# The Role of the Operating System in the Era of Cyberwar

**Vasileios (Vasilis) Kemerlis**

Department of Computer Science
Brown University

CSCI 1800: Cybersecurity and International Relations, 04/06/2016

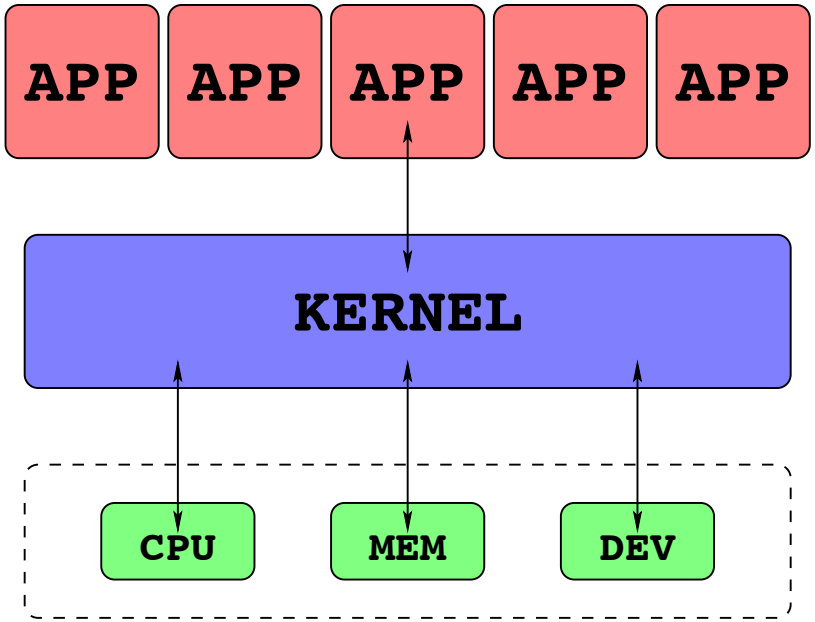
BROWN

# The Role of the Kernel

## What is a kernel anyway?

### "Heart" of the Operating System

- Abstracts the hardware
  - `CPU` ⇝ Execution thread
  - `MEM` ⇝ Virtual address space
  - `DEV` ⇝ I/O object
- Manages resources
  - ✓ Performance
  - ✓ Protection



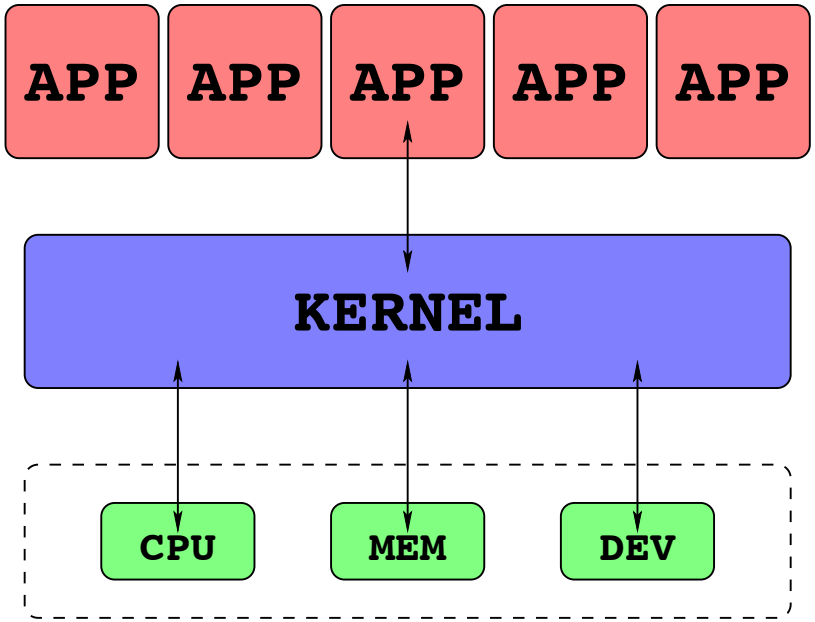

BROWN

# The Role of the Kernel

## What is a kernel anyway?

### "Heart" of the Operating System

- Abstracts the hardware
  - `CPU` ⇝ Execution thread
  - `MEM` ⇝ Virtual address space
  - `DEV` ⇝ I/O object
- Manages resources
  - ✓ Performance
  - ✓ Protection
    - Isolation ⇝ Virtual memory
    - Confinement ⇝ Namespaces, Capabilities
    - Access control ⇝ DAC/MAC, ACLs



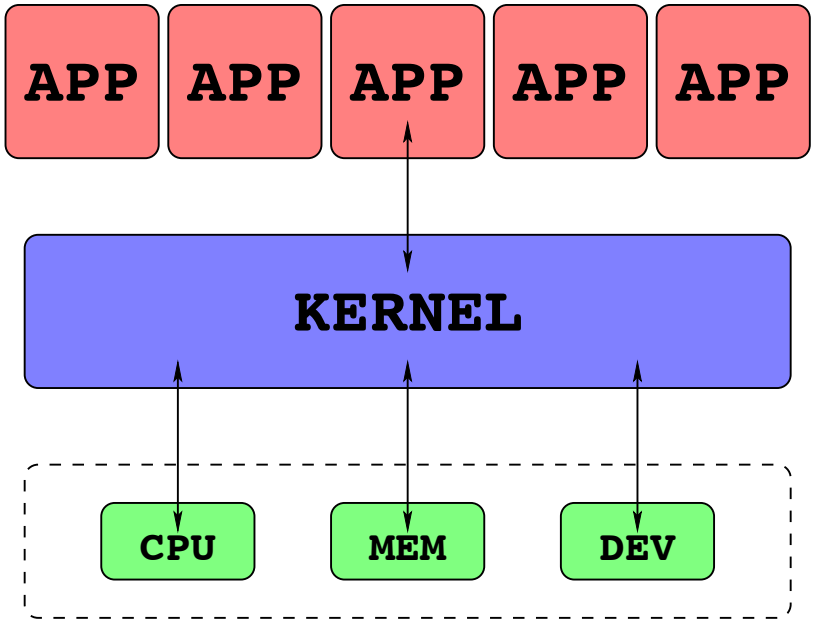

BROWN

# The Role of the Kernel

## What is a kernel anyway?

### "Heart" of the Operating System

- Abstracts the hardware
  - `CPU` ⇝ Execution thread
  - `MEM` ⇝ Virtual address space
  - `DEV` ⇝ I/O object
- Manages resources
  - ✓ Performance
  - ✓ Protection
    - Isolation ⇝ Virtual memory
    - Confinement ⇝ Namespaces, Capabilities
    - Access control ⇝ DAC/MAC, ACLs




Kernel → **Privilege**

*The security of a computer system can only be as good as that of the underlying OS kernel!*

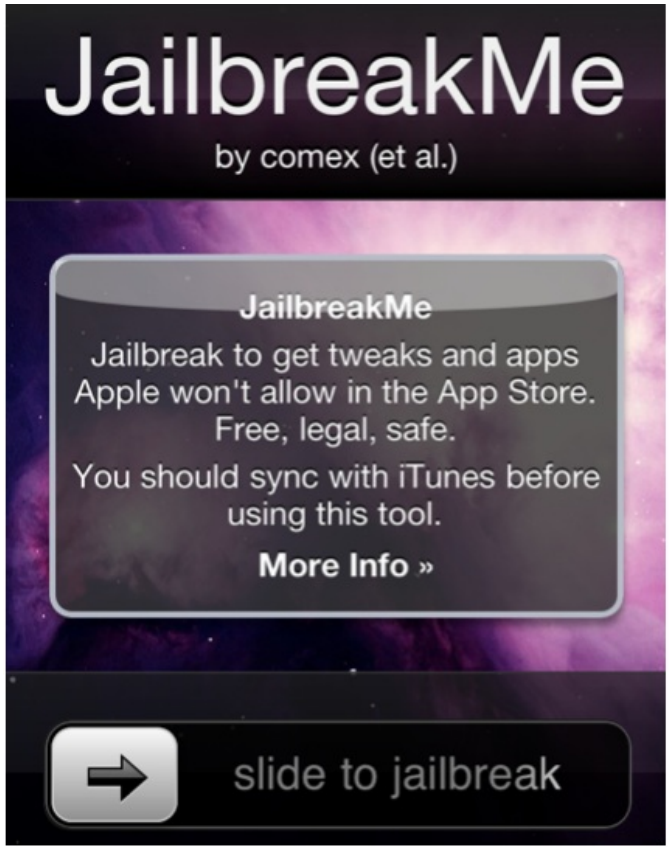
BROWN

# The Kernel as a Target

## Why care?

### Increased focus on kernel exploitation

- iOS kernel exploits used for jailbreaking (v3.x – v9.x)
  - ✗ `IOHIDFamily` (CVE-2015-6974)
  - ✗ `IOSharedDataQueue` (CVE-2014-4461)
  - ✗ `TempSensor` (CVE-2014-4388)
  - ✗ `ptmx_get_ioctl` (CVE-2014-1278)
  - ✗ `IOUSBDeviceFamily` (CVE-2013-0981)
  - ✗ Debug syscall (CVE-2012-0643)
  - ✗ Packet Filter (CVE-2012-3728)
  - ✗ HFS Heap (CVE-2012-0642)
  - ✗ `ndrv_setspec`
  - ✗ HFS Legacy Volume Name
  - ✗ Packet Filter (CVE-2010-3830)
  - ✗ `IOSurface` (CVE-2010-2973)
  - ✗ `BPF_STX`



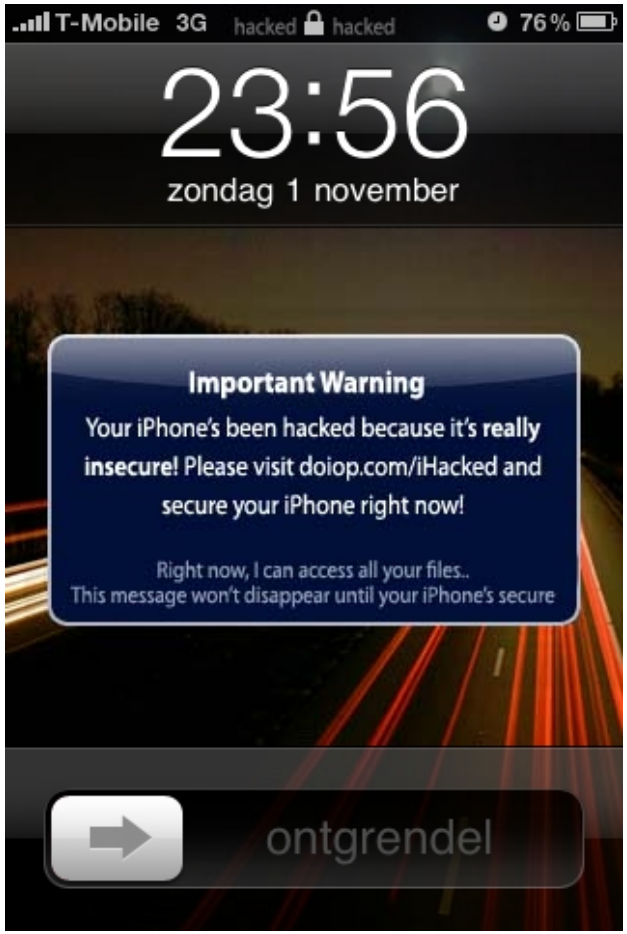

BROWN

# The Kernel as a Target

## Why care?

### Increased focus on kernel exploitation

- iOS kernel exploits used for jailbreaking (v3.x – v9.x)
  - ✗ `IOHIDFamily` (CVE-2015-6974)
  - ✗ `IOSharedDataQueue` (CVE-2014-4461)
  - ✗ `TempSensor` (CVE-2014-4388)
  - ✗ `ptmx_get_ioctl` (CVE-2014-1278)
  - ✗ `IOUSBDeviceFamily` (CVE-2013-0981)
  - ✗ Debug syscall (CVE-2012-0643)
  - ✗ Packet Filter (CVE-2012-3728)
  - ✗ HFS Heap (CVE-2012-0642)
  - ✗ `ndrv_setspec`
  - ✗ HFS Legacy Volume Name
  - ✗ Packet Filter (CVE-2010-3830)
  - ✗ `IOSurface` (CVE-2010-2973)
  - ✗ `BPF_STX`

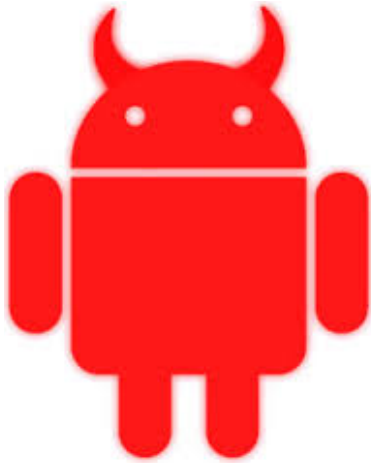


# The Kernel as a Target

## Why care?

### Increased focus on kernel exploitation

- Linux kernel exploits used by Android malware
  - ✗ TowelRoot (CVE-2014-3153)
  - ✗ `perf_swevent_init` (CVE-2013-2094)
  - ✗ Levitator (CVE-2011-1350)
  - ✗ Wunderbar (CVE-2009-2692)
  - ✗ ...



BROWN

# The Kernel as a Target

## Why care?

### Increased focus on kernel exploitation

- Windows kernel exploits used in cyber attacks
  - ✗ Duqu2 (CVE-2015-2360)
  - ✗ Pawn Storm (CVE-2015-1701)
  - ✗ Sandworm (CVE-2014-4114)
  - ✗ TrueType Font Parsing (CVE-2011-3042)
  - ✗ ...

