Detecting semantic bugs using differential fuzzing

Mara Mihali

Linux Plumbers 2021

Many classes of bugs are easy to detect...

Many classes of bugs are easy to detect...

- cause assertion failures
- crash the system
- trigger other forms of undefined behaviour
    - detectable using dynamic or static analysis tools (e.g. KASAN)

Many classes of bugs are easy to detect...

- cause assertion failures
- crash the system
- trigger other forms of undefined behaviour
  - detectable using dynamic or static analysis tools (e.g. KASAN)
- **Why are semantic bugs different?**
  - make program operate incorrectly, possibly producing unintended output

Many classes of bugs are easy to detect...

- cause assertion failures
- crash the system
- trigger other forms of undefined behaviour
  - detectable using dynamic or static analysis tools (e.g. KASAN)
- **Why are semantic bugs different?**
  - make program operate incorrectly, possibly producing unintended output
  - **but** might not crash the program or trigger assertion failures

Many classes of bugs are easy to detect...

- cause assertion failures
- crash the system
- trigger other forms of undefined behaviour
  - detectable using dynamic or static analysis tools (e.g. KASAN)
- **Why are semantic bugs different?**
  - make program operate incorrectly, possibly producing unintended output
  - **but** might not crash the program or trigger assertion failures
    - not detectable using existing analysis tools
    - require the developer to manually inspect and test the program

How can we find semantic bugs?

Testing a system’s specification

- a specification formalises the system’s *intended behaviour*
- this could be used to write tests in order to detect semantic bugs

Testing a system’s specification

- a specification formalises the system’s *intended behaviour*
- this could be used to write tests in order to detect semantic bugs
- gets more difficult to achieve and maintain as the size of the system increases

Testing a system’s specification

- a specification formalises the system’s *intended behaviour*
- this could be used to write tests in order to detect semantic bugs
- gets more difficult to achieve and maintain as the size of the system increases
- some *existing*, large systems have no centralised specification

Testing a system’s specification

- a specification formalises the system’s *intended behaviour*
- this could be used to write tests in order to detect semantic bugs
- gets more difficult to achieve and maintain as the size of the system increases
- some *existing*, large systems have no centralised specification
- **Linux kernel**
    - specification = documentation + `man` pages + implied expectations of user programs

Testing a system's specification

- a specification formalises the system's *intended behaviour*
- this could be used to write tests in order to detect semantic bugs
- gets more difficult to achieve and maintain as the size of the system increases
- some *existing*, large systems have no centralised specification
- **Linux kernel**
  - specification = documentation + `man` pages + implied expectations of user programs
  - test suites available to detect *regressions*
    - **but** require significant amount of engineering effort to extend and maintain

Differential Fuzzing

- automates detection of semantic bugs
- provides same input to different implementations of the same system and cross-compares resulting behaviour
- if systems disagree, *at least* one of them is wrong

Differential Fuzzing

- automates detection of semantic bugs
- provides same input to different implementations of the same system and cross-compares resulting behaviour
- if systems disagree, *at least* one of them is wrong
- **Differential fuzzing for Linux Kernel**
    - non-trivial, several technical challenges involved

Differential Fuzzing

- automates detection of semantic bugs
- provides same input to different implementations of the same system and cross-compares resulting behaviour
- if systems disagree, *at least* one of them is wrong
- **Differential fuzzing for Linux Kernel**
    - non-trivial, several technical challenges involved
        - kernel nondeterminism

Differential Fuzzing

- automates detection of semantic bugs
- provides same input to different implementations of the same system and cross-compares resulting behaviour
- if systems disagree, *at least* one of them is wrong
- **Differential fuzzing for Linux Kernel**
  - non-trivial, several technical challenges involved
    - kernel nondeterminism
      - programs with non-deterministic behaviour
      - concurrency
      - resource exhaustion
      - background activity
      - timing dependencies
      - global accumulated state

Differential Fuzzing

- automates detection of semantic bugs
- provides same input to different implementations of the same system and cross-compares resulting behaviour
- if systems disagree, *at least* one of them is wrong
- **Differential fuzzing for Linux Kernel**
  - non-trivial, several technical challenges involved
    - kernel nondeterminism
      - programs with non-deterministic behaviour
      - concurrency
      - resource exhaustion
      - background activity
      - timing dependencies
      - global accumulated state
    - implementation-defined behaviour

