Google

Revisiting Address Space Isolation

Google, LPC 2022

Ofir Weisse, Junaid Shahid

Sep 13, 2022

The Speculative Attacks Threat

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

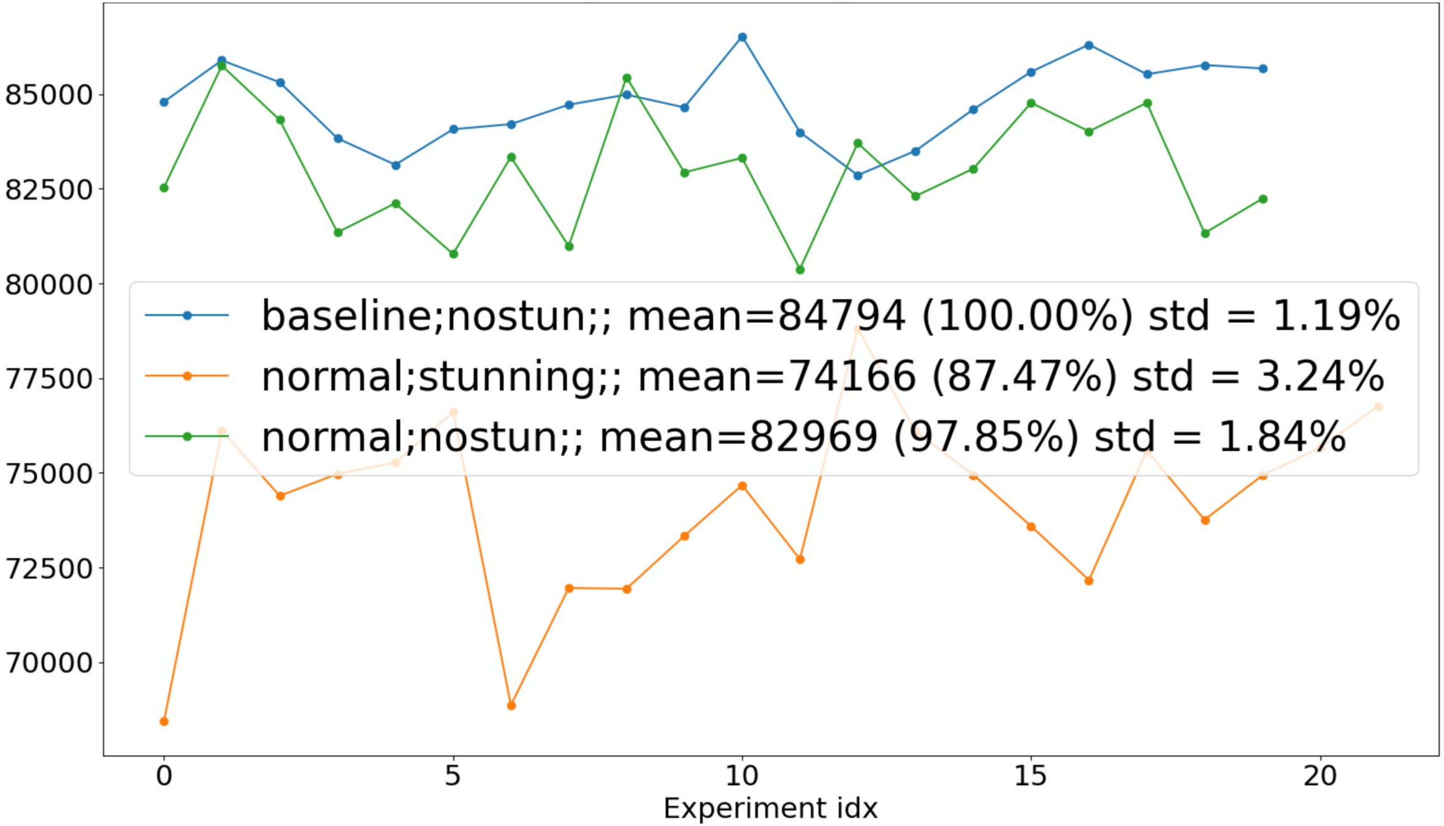
What happened since last time we presented ASI?