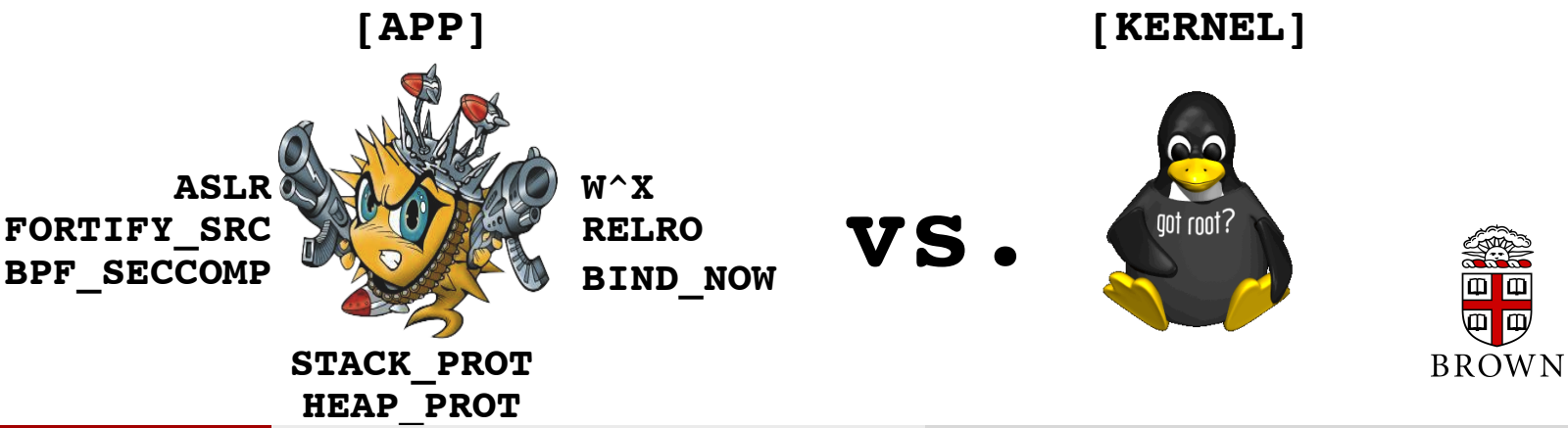
# The Kernel as a Target

## Why care?

### Increased focus on kernel exploitation

1. High-value asset → **Privileged** piece of code
   - Responsible for the integrity of OS security mechanisms
2. Large attack surface → syscalls, device drivers, pseudo fs, ...
   - New features & optimizations → **New attack opportunities**
3. Exploiting privileged userland processes has become harder → Canaries+ASLR+W^X+Fortify+RELRO+BIND_NOW+BPF_SECCOMP+...
   - Sergey Glazunov (Pwn2Own '12) ⇝ 14 bugs to takedown Chrome

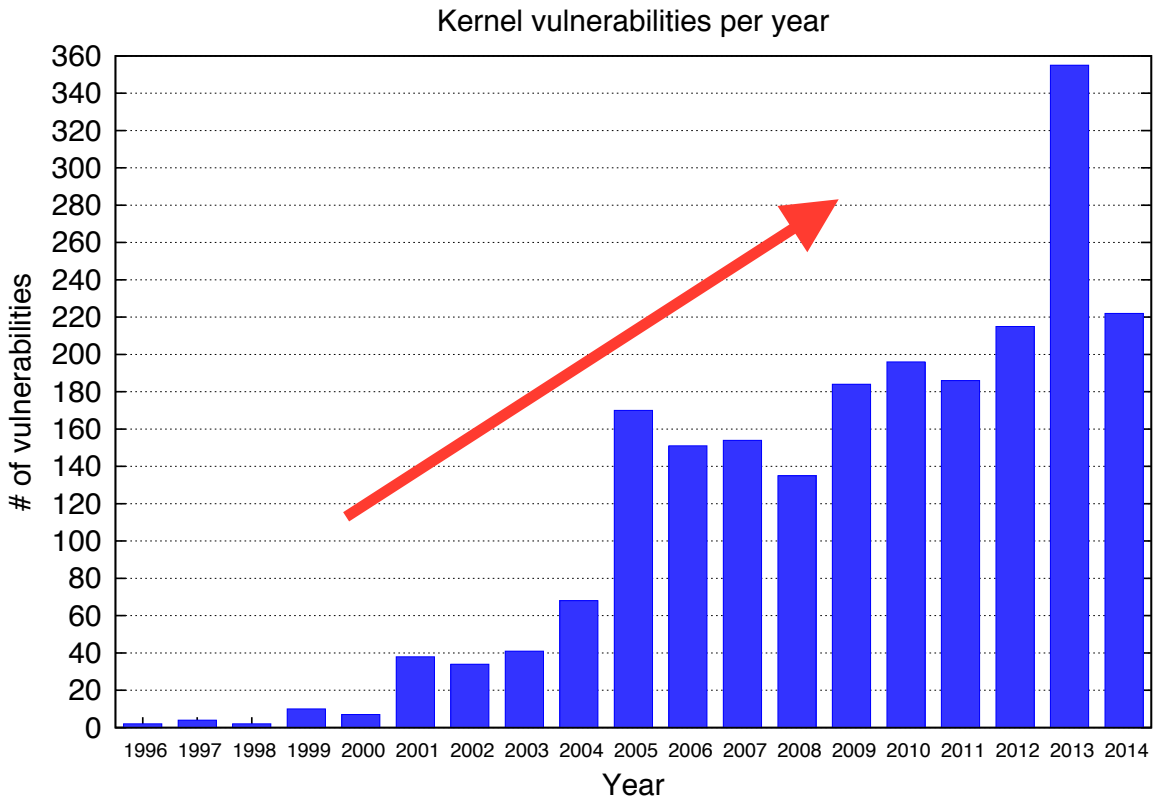
"A Tale of Two Pwnies" (`http://blog.chromium.org`)



# Kernel Vulnerabilities

## Current state of affairs (all vendors)

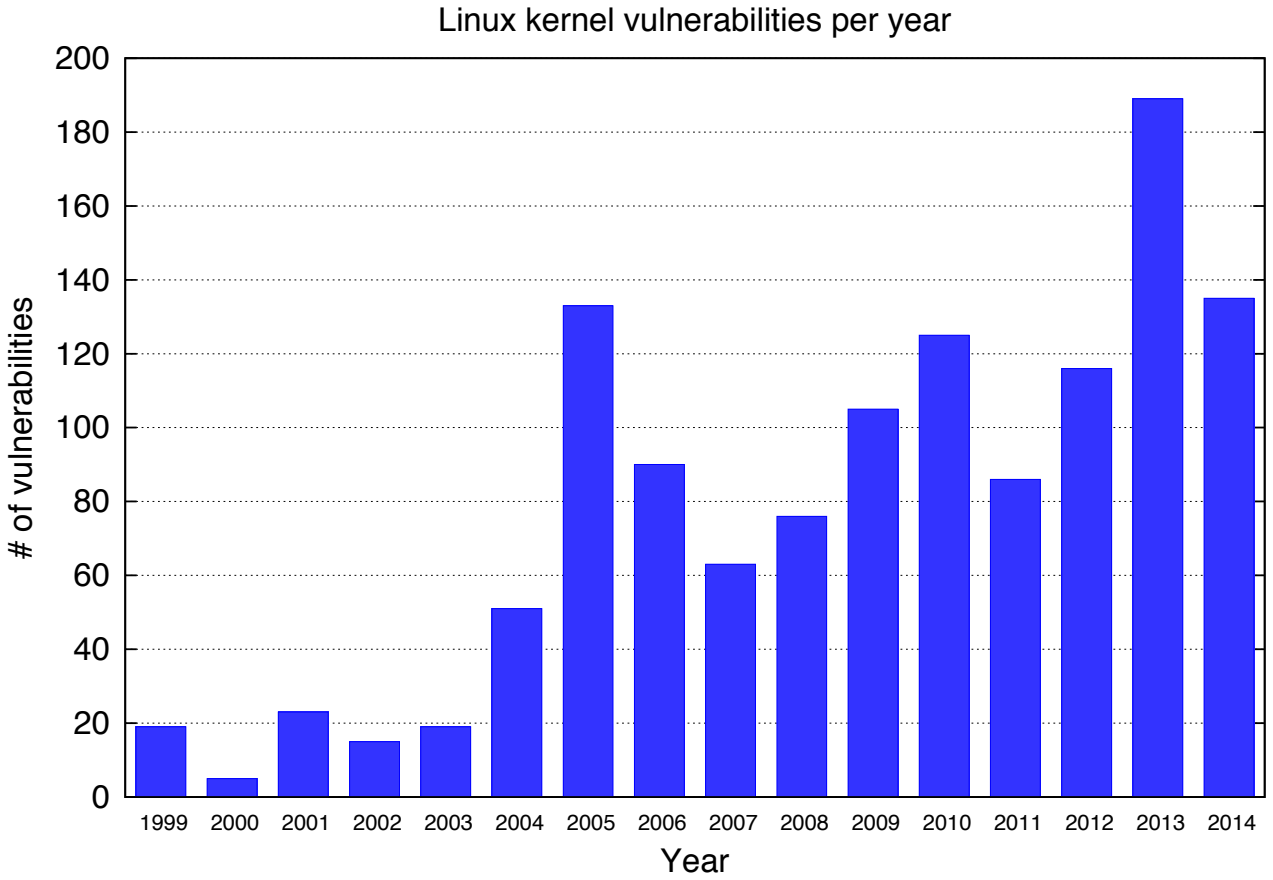
- ✗ Kernel stack smashing
- ✗ Kernel heap overflows
- ✗ Wild writes, off-by-*n*
- ✗ Poor arg. sanitization
- ✗ User-after-free, double free, dangling pointers
- ✗ Signedness errors, integer overflows
- ✗ Race conditions, memory leaks
- ✗ Missing authorization checks




Source: National Vulnerability Database (`http://nvd.nist.gov`)

# Kernel Vulnerabilities (cont'd)

## Current state of affairs (Linux only)

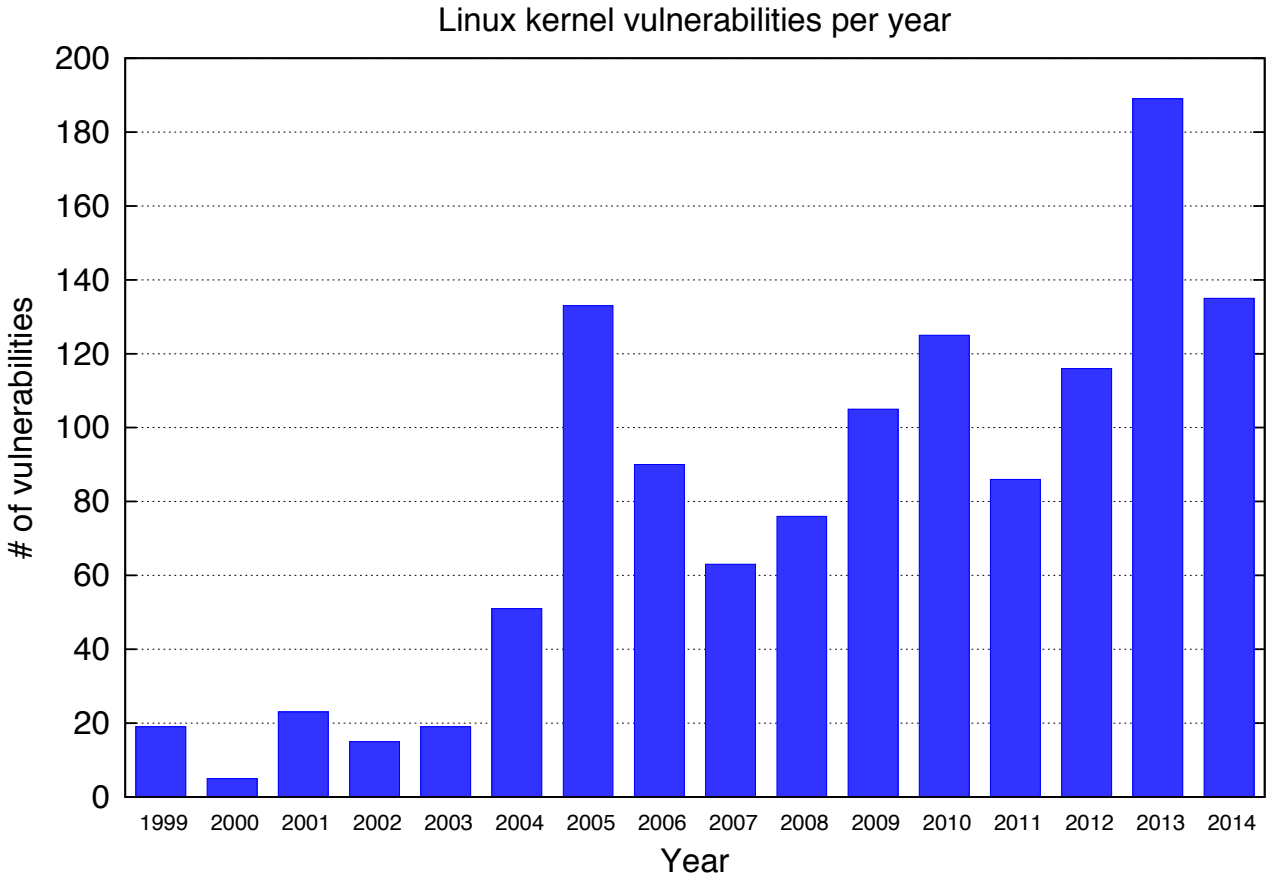



| Kernel ver. | Size | Dev. days | Patches | Changes/hr | Fixes |
|---|---|---|---|---|---|
| 2.6.11 (03/02/05) | 6.6 MLOC | 69 | 3.6K | 2.18 | 79 |
| 3.10 (30/06/13) | 16.9 MLOC | 63 | 13.3K | 9.02 | 670 |

