

gRPConf

INDIA 2026

gRPConf INDIA 2026

Reachability Across the Wire

Extending static call graphs through gRPC service boundaries

Birajendu Sahu · Senior Engineering Manager, Endor Labs · Bengaluru

It is 6 pm. A critical CVE is in your dependency tree.

gRPConf INDIA 2026

Do you patch tonight?

48,185

CVEs published in 2025 alone — +20% year over year

Jerry Gamblin, 2025 CVE review

65%

of open-source CVEs arrive with no NVD severity score

Sonatype, State of the Software Supply Chain

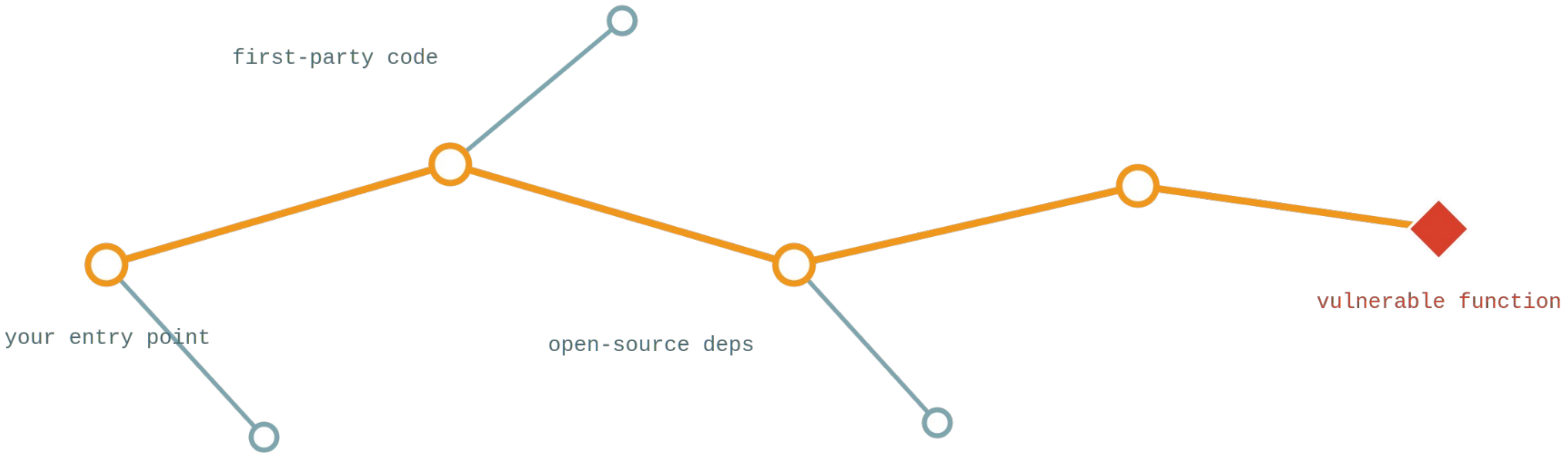
< 9.5%

of dependency findings are exploitable at the function level

Endor Labs, Dependency Management Report

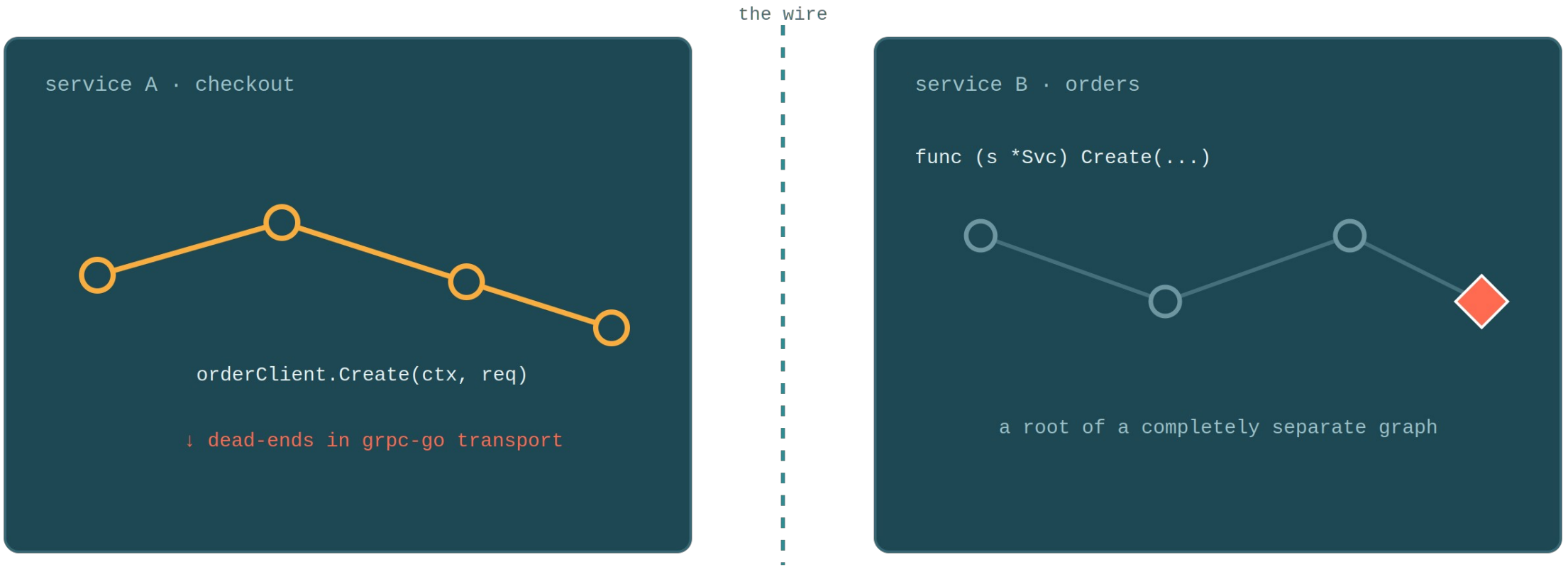
The question that separates signal from noise: **can an attacker actually reach the vulnerable function?**

