



Interprocedural Symbolic Static Analysis for Go

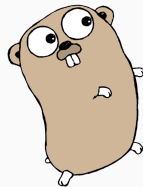
Varvara Dvortsova, Alexey Borodin

Ivannikov Institute for System Programming of the Russian Academy of Sciences

KLEE Workshop 2026
Munich

Why another static analyzer for Go?

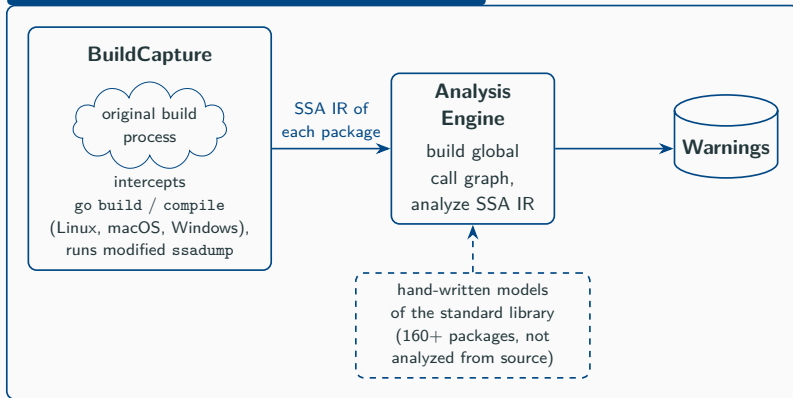
- `go vet`, `staticcheck`, `golanci-lint` and other linters:
 - support only the `go build` command
 - large projects use complex build systems (Bazel, Make)
 - find only simple bugs limited to a single function scope
- They miss bugs that go through several functions, for example:
 - nil dereference several call frames away
 - file descriptor leaked across functions
 - division by zero inside a helper function
 - send on closed channel



Gopher: Renée French, CC BY 4.0

Most mainstream Go analyzers both depend on the build system and have limited interprocedural analysis.

Svace for Go = BuildCapture + Analysis Engine



Svace Analysis Engine was already used for C, C++, Java and other languages.

Our approach: symbolic execution with state merging and attributes

- **Intraprocedural.** Regular forward symbolic execution inside a function: parameters and globals are symbolic variables, other values are expressions over them
- **State merging.** States are joined at CFG join points; if a variable has different values on the incoming paths, its value becomes $ite(\mathbf{cond}, e_1, e_2)$
- **Multiple symbolic storages.** Variables $\rightarrow$ values, values $\rightarrow$ attributes. Attributes store properties of values, e.g. “used as a divisor”
- **Interprocedural.** Bottom-up traversal of call graph: callees are analyzed before their callers. Callee analysis summary is used as transfer function at every call site, to transfer attributes or to check for errors
- **SMT.** Z3 is called only to confirm an error, e.g. when a source meets a sink

- **tuple results:** (value, error) and (value, ok); the value gets the attribute “unchecked”, which is cleared only after the status is checked
- **defer:** LIFO stack of deferred calls with the argument values calculated at the defer instruction; unwound at the function exit
- **map:** an element `m[k]` is a memory location, like a slice element `s[i]`; its address is a symbolic expression: the map with the access step `[k]`. For a map created in the function (`make`) we track the written keys, so any other key gives `nil`
- **closure:** captured variables are handled like globals: their addresses are passed through a hidden struct parameter

TODO

model concurrency (now a goroutine is an ordinary call)

Does it find real-life bugs?

- 8 open-source projects, 24 KLOC (act) to 4 468 KLOC (kubernetes)
- median: build 45 s, analysis 53 s
- precision on manually reviewed warnings:

Caddy 72%, Kubernetes 66%

Project	Status
prometheus/prometheus	merged
prometheus/client_golang	merged
go-chi/chi	merged
go-jose/go-jose	accepted
libvirt-go-module	merged
cheggaaa/pb	merged
caddyserver/caddy	PR pending