- New vulnerabilities discovered
- Most recent, most (in)famous - Retbleed
- Every vuln is a fire drill
  - 10s of engineers working on a fix
  - Months of preparation
- Performance degradation - 15-40% !!!!
  - E.g. phoronix.com/review/retbleed-benchmark
- Code investment, e.g.:
  - 52 files changed, 1634 insertions(+), 214 deletions(-)

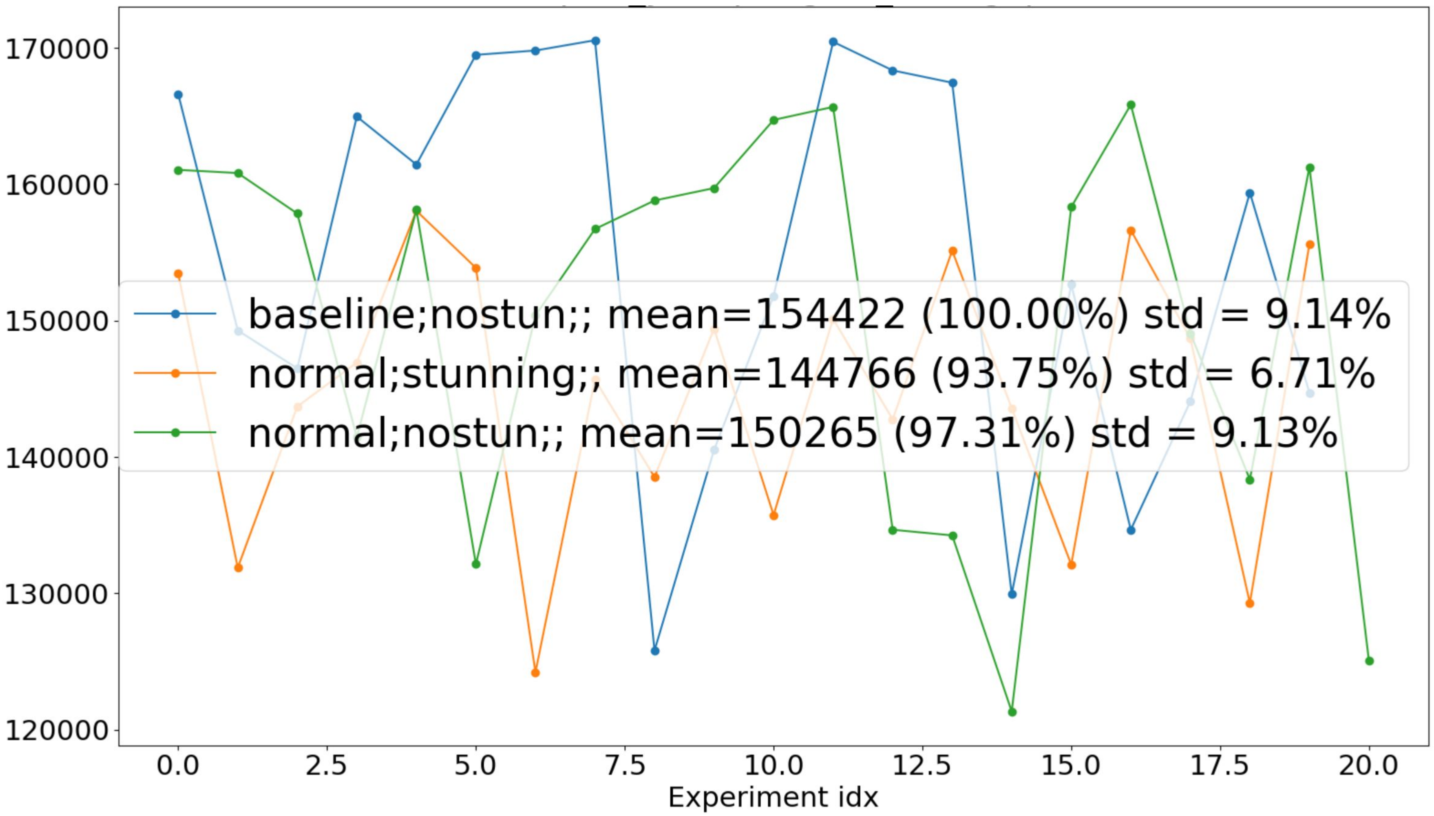
Should we rethink ASI?

