Improving bpftrace reliability

Daniel Xu

Agenda

- Philosophy
- Techniques
- Current focus

What is reliability?

- Abstract and context dependent
- For bpftrace, consider this:
  - You have a problem
  - You use bpftrace to troubleshoot/debug it
  - You don’t want a second problem

No second problems

- Clear error if something is not possible
- Principle of least surprise
- Misleading data is the worst outcome

Challenges

- Complex intersection between compiler, language design, kernel, and BPF
  - LLVM API changes often
  - LLVM IR is subtle and tricky to learn
  - Language needs to be suitable for casual use yet powerful enough to develop tools
  - Kernel is tough environment, lots of gotchas (faulting memory, NMIs, etc.)
  - Kernel internals frequently change
  - BPF rapidly evolves, need to keep up in order to reap benefits
- Open source environment
  - No hiring powers (eg. no full time QA team possible)
  - Friendliness to new contributors (to reap bottom up innovation)
  - Have to work with what you got (Github)

No silver bullet

- **Integer assignment types don't get matched to the variable type** bug reliability
  #3415 by jordalgo was closed 5 days ago
- **Failed LLVM Assertion with Nested For-Loops Using Variable Context** bug priority: medium reliability
  #3307 by ajor was closed on Jul 16 v0.22
- **misaligned stack access off (0x0; 0x0)+0+-23 size 8** bug reliability
  #3294 by mtijanic was closed on Jul 8
- **min/max aggregations are broken** bug reliability
  #3286 by jordalgo was closed on Jul 29
- **Attaching to non-existing uprobes fails** bug priority: high reliability
  #3235 by viktormalik was closed on Jun 20
- **btf_type_tag attributes cause problems with member dereferencing** bug reliability
  #3221 by tyroguru was closed on Jun 18
- **Crash when assigning a record type to map** bug reliability
  #3218 by danobi was closed on Jun 5
- **Crash when looping over map containing avg()** bug reliability
  #3216 by danobi was closed on Jun 28
- **Data corruption when using `printf` and/or `if` on an associative-array map** bug reliability
  #3194 by dkogan was closed on Jun 6
- **Assigning string literals to variables of different size doesn't clear old data** bug priority: high reliability
  #3172 by tnovak was closed on May 17
- **Logging errors results in abnormal termination (abort)** bug reliability
  #3163 by ajor was closed on May 23 v0.21.0

