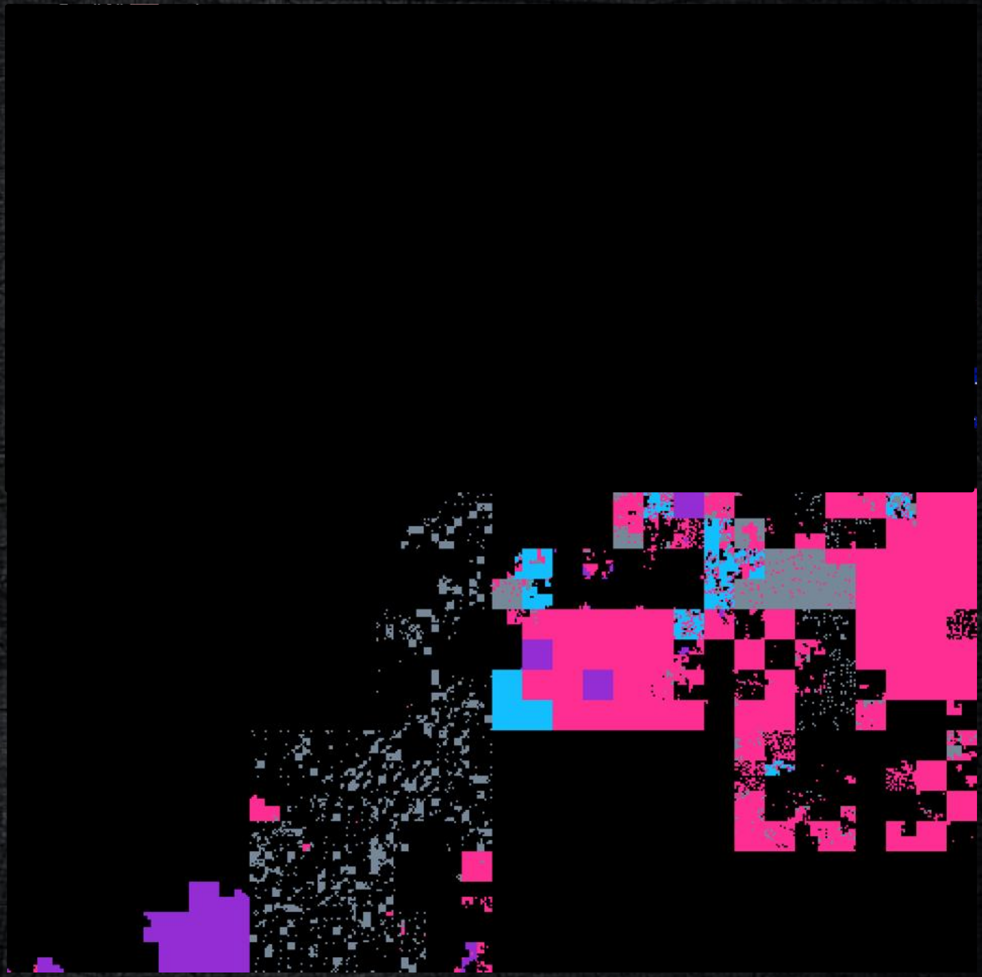
Surface RT Layout

- TTBR0 only
  - TTBR1 not in use
- Standard 32-bit Windows split
  - 0x00000000->0x7FFFFFFF U
  - 0x80000000->0xFFFFFFFF K
- Physical memory starts at 0x80000000
- 256 ASIDs used for TLB perf



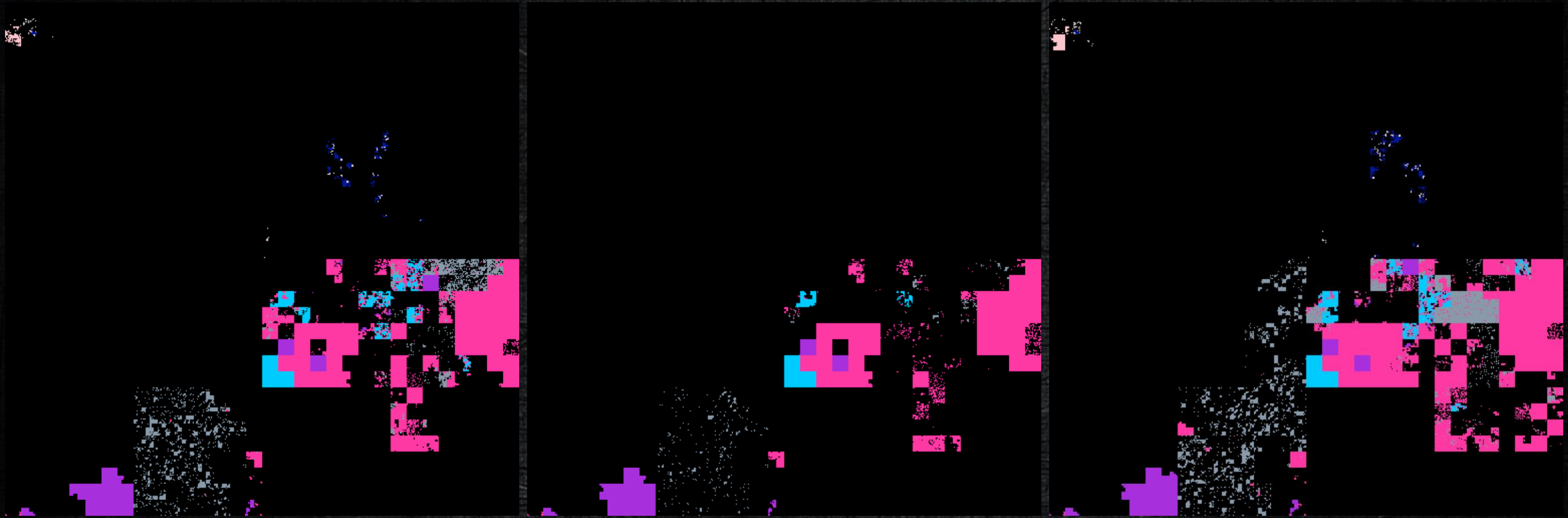
Kernel Observations

- Kernel ASLR has difficulties dueto limited address space
- Limited Kernel W^X
- Vector Page is RWX (kind of)
- I/O mappings are RWX
- Very similar layout to x86, but less RWX pages due to NonPagedPoolNx being default