- In current world, new attack means
  - Months of (urgent) work
  - Many engineers
  - Scattered around the kernel
- In ASI world, a new attack mean
  - A few more lines in `asi_enter()/asi_exit()`
  - Probably a single engineer to write
- Performance estimation: 2-14%
- Can be improved by increasing the allow-list

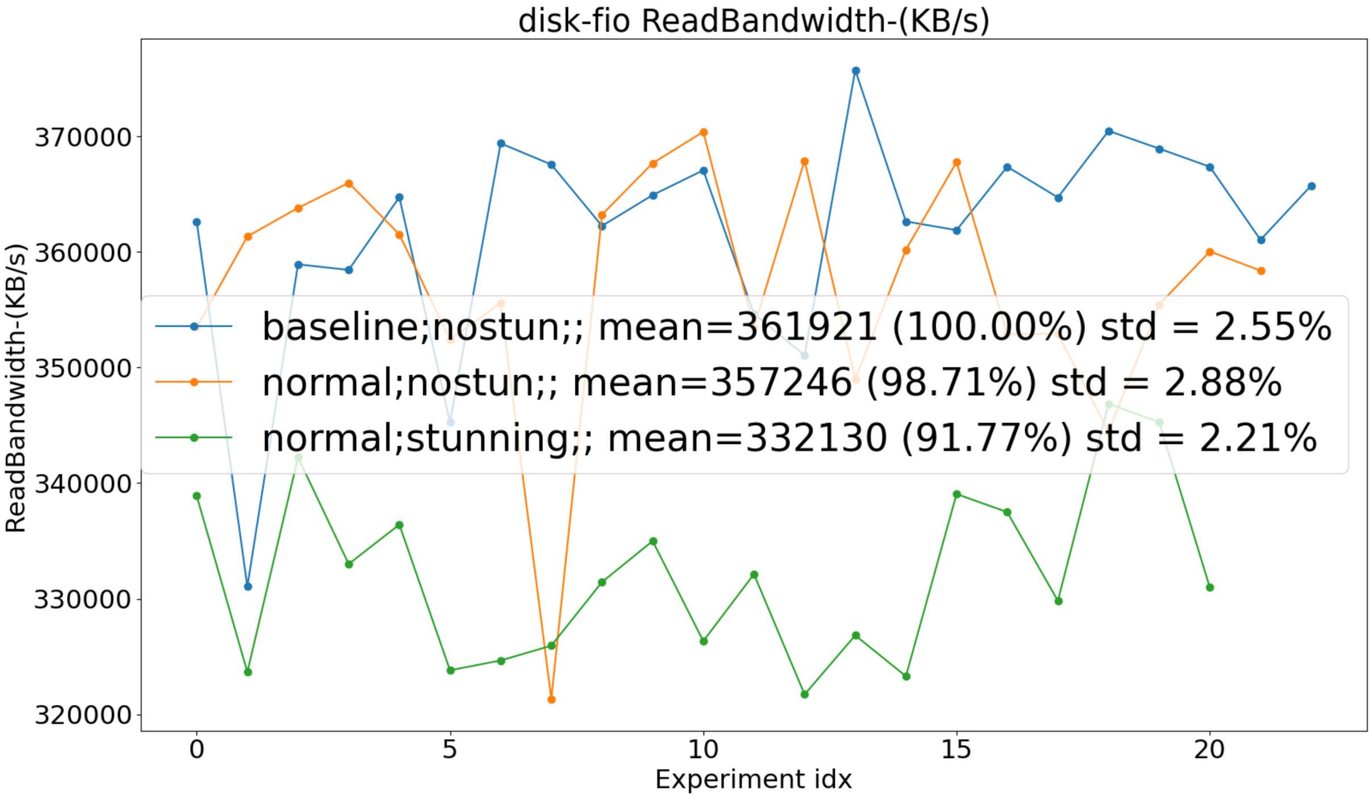
ASI performance - Redis throughput



ASI Performance - Aerospike throughput



ASI Performance - Disk-fIO bandwidth

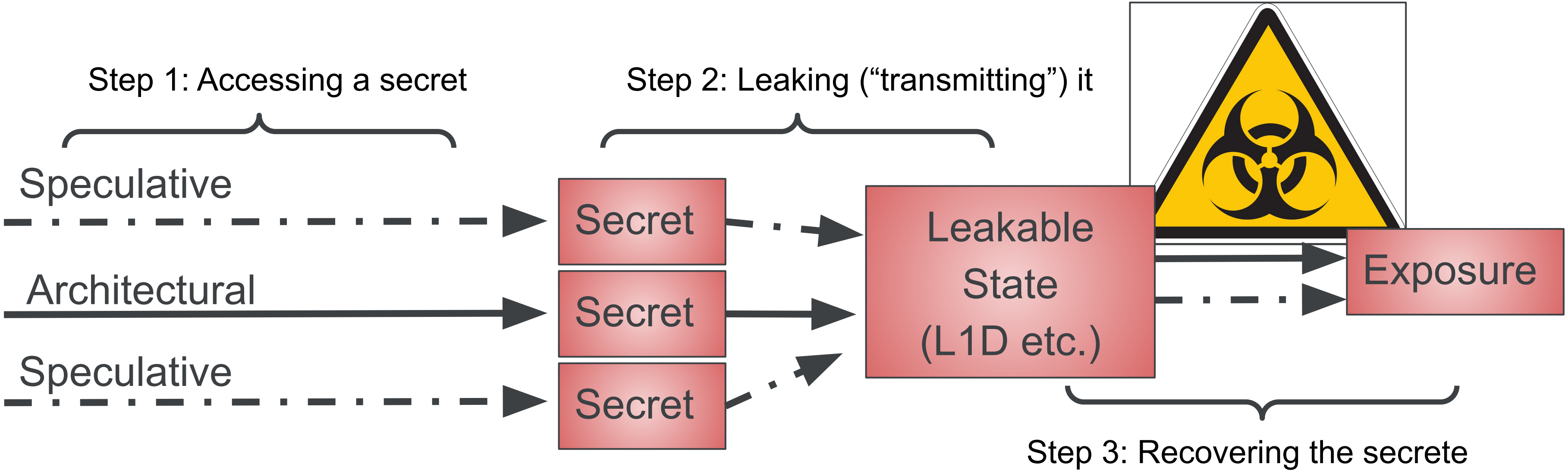


Bitter ASI pill to swallow

- The mechanism is not small/trivial
  - Modifying memory management, interrupt handling, KVM code
  - Well, neither are the ad-hoc mitigation mechanisms for retbleed etc.
- Discovering the allow-list requires a framework + expertise
  - So does the effort for mitigating the stream of vulnerabilities
- Annotating `kmalloc’s/vmalloc’s` with `GFP_X_NONSENSITIVE` pollutes the the source tree
  - There are some alternatives
  - We can try moving to a deny-list approach, but risk unknown exposure