Differential Fuzzing

- automates detection of semantic bugs
- provides same input to different implementations of the same system and cross-compares resulting behaviour
- if systems disagree, *at least* one of them is wrong
- **Differential fuzzing for Linux Kernel**
  - non-trivial, several technical challenges involved
    - kernel nondeterminism
      - programs with non-deterministic behaviour
      - concurrency
      - resource exhaustion
      - background activity
      - timing dependencies
      - global accumulated state
    - implementation-defined behaviour
    - state space of the input is unbounded

Comparison Candidates

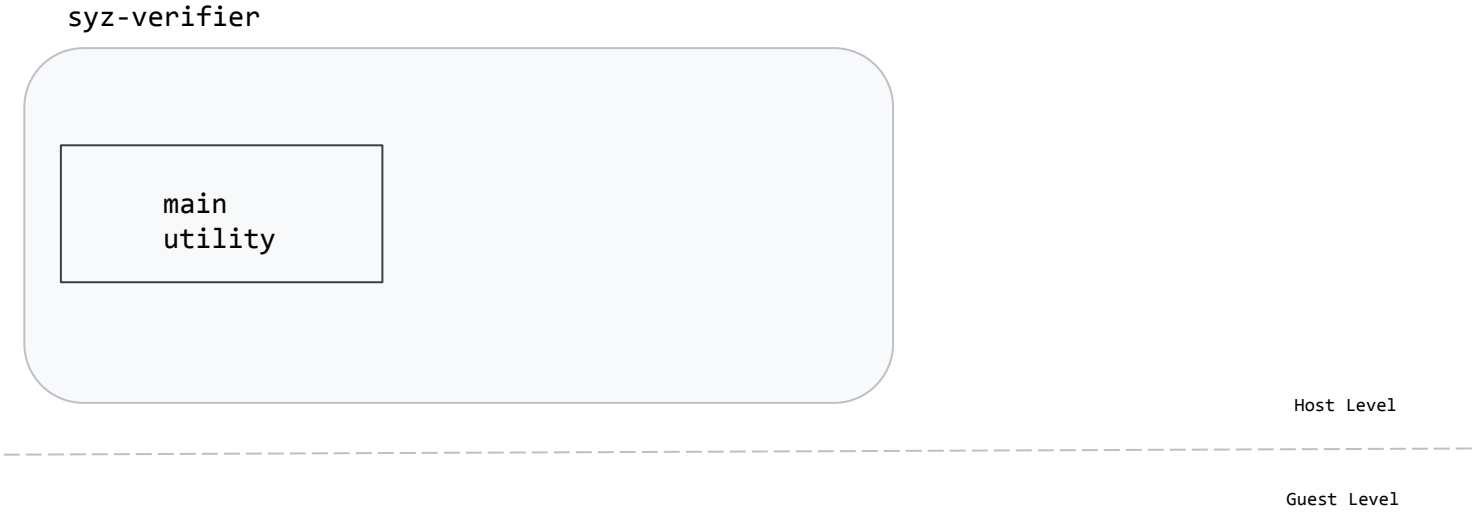
- **LTS vs mainline**
  - prevent bugs from reaching the next release
- **different LTS releases**
  - *neighbouring:* not many intentional differences but most bugs are present in both versions
  - *distant:* need a mechanism to whitelist intentional differences
- **minor LTS updates**
  - a way to ensure bugs were actually fixed by the update
- **different kernel implementation (Linux vs gVisor)**
  - could uncover real semantic bugs
  - however, many false positives (due to intentional differences) that need to be accounted for