CROWDSTRIKE

Two devices side by side: Virtual view

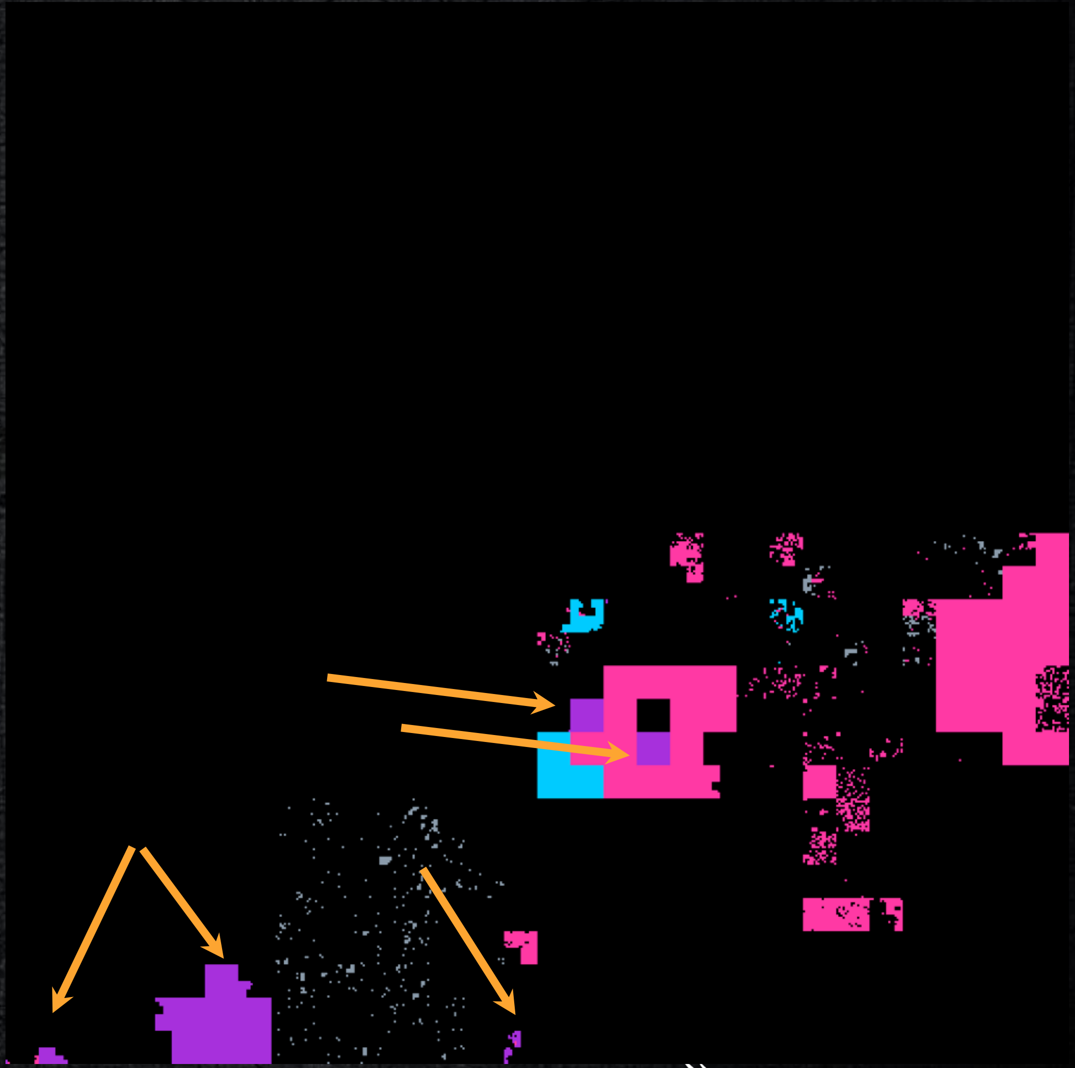


Two devices side by side: Virtual view

- Fixed IO mappings (*MmMapIoSpace*) and usually RWX
  - Locality: *MmMapIoSpace* does not have randomization enabled, and I/O mappings are usually done quite early at boot
  - Permissions: Windows 8 introduced hacky (backward-compatible) way of asking for W^X I/O Mappings, likely not used across the board
- Fixed kernel code and RWX kernel memory was seen
  - We don’t believe fixed data was seen, however (TBD)
- Userland ASLR seems very strong: nothing static seen

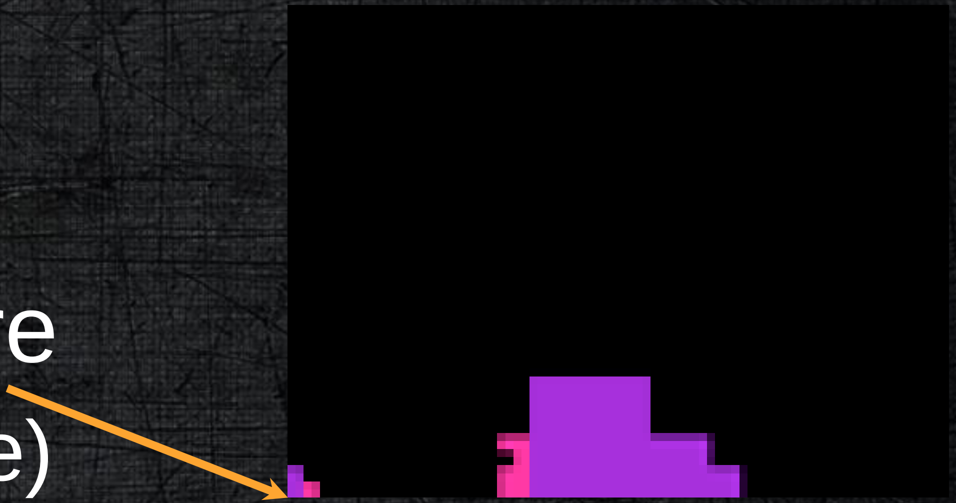
Where to “snap-in” a kernel payload

- Local exploits
  - No dereference protection
- Remote exploits
  - Fixed, RWX memory
  - Vector page and plenty more



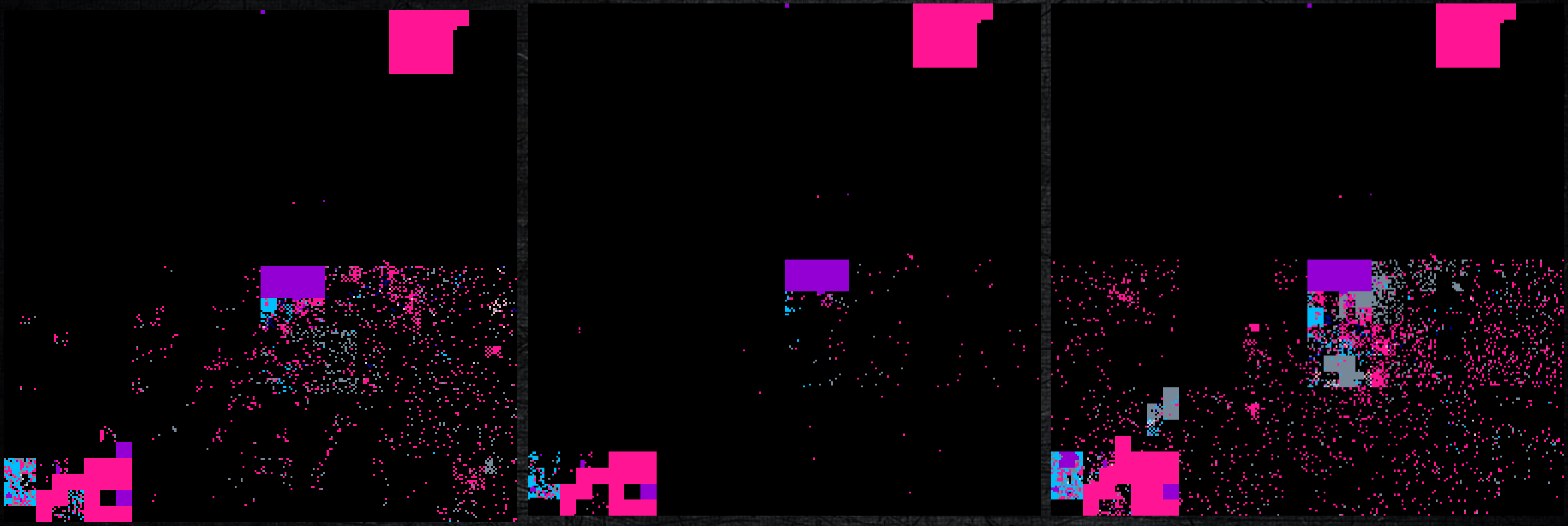
ARM Vector page

- Handles interrupts
  - 0x00000000 or 0xFFFF0000
  - No requirement for RWX
- Windows has a mapping there
  - Kernel Read Execute (no Write)