Speculative Attacks and ASI refresher

Rethinking Mitigation - Understanding the Leak

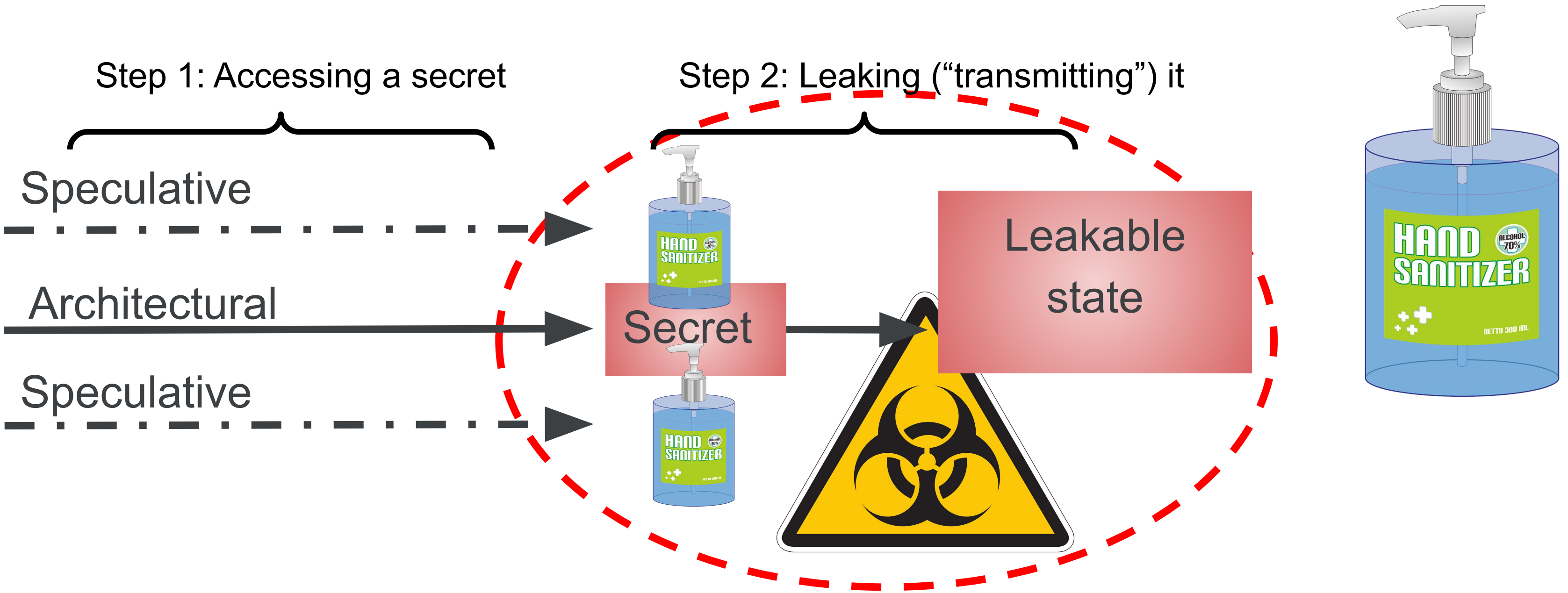


Status quo: u-arch buffers are always (potentially) contaminated with secrets

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

For more details: ofirweisse.com/MICRO2019_NDA.pdf

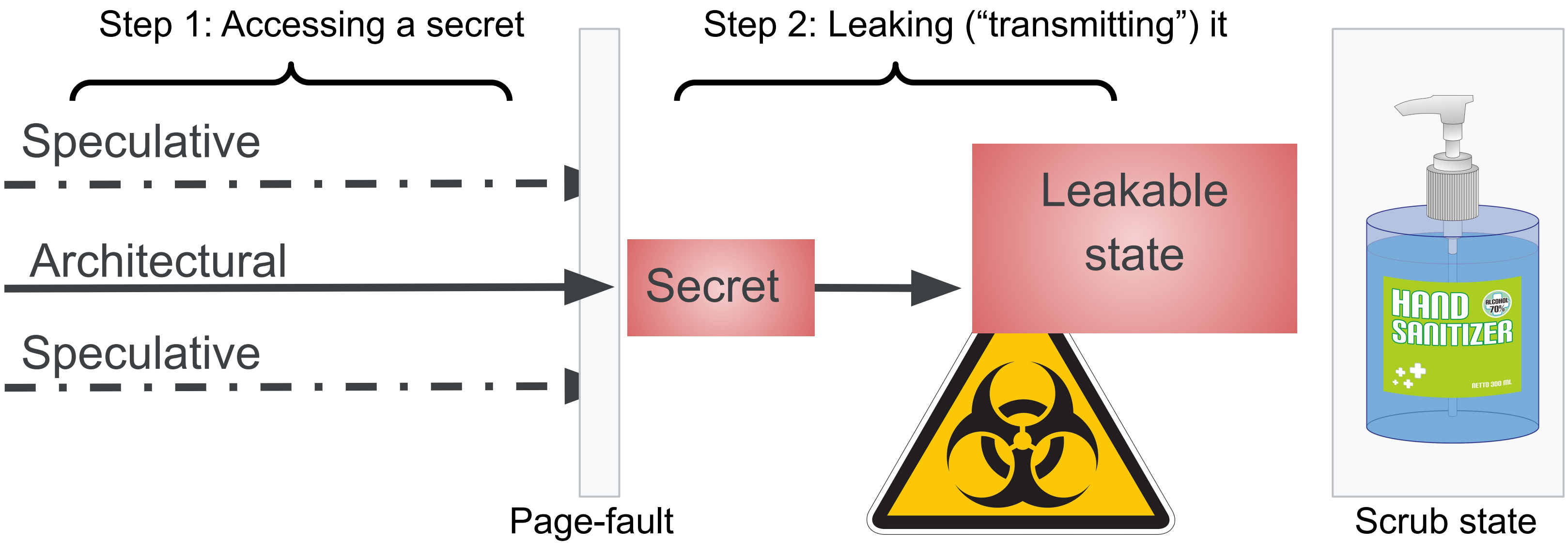
Rethinking Mitigation - Limiting Exposure