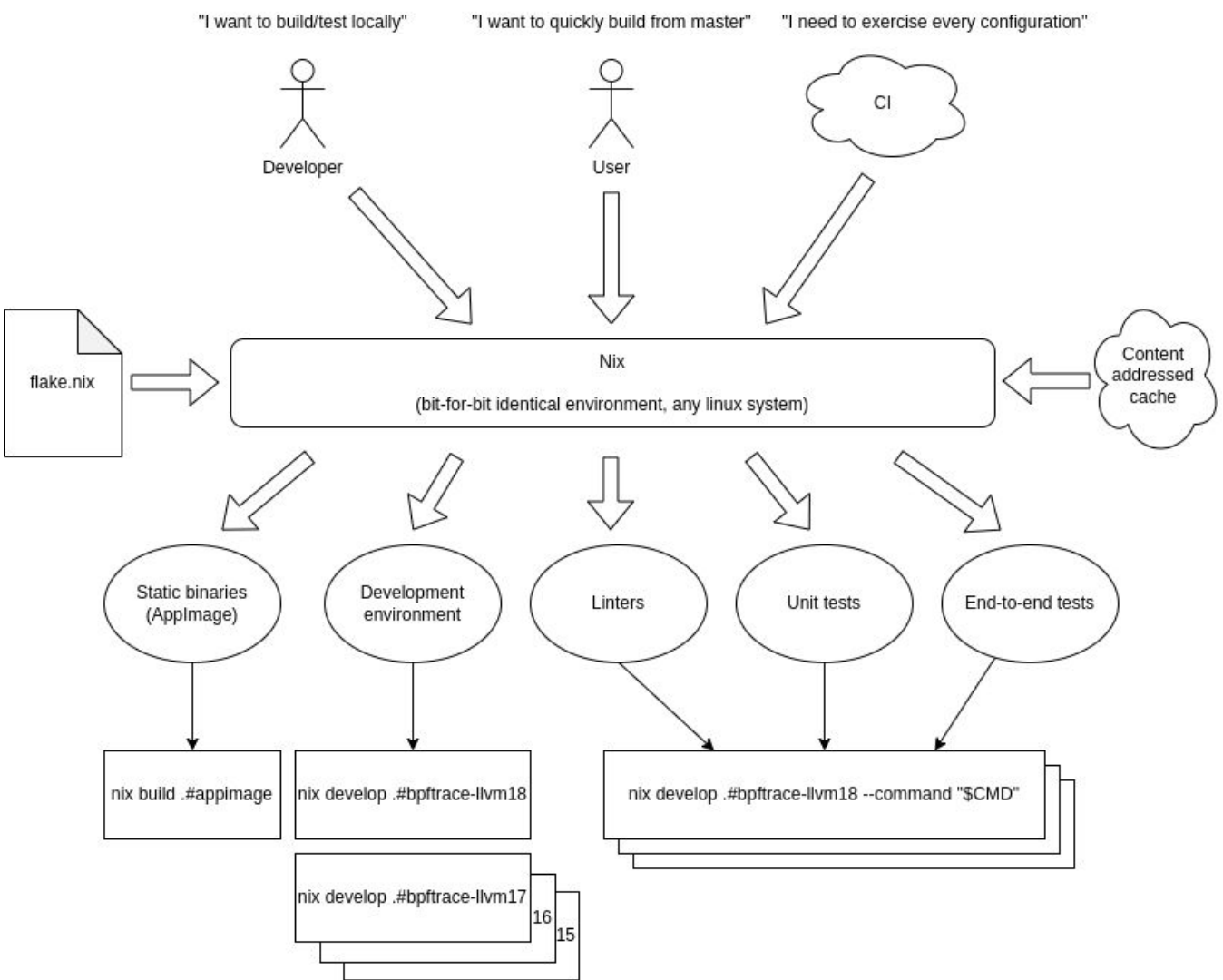
Our solution

- Holistic approach
- Heavy investment in CI
- Why?
  - Lever on entire project
  - Automated feedback for contributors
  - Increases development velocity
    - Speeds up reviews - maintainers can focus on code review
    - Refactor / cleanup with confidence
    - Main branch always release ready
  - Can run matrix of configurations (LLVM, kernel, compiler, etc.)
  - Mechanism to test almost all possible behavior (in our domain)
  - Can codify learned lessons