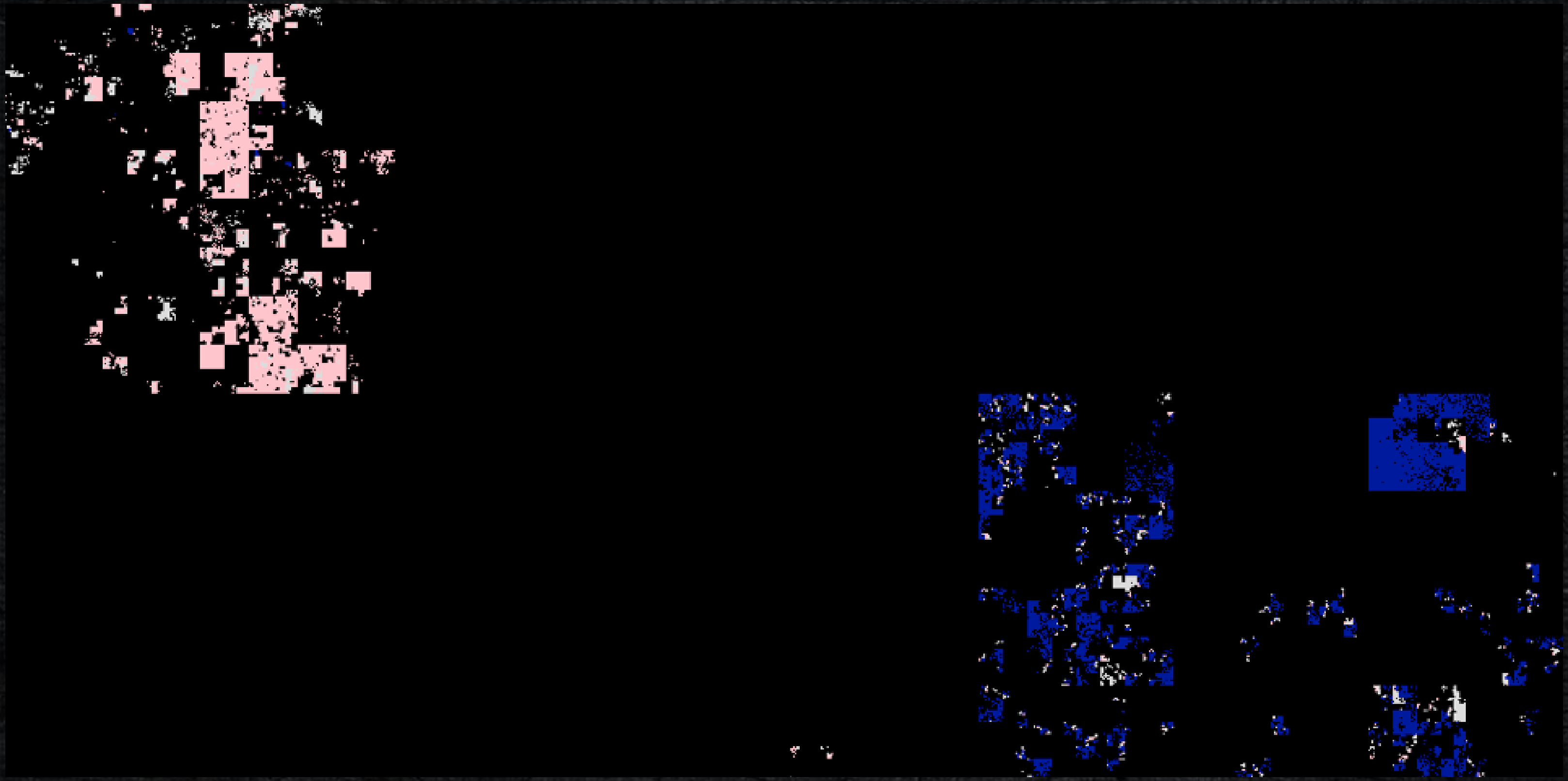
- The region is all zeroes however…
- Windows uses VBAR to randomize vector table ☺

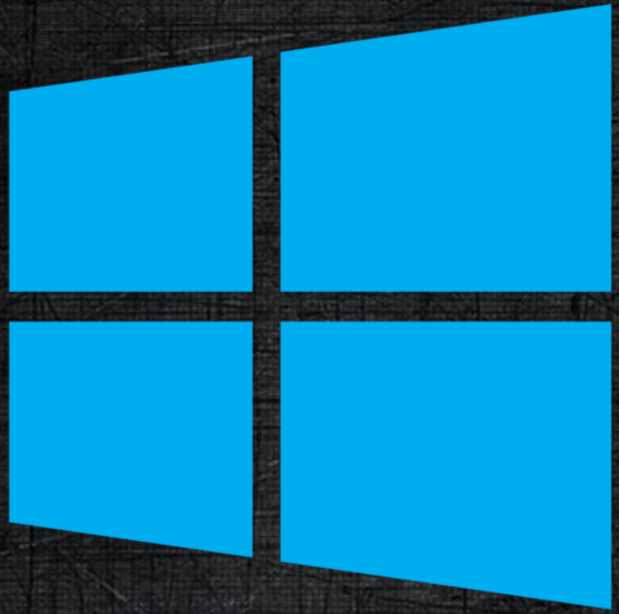
Two devices side by side: Physical view



Internet Explorer

- No obvious RWX JIT region





Windows® 8

64-bit Kernel Address Space

Observations

Windows® Blue
TM

64-bit Kernel Address Space

Observations

iOS 6
The world's most advanced mobile operating system.

Dumping iOS

- Pictures depict samples from an iPhone 4S
- Injected kernel code
- Modified evas10n exploit to undo pagetable tomfoolery

