Systems and Internet Infrastructure Security

Network and Security Research Center
Department of Computer Science and Engineering
Pennsylvania State University, University Park PA

Advanced Systems Security: Hardware Security

Trent Jaeger
Systems and Internet Infrastructure Security (SIIS) Lab
Computer Science and Engineering Department
Pennsylvania State University

Security Problems

PENNSTATE 1855

- We have discussed lots of security problems
  - Malware on your computer
  - Attacks on memory errors
  - Return-oriented attacks
  - Compromised software
  - Compromised operating systems, etc.
- Is there any way new hardware features could prevent some attack vectors?

Hardware Features

PENNSTATE 1855

- ARM TrustZone
  - Restrict execution of compromised operating systems
- Intel Processor Trace (IPT)
  - Track control flow events
- Intel Memory Protection Extensions (MPX)
  - Check and enforce memory bounds

Goals

PENNSTATE 1855

- Restrict kernel to only execute approved code
- Monitor kernel operations to enforce security
  - Even when the kernel has been compromised

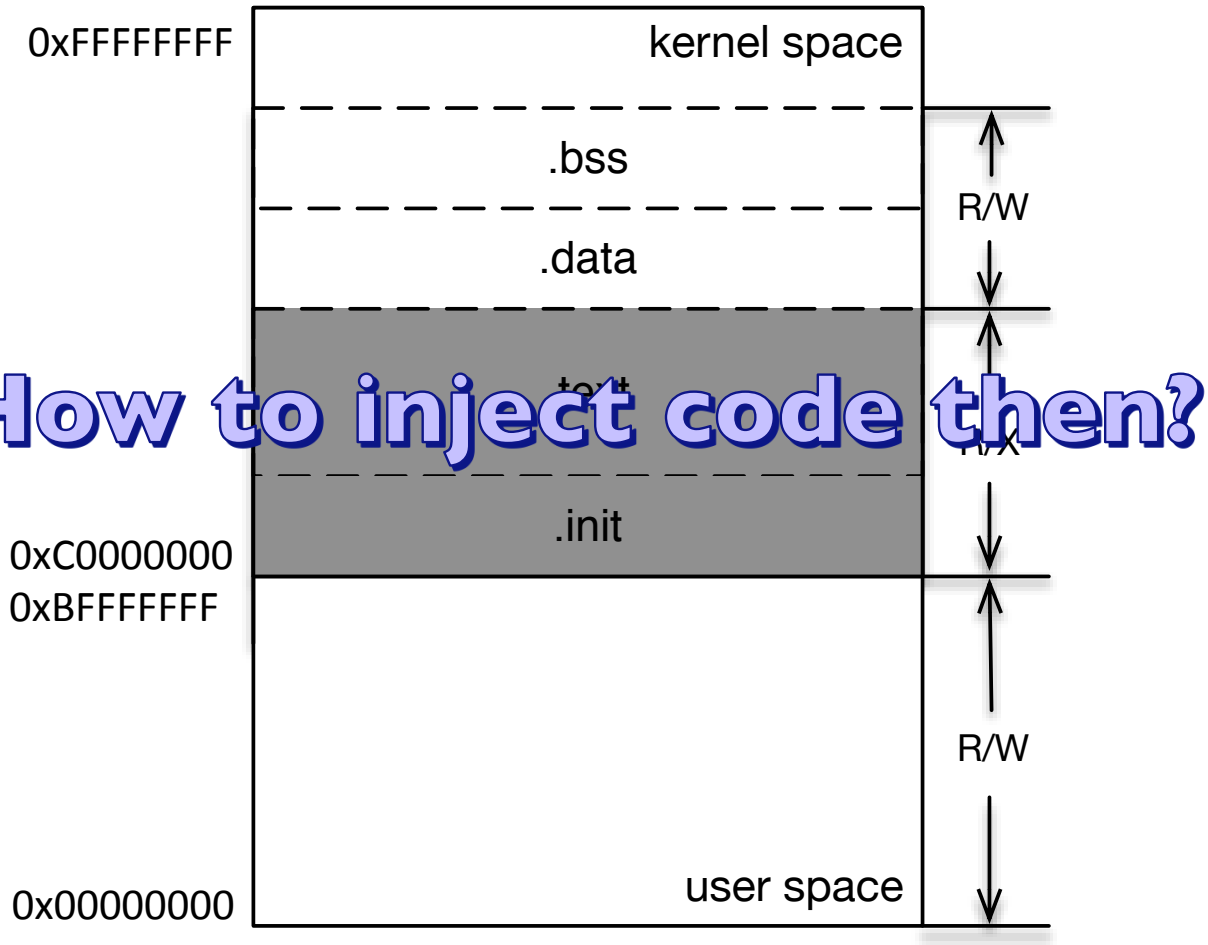
Execution Integrity

PENNSTATE 1855

- All programs run approved code in expected ways
  - Lifetime Code Integrity
    - Even if compromised
    - Restrict execution to **approved code only**
  - Control-Flow Integrity
    - Mediate indirect branches in programs
    - Reject those that are **unexpected**