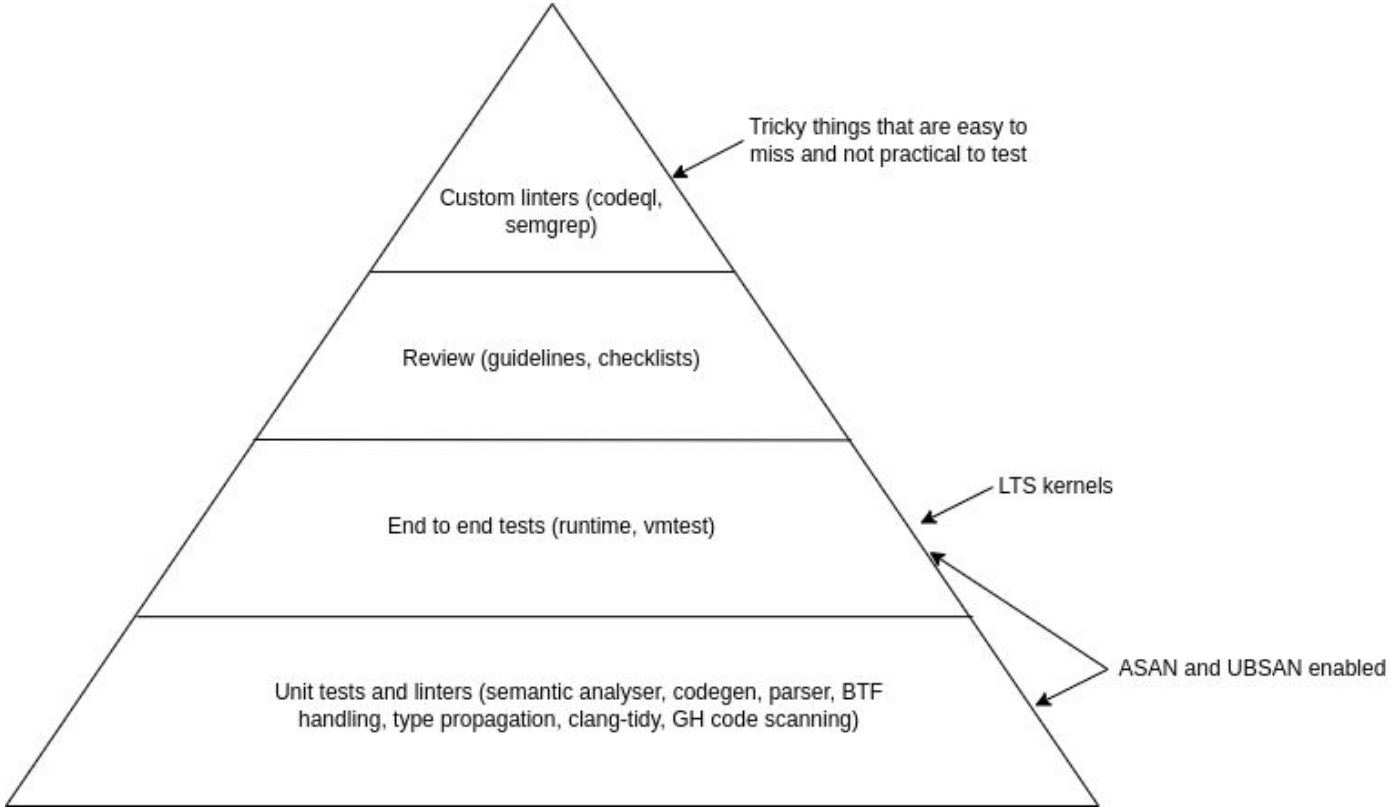
Typical challenges with CI

- Slow
- Non-reproducible
- Flakiness
- Requires highly specific environment

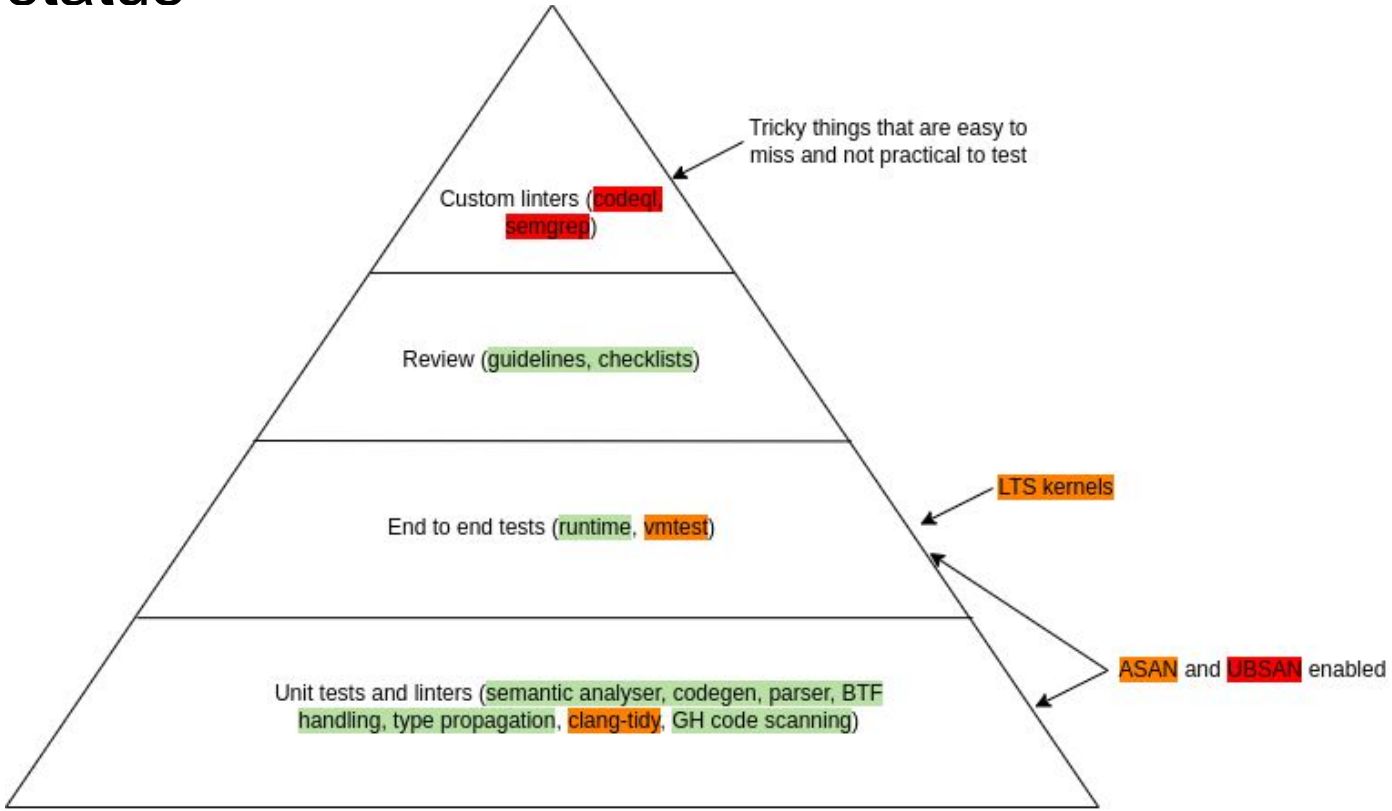
Nix-based CI


"I want to build/test locally"
"I want to quickly build from master"
"I need to exercise every configuration"
Developer
User
CI
flake.nix
Nix
(bit-for-bit identical environment, any linux system)
Content addressed cache
Static binaries (AppImage)
Development environment
Linters
Unit tests
End-to-end tests
nix build .#appimage
nix develop .#bpftrace-llvm18