iOS Layout

- TTBR Split depends on device RAM
- TTBR0 swapping on context switch

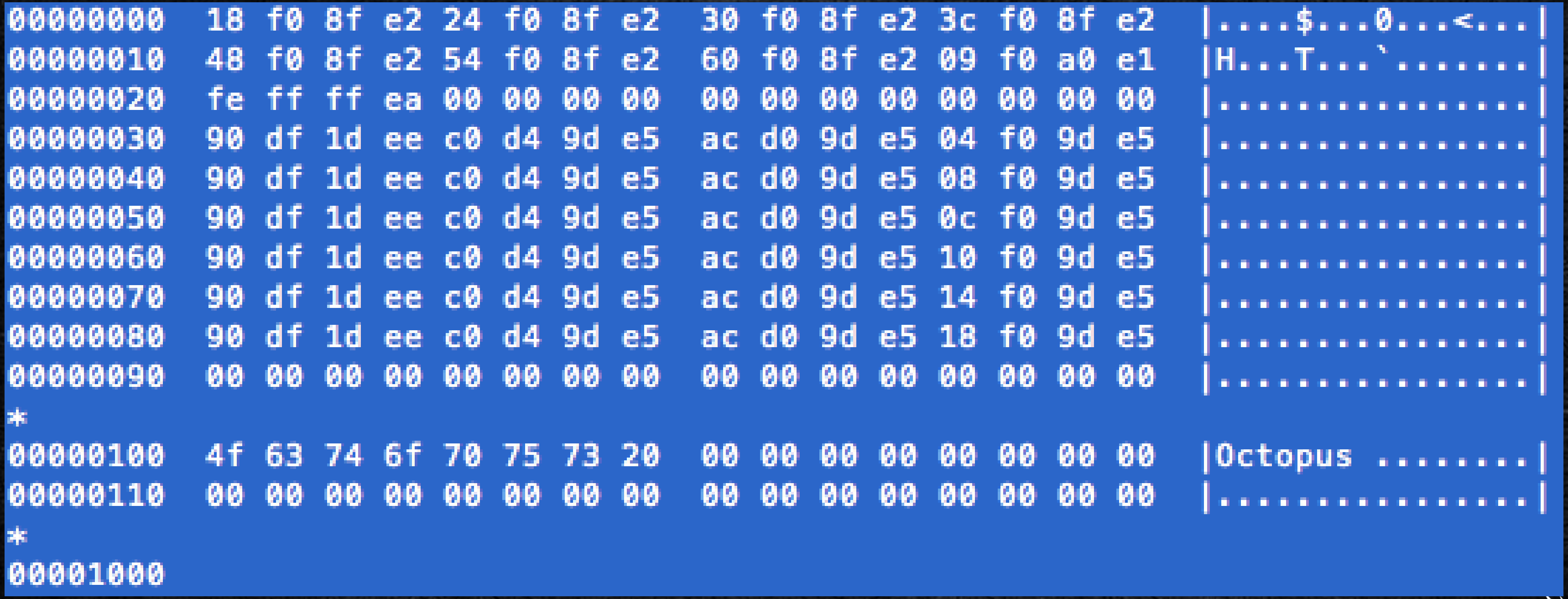
- No shared address space

iOS 6 Security Properties

- Userland
  - Per-boot randomization (shared cache)
  - Per-execution randomization (dyld, .text, stack, heap)
  - Heap and stack separately randomized
  - W^X

iOS 6 Vector page

- Execute supervisor, read only
- No obviously useful ROP primitives

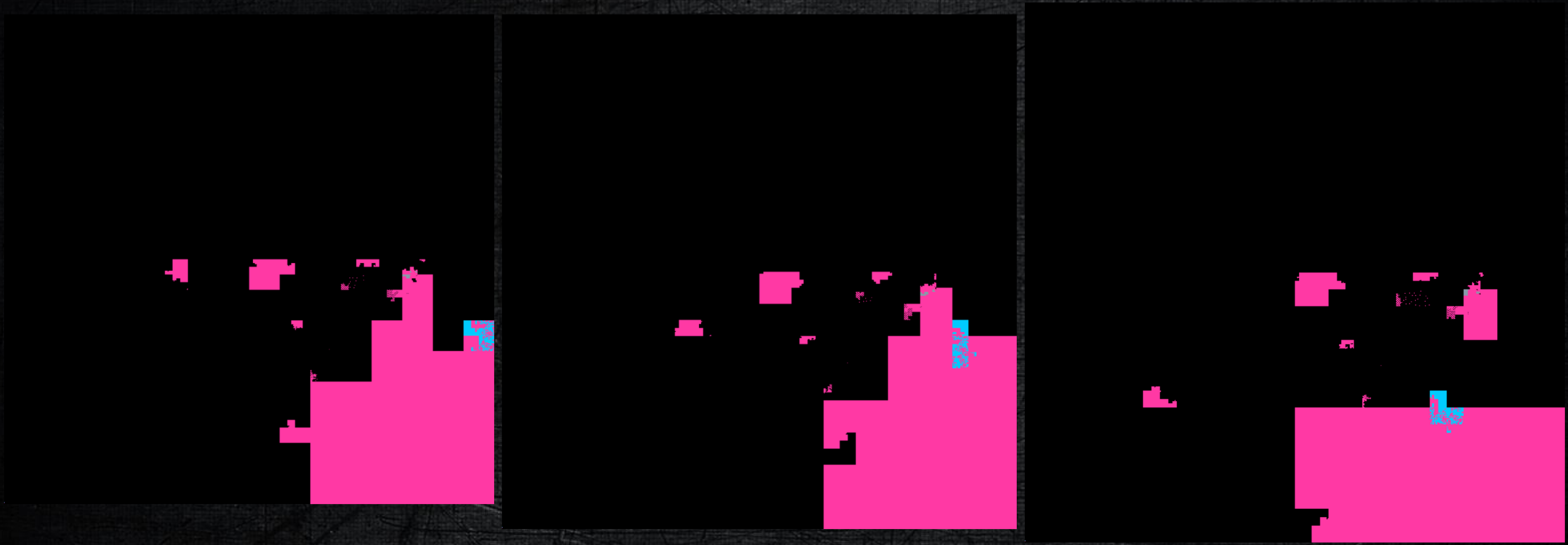


```
00000000  18 f0 8f e2 24 f0 8f e2  30 f0 8f e2 3c f0 8f e2  |....$...0...<...|
00000010  48 f0 8f e2 54 f0 8f e2  60 f0 8f e2 09 f0 a0 e1  |H...T...`.......|
00000020  fe ff ff ea 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000030  90 df 1d ee c0 d4 9d e5  ac d0 9d e5 04 f0 9d e5  |................|
00000040  90 df 1d ee c0 d4 9d e5  ac d0 9d e5 08 f0 9d e5  |................|
00000050  90 df 1d ee c0 d4 9d e5  ac d0 9d e5 0c f0 9d e5  |................|
00000060  90 df 1d ee c0 d4 9d e5  ac d0 9d e5 10 f0 9d e5  |................|
00000070  90 df 1d ee c0 d4 9d e5  ac d0 9d e5 14 f0 9d e5  |................|
00000080  90 df 1d ee c0 d4 9d e5  ac d0 9d e5 18 f0 9d e5  |................|
00000090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000100  4f 63 74 6f 70 75 73 20  00 00 00 00 00 00 00 00  |Octopus ........|
00000110  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001000
```

