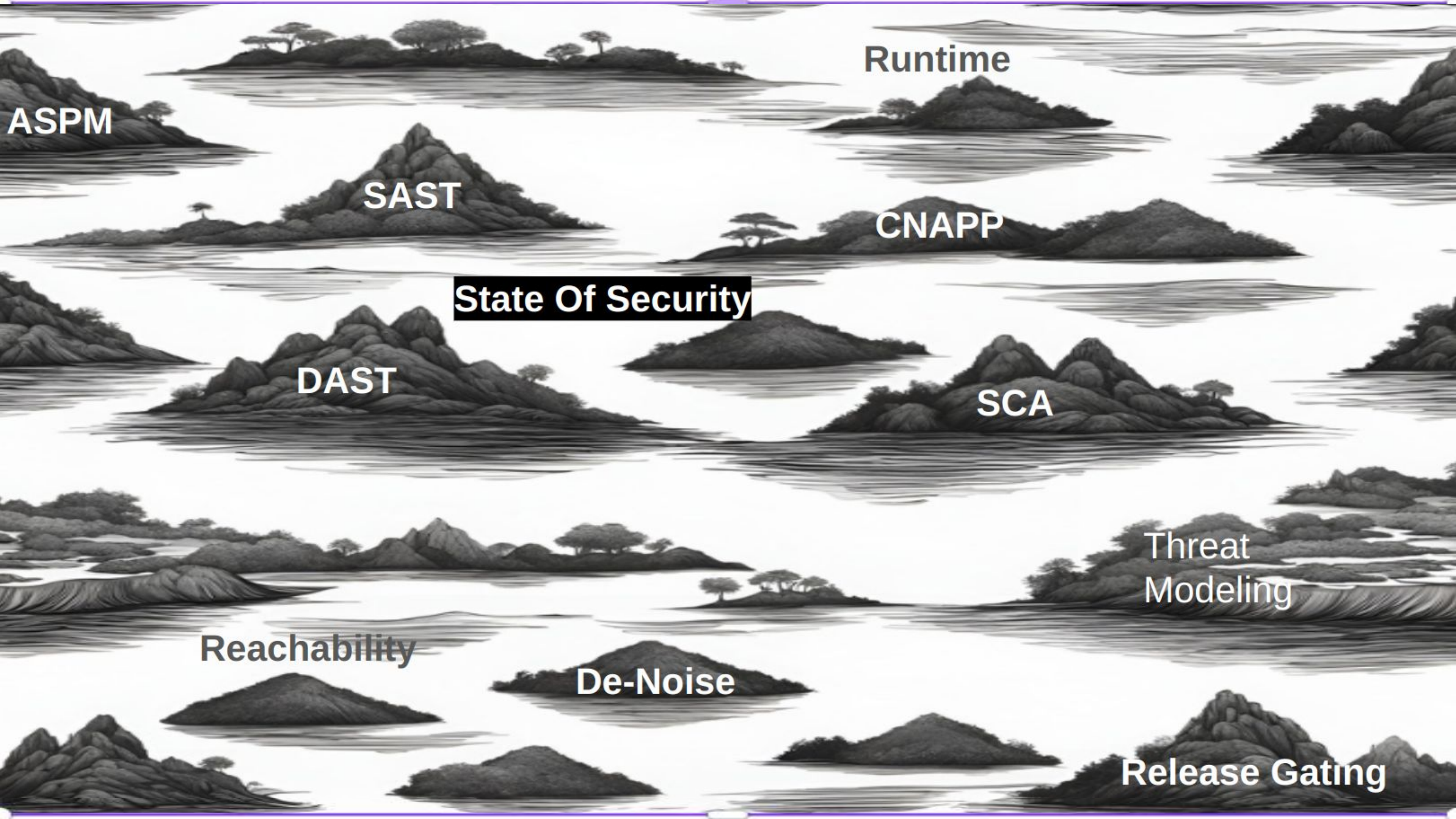
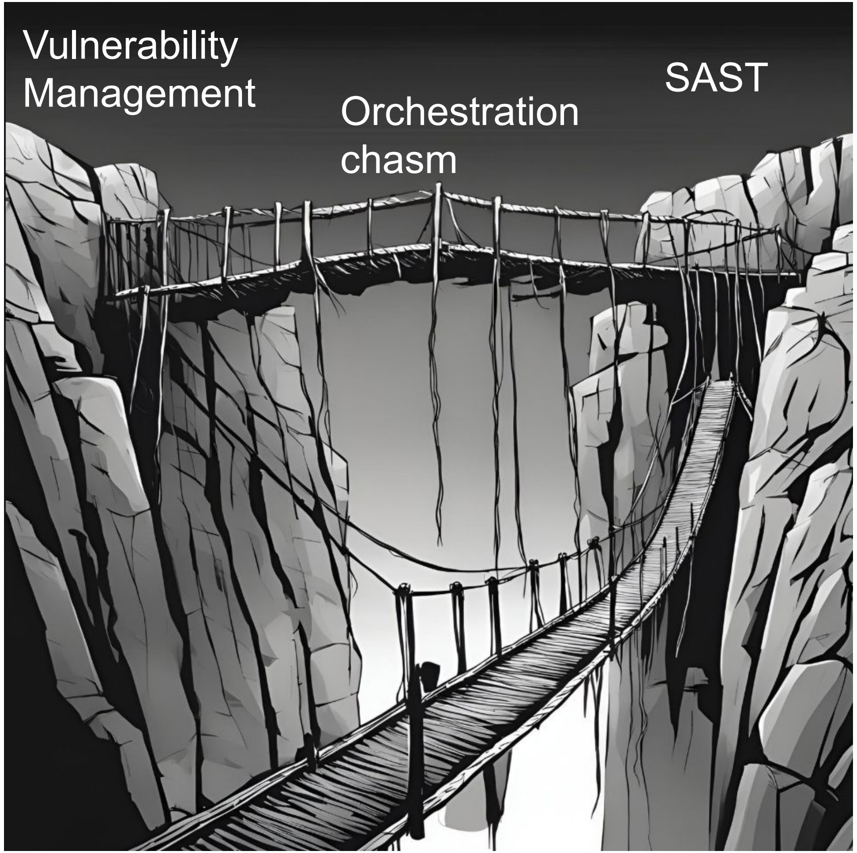
Unifying Security Tools

