# Address Space Isolation (ASI)

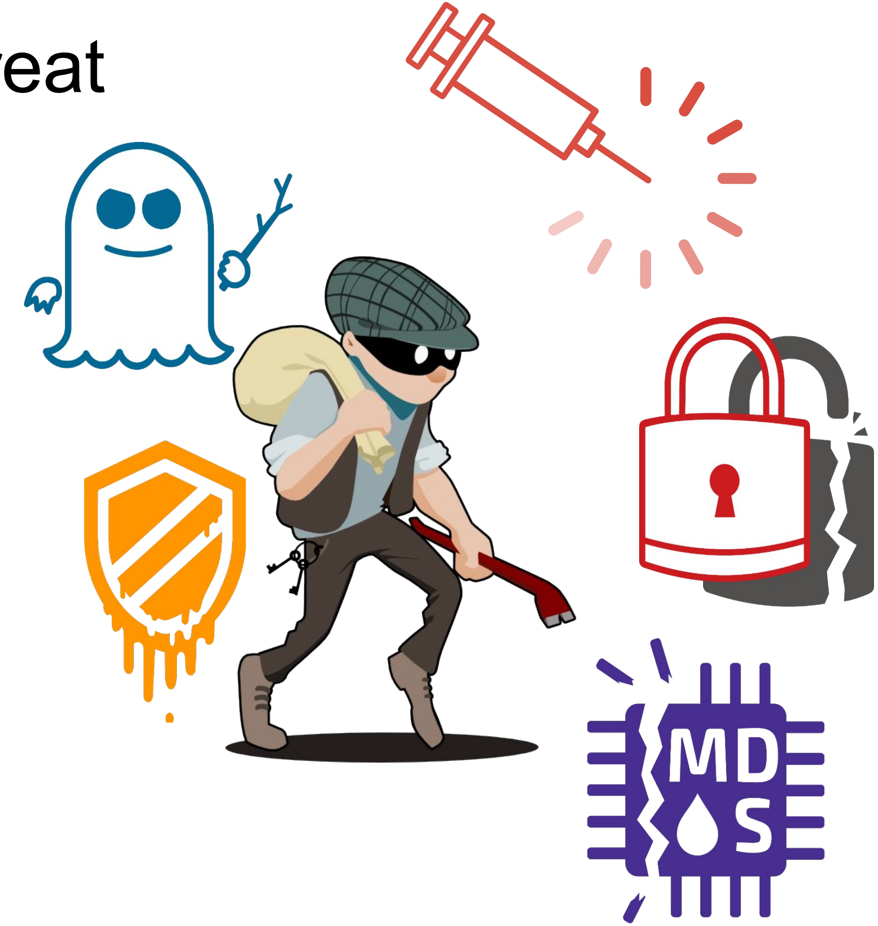
## Speculative execution protection

Google

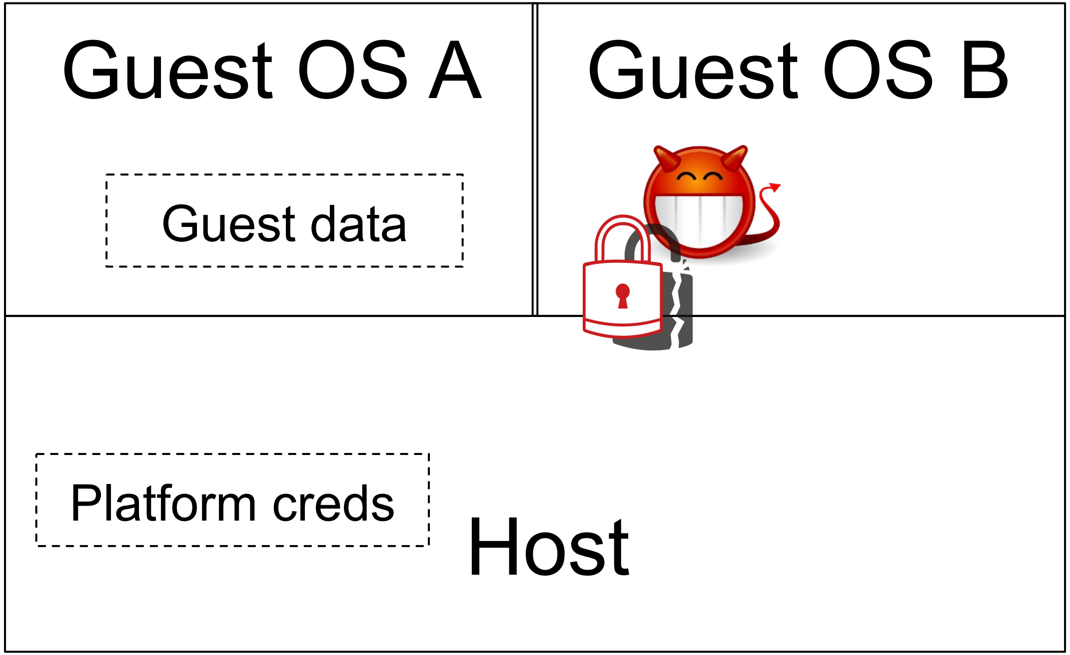
Ofir Weisse, Junaid Shahid, Oleg Rombakh, and Paul Turner

# The Speculative Attacks Threat

- These are µ-architectural attacks
- They break architectural boundaries
  - User/kernel boundary
  - Inter-process boundary
  - **VM/host boundary**
- They therefore compromise
  - Our customer’s data
  - Infrastructure (host) credentials
- Current mitigations are either
  - High overhead, or
  - Incomplete



# What Can be Stolen



# Roadmap

- The Speculative Attacks Threat
- **L1TF Refresher**
- Why Mitigation is Challenging
- Address Space Isolation (ASI)