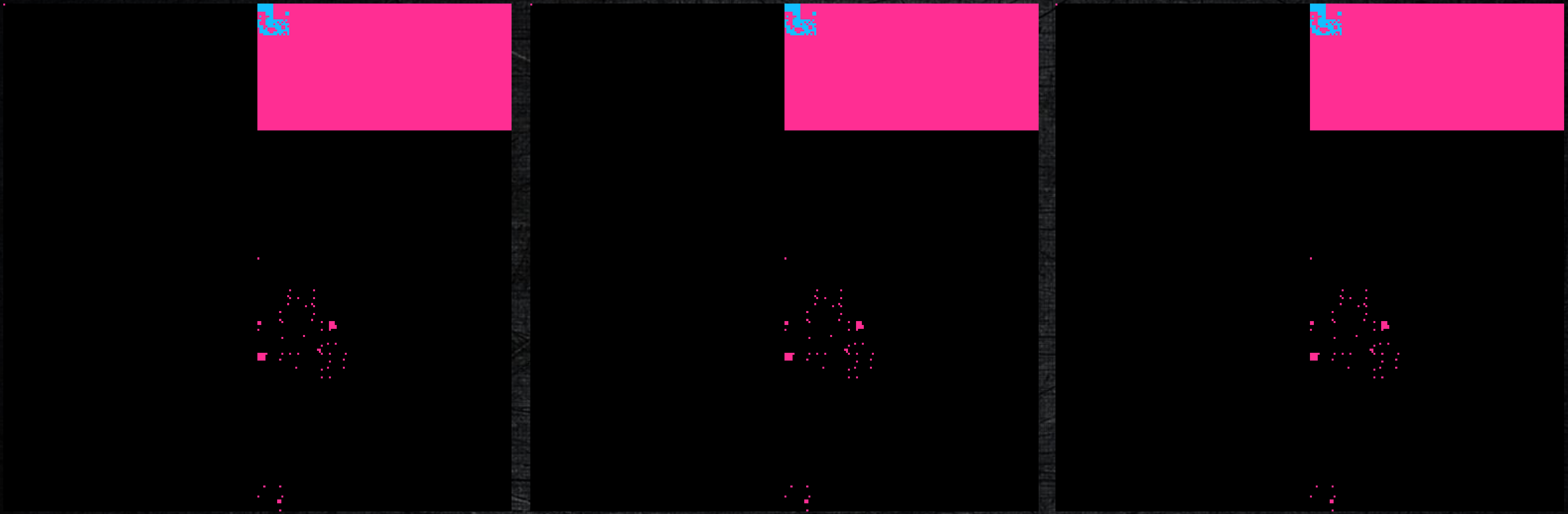
iOS 6 0xffff1000

- User readable, supervisor writable page adjacent to vector table
- Available in every process
- Time, performance counters (sidechannel?)

KASLR: Kernel Code Peekaboo

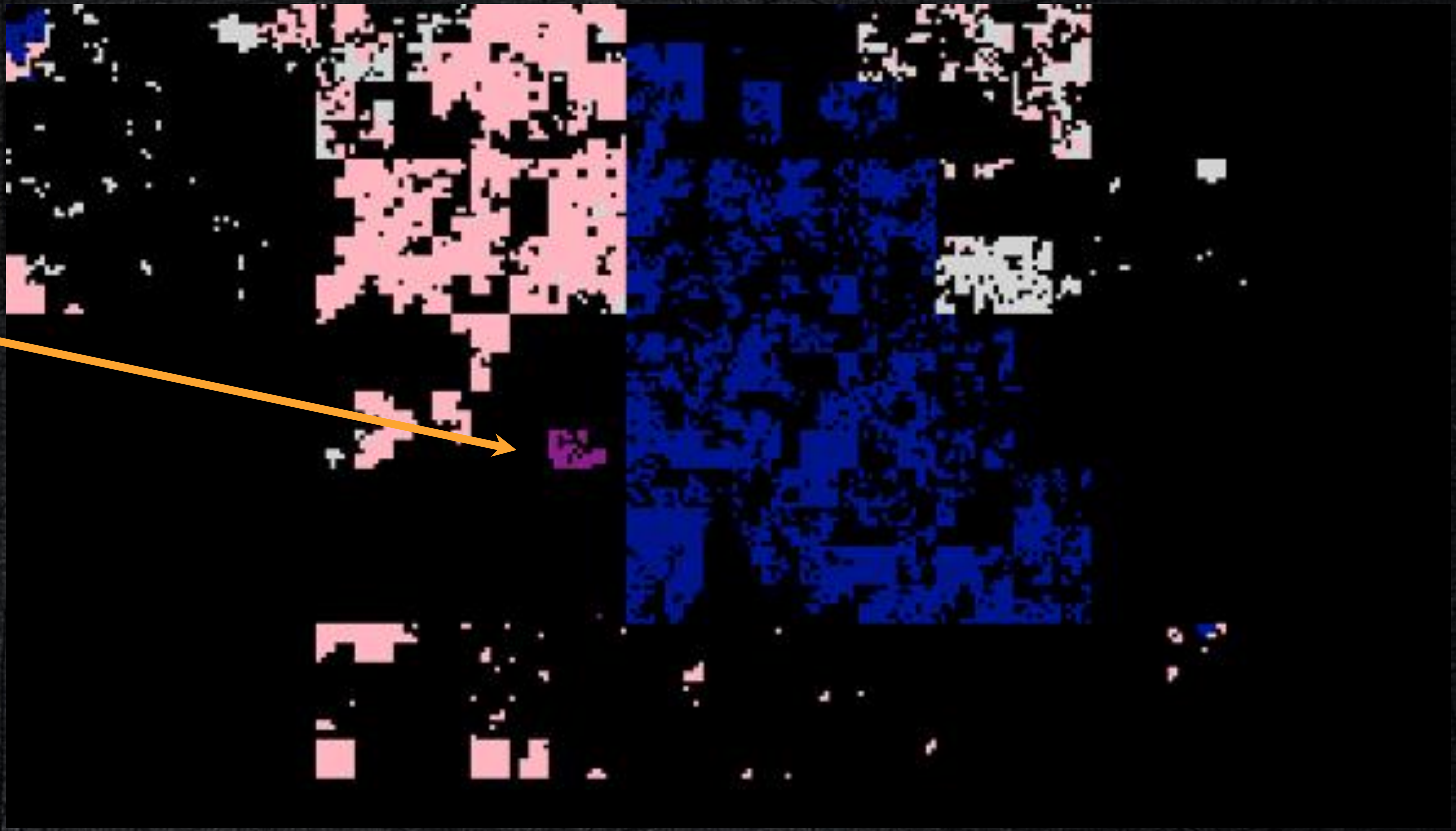


Physical View (Three Captures)



