ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

Ward:

Efficiently Mitigating Transient Execution Attacks using the Unmapped Speculation Contract

Jonathan Behrens, Anton Cao, Cel Skeggs, Adam Belay, M. Frans Kaashoek, Nickolai Zeldovich

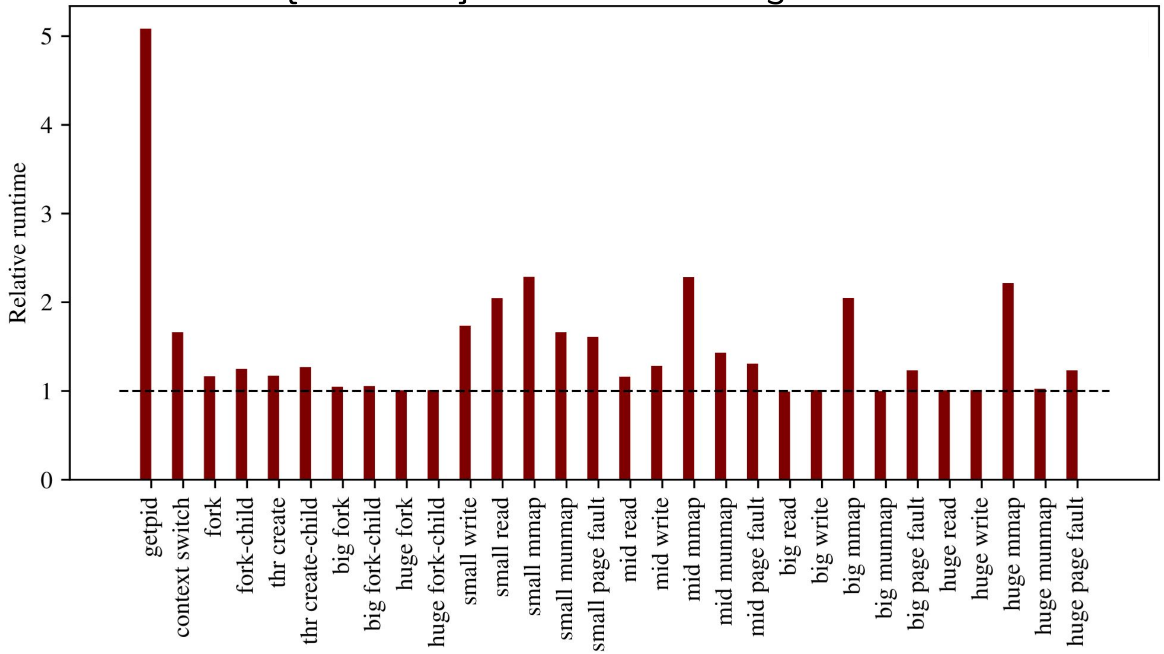
CSAIL

Transient execution attacks risk leaking information

Linux maintains security using software mitigations

Software mitigations are expensive

LEBench [SOSP '19] with/without mitigations on Linux



Goal: faster mitigations

Threat model

- Similar security to Linux

Main ideas

- Unmapped Speculation Contract
- Ward kernel design

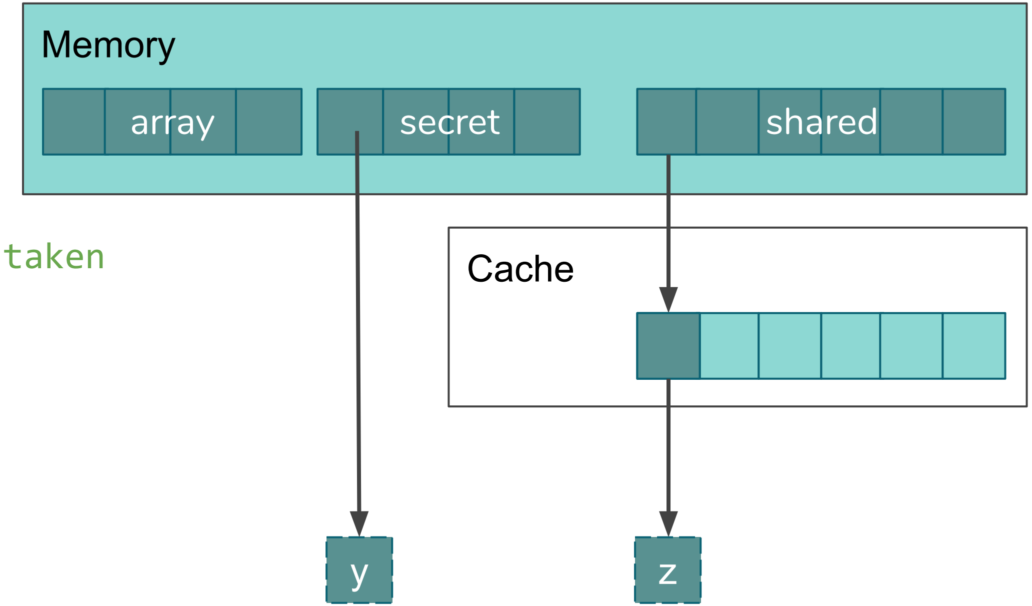
Transient execution attack example

```
char array[SIZE];
int secret;
char shared[256 * CACHE_LINE];

// vulnerable system call code
// if sysarg >= SIZE
if (sysarg < SIZE) { // speculate taken

    y = array[sysarg];
    z = shared[y * CACHE_LINE];
}

// userspace attacker code
secret = is_in_cache(&shared[0]);
```