Reachability in 30 seconds: a path search over a call graph



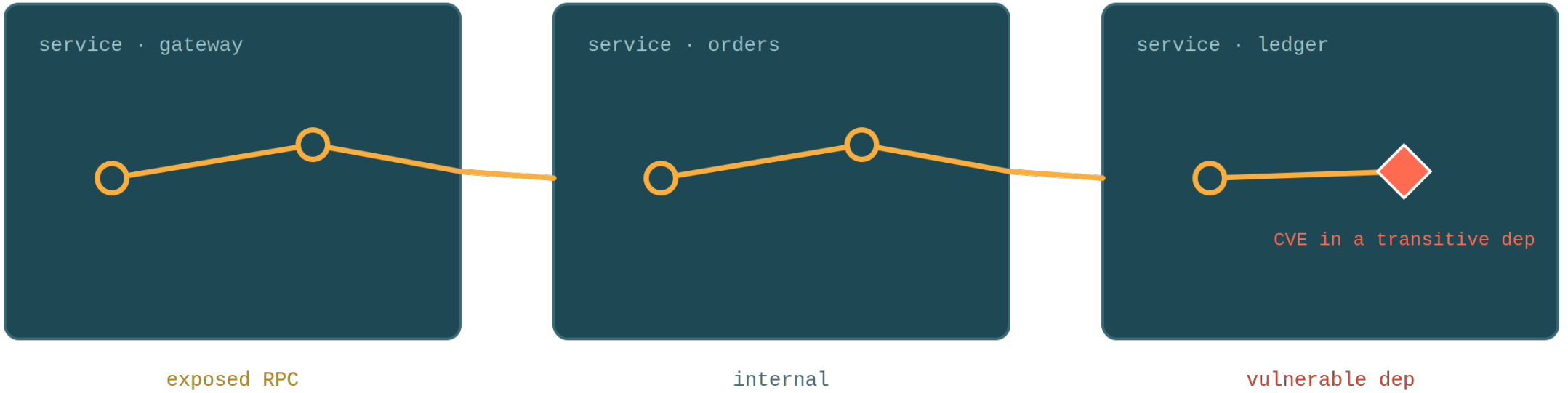
Functions are nodes, calls are edges. CHA and RTA build the graph from compiled code; a vulnerability is “reachable” when a path exists from your entry points to the function the advisory names.

