eBPF-Based Auto-Instrumentation for Java

OpenTelemetry eBPF Instrumentation (OBI) Project

Endre Sara
Co-Founder
Causely

Stephen Lang
Staff Software Engineer
Grafana Labs

Why talk about Java eBPF

Talk Agenda

01
The Problem Space
When Java agents can't help

02
OBI Architecture
How OBI correlates kernel events with app semantics

03
Java + eBPF Challenges
JDKs, JVM internals, TLS, frameworks

04
TLS Decryption Deep-Dive
Seeing through encrypted connections

05
Operational Value
Real-world scenarios & full-stack correlation

06
Live Examples
Spring Boot · gRPC · Quarkus · Keycloak · Kafka

07
Trade-offs & Limits
What works today, what doesn't

The Problem Space

When Java agents can't help

Production Lock-Down

Ops/security teams prohibit JVM deployment modifications

Third-Party Apps

Can't modify startup of vendor-managed services

Regulated Environments

Compliance requirements restrict agent injection

Legacy Systems

Old app servers that are incompatible or run old JVMs

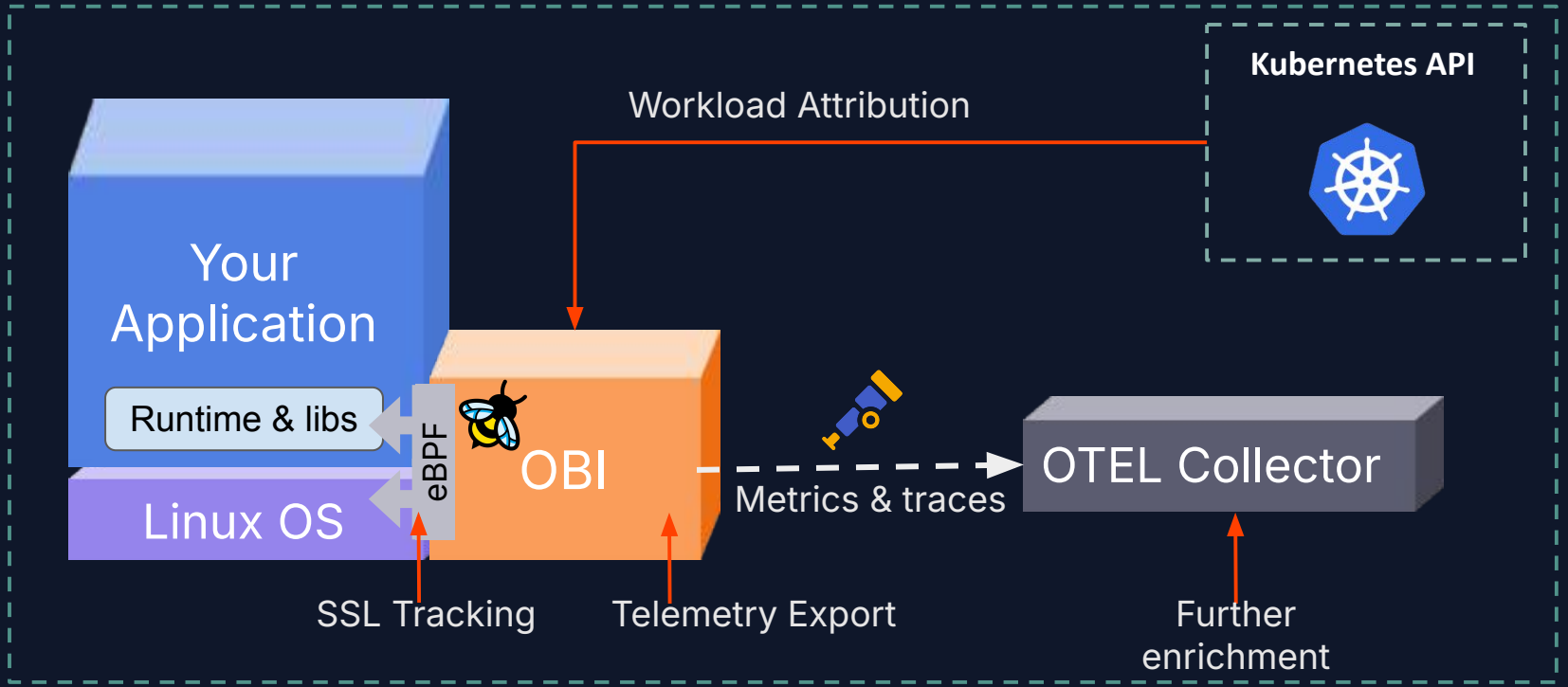
~60%

of enterprises report restrictions on agent deployment in production

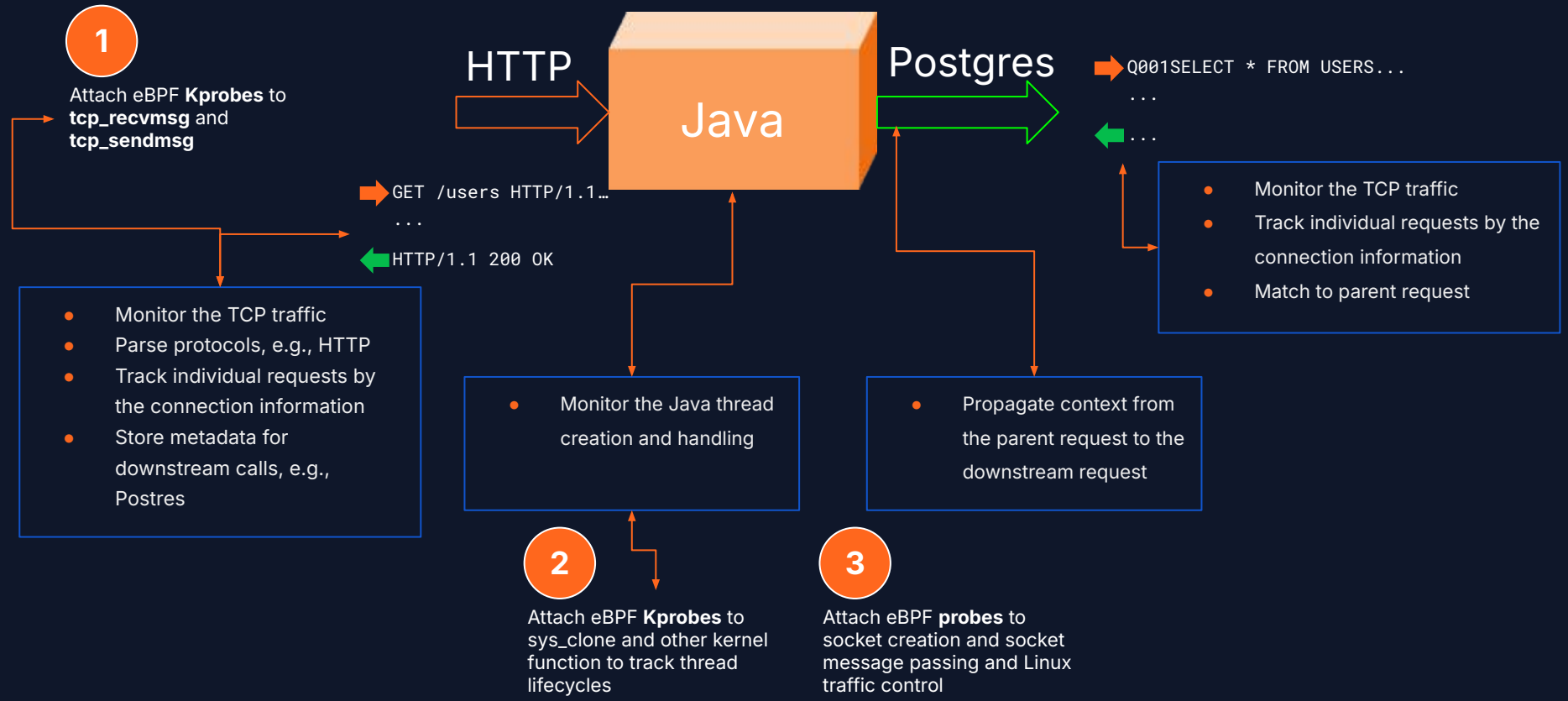
eBPF offers a path forward without touching the JVM