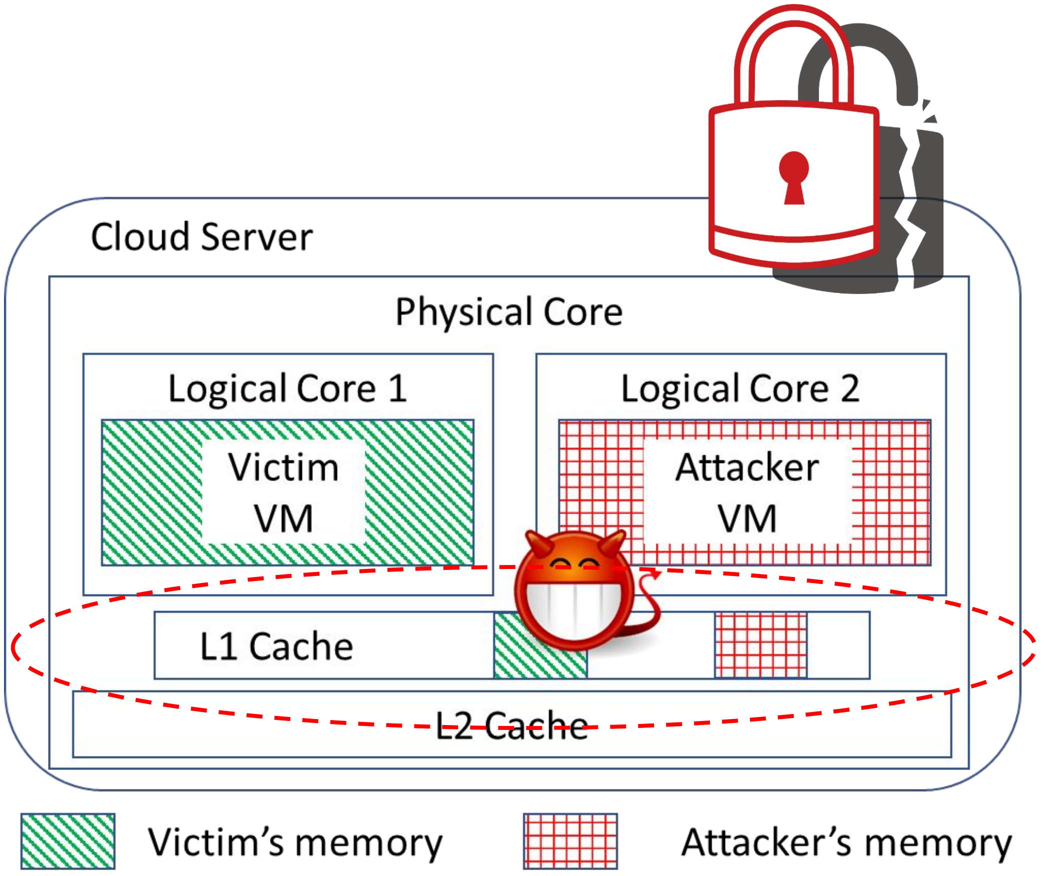
To learn more about speculative attacks:

foreshadowattack.eu

ofirweisse.com/MICRO2019_NDA.pdf

# L1TF in a Nutshell

- Shared μ-arch state can be stolen
  - L1TF - L1 cache
  - MDS - other μ-buffers
- The state can be left by previous context
- Or provoked by the attacker
  - Via calling an API

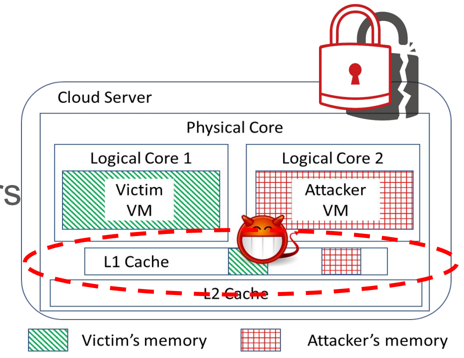


# Roadmap

- The Speculative Attacks Threat
- L1TF Refresher
- **Why Mitigation is Challenging**
- Address Space Isolation (ASI)
- Initial Results

# The Challenge: Mitigations are Hard

1. Stop speculation, e.g., with lfences everywhere
   - X Extremely slow
2. Stop side-channels - that's a cat and mouse came
   - X E.g., L1D-cache, L1I-cache, BTB, branch-direction-predictor, etc. etc.
3. Stop speculation after branches
   - X Slow
   - X Error-prone
4. Scrub/flush secrets from state (L1 cache and other buffers)
   - X The attacker can trigger execution bringing data to these buffers
   - X The execution above can even be speculative!
   - X Async execution (interrupts), Hardware prefetch are additional vectors
5. HyperThreading complicates defenses event more!
   - X A sibling thread can snoop shared resources



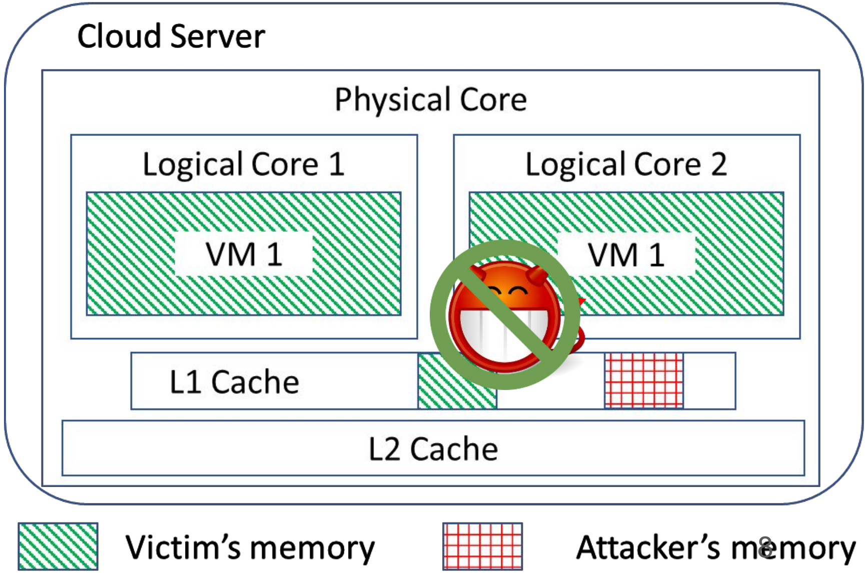
# What mitigations are applied today? (1)

Disabling HyperThreading infeasible (cost, performance, etc)

So what can we do?

- Secure core scheduling
    - Never run two VMs on the same physical core



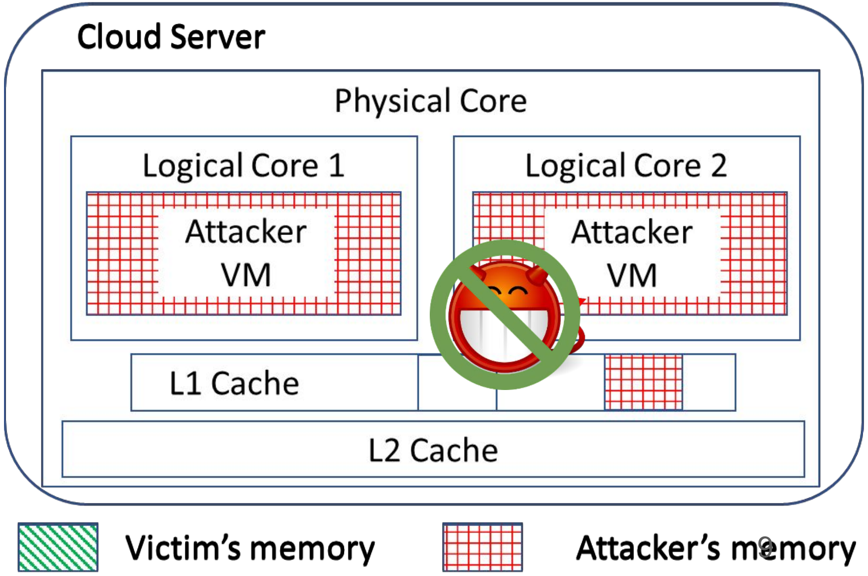


# What mitigations are applied today? (2)

Disabling HyperThreading is costly for performance/capacity

So what can we do?