Then the graph hits a gRPC stub — and goes blind



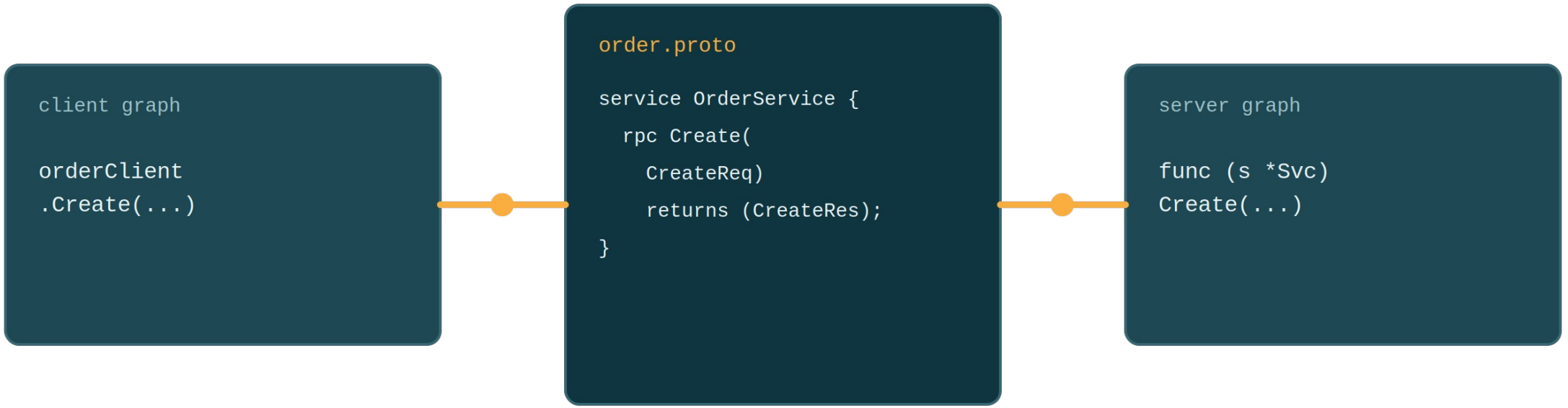
To a static analyzer these are two unrelated programs. No edge crosses the wire — so no path, no reachability verdict, no fleet-level answer.

Real attack paths are fleet-shaped



Scanned one service at a time, each report says “potentially reachable.” **Stitched, the fleet says: an exposed RPC method on the gateway reaches a vulnerable function two services away — patch tonight.**

The insight: your .proto file is the missing linker



Both sides were generated from the same service definition. **The fully-qualified method name — /pkg.OrderService/Create — is a join key that already exists in both binaries. Stitching is a metadata match, not a guess.**

What you actually match on (gRPC-Go)

gRPConf INDIA 2026

1

Client side

Generated stub methods in *_grpc.pb.go invoke cc.Invoke / NewStream with the literal full method string — harvest call sites into these stubs.

2

Server side

RegisterOrderServiceServer binds your implementation; the generated ServiceDesc maps each method name to its handler function.

3

Join

Edge: every call site of stub method M → the handler ServiceDesc maps to M, keyed on /package.Service/Method.

4

Deploy graph

Which service talks to which (mesh config, DNS targets, manifests) scopes the join — so two fleets sharing a proto don't cross-wire.

Where it gets ugly — and what holds up

Streaming RPCs

The edge lands on Send/Recv loops, not one call site. Treat the stream handler body as reached-whole; per-message flow stays out of scope.

Interceptor chains

Calls route through middleware before the handler. Model interceptors as pass-through edges, declared once per server — cheap and sound.

Reflection dispatch

Dynamic registration via server reflection or generic handlers breaks the static map. Fall back to “potentially reachable” — honestly.

Shared protos

One proto, many deployments: without a deployment graph you over-connect. Scope joins to services that actually talk in this environment.

A two-service prototype: research-stage, deliberately small




```
$ stitch --svc checkout/ --svc orders/ --deploy mesh.yaml

joined 14 RPC edges across 2 services (go/callgraph: cha)

path: Gateway.Checkout → /orders.OrderService/Create → ledger.apply → CVE-2025-XXXX [REACHABLE]
```

Built on golang.org/x/tools/go/callgraph + protobuf descriptors — nothing proprietary. Early results, not a shipped product: anyone in this room can reproduce it this weekend.

What the backlog looks like after the filter



Public industry data already shows reachability removes the large majority of dependency findings. **Crossing the wire turns “potentially reachable, three times” into one defensible verdict — in either direction.**

What you can do on Monday

gRPConf INDIA 2026

1

Treat protos as analysis metadata

Your .proto files are not just codegen input — they are the symbol table of your fleet. Keep them canonical, versioned, and queryable (a proto registry helps).

2

Start with go/callgraph

golang.org/x/tools/go/callgraph (CHA/RTA/VTA) + your generated ServiceDescs gets a single-language prototype running in days, not months.

3

Design schemas for analyzability

Generic “Any-typed” pass-through methods and reflection-heavy dispatch are blind spots by construction. Explicit RPCs are auditable RPCs.

gRPConf INDIA 2026

Your call graph should not stop where your architecture starts.

The proto file was the linker all along.

Thank you · Birajendu Sahu · linkedin.com/in/birajendup