Source: CVE Details (`http://www.cvedetails.com`), The Linux Foundation

# Kernel Vulnerabilities (cont'd)

## Current state of affairs (Linux only)

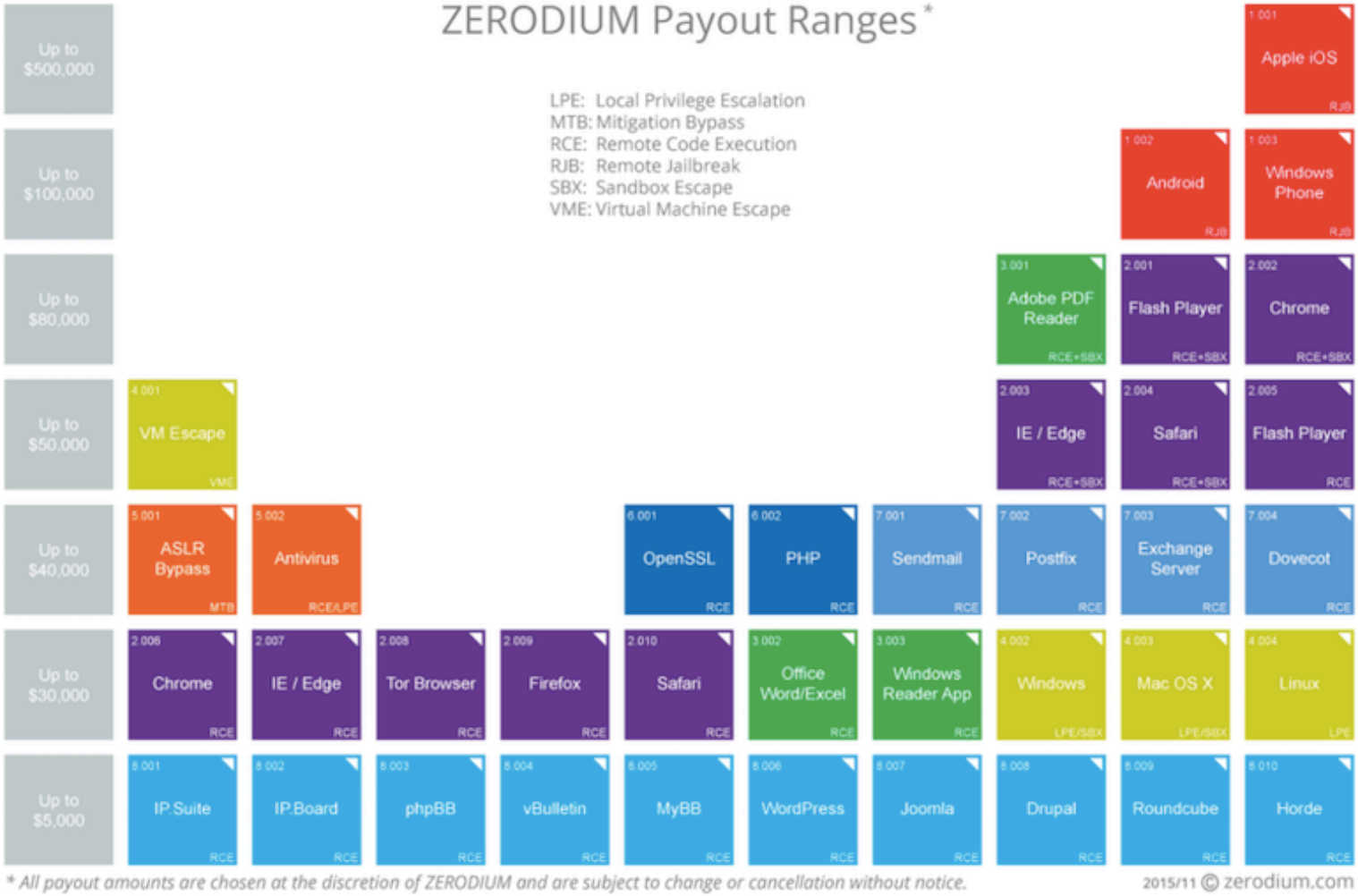



| Kernel ver. | Size | Dev. days | Patches | Changes/hr | Fixes |
|---|---|---|---|---|---|
| 2.6.11 (03/02/05) | 6.6 MLOC | 69 | 3.6K | 2.18 | 79 |
| 3.10 (30/06/13) | 16.9 MLOC | 63 | 13.3K | 9.02 | 670 |

Source: CVE Details (`http://www.cvedetails.com`), The Linux Foundation

# Kernel Vulnerabilities (cont'd)

## Pricelist




Source: Zerodium

# Attacking the "Core"

## Threats classification

1. Privilege escalation 

   - Arbitrary code execution $\rightsquigarrow$ return-to-user (`ret2usr`)

   ✗ Kernel stack smashing
   ✗ Kernel heap overflows
   ✗ Wild writes, off-by-$n$
   ✗ Poor arg. sanitization
   ✗ User-after-free, double free, dangling pointers
   ✗ Signedness errors, integer overflows
   ✗ Race conditions, memory leaks
   ✗ Missing authorization checks

2. Persistent foothold
   - Kernel object hooking (KOH) $\rightsquigarrow$ control-flow hijacking
     ✗ Kernel control data (function ptr., dispatch tbl., return addr.)
     ✗ Kernel code (`.text`)
   - Direct kernel object manipulation (DKOM) $\rightsquigarrow$ cloaking
     ✗ Kernel non-control data

BROWN

# Attacking the "Core"

## Threats classification

1. Privilege escalation
   - Arbitrary code execution $\rightsquigarrow$ return-to-user (`ret2usr`)

   ✗ Kernel stack smashing
   ✗ Kernel heap overflows
   ✗ Wild writes, off-by-*n*
   ✗ Poor arg. sanitization
   ✗ User-after-free, double free, dangling pointers
   ✗ Signedness errors, integer overflows
   ✗ Race conditions, memory leaks
   ✗ Missing authorization checks

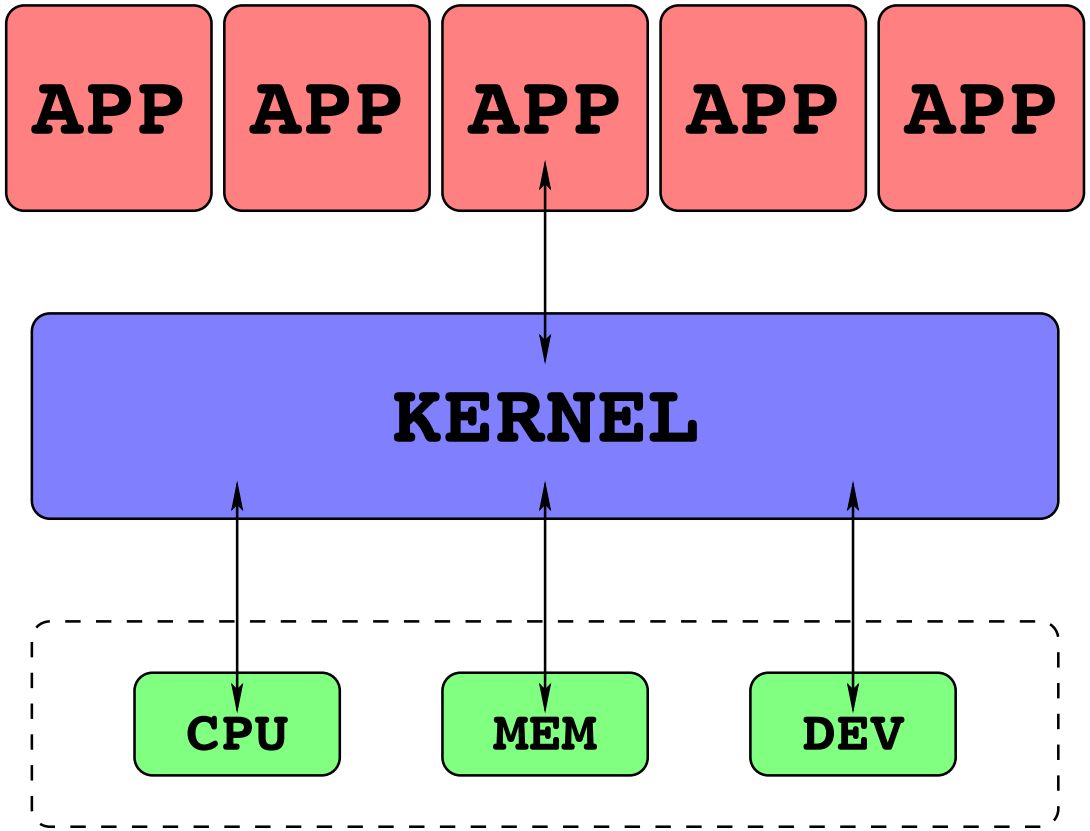
2. Persistent foothold
   - Kernel object hooking (KOH) $\rightsquigarrow$ control-flow hijacking
     ✗ Kernel control data (function ptr., dispatch tbl., return addr.)
     ✗ Kernel code (`.text`)
   - Direct kernel object manipulation (DKOM) $\rightsquigarrow$ cloaking
     ✗ Kernel non-control data

