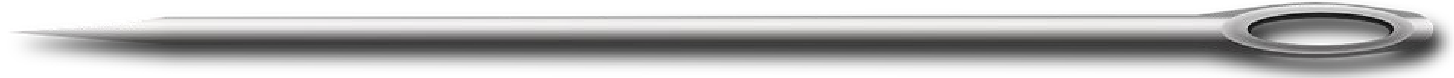
# Fuzzing AOSP For the Masses

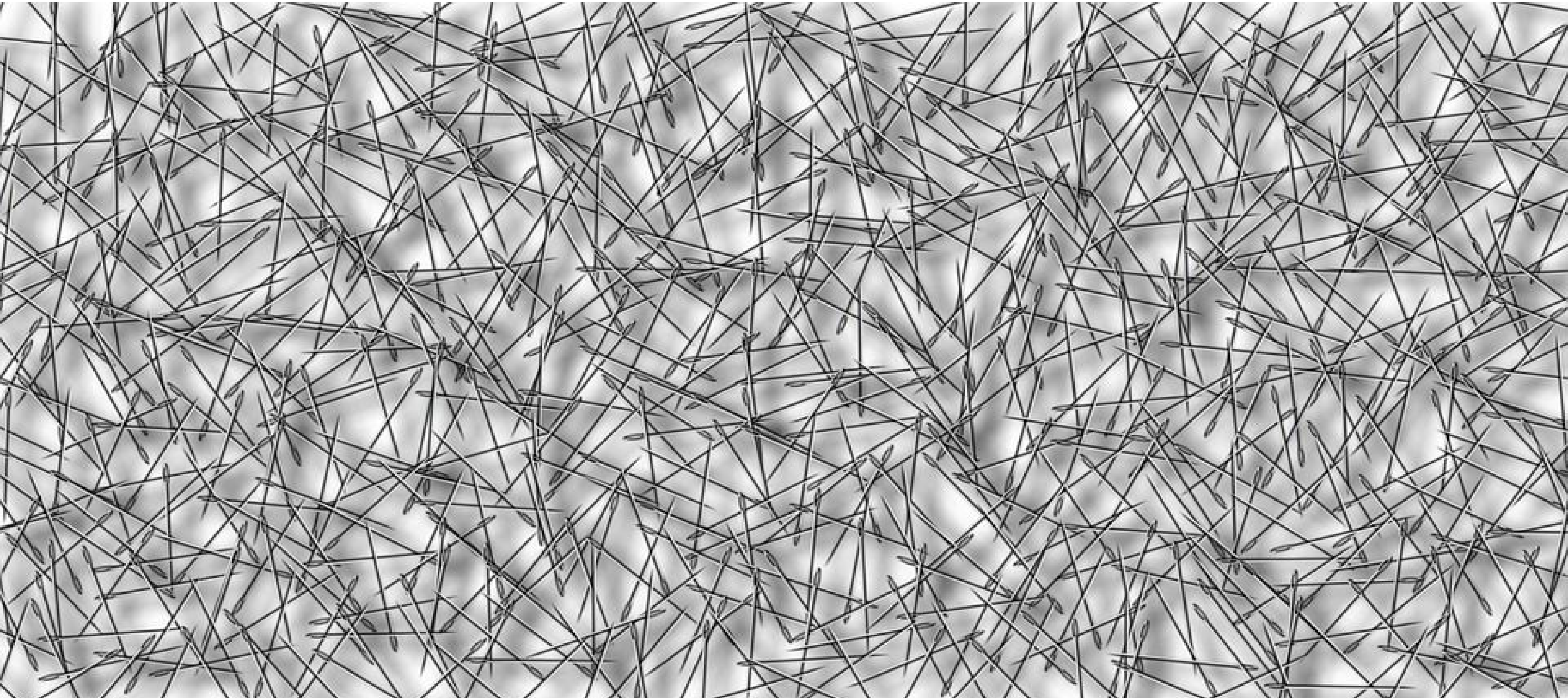
Dan Austin
Google
Android SDL Research Team

## Exploitation: Find the Needle

## Needles are Interesting



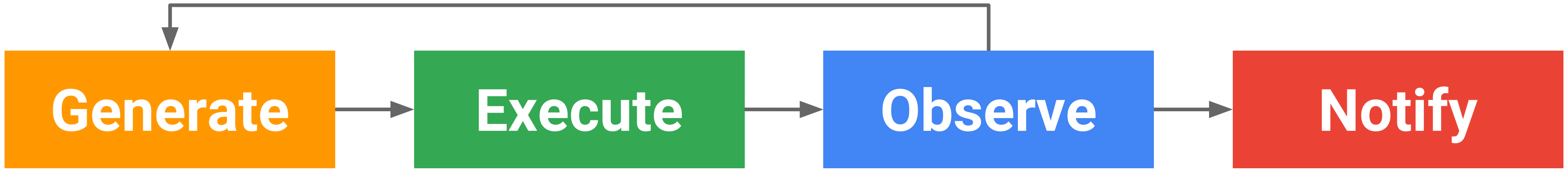
## We’d like to find needles at scale



## How can we do this?

## Fuzzing

## Fuzzing: What is it?



## Should I fuzz?

## Why should I fuzz?

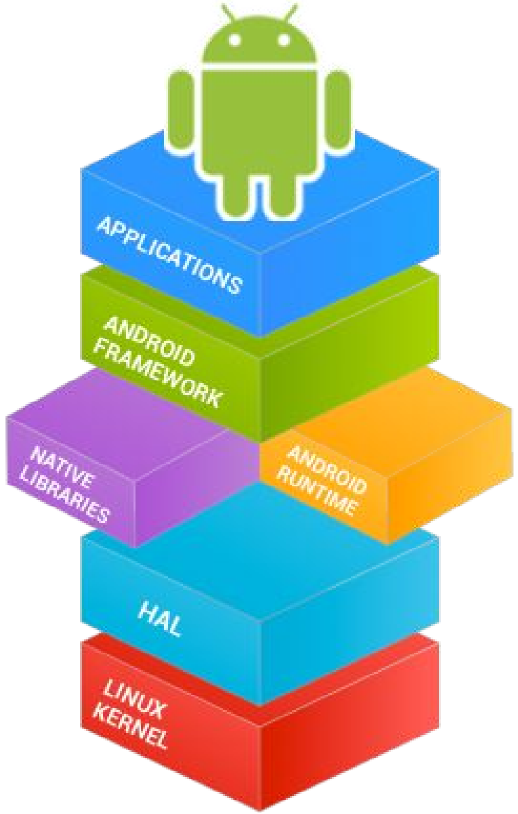
- Ensures edge cases and unexpected input are properly handled
- Increases program robustness & code quality
- Tests for regressions
    - Fuzz-test to generate inputs that result in program crash
    - Leverage these inputs with future iterations of the program
- Low investment method to test complex systems

Android is a complex system.