- Secure core scheduling
- Flush L1 cache on VMENTER
    - Expensive

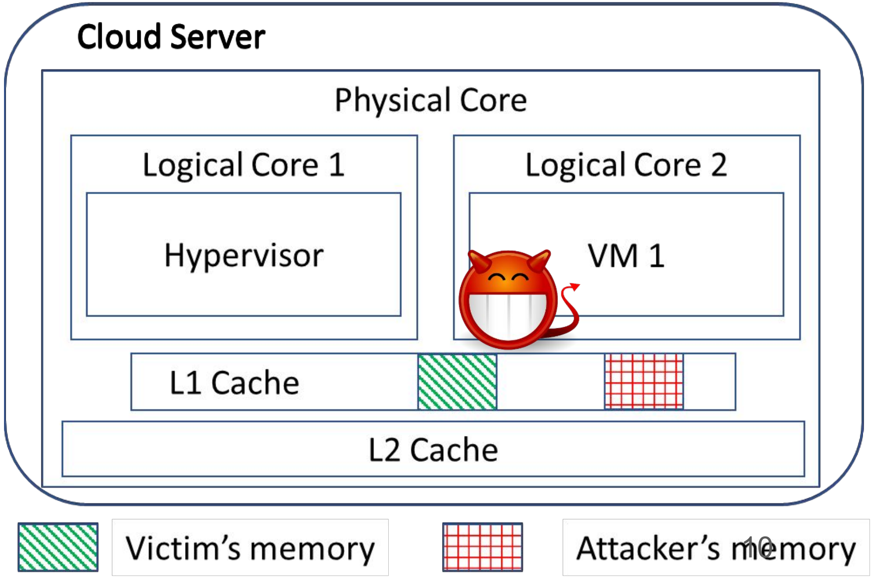


# What mitigations are applied today? (3)

Disabling HyperThreading is devastating for performance

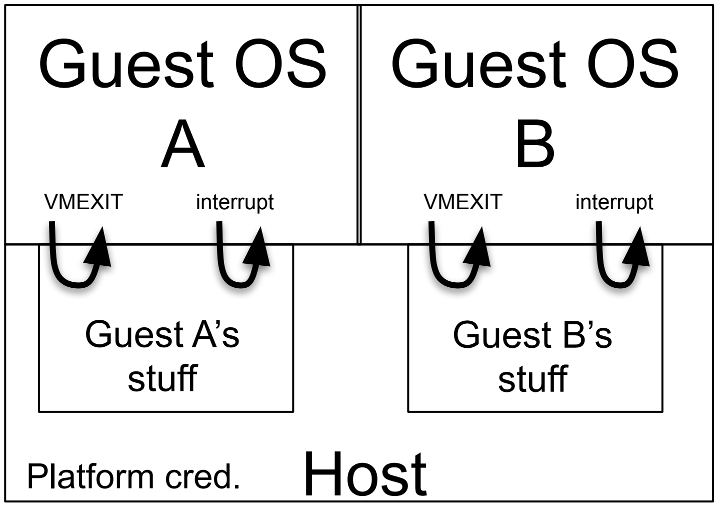
So what can we do?

- Secure core scheduling
- Flush L1 cache on VMENTER
- On VMEXIT to hypervisor – make sure other sibling core is stunned (not running)
    - Very expensive

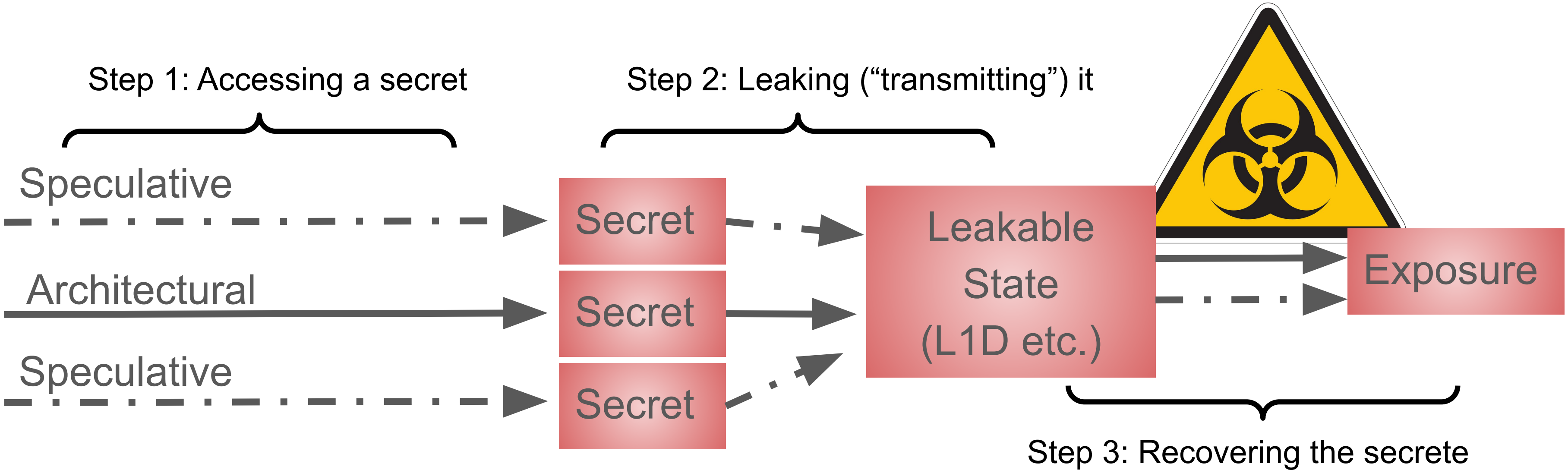


# What attack surface is open w/o constant flushes?

- On VMEXIT, interrupt handling may bring into cache/uarch-buffers data that
  - Belongs to other guests or
  - Is a platform secret
- That data can later be stolen via, e.g., L1TF
  - By the VM running after VMENTER
  - By sibling core during hypervisor execution