BROWN

# Attacking the "Core" (cont'd)

## From `RING 0` to `UID 0`

- **Privilege escalation** 

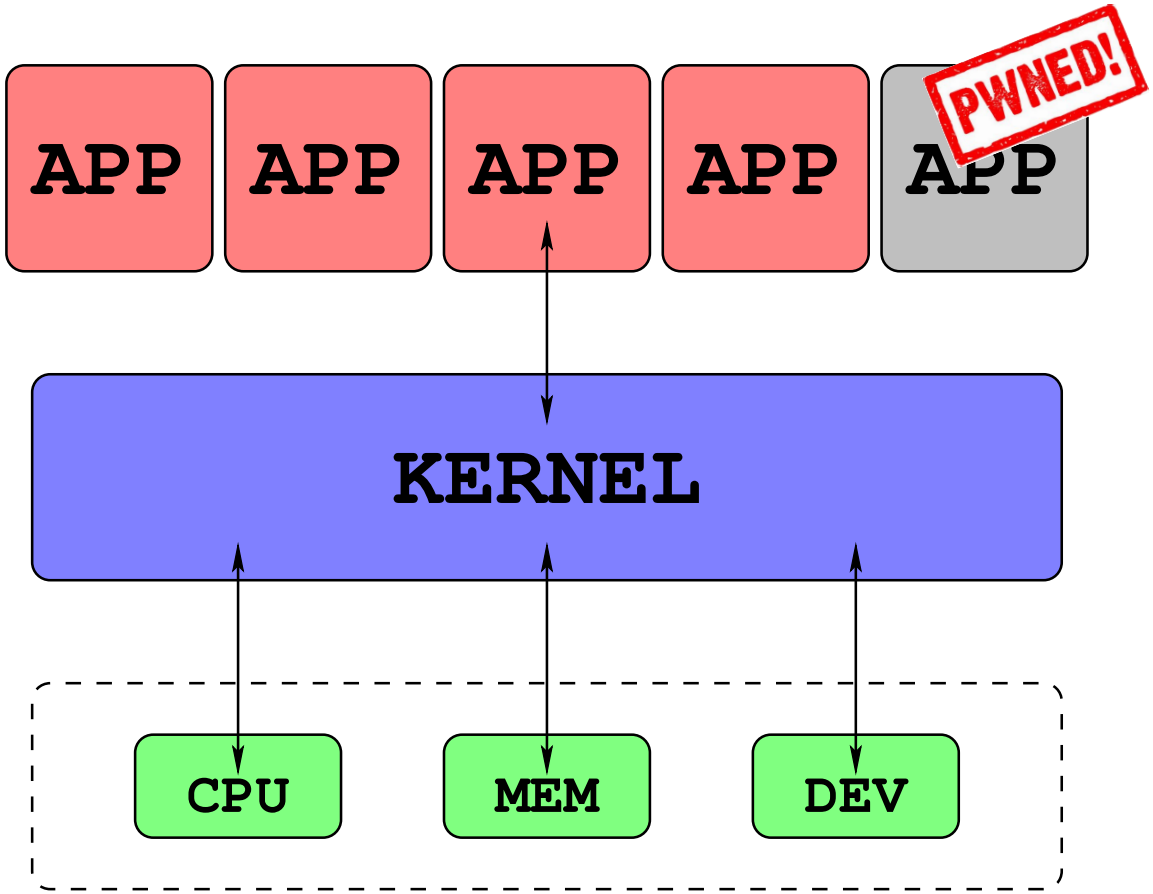




BROWN

# Attacking the "Core" (cont'd)

## From `RING 0` to `UID 0`

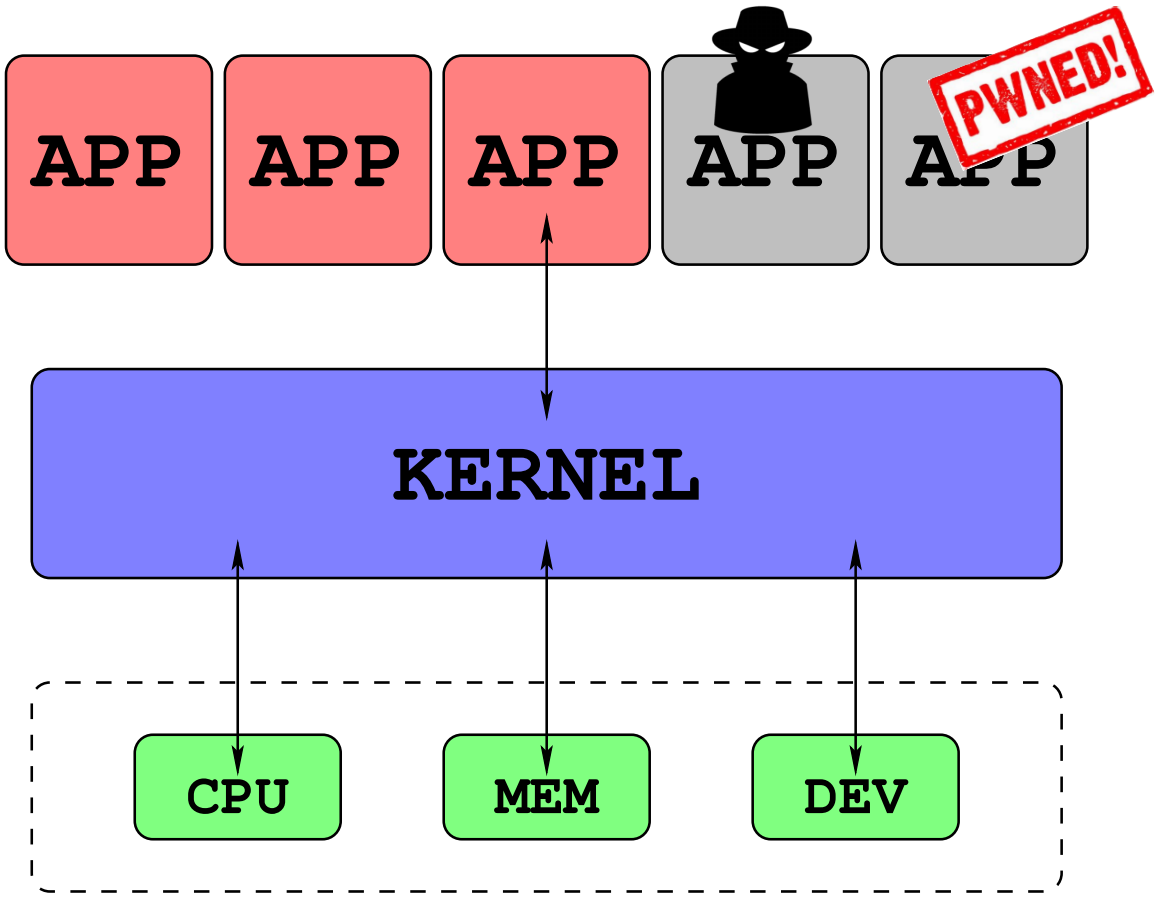
- **Privilege escalation** 

# Attacking the "Core" (cont'd)

## From `RING 0` to `UID 0`

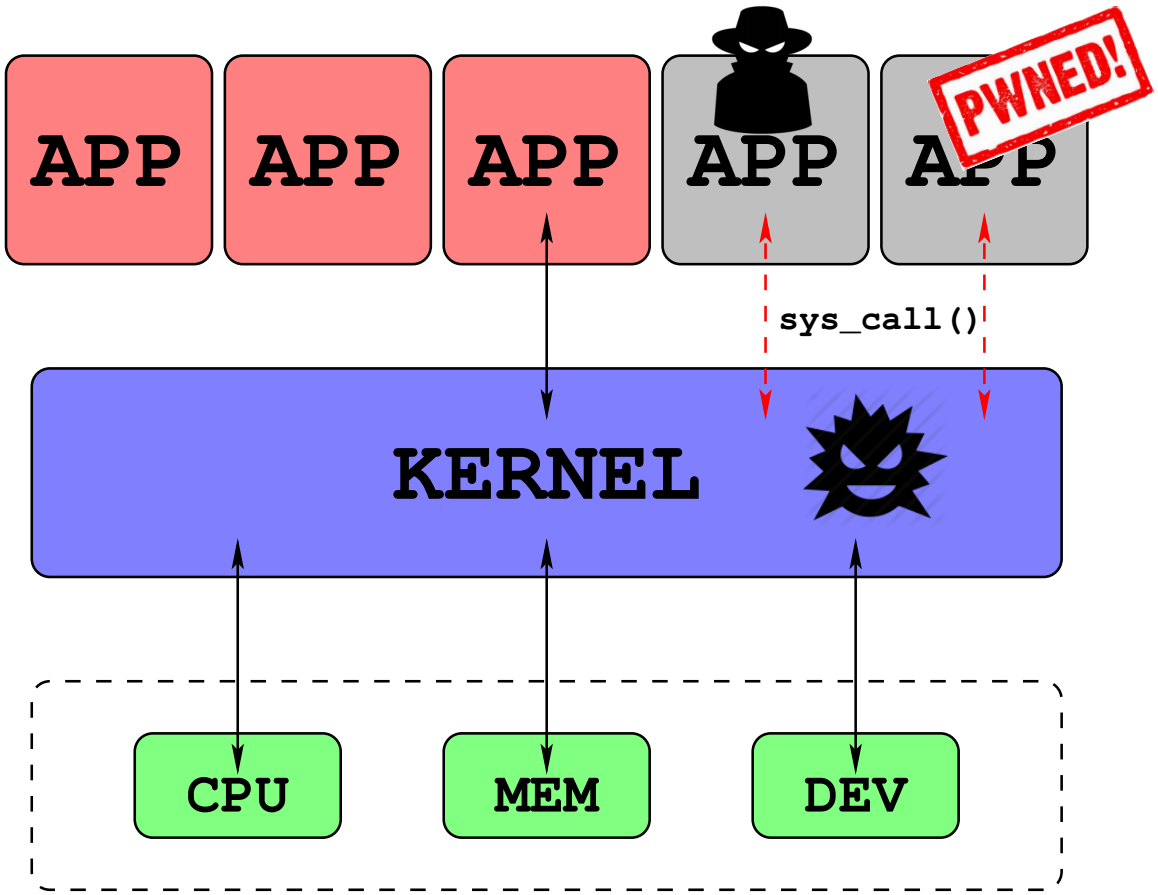
- **Privilege escalation** 

# Attacking the "Core" (cont'd)

## From `RING 0` to `UID 0`

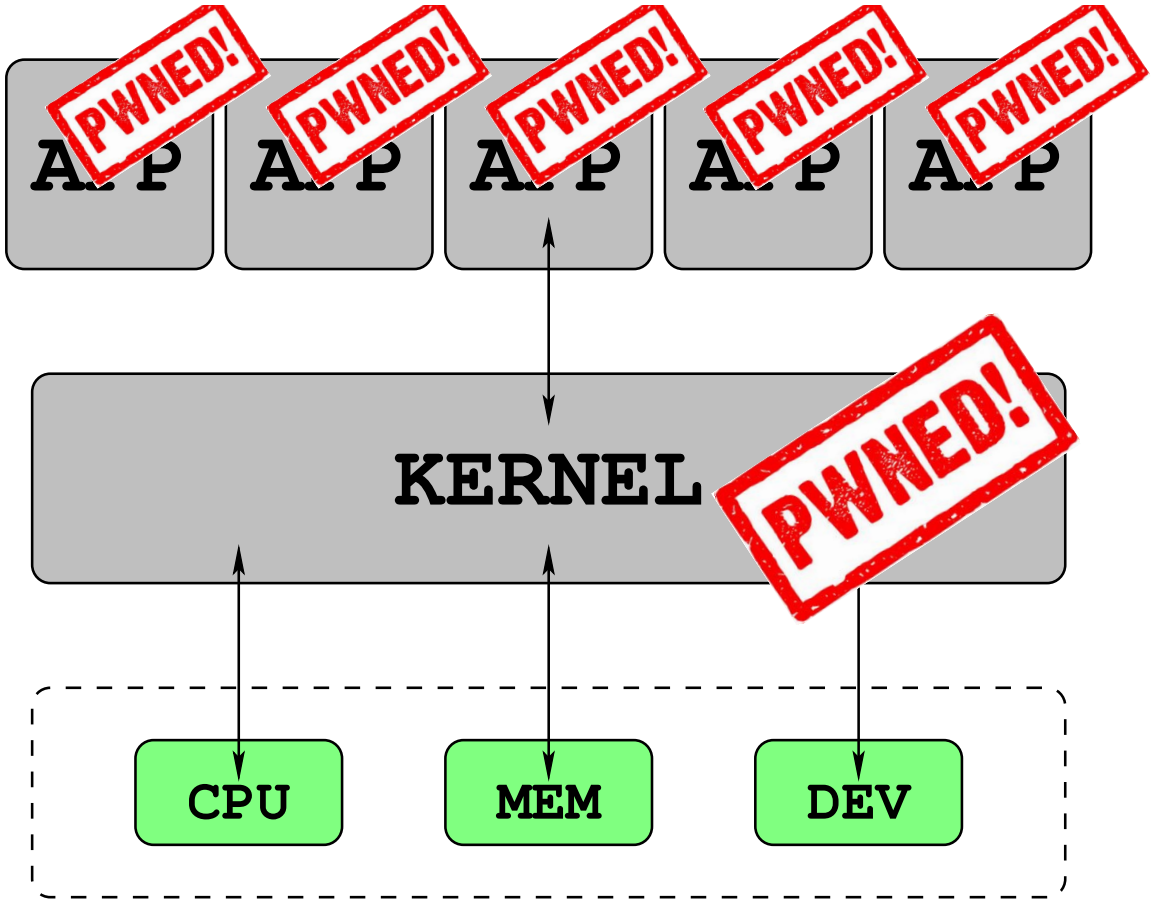
- **Privilege escalation** 

# Attacking the "Core" (cont'd)

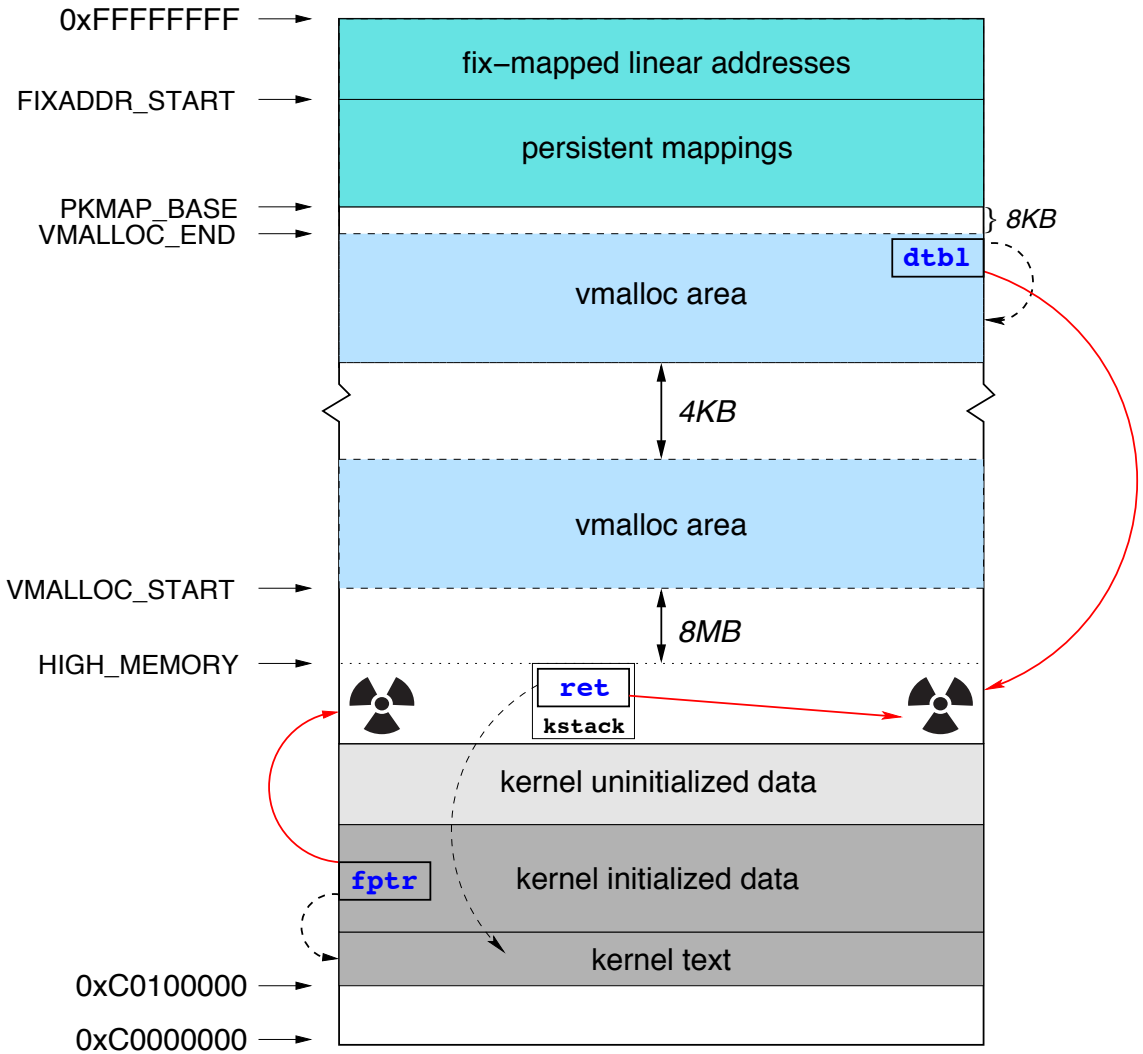
## From `RING 0` to `UID 0`

- **Privilege escalation** 

# Code-{injection, reuse} Attacks

## Linux example (x86)

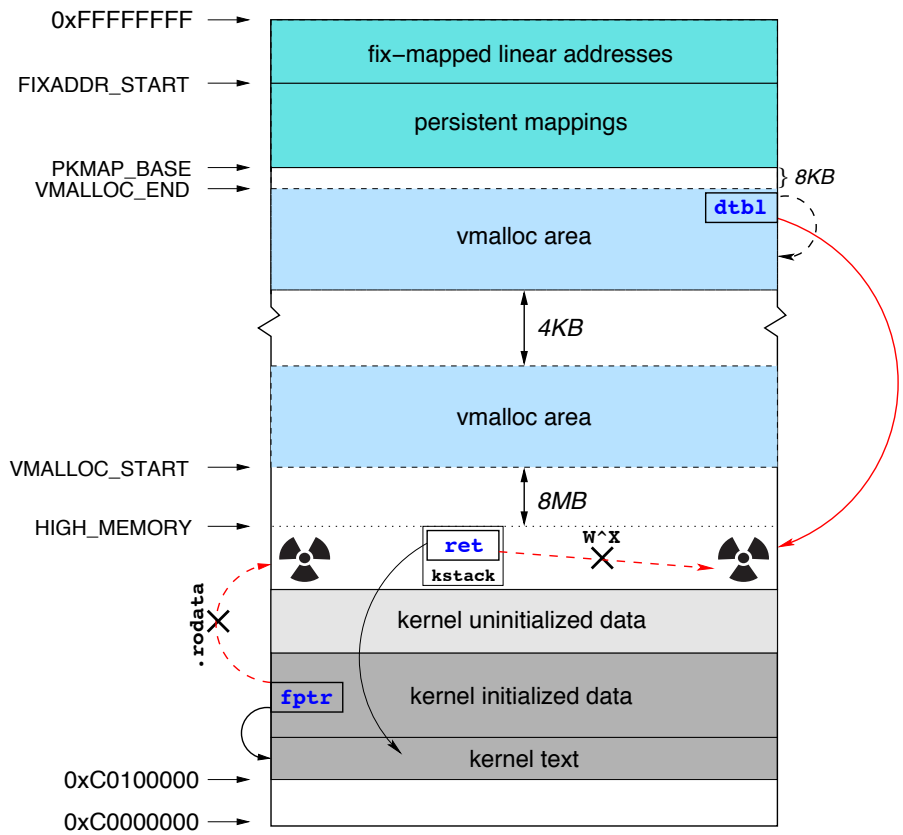


# Code-{injection, reuse} Attacks (cont'd)

## Classic defenses

Similar to userland exploitation
→ Many protection schemes

- ✓ stack canaries (SSP), SLAB red zones, KASLR, W^X
- ✓ `const` dispatch tables (IDT, GDT, syscall)
- ✓ `.rodata` sections
- ✓ ...