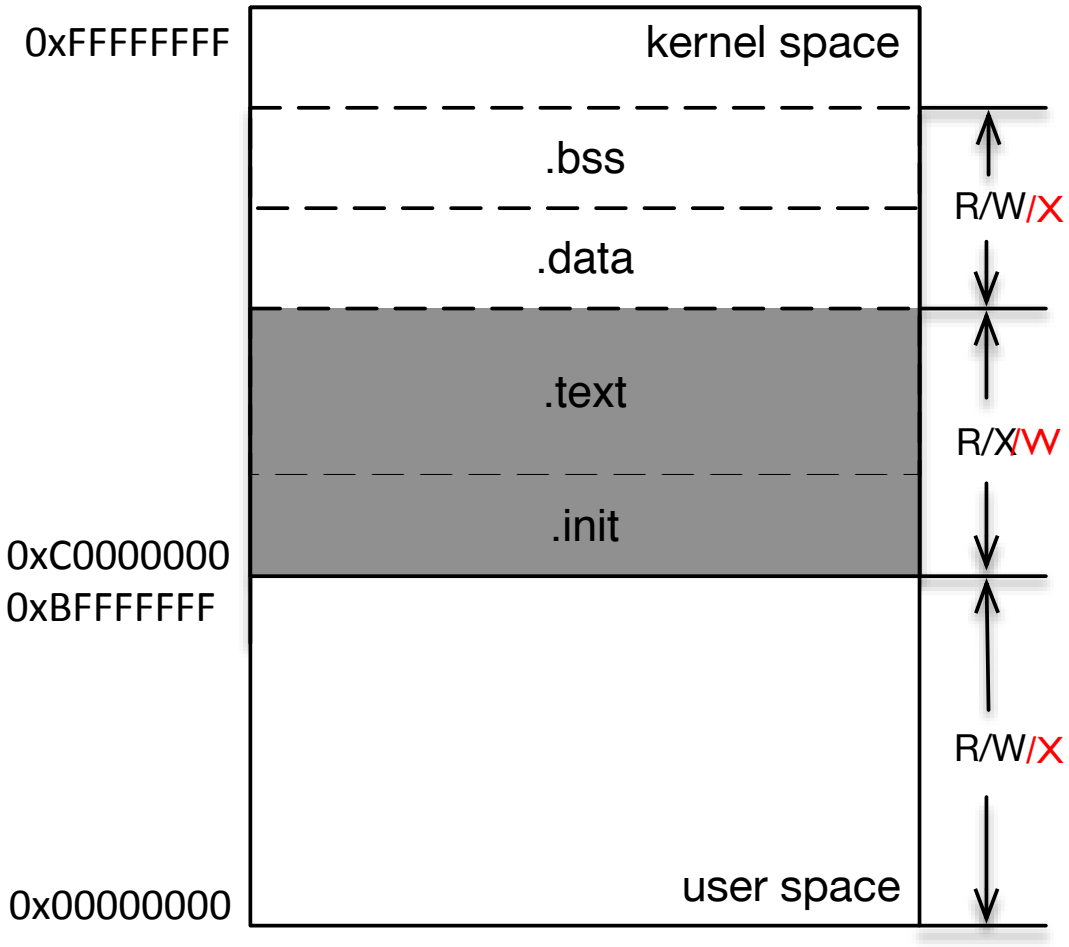
Execution integrity can be enforced efficiently and comprehensively for software systems

Lifetime Kernel Code Integrity



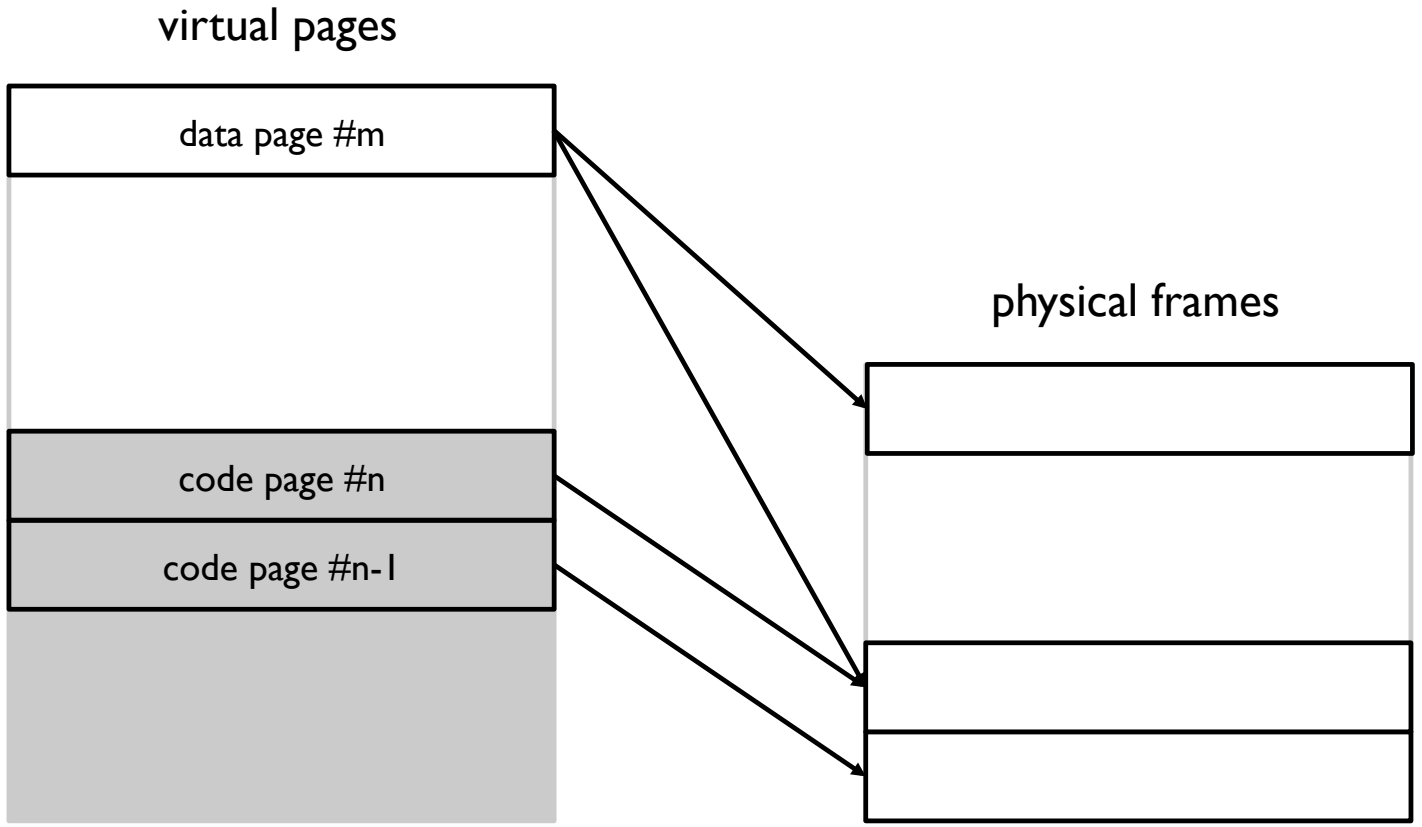
Attack on Permissions

- Tamper with permissions



Attack on Mappings

- Tamper with mappings



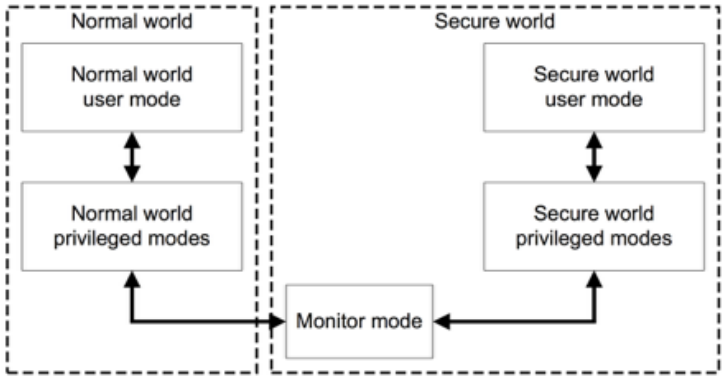
Goal

PENNSTATE 1855

Prevent both types of attacks and limit the adversary to approved kernel code on the TrustZone architecture

