KIT: Testing OS-Level Virtualization for Functional Interference Bugs

Congyu Liu, Sishuai Gong, Pedro Fonseca

Purdue University
Containers are widely deployed

In 2022,

79% of organizations use containers in production, with 44% using them for most or all production applications.
Containers efficiently and securely share the same kernel
Kernel is responsible for isolating containers

Kernel isolates resources for containers
This ‘wall’ has holes

Functional interference bug:
A container affects other containers’ system call results (functionality)
Functional interference bugs are harmful

Integrity violation (data corruption) or cause denial of service
Functional interference bugs are harmful
A functional interference bug in Linux namespaces

CVE-2021-38209

Network sysctls should be isolated by namespaces
A functional interference bug in Linux namespaces

CVE-2021-38209

sysctl.net.nf_conntrack_max

Firewall connection tracking table limit

A container can easily affect firewall of other containers
A functional interference bug in Linux namespaces

CVE-2021-38209

Root cause: share the same global variable
Functional interference bugs are usually **semantic bugs**

**Semantic bug**

- Do not cause crashes
- Do not involve memory errors or data races

```
fd = open()
write(fd, "100")
```

```
fds = open()
read(fds)
```
Challenge: detect semantic functional interference bugs

Semantic bug

Sanitizers

Crash checker

Static analysis

Bug missed

False positives
Goal: check system call results for **correct isolation**

- Detect semantic bug
- Simplify analysis
Observation: affected results usually change

```
fd = open("/proc/sys/")
write(fd, "100")
read(fd)
```

```
fd = open("/proc/sys/")
write(fd, "100")
read(fd)
```

```
open("/proc/sys/") = 4
read(4, "100") = 3
```

```
open("/proc/sys/") = 4
read(4, "65536") = 6
```

Trace collection

Kernel

Net sysctl

Trace collection

Kernel

Net sysctl
Approach: compare system call trace across executions
Exception I: Non-deterministic system call results

Non-determinism can also cause traces to be different and should be filtered
Filter non-deterministic system call results

Non-determinism can be identified by running multiple times
Filter non-deterministic system call results

Trace text

\texttt{fstat( )}

\texttt{fstat( )}

Trace analysis

Different!

No need to report it
Exception II: non-isolated kernel resources

Some resources are deliberately not isolated (e.g., some files in /proc)

System calls on non-isolated kernel resources can interfere

Expected behavior, not bugs
Filter system call results on non-isolated kernel resources

1. Container configuration

2. Isolated resource specification

Limit accesses to non-isolated resources

Progressively refine the specification to check the isolation on given resources
How to detect functional interference bugs?
Next: trigger functional interference bugs

How to detect functional interference bugs?  
How to generate effective test cases?
Next: trigger functional interference bugs

How to detect functional interference bugs?

How to generate effective test cases?
What makes a test case effective?

Functional interference bugs require kernel data flows
Generates test cases with inter-container kernel data flows

Input: Programs

Predict inter-container kernel data flows

Output: Program pairs

How?
Predict by analyzing kernel memory traces

1. Profile kernel memory address accessed from each program
2. Predict kernel inter-container data flow if two programs access same memory
Implementation

KIT is implemented in 8K LOC

We wrote a (partial) namespace specification

Support clustering similar test reports to reduce analysis effort
Evaluation: bug detection

KIT found 9 bugs in Linux kernel 5.13

Disabling network fast path, leaking network statistics, resource contention…
Kit found 2X bugs than random while using 1/8 test cases.
Evaluation: result filtering and report clustering effectiveness

Filtering and clustering significantly reduce analysis effort

Only 4 are false positives
**Conclusion**

OS-level virtualization suffers from functional interference bugs

KIT detects functional interference bugs by analyzing syscall traces

KIT finds new functional interference bugs in Linux namespaces

KIT artifact: [https://github.com/rssys/kit-artifact](https://github.com/rssys/kit-artifact)