nix develop .#bpftrace-llvm18 --command "$CMD"
nix develop .#bpftrace-llvm17
16
15

Testing


Tricky things that are easy to miss and not practical to test
Custom linters (codeql, semgrep)
Review (guidelines, checklists)
LTS kernels
End to end tests (runtime, vmtest)
ASAN and UBSAN enabled
Unit tests and linters (semantic analyser, codegen, parser, BTF handling, type propagation, clang-tidy, GH code scanning)

Testing status

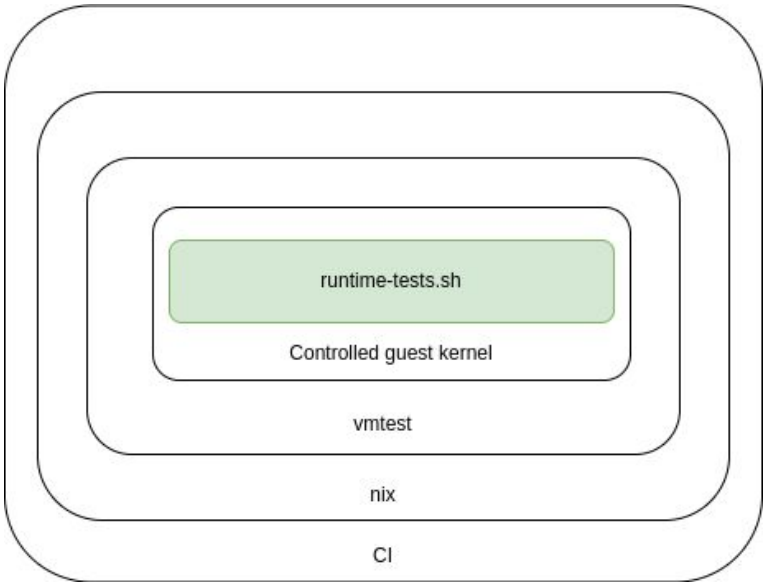


By the numbers

- ~560 unit tests
- ~663 runtime tests
- ~16 clang-tidy lints
- ~174 GH code scanning checks