Background: TrustZone

- Resources are partitioned into two distinct worlds
  - Physical memory, interrupts, peripherals, etc.
- Each world has its autonomy over its own resources
- Secure world can access normal world resources, but not vice versa
- Run in time-sliced fashion



SPROBE Placement

PENNSTATE 1855

- Recall the specific attacks
  - Change to a different set of page tables that are under attacker's control
    - instrument all instructions that can be potentially used to switch the page table root
  - Modify page table entries in place
    - write-protect the whole page tables and instrument the first instruction in page fault handler

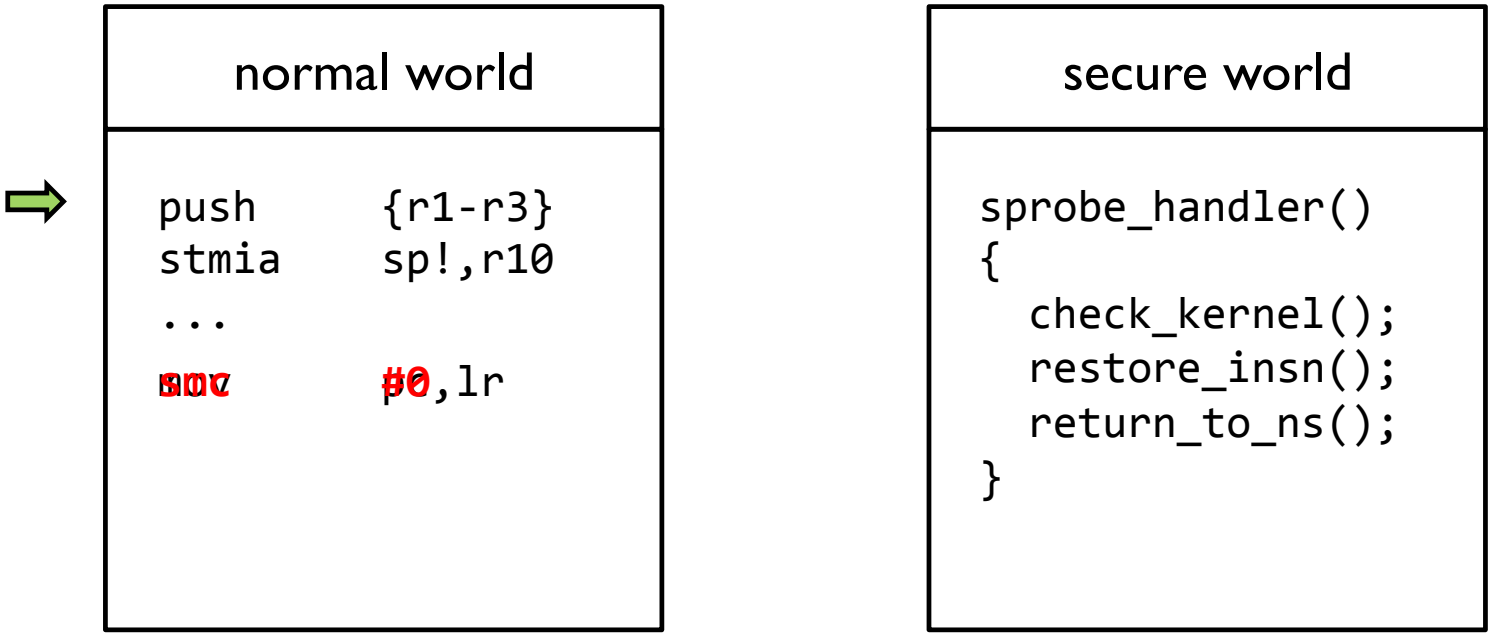
SPROBES Invariants

PENNSTATE 1855

- **S1**: Execution of user space code from the kernel must never be allowed.
- **S2**: W⊕X protection employed by the operating system must always be enabled.
- **S3**: The page table base address must always correspond to a legitimate page table.
- **S4**: Any modification to the page table entry must not make a kernel code page writable or make a kernel data page executable.
- **S5**: MMU must be kept enabled to ensure all existing memory protections function properly.

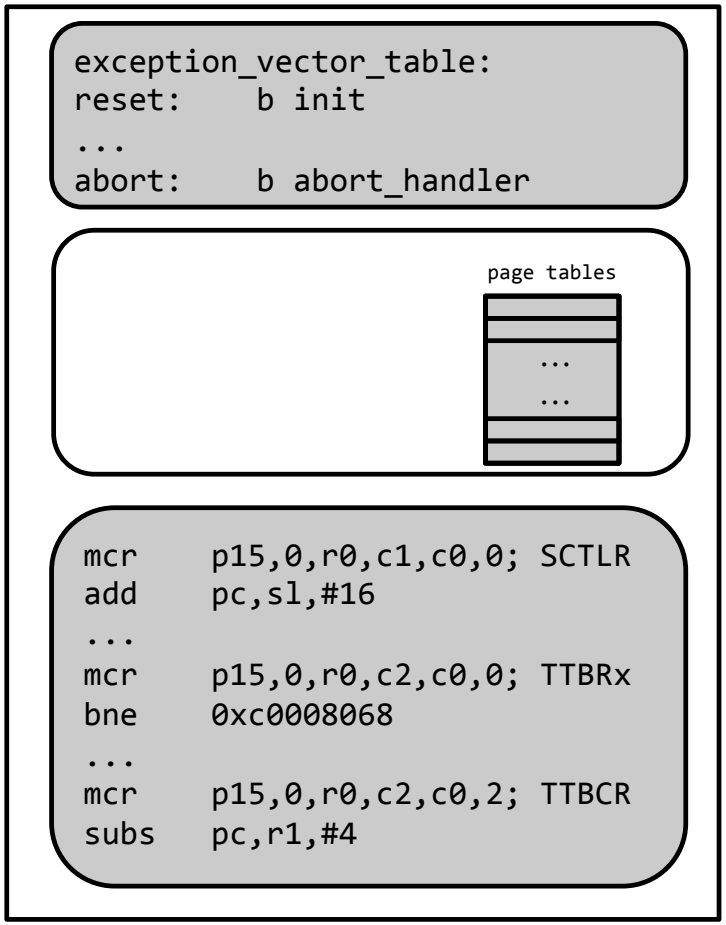
SPROBE Mechanism

- We need an instrumentation mechanism that enables the secure world to be notified upon events of its choice in the normal world