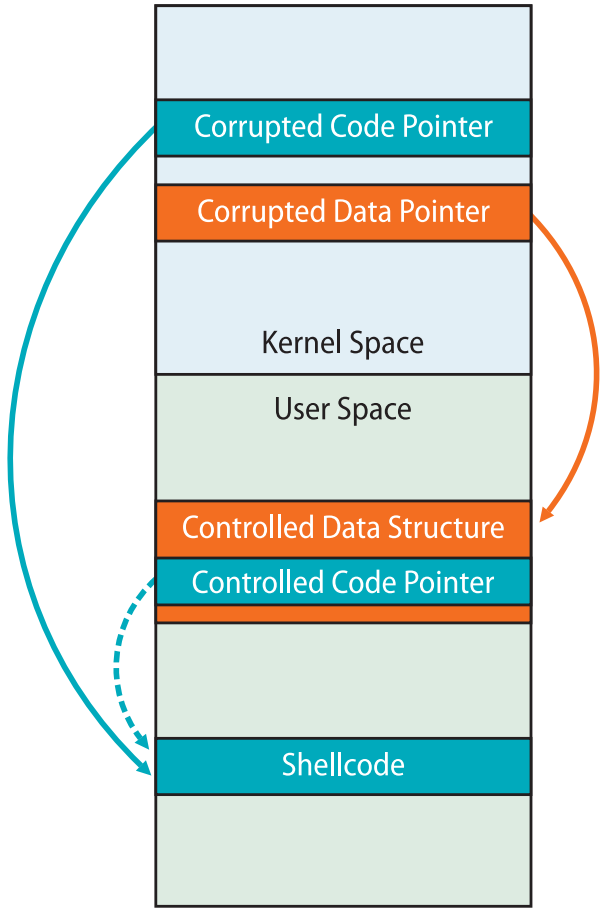

BROWN

# Return-to-user (`ret2usr`) Attacks

## What are they?

### Attacks against OS kernels with shared kernel/user address space

- Overwrite kernel code (or data) pointers with **user space** addresses
  - ✗ return addr., dispatch tbl., function ptr.,
  - ✗ data ptr.
- Payload $\rightsquigarrow$ Shellcode, ROP payload, tampered-with data structure(s)
  - Placed in user space
  - ✗ Executed (referenced) in kernel context

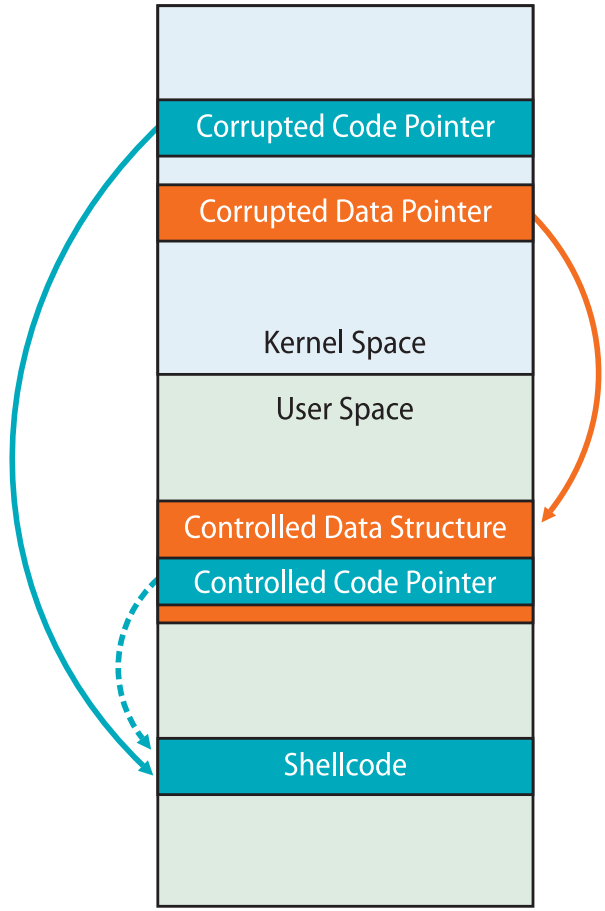


# Return-to-user (`ret2usr`) Attacks

## What are they?

### Attacks against OS kernels with shared kernel/user address space

- Overwrite kernel code (or data) pointers with **user space** addresses
  - ✗ return addr., dispatch tbl., function ptr.,
  - ✗ data ptr.
- Payload $\rightsquigarrow$ Shellcode, ROP payload, tampered-with data structure(s)
  - Placed in user space
  - ✗ Executed (referenced) in kernel context
- De facto kernel exploitation technique
  - ✗ `http://www.exploit-db.com/exploits/34134/` (21/07/14)
  - ✗ `http://www.exploit-db.com/exploits/131/` (05/12/03)



# `ret2usr` Attacks (cont'd)

## Retro pwn!

- G. J. Popek and D. A. Farber. "A Model for Verification of Data Security in Operating Systems." *Commun. ACM*, 21(9):737–749, Sept. 1978.

# ret2usr Attacks (cont'd)

## Retro pwn!

- G. J. Popek and D. A. Farber. "A Model for Verification of Data Security in Operating Systems." *Commun. ACM*, 21(9):737–749, Sept. 1978.

### PDP-10 address wraparound fault

**"Control was therefore returned to user code at his virtual location zero–in privileged mode!"**

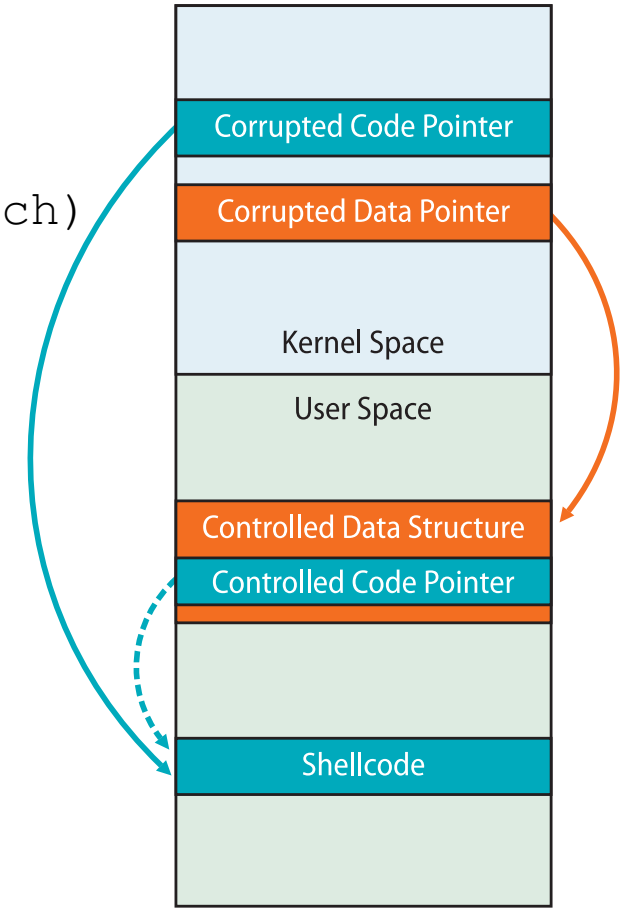
Source: Wikipedia (https://en.wikipedia.org/wiki/PDP-10)

# `ret2usr` Attacks (cont'd)

## Why do they work?

### Weak address space (kernel/user) separation

- Shared kernel/process model → Performance
  - ✓ `cost(mode_switch)` ≪ `cost(context_switch)`
- The kernel is protected from userland → Hardware-assisted isolation
  - ✗ The opposite is **not** true
  - ✗ Kernel ⇝ **ambient authority** (unrestricted access to all memory and system objects)

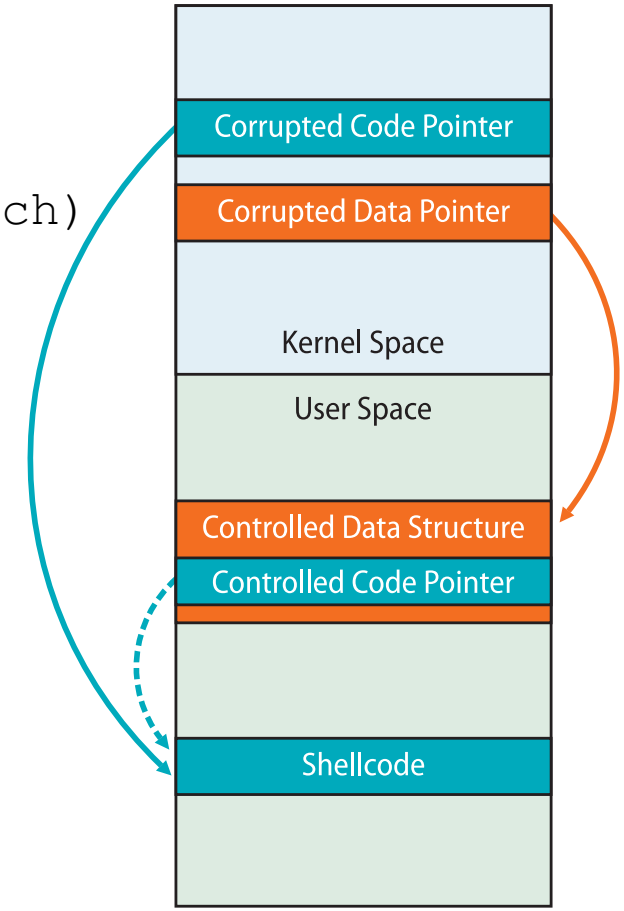


# `ret2usr` Attacks (cont'd)

## Why do they work?

### Weak address space (kernel/user) separation

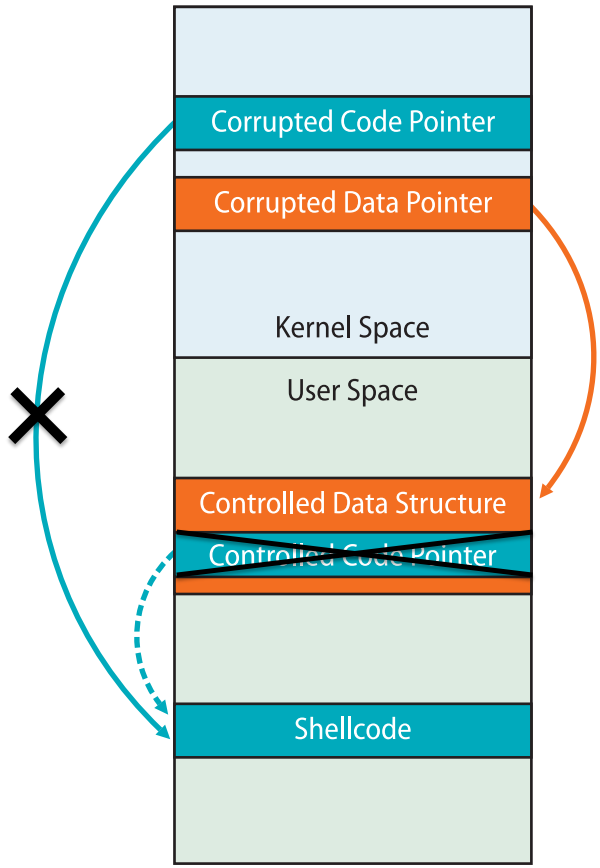
- Shared kernel/process model → Performance
  - ✓ `cost(mode_switch)` ≪ `cost(context_switch)`
- The kernel is protected from userland → Hardware-assisted isolation
  - ✗ The opposite is **not** true
  - ✗ Kernel ⇝ **ambient authority** (unrestricted access to all memory and system objects)
- The attacker completely controls user space memory
  - Contents & perms.



