All Flutter Engine builds can now enable a combination of the following sanitizers. The different sanitizers are used to isolate specific classes of construction issues.

- **Thread Sanitizer**: Detects data races.
- **Address Sanitizer**: Detects memory errors like use-after-free, buffer overflows...
- **Memory Sanitizer**: Detects reads on uninitialized memory.
- **Undefined Behavior Sanitizer**: Detects undefined behavior.
- **Leak Sanitizer**: Detects memory leaks.

While the buildroot is wired up to attempt sanitizer enabled builds for all targets, not all target architectures, host platform and sanitizer combinations are supported. Support for each sanitizer also varies greatly with each toolchain version. For this reason, and due to general constraints in available build/test infrastructure, these sanitizers are not enabled by default on presubmits or build-bots. It is best to work backwards from a problem and choose a sanitizer that has the best shot of isolating the same.

General Usage Guidelines

Most sanitizer enabled variants will either fail to build at all or fail when running specific unit-test targets. These failures are either because there are false positives that have not been annotated correctly or genuine issues in either the Flutter Engine or Dart VM. However, it is possible to discover further issues without fixing them all. This is done by selectively suppressing known issues. The suppressions and other sanitizer options have to be specified as environment options. These options can be specified in one shot by invoking the following in the current shell:

```
source ./flutter/testing/sanitizer_suppressions.sh
```

This should enable relevant options for all known sanitizers and also print the files containing suppressions for each sanitizer.

Sample output:

```
Using Thread Sanitizer suppressions in ./flutter/testing/tsan_suppressions.txt
Using Leak Sanitizer suppressions in ./flutter/testing/lsan_suppressions.txt
```

You can run the specific unit-test binaries in the terminal and all sanitizer options and suppressions will take hold.

If an issue is raised that is not previously known, a suppression may be added and a bug filed to track the same. Instructions on how to add suppressions for the specific sanitizer are specific at the top of the relevant suppressions file.

Make sure to add suppression that are as detailed as possible. Adding suppressions that are too broad may suppress errors from issues that are actually new and untracked. If you find that the issue you expect is not caught by the sanitizer, make sure to check the list of suppressions to see if any of the suppressed issues look relevant. If so, you might need to selectively disable suppressions:

```
-----------------------------------------------------
Suppressions used:
  count      bytes template
      1        120 class_createInstance
      5         80 MakeSkSurfaceFromBackingStore
      3        128 _dispatch_once_callout
-----------------------------------------------------
```

Support for sanitizer enabled builds using Goma is spotty. Disable Goma for more reliable builds till these issues are addressed.

All sanitizer caught issues are tagged with the `sanitizer` label.

Leak Sanitizer

A tool used to detect memory leaks. Official documentation is at https://clang.llvm.org/docs/LeakSanitizer.html. Enable using the `--lsan` GN flag on any target. For best results, use this with unoptimized build variants. The buildroot is configured to use the ideal arguments in this mode. Full documentation on how to add suppressions to this file is at https://github.com/google/sanitizers/wiki/AddressSanitizerLeakSanitizer#suppressions. This tool can be used either in standalone mode or in conjunction with Address Sanitizer.

```
$ ./flutter/tools/gn --runtime-mode debug --lsan --unoptimized --no-goma
$ autoninja -C out/host_debug_unopt
$ source ./flutter/testing/sanitizer_suppressions.sh
$ ./out/host_debug_unopt/embedder_unittests
```

Address Sanitizer

Address sanitizer detects memory errors. Official documentation is at https://github.com/google/sanitizers/wiki/AddressSanitizer. Enable using the `--asan` flag. Enabling address sanitizer also implicitly enables Leak Sanitizer. The `./flutter/testing/sanitizer_suppressions.sh` script enables leak sanitization in Address Sanitizer builds.

```
$ ./flutter/tools/gn --runtime-mode debug --asan --unoptimized --no-goma
$ autoninja -C out/host_debug_unopt
$ source ./flutter/testing/sanitizer_suppressions.sh
$ ./out/host_debug_unopt/embedder_unittests
```

Address sanitizer has spotty support on aarch64 targets. Use it on x64 target for best results.

Undefined Behavior Sanitizer

Catches undefined behavior. Official documentation at https://clang.llvm.org/docs/UndefinedBehaviorSanitizer.html. Enable using the `--ubsan` flag. The `sanitizer_suppressions.sh` file will specify a suppressions files. Classes of errors can be disabled at runtime.

```
$ ./flutter/tools/gn --runtime-mode debug --ubsan --unoptimized --no-goma
$ autoninja -C out/host_debug_unopt
$ source ./flutter/testing/sanitizer_suppressions.sh
$ ./out/host_debug_unopt/embedder_unittests
```

Thread Sanitizer

Catches data races. Official documentation at https://clang.llvm.org/docs/ThreadSanitizer.html.

```
$ ./flutter/tools/gn --runtime-mode debug --tsan --unoptimized --no-goma
$ autoninja -C out/host_debug_unopt
$ source ./flutter/testing/sanitizer_suppressions.sh
$ ./out/host_debug_unopt/embedder_unittests
```

Memory Sanitizer

Detects reads of uninitialized memory. This sanitizer is only available on Linux. Enable using the `--msan` flag.

```
$ ./flutter/tools/gn --runtime-mode debug --msan --unoptimized --no-goma
$ autoninja -C out/host_debug_unopt
$ source ./flutter/testing/sanitizer_suppressions.sh
$ ./out/host_debug_unopt/embedder_unittests
```