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

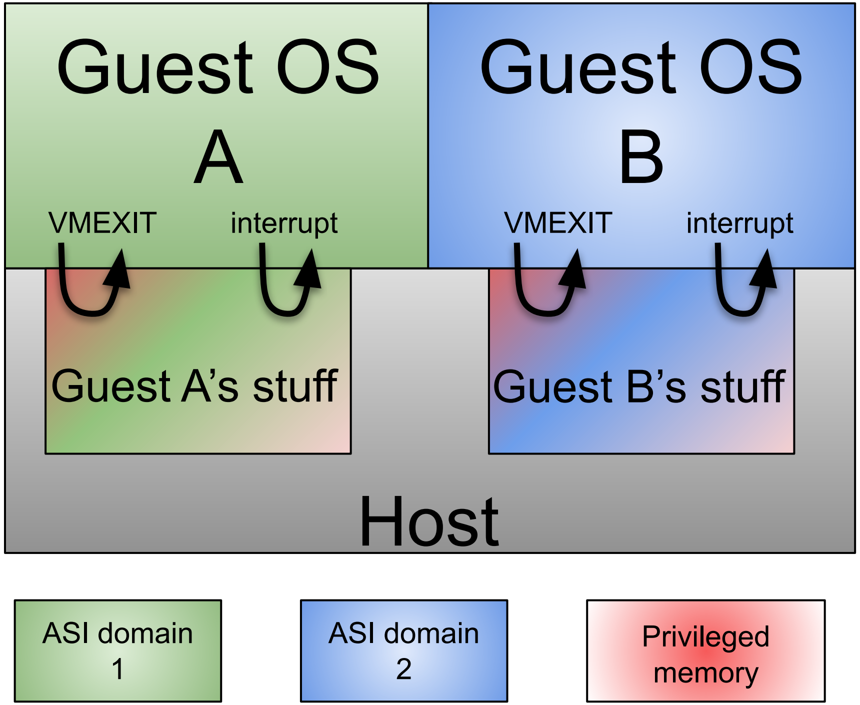


We want a way to circumscribe access to secrets and leakable state.