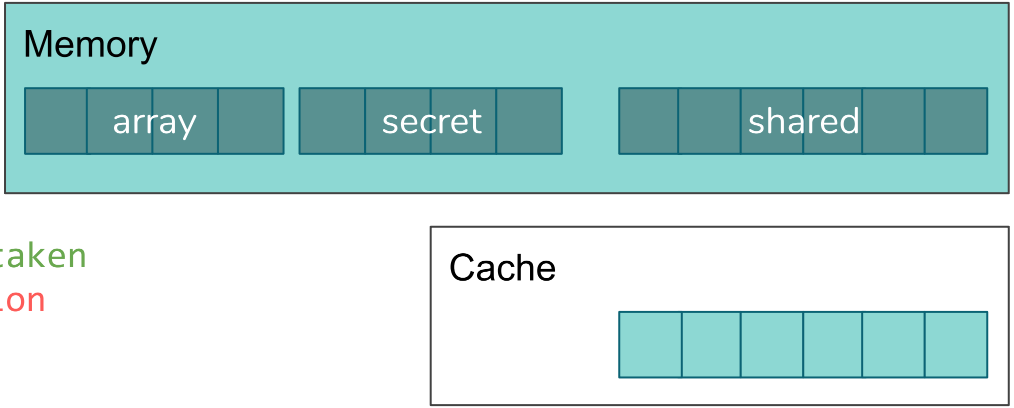


Typical mitigation approach

```
char array[SIZE];
int secret;
char shared[256 * CACHE_LINE];

// vulnerable system call code
// if sysarg >= SIZE
if (sysarg < SIZE) { // speculate taken
    lfence(); // prevents speculation
    y = array[sysarg];
    z = shared[y * CACHE_LINE];
}

// userspace attacker code
secret = is_in_cache(&shared[0]);
```



Ward has a different approach

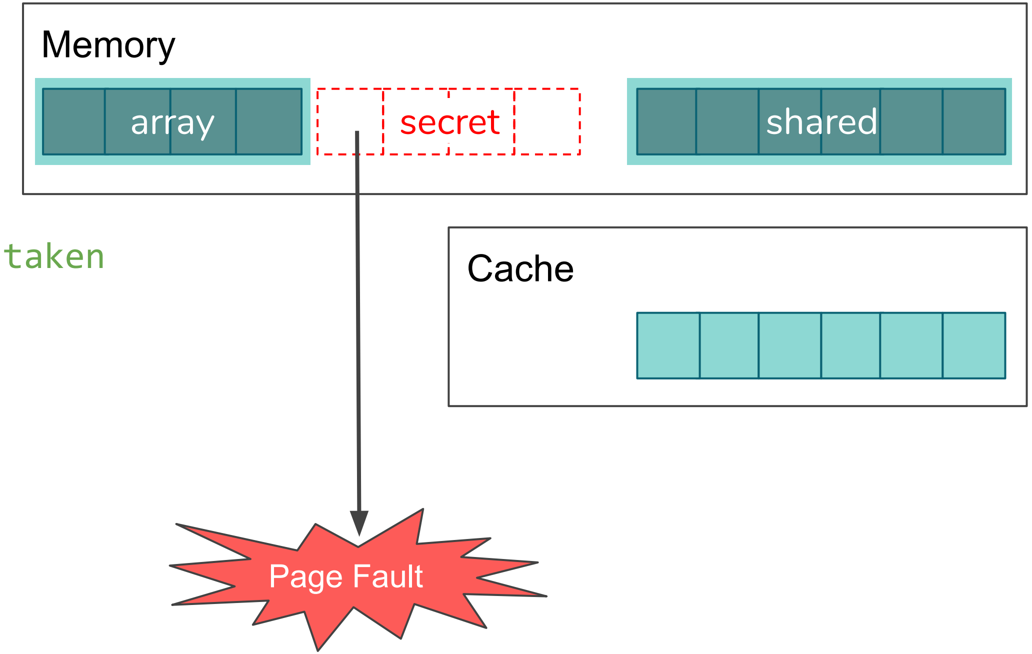
```
char array[SIZE];
int secret;
char shared[256 * CACHE_LINE];

// vulnerable system call code
// if sysarg >= SIZE
if (sysarg < SIZE) { // speculate taken

    y = array[sysarg];
    z = shared[y * CACHE_LINE];
}

// userspace attacker code
secret = is_in_cache(&shared[0]);
```

Secret not mapped...



Our observation: Unmapped Speculation Contract (USC)

If some memory has never been mapped in the current address space…

CPU state should be unaffected by values stored there

USC is a good hardware-software contract

- Allows most speculation
- Processors seem to be able to provide it:

“AMD processors are designed to not speculate into memory that is not valid in the current virtual address memory range defined by the software defined page tables.”