# Rethinking Mitigation - Understanding the Leak




**Status quo**: u-arch buffers are always (potentially) contaminated with secrets

**Sad conclusion:** Need to either a) stop speculation or b) continuously scrub state

For more details: ofirweisse.com/MICRO2019_NDA.pdf

# Rethinking Mitigation - Understanding the Leak

Step 1: Accessing a

Speculative

Architectural

Speculative

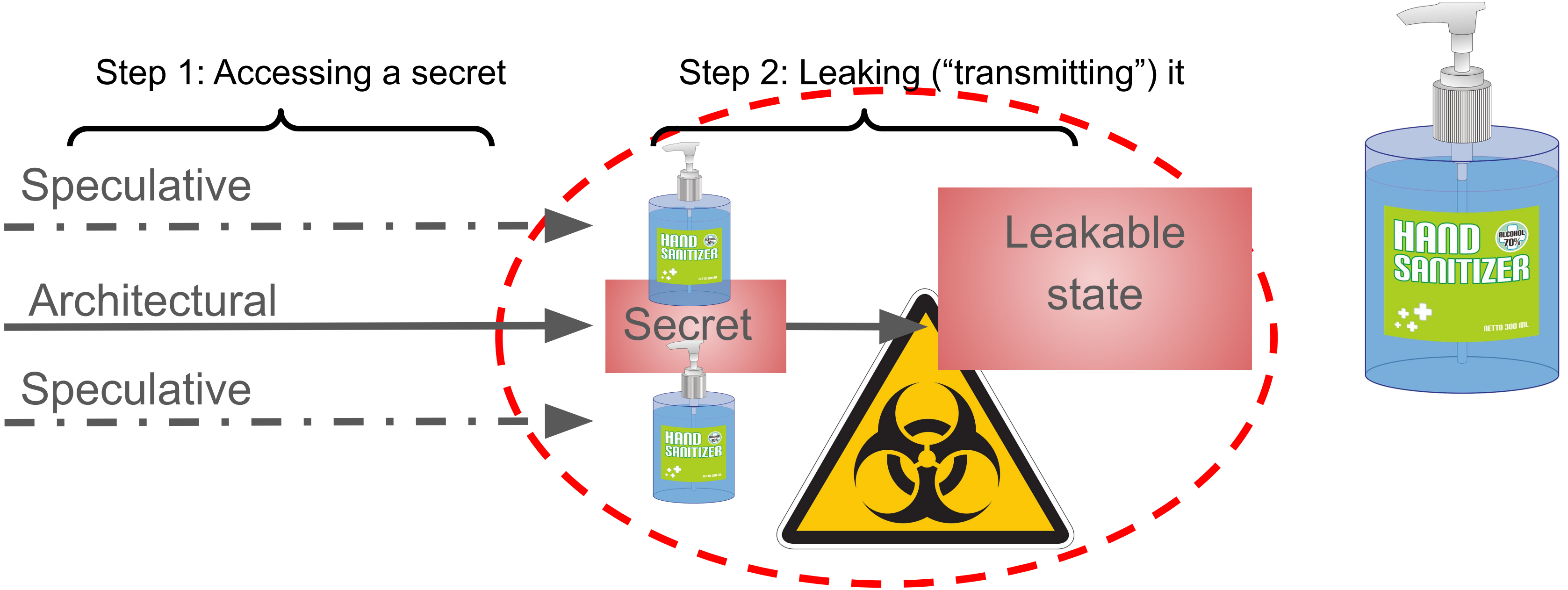

Exposure

overing the secrete

**Status quo**: u-arch buffers are always (potentially) contaminated with secrets

**Sad conclusion:** Need to either a) stop speculation or b) continuously scrub state

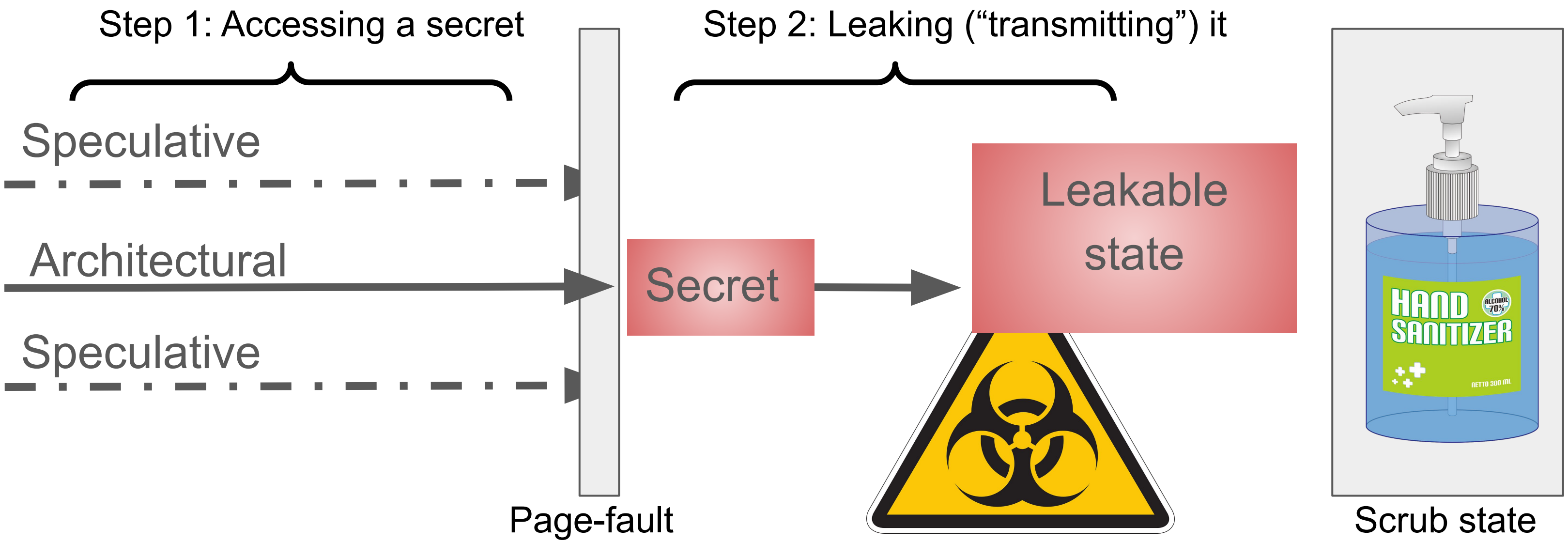
# Rethinking Mitigation - Limiting Exposure




**We want a way to circumscribe access to secrets and leakable state.**

**We then apply protection only when secrets are "in flight"**

# Idea: #PF as a fork between speculative & non-spec exec




**We want a way to circumscribe access to secrets and leakable state.**

**We then apply protection only when secrets are "in flight"**

# Trivial example: Spectre V1 (bounds check bypass)

```
int foo(u8 *arr, int size, int index) {
        if (index < size) {
                // Should lfence
                return global_array[ arr[index]* 64 ];
        }
        // ...
}
```

If index is out of bounds, “`arr`” might speculatively still be accessed.