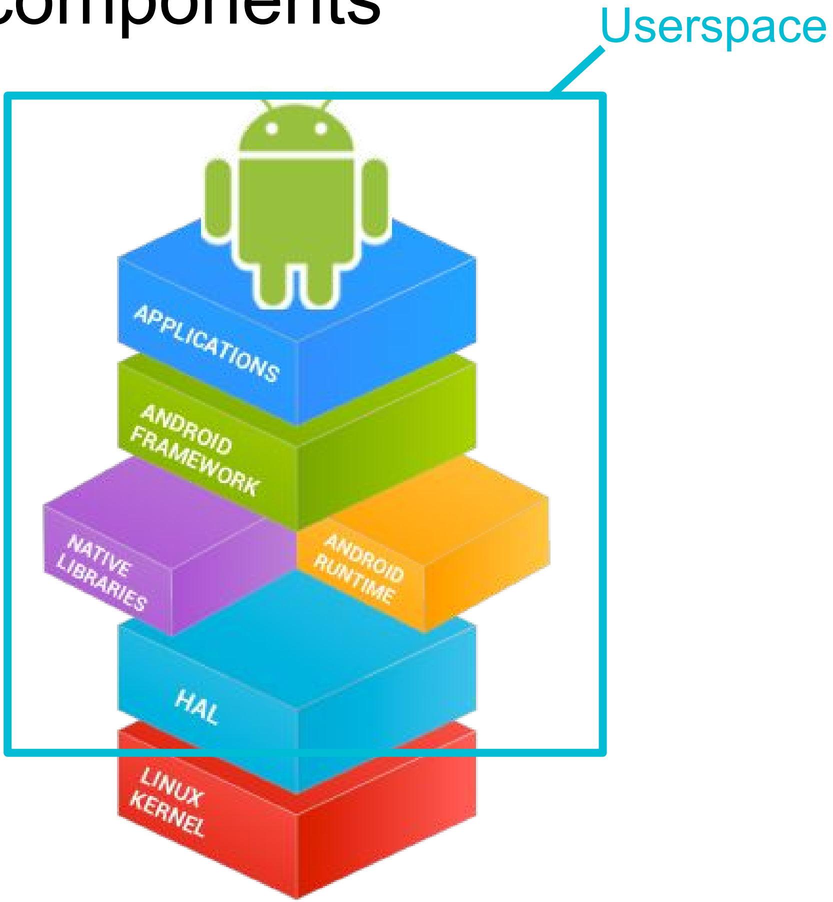
Complex systems have bugs.

Bugs could result in security vulnerabilities.

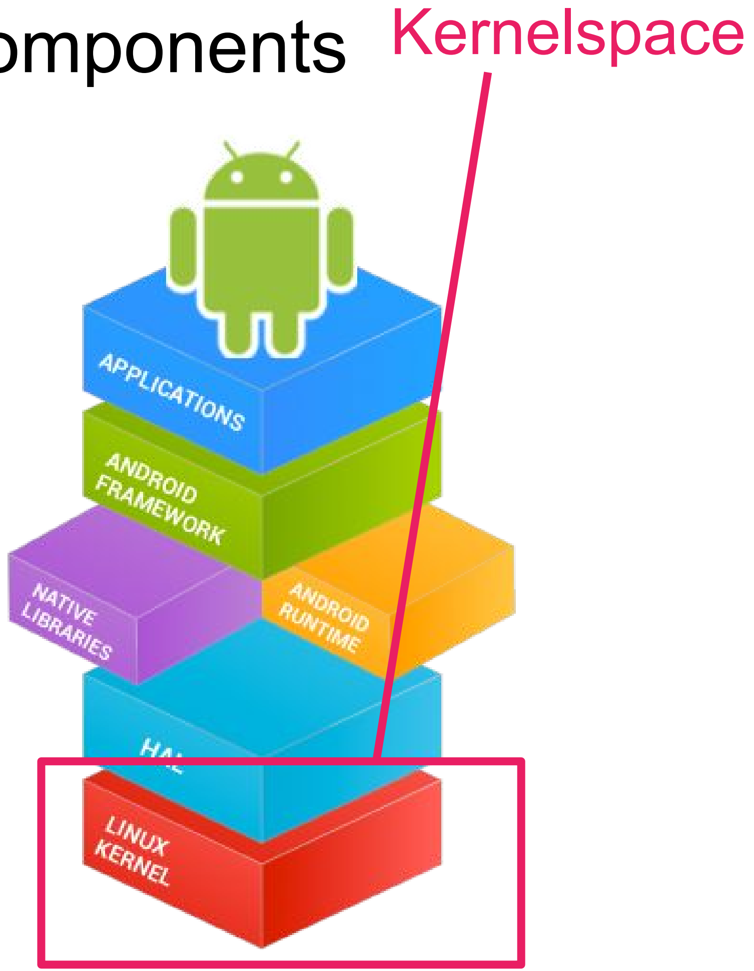
## Android: Lots of components



## Android: Lots of components

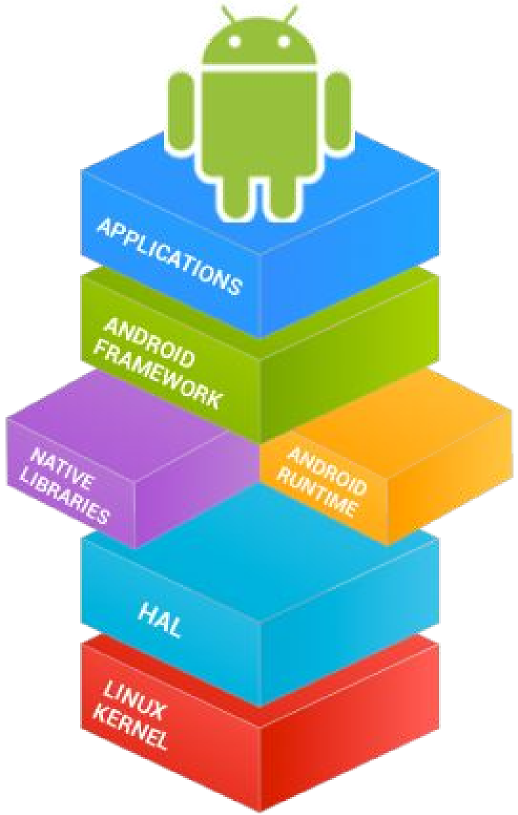


## Android: Lots of components



## Android: Lots of components

And it's all fuzzable!



## Android: Lots to focus on

Where do we start?

- Remotely accessible
  - Media (audio/video)
  - Parsing code (XML, etc.)
  - Fonts
  - WiFi/Bluetooth/Radio
- Allows for privesc or sandbox escape
  - Graphics
  - Kernel/Drivers
  - Firmware Interfaces
- Rarely executed == less likely to be tested

What would be a convenient place to search?

