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
  - VM/host boundary
- 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 means
  ○ 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

- **baseline;nostun;; mean=84794 (100.00%) std = 1.19%**
- **normal;stunning;; mean=74166 (87.47%) std = 3.24%**
- **normal;nostun;; mean=82969 (97.85%) std = 1.84%**
ASI Performance - Aerospike throughput

- **baseline;nostun;; mean=154422 (100.00%) std = 9.14%**
- **normal;stunning;; mean=144766 (93.75%) std = 6.71%**
- **normal;nostun;; mean=150265 (97.31%) std = 9.13%**
ASI Performance - Disk-fIO bandwidth

![Graph showing disk-fio ReadBandwidth (KB/s)](image)

- baseline;nostun: mean=361921 (100.00%) std = 2.55%
- normal;nostun: mean=357246 (98.71%) std = 2.88%
- normal;stunning: mean=332130 (91.77%) std = 2.21%
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 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

Step 1: Accessing a secret

Speculative

Architectural

Speculative

Step 2: Leaking ("transmitting") it

Secret

Secret

Secret

Leakable State (L1D etc.)

Exposure

Step 3: Recovering the secret

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

Step 1: Accessing a secret

Step 2: Leaking ("transmitting") it

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

Step 1: Accessing a secret

Speculative

Architectural

Speculative

Step 2: Leaking ("transmitting") it

Page-fault

Secret

Leakable state

Hand sanitizer

Scrub state

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 separate page-table
- Touching data out of a domain results in a page-fault - cannot be speculative
- At first, only include kernel addresses
- ASI can be extended to include userspace memory
ASI Lifecycle

```c
// 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 preasi_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

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Time (seconds)</th>
<th>ASI/VM Exits</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>KVM/VCPU 0xffffc9001da89000/0</td>
<td>309.05</td>
<td>46160 / 4506402</td>
<td>1.02%</td>
</tr>
<tr>
<td>KVM/VCPU 0xffffc9001da89000/1</td>
<td>291.67</td>
<td>400531 / 1267665</td>
<td>31.60%</td>
</tr>
<tr>
<td>KVM/VCPU 0xffffc9001da89000/2</td>
<td>291.67</td>
<td>413946 / 2323131</td>
<td>17.82%</td>
</tr>
<tr>
<td>KVM/VCPU 0xffffc9001da89000/3</td>
<td>291.63</td>
<td>499027 / 1045507</td>
<td>47.73%</td>
</tr>
<tr>
<td>KVM/VCPU 0xffffc9001da89000/4</td>
<td>291.69</td>
<td>482687 / 2013058</td>
<td>23.98%</td>
</tr>
<tr>
<td>KVM/VCPU 0xffffc9001da89000/5</td>
<td>291.62</td>
<td>500809 / 2170556</td>
<td>23.07%</td>
</tr>
<tr>
<td>KVM/VCPU 0xffffc9001da89000/6</td>
<td>291.68</td>
<td>478710 / 1775451</td>
<td>26.96%</td>
</tr>
<tr>
<td>KVM/VCPU 0xffffc9001da89000/7</td>
<td>291.61</td>
<td>482880 / 2059408</td>
<td>23.45%</td>
</tr>
<tr>
<td>Total ASI exits</td>
<td>3304750</td>
<td>489981 / 6257089</td>
<td>7.83%</td>
</tr>
<tr>
<td>KVM/VCPU 0xffffc90039f35000/0</td>
<td>225.19</td>
<td>493745 / 1009584</td>
<td>48.91%</td>
</tr>
<tr>
<td>KVM/VCPU 0xffffc90039f35000/1</td>
<td>225.00</td>
<td>756191 / 2425297</td>
<td>31.18%</td>
</tr>
<tr>
<td>KVM/VCPU 0xffffc90039f35000/2</td>
<td>225.00</td>
<td>521712 / 1051189</td>
<td>49.63%</td>
</tr>
<tr>
<td>KVM/VCPU 0xffffc90039f35000/3</td>
<td>225.00</td>
<td>23353 / 73144</td>
<td>31.93%</td>
</tr>
<tr>
<td>KVM/VCPU 0xffffc90039f35000/4</td>
<td>224.91</td>
<td>19609 / 60075</td>
<td>32.64%</td>
</tr>
<tr>
<td>KVM/VCPU 0xffffc90039f35000/5</td>
<td>224.93</td>