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

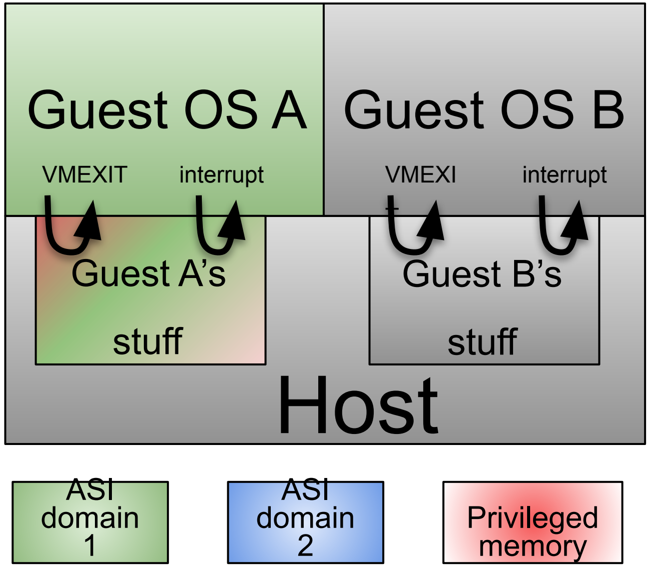
Address Space Isolation - Basic Idea

- 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



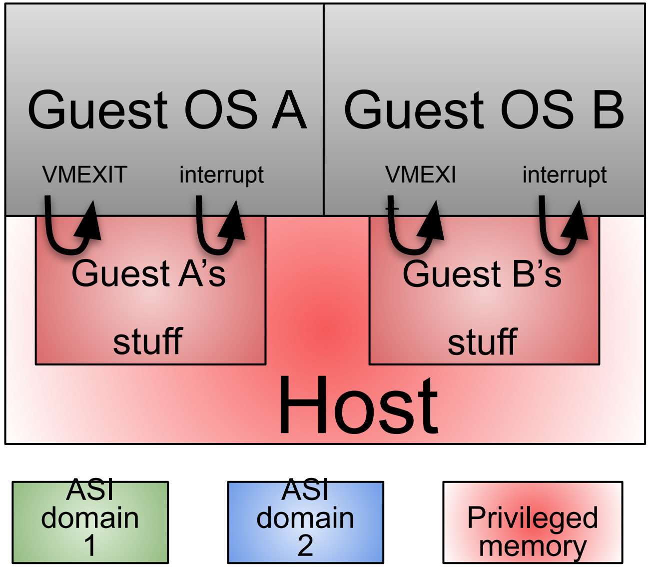
ASI Lifecycle

```
//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()
}
```



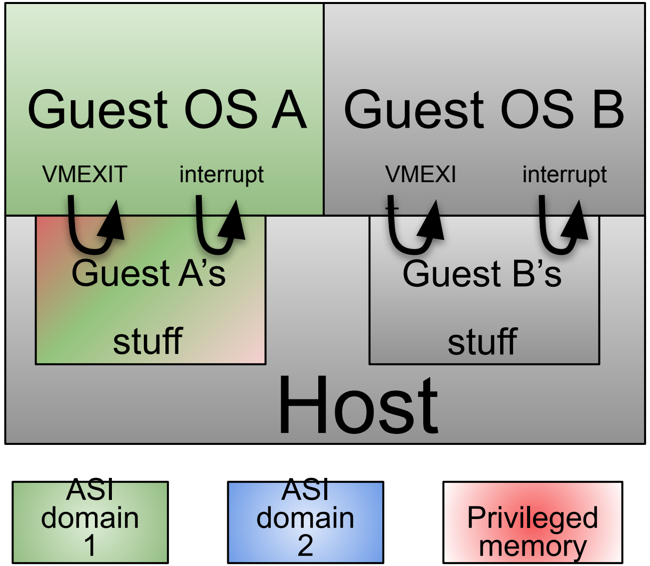
What happens on a page-fault?

1. Call asi_exit() which will:
2. Call pre_asi_exit() callback which will
   a. Stun sibling core
   b. Retbleed add-on: flush branch predictors
   c. Log exit stat
3. Switch page table (CR3 in Intel) to the privileged page-table



What happens on re-entry via asi_enter()?

1. Switch page table (CR3 in Intel) to the un-privileged Page-table
2. Call post_asi_enter() callback which will
   a. Flush L1D cache
   b. New attack add-on: and other uarch buffer
   c. Unstun sibling core



How to discover the appropriate allow-list?

How to discover the appropriate allow-list?

- We can count ASI-exit/VM-exit ratio
- Log stack traces of accessing code paths
- Log stack traces of memory allocation code paths

Analyzing 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
```

Analyzing Redis YCSB

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

Analyzing Redis YCSB

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);
```

Analyzing Redis YCSB

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%
```

What's next? Will upstream adopt ASI?