SYZOS: practical KVM fuzzing

Alexander Potapenko <glider@google.com>
Linux Plumbers Conference 2025

The Challenge of Fuzzing KVM

1. Host (L0): complex, non-atomic setup:
    - Requires valid sequences of `KVM_SET_CPUID2`, `KVM_SET_SREGS`, `KVM_SET_USER_MEMORY_REGION` etc.
2. Guest (L1/L2): existing solutions within syzkaller were limited:
    - Strategy A: Hardcoded Assembly Blobs
        - Injecting static blobs via legacy setup calls.
        - Limitation: Zero flexibility. The fuzzer cannot mutate parameters or logic, restricting coverage to a single "happy path."
    - Strategy B: Pseudo-Instructions
        - Generating macros (e.g., `PSEUDO_WRMSR`, `PSEUDO_HYPERCALL`).
        - Limitation: These lower to raw assembly. The fuzzer mutates individual bytes, destroying the logic immediately.
3. Reachability: Some CPU state is inaccessible to L0 ioctls.
4. Coordination: Complex scenarios require turn-based execution (e. g. L0 ↔ L1).

Foundation: Fix KVM Descriptions

- Not too many updates of `sys/linux/dev_kvm.txt` since 2019
- 50+ changes refining the descriptions to prune the search space
  - Opaque blobs → architecture-specific structs
  - Abstract int64 → constants or flags
  - New resource types to properly chain the syscalls
  - Splitting polymorphic ioctls (e.g. `KVM_GET_MSRS` is both a system and a VCPU ioctl)
  - Apply `no_squash` for complex struct argument to forbid arbitrary bitflips

Host: pseudo-syscalls to the rescue

- `syz_kvm_setup_syzos_vm()`:
  - `ioctl(vmfd, KVM_SET_USER_MEMORY_REGION, &memreg)` x N
- `syz_kvm_add_vcpu$x86()`:
  - `ioctl(vmfd, KVM_CREATE_VCPU, cpu_id)`
  - `ioctl(cpufd, KVM_GET_SREGS, &sregs)`
  - Set up segments, page table, IDT
  - `ioctl(cpufd, KVM_SET_SREGS, &sregs)`

Syzkaller will generate these syscalls along with the normal ones, increasing the chance to hit more interesting paths.
They are implemented in C and are part of syz-executor.

L1 Guest: Introducing SYZOS

SYZOS is an immutable C library with an easy-to-fuzz API.
The fuzzer generates a sequence of API calls (commands) for L1 to execute.

SYZOS primitives for x86 (ARM is also supported):

- Exit, execution, I/O:
  - `SYZOS_API_UEXIT` – trigger a page fault, return 1 argument to Host.
  - `SYZOS_API_CODE` – execute a Host-supplied instruction blob.
  - `SYZOS_API_IN_DX`, `SYZOS_API_OUT_DX` – execute the `in` and `out` instruction (port in DX).
- Privileged operations:
  - `SYZOS_API_CPUID` – execute the `cpuid` instruction.
  - `SYZOS_API_WRMSR`, `SYZOS_API_RDMSR` – read/write MSRs.
  - `SYZOS_API_WR_CRN`, `SYZOS_API_WR_DRN` – write CR0..CR8, DR0..DR7.
- Interrupt handling:
  - `SYZOS_API_SET_IRQ_HANDLER` – install an IRQ handler for the given interrupt.

Nested Virtualization: x86 L2 Primitives

SYZOS acts as a lightweight L1 hypervisor to fuzz nested virtualization. It abstracts architectural differences (VMX vs SVM) into high-level commands.

SYZOS Primitives:

- VM Lifecycle Setup:
  - `SYZOS_API_ENABLE_NESTED` – Enable virtualization extensions (VMX/SVM).
  - `SYZOS_API_NESTED_CREATE_VM` – Initialize VMCS/VMCB and Nested Page Tables.
  - `SYZOS_API_NESTED_LOAD_CODE` – Inject small instruction blobs into L2 memory. (Note: Currently supports code snippets, not the full SYZOS kernel)
- Execution Control:
  - `SYZOS_API_NESTED_VMLAUNCH` – Perform initial entry into the L2 Guest.
  - `SYZOS_API_NESTED_VMRESUME` – Re-enter the L2 Guest after an exit.
- State Mutation:
  - `SYZOS_API_NESTED_*WRITE_MASK` – Mutate VMCS/VMCB control fields.

ARM64 Specifics

1. Fuzzing the GICv3 ITS (Interrupt Translation Service)
   - The ITS relies on complex in-memory command queues and translation tables.
   - SYZOS Primitives:
     - `SYZOS_API_ITS_SETUP`: Allocates tables and configures the GIC base.
     - `SYZOS_API_ITS_SEND_CMD`: Injects structured GIC commands (e.g., MAPD, MOVI) into the queue to fuzz the kernel's command processor.