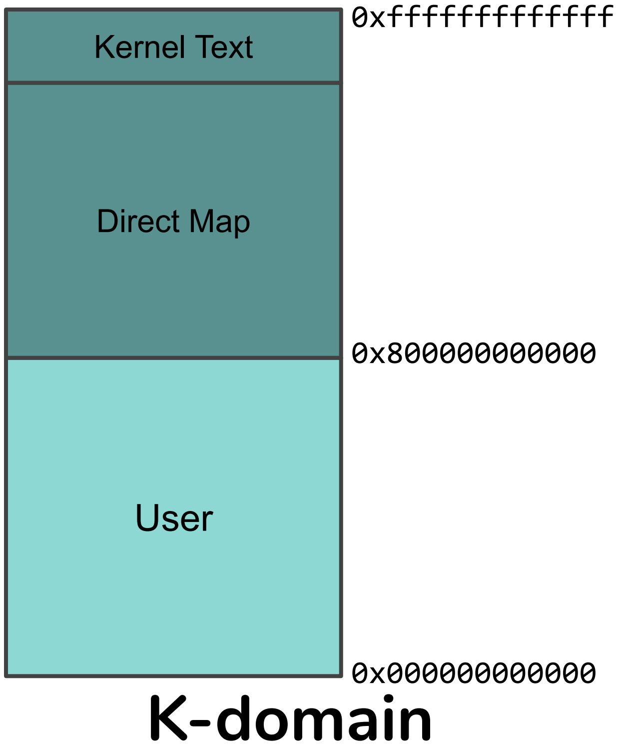
— “Speculation behavior in AMD micro-architectures” white paper

Design

Split kernel to leverage USC

Ward extends Linux's PTI:

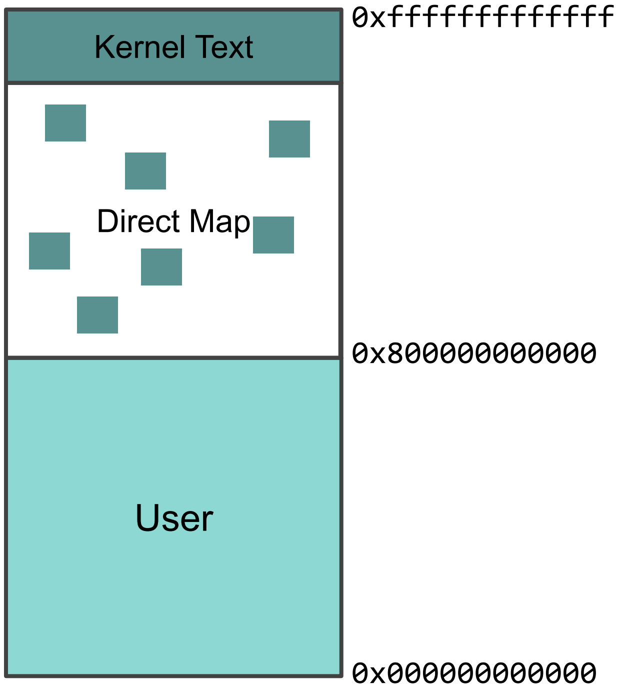
- **K-domain** ("kernel domain") has a page table with all physical memory



The Ward kernel is split in half

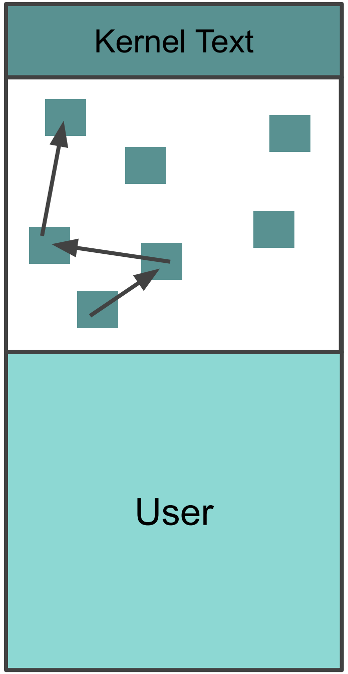
Ward extends Linux’s PTI:

- **K-domain** (“kernel domain”) has a page table with all physical memory
- **Q-domain** (“quasi-visible domain”) has a page table with user mappings, *and safe kernel mappings.*



Q-domain

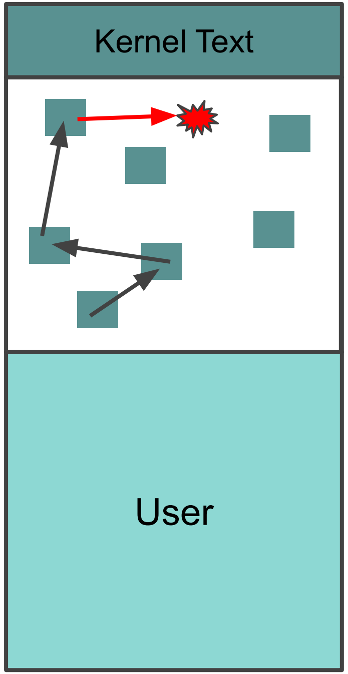
Syscalls start executing in the Q-domain