## Android Open Source Project

Easier fuzzing with source-level tools

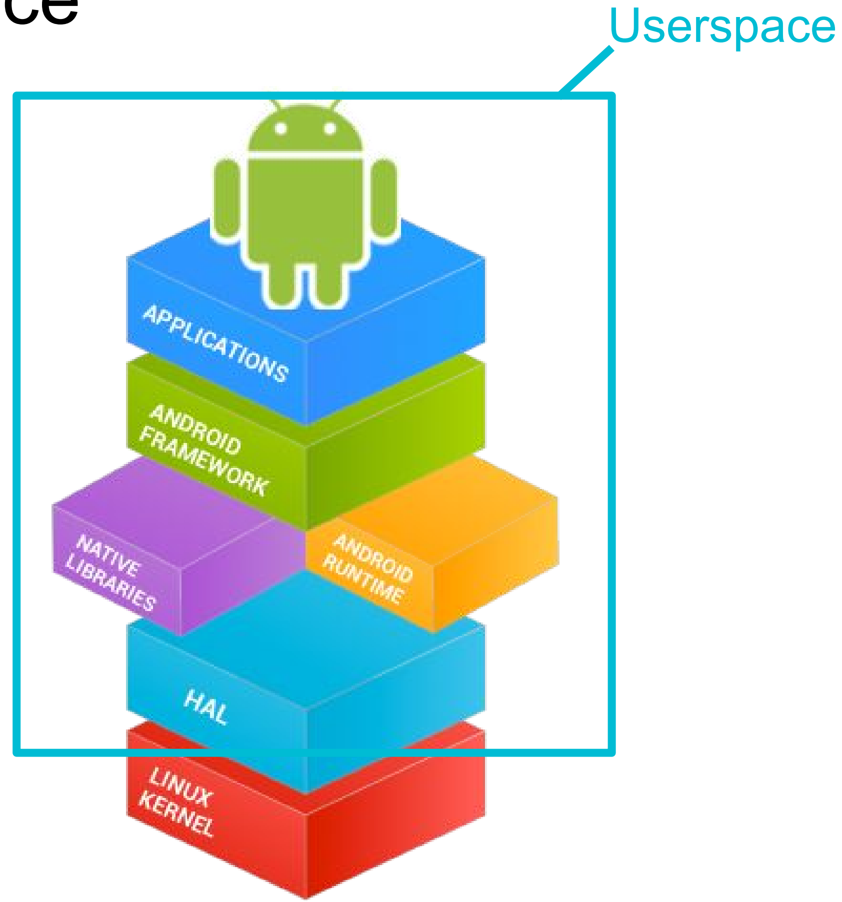
We provide the tools!

Bugs found are likely work on other targets

Fuzz once, test everywhere!

## Fuzzing Android

## Fuzzing Userspace



## Fuzzing userspace: Sanitizers

LLVM Compile time tools allow for efficient dynamic analysis.

Two sanitizers currently supported in Android that can aid with fuzzing

- AddressSanitizer
  - source.android.com/devices/tech/debug/asan
- SanitizerCoverage
  - clang.llvm.org/docs/SanitizerCoverage.html

## AddressSanitizer (ASAN)

Fast memory error detector

Two parts:

- Compiler instrumentation
- Run-time library

ASAN can detect:

- Out-of-bounds accesses to heap, stack and globals
- Use-after-free
- Use-after-return (runtime flag ASAN_OPTIONS=detect_stack_use_after_return=1)
- Use-after-scope (clang flag -fsanitize-address-use-after-scope)
- Double-free, invalid free
- Memory leaks (experimental)

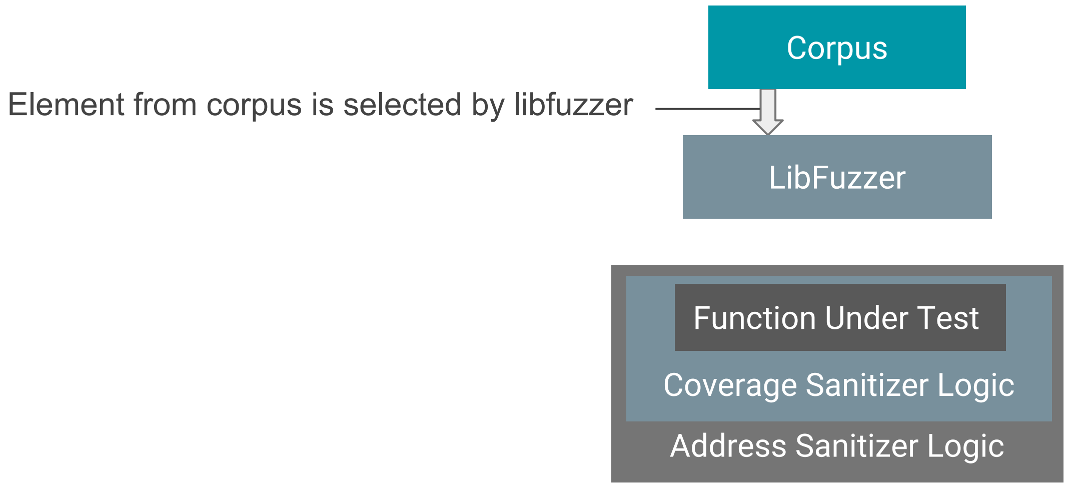
## SanitizerCoverage

- Allows for simple code coverage instrumentation
- Two parts:
  - Compiler instrumentation
  - Run-time library
- Inserts calls to user-definable functions at each
  - function
  - basic-block
  - edge
- Can provide coverage reporting and visualization
- And be used to guide fuzzing sessions!

## Fuzzing userspace: libFuzzer

- In-process, in-memory fuzzing library
- Allows for coverage-guided fuzzing
- Function-level, tends to be faster than traditional fuzzing
- Fuzzers are unit-test friendly
- And easy to write!

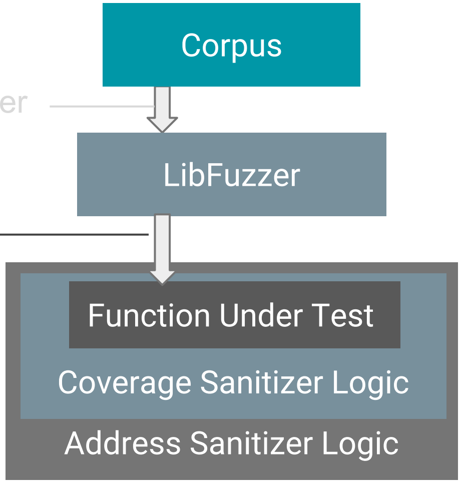
# Sanitizers & LibFuzzer walkthrough



# Sanitizers & LibFuzzer walkthrough

Element from corpus is selected by libfuzzer

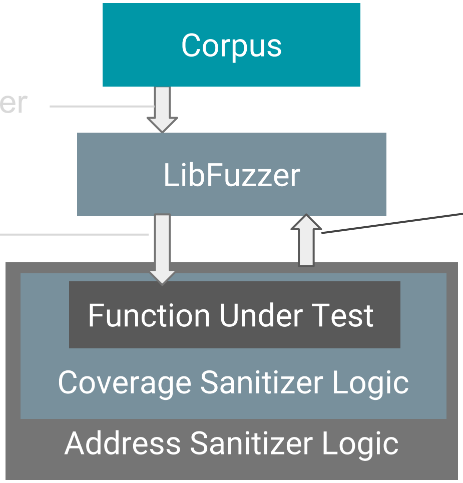
That element is mutated/truncated by libfuzzer and passed to the function under test.