2. Fuzzing the Hypervisor Interface (L0 Traps)
   - Testing the boundary between the Guest and the L0 Hypervisor/Firmware.
   - Hypercalls: `SYZOS_API_HVC` / `SYZOS_API_SMC` with fuzzer-controlled registers.
   - Goal: Fuzz L0 handlers for PSCI (Power State Coordination) and SMCCC (SiP service) calls.

Execution Model

- Finite execution:
  - syzkaller generates a finite, end-to-end program (sequence of system calls).
  - The "Run Loop" is unrolled into discrete `KVM_RUN` syscalls.
- The yield mechanism:
  - `SYZOS_API_UEXIT` is the explicit handshake to yield control from L1 to L0.
  - This allows the fuzzer to interleave L0 mutations (e.g., memory unmapping) precisely between Guest steps.
- Concurrency:
  - Syzkaller can mark syscalls as `async` to provoke Host/Guest data races.

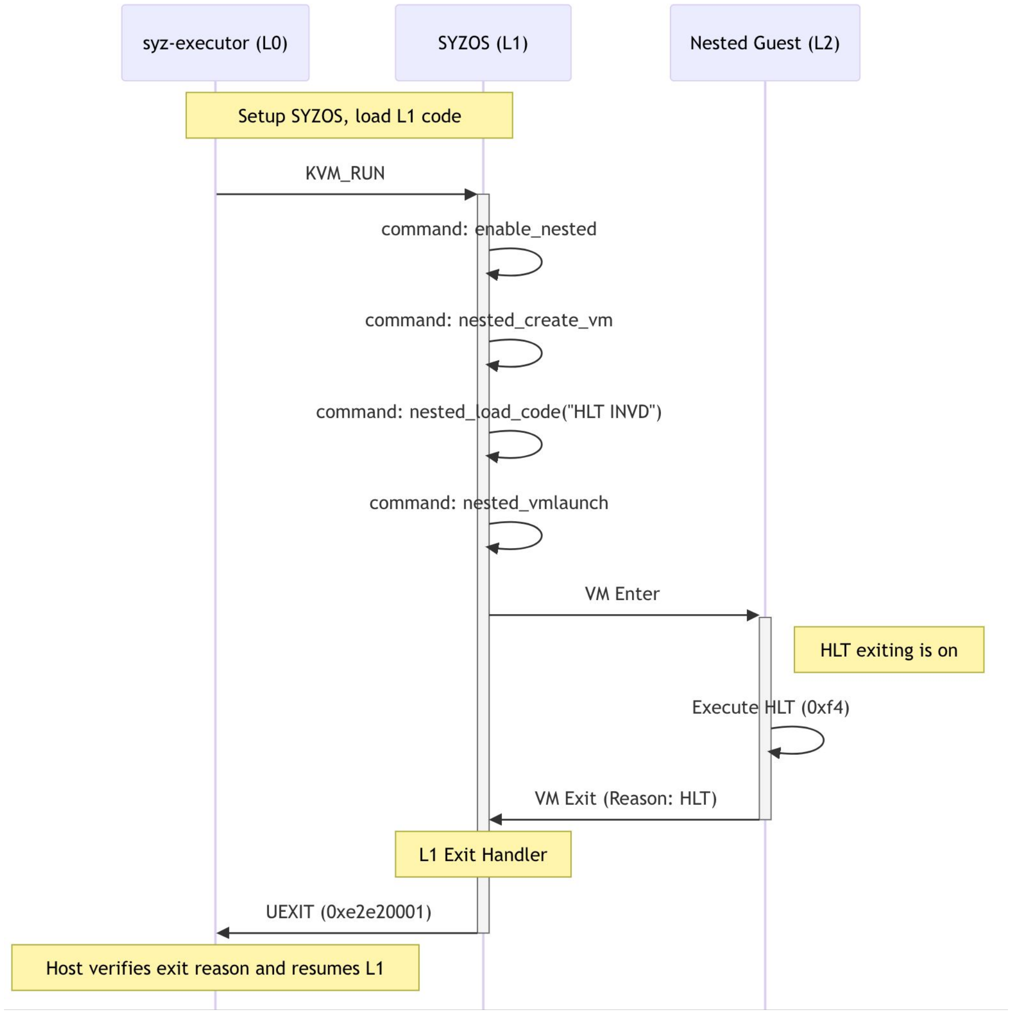
A Case Study

```
# Setup (not shown on the diagram):

r0 = openat$kvm(...)
r1 = ioctl$KVM_CREATE_VM(r0, ...)
r2 = syz_kvm_setup_syzos_vm$x86(...)
r3 = syz_kvm_add_vcpu$x86(...,
      [@enable_nested,
       @nested_create_vm,
       @nested_load_code("f40f08"),
       @nested_vmlaunch,
       @nested_intel_vmwrite_mask(
         {0x4002, 0x0, 0x80, 0x0}
       ),
       @nested_vmresume
      ])


# Start:

ioctl$KVM_RUN(r3)
```