Q-domain

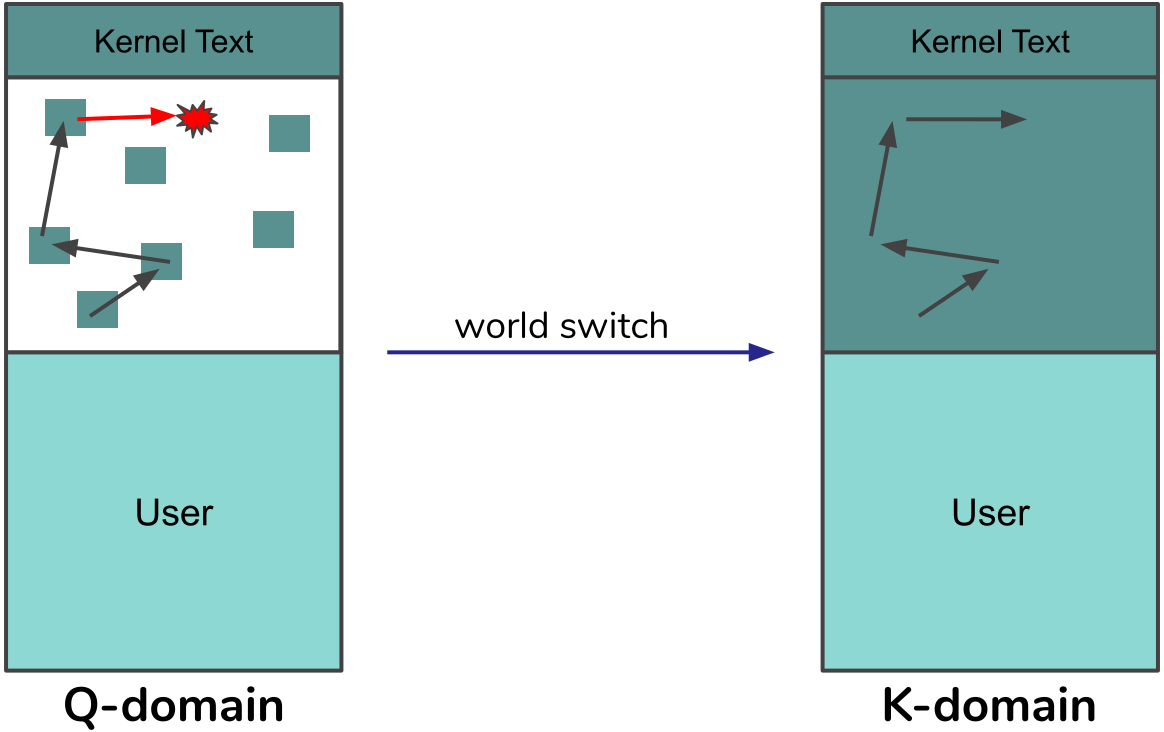
- Any syscall or trap handler that doesn’t access any secret data will run entirely in the Q-domain.
- When this happens, we are able to avoid many mitigations:
    - No need for page table swap
    - Don’t have to flush microarchitectural buffers
    - Retpolines are not required

…but sometimes we must enter the K-domain



Q-domain

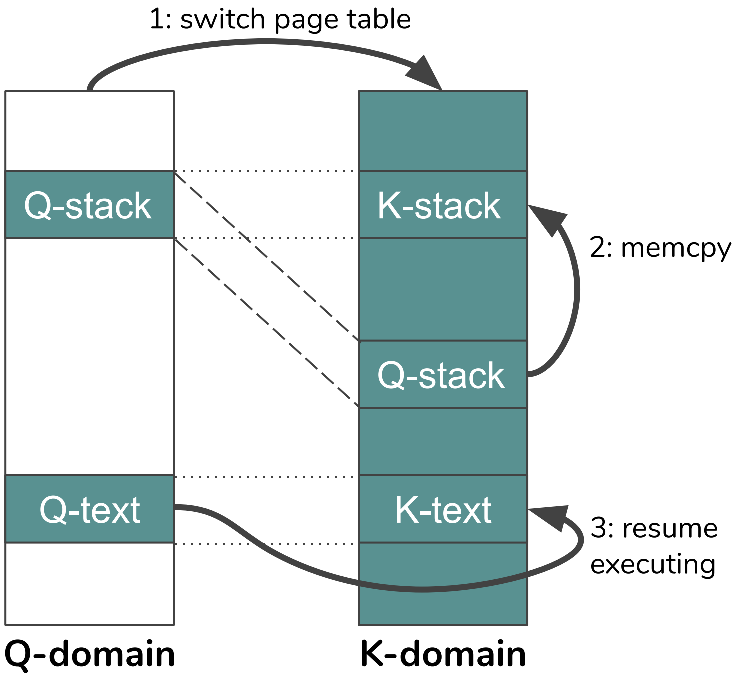
...but sometimes we must enter the K-domain