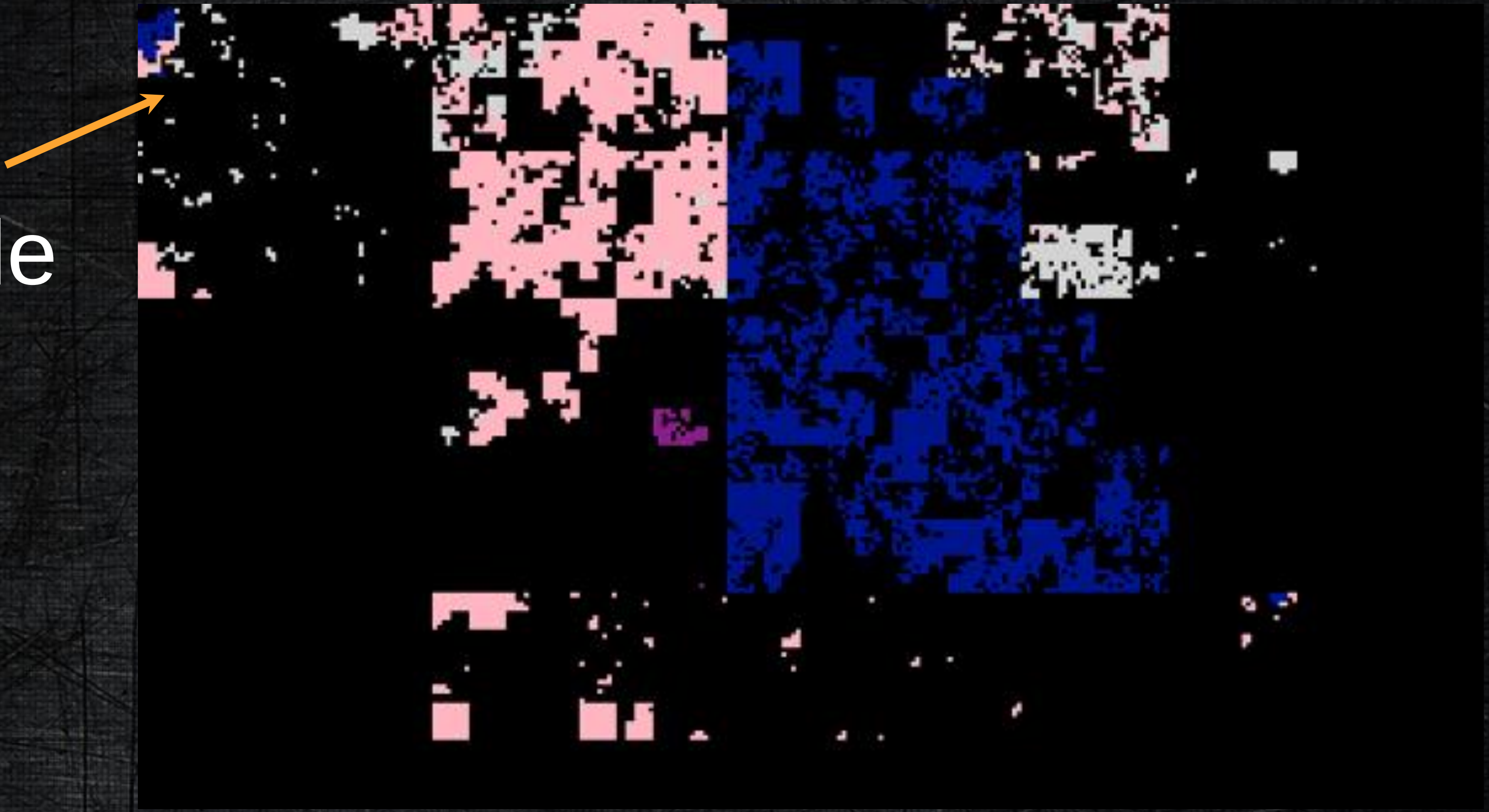
iOS: Example process (MobileSafari)

- JIT (1.2MB)



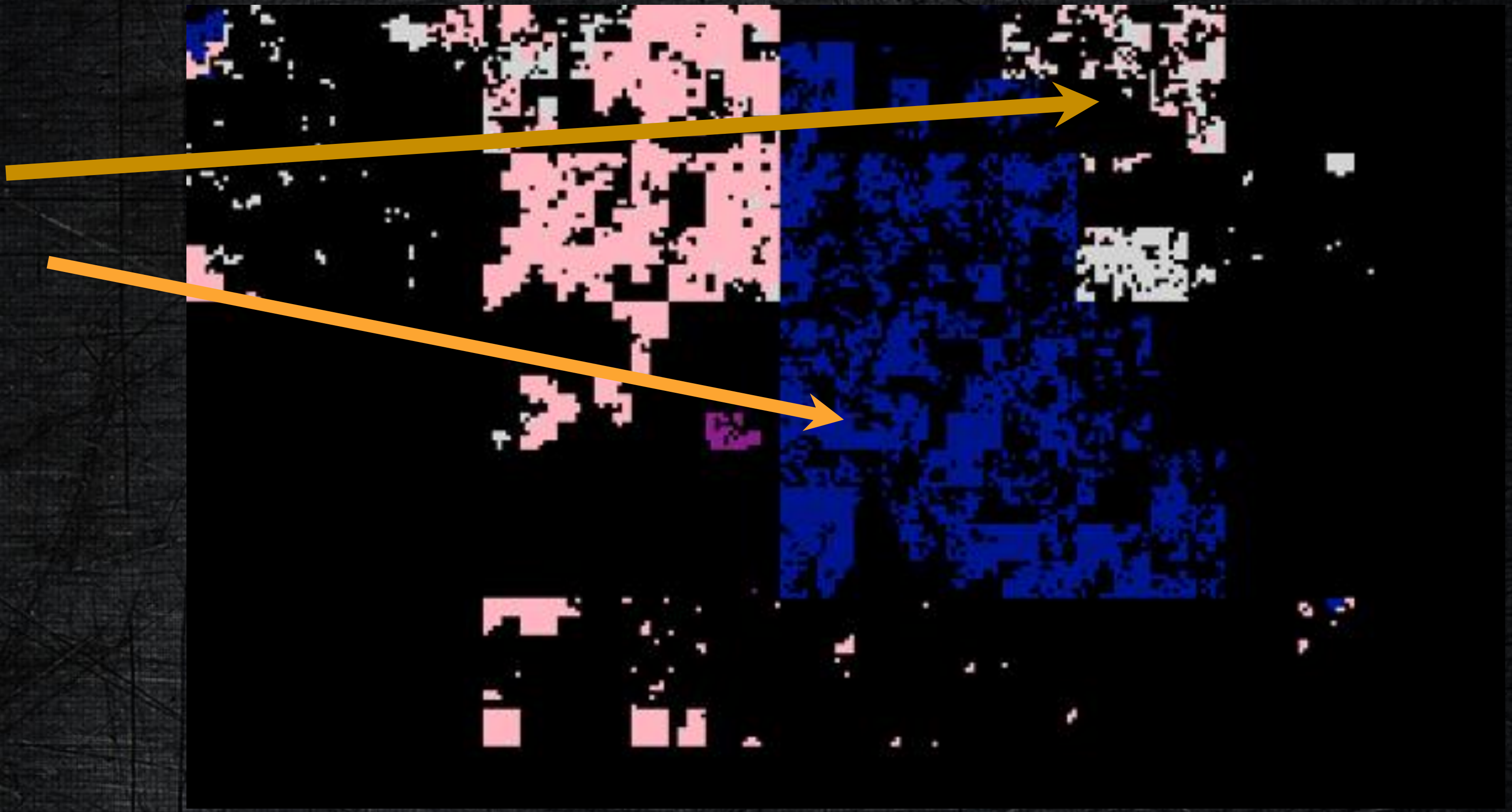
iOS: Example process (MobileSafari)