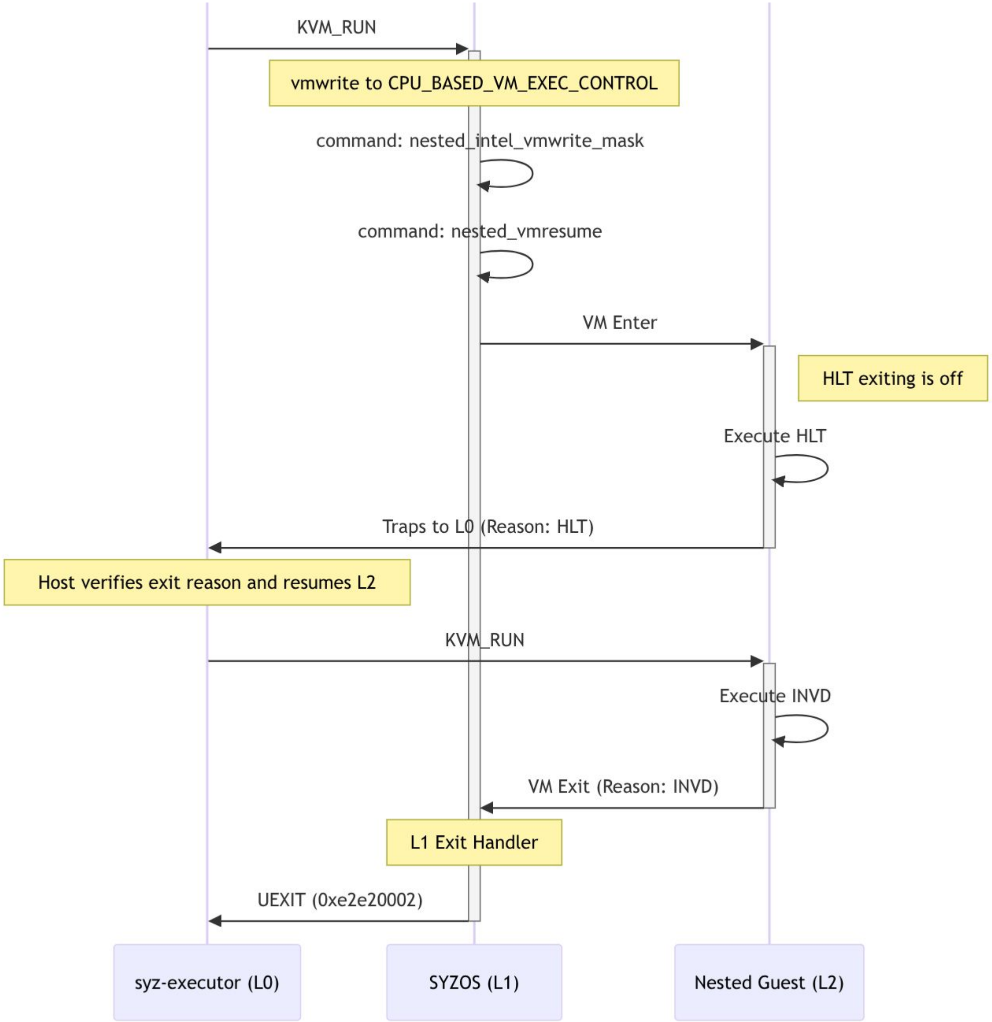
A Case Study (cont.)

```
# Continue after UEXIT:

ioctl$KVM_RUN(r3)

# Continue after HLT:

ioctl$KVM_RUN(r3)
```



Validation and Findings

Qualitative impact

- Early fruit: **9 bugs** with SYZOS-based reproducers reported in 2025.
- "Collateral Damage": overhauling KVM syscall descriptions as part of SYZOS development.
- Deeper state access: replaced rigid "Happy path" instruction blobs with mutable logic to reach previously invisible states.

Corpus adoption

- The fuzzer actively chooses SYZOS programs (20–30% of the programs in the minimal x86 KVM corpus, 3–4x growth in the past two months).
- 294 Intel and 157 AMD programs successfully trigger nested VM launches.

Coverage Improvements

Fuzzing on GCE (syzbot.org): 200+ machines, different kernel versions.

- `arch/arm64/kvm` coverage* up by 5% with SYZOS rollout.
- x86-related KVM coverage up by 7–12%.

Fuzzing on bare-metal: 1–3 machines per architecture, single kernel, disabled the legacy guest blobs.

- Intel and AMD-specific KVM coverage doubled since SYZOS adoption (Intel: 58%, AMD: 54%).
- Legacy guest blobs were disabled to prove that the coverage gains are purely driven by SYZOS logic, not the old "happy path" instructions.

* Here and below we mean line coverage that is easier to aggregate. No 1:1 correspondence to basic block coverage.

Future Plans

- Deeper nesting: run SYZOS inside SYZOS.
- Better ARM support: page tables, NV, devices.
- Non-64-bit guest modes on x86.
- Analyze coverage gaps.
- Continuous testing.