Google

# Trivial example: Spectre V1 (bounds check bypass)

```
int foo(u8 *arr, int size, int index) {
        if (index < size) {
                // Should lfence
                return global_array[ arr[index]* 64 ];
        }
        // ...
}
```

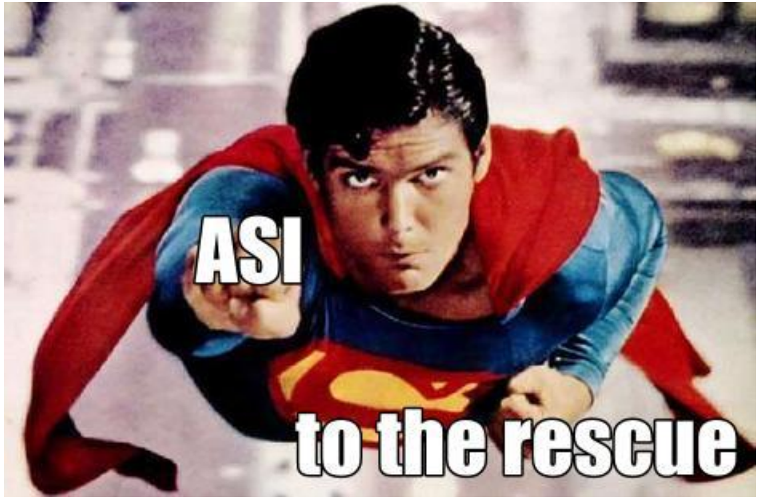
If index is out of bounds, “`arr`” might speculatively still be accessed.

If `&arr[index]` is not mapped in the page-table → page-fault

**Question:** When do we scrub clean??

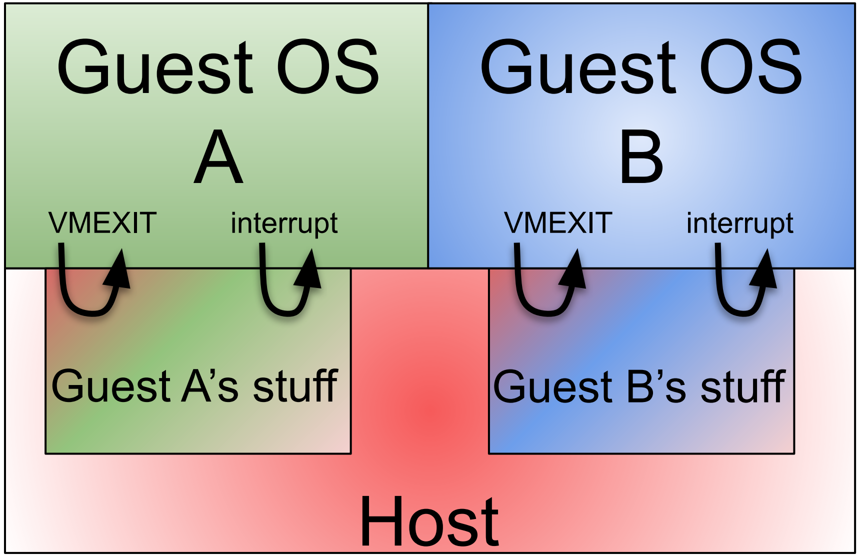
# Roadmap

- The Speculative Attacks Threat
- L1TF Refresher
- Why Mitigation is Challenging
- **Address Space Isolation (ASI)**
- Initial Results



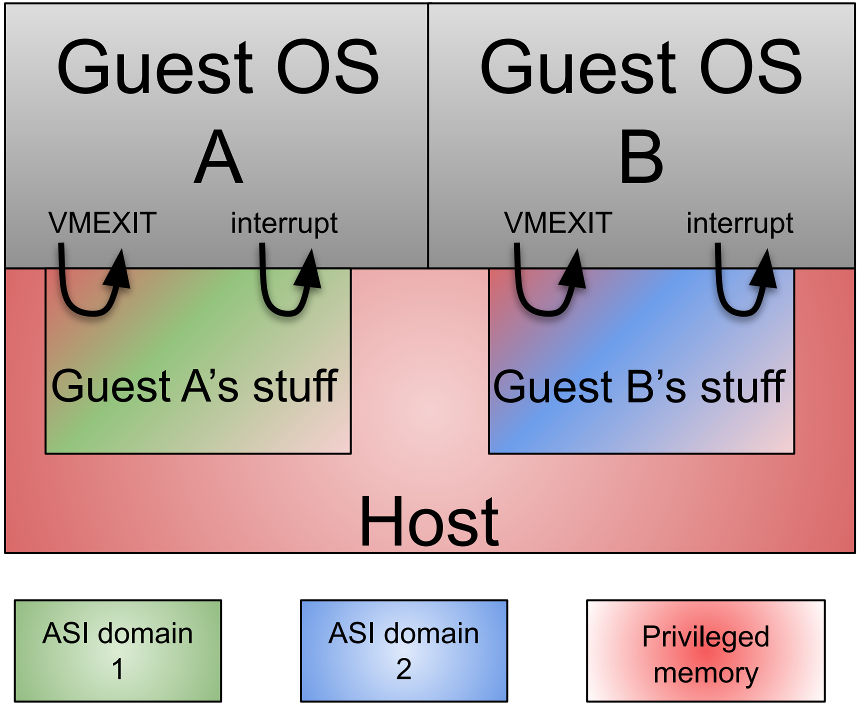
# Address Space Isolation - Premise

- On most VMEXIT's, the hypervisor only touches
  - Current guest stuff
  - Non sensitive data at the host