OBI at a glance

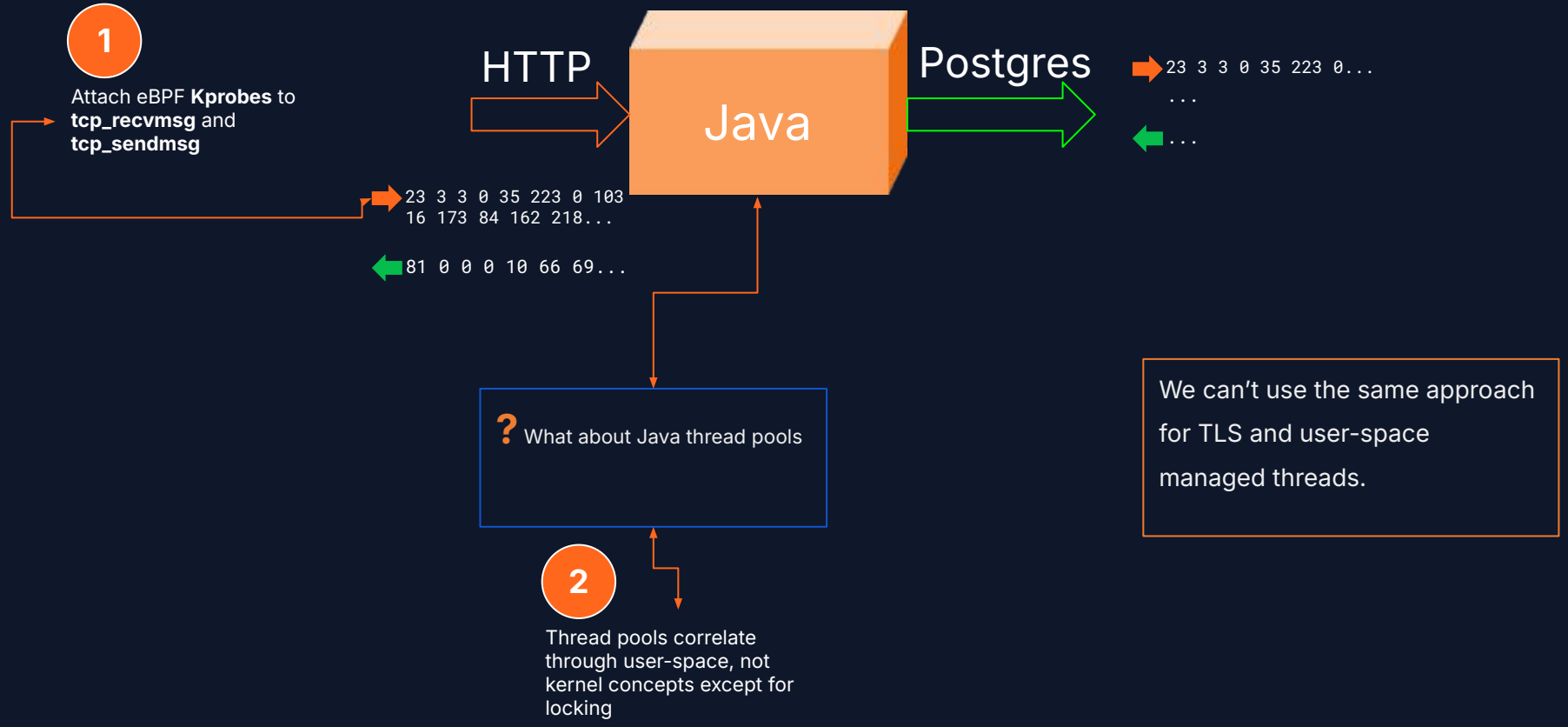
OBI architecture overview



OBI instrumenting a Java application



OBI instrumenting a Java TLS application



We can't use the same approach for TLS and user-space managed threads.

