Phase 1J

Got it. Here's a tiny, surgical PR diff to implement "1J — Security-in-Dev (Mocks)" guards: redaction defaults + a test-time secret detector + docs + safe .env patterns. It assumes you already have src/logger.ts (from 1G). Apply as-is.

```
diff --git a/.gitignore b/.gitignore
index 2b3c1aa..e7c9f7b 100644
--- a/.gitignore
+++ b/.gitignore
@@ -1,3 +1,10 @@
 node_modules
 dist
 .DS_Store
+*.log
+.env
+.env.local
+.env.development
+.env.test
+.env.production
+logs/
diff --git a/.env.example b/.env.example
new file mode 100644
index 0000000..8a3b1d2
--- /dev/null
+++ b/.env.example
@@ -0,0 +1,10 @@
+# Example ONLY — no real secrets. Values intentionally fake.
+# Fill via CI secrets or local .env files that are .gitignored.
+API_BASE_URL=https://api.example.test
+API_KEY=sk_test_example_123
+GITHUB_TOKEN=ghs_test_example
+DB_URL=postgres://user:pass@localhost:5432/app_test
+OTEL_EXPORTER_OTLP_ENDPOINT=https://otel.example.test
+# Use 'test_' or 'example_' prefixes to signal fake values in fixtures.
+FEATURE_FLAG_EXAMPLE=true
+LOG_LEVEL=info
diff --git a/docs/SECURE_MOCKS.md b/docs/SECURE_MOCKS.md
new file mode 100644
index 0000000..6c95c0d
--- /dev/null
+++ b/docs/SECURE_MOCKS.md
@@ -0,0 +1,75 @@
+# Security-in-Dev (Mocks): Guardrails
+
+**Principles**
+1) Never log secrets (env, tokens, passwords). 2) Redact at source. 3) Test data must be obviously fake.
+
+**Redaction (logger)**
+- Path-based redaction for: `password`, `token`, `apiKey`, `authorization`, `cookie`, `set-cookie`, `user.email`, `payment.card.number`.
+- Deterministic replacement: `[REDACTED]`. Prefer removal for highly sensitive headers/cookies.
+
+**Fixtures & .env**
+- Real secrets never committed. `.env*` files are gitignored. Use `.env.example` with placeholders (e.g. `sk_test_…`).
+- Use faker/generated data in tests; prefer `test_` / `example_` prefixes.
+
+**CI Usage**
+- Use provider secrets; never `echo` secrets. If a runtime token is fetched, mask immediately (`::add-mask::...`) before any output.
```

```
+- Keep actions pinned by SHA and set least-privilege `permissions`.
+
+**Attribution**
+- OWASP Cheat Sheets (CC BY 3.0/4.0).
+- GitHub Actions docs (CC BY 4.0).
+- Pino examples (MIT).
+- NIST SP 800-204D (US Gov, public domain).
+
+See `src/security/secretGuard.ts` for a lightweight detector used in tests to prevent accidental real secrets in fixtures/log args.
diff --git a/src/logger.ts b/src/logger.ts
index 7f9ceab..b5b3db2 100644
--- a/src/logger.ts
+++ b/src/logger.ts
@@ -1,17 +1,43 @@
 import pino from 'pino';
-import { trace } from '@opentelemetry/api';
+import { trace } from '@opentelemetry/api';

 export const logger = pino({
-  level: process.env.LOG_LEVEL || 'info',
-  timestamp: pino.stdTimeFunctions.isoTime,
-  formatters: {
-    level(label: string) {
-      return { level: label.toUpperCase() };
-    }
-  },
-  base: {
-    service: 'poc-service',
-    environment: process.env.NODE_ENV || 'development'
-  }
+  level: process.env.LOG_LEVEL || 'info',
+  timestamp: pino.stdTimeFunctions.isoTime,
+  formatters: {
+    level(label: string) {
+      return { level: label.toUpperCase() };
+    },
+  },