with OCSF and 60 lines of code



Runtime
ASPM
SAST
CNAPP
State Of Security
DAST
SCA
Threat Modeling
Reachability
De-Noise
Release Gating

Dirty scripts to the rescue



Nice to meet you

Spyros Gasteratos

- OSS Dev – SecEng
- Founder / CEO Smithy
- Creator of OpenCRE.org
- OWASP volunteer

Itinerary: a tale of discovery and invention

1. ~~Problem statement~~
2. Unifying Technologies: Sarif, OCSF, Orchestration
3. Our Solution: SMITHY
4. How to build a component
5. Demo
6. Pitfalls
7. Next steps
8. Closing remarks

SARIF (Static Analysis Results Interchange Format)

- Open Source Standard for reporting SAST vulnerability findings

- Pros:
  - support from a lot of SAST vendors – Github
  - Human and machine readable
  - JSON Schema

- Cons:
  - support MOSTLY by SAST vendors shallow details
  - weak schemas

OCSF (Open Cybersecurity Schema Framework)

- Security agnostic schemas

- Pros:
  - SAST++++
  - Schemas AND tools (JSON, Protobuf)
  - More expressive than SARIF
  - Extensible

- Cons:
  - Designed by committee
  - Tools STILL don’t map the same way
  - Steep learning curve

OCSF

Open Cybersecurity Schema Framework
v1.3.0
Extensions
Linux (1)
v1.3.0
Windows (2)
v1.3.0
Profiles
Cloud
Container
Data Classification
Date/Time
Host
Linux Users
Load Balancer
Network Proxy
OSINT
Security Control
OCSF Schema
Categories
Classes
Base Event
Dictionary
Objects
Resources
Search
Categories
The OCSF categories organize event classes, each aligned with a specific domain or area of focus.
System Activity (1)
File System Activity (1001)
Kernel Extension Activity (1002)
Kernel Activity (1003)
Memory Activity (1004)
Module Activity (1005)
Scheduled Job Activity (1006)
Process Activity (1007)
Event Log Activity (1008)
Findings (2)
Security Finding (2001)
Vulnerability Finding (2002)
Identity & Access Management (3)
FTP Activity (4008)
Email Activity (4009)
Network File Activity (4010)
Email File Activity (4011)
Email URL Activity (4012)
NTP Activity (4013)
Tunnel Activity (4014)
Kernel Object Query (5006)
File Query (5007)
Folder Query (5008)
Admin Group Query (5009)
Job Query (5010)
Module Query (5011)
Network Connection Query (5012)
Networks Query (5013)
Peripheral Device Query (5014)
Process Query (5015)
Service Query (5016)
User Session Query (5017)
User Query (5018)
Device Config State Change (5019)
Software Inventory Info (5020)
Application Activity (6)
Web Resources Activity (6001)
Application Lifecycle (6002)
API Activity (6003)
Web Resource Access Activity (6004)
Datastore Activity (6005)
File Hosting Activity (6006)
Scan Activity (6007)
Remediation (7)
Remediation Activity (7001)
File Remediation Activity (7002)
Process Remediation Activity (7003)
Network Remediation Activity (7004)
20+ categories/types!

