Verifiable eBPF Traces for Supply Chain Artifacts with Witness and Tetragon

Cole Kennedy

TESTIFYSEC

Cole Kennedy

CEO

linkedin.com/in/thecolekennedy

@colek42c

We are building a platform that enables enterprise to verify the integrity, compliance and trustworthiness of their software.

*"Bill in S**security**ain sure regrets the day he said to **dev**, 'Then why don't you show me how it's done?'"*

scdigest.com

Why Collect Traces of Software Builds?

- Find Hidden CVEs
- Thwart Malicious Actors
- Automate pipeline compliance (SSDF Controls)

- Accounting of build materials
- Accounting of build processes
- Tamper detection

What is Witness?

Witness implements the in-toto spec

Allows software producers to make and verify attestations about the software they produce

Integrations with tooling such as Sigstore and SPIRE for keyless signing.

Integrations into GitHub and GitLab

Makes it easy to produce verifiable evidence for software builds.

Supports containerized and non-containerized workloads

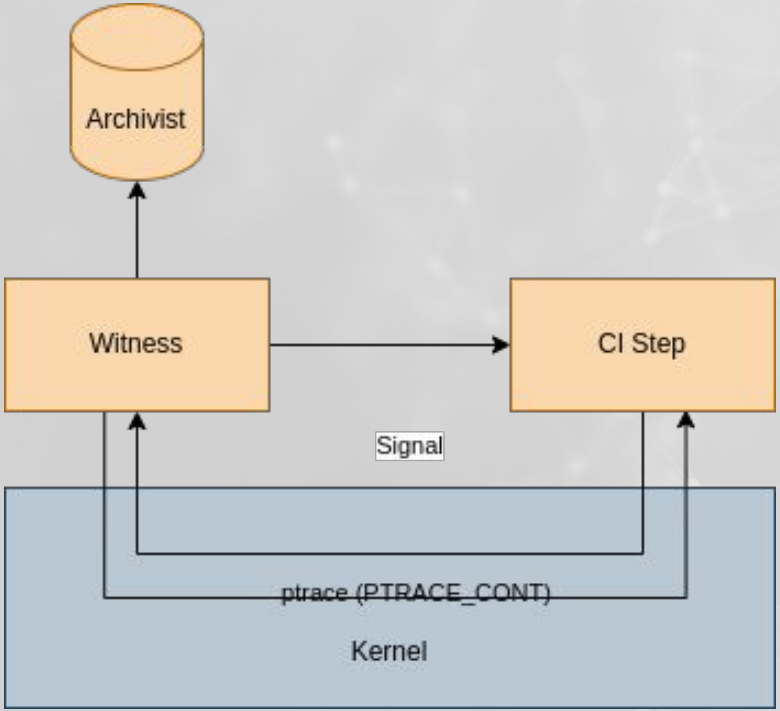
What is Archivist?

Untrusted store

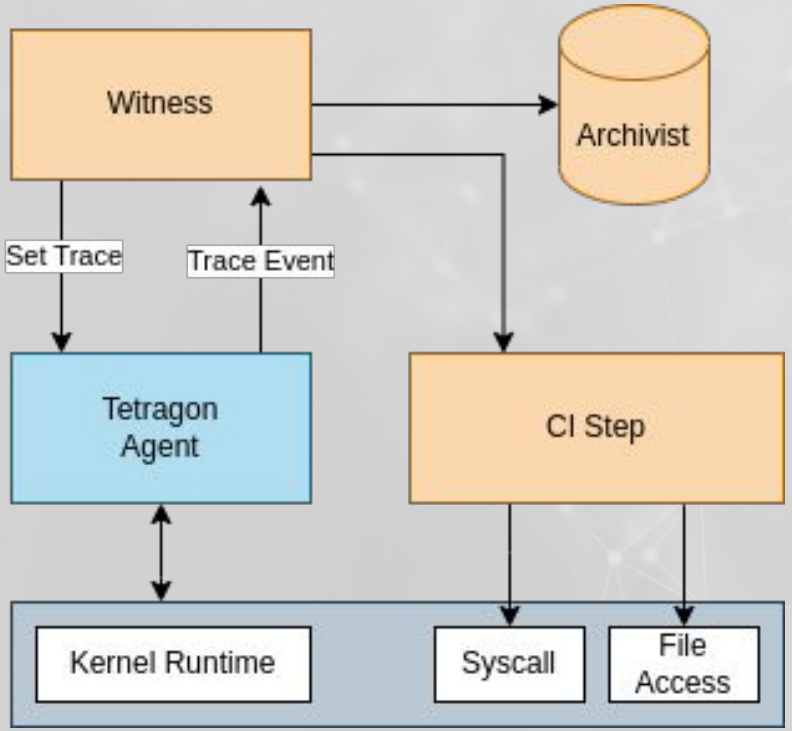
Indexes attestations in a graph database (ent.io)