Java + eBPF: The Hard Parts

A combinatorial explosion of complexity

JDK Diversity

Oracle JDK · OpenJDK · Eclipse Temurin · Amazon Corretto · GraalVM — each with internal differences

JIT Compilation

JVM optimizes code at runtime; symbol addresses shift, inlining changes what to instrument.

TLS Everywhere

HTTPS, gRPC, encrypted DB/Kafka — payloads are ciphertext to the kernel without special handling.

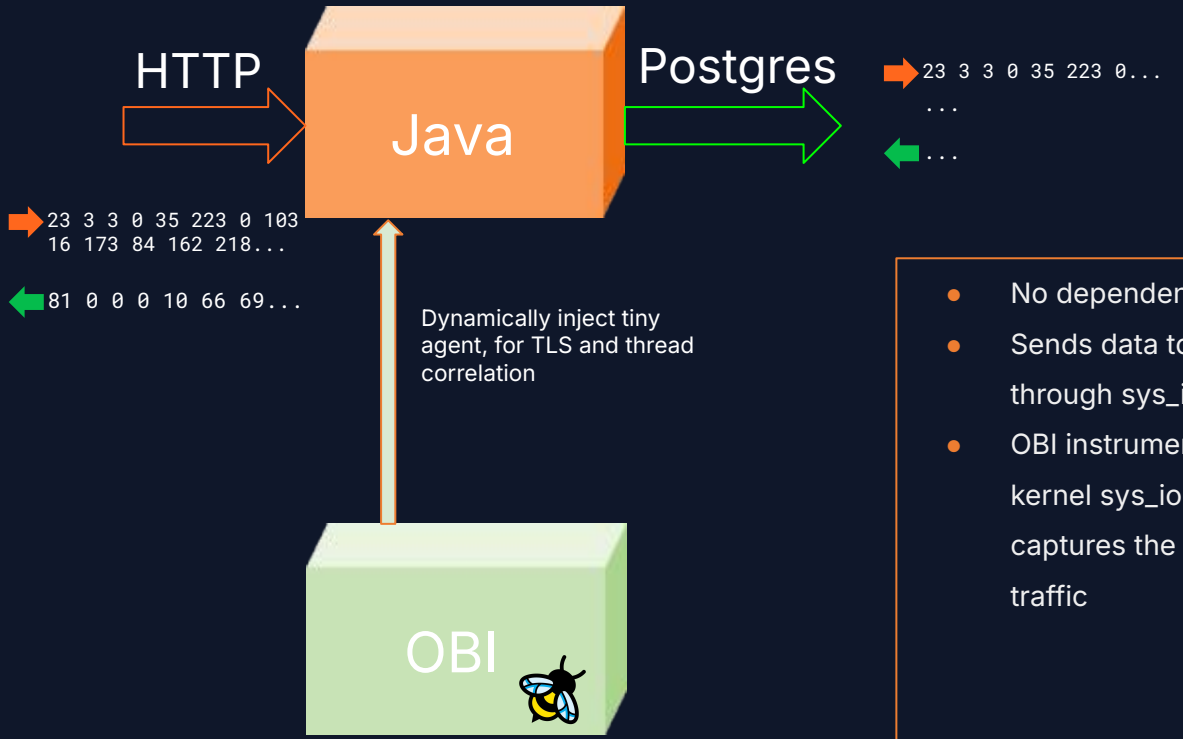
Async / Reactive

Project Reactor, CompletableFuture, virtual threads — correlation across async boundaries is non-trivial.

Anonymous JIT generated code

No uprobes support for anonymous code regions, not without serious hacks.