Challenges with orchestration

- Running security tools **reliably** not trivial
- Leveraging common knowledge is hard
- Not straightforward feedback loops

Taming the chaos

- Standardise tools execution and implementation
- Automatic instrumentation:
  - metrics
  - logs
  - traces
  - panic handling
  - …
- Not impacting on production CI pipelines

Orchestration

Tools can

- Report in OCSF format - Data Lake
- Run in the same and predictable ways - Reliability
- Be built in the same way with an SDK - Maintainability/Adoption
- Be orchestrated locally, on CI, on Premises or on SAAS

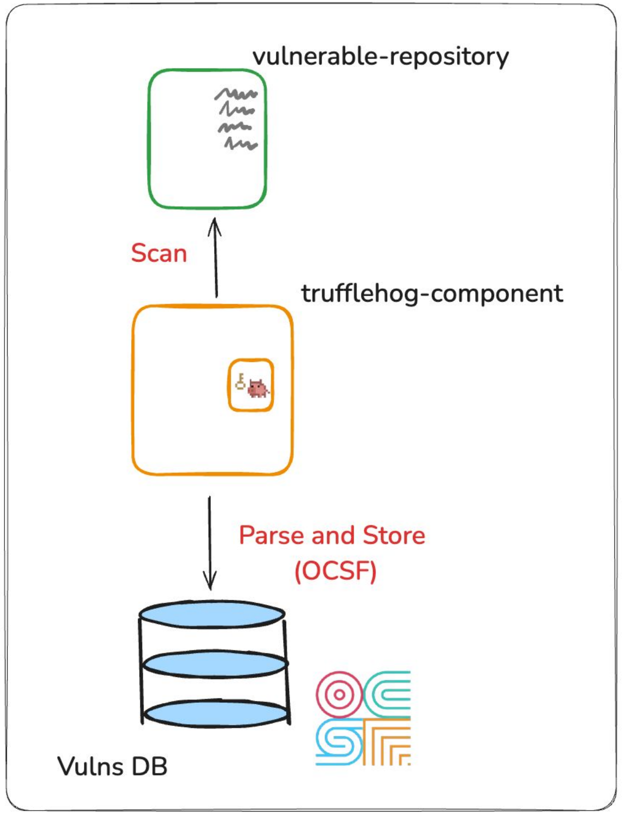
SMITHY : SDK for OCSF and Orchestration

- SDK + workflow engine for security tooling
- Fetch Artifacts, Scan, Enrich, Filter and Report functionality
- Run locally, on CI or wherever you can orchestrate containers

Components

What do they do?