SPROBE Placement

Normal World Kernel Space



Evaluation

PENNSTATE 1855

- Setup
  - Linux 2.6.38 in the normal world
  - Fast Models 8.1 for emulation
- 12 SPROBES are inserted into the Linux kernel
  - 6 for enforcing W⊕X protection and MMU Enable (S2+S5)
  - 4 for monitoring changes to page table root (S3)
  - 1 for monitoring changes to page table configuration (S4)
  - 1 for monitoring modifications to page table entries (S4)
  - Reject page table entries with wrong user/kernel bits (S1)

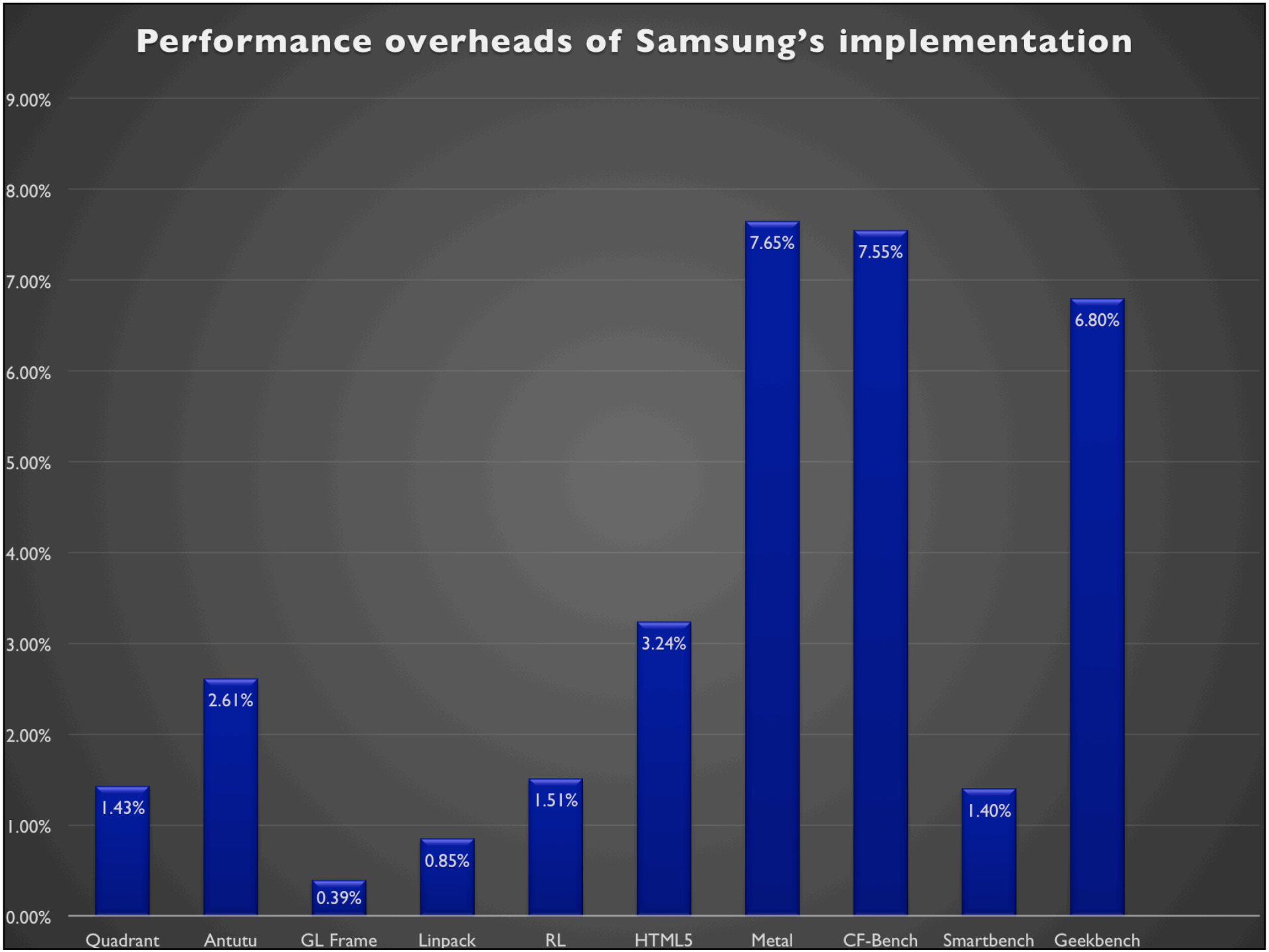
Evaluation

- Setup
  - 3 for enforcing W⊕X protection
  - 4 for monitoring changes to page table root
  - 1 for monitoring changes to page table configuration
  - 1 for monitoring modifications to page table entries

SPROBES Type	Hit Frequency	Overheads
1	N/A	0%
2	313,836	1.8%
3	N/A	0%
4	85,982	6.5%

A Little Bit More…

- Samsung has implemented the same idea and deployed this technique on millions of devices [CCS 2014]


Performance overheads of Samsung's implementation
9.00%
8.00%
7.00%
6.00%
5.00%
4.00%
3.00%
2.00%
1.00%
0.00%
1.43%
2.61%
0.39%
0.85%
1.51%
3.24%
7.65%
7.55%
1.40%
6.80%
Quadrant
Antutu
GL Frame
Linpack
RL
HTML5
Metal
CF-Bench
Smartbench
Geekbench