OBI instrumenting a Java TLS application, again



- No dependencies agent
- Sends data to the kernel through sys_ioctl
- OBI instruments the kernel sys_ioctl and captures the decrypted traffic

TLS Decryption Deep-Dive

Seeing through encrypted connections — without a proxy

THE CHALLENGE

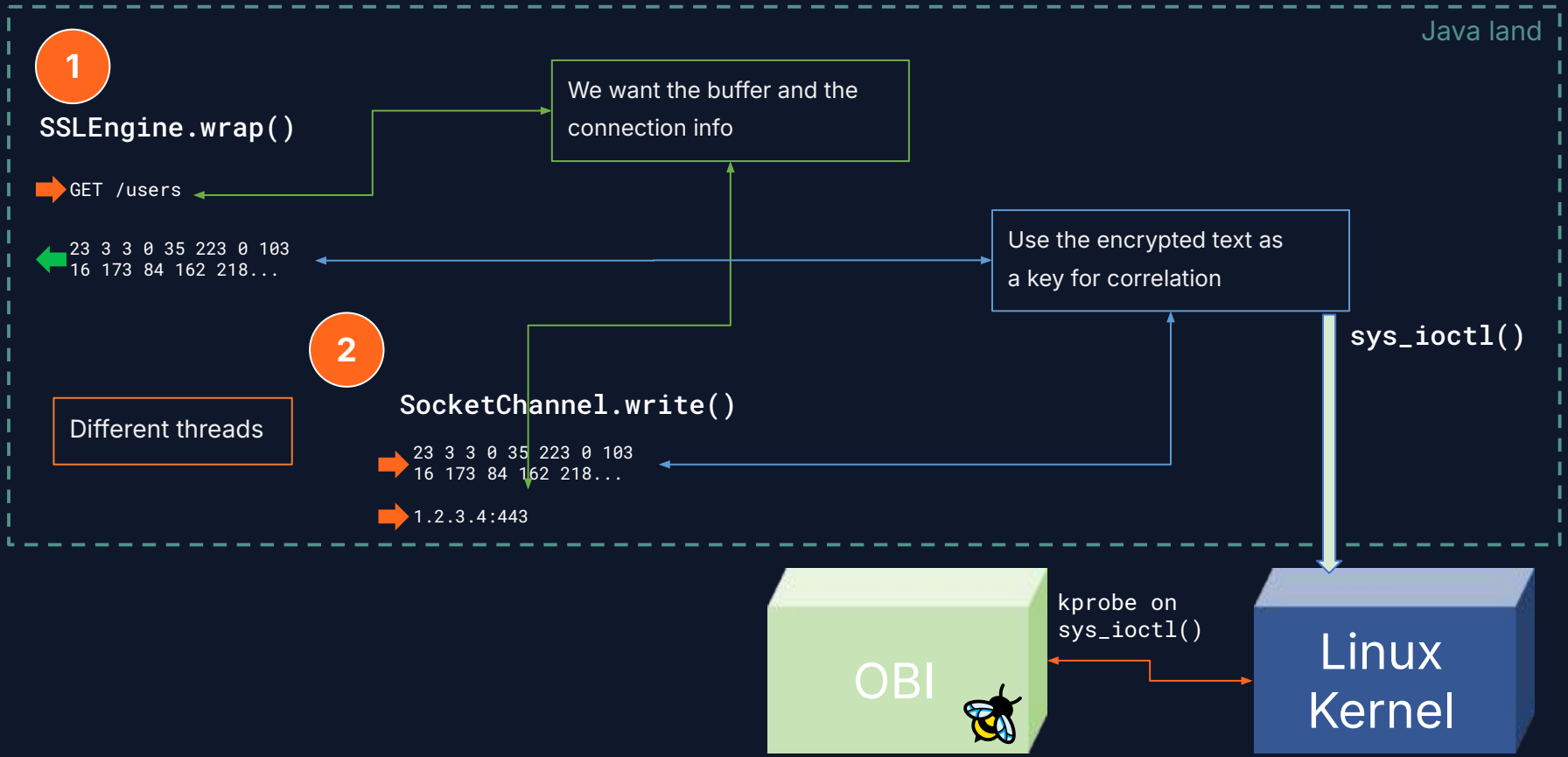
- ✗ Kernel sees only ciphertext on the wire
- ✗ HTTPS, gRPC, TLS-Postgres, TLS-Kafka all affected
- ✗ Traditional MITM proxies break mTLS & add latency
- ✗ SSL_read/SSL_write in OpenSSL/BoringSSL hold plaintext in memory
- ✗ Java has its own TLS implementation written in Java

OBI'S APPROACHES

- Uprobe on SSL_write / SSL_read (OpenSSL & BoringSSL)
- Hook fires before encrypt / after decrypt — plaintext visible
- Injects tiny OBI agent to read Java TLS
- Correlate via encrypted/decrypted buffer pairs
- Works with Java 8 through 25+, GraalVM native images partially

PR #891 — open-telemetry/opentelemetry-ebpf-instrumentation

Async Java TLS encryption



Real-World Examples

Public, reproducible repositories — try them yourself!

01

Spring Boot + HTTPS + Keycloak

Spring Boot · OpenID Connect · TLS

OBI captures OAuth token exchange and downstream REST calls, all over HTTPS

github.com/esara/java-keycloak-microservices

02

Spring Boot + gRPC + Google Pub/Sub

Spring Boot · gRPC · Protobuf · mTLS

OBI decodes HTTP/2 framing and gRPC method semantics from encrypted streams

github.com/esara/pubsub-java-microservices

03

Quarkus + TLS PostgreSQL + Kafka

Quarkus · JDBC TLS · Kafka TLS

Database queries and message production captured with full span correlation

github.com/esara/quarkus-super-heroes

Trade-offs & Current Limitations

What works today — and where the edges are

✓ WORKS RELIABLY TODAY

- HTTP/1.1 & HTTP/2 over TLS (OpenSSL/BoringSSL)
- gRPC method name & status extraction
- Spring Boot (2.x, 3.x) service-to-service calls
- JDBC over TLS (PostgreSQL driver)
- Kafka producer/consumer with TLS
- Distributed trace context propagation via W3C headers
- Quarkus and most CDI-based frameworks

⚠ CURRENT LIMITATIONS

- GraalVM native image (AOT): partial support
- Java TLS requires OBI to inject a small agent
- Reactive / async span correlation (in progress)
- JDK 21+ virtual threads: correlation gaps
- Windows / macOS (Linux-only today)

eBPF vs OTel Java Agent: When to Choose What

Complementary tools — not competitors

Scenario	OTel Java Agent	eBPF (OBI)
Full semantic instrumentation needed	✓ Best choice	Partial
Code / JVM modification prohibited	✗	✓ Best choice
Third-party / black-box services	✗	✓ Best choice
gRPC + TLS traffic	✓ Agent needed	✓ OBI (PR #891)
Custom business metrics	✓ Best choice	Limited
Zero-touch production roll-out	Complex	✓ Best choice
GraalVM native binary	✗ No JVM	Partial (improving)
Multi-language service mesh	Per-language	✓ Unified

Key Takeaways

- eBPF instrumentation is a viable alternative when Java agents cannot be deployed
- OBI correlates kernel-level events with application semantics — no JVM changes needed
- TLS decryption via small dynamically injected agent unlocks visibility into all major encrypted protocols
- Trade-offs exist: prefer OTel JDK agents where full semantic depth is required
- Active upstream development — join us in the OBI project!

Links & Resources

OBI Project
github.com/open-telemetry/opentelemetry-ebpf-instrumentation

Feature Discussion (#880)
github.com/open-telemetry/opentelemetry-ebpf-instrumentation/issues/880

Implementation PR (#891)
github.com/open-telemetry/opentelemetry-ebpf-instrumentation/pull/891

Example: Spring Boot + Keycloak HTTPS
github.com/esara/java-keycloak-microservices

Example: Spring Boot + gRPC + Pub/Sub
github.com/esara/pubsub-java-microservices

Example: Quarkus + TLS Postgres + Kafka
github.com/esara/quarkus-super-heroes