LLVM and the state of sanitizers on BSD

Speaker : David Carlier
Software engineer living in Ireland, contribute to various opensource projects directly or indirectly related to FreeBSD and OpenBSD mainly, from enterprise solutions to more entertaining ones like video games, contributor LLVM since end of 2017, committer since May 2018.
Write time to time for BSDMag.

Status on FreeBSD and OpenBSD

How it had started ?

- It often starts from “frustration” :-).
- Indeed, after having tried fuzzer under Linux, I realized it was not supported under FreeBSD.
- After this came Xray instrumentation and MemorySanitizer.
- Somewhere in between, started to port UndefinedBehaviorSanitizer, libFuzzer and Xray as well..
- After getting into enough people's nerves, you get commit access.

Where are we ?

FreeBSD	OpenBSD
asan	ubsan
safestack	libFuzzer
msan	Xray instrumentation
libFuzzer	Cannot have asan/msan/tsan ASLR cannot be disabled Cannot map large regions (shadow memory)
Xray instrumentation	
ubsan	

What is fuzzing all about ?

- It is a testing technique, “invented” in late 80'a by Barton Miller, when basically you try to give random data to your software and its dependencies included.
- Inputs source come from what call “corpus”
- It is good to find particular set of bugs, based on input handling basically while trying to cover as much as possible code paths by mutating these inputs.
- Ideally running long, as the data will undergo some “mutation” in the process, as necessary until it crashes eventually.
- Completes traditional unit tests set too.

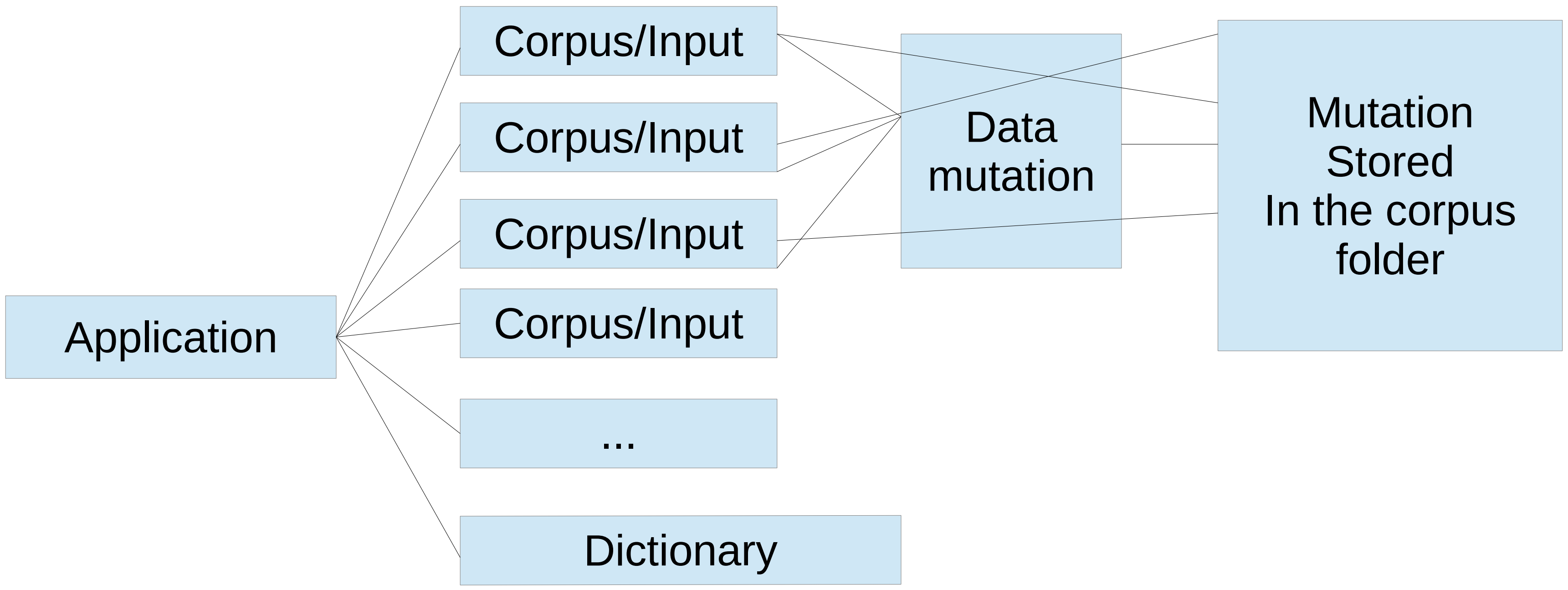
What corpus means ?