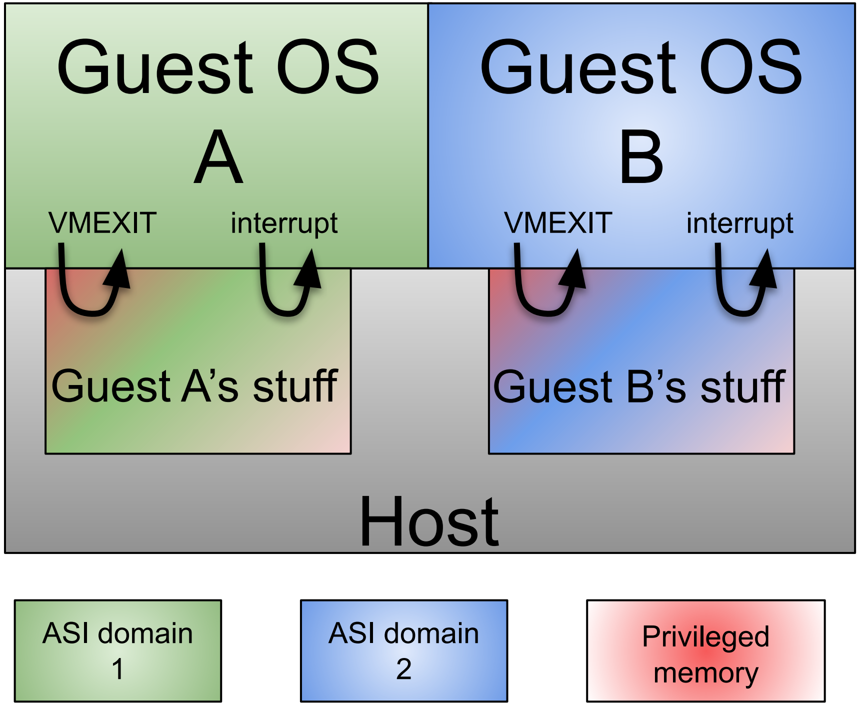
# Address Space Isolation - Basic Idea

- Split kernel memory to privileged and unprivileged-domains
- Each domain has a seperate page-table
- Touching data out of a domain results in a page-fault - <u>cannot be speculative</u>
- At first, only include kernel addresses



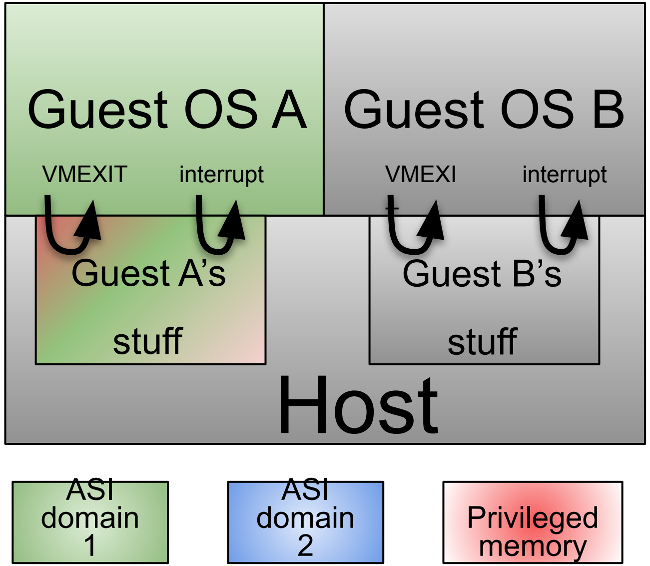
# Address Space Isolation - Basic Idea

- Split kernel memory to privileged and unprivileged-domains
- Each domain has a seperate page-table
- Touching data out of a domain results in a page-fault - <u>cannot be speculative</u>
- At first, only include kernel addresses
- ASI can be extended to include userspace memory



# ASI Lifecycle

```
//IOCTL KVM_RUN
for (;;) { // in vcpu_run()
    // call vmx_vcpu_run()
      asi_enter(); // Switch CR3 to unprivileged map
        // VMENTER
        // VMEXIT by the platform
        // Try to handle exit, may touch
           privileged data, which will cause
           A page fault --> asi_exit()
}
```

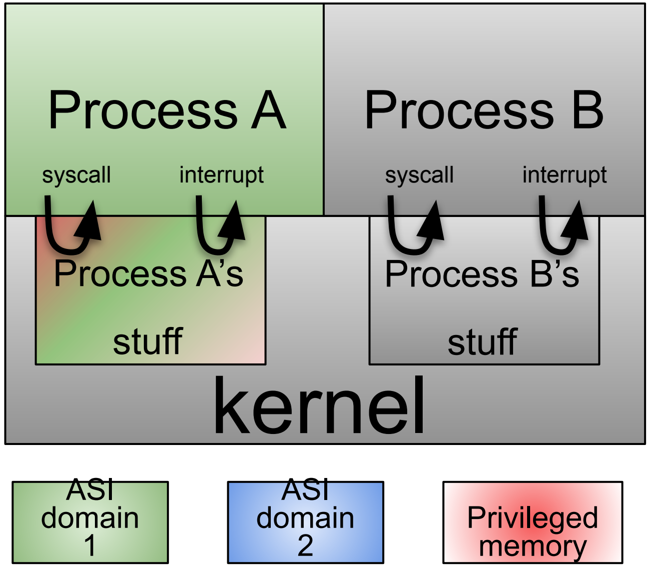


# Challenges

1. What data is OK to place within the unprivileged map?
   a. Anything that belongs to the guest anyhow
   b. Kernel maintenance structures which are used frequently and are not sensitive
2. How to handle PF/asi_exits within interrupts, nmi's, etc.?
   a. Must automatically re-asi_enter() when done


# ASI as a replacement for KPTI

- KPTI switches page-tables upon entry/exit to the kernel
- ASI (sometimes) switches page-tables upon entry/exit from a VM
- The same approach can, therefore, replace KPTI
  - To minimize page-table switches



# Initial Results - Redis YCSB

Ratio of ASI-exits/VM-exits

```
KVM/VCPU 0xffffc9001da89000/0: Time 309.05 seconds, asi/vm exits = 46160 / 4506402 = 1.02 %
KVM/VCPU 0xffffc9001da89000/1: Time 291.67 seconds, asi/vm exits = 400531 / 1267665 = 31.60 %
KVM/VCPU 0xffffc9001da89000/2: Time 291.67 seconds, asi/vm exits = 413946 / 2323131 = 17.82 %
KVM/VCPU 0xffffc9001da89000/3: Time 291.63 seconds, asi/vm exits = 499027 / 1045507 = 47.73 %
KVM/VCPU 0xffffc9001da89000/4: Time 291.69 seconds, asi/vm exits = 482687 / 2013058 = 23.98 %
KVM/VCPU 0xffffc9001da89000/5: Time 291.62 seconds, asi/vm exits = 500809 / 2170556 = 23.07 %
KVM/VCPU 0xffffc9001da89000/6: Time 291.68 seconds, asi/vm exits = 478710 / 1775451 = 26.96 %
KVM/VCPU 0xffffc9001da89000/7: Time 291.61 seconds, asi/vm exits = 482880 / 2059408 = 23.45 %
total_asi_exits = 3304750
KVM/VCPU 0xffffc90039f35000/0: Time 225.19 seconds, asi/vm exits = 489981 / 6257089 = 7.83 %
KVM/VCPU 0xffffc90039f35000/1: Time 225.00 seconds, asi/vm exits = 493745 / 1009584 = 48.91 %
KVM/VCPU 0xffffc90039f35000/2: Time 225.00 seconds, asi/vm exits = 756191 / 2425297 = 31.18 %
KVM/VCPU 0xffffc90039f35000/3: Time 225.00 seconds, asi/vm exits = 521712 / 1051189 = 49.63 %
KVM/VCPU 0xffffc90039f35000/4: Time 224.91 seconds, asi/vm exits = 23353 / 73144 = 31.93 %
KVM/VCPU 0xffffc90039f35000/5: Time 224.93 seconds, asi/vm exits = 19609 / 60075 = 32.64 %
KVM/VCPU 0xffffc90039f35000/6: Time 224.93 seconds, asi/vm exits = 26320 / 81998 = 32.10 %
KVM/VCPU 0xffffc90039f35000/7: Time 224.99 seconds, asi/vm exits = 22509 / 85046 = 26.47 %
total_asi_exits = 2353420
```