<td>26320 / 81998</td>
<td>32.10%</td>
</tr>
<tr>
<td>KVM/VCPU 0xffffc90039f35000/6</td>
<td>224.93</td>
<td>22509 / 85046</td>
<td>26.47%</td>
</tr>
<tr>
<td>Total ASI exits</td>
<td>2353420</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Analyzing Redis YCSB

### Exit details

<table>
<thead>
<tr>
<th>RIP</th>
<th>data_addr</th>
<th>accessor</th>
<th>est_alloc_site</th>
<th>count</th>
<th>CDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xffffffff811ced3</td>
<td>0xffffffff885363e42c938</td>
<td>el/sched/exclusive.c:7283</td>
<td>PO: ./kernel/fork.c:1636</td>
<td>276673</td>
<td>1.000000</td>
</tr>
<tr>
<td>0xffffffff811ced3</td>
<td>0xffffffff88554bc49938</td>
<td>el/sched/exclusive.c:7283</td>
<td>PO: ./kernel/events/core.c:10843</td>
<td>233775</td>
<td>0.887946</td>
</tr>
<tr>
<td>0xffffffff811c79b1</td>
<td>0xffffffff8612b00070</td>
<td>rnel/sched/cpuacct.c:1284</td>
<td>PO: ./mm/percpu-vm.c:284</td>
<td>151020</td>
<td>0.793267</td>
</tr>
<tr>
<td>0xffffffff811c79b1</td>
<td>0xffffffff8612f00070</td>
<td>rnel/sched/cpuacct.c:1284</td>
<td>PO: ./mm/percpu-vm.c:284</td>
<td>54685</td>
<td>0.732103</td>
</tr>
<tr>
<td>0xffffffff811c79b1</td>
<td>0xffffffff8612f00070</td>
<td>rnel/sched/cpuacct.c:1284</td>
<td>PO: ./mm/percpu-vm.c:284</td>
<td>45065</td>
<td>0.709956</td>
</tr>
<tr>
<td>0xffffffff8119268d</td>
<td>0xffffffff8854bc49938</td>
<td>rnel/sched/cputime.c:154</td>
<td>PO: ./kernel/events/core.c:10843</td>
<td>37279</td>
<td>0.691704</td>
</tr>
<tr>
<td>0xffffffff811c79b1</td>
<td>0xffffffff8a05cc6efc0</td>
<td>rnel/sched/cpuacct.c:1284</td>
<td>PO: ./mm/percpu-vm.c:284</td>
<td>32923</td>
<td>0.676606</td>
</tr>
<tr>
<td>0xffffffff811a2686</td>
<td>0xffffffff885363e42c938</td>
<td>rnel/sched/cpuacct.c:1284</td>
<td>PO: ./kernel/fork.c:1636</td>
<td>31714</td>
<td>0.663272</td>
</tr>
<tr>
<td>0xffffffff811ced4d</td>
<td>0xffffffff885596c45c8</td>
<td>rnel/sched/cpuacct.c:154</td>
<td>PO: ./mm/percpu-vm.c:284</td>
<td>30228</td>
<td>0.650428</td>
</tr>
<tr>
<td>0xffffffff811c79b1</td>
<td>0xffffffff8883a2b930</td>
<td>rnel/sched/cpuacct.c:1284</td>
<td>PO: ./kernel/fork.c:1636</td>
<td>29209</td>
<td>0.638185</td>
</tr>
<tr>
<td>0xffffffff811c79b2</td>
<td>0xffffffff88555c508c8</td>
<td>rnel/sched/cpuacct.c:1284</td>
<td>config consume rt capacity</td>
<td>24593</td>
<td>0.626356</td>
</tr>
<tr>
<td>0xffffffff815f0880</td>
<td>0xffffffff885486480380</td>
<td>./lib/lmalloc.c:97</td>
<td>./fs/eventfd.c:658</td>
<td>24471</td>
<td>0.616395</td>
</tr>
<tr>
<td>0xffffffff811c79b1</td>
<td>0xffffffff8a060a6de0</td>
<td>rnel/sched/cpuacct.c:1284</td>
<td>PO: ./mm/percpu-vm.c:284</td>
<td>21122</td>
<td>0.606485</td>
</tr>
<tr>
<td>0xffffffff811c79b1</td>
<td>0xffffffff8a060aece90</td>
<td>rnel/sched/cpuacct.c:1284</td>
<td>PO: ./mm/percpu-vm.c:284</td>
<td>20673</td>
<td>0.597930</td>
</tr>
</tbody>
</table>
Analyzing Redis YCSB

Exit details

```c
curr->se.exec_start = now;

schedstat_set(curr->se.statistics.exec_max,
               max(curr->se.statistics.exec_max, delta_exec));

curr->se.sum_exec_runtime += delta_exec;

account_group_exec_runtime(curr, delta_exec);
```
Analyzing Redis YCSB

Exit details

```
static int copy_signal(unsigned long clone_flags, struct task_struct *tsk)
{
    struct signal_struct *sig;

    if (clone_flags & CLONE_THREAD)
        return 0;

    #ifdef CONFIG_ADDRESS_SPACE_ISOLATION
        sig = kzalloc(sizeof(struct signal_struct),
                        GFP_KERNEL | GFP_NONSENSITIVE);
    ```
What’s next? Will upstream adopt ASI?