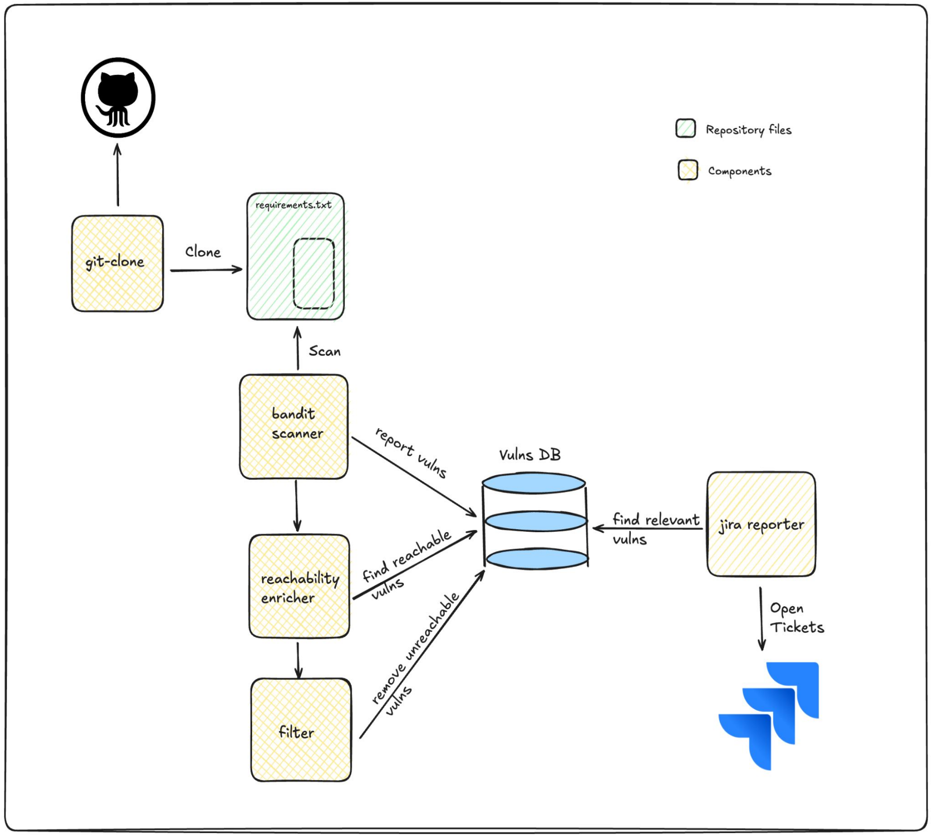
1. Wrap security tooling
2. Execute and parse results
3. Parse results to OCSF
4. Store



Workflows

- Define component execution order and configuration

- Configurable via yaml or CLI



Component Configuration

```yaml
name: gosec-parser
description: "Parses gosec findings into OCSF format"
type: scanner
steps:
  - name: scanner
    image: "docker.io/securego/gosec:2.15.0"
    executable: >
      -fmt=sarif
      -no-fail
      -out=/workspace/repos/gosec.json
      /workspace/repos/govwa
  - name: parser
    image: "ghcr.io/smithy-security/images/components/scanners/gosec:latest"
    env_vars:
      GOSEC_RAW_OUT_FILE_PATH: repos/gosec.json
```

Workflow Configuration

```yaml
description: "GoSec sample pipeline"
name: "gosec-pipeline"
components:
  - component: "ghcr.io/smithy-security/manifests/components/target/git-cloner:v0.1.0"
  - component: "ghcr.io/smithy-security/manifests/components/scanner/gosec-parser:v1.0.0"
  - component: "ghcr.io/smithy-security/manifests/components/enricher/custom-annotation:v1.2.0"
  - component: "ghcr.io/smithy-security/manifests/components/reporter/json-logger:v2.0.0"
```

SDK

- Go SDK to write components
- Plug and play
- Focus on writing business logic
- Speaks OCSF
- Reliability, Storage and Monitoring instrumentation capabilities built in

Component Specification

```go
type Enricher interface {
	// Annotate enriches vulnerability findings by some criteria.
	Annotate(ctx context.Context, findings []VulnerabilityFinding) ([]VulnerabilityFinding, error)
}
```

Example Implementation

```go
type esReporter struct {
  esClient *elasticsearch.Client
}

func (e esReporter) Report(
    ctx context.Context,
    findings []VulnerabilityFinding,
) error {
    logger := component.
        LoggerFromContext(ctx).
        With(
            slog.Int("num_findings", len(findings)),
        )

    for _, finding := range findings {
        b, err := protojson.Marshal(finding.Finding)
        if err != nil {
            return errors.Errorf("could not json marshal finding: %w", err)
        }
        e.esClient.Index("findings", bytes.NewBuffer(b))
    }

    return nil
}
```