# Initial Results - Redis

## Exit details

| RIP | data_addr | accessor | est_alloc_site | count | CDF |
|---|---|---|---|---|---|
| 0xffffffff811cecd3 | 0xffff88563e42c938 | el/sched/exclusive.c:7283 | PO: ./kernel/fork.c:1636 | 276673 | 1.000000 |
| 0xffffffff811cecd3 | 0xffff88554bc49938 | el/sched/exclusive.c:7283 | PO: ./kernel/events/core.c:10843 | 233775 | 0.887946 |
| 0xffffffff811c79b1 | 0xffffe8a0612b0070 | rnel/sched/cpuacct.c:1284 | PO: ./mm/percpu-vm.c:284 | 151020 | 0.793267 |
| 0xffffffff811da155 | 0xffff885585e57c58 | el/sched/exclusive.c:7664 | ./net/core/skbuff.c:213 | 54685 | 0.732103 |
| 0xffffffff811c79b1 | 0xffffe8a0612f0070 | rnel/sched/cpuacct.c:1284 | PO: ./mm/percpu-vm.c:284 | 45065 | 0.709956 |
| 0xffffffff81192686 | 0xffff88554bc49938 | ernel/sched/cputime.c:154 | PO: ./kernel/events/core.c:10843 | 37279 | 0.691704 |
| 0xffffffff811c79b1 | 0xffffe8a05ccf6cf0 | rnel/sched/cpuacct.c:1284 | PO: ./mm/percpu-vm.c:284 | 32923 | 0.676606 |
| 0xffffffff81192686 | 0xffff88563e42c938 | ernel/sched/cputime.c:154 | PO: ./kernel/fork.c:1636 | 31714 | 0.663272 |
| 0xffffffff811da155 | 0xffff8855596c4c58 | el/sched/exclusive.c:7664 | ./net/core/skbuff.c:213 | 30228 | 0.650428 |
| 0xffffffff811ced4d | 0xffffffff83a2b930 | el/sched/exclusive.c:7315 | config_consume_rt_capacity | 29209 | 0.638185 |
| 0xffffffff811c79a2 | 0xffff885551c508d8 | rnel/sched/cpuacct.c:1284 | ./net/core/skbuff.c:213 | 24593 | 0.626356 |
| 0xffffffff815f0880 | 0xffff8854864b0380 | ./lib/llist.c:97 | ./fs/eventfd.c:658 | 24471 | 0.616395 |
| 0xffffffff811c79b1 | 0xffffe8a060a6dfe0 | rnel/sched/cpuacct.c:1284 | PO: ./mm/percpu-vm.c:284 | 21122 | 0.606485 |
| 0xffffffff811c79b1 | 0xffffe8a060aece90 | rnel/sched/cpuacct.c:1284 | PO: ./mm/percpu-vm.c:284 | 20673 | 0.597930 |

# Initial Results - Redis

## Exit details

| RIP | data_addr | accessor | est_alloc_site | count | CDF |
|---|---|---|---|---|---|
| 0xffffffff811cecd3 | 0xffff88563e42c938 | el/sched/exclusive.c:7283 | PO: ./kernel/fork.c:1636 | 276673 | 1.000000 |
| 0xffffffff811cecd3 | 0xffff88554bc49938 | el/sched/exclusive.c:7283 | PO: ./kernel/events/core.c:10843 | 233775 | 0.887946 |
| 0xffffffff811c79b1 | 0xffffe8a0612b0070 | rnel/sched/cpuacct.c:1284 | PO: ./mm/percpu-vm.c:284 | 151020 | 0.793267 |
| 0xffffffff811da155 | 0xffff885585e57c58 | el/sched/exclusive.c:7664 | ./net/core/skbuff.c:213 | 54685 | 0.732103 |
| 0xffffffff811c79b1 | 0xffffe8a0612f0070 | rnel/sched/cpuacct.c:1284 | PO: ./mm/percpu-vm.c:284 | 45065 | 0.709956 |
| 0xffffffff81192686 | 0xffff88554bc49938 | ernel/sched/cputime.c:154 | PO: ./kernel/events/core.c:10843 | 37279 | 0.691704 |
| 0xffffffff811c79b1 | 0xffffe8a05ccf6cf0 | rnel/sched/cpuacct.c:1284 | PO: ./mm/percpu-vm.c:284 | 32923 | 0.676606 |
| 0xffffffff81192686 | 0xffff88563e42c938 | ernel/sched/cputime.c:154 | PO: ./kernel/fork.c:1636 | 31714 | 0.663272 |
| 0xffffffff811da155 | 0xffff8855596c4c58 | el/sched/exclusive.c:7664 | ./net/core/skbuff.c:213 | 30228 | 0.650428 |
| 0xffffffff811ced4d | 0xffffffff83a2b930 | el/sched/exclusive.c:7315 | config_consume_rt_capacity | 29209 | 0.638185 |
| 0xffffffff811c79a2 | 0xffff885551c508d8 | rnel/sched/cpuacct.c:1284 | ./net/core/skbuff.c:213 | 24593 | 0.626356 |
| 0xffffffff815f0880 | 0xffff8854864b0380 | ./lib/llist.c:97 | ./fs/eventfd.c:658 | 24471 | 0.616395 |
| 0xffffffff811c79b1 | 0xffffe8a060a6dfe0 | rnel/sched/cpuacct.c:1284 | PO: ./mm/percpu-vm.c:284 | 21122 | 0.606485 |
| 0xffffffff811c79b1 | 0xffffe8a060aece90 | rnel/sched/cpuacct.c:1284 | PO: ./mm/percpu-vm.c:284 | 20673 | 0.597930 |

```
7278        curr->se.exec_start = now;
7279        schedstat_set(curr->se.statistics.exec_max,
7280                      max(curr->se.statistics.exec_max, delta_exec));
7281
7282        curr->se.sum_exec_runtime += delta_exec;
7283        account_group_exec_runtime(curr, delta_exec);
```