Another Problem

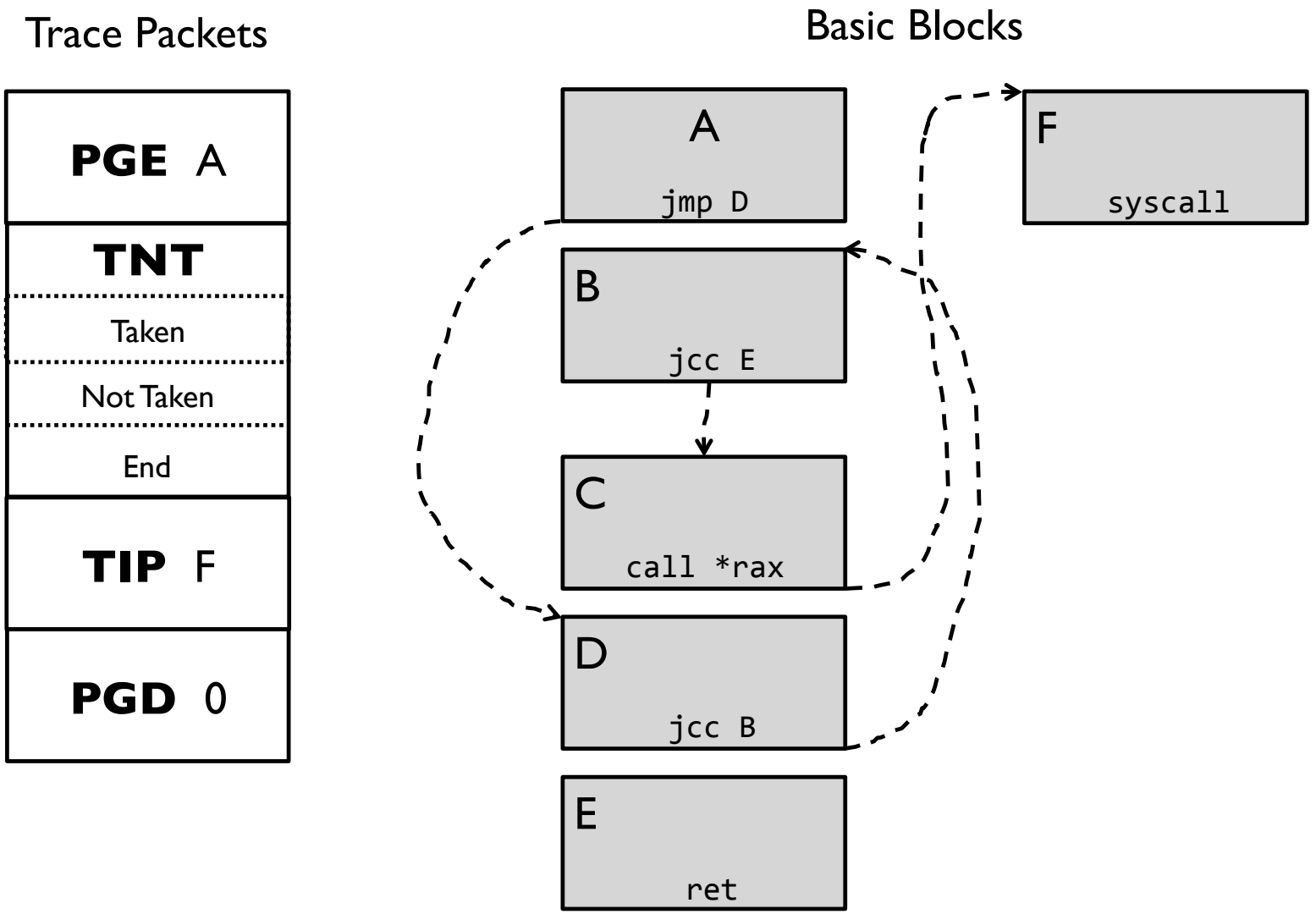
PENNSTATE 1855

- Return-oriented attacks
  - Can hardware help detect those attacks?

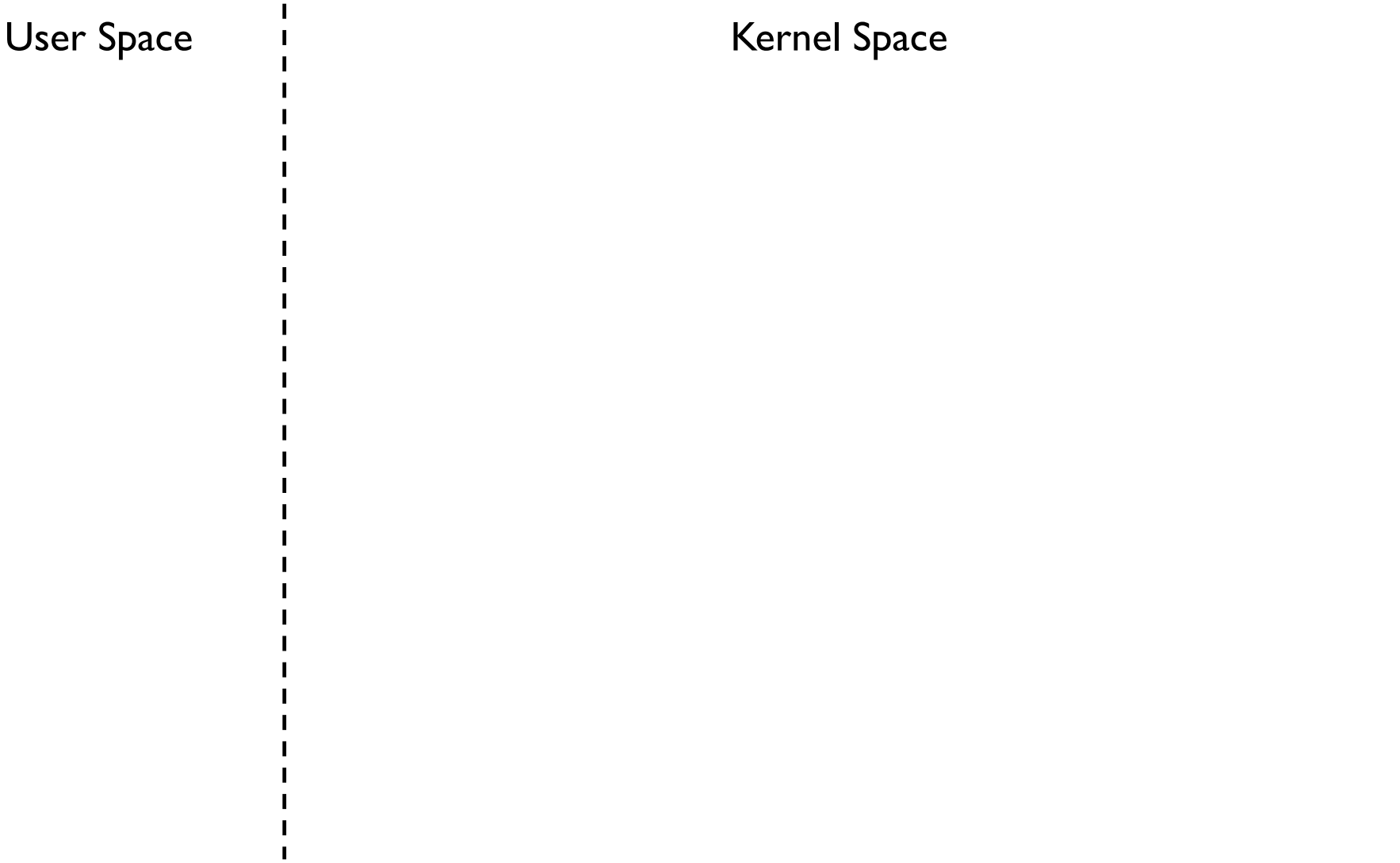
Intel Processor Trace

- A new hardware feature that enables efficient recording of control-flow and timing information about software execution (3-5% overhead)
  - Initially available on the Broadwell processor
  - Fully implemented on the Skylake processor
- At each control choice, record a packet in memory
  - Conditional branches
  - Indirect call
  - Returns
- Enough to reconstruct the actual control flow