`syz-verifier`

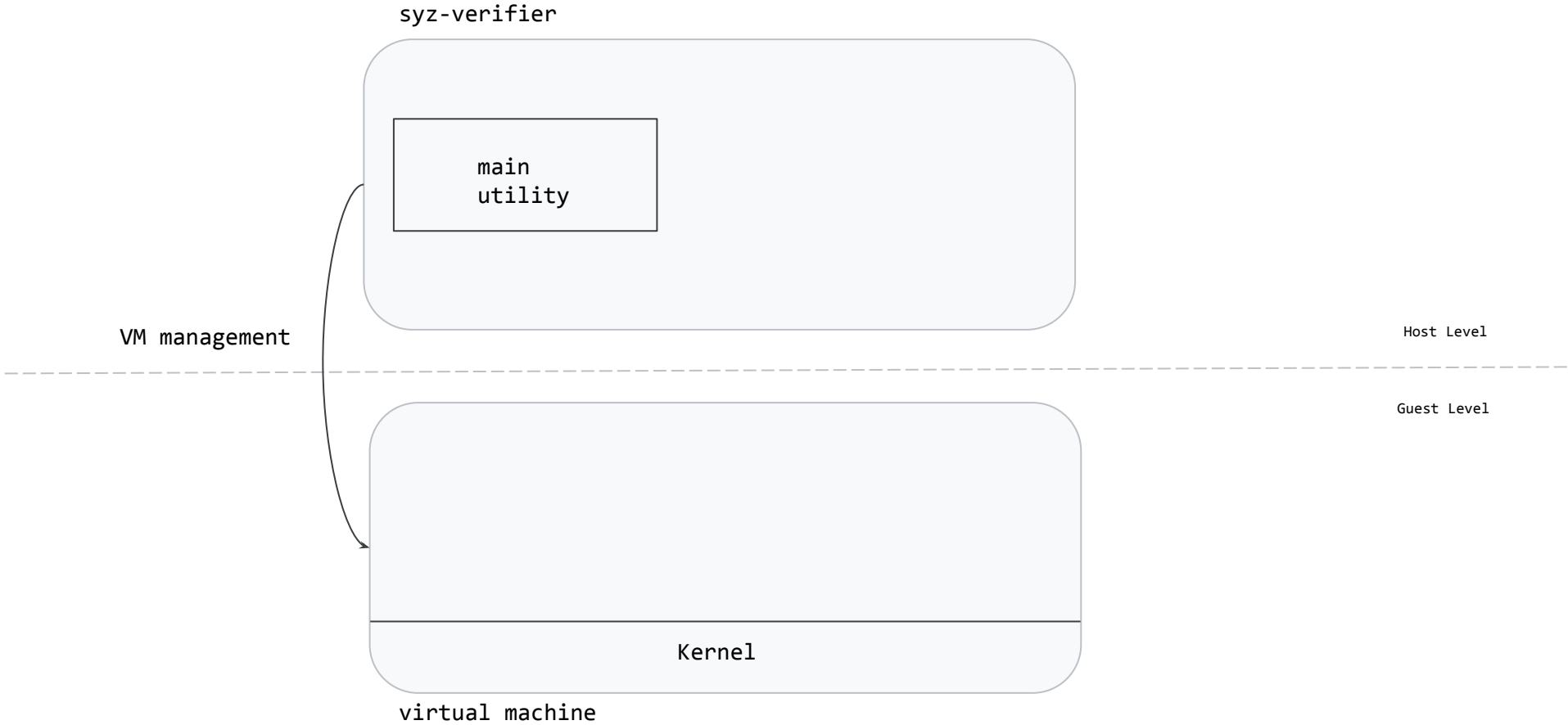
- differential fuzzing tool for the Linux kernel
- part of the `syzkaller` project, additionally providing unsupervised coverage-guided kernel fuzzing
- generates a continuous stream of random programs (i.e. sequences of syscalls)
- dispatches the programs for execution on different versions of the Linux kernel
- gathers and verifies whether the returned results are the same for all kernels
- for each syscall, `syz-verifier` reports:
  - errno
  - whether the VM crashed executing the program
- in cases of mismatches, `syz-verifier` creates an execution report for the program for further inspection