# Initial Results - Redis

## Exit details

| RIP | data_addr | accessor | est_alloc_site | count | CDF |
|---|---|---|---|---|---|
| 0xffffffff811cecd3 | 0xffff88563e42c938 | el/sched/exclusive.c:7283 | PO: ./kernel/fork.c:1636 | 276673 | 1.000000 |
| 0xffffffff811cecd3 | 0xffff88554bc49938 | el/sched/exclusive.c:7283 | PO: ./kernel/events/core.c:10843 | 233775 | 0.887946 |
| 0xffffffff811c79b1 | 0xffffe8a0612b0070 | rnel/sched/cpuacct.c:1284 | PO: ./mm/percpu-vm.c:284 | 151020 | 0.793267 |
| 0xffffffff811da155 | 0xffff885585e57c58 | el/sched/exclusive.c:7664 | ./net/core/skbuff.c:213 | 54685 | 0.732103 |
| 0xffffffff811c79b1 | 0xffffe8a0612f0070 | rnel/sched/cpuacct.c:1284 | PO: ./mm/percpu-vm.c:284 | 45065 | 0.709956 |
| 0xffffffff81192686 | 0xffff88554bc49938 | ernel/sched/cputime.c:154 | PO: ./kernel/events/core.c:10843 | 37279 | 0.691704 |
| 0xffffffff811c79b1 | 0xffffe8a05ccf6cf0 | rnel/sched/cpuacct.c:1284 | PO: ./mm/percpu-vm.c:284 | 32923 | 0.676606 |
| 0xffffffff81192686 | 0xffff88563e42c938 | ernel/sched/cputime.c:154 | PO: ./kernel/fork.c:1636 | 31714 | 0.663272 |
| 0xffffffff811da155 | 0xffff8855596c4c58 | el/sched/exclusive.c:7664 | ./net/core/skbuff.c:213 | 30228 | 0.650428 |
| 0xffffffff811ced4d | 0xffffffff83a2b930 | el/sched/exclusive.c:7315 | config_consume_rt_capacity | 29209 | 0.638185 |
| 0xffffffff811c79a2 | 0xffff885551c508d8 | rnel/sched/cpuacct.c:1284 | ./net/core/skbuff.c:213 | 24593 | 0.626356 |
| 0xffffffff815f0880 | 0xffff8854864b0380 | ./lib/llist.c:97 | ./fs/eventfd.c:658 | 24471 | 0.616395 |
| 0xf | | | | 21122 | 0.606485 |
| 0xf | | | | 20673 | 0.597930 |

```
1628 static int copy_signal(unsigned long clone_flags, struct task_struct *tsk)
1629 {
1630         struct signal_struct *sig;
1631
1632         if (clone_flags & CLONE_THREAD)
1633                 return 0;
1634
1635 #ifdef CONFIG_ADDRESS_SPACE_ISOLATION
1636         sig = kzalloc(sizeof(struct signal_struct),
1637                         GFP_KERNEL | GFP_NONSENSITIVE);
 NORMAL  PASTE  kernel/fork.c                                   55%
```

# Initial Results - Redis

## Exit details by allocation site

| variable | count | CDF |
|---|---|---|
| PO: ./mm/percpu-vm.c:284 | 760078 | 1.000000 |
| PO: ./kernel/fork.c:1636 | 319451 | 0.692166 |
| PO: ./kernel/events/core.c:10843 | 293764 | 0.562787 |
| ./net/core/skbuff.c:213 | 208683 | 0.443812 |
| PO: ./kernel/fork.c:249 | 193298 | 0.359294 |
| PO: ./kernel/sched/topology.c:1766 | 157080 | 0.281008 |
| ./kernel/fork.c:1860 | 63355 | 0.217390 |

| RIP | data_addr | accessor | est_alloc_site | count |
|---|---|---|---|---|
| 0xffffffff811c79b1 | 0xffffe8a0612b0070 | rnel/sched/cpuacct.c:1284 | PO: ./mm/percpu-vm.c:284 | 151020 |
| 0xffffffff811c79b1 | 0xffffe8a0612f0070 | rnel/sched/cpuacct.c:1284 | PO: ./mm/percpu-vm.c:284 | 45065 |
| 0xffffffff811c79b1 | 0xffffe8a05ccf6cf0 | rnel/sched/cpuacct.c:1284 | PO: ./mm/percpu-vm.c:284 | 32923 |
| 0xffffffff811c79b1 | 0xffffe8a060a6dfe0 | rnel/sched/cpuacct.c:1284 | PO: ./mm/percpu-vm.c:284 | 21122 |
| 0xffffffff811c79b1 | 0xffffe8a060aece90 | rnel/sched/cpuacct.c:1284 | PO: ./mm/percpu-vm.c:284 | 20673 |
| 0xffffffff811c79b1 | 0xffffe8a05ccb6cf0 | rnel/sched/cpuacct.c:1284 | PO: ./mm/percpu-vm.c:284 | 20118 |
| 0xffffffff811c79b1 | 0xffffe8a05cc36cf0 | rnel/sched/cpuacct.c:1284 | PO: ./mm/percpu-vm.c:284 | 19819 |
| 0xffffffff811c79b1 | 0xffffe8a060ab0070 | rnel/sched/cpuacct.c:1284 | PO: ./mm/percpu-vm.c:284 | 14848 |
| 0xffffffff8120541c | 0xffffe8a05b682f40 | kernel/rcu/srcutree.c:418 | PO: ./mm/percpu-vm.c:284 | 14166 |
| 0xffffffff811c79b1 | 0xffffe8a05cc76cf0 | rnel/sched/cpuacct.c:1284 | PO: ./mm/percpu-vm.c:284 | 13879 |
| 0xffffffff811c79b1 | 0xffffe8a0612adfe0 | rnel/sched/cpuacct.c:1284 | PO: ./mm/percpu-vm.c:284 | 13765 |
| 0xffffffff811c79b1 | 0xffffe8a060a2dfe0 | rnel/sched/cpuacct.c:1284 | PO: ./mm/percpu-vm.c:284 | 12276 |

# Summary - efficiently defeating speculative attacks

1. ASI redefines access-control based on the data
   a. Namely, sensitive vs. non-sensitive **data**
   b. Instead of based on control-flow: **userspace vs. kernel**
2. A allow-list approach is more sustainable than block-list
3. Apply expensive (e.g., L1D flush, stunning) mitigations only when necessary
   a. Yields a complete **and** efficient solution
4. Can extend KPTI model and even improve performance
5. We want to integrate with concurrent efforts!