World switches use two stacks

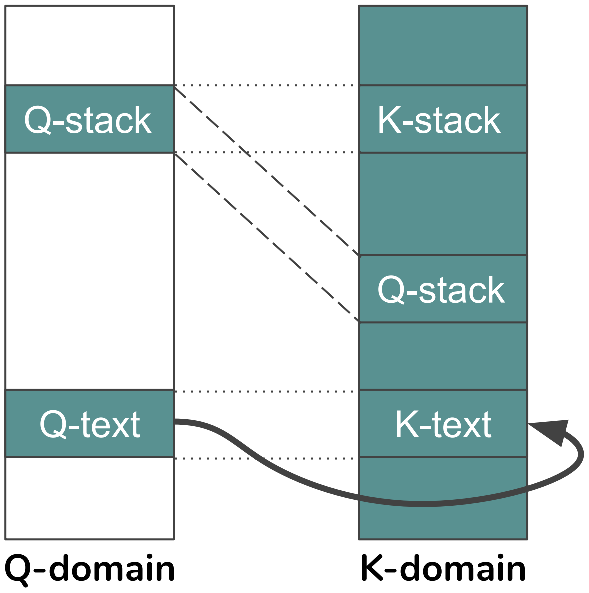
Steps in a world switch...

1. Switch to K-domain page table
2. Copy Q-stack contents to K-stack
3. Resume executing



Q and K Kernel

- Both code segments are compiled the same
  - Matching instruction addresses and stack layouts
- At runtime, Q-text has mitigations patched out
  - lfence
  - verw
  - retpoline



Redesigning the kernel to avoid switches

- Kernel data structures may mix secret and non-secret data

```
struct proc {
    proc* next;
    int pid;
    uint64_t saved_regs[16];
    ...
};
```

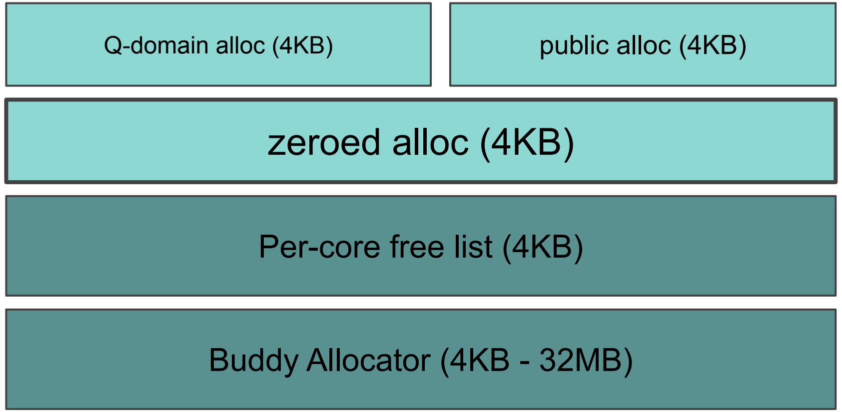
```
struct proc_public {
    proc_public* next;
    int pid;
    ...
};
struct proc_private {
    proc_public* pproc;
    uint64_t saved_regs[16];
    ...
};
```

Manipulating page tables while in the Q-domain

- The physical memory pages backing the page tables, are themselves in the Q-domain
- Powerful capability which enables Q-domain to…
  - Allocate anonymous memory
  - Create temporary mappings
  - Move kernel pages into/out of the Q-domain

Allocating memory without world switches

- Have a per-core list of zeroed memory pages ***mapped in the Q-domain***
  - Refreshed in batches
- Used for a variety of purposes:
  - Page tables
  - Q-domain kernel data structures
  - Lazy allocation of user memory



Implementation

- Based on sv6 research kernel
    - 34K lines of C++ code, plus libraries
- Supports all relevant mitigations from Linux
    - Focus on Skylake (2015-19) microarchitecture
- Binary compatible with a subset of Linux's syscall API
    - Can run unmodified binaries!

Results

Does Ward reduce overhead?

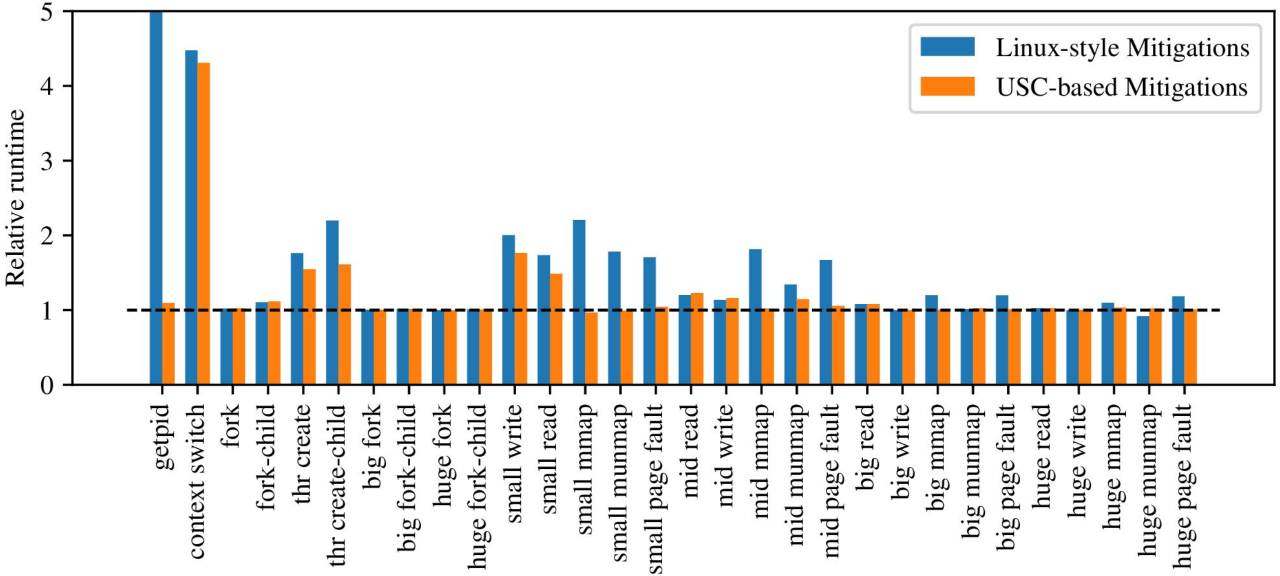
Ward configurations:

- **Linux-style**: Standard mitigations like the ones in Linux
- **USC-based:** Fast mitigations
- **Baseline:** All mitigations disabled

Workloads:

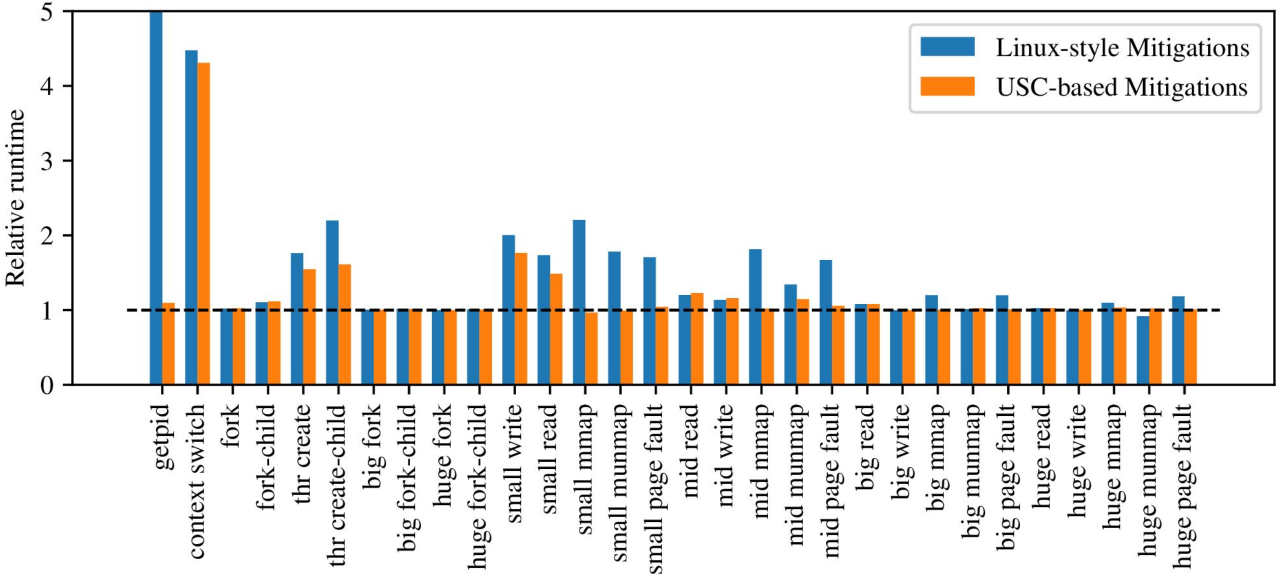
- LEBench
- git

Ward does better on LEBench



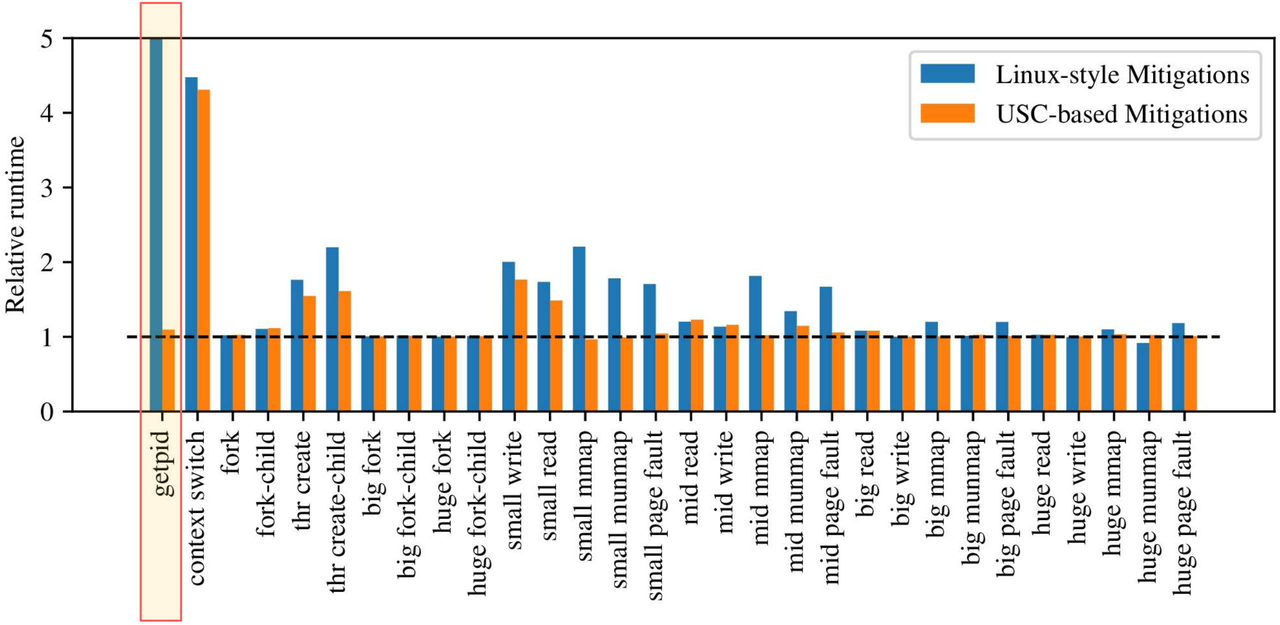
Ward does better on LEBench

Lower is better



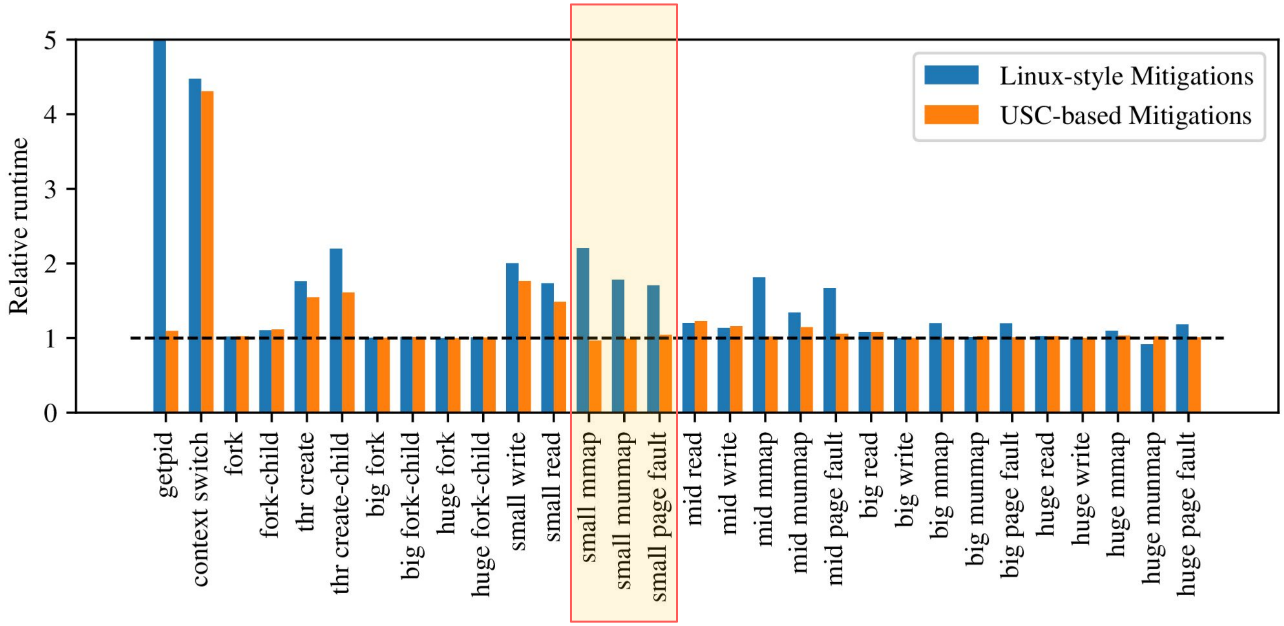
Ward does better on LEBench

Lower is better

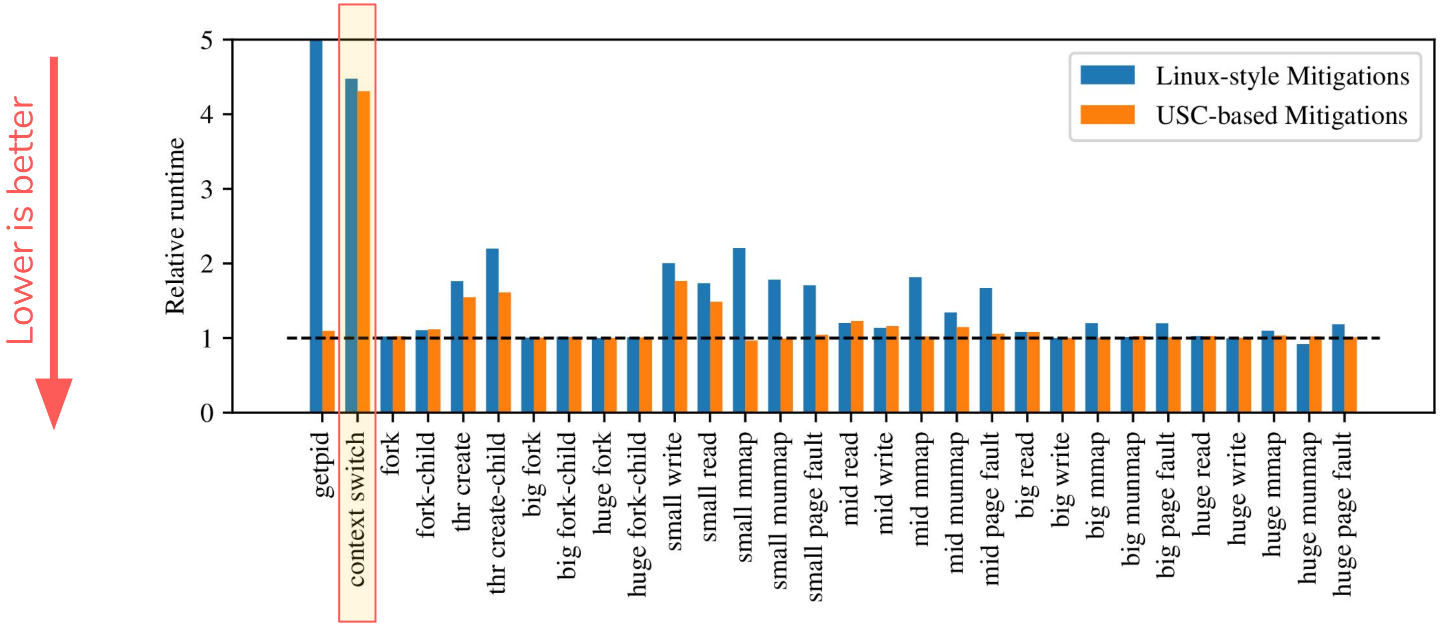


Ward does better on LEBench

Lower is better



Ward does better on LEBench



Git benchmark

- Ward also demonstrates application-level performance improvements