Intel PT Example



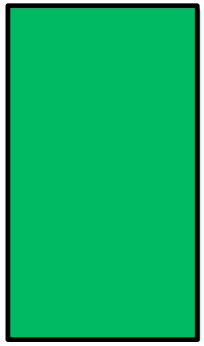
System Overview



System Overview

User Space

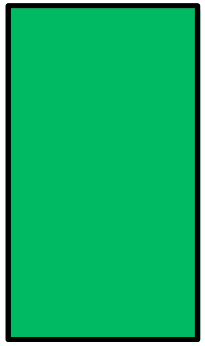
Kernel Space



System Overview

User Space

Kernel Space



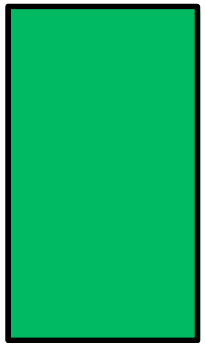
System Overview

User Space

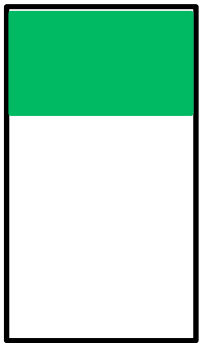
Kernel Space

System Overview

User Space

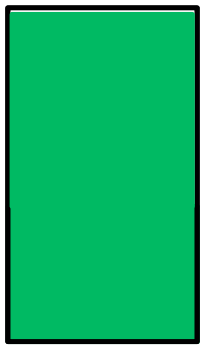


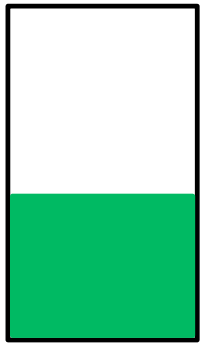
Kernel Space



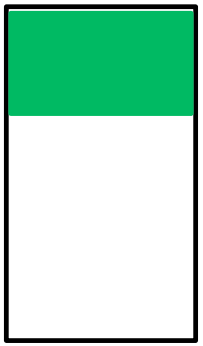
System Overview

User Space





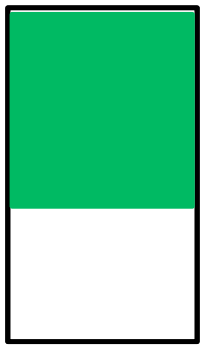
Kernel Space



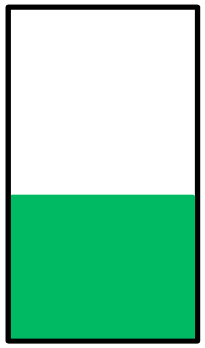
System Overview

User Space

Kernel Space



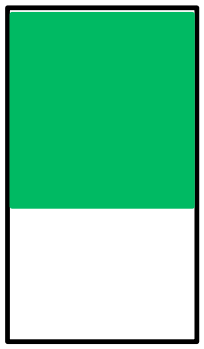
SYSCALL

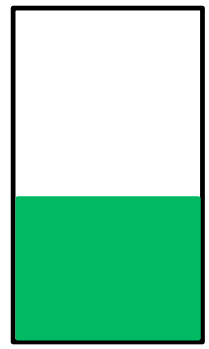


System Overview

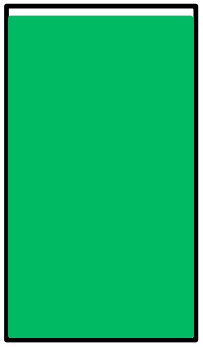
User Space

Kernel Space





SYSCALL



What To Do?

Depends on the enforced policy

CFI Policies

- Coarse-grained Policy
  - Check if the targets of indirect control transfers are valid
  - Requires decoding the trace packets to find each target
- Fine-grained Policy
  - Check if the source and destination are a legitimate pair
  - Requires control-flow recovery to identify source
- Stateful Policy
  - Check if an indirect control transfer is legitimate based on the program state (e.g., shadow stack)
  - Requires sequential processing if state spans trace buffers

CFI Policies

- Coarse-grained Policy
  - Check if the targets of indirect control transfers are valid
  - Requires decoding the trace packets to find each target
- Fine-grained Policy
  - Check if the source and destination are a legitimate pair
  - Requires control-flow recovery to identify source
- Stateful Policy
  - Check if an indirect control transfer is legitimate based on the program state (e.g., shadow stack)
  - Requires sequential processing if state spans trace buffers

Fine-Grained CFI

PENNSTATE 1855

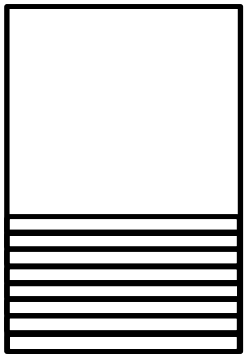
- Recover the control flow from the trace buffer and the program binaries to identify sources
  - Disassemble the binary online in basic blocks
  - Traverse basic blocks using the trace buffer to find sources of indirect control transfers
- Authorize each indirect control transfer target against that program's fine-grained policy for source
  - For each indirect control transfer found in the trace ensure that the destination is in the legal target set of the corresponding source

