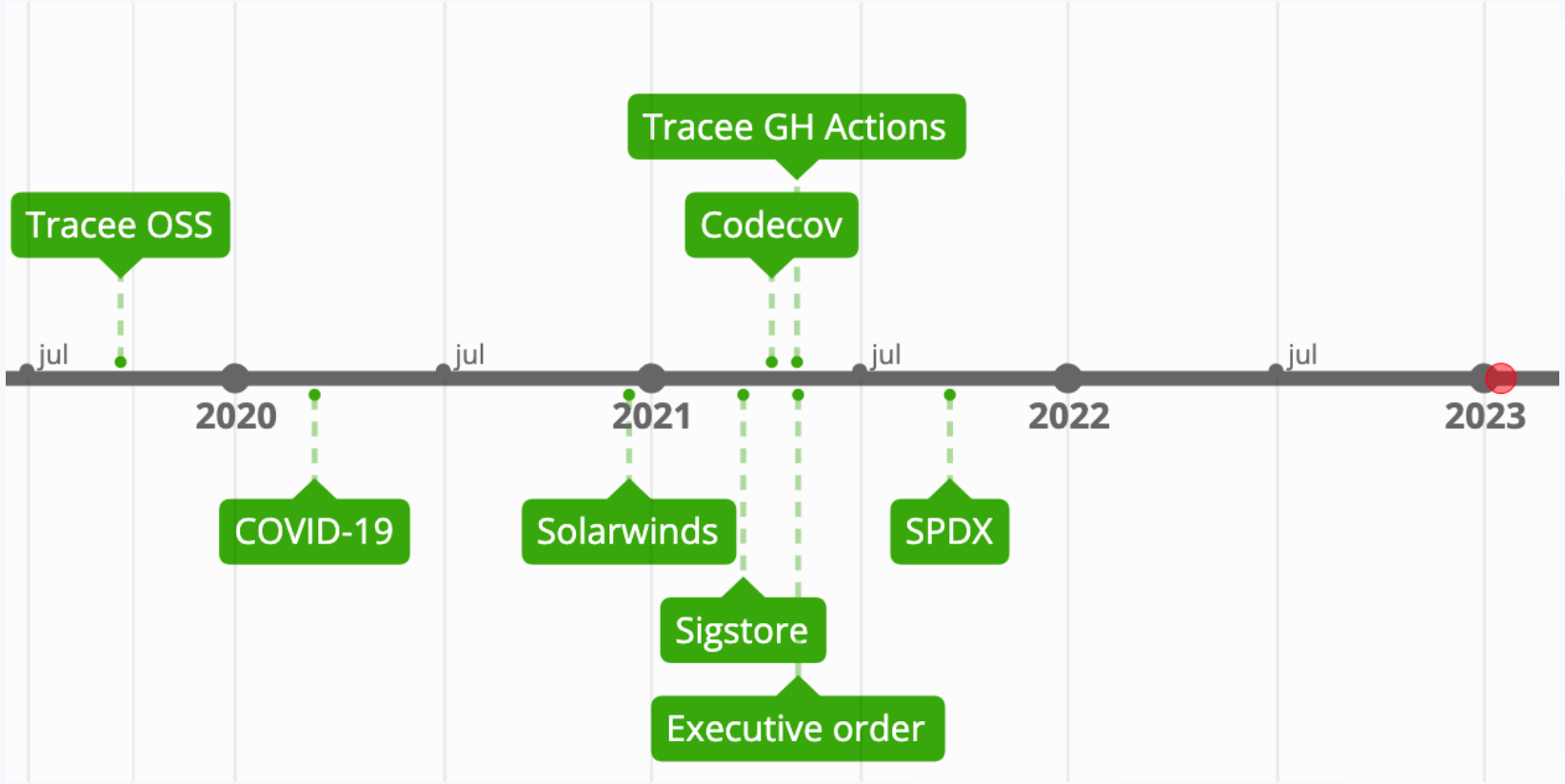
aqua

Verfiable GitHub Actions using eBPF