- Place holder for inputs, inputs which suit the particular software.
- Let's imagine an image library reader which relies on specific binary format header to recognise if it is png/jpeg and so on.
- The corpus will then contain hand crafted data for a particular test.
- There is possibly more than one corpus but those corpus could possibly be merged, keeping only the relevant mutation results.
- Those mutations will be then stored (and to be reused) in this corpus.

No worries libfuzzer is not radioactive :-)

- Not yet at least
- Mutation simply means some bytes are deleted, some others are inserted, got shuffled at random offsets. A dictionary (format key=value) can be used too.
- A dictionary is to guide the mutation when the target software’s vocabulary is complex.
- Software developers might start to sweat ... That would trigger a segfault, stack overflows or SIGBUS throwing.
- Well ... that s the whole point ;-)

Fuzzer workflow



How does it works under LLVM

- Basic flag is -fsanitize=fuzzer to gives to either the C or C++ frontend.
- The code in question needs to have a specific entry point at minimum to receive the input data, main is already present.
- You can also customize the mutation’s part.
- Can be combined with another sanitizer flag as ubsan, asan, msan, even lsan ...
- Once the binary built, has plethora of options.
- Test cases which crashes the code are stored into crash-* files.

We said options ?

- A fuzzed library run each time the whole process, we can limit the number of runs. Limits the max length of the incoming inputs. How long the tests runs.
- Can be ran with parallel jobs and workers with a specific logging for each.
- Enable/disable certain signal interruption handlings.
- Limit the memory usage.
- Control the degree of mutation.
- Merge N corpus into one.

X-Ray instrumentation

- Is a run time call tracing facility. Mainly made for function timing measurements. Mainly maintained by Dean Berris.
- Can be refined by explicitly tracing or not tracing certain functions via clang attributes, configuration files or at least by function thresholds.
- Can be enabled/disabled at runtime.
- When disabled, the performance overhead is usually non existent but has a more noticeable performance difference when enabled but somehow suited to be ran in production.
- But usually only to be run for a certain time and for a subset of functions in order to collect enough data dependent also on the function threshold and the memory usage limit wish for the in memory buffer data collection.

How does it work

- Xray injects instrumentation hooks at function entry and exits.
- Empty hooks until xray is enabled so as runtime thus replaced by cycle counter, function identifier, thread id, base address metadata.
- The basic flag is -fxray-instrument
- Our binaries contains now xray_instr_map and xray_fn_idx sections.
- We need means to extract those data from the Elf binary to generate our call graphs => llvm-xray
- Can generate call graphs. Can generate tracing format for Chrome.
- Accounting is also a feature to display where the code spends most of the time.
- There is logging options settable via XRAY_OPTIONS => tracing from beginning to end (patch_premain), verbosity, mode ...