## Sanitizers & LibFuzzer walkthrough

Element from corpus is selected by libfuzzer

That element is mutated/truncated by libfuzzer and passed to the function under test.

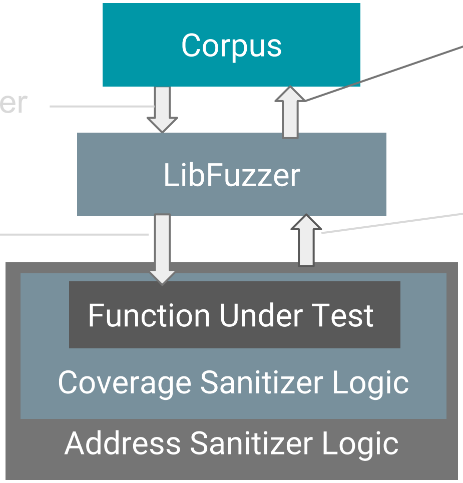


The provided input did not cause a crash. Use information from Coverage Sanitizer to determine if a new path was discovered.

# Sanitizers & LibFuzzer walkthrough

Element from corpus is selected by libfuzzer

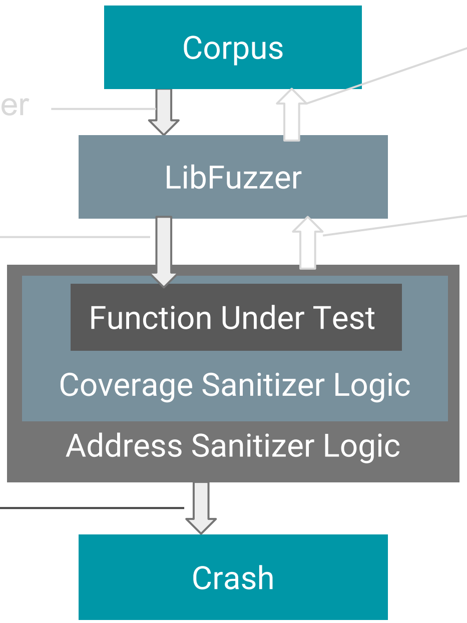
That element is mutated/truncated by libfuzzer and passed to the function under test.



LibFuzzer processes the input and sends it back for inclusion into the corpus. Input selection & execution continues.

The provided input did not cause a crash. Use information from Coverage Sanitizer to determine if a new path was discovered.

## Sanitizers & LibFuzzer walkthrough: Crash!



Element from corpus is selected by libfuzzer

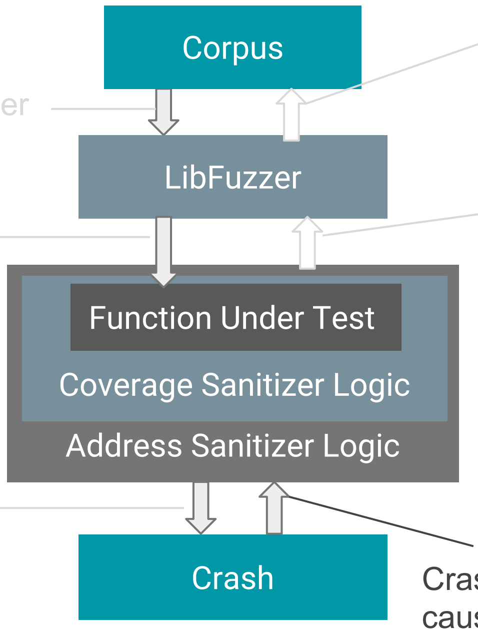
That element is mutated/truncated by libfuzzer and passed to the function under test.

LibFuzzer processes the input and sends it back for inclusion into the corpus. Input selection & execution continues.

The provided input did not cause a crash. Use information from Coverage Sanitizer to determine if a new path was discovered.

The provided input caused a crash!

# Sanitizers & LibFuzzer walkthrough: Crash!



Element from corpus is selected by libfuzzer

That element is mutated/truncated by libfuzzer and passed to the function under test.

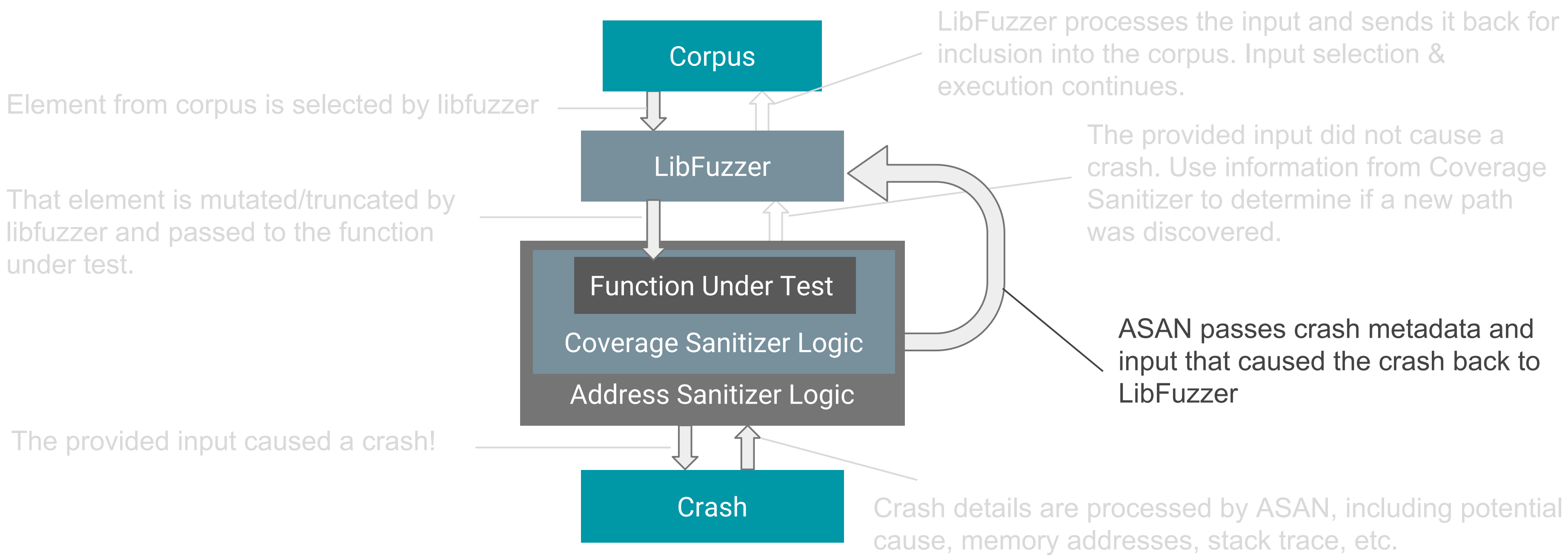
The provided input caused a crash!