+  // Security-in-Dev: redact at serialization time (CI + dev parity)
+  redact: {
+    paths: [
+      // Common secrets/headers
+      'password',
+      'token',
+      'apiKey',
+      'authorization',
+      'req.headers.authorization',
+      'req.headers.cookie',
+      'res.headers.set-cookie',
+      // PII examples
+      'user.email',
+      'payment.card.number',
+      // Wildcards
+      '*.secret',
+      '*.apiKey',
+      'items[*].token'
+    ],
+    censor: '[REDACTED]',
+    remove: false
```

```
+  },
+  base: {
+    service: 'poc-service',
+    environment: process.env.NODE_ENV || 'development'
+  },
+  mixin() {
+    const span = trace.getActiveSpan();
+    if (!span) return {};
+    const { traceId, spanId, traceFlags } = span.spanContext();
+    return { traceId, spanId, traceFlags };
+  }
 });

 export default logger;

diff --git a/src/security/secretGuard.ts b/src/security/secretGuard.ts
new file mode 100644
index 0000000..0a6f4b2
--- /dev/null
+++ b/src/security/secretGuard.ts
@@ -0,0 +1,126 @@
+/**
+ * Lightweight detector to catch accidental real secrets in test fixtures/mocks.
+ * Intended for use ONLY in tests/PoC CI. Favors false positives over leaks.
+ */
+
+export const SECRET_PATTERNS: ReadonlyArray<{ name: string; re: RegExp }> = [
+  // GitHub classic & fine-grained tokens
+  { name: 'GitHub token (classic)', re: /\bghp_[A-Za-z0-9]{36}\b/ },
+  { name: 'GitHub token (server/app)', re: /\bghs_[A-Za-z0-9_\-]{36,}\b/ },
+  // Stripe live keys (test keys should start with sk_test_)
+  { name: 'Stripe live secret', re: /\bsk_live_[A-Za-z0-9]{16,}\b/ },
+  // AWS Access Key ID
+  { name: 'AWS access key id', re: /\bAKIA[0-9A-Z]{16}\b/ },
+  // Generic JWT-like blobs
+  { name: 'JWT', re: /\beyJ[A-Za-z0-9_\-]+?\.[A-Za-z0-9_\-]+?\.[A-Za-z0-9_\-]+?\b/ },
+  // Private key markers
+  { name: 'PEM private key header', re: /-----BEGIN (?:RSA|EC|OPENSSH)? ?PRIVATE KEY-----/ },
+  // Generic password parametrization e.g. password=..., pwd=...
+  { name: 'password param', re: /\b(pass(word)?|pwd)\s*=\s*[^&\s]{6,}/i },
+];
+
+type Seen = WeakSet<object>;
+
+function isPlainObject(v: unknown): v is Record<string, unknown> {
+  return Object.prototype.toString.call(v) === '[object Object]';
+}
+
+function* walk(value: unknown, seen: Seen, path: string[] = []): Generator<{ path: string; str: string }> {
+  if (typeof value === 'string') {
+    yield { path: path.join('.'), str: value };
+    return;
+  }
+  if (Array.isArray(value)) {
+    for (let i = 0; i < value.length; i++) {
+      yield* walk(value[i], seen, [...path, String(i)]);
+    }
+    return;
+  }
```

```
+  if (isPlainObject(value)) {
+    if (seen.has(value)) return;
+    seen.add(value);
+    for (const [k, v] of Object.entries(value)) {
+      yield* walk(v, seen, [...path, k]);
+    }
+  }
+}
+
+export interface SecretHit {
+  path: string;
+  pattern: string;
+  sample: string;
+}
+
+/**
+ * Throws if a likely real secret is found.
+ * Designed for: test fixtures, mock payloads, and log-argument sanity checks in tests.
+ */
+export function assertNoRealSecrets(candidate: unknown): void {
+  const seen: Seen = new WeakSet();
+  const hits: SecretHit[] = [];