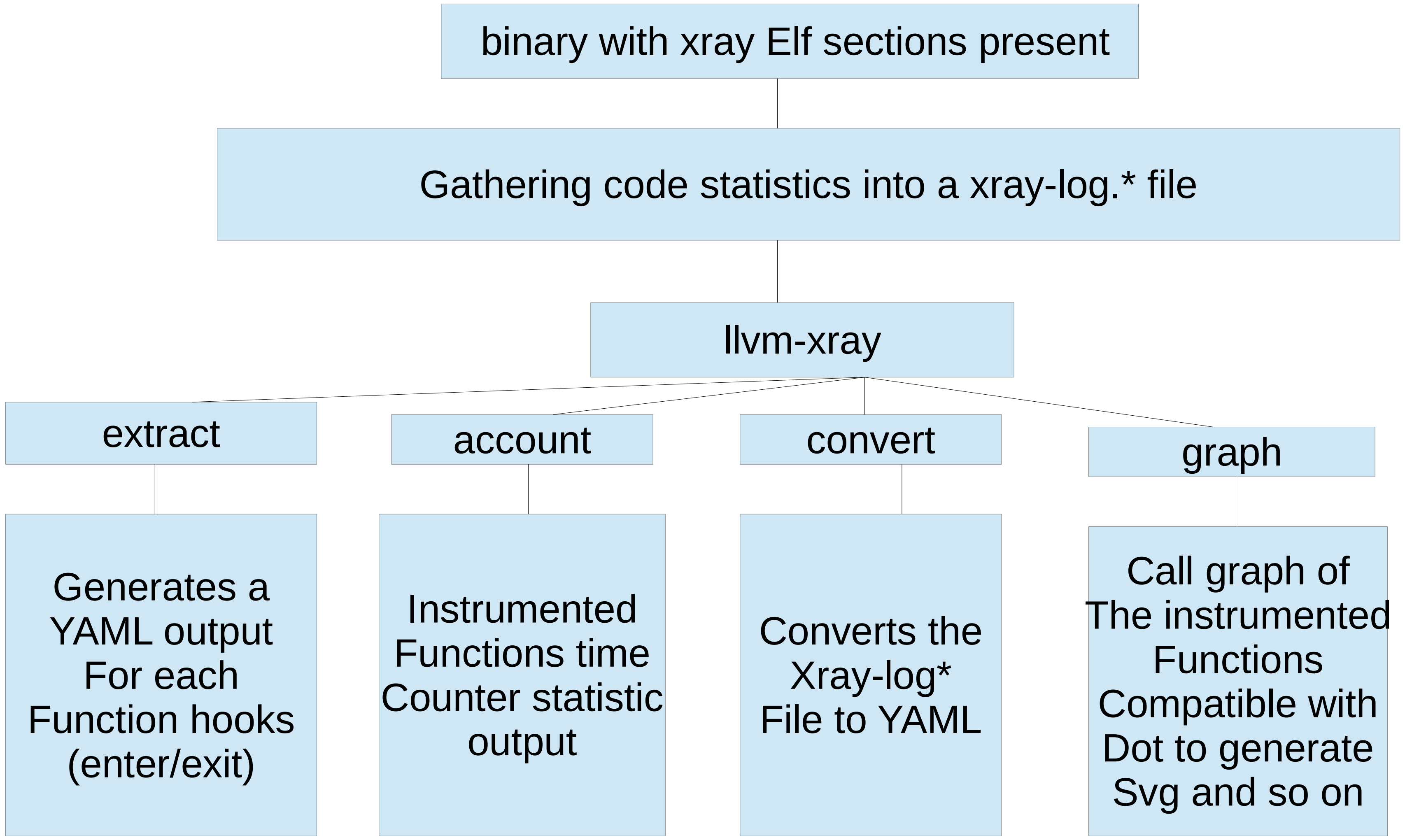
Accounting is not what you think !

- ... But Is more about to display the code used and the cumulative time spent for each.
- If it is a multithread program, data can possibly be aggregated.
- Can be sorted by any column, formatted as csv
- Can give a good idea of the possible bottlenecks.

Modes ?

- Comes with basic mode (ie generating the xray-log* files) and a more advanced one called FDR (Flight Data Recorder).
- FDR allows to, programmatically, trace a precise amount of data basically by adding the start point of recording and the flushing end point.
- Llvm-xray supports for sure both modes, output formats will differ.
- For now the basic mode is the most reliable.

Xray workflow


binary with xray Elf sections present
Gathering code statistics into a xray-log.* file
llvm-xray
extract
account
convert
graph
Generates a YAML output For each Function hooks (enter/exit)
Instrumented Functions time Counter statistic output
Converts the Xray-log* File to YAML
Call graph of The instrumented Functions Compatible with Dot to generate Svg and so on

Other features and ongoing/future work

- LibFuzzer mutation/coverage increase statistics (ongoing).
- A new (optional) basic W^X detection had been added and available for most of sanitizers (asan, msan, tsan...).
- Similar feature had been added in the code verification at compile time (aka scan-build toolsuite).
- Porting CFI to FreeBSD/NetBSD (in review).
- Despite the build is supported under FreeBSD, lsan is not workable and not doable yet (needs to be able to suspend the thread (aka Stop the world)).