LibFuzzer processes the input and sends it back for inclusion into the corpus. Input selection & execution continues.

The provided input did not cause a crash. Use information from Coverage Sanitizer to determine if a new path was discovered.

Crash details are processed by ASAN, including potential cause, memory addresses, stack trace, etc.

# Sanitizers & LibFuzzer walkthrough: Crash!



LibFuzzer processes the input and sends it back for inclusion into the corpus. Input selection & execution continues.

Element from corpus is selected by libfuzzer

The provided input did not cause a crash. Use information from Coverage Sanitizer to determine if a new path was discovered.

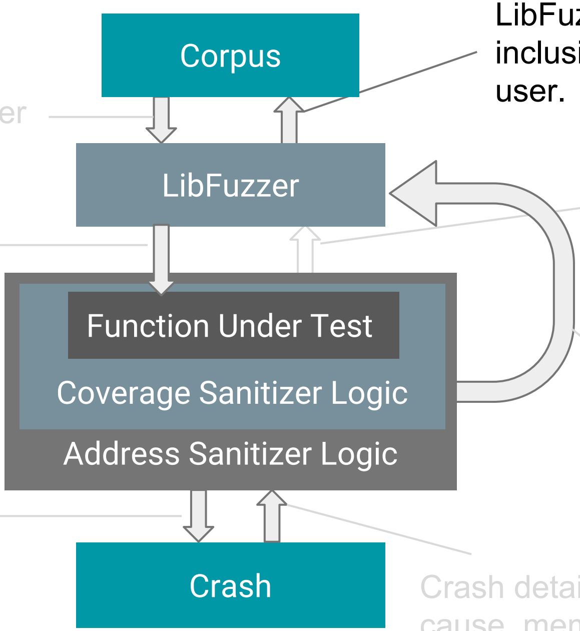
That element is mutated/truncated by libfuzzer and passed to the function under test.

ASAN passes crash metadata and input that caused the crash back to LibFuzzer

The provided input caused a crash!

Crash details are processed by ASAN, including potential cause, memory addresses, stack trace, etc.

# Sanitizers & LibFuzzer walkthrough: Crash!



LibFuzzer processes the input and sends it back for inclusion into the corpus. Crash metadata passed to user.

Element from corpus is selected by libfuzzer

That element is mutated/truncated by libfuzzer and passed to the function under test.

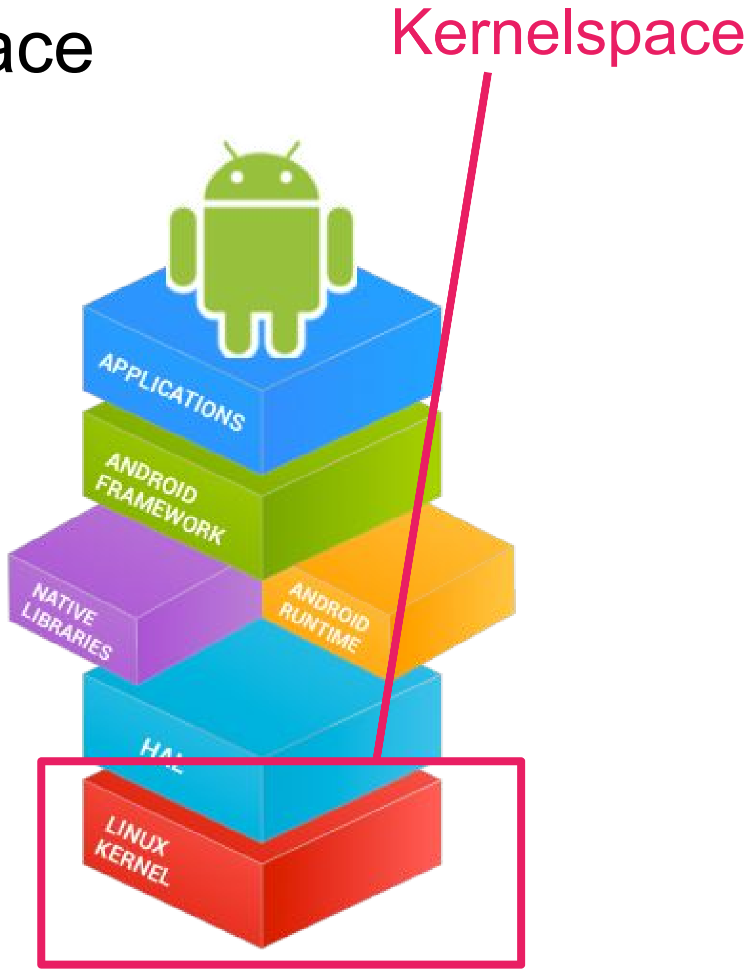
The provided input did not cause a crash. Use information from Coverage Sanitizer to determine if a new path was discovered.

ASAN passes crash metadata and input that caused the crash back to LibFuzzer

The provided input caused a crash!

Crash details are processed by ASAN, including potential cause, memory addresses, stack trace, etc.

# Fuzzing Kernelspace



## Fuzzing Kernelspace: KASAN

TL;DR: ASAN in the Linux kernel

Dynamic memory error detector capable of discovering:

- Use after free
- Out of bounds access

Implemented using:

- Compile time modifications (gcc 4.9.2 or later)
- Custom memory handling (Shadow memory)

Enabled with CONFIG_KASAN & CONFIG_KASAN_INLINE on Android kernels

## Fuzzing Kernelspace: KCOV

- TL;DR: SanitizerCoverage in the Kernel
- Allows for simple code coverage instrumentation
- Basic-block level instrumentation
- Enabled with CONFIG_KCOV
- Implemented with kernel debugfs extension that collects and exposes coverage per-thread

## Fuzzing Kernelspace: syzkaller

Coverage guided Linux syscall fuzzer

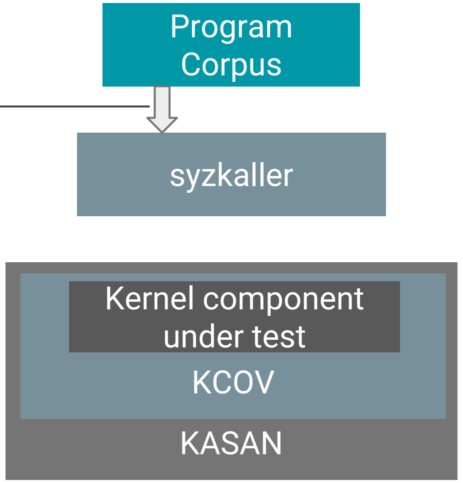
Supported in android on pixel devices

Requires a kernel with KASAN and KCOV enabled

Uses syscall descriptions to generate “programs” that correspond to fuzzing inputs

## syzkaller walkthrough

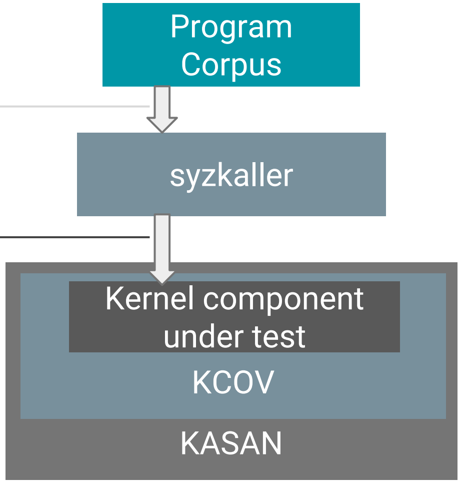
Program from corpus is selected by syzkaller



# syzkaller walkthrough

Program from corpus is selected by syzkaller

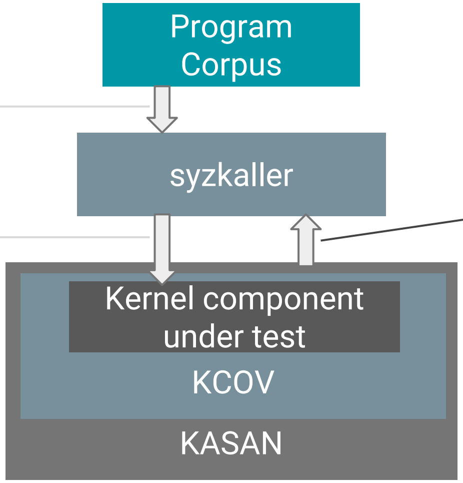
That element is mutated/ arguments modified by syzkaller and runs in the modified kernel.



# syzkaller walkthrough

Program from corpus is selected by syzkaller

That element is mutated/ arguments modified by syzkaller and runs in the modified kernel.



The provided input did not cause a crash. Use information from KCOV to determine if a syscall was executed that discovered a new path.

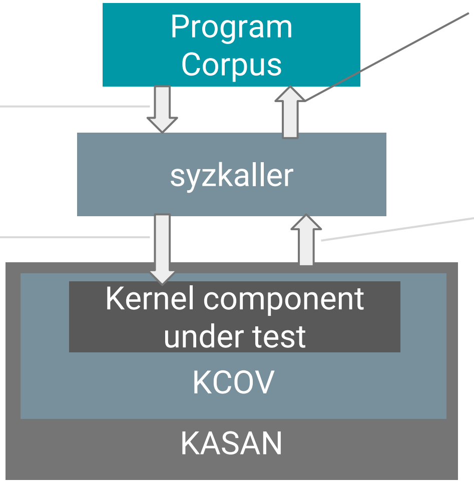
# syzkaller walkthrough

A new path was discovered! syzkaller minimizes the program and adds it to the corpus.

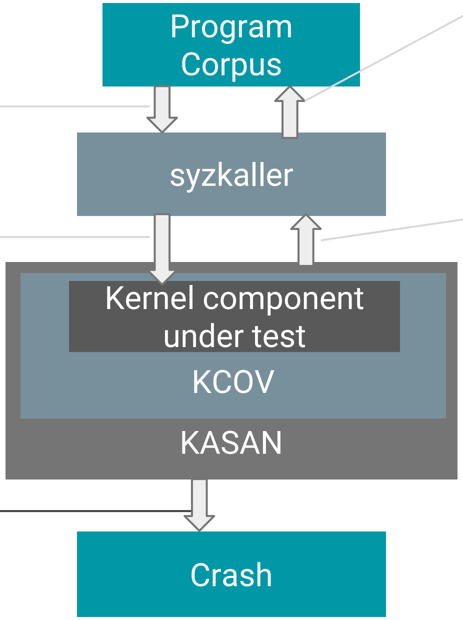
Program from corpus is selected by syzkaller

That element is mutated/ arguments modified by syzkaller and runs in the modified kernel.

The provided input did not cause a crash. Use information from KCOV to determine if a syscall was executed that discovered a new path.



# syzkaller walkthrough: crash!



A new path was discovered! syzkaller minimizes the program and adds it to the corpus.

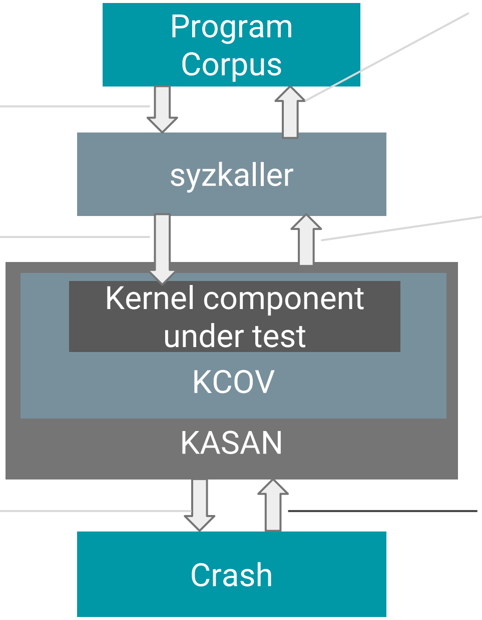
Element from corpus is selected by syzkaller

The provided input did not cause a crash. Use information from KCOV to determine if a syscall was executed that discovered a new path.

That element is mutated/ arguments modified by syzkaller and run in the modified kernel.

The program caused a crash!

# syzkaller walkthrough: crash!



A new path was discovered! syzkaller minimizes the program and adds it to the corpus.

Element from corpus is selected by syzkaller

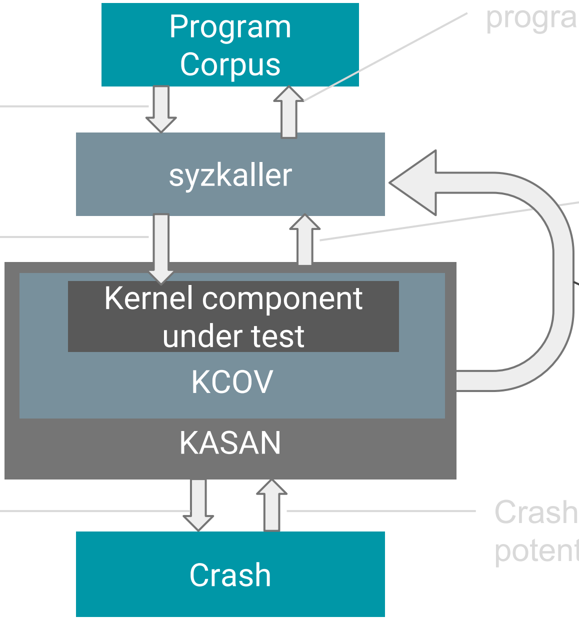
The provided input did not cause a crash. Use information from KCOV to determine if a syscall was executed that discovered a new path.

That element is mutated/ arguments modified by syzkaller and run in the modified kernel.

The program caused a crash!

Crash details are processed by KASAN, including potential cause, memory addresses, stack trace, etc.

# syzkaller walkthrough: crash!



A new path was discovered! syzkaller minimizes the program and adds it to the corpus.

Element from corpus is selected by syzkaller

The provided input did not cause a crash. Use information from KCOV to determine if a syscall was executed that discovered a new path.

That element is mutated/ arguments modified by syzkaller and run in the modified kernel.

syzkaller reads KASAN provided crash metadata and input program

The program caused a crash!

Crash details are processed by KASAN, including potential cause, memory addresses, stack trace, etc.

# syzkaller walkthrough: crash!

syzkaller processes the input and sends it back for inclusion into the corpus. Crash metadata passed to user via syzkaller interface.

Element from corpus is selected by syzkaller

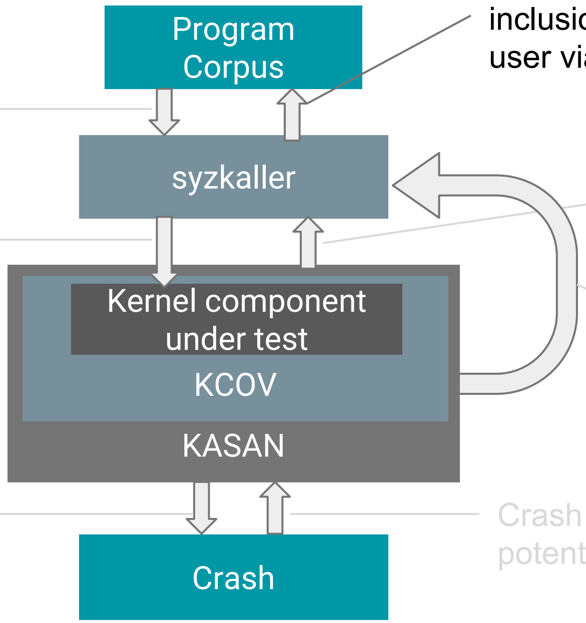
That element is mutated/ arguments modified by syzkaller and run in the modified kernel.

The provided input did not cause a crash. Use information from KCOV to determine if a syscall was executed that discovered a new path.

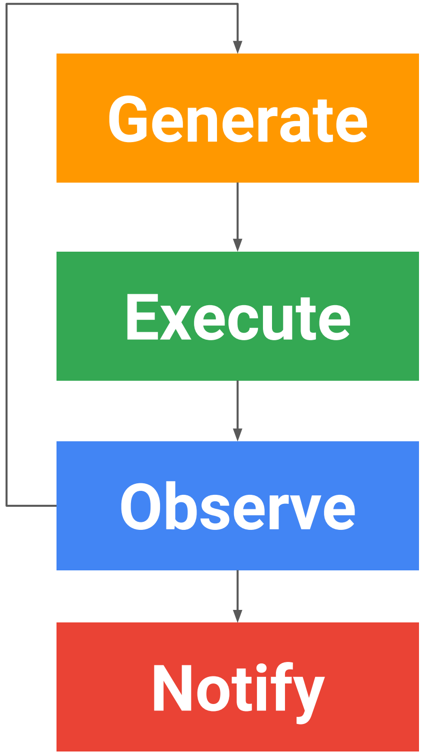
syzkaller reads KASAN provided crash metadata and input program

The program caused a crash!

Crash details are processed by KASAN, including potential cause, memory addresses, stack trace, etc.



## Repeatable and organized fuzzing



## Repeatable and organized fuzzing: Tradefed

Continuous test framework integrated into Android

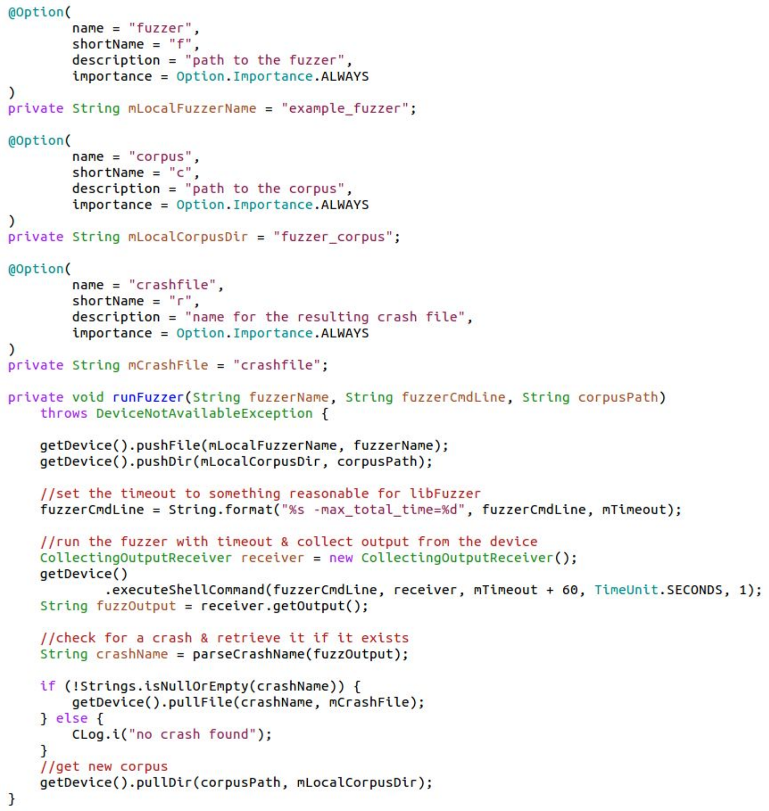
Basically, Java classes + adb

Built in support for different types of tests

Supports test scheduling, parallelizable tests

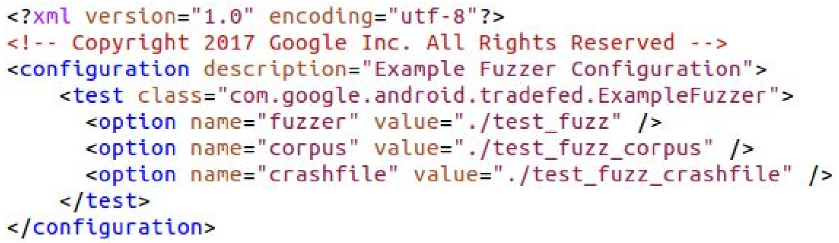
Also handles device recovery

## Repeatable and organized fuzzing: Test Harness



```java
@Option(
        name = "fuzzer",
        shortName = "f",
        description = "path to the fuzzer",
        importance = Option.Importance.ALWAYS
)
private String mLocalFuzzerName = "example_fuzzer";

@Option(
        name = "corpus",
        shortName = "c",
        description = "path to the corpus",
        importance = Option.Importance.ALWAYS
)
private String mLocalCorpusDir = "fuzzer_corpus";

@Option(
        name = "crashfile",
        shortName = "r",
        description = "name for the resulting crash file",
        importance = Option.Importance.ALWAYS
)
private String mCrashFile = "crashfile";

private void runFuzzer(String fuzzerName, String fuzzerCmdLine, String corpusPath)
    throws DeviceNotAvailableException {

    getDevice().pushFile(mLocalFuzzerName, fuzzerName);
    getDevice().pushDir(mLocalCorpusDir, corpusPath);

    //set the timeout to something reasonable for libFuzzer
    fuzzerCmdLine = String.format("%s -max_total_time=%d", fuzzerCmdLine, mTimeout);

    //run the fuzzer with timeout & collect output from the device
    CollectingOutputReceiver receiver = new CollectingOutputReceiver();
    getDevice()
            .executeShellCommand(fuzzerCmdLine, receiver, mTimeout + 60, TimeUnit.SECONDS, 1);
    String fuzzOutput = receiver.getOutput();

    //check for a crash & retrieve it if it exists
    String crashName = parseCrashName(fuzzOutput);

    if (!Strings.isNullOrEmpty(crashName)) {
        getDevice().pullFile(crashName, mCrashFile);
    } else {
        CLog.i("no crash found");
    }
    //get new corpus
    getDevice().pullDir(corpusPath, mLocalCorpusDir);
}
```

+



```xml
<?xml version="1.0" encoding="utf-8"?>
<!-- Copyright 2017 Google Inc. All Rights Reserved -->
<configuration description="Example Fuzzer Configuration">
    <test class="com.google.android.tradefed.ExampleFuzzer">
      <option name="fuzzer" value="./test_fuzz" />
      <option name="corpus" value="./test_fuzz_corpus" />
      <option name="crashfile" value="./test_fuzz_crashfile" />
    </test>
</configuration>
```

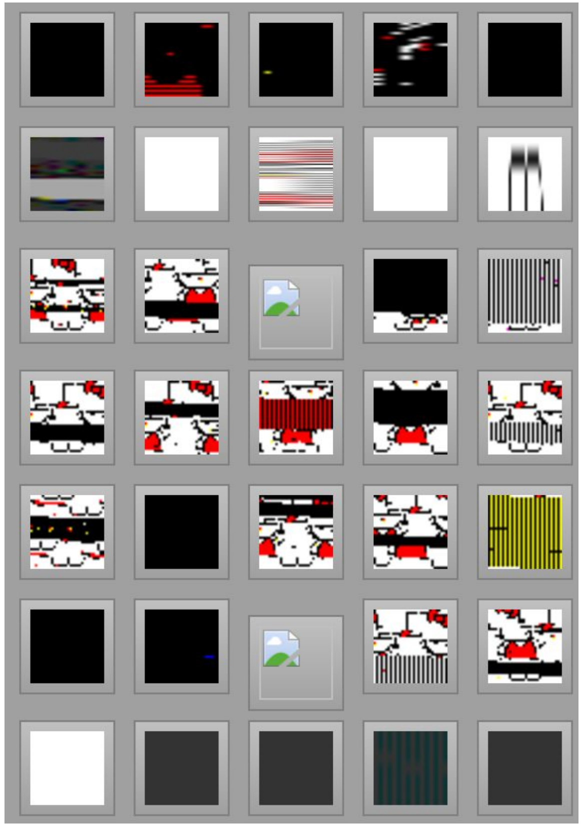
# Repeatable and organized fuzzing: Corpora

Fuzzers need seed inputs

New paths correspond to new inputs

Multiple inputs can correspond to the same path

libFuzzer can keep corpus size reasonable



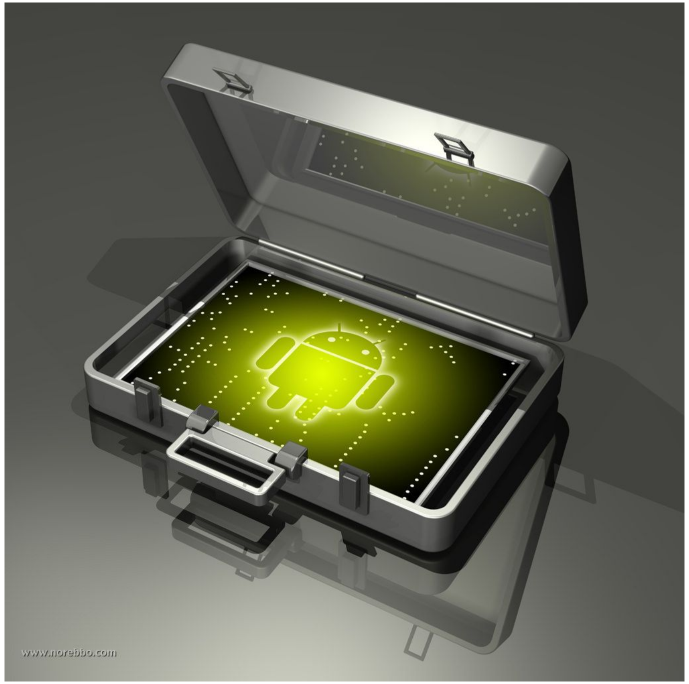
## Repeatable and organized fuzzing: Preparation

What to gather:

- Device specifics
- Android Version Information
- Fuzzer
- Offending input(s)
- Crash information

Test with TF for automation & simple reproduction

Package & send our way!



## Android Vulnerability Reward Program

Android recognizes contributions of security researchers

and we provide monetary rewards!

For submission details:

> sites.google.com/site/bughunteruniversity/improve/how-to-submit-an-android-platform-bug-report

Rules and Pricing information:

> www.google.com/about/appsecurity/android-rewards

## Keep on fuzzing

Adding new fuzzing engines!

New fuzzing techniques!

Better kernel support!

## References

source.android.com/devices/tech/debug/asan

clang.llvm.org/docs/SanitizerCoverage.html

source.android.com/devices/tech/debug/sanitizers

llvm.org/docs/LibFuzzer.html

source.android.com/devices/tech/debug/kasan-kcov

github.com/google/syzkaller

source.android.com/devices/tech/test_infra/tradefed/

sites.google.com/site/bughunteruniversity/improve/how-to-submit-an-android-platform-bug-report

www.google.com/about/appsecurity/android-rewards

Happy fuzzing!

Questions?