Architecture Overview

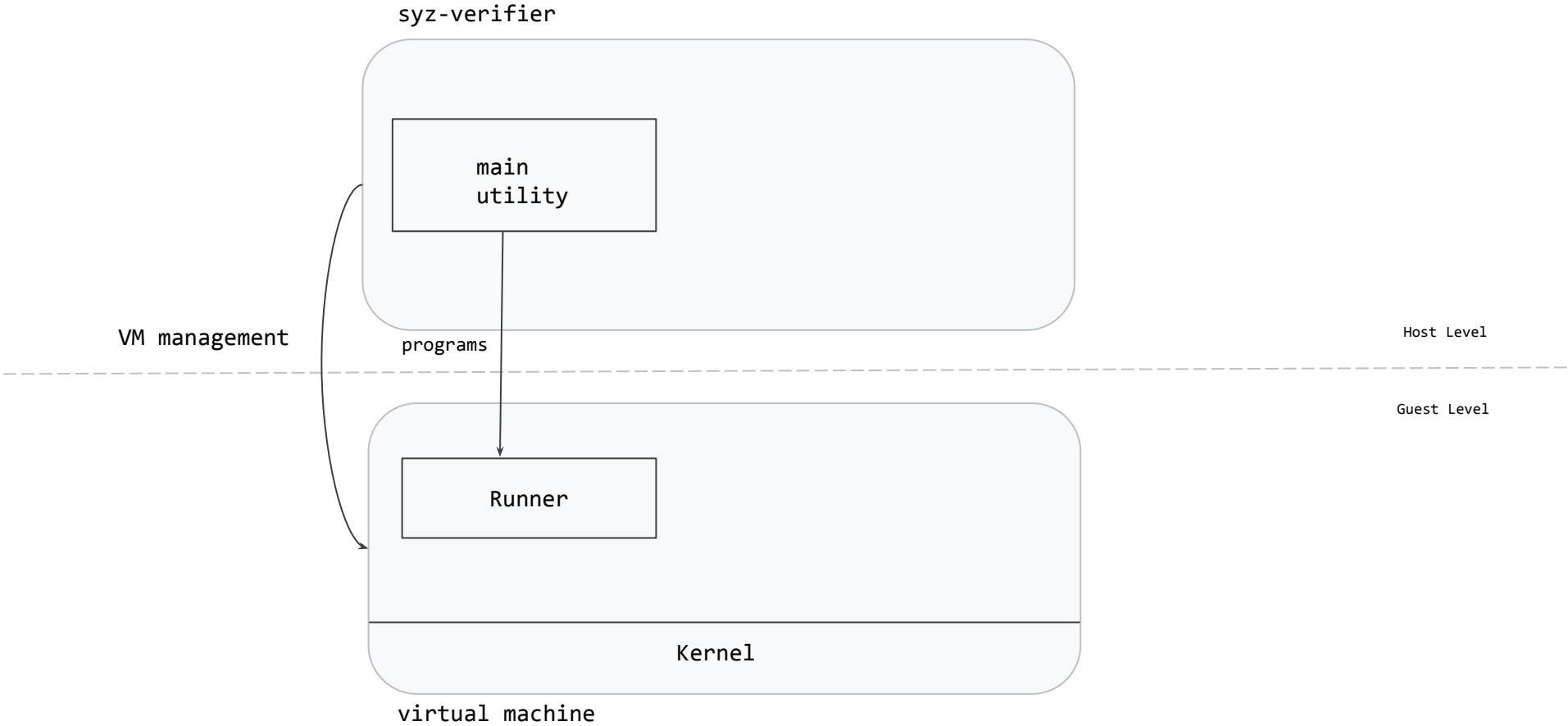
Architecture Overview



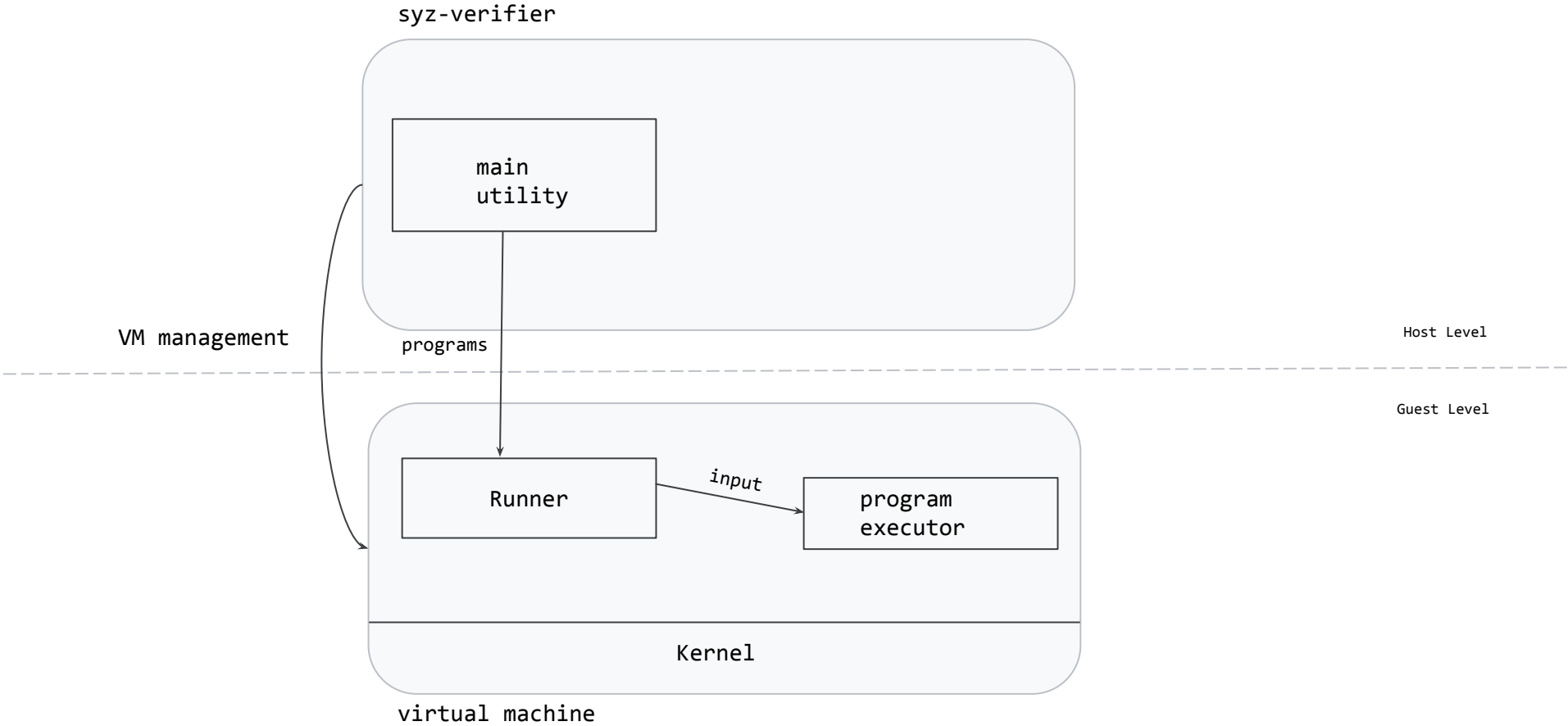
Architecture Overview