# kGuard [USENIX Sec '12, ;login: '12]

## Lightweight protection against ret2usr

- Cross-platform solution that enforces address space separation between user and kernel space
  - x86, x86-64, ARM, ...
  - Linux, Android, {Free, Open}BSD, ...
- Defensive mechanism that builds upon inline monitoring and code diversification
- Non-intrusive & low overhead



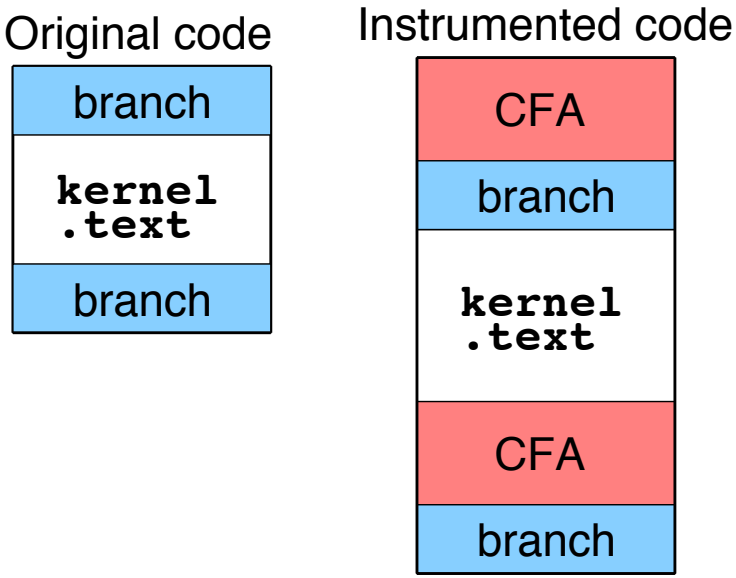

## Goal

✓ Cast a **realistic** threat ineffective

# kGuard Design

## Control-flow assertions (key technology #1; confinement)

- Compact, inline guards injected at compile time
  - Two flavors $\rightsquigarrow$ $\texttt{CFA}_{\texttt{R}}$ & $\texttt{CFA}_{\texttt{M}}$
- Placed before every exploitable control-flow transfer
  - `call`, `jmp`, `ret` in x86/x86-64
  - `ldm`, `blx`, ..., in ARM

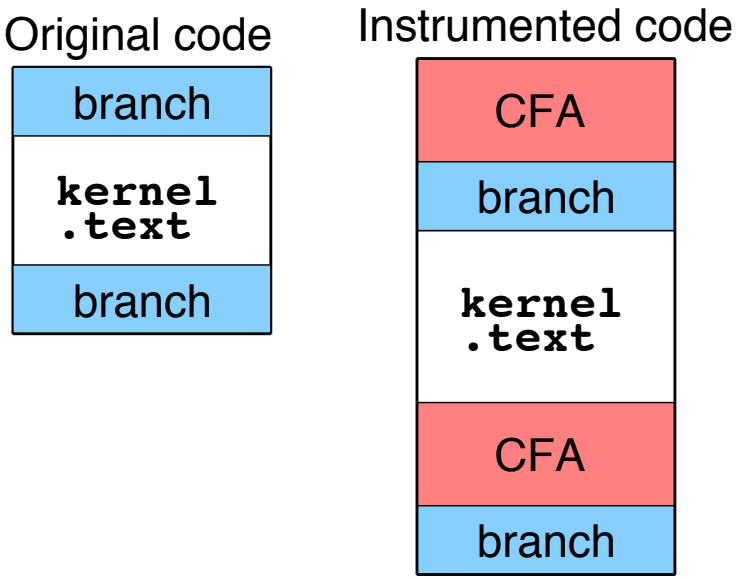



BROWN

# kGuard Design

## Control-flow assertions (key technology #1; confinement)

- Compact, inline guards injected at compile time
  - Two flavors $\rightsquigarrow$ $\texttt{CFA}_{\texttt{R}}$ & $\texttt{CFA}_{\texttt{M}}$
- Placed before every exploitable control-flow transfer
  - `call`, `jmp`, `ret` in x86/x86-64
  - `ldm`, `blx`, ..., in ARM

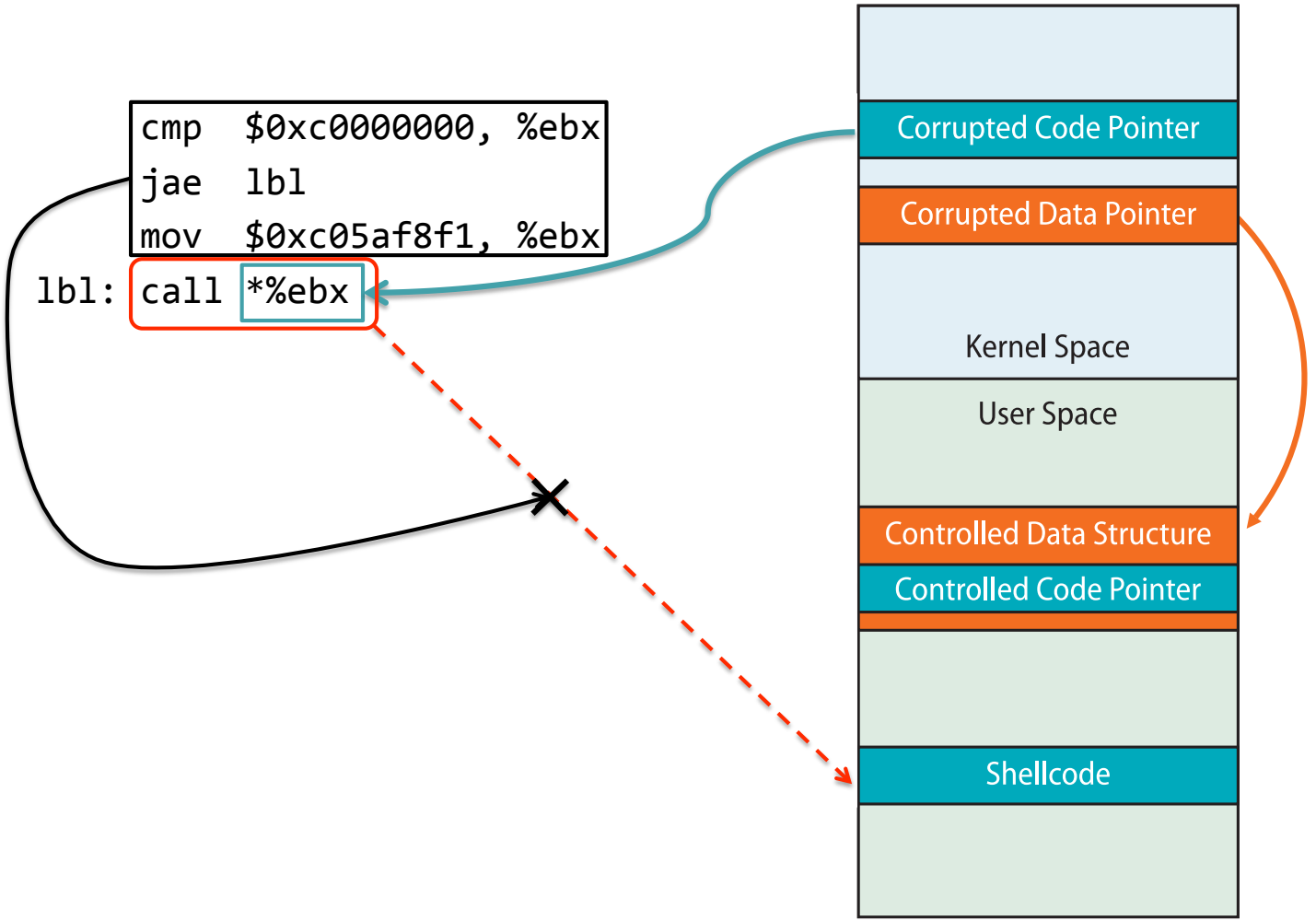



- Verify that the target address of an **indirect** branch is always inside kernel space
- If the assertion is true, execution continues normally; otherwise, control-flow is transfered to a runtime violation handler

BROWN

# kGuard Design (cont'd)

## $\mathrm{CFA}_R$ example

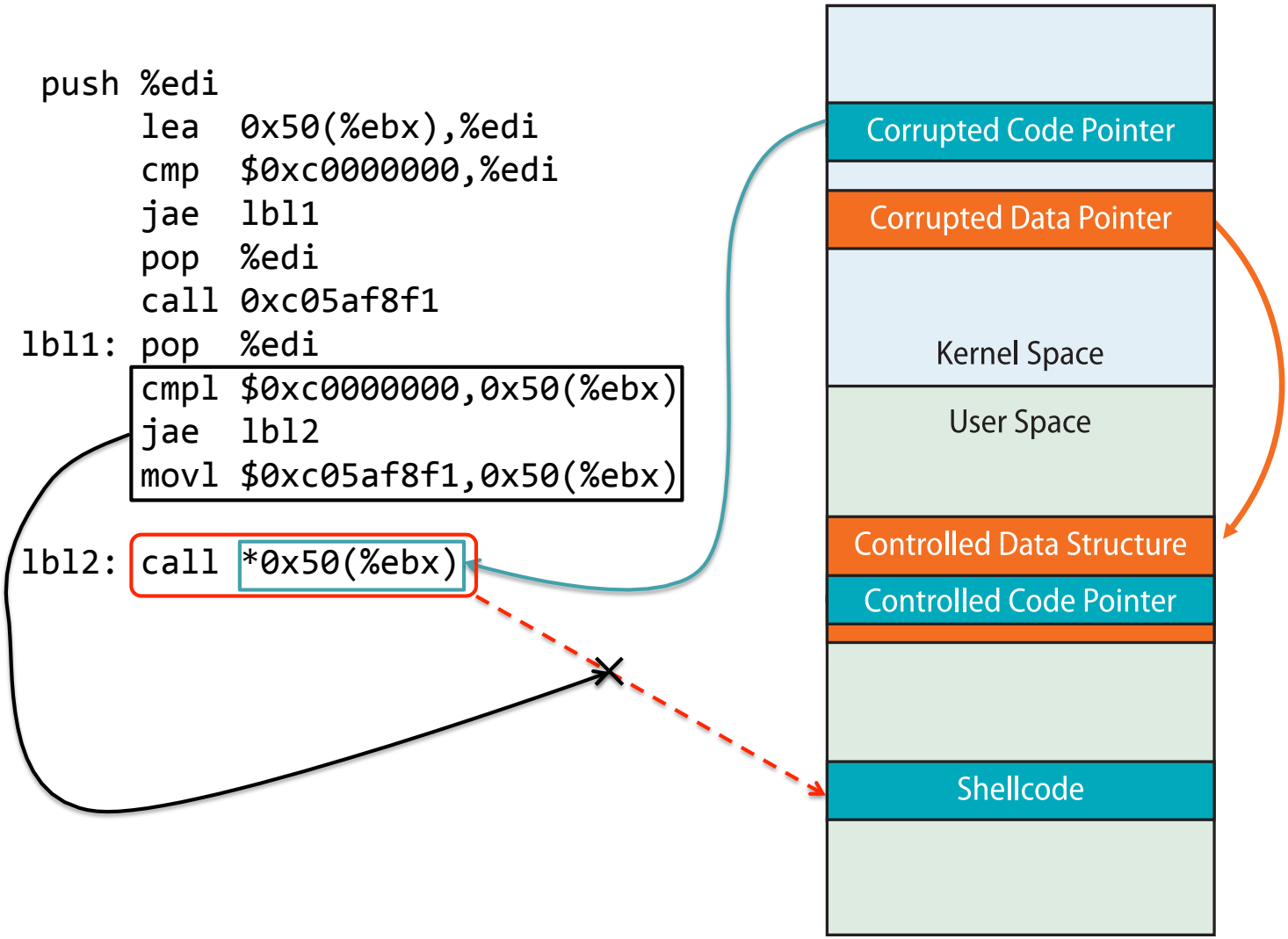



Indirect call in `drivers/cpufreq/cpufreq.c` (x86 Linux)

BROWN

# kGuard Design (cont'd)

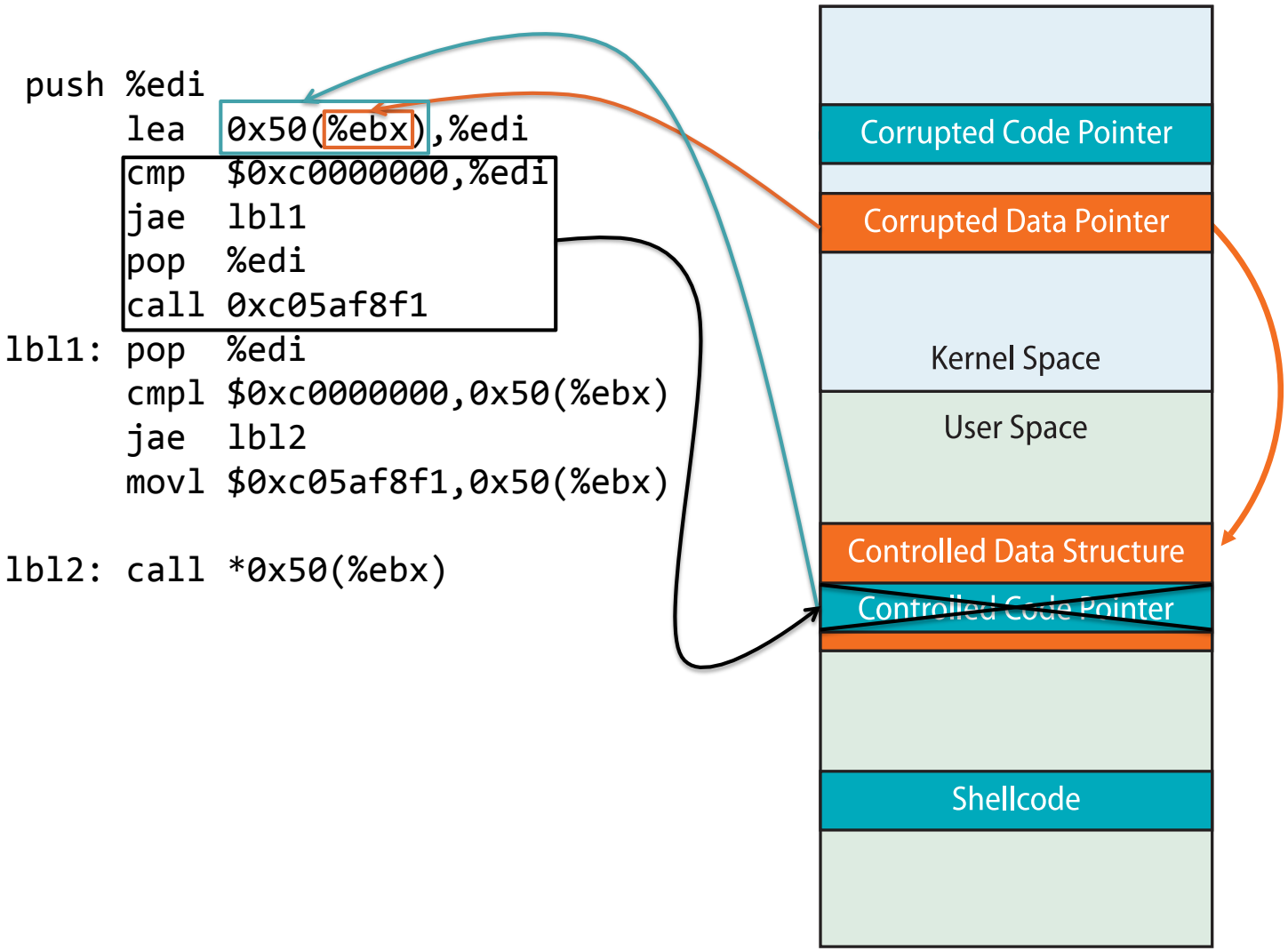
## $\mathrm{CFA_M}$ example




Indirect call in `net/socket.c` (x86 Linux)

# kGuard Design (cont'd)

## $\mathrm{CFA_M}$ example




Indirect call in `net/socket.c` (x86 Linux)

# Bypassing kGuard

## Bypass trampolines

- CFAs provide reliable protection if the attacker **partially** controls a computed branch target
- What about vulnerabilities that allow overwriting kernel memory with **arbitrary** values?