- Main executable



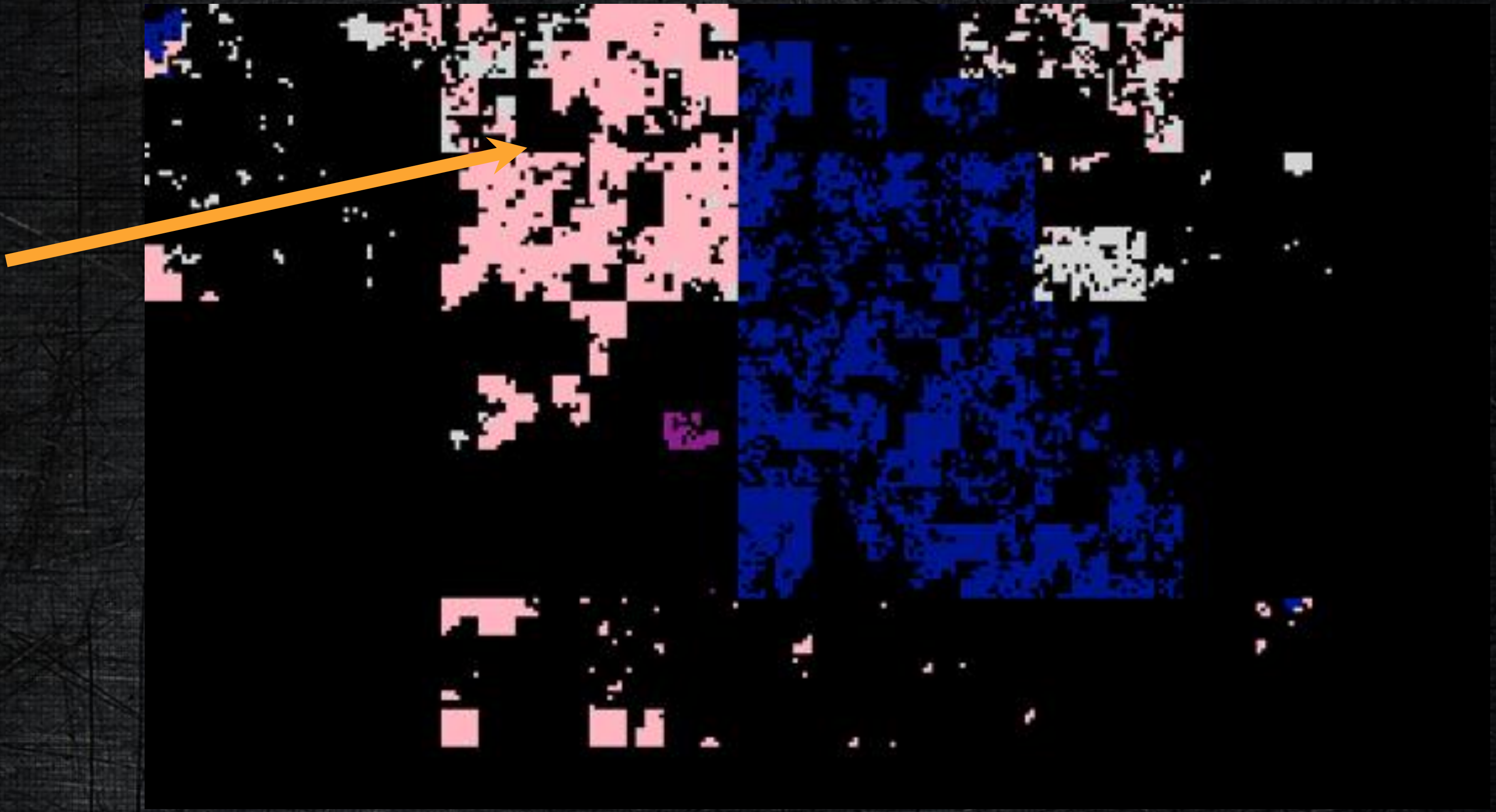
iOS: Example process (MobileSafari)

- Shared cache

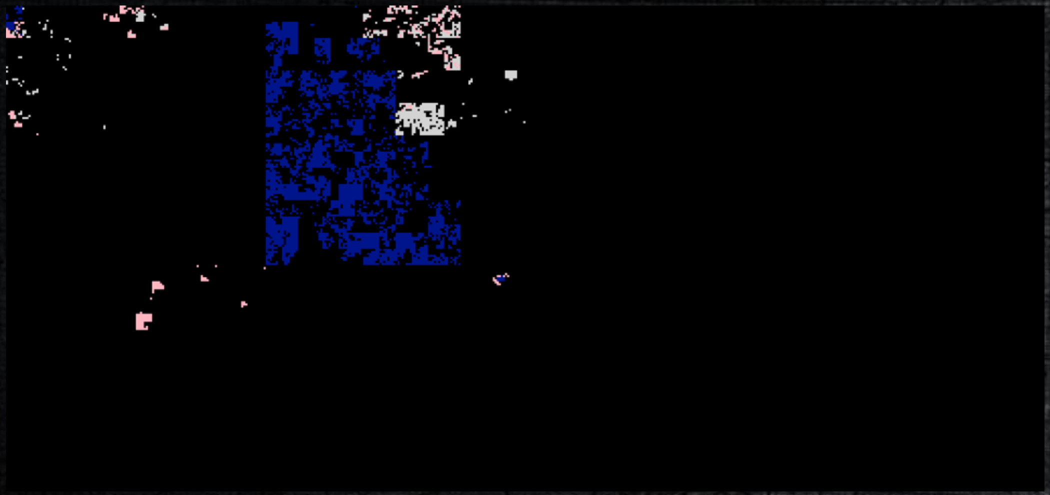


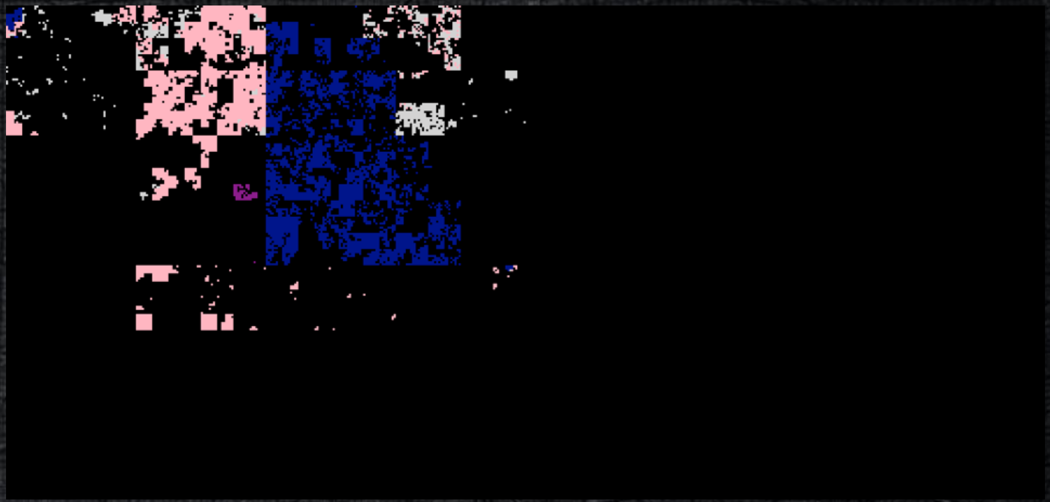
iOS: Example process (MobileSafari)

- Heap



MobileMail vs MobileSafari





iOS 6 Observations

- Evasi0n jailbreak leaves kernel mappings as RWX
- Fixed physical memory mappings across boots
  - Theoretically aids invasive attacks
  - May be useful to exploit DMA flaws from device hardware