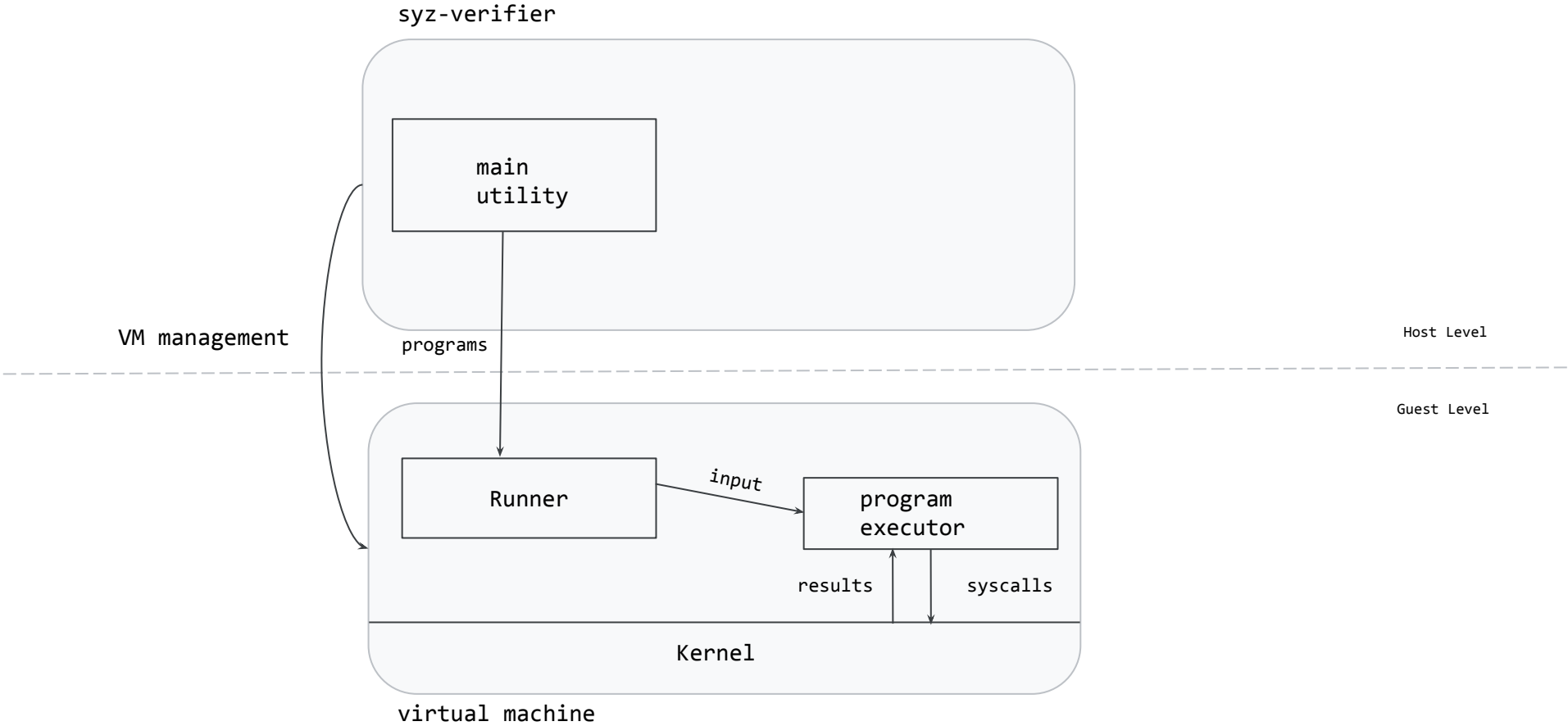
Architecture Overview



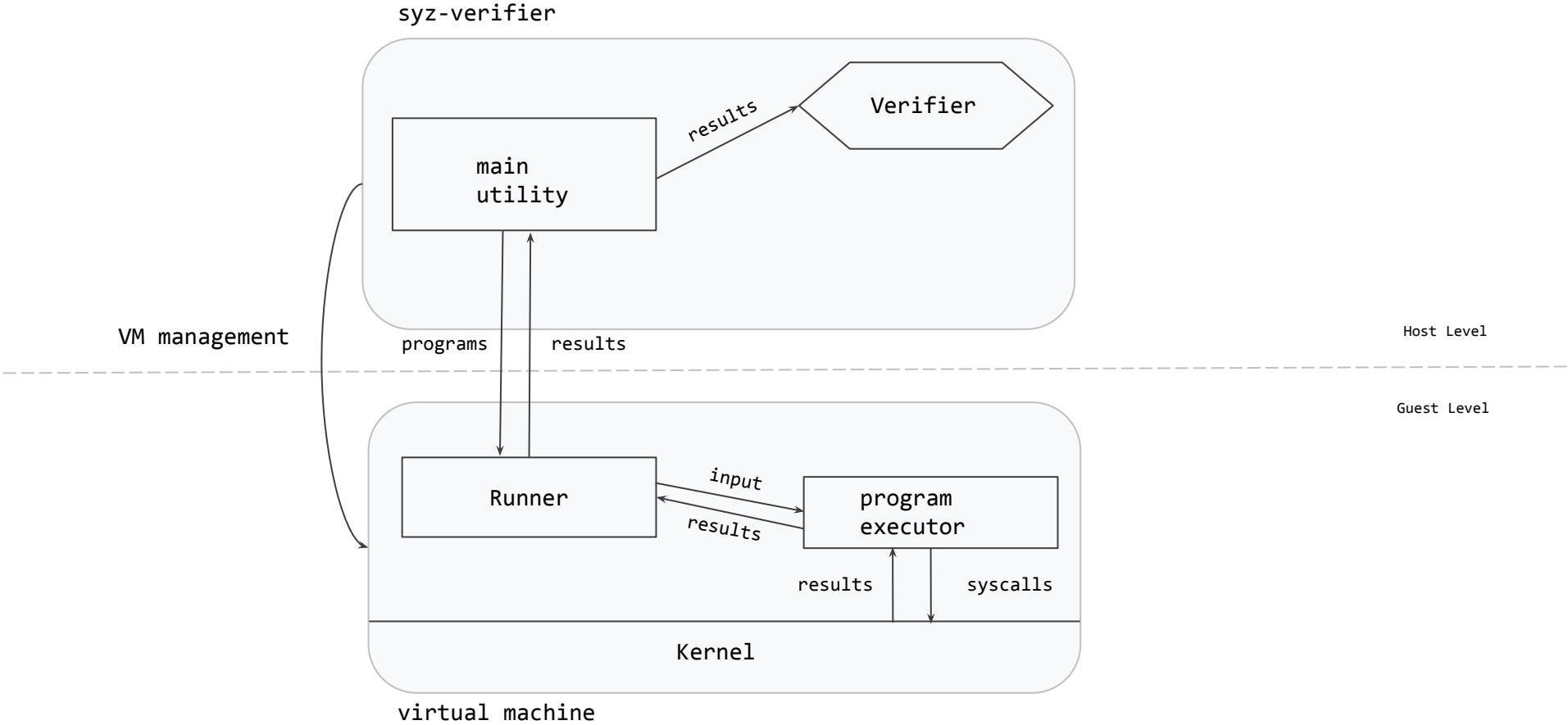
Architecture Overview