BROWN

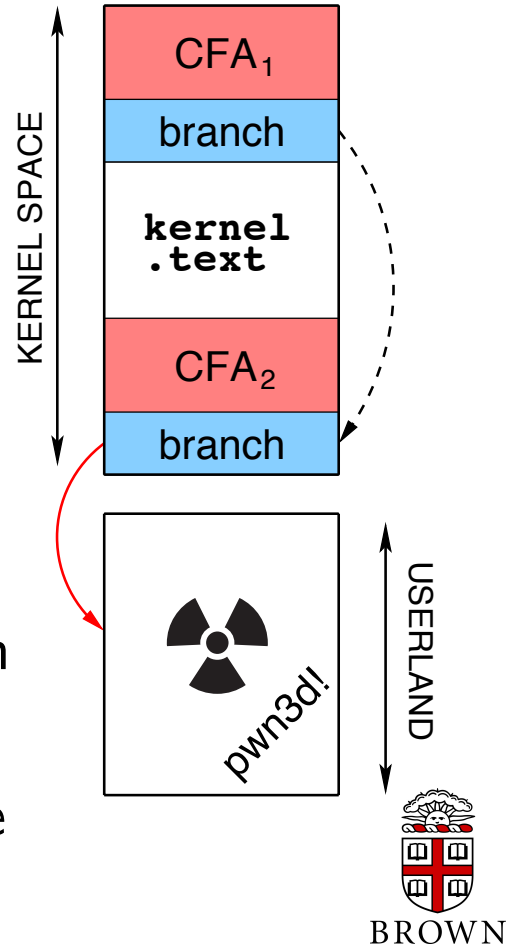
# Bypassing kGuard

## Bypass trampolines

- CFAs provide reliable protection if the attacker **partially** controls a computed branch target
- What about vulnerabilities that allow overwriting kernel memory with **arbitrary** values?

### Attacking kGuard

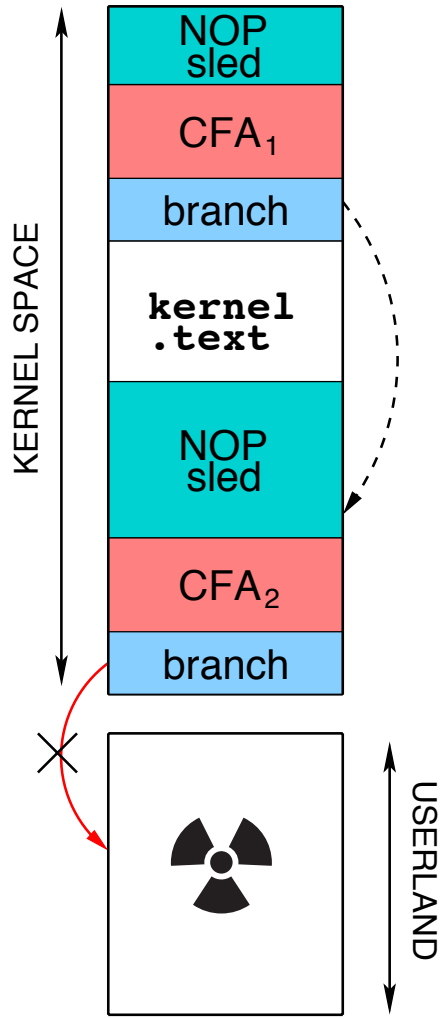
1. Find **two** computed branch instructions whose operands can be reliably overwritten
2. Overwrite the value (branch target) of the first with the address of the second
3. Overwrite the value of the second with a user-space address



# Countermeasures

## Code inflation (key technology #2; diversification)

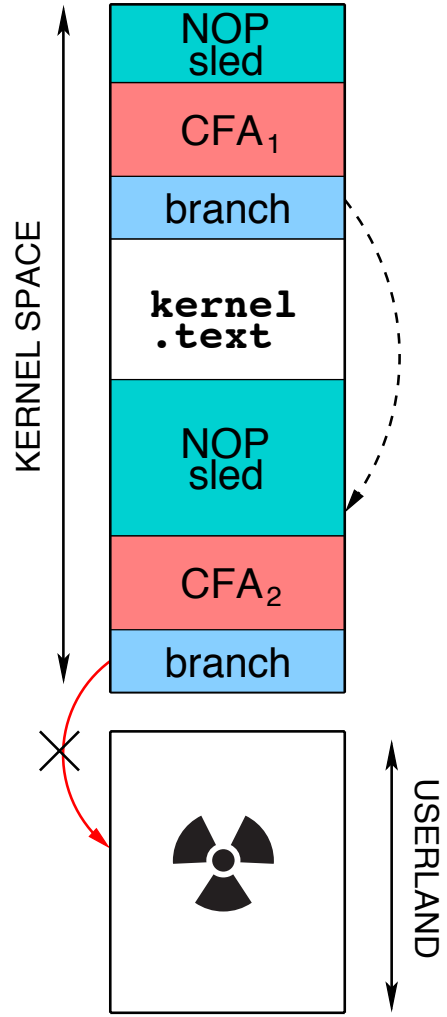
- Reshape kernel's `.text` area
  - Insert a random `NOP` sled at the **beginning** of `.text`
  - Inject a `NOP` sled of random length **before** every CFA
- Each `NOP` sled "pushes" further instructions at higher memory addresses (cumulative effect)



# Countermeasures

## Code inflation (key technology #2; diversification)

- Reshape kernel's `.text` area
  - Insert a random `NOP` sled at the **beginning** of `.text`
  - Inject a `NOP` sled of random length **before** every CFA
- Each `NOP` sled "pushes" further instructions at higher memory addresses (cumulative effect)




## Result

- The location of each indirect control transfer is randomized (per build)

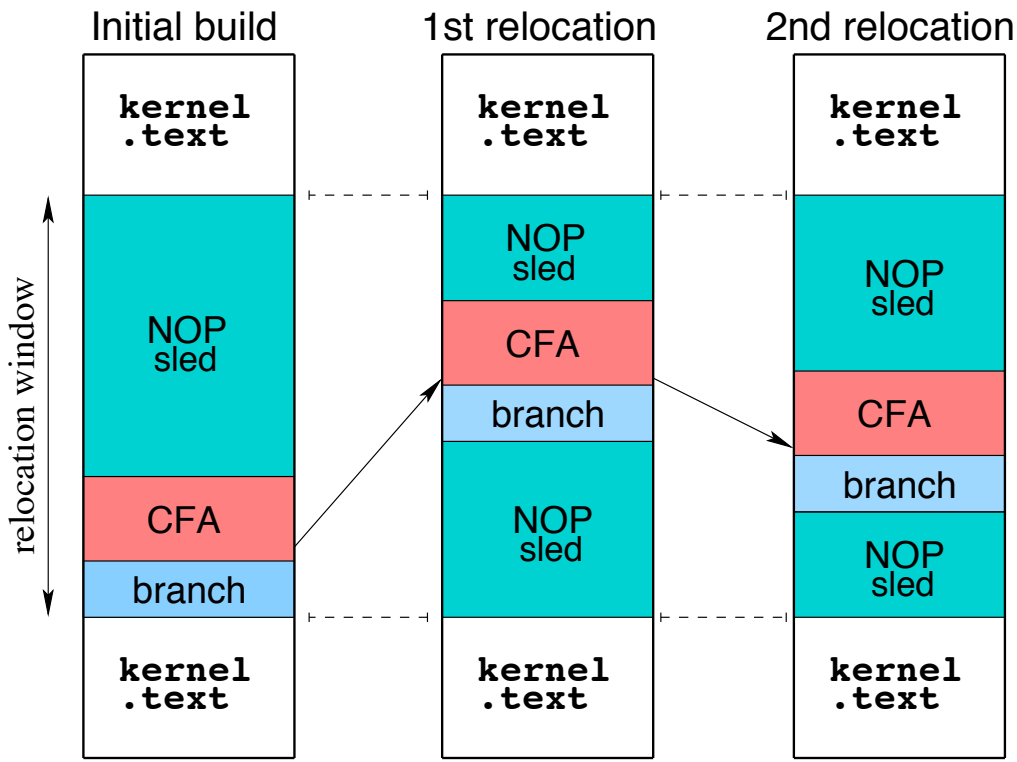
## Important assumption

- Kernel `.text` & symbols secrecy (proper fs privs., `dmesg`, `/proc`)

# Countermeasures (cont'd)

## CFA motion (key technology #3; diversification)

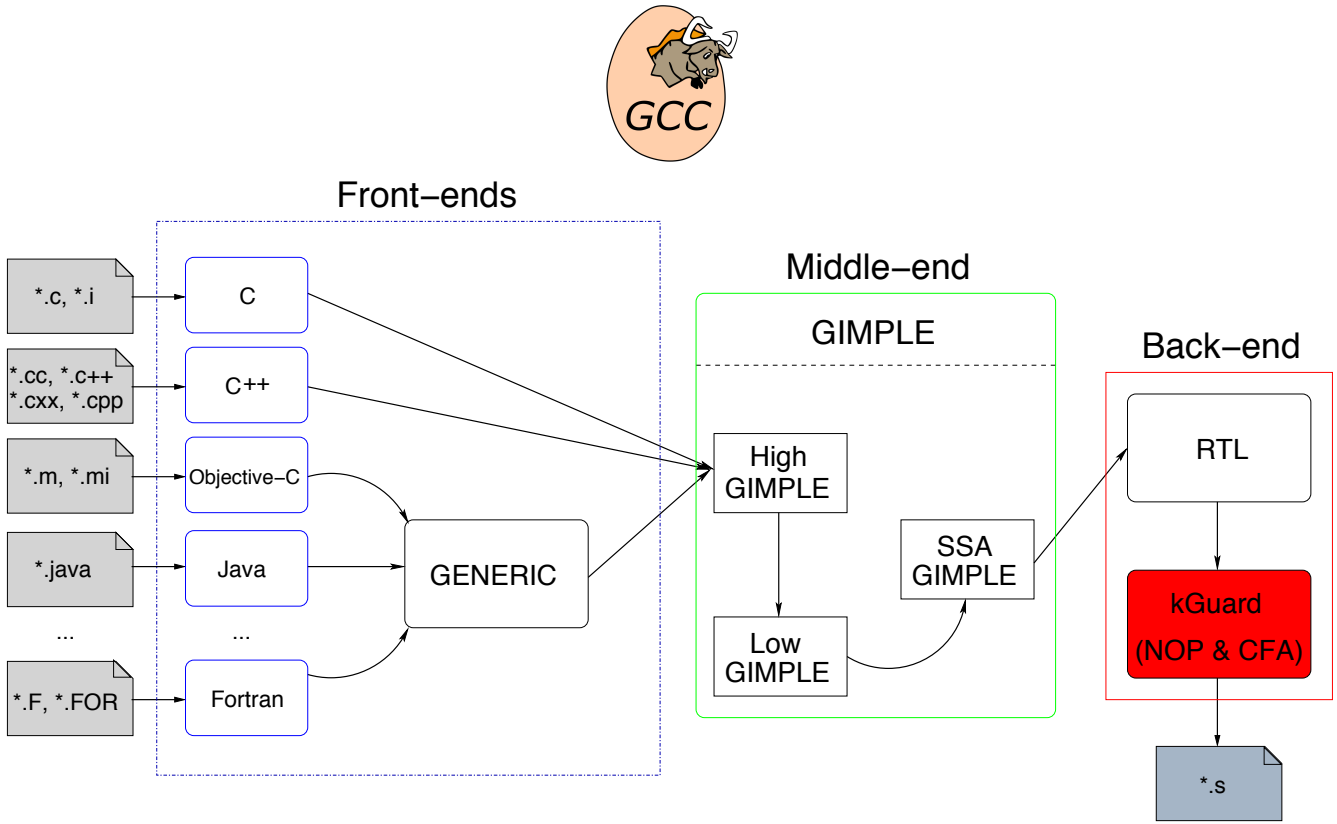
- Relocate the injected guards & protected branches
- Make it harder for an attacker to find a bypass trampoline



# kGuard Implementation

## Fun times with GCC

- Implemented kGuard as a set of modifications to the pipeline of GCC ("de-facto" compiler for Linux, BSD, Android, ...)
- Back-end plugin $\rightarrow \sim$ 1KLOC in C




BROWN

# kGuard Evaluation

## Effectiveness

| Vulnerability | Description | Impact | Exploit | |
|---|---|---|---|---|
| | | | x86 | x86-64 |
| CVE-2009-1897 | NULL *function* pointer | 2.6.30–2.6.30.1 | ✓ | N/A |
| CVE-2009-2692 | NULL *function* pointer | 2.6.0–2.6.30.4 | ✓ | ✓ |
| CVE-2009-2908 | NULL *data* pointer | $\leq$ 2.6.31 | ✓ | ✓ |
| CVE-2009-3547 | *data* pointer corruption | $\leq$ 2.6.32-rc6 | ✓ | ✓ |
| CVE-2010-2959 | *function* pointer overwrite | 2.6.{27.x, 32.x, 35.x} | ✓ | N/A |
| CVE-2010-4258 | *function* pointer overwrite | $\leq$ 2.6.36.2 | ✓ | ✓ |
| EDB-15916 | NULL *function* pointer overwrite | $\leq$ 2.6.34 | ✓ | ✓ |
| CVE-2009-3234 | ret2usr via kernel stack buffer overflow | 2.6.31 | ✓ | ✓ |

✓: detected and prevented successfully, N/A: exploit unavailable

# kGuard Evaluation

## Effectiveness

| Vulnerability | Description | Impact | Exploit | |
|---|---|---|---|---|
| | | | x86 | x86-64 |
| CVE-2009-1897 | NULL *function* pointer | 2.6.30–2.6.30.1 | ✓ | N/A |
| CVE-2009-2692 | NULL *function* pointer | 2.6.0–2.6.30.4 | ✓ | ✓ |
| CVE-2009-2908 | NULL *data* pointer | $\leq$ 2.6.31 | ✓ | ✓ |
| CVE-2009-3547 | *data* pointer corruption | $\leq$ 2.6.32-rc6 | ✓ | ✓ |
| CVE-2010-2959 | *function* pointer overwrite | 2.6.{27.x, 32.x, 35.x} | ✓ | N/A |
| CVE-2010-4258 | *function* pointer overwrite | $\leq$ 2.6.36.2 | ✓ | ✓ |
| EDB-15916 | NULL *function* pointer overwrite | $\leq$ 2.6.34 | ✓ | ✓ |
| CVE-2009-3234 | ret2usr via kernel stack buffer overflow | 2.6.31 | ✓ | ✓ |