Control-Flow Recovery

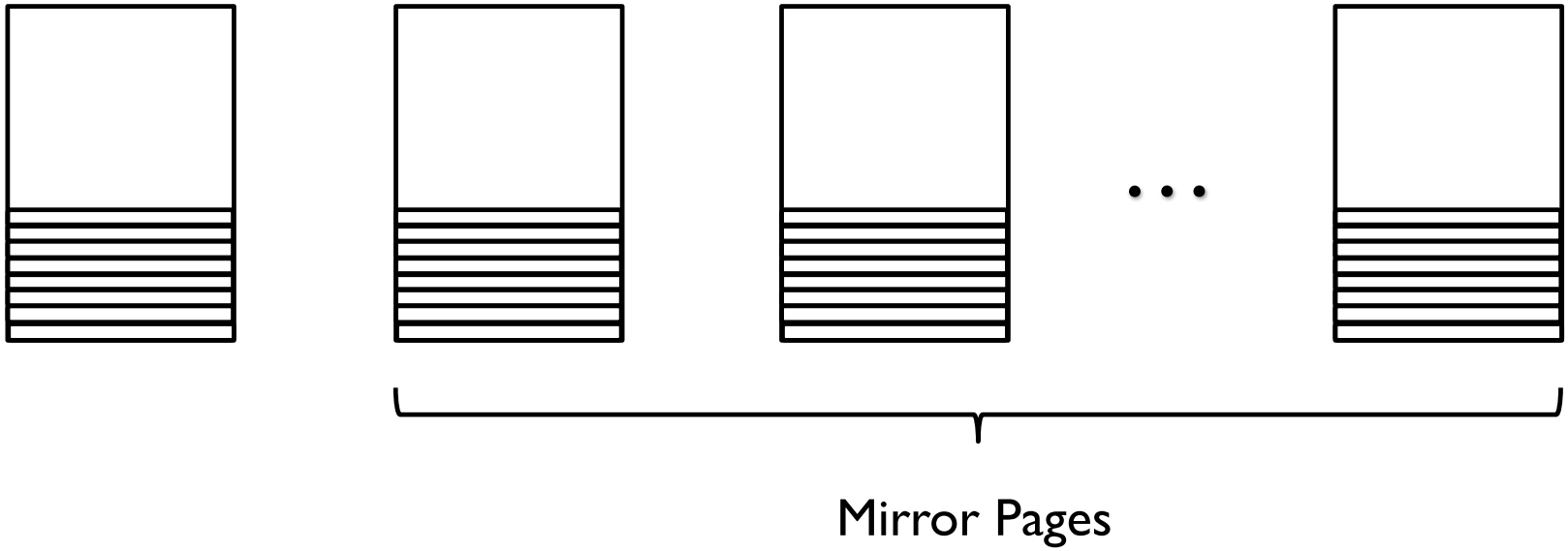
PENNSTATE 1855

- Basic blocks are executed many times in practice, but we only want to disassemble the same basic block once…
- Question: given a block address, how do we find the disassembled information efficiently?
- Hashtable? No!
  - The hash function takes a few cycles
  - A conflict could take even more cycles
  - Requires locking when accessed by multiple threads