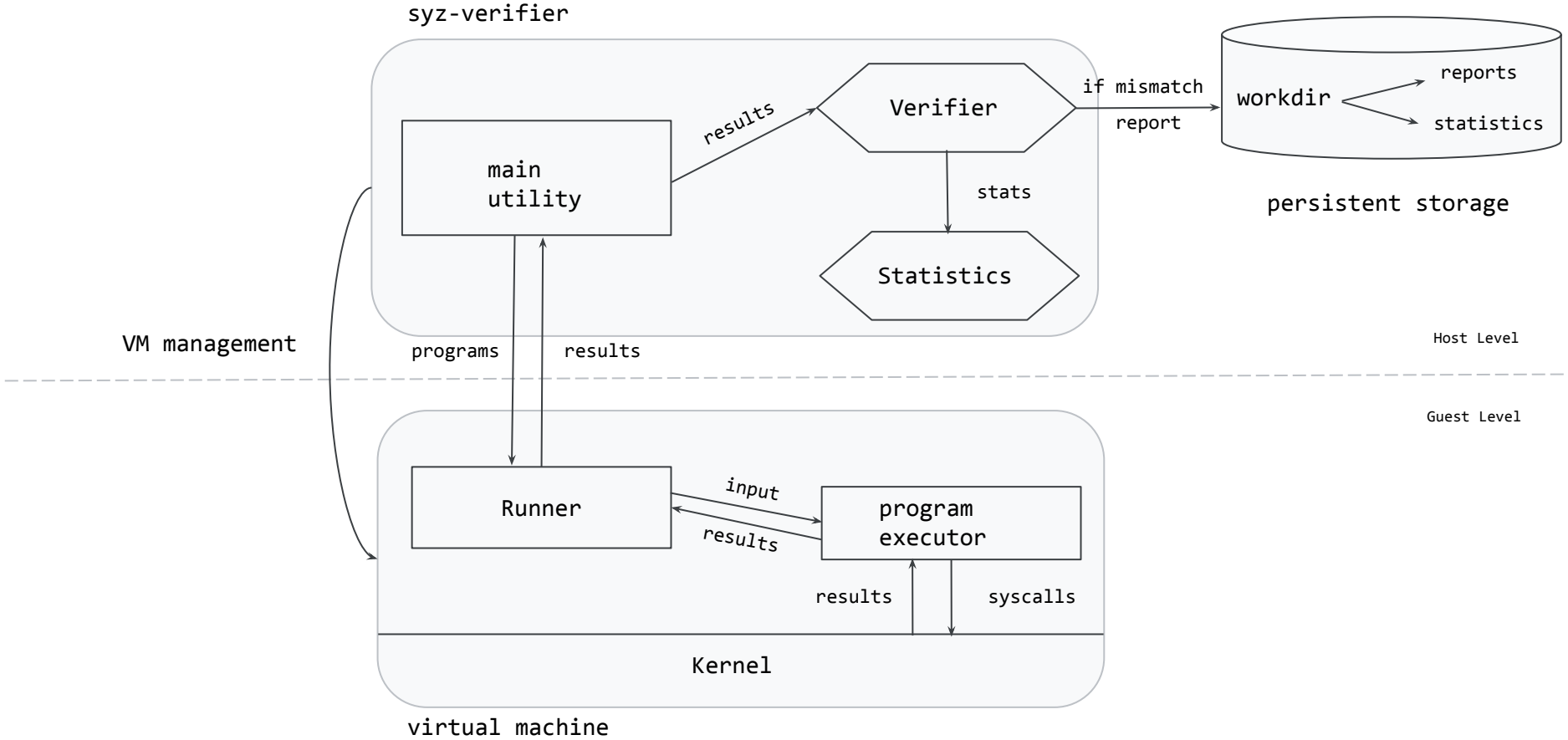
Architecture Overview



Architecture Overview



Architecture Overview



Bisecting Mismatches

io_uring_setup

io_uring_setup

- Old Kernel (v5.10.47) : `EBADF` (`bad file descriptor`)
- New Kernel (v5.13): `ENXIO` (`no such device or address`)