Orchestration

Locally

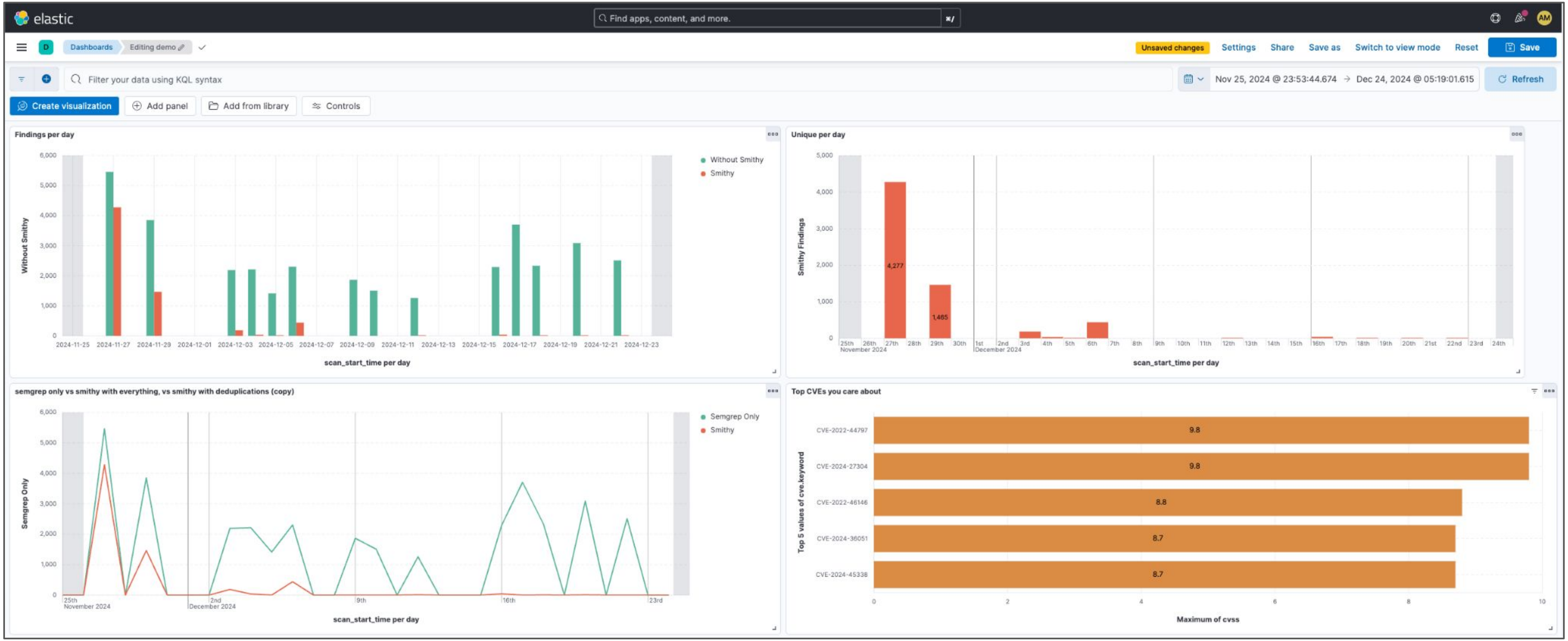
smithyctl executes workflows with a simple execution engine

Or wherever you want

smithyctl is a single binary and can run anywhere:

- CI
- Container orchestrators
- ???

Demo


elastic
Find apps, content, and more.
Dashboards
Editing demo
Unsaved changes
Settings
Share
Save as
Switch to view mode
Reset
Save
Filter your data using KQL syntax
Nov 25, 2024 @ 23:53:44.674
Dec 24, 2024 @ 05:19:01.615
Refresh
Create visualization
Add panel
Add from library
Controls
Findings per day
Without Smithy
Smithy
scan_start_time per day
Unique per day
Smithy Findings
4,277
1,465
scan_start_time per day
semgrep only vs smithy with everything, vs smithy with deduplications (copy)
Semgrep Only
Smithy
scan_start_time per day
Top CVEs you care about
CVE-2022-44797
9.8
CVE-2024-27304
9.8
CVE-2022-46146
8.8
CVE-2024-36051
8.7
CVE-2024-45338
8.7
Top 5 values of cve.keyword
Maximum of cvss

Pitfalls

- Not using open standards and SDKs
- Raw Data dumping in human - focused fields
- Not being strict about original tool info – less is more
- Relying only on AI mappings

Next Steps

- SDK V1 – helper methods, shortcuts
- Community registry – publish, discover and download components and workflows
- SDK V2 – more observability, support more of OCSF natively
- More composable workflows
- Native LLM bindings
- And many more

To recap

- **Dirty scripts don't scale**
- **Interoperability:** The only way to do security is Open Standards
- **Short Feedback loops:** Fast and flexible integrations
- Smithy is Open Source, you can find it at: https://github.com/smithy-security/smithy

Thank You

You can find the slides here: