Requirements

- Easy for existing fuzzers to adopt:
  - AFL++
  - LibAFL
  - LibFuzzer
- Easy for users to consume
- Support extra fields
- Usable by many types of fuzzers
  - Binary, source, web, network
- Reasonable to write or create by hand

Static Analysis Results Interchange Format (SARIF)

- Existing open specification for a similar need
- Specification in JSON schema
  - Support for YAML and JSON
- Well defined objects for many concepts from static analysis:
  - Code/system locations
  - Flows, stacks, edges, web requests and responses
- Usable for streaming logs and final reports via external property files

Requirements

Requirement	SARIF Provision
Easy for existing fuzzers to adopt	Python, Rust, C# libraries, also supported by any JSON Schema validator
Easy for users to consume	Reasonably readable by hand, very easy for tools to interpret. Many existing tools including GitHub ingest.
Support additional fields	Schema explicitly supports additional "properties" on all objects.
Usable by many types of fuzzers	Resource/code-location design is URI-based, allows for any type of location for code flows
Reasonable to write by hand	Not 100% straightforward, but anecdotally doable (Editor support for schema helps significantly)

Useful Objects: tool

- Specifies a tool, including a driver (e.g. LibFuzzer) and extensions (e.g. ASAN),
- Allows tracking of which fuzzer is used and how
  - e.g. AFL-Clang compiled code fuzzed with AFL++ with QEMU mode, QASAN,
- Rules for a tool can be provided by the tool itself, allows the output producer to be its own source of truth

Useful Objects: invocation

- An invocation is one run of a tool, allows capturing setups like 1 main afl-fuzz instance and 7 secondary afl-fuzz instances
- Each invocation can provide:
  - Start/stop time, user account, command line, working directory, environment variables, stdin/out (not useful for *target* stdin/out, see later)
  - Notifications (like intermediate results, with a timestamp). Useful for things like new coverage, new function, new PC discovered
- Can be externalized to separate files to accommodate streaming logs

Useful objects: result

- Fuzzer findings would be output as "result" objects, and include a rule that was "violated" e.g. crash, ASAN, UBSAN and metadata
- Results can be externalized to separate files to accommodate streaming logs
- Can include information like code flows, crash reason, stack traces, etc.

Proposed Field	SARIF Equivalent
Timestamp (log entry)	runs[].invocations[].toolExecutionNotifications[].timeUtc
Timestamps (start time, stop time, Runtime)	Runs[].invocations[].startTimeUtf, runs[].invocations[].endTimeUtc (includes restarts by considering multiple invocations/runs)
Crash Paths/Crash encoding/Crash Size	Results[].locations[].physicalLocation.artifactLocation Results[].locations[].annotations.sourceLanguage Results[].locations[].annotations.byteLength
Corpus count/size/type (on disk)	Runs[].invocations[].toolExecutionnotifications[].locations (Same as above)
Corpus count/size/type (in memory)	Runs[].invocations[].toolExecutionNotifications[].locations[].logicalLocations or properties
Coverage points/objectives covered	CodeFlow objects
Timeouts	Timeout type results
Last new crash/corpus	Timestamp of tool execution notification
State	ToolConfigurationNotification or ToolExecutionNotification for fuzzer state changes

Small Example

- https://github.com/novafacing/fuzzer-output

Next Steps

- Discussion, community & fuzzer developer team feedback
- Prototype implementation in LibAFL
- External property files for tools and extensions
  - e.g. an ASAN .sarif-external-properties file would standardize the rules ASAN has and how its outputs are represented
- Discussion of whether any desired fuzzing properties are not represented in the SARIF specification and should be RFC-ed
- More implementations and PRs!