io_uring_setup

- Old Kernel (v5.10.47) : `EBADF` (`bad file descriptor`)
- New Kernel (v5.13): `ENXIO` (`no such device or address`)

io_uring: disable io-wq attaching

```
Moving towards making the io_wq per ring per task, so we can't really
share it between rings. Which is fine, since we've now dropped some
of that fat from it.

Retain compatibility with how attaching works, so that any attempt to
attach to an fd that doesn't exist, or isn't an io_uring fd, will fail
like it did before.

-       f = fdget(p->wq_fd);
-       if (!f.file)
-               return -EBADF;


+       f = fdget(p->wq_fd);
+       if (!f.file)
+               return -ENXIO;
```

io_uring_setup

- Old Kernel (v5.10.47) : `EBADF` (`bad file descriptor`)
- New Kernel (v5.13): `ENXIO` (`no such device or address`)

io_uring: disable io-wq attaching

```
Moving towards making the io_wq per ring per task, so we can't really
share it between rings. Which is fine, since we've now dropped some
of that fat from it.

Retain compatibility with how attaching works, so that any attempt to
attach to an fd that doesn't exist, or isn't an io_uring fd, will fail
like it did before.
```

```
-       f = fdget(p->wq_fd);
-       if (!f.file)
-               return -EBADF;

+       f = fdget(p->wq_fd);
+       if (!f.file)
+               return -ENXIO;
```

Change not documented in the commit description

perf_event_open

perf_event_open

- Old Kernel (v5.12) : `E2BIG` (`argument list too long`)
- New Kernel (v5.13): `EINVAL` (`invalid argument`)

perf_event_open

- Old Kernel (v5.12) : `E2BIG` (`argument list too long`)
- New Kernel (v5.13): `EINVAL` (`invalid argument`)

author Marco Elver <elver@google.com> 2021-04-08 12:36:01 +0200

perf: Add support for SIGTRAP on perf events

```
Adds bit perf_event_attr::sigtrap, which can be set to cause events to
send SIGTRAP (with si_code TRAP_PERF) to the task where the event
```

```
+       if (attr->sigtrap && !attr->remove_on_exec)
+               return -EINVAL;
+
```

Fixing sources of nondeterminism

Fixing sources of nondeterminism

- favoring single-threaded mode in program execution
  - avoids a system call failing because a previous one that it depends on hasn’t executed yet
  - e.g. calling `write` before calling `open` on a file descriptor

Fixing sources of nondeterminism

- favoring single-threaded mode in program execution
  - avoids a system call failing because a previous one that it depends on hasn’t executed yet
  - e.g. calling `write` before calling `open` on a file descriptor
- ensure initial state for each executed program is identical
  - avoids false positives occuring because of *accumulated hidden state*

Fixing sources of nondeterminism

- favoring single-threaded mode in program execution
  - avoids a system call failing because a previous one that it depends on hasn't executed yet
  - e.g. calling `write` before calling `open` on a file descriptor
- ensure initial state for each executed program is identical
  - avoids false positives occuring because of *accumulated hidden state*
- rerun programs that returned mismatches
  - eliminates *flaky* mismatches caused by
    - the current state of the system
    - background activity

Next Potential Steps

- research and eliminate other sources of false positives
- automatic bisection
- extending the return state of each system call to include information about
    - memory
    - registers
    - contents of disk
    - privileges assigned to system call
- comparing Linux with other kernels (e.g. *BSD, gVisor) on a subset of syscalls
- creating a model of the Linux kernel to compare against

Summary

- differential fuzzing automates the process of finding semantic bugs
- `syz-verifier` is a differential fuzzing prototype for the Linux kernel
- repository and documentation: https://github.com/google/syzkaller/blob/master/docs/syz_verifier.md