- Runtime for `git status` on a 100 MB repository:

Configuration	Overhead
Linux-style	24.6%
USC-based	11.2%

Related Work: Spectrum of defenses

- Pure software defenses like Linux’s PTI, retpoline, etc.
- Hardware-software co-designs like **ConTExT** [CoRR], and **SpecCFI** [SP ‘20]
- Hardware defenses: Intel/AMD designs, **Specshield** [PACT ‘19], **NDA** [MICRO ‘19], and **Speculative Taint Tracking** [MICRO ‘19]

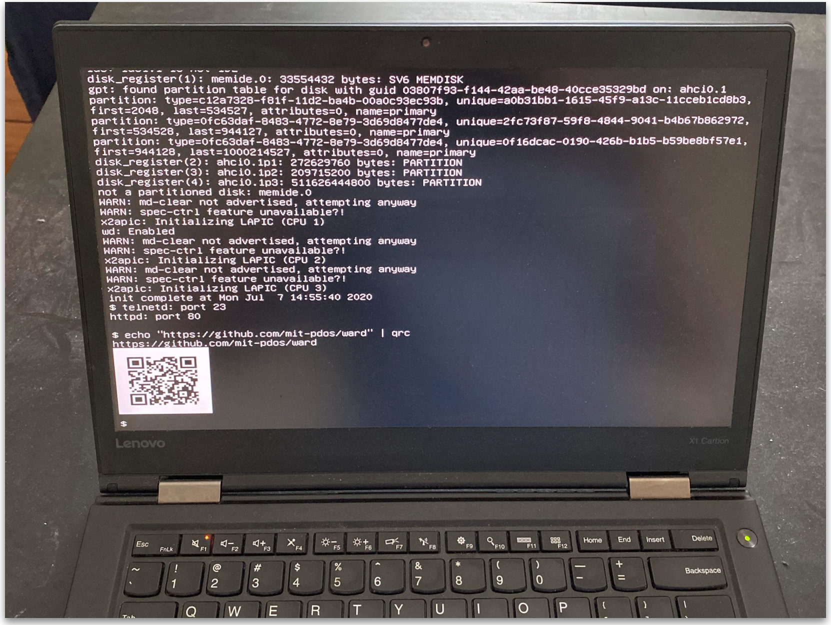
Open question: what is the best way to mitigate attacks?

- Intel Cascade Lake (2019) has hardware mitigations for many attacks
  - Eliminates need for software mitigations
  - Toggling mitigations is almost free, but...
- New processor up to **33% slower** executing LEBench syscalls
  - Compared to 2016 CPU model with same clock speed and core count
  - When mitigations disabled for both

Can hardware mitigations leverage the USC to get better performance?

Conclusion

- The Unmapped Speculation Contract defines a division of responsibility between hardware and software
- Using USC, Ward reduces the performance cost of mitigations in software



github.com/mit-pdos/ward

Contact: **behrensj@mit.edu**