bpftrace/tests/runtime

```
NAME Implicit truncation of ints
PROG BEGIN{ $a = (int16)0; $a = 2; $b = (int8)3; $b = -100; print(($a, $b)); exit(); }
EXPECT (2, -100)
```



bpftrace/tests/codegen

```
TEST(codegen, bitshift_left)
{
  test("kprobe:f { @x = 1 << 10; }",

       NAME);
}
```

```
; ModuleID = 'bpftrace'
source_filename = "bpftrace"
target datalayout = "e-m:e-p:64:64-i64:64-i128:128-n32:64-S128"
target triple = "bpf-pc-linux"

%"struct map_t" = type { ptr, ptr, ptr, ptr }
%"struct map_t.0" = type { ptr, ptr }
%"struct map_t.1" = type { ptr, ptr, ptr, ptr }

@LICENSE = global [4 x i8] c"GPL\00", section "license"
@AT_x = dso_local global %"struct map_t" zeroinitializer, section ".maps", !dbg !0
@ringbuf = dso_local global %"struct map_t.0" zeroinitializer, section ".maps", !dbg !22
@event_loss_counter = dso_local global %"struct map_t.1" zeroinitializer, section ".maps", !dbg !36

; Function Attrs: nounwind
declare i64 @llvm.bpf.pseudo(i64 %0, i64 %1) #0

define i64 @kprobe_f_1(ptr %0) section "s_kprobe_f_1" !dbg !41 {
entry:
  %"@x_val" = alloca i64, align 8
  %"@x_key" = alloca i64, align 8
  call void @llvm.lifetime.start.p0(i64 -1, ptr %"@x_key")
  store i64 0, ptr %"@x_key", align 8
  call void @llvm.lifetime.start.p0(i64 -1, ptr %"@x_val")
  store i64 1024, ptr %"@x_val", align 8
  %update_elem = call i64 inttoptr (i64 2 to ptr)(ptr @AT_x, ptr %"@x_key", ptr %"@x_val", i64 0)
  call void @llvm.lifetime.end.p0(i64 -1, ptr %"@x_val")
  call void @llvm.lifetime.end.p0(i64 -1, ptr %"@x_key")
  ret i64 0
}

; Function Attrs: nocallback nofree nosync nounwind willreturn memory(argmem: readwrite)
declare void @llvm.lifetime.start.p0(i64 immarg %0, ptr nocapture %1) #1

; Function Attrs: nocallback nofree nosync nounwind willreturn memory(argmem: readwrite)
declare void @llvm.lifetime.end.p0(i64 immarg %0, ptr nocapture %1) #1

attributes #0 = { nounwind }
attributes #1 = { nocallback nofree nosync nounwind willreturn memory(argmem: readwrite) }
```