✓: detected and prevented successfully, N/A: exploit unavailable

# kGuard Evaluation (cont'd)

## Macro benchmarks

| App. (Bench.) | x86 | x86-64 |
|---|---|---|
| Kernel build (`time(1)`) | 1.03% | 0.93% |
| MySQL (`sql-bench`) | 0.92% | 0.85% |
| Apache (`ApacheBench`) | $\leq$0.01% | $\leq$0.01% |

Impact on real-life applications: $\sim$ **1%**

# Summary

## kGuard

- Versatile & lightweight mechanism against `ret2usr` attacks
- Builds upon inline monitoring and code diversification
  - Control-flow assertions (CFAs)
  - Code inflation & CFA motion
- Cross-platform solution
  - x86, x86-64, ARM, ...
  - Linux, Android, {Free, Open}BSD, ...
- Non-intrusive & low overhead
  - $\sim$ 1% on real-life applications

BROWN

# ret2usr Defenses

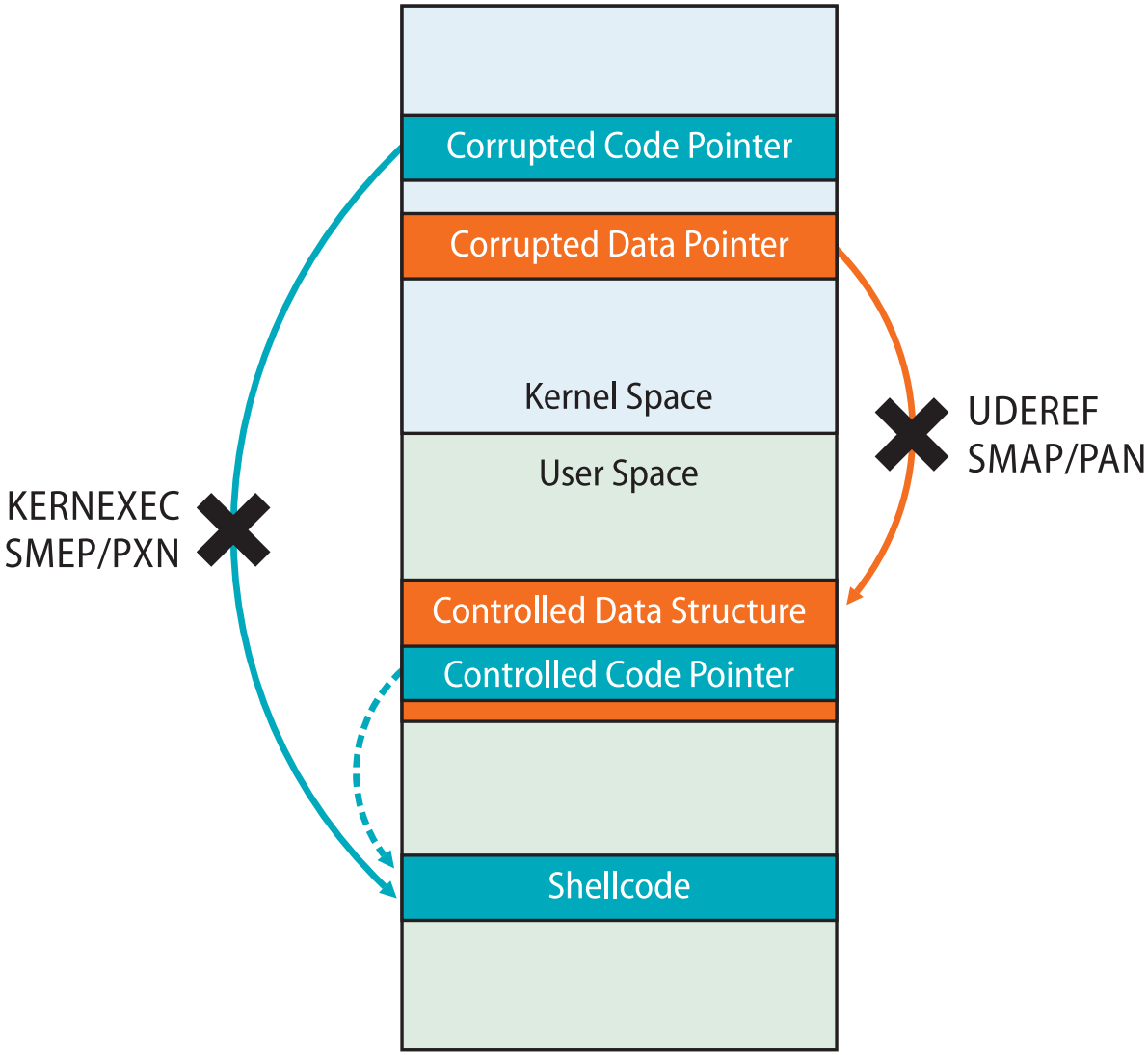
## State of the art overview

- ✓ KERNEXEC/UDEREF → PaX 
  - 3rd-party Linux patch(es) → x86-64/x86/AArch32 only
  - HW/SW-assisted address space separation
    - x86 ⇝ Seg. unit (reload %cs, %ss, %ds, %es)
    - x86-64 ⇝ Code instr. & temporary user space re-mapping
    - ARM (AArch32) ⇝ ARM domains
- ✓ SMEP/SMAP, PXN/PAN → intel SMEP, ARM POWERED
  - HW-assisted address space separation
    - Access violation if priv. code (ring 0) executes/accesses instructions/data from user pages (U/S = 1)
  - Vendor and model specific (Intel x86/x86-64, ARM)

BROWN

# ret2usr Defenses

## State of the art overview




BROWN

# Rethinking Kernel Isolation [USENIX Sec '14, BHEU '14]

## Kernel isolation is hard

Focus on `ret2usr` defenses → `SMEP/SMAP`, `PXN/PAN`, PaX, kGuard

# Rethinking Kernel Isolation [USENIX Sec '14, BHEU '14]

## Kernel isolation is hard

Focus on `ret2usr` defenses → `SMEP/SMAP`, `PXN/PAN`, PaX, kGuard

- Can we subvert them?
  - Force the kernel to execute/access user-controlled code/data
- Conflicting design choices or optimizations?
  - "Features" that weaken the (strong) separation of address spaces

BROWN

# Rethinking Kernel Isolation [USENIX Sec '14, BHEU '14]

## Kernel isolation is hard

Focus on `ret2usr` defenses → `SMEP/SMAP`, `PXN/PAN`, PaX, kGuard

- Can we subvert them?
  - Force the kernel to execute/access user-controlled code/data
- Conflicting design choices or optimizations?
  - "Features" that weaken the (strong) separation of address spaces

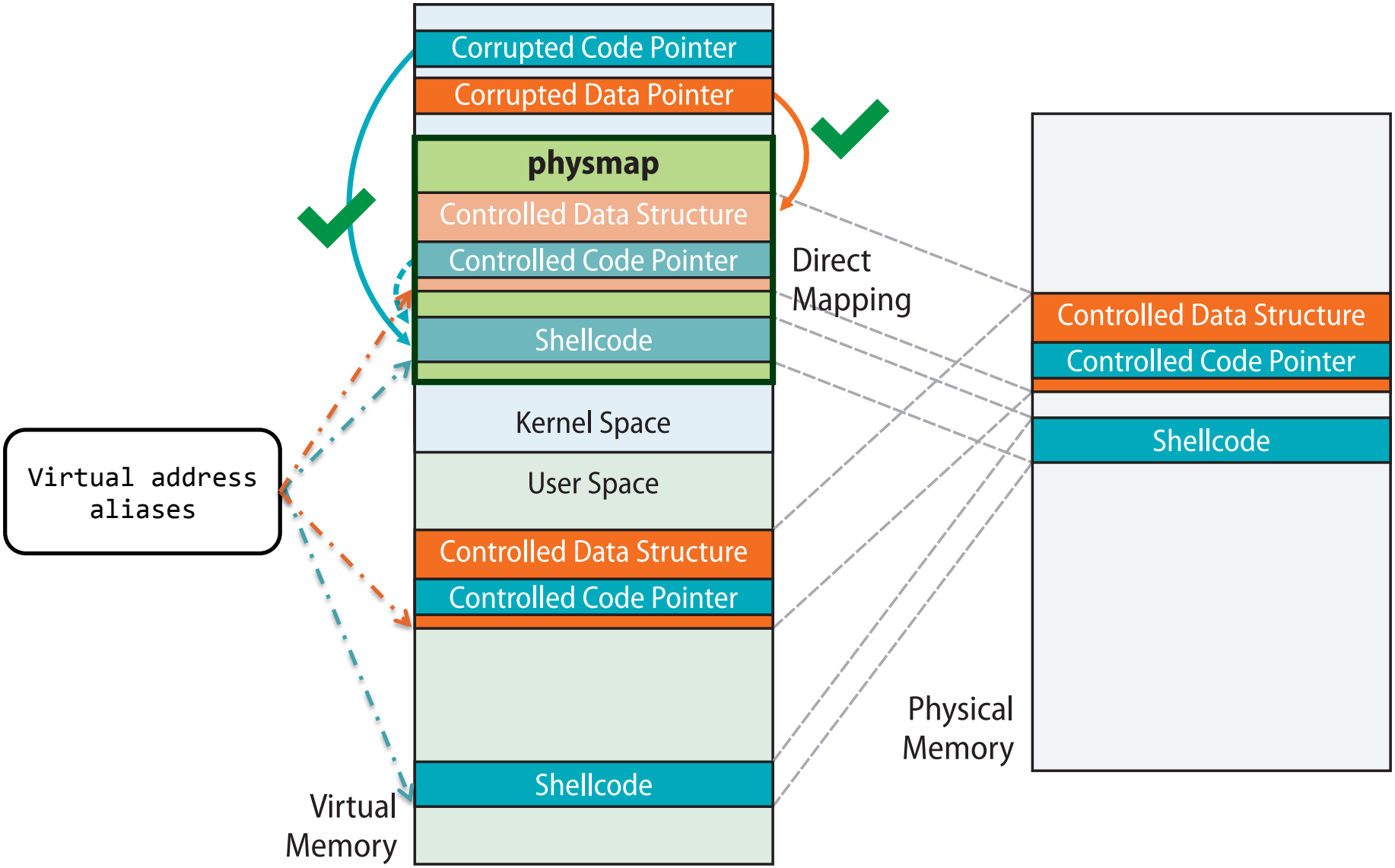
Return-to-direct-mapped memory (**`ret2dir`**)

- Attack against hardened (Linux) kernels
  - ✓ Bypasses **all** existing `ret2usr` protection schemes
  - ✓ ∀ `ret2usr` exploit ⇝ ∃ `ret2dir` exploit


BROWN

# The ret2dir Attack

## Operation



# The ret2dir Attack (cont'd)

## Impact

### Press

- **Reddit.** *ret2dir: Deconstructing Kernel Isolation.* http://goo.gl/wslaaQ
- **Hacker News.** *ret2dir: Rethinking Kernel Isolation.* http://goo.gl/ON1wyk
- **Dark Reading.** *Black Hat Europe 2014: Gullible Computers.* http://goo.gl/DniZ7O

### Research

- 
  - Aaron Adams, NCC Group ⇝ https://goo.gl/QKDFUp
  - "**Xen SMEP (and SMAP) bypass**"
- 
  - Wen Xu & Yubin Fu, Keen Team ⇝ https://goo.gl/iwp3Lk
  - "**Own Your Android! Yet Another Universal Root**"

# The ret2dir Attack (cont'd)

## Impact

### Industry

- QUALCOMM
  - ✓ Non-executable physmap on MSM Android ⇝ http://goo.gl/NL0L3D
- OpenBSD
  - ✓ Non-executable pmap on x86-64, PPC ⇝ http://goo.gl/vskTwA
- 
  - ✓ Restrict (?) /proc/<pid>/pagemap ⇝ https://goo.gl/ctMy8R

BROWN

# Summary

- Kernel isolation is hard
  - ✗ **Shared kernel/process model** → **`ret2usr`**
  - ✗ **`physmap` region(s) in kernel space** → **`ret2dir`**
  - ✗ **...** → **`ret2...?`**

BROWN

# Summary

- Kernel isolation is hard
  - ✗ **Shared kernel/process model** → **`ret2usr`**
  - ✗ **`physmap` region(s) in kernel space** → **`ret2dir`**
  - ✗ **...** → **`ret2...?`**
- **kGuard** ⇝ Versatile & lightweight mechanism against `ret2usr`
  - ∼ 1% on real-life applications

BROWN

# Summary

- Kernel isolation is hard
  - ✗ **Shared kernel/process model** → **`ret2usr`**
  - ✗ **`physmap` region(s) in kernel space** → **`ret2dir`**
  - ✗ **...** → **`ret2...?`**
- **kGuard** ⇝ Versatile & lightweight mechanism against `ret2usr`
  - ∼ 1% on real-life applications
- **`ret2dir`** ⇝ Deconstructs the isolation guarantees of `ret2usr` protections (`SMEP`/`SMAP`, `PXN`, PaX, kGuard)

BROWN

# Summary

- Kernel isolation is hard
  - ✗ **Shared kernel/process model** → **`ret2usr`**
  - ✗ **`physmap` region(s) in kernel space** → **`ret2dir`**
  - ✗ **...** → **`ret2...?`**
- **kGuard** ⇝ Versatile & lightweight mechanism against `ret2usr`
  - ∼ 1% on real-life applications
- **`ret2dir`** ⇝ Deconstructs the isolation guarantees of `ret2usr` protections (`SMEP`/`SMAP`, `PXN`, PaX, kGuard)

Liked today's lecture?

Register for **CSCI1951-H**: Software Security and Exploitation

BROWN