+  for (const node of walk(candidate, seen)) {
+    for (const { name, re } of SECRET_PATTERNS) {
+      const m = node.str.match(re);
+      if (!m) continue;
+      // Heuristic: allow obviously fake prefixes
+      const s = m[0];
+      if (s.startsWith('sk_test_') || s.startsWith('test_') || s.startsWith('example_')) {
+        continue;
+      }
+      hits.push({ path: node.path, pattern: name, sample: s.slice(0, 8) + '…' });
+      // stop early after a few hits
+      if (hits.length >= 3) break;
+    }
+    if (hits.length >= 3) break;
+  }
+  if (hits.length) {
+    const details = hits.map(h => `• ${h.pattern} at "${h.path}" (ex: ${h.sample})`).join('\n');
+    throw new Error(`Possible real secret(s) detected in test data:\n${details}\n` +
+      `Use clearly fake values (e.g., "sk_test_", "test_", "example_").`);
+  }
+}
+
+/**
+ * Small helper for test code: wrap a call that should not receive real secrets.
+ */
+export function guardArgs<T extends unknown[]>(fn: (...args: T) => unknown, ...args: T): unknown {
+  for (const a of args) assertNoRealSecrets(a);
+  return fn(...args);
+}
+
diff --git a/test/secretGuard.test.ts b/test/secretGuard.test.ts
new file mode 100644
index 0000000..b8a9a2f
--- /dev/null
+++ b/test/secretGuard.test.ts
@@ -0,0 +1,55 @@
+import { describe, it, expect } from 'vitest';
```

```
+import { assertNoRealSecrets, guardArgs } from '../src/security/secretGuard';
+
+describe('secretGuard', () => {
+  it('allows clearly fake values', () => {
+    const mock = {
+      headers: { authorization: 'Bearer sk_test_example_123' },
+      token: 'test_token_abc',
+      note: 'example_value',
+    };
+    expect(() => assertNoRealSecrets(mock)).not.toThrow();
+  });
+
+  it('flags likely real secrets (GitHub token)', () => {
+    const bad = { token: 'ghp_1234567890abcdef1234567890abcdef1234' };
+    expect(() => assertNoRealSecrets(bad)).toThrow(/GitHub token/);
+  });
+
+  it('flags likely real secrets (AWS key)', () => {
+    const bad = { key: 'AKIA1234567890ABCD' };
+    expect(() => assertNoRealSecrets(bad)).toThrow(/AWS access key id/);
+  });
+
+  it('guardArgs rejects unsafe payloads before invoking', () => {
+    const sink = (x: unknown) => JSON.stringify(x).length;
+    expect(() => guardArgs(sink, { jwt: 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.bad.sig' }))
+      .toThrow(/JWT/);
+  });
+});
+
diff --git a/.github/workflows/ci.yml b/.github/workflows/ci.yml
index a2c9b33..6b0f0f1 100644
--- a/.github/workflows/ci.yml
+++ b/.github/workflows/ci.yml
@@ -35,6 +35,18 @@ jobs:
       - run: npm ci
       - run: npm run lint:tsc
       - run: npm test
+      # Security-in-Dev (Mocks): demonstrate safe masking for any runtime tokens
+      - name: Mask runtime tokens (defense-in-depth)
+        shell: bash
+        run: |
+          # Example: if a token is fetched/generated earlier in the job:
+          TEST_TOKEN="${GITHUB_TOKEN:-example-token}"
+          echo "::add-mask::${TEST_TOKEN}"
+          # DO NOT echo secrets; pipeline tools may still capture stdout/stderr.
+          # This step exists to codify the policy (no-op for forks/PRs).
+
```

What this gives you (Phase 1J, PoC-safe)

- Default redaction in src/logger.ts for common secret/PII paths (applies in dev + CI).
- Fixture/mocks guard (assertNoRealSecrets) that fails tests if something looks real.
- Unit tests proving both safe vs unsafe cases.
- .env hygiene via .env.example + .gitignore.
- Docs (docs/SECURE_MOCKS.md) with attributions (OWASP, GitHub, Pino, NIST).
- CI nudge step showing how to mask any runtime token (policy codified in code).

If you want, I can also drop a tiny pre-commit hook (optional) that runs vitest -t secretGuard and a lightweight gitleaks scan, but this patch keeps it minimal and fast for Phase-1.

Absolutely—here's a tiny, optional pre-commit hook that runs the secretGuard vitest and a lightweight gitleaks scan against staged changes. It uses Husky and skips gitleaks if it's not installed (so it won't block you locally).

```
diff --git a/package.json b/package.json
```

```
index 6d8a0ef..a3b8c22 100644
--- a/package.json
+++ b/package.json
@@ -5,12 +5,20 @@
   "type": "module",
   "scripts": {
     "test": "vitest --run",
+    "test:secrets": "vitest --run -t secretGuard",
+    "scan:secrets": "gitleaks protect --staged --no-banner --redact",
+    "prepare": "husky install",
     "lint": "eslint .",
     "lint:fix": "eslint --fix .",
     "type-check": "tsc --noEmit",
     "check-all": "pnpm format && pnpm lint:fix && pnpm type-check"
   },
   "devDependencies": {
+    "husky": "^9.0.0",
     "typescript": "^5.6.0",
     "vitest": "^2.0.0"
   }
 }
diff --git a/.husky/pre-commit b/.husky/pre-commit
new file mode 100755
index 0000000..b7b2b7a
--- /dev/null
+++ b/.husky/pre-commit
@@ -0,0 +1,30 @@
+#!/usr/bin/env sh
+# Minimal pre-commit: run secret guard test and a lightweight staged gitleaks scan.
+# Note: this hook is optional; gitleaks is skipped if not installed.
+
+set -eu
+
+# run vitest secret-guard (prefer pnpm if available)
+if command -v pnpm >/dev/null 2>&1; then
+  pnpm run -s test:secrets
+else
+  npm run -s test:secrets
+fi
+
+# run gitleaks against staged changes if present
+if command -v gitleaks >/dev/null 2>&1; then
+  # Use repo config if present; otherwise default rules
+  if [ -f ".gitleaks.toml" ]; then
+    gitleaks protect --staged --no-banner --redact --config .gitleaks.toml
+  else
+    gitleaks protect --staged --no-banner --redact
+  fi
+else
+  echo "[pre-commit] gitleaks not found; skipping secret scan." >&2
+  echo "  Install with: brew install gitleaks  # or see: https://github.com/gitleaks/gitleaks" >&2
+fi
+
+exit 0
diff --git a/.gitleaks.toml b/.gitleaks.toml
new file mode 100644
index 0000000..9b59e34
--- /dev/null
+++ b/.gitleaks.toml
```

```
@@ -0,0 +1,18 @@
+title = "PoC gitleaks allowlist (test tokens)"
+
+[allowlist]
+description = "Permit obvious test/example tokens to reduce false positives."
+paths = [] # keep empty; we want staged scan coverage
+regexes = [
+  '''sk_test_[A-Za-z0-9]{2,}''',
+  '''test_[A-Za-z0-9_\-]{2,}''',
+  '''example[_-][A-Za-z0-9_\-]{2,}'''
+]
+
+# Optional: exclude typical fixture folders if you prefer
+#paths = [
+#  '''^test/fixtures/''',
+#  '''^__fixtures__/'''
+#]
diff --git a/docs/SECURE_MOCKS.md b/docs/SECURE_MOCKS.md
index 6c95c0d..9fa1a22 100644
--- a/docs/SECURE_MOCKS.md
+++ b/docs/SECURE_MOCKS.md
@@ -1,6 +1,16 @@
 # Security-in-Dev (Mocks): Guardrails
@@
 **CI Usage**
 - Use provider secrets; never `echo` secrets. If a runtime token is fetched, mask immediately (`::add-mask::...`) before any output.
 - Keep actions pinned by SHA and set least-privilege `permissions`.
+
+**Pre-commit (optional)**
+- This repo includes a Husky pre-commit hook that:
+  1) Runs `vitest -t secretGuard`
+  2) Runs `gitleaks protect --staged` (skips if gitleaks not installed)
+- Install Husky hooks after cloning:
+  ```sh
+  npm i  # or pnpm i
+  ```
+  (The `"prepare": "husky install"` script wires up `.husky/`.)
@@
 **Attribution**
```

Notes

- The hook fails the commit if secretGuard fails or if gitleaks finds a staged secret.
- gitleaks is optional locally; the hook prints a hint and continues if it’s missing.
- The .gitleaks.toml allowlists *obvious* test tokens (sk_test_, test_, example_) to reduce noise while still catching real creds.

If you prefer Docker-based gitleaks instead of a local install, I can swap the hook to:

```
docker run --rm -v "$PWD:/repo" -w /repo zricethezav/gitleaks:latest protect --staged --no-banner --redact
```

Just say the word.