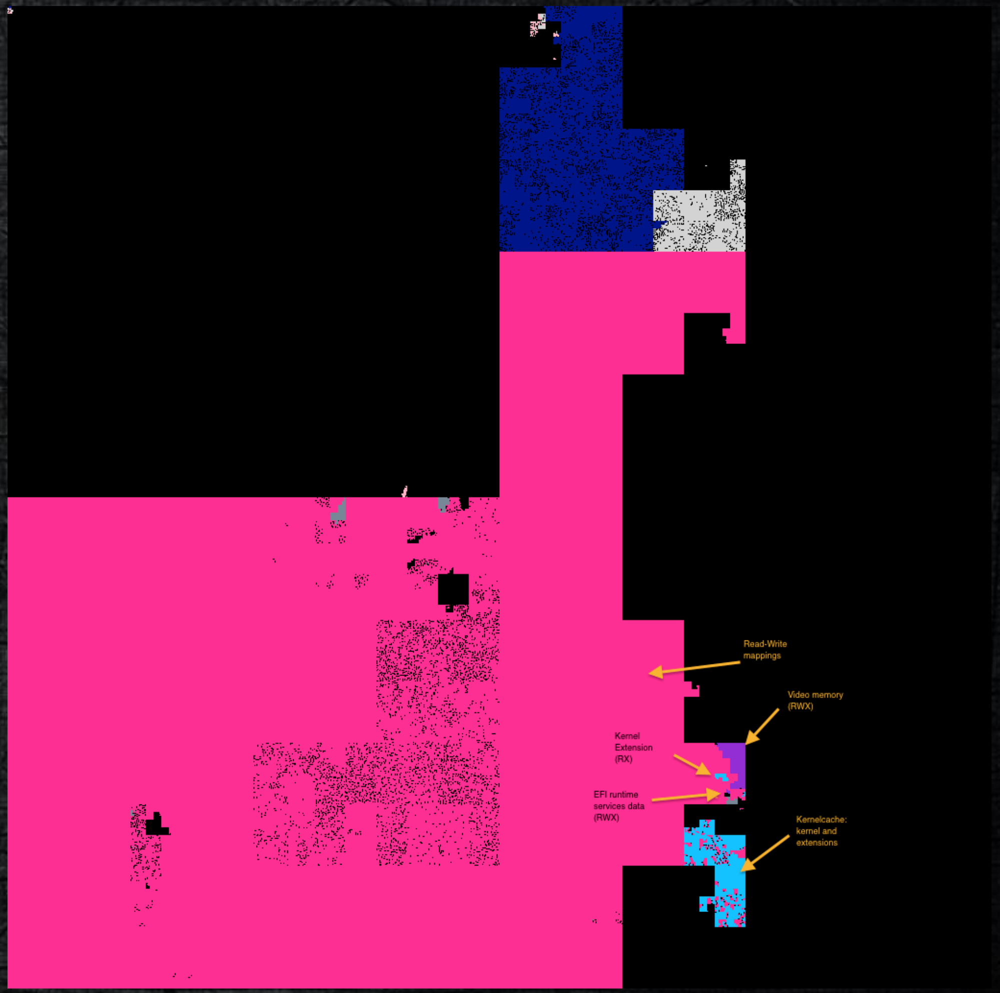
OS X
Mountain Lion

OS X Observations

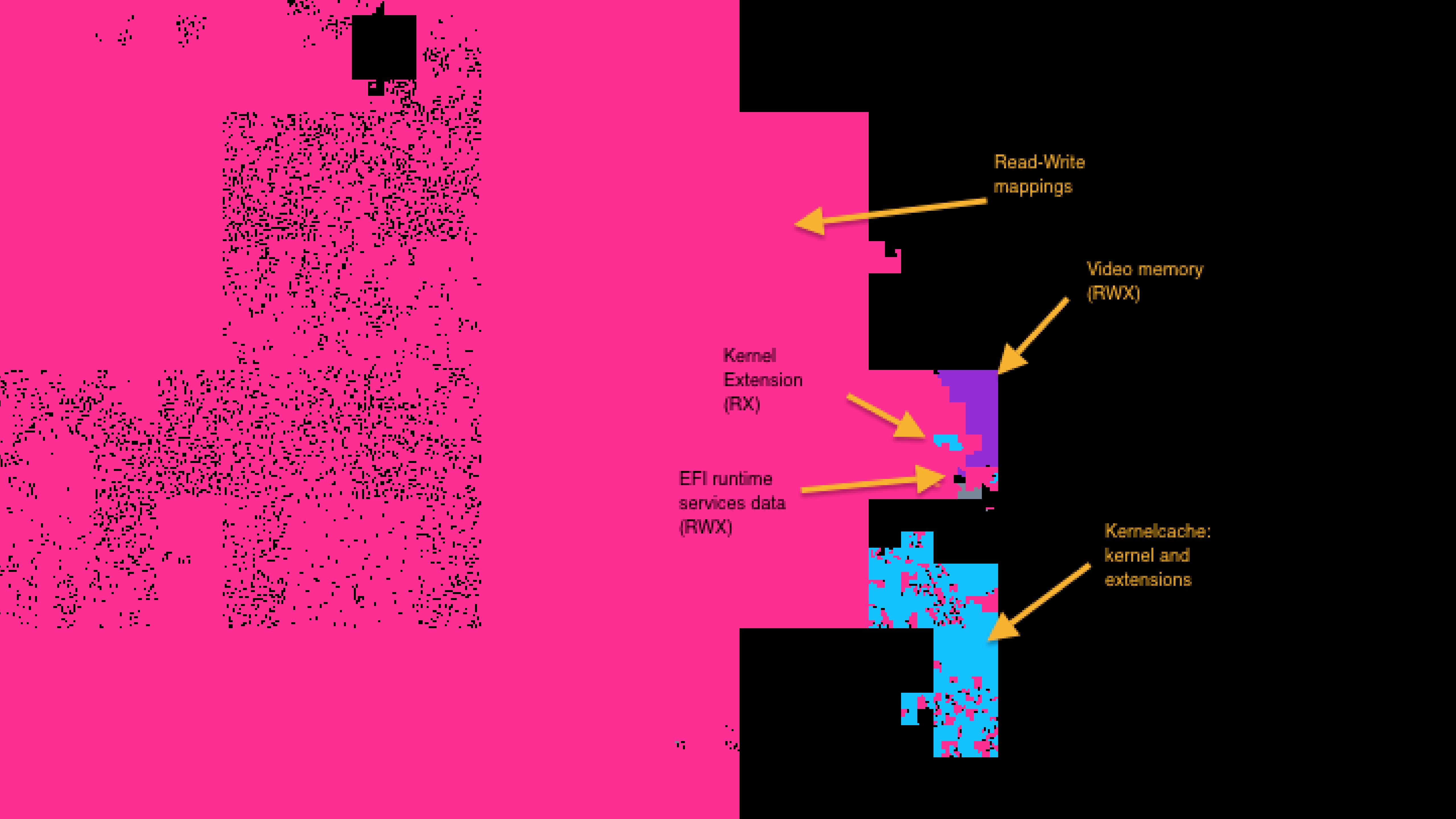
- Userland
  - Per-boot randomization (shared cache)
  - Per-execution randomization (dyld, pfz, commpage, stack, heap)
  - W^X not enforced

OS X Observations

- Kernel
  - KASLR
  - Incomplete W^X
    - Unpredictable RWX regions
  - Shared address space
    - SMEP available
  - Physical addresses also randomized



Read-Write mappings
Video memory (RWX)
Kernel Extension (RX)
EFI runtime services data (RWX)
Kernelcache: kernel and extensions



Read-Write mappings
Video memory (RWX)
Kernel Extension (RX)
EFI runtime services data (RWX)
Kernelcache: kernel and extensions

Special Thanks

- Shawn Denbow for Surface RT dumps
- Evad3rs & iPhone Dev Team for iOS jailbreaks
  - http://evasi0n.com/
  - http://blog.iphone-dev.org
- Aldo Cortesi for binvis
  - http://corte.si/posts/visualisation/binvis/

CROWDSTRIKE