bpftrace/tests/codegen

```
TEST(codegen, bitshift_left)
{
  test("kprobe:f { @x = 1 << 10; }",

       NAME);
}
```

```
; ModuleID = 'bpftrace'
source_filename = "bpftrace"
target datalayout = "e-m:e-p:64:64-i64:64-i128:128-n32:64-S128"
target triple = "bpf-pc-linux"

%"struct map_t" = type { ptr, ptr, ptr, ptr }
%"struct map_t.0" = type { ptr, ptr }
%"struct map_t.1" = type { ptr, ptr, ptr, ptr }

@LICENSE = global [4 x i8] c"GPL\00", section "license"
@AT_x = dso_local global %"struct map_t" zeroinitializer, section ".maps", !dbg !0
@ringbuf = dso_local global %"struct map_t.0" zeroinitializer, section ".maps", !dbg !22
@event_loss_counter = dso_local global %"struct map_t.1" zeroinitializer, section ".maps", !dbg !36

; Function Attrs: nounwind
declare i64 @llvm.bpf.pseudo(i64 %0, i64 %1) #0

define i64 @kprobe_f_1(ptr %0) section "s_kprobe_f_1" !dbg !41 {
entry:
  %"@x_val" = alloca i64, align 8
  %"@x_key" = alloca i64, align 8
  call void @llvm.lifetime.start.p0(i64 -1, ptr %"@x_key")
  store i64 0, ptr %"@x_key", align 8
  call void @llvm.lifetime.start.p0(i64 -1, ptr %"@x_val")
  store i64 1024, ptr %"@x_val", align 8
  %update_elem = call i64 inttoptr (i64 2 to ptr)(ptr @AT_x, ptr %"@x_key", ptr %"@x_val", i64 0)
  call void @llvm.lifetime.end.p0(i64 -1, ptr %"@x_val")
  call void @llvm.lifetime.end.p0(i64 -1, ptr %"@x_key")
  ret i64 0
}

; Function Attrs: nocallback nofree nosync nounwind willreturn memory(argmem: readwrite)
declare void @llvm.lifetime.start.p0(i64 immarg %0, ptr nocapture %1) #1

; Function Attrs: nocallback nofree nosync nounwind willreturn memory(argmem: readwrite)
declare void @llvm.lifetime.end.p0(i64 immarg %0, ptr nocapture %1) #1

attributes #0 = { nounwind }
attributes #1 = { nocallback nofree nosync nounwind willreturn memory(argmem: readwrite) }
```

CodeQL

```
import cpp

from MemberVariable member
where
  member.getNamespace().getName() = "bpftrace" and
  member.getDeclaringType().getAMemberFunction().getName() = "serialize" and
  not exists(VariableAccess va | va.getEnclosingFunction().getName() = "serialize" | va.getTarget() = member)
select
  member,
  "Member is not being serialized in a serialized class",
  member.getLocation().getFile().getBaseName()
```

```
$ codeql query run --database ~/scratch/codeql/bpftrace-db queries/UnserializedMember.ql
[...]
|      member       |                          col1                          |         col2         |
+-------------------+--------------------------------------------------------+----------------------+
| cgroup_path_args  | Member is not being serialized in a serialized class | required_resources.h |
| skboutput_args_   | Member is not being serialized in a serialized class | required_resources.h |
| helper_error_info | Member is not being serialized in a serialized class | required_resources.h |
| str_buffers       | Member is not being serialized in a serialized class | required_resources.h |
| watchpoint_probes | Member is not being serialized in a serialized class | required_resources.h |
| probes_using_usym | Member is not being serialized in a serialized class | required_resources.h |
| btf_type_tags_    | Member is not being serialized in a serialized class | types.h              |
| ts_mode           | Member is not being serialized in a serialized class | types.h              |
| need_expansion    | Member is not being serialized in a serialized class | types.h              |
| expected_types_   | Member is not being serialized in a serialized class | format_string.h      |
| parts_            | Member is not being serialized in a serialized class | format_string.h      |
| allow_override    | Member is not being serialized in a serialized class | struct.h             |
```

Stack allocations

```
Looks like the BPF stack limit is exceeded. Please move large on
stack variables into BPF per-cpu array map. For non-kernel uses, the
stack can be increased using -mllvm -bpf-stack-size.
```

- Stack is currently precious resource
- Some types sizes scale with work being done (strings)
- Such allocations need to be moved onto percpu scratch map
  - https://github.com/bpftrace/bpftrace/issues/3431

Dropped events

- Probes may not always be safe to run in kernel
  - This is fine - fundamental limitation
- But all missed events _must_ be reported
- https://github.com/bpftrace/bpftrace/issues/835

Map lookup null elisions

- Scratch maps are BPF_MAP_TYPE_PERCPU_ARRAY
  - Currently null checks on lookup are required, even when key is statically known
    - Failure branch just returns out of prog
  - Opportunity for codegen bugs to lose events
- https://lore.kernel.org/bpf/cover.1726458273.git.dxu@dxuuu.xyz/T/#u

Comments/questions