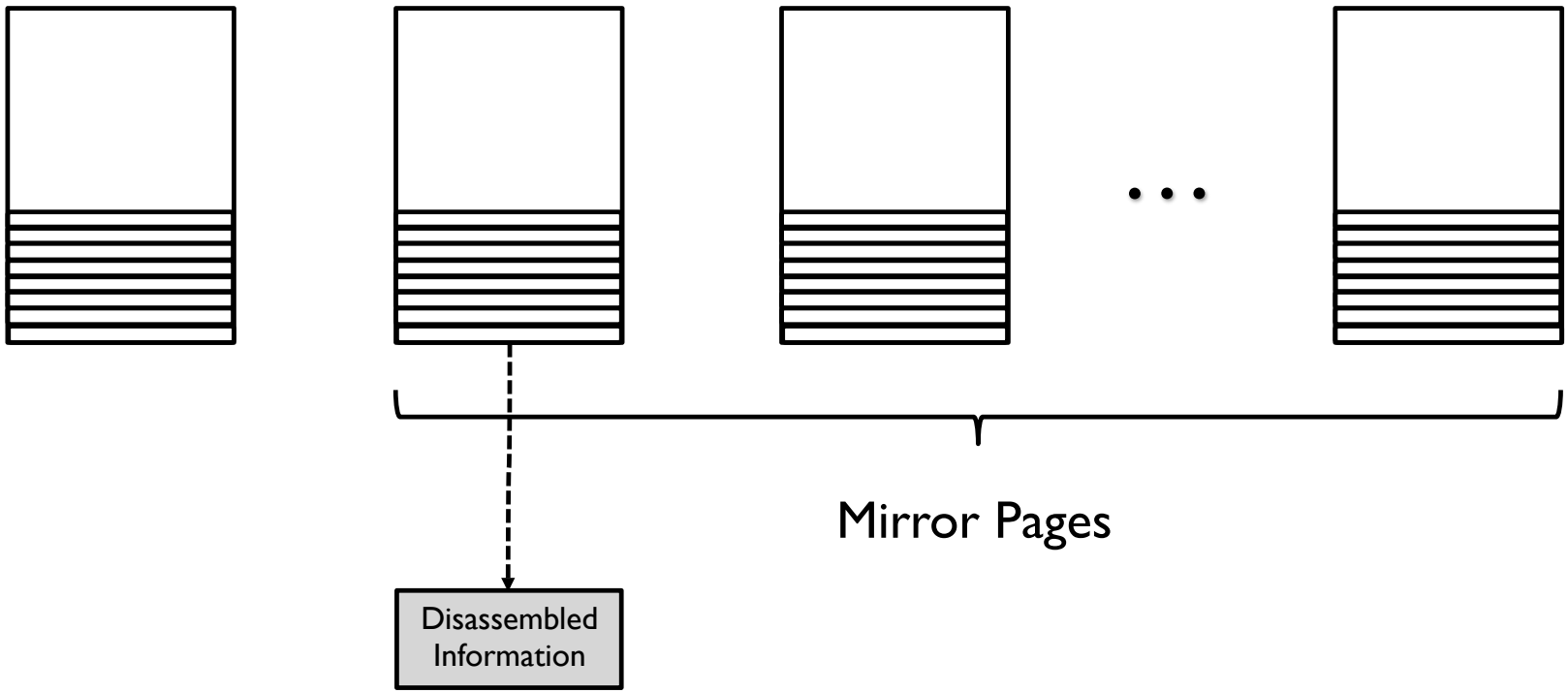
Our Solution: Mirror Pages



Our Solution: Mirror Pages

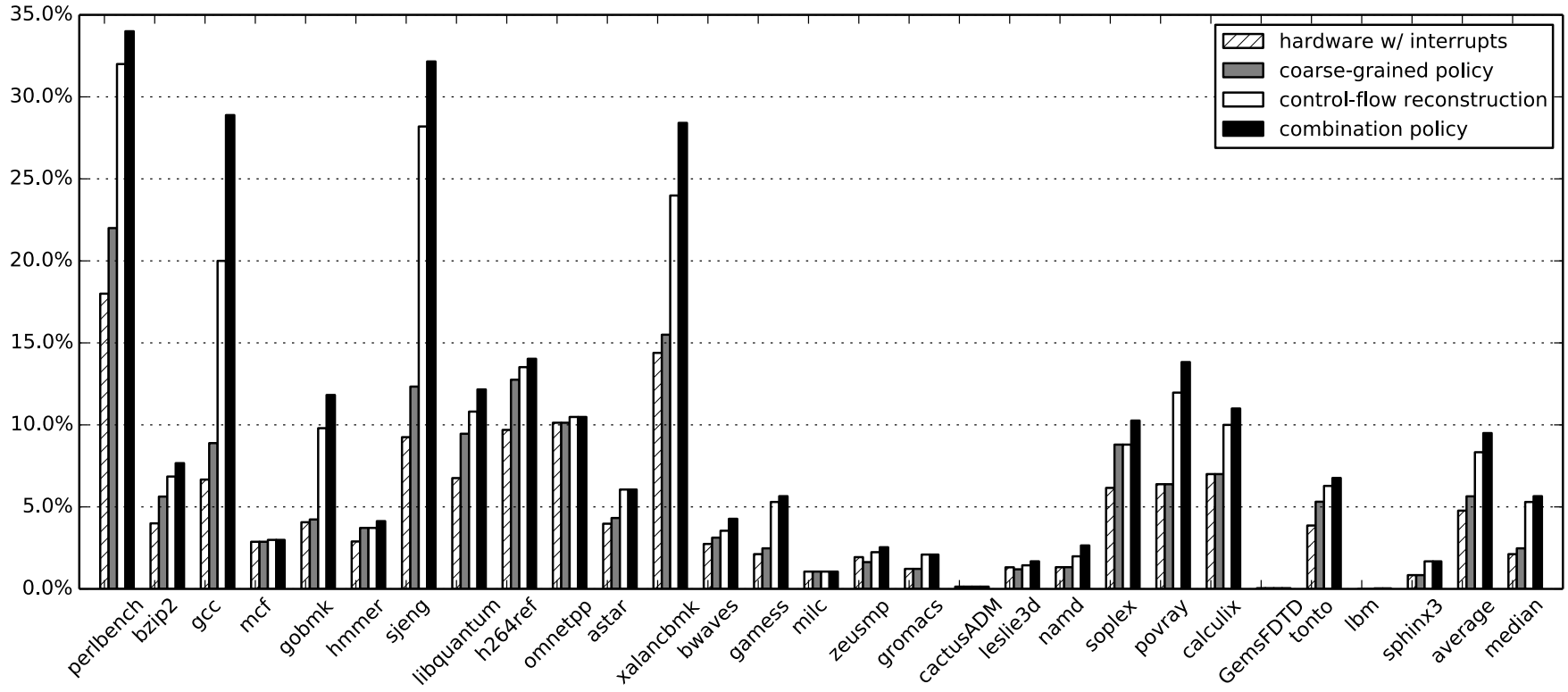


Our Solution: Mirror Pages



Evaluation

- SPEC CPU2006
  - Average: 9.5%, Median: 5.6% for the combination policy
  - Comparable to the state-of-the-art shadow stack impl.



CFI-Focused Logging

PENNSTATE 1855

- What if hardware logging was designed for CFI enforcement?
  - Can we eliminate need for control-flow recovery to enforce fine-grained CFI policies?
    - Just need the source-destination pair for each indirect control transfer? 1 extra packet for each indirect op, no TNT packets
    - Reduce trace size 58% and processing time 92% on average
  - Can we focus control-flow recovery to enforce stateful CFI policies? E.g., shadow-stack and forward edge
    - Shadow stack depends on complete control flow → CET
    - Can enforce PathArmor [CCS 2015] with ~1% of original trace

Intel CET

- Intel Control-Flow Enforcement Technology (CET) aims to enforce shadow stack defenses in hardware
  - Announced in June 2017
- Shadow Stack on backward edge
  - Exception on failure – for handler to deal with
- Indirect Branch Tracking on forward edge
  - Restrict indirect calls/jumps to valid targets
    - Issue: precision of these restrictions
  - Weak – Single class of valid targets for all calls (coarse)

Hardware Security Issues

- Meltdown and Spectre attacks
  - Both based on branch prediction and speculative execution
    - A branch prediction causes a speculative execution to occur that is only committed when the prediction is correct
  - But the speculative execution causes measurable side effects
    - That can enable an adversary to read arbitrary memory from a victim process
- Sound solutions require fixes to processors and updates to ISAs – ad hoc solutions used for now

Spectre Attack

PENNSTATE 1855

- Attacker locates a sequence of instructions within a victim program that would act as a covert channel
  - From knowledge of victim binary
- Attacker tricks the CPU to execute these instructions speculatively and erroneously
  - Leak victims info to measurable channel
    - Cache contents can survive nominal state reversion
- To make real, use a cache-based side channel, such as Flush+Reload

Spectre Attack

- Exploiting Conditional Branches

```
if (x < array1_size)
   y = array2[array1[x] * 256];
```

- Suppose an adversary controls the value of 'x'
- Adversary performs the following sequence
  - First, invoke the program with legal inputs to train the branch predictor to speculatively execution the branch to compute 'y'
  - Next, invoke the program with an 'x' outside bounds of `array1` and where `array1_size` is uncached
  - The operation will read a value from outside the array, and update the cache at a memory location based on the value at `array1[x]`
    - Can learn the value at `array1[x]` from location of cache update

Meltdown

- Meltdown has some similarities

```
1 raise_exception();
2 // the line below is never reached
3 access(probe_array[data * 4096]);
```

- Uses the speculative execution of the above code with an illegal address in 'data' to read arbitrary kernel memory
- Adversary performs the following sequence
  - Set data to a kernel memory address
  - The cache entry corresponding to probe_array(data*4096) will be updated based on the value at 'data'
    - Flush+Reload to detect
- Can leak entire kernel memory

Spectre v Meltdown

PENNSTATE 1855

- Which is worse?
- Meltdown exploits a privilege escalation vulnerability in Intel processors that bypasses kernel memory protections
  - That is a big channel, but only applies to Intel processors
  - Also, the KAISER patch has already been proposed to address the vulnerability being exploited
  - Can be fixed
- Spectre applies to AMD, ARM, and Intel
  - And there is no patch
  - And there are variants that can be exploited – e.g., via JavaScript
  - Do need to find some appropriate victim code tho

Take Away

- Lots of efforts in exploring hardware features to improve security
  - Isolate code from untrusted kernel – SGX and TZ
  - Remote attestation – TPMs
  - Software bounds checking MPX
    - MPX and PT can be applied to CFI enforcement
- However, there are also security issues with hardware
  - Meltdown and Spectre
  - Hardware Trojans