Provides a GraphQL API into attestation data

How Witness Currently Collects Traces with ptrace



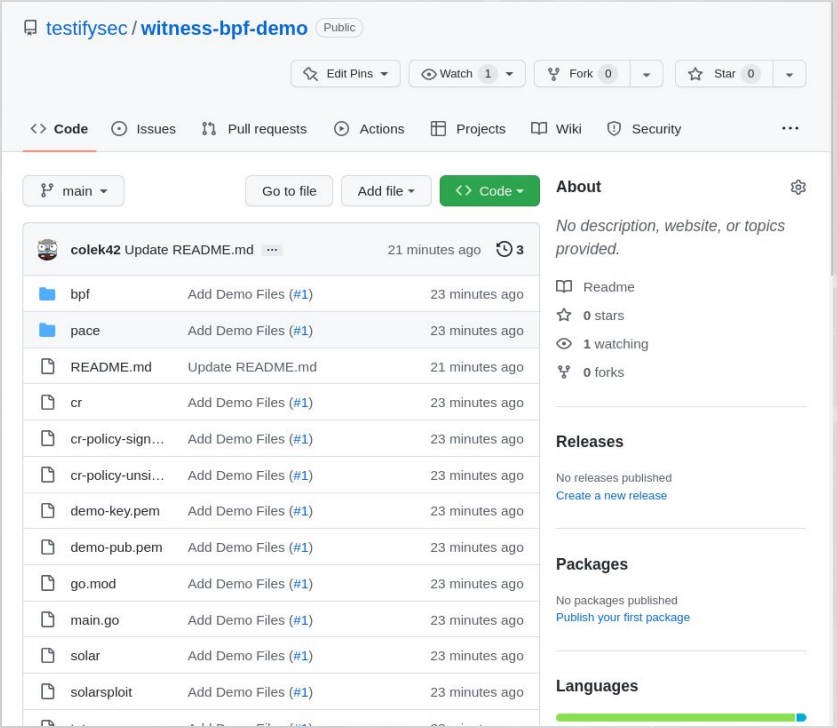
How we implement eBPF support with Tetragon



Ptrace vs BPF Tradeoffs

BPF	ptrace
Asynchronous - No guarantees on file identity - Small performance impact	Synchronous - File hashes calculated after syscall, before program reads file. - Larger performance impact
Can monitor entire system	Only monitors CI process and descendants
Easier to write complex collection rules with tetragon abstractions	Parsing syscalls is complex
Records all network and file events	Currently no support for network calls
Agent needs to run as root	Works with non-root, in containers, and in shared runners.
Requires modern Kernel	Introduced in Unix V6 (1976)

Follow Along



Use Case: Detect "SolarBurst" Tampering

Scenario:

An attacker has compromised your build system and installed a agent that hot patches files loaded by the compiler with malicious code

Mitigation:

Build system security engineers detect irregular activity in Witness trace logs. The file loaded in the compiler does not match files checked into the repository. Build is NOT approved for deployment to production systems.

Use Case: IR - Upstream Build System Compromise

Scenario:

An attacker has compromised a compiler used to produce software artifacts across your organization.

Mitigation:

Incident response team searches the Witness log for all artifacts deployed, then filters that search using the SHASUM for the trojanized compiler and Witness tracing data. New workloads are blocked from compute and network resources. CVEs are issued for all affected publicly released artifacts.

```
package trace_demo

deny[msg]{
    input.processes[_].programdigest.sha256 ==
        "d6a989a902300b4ac75adf53a0251593e4b14110d2ebcef5d03eb59a9214a052"
    msg := "Malicious Compiler"
}
```

What are your Questions?

https://witness.dev
https://testifysec.com

linkedin.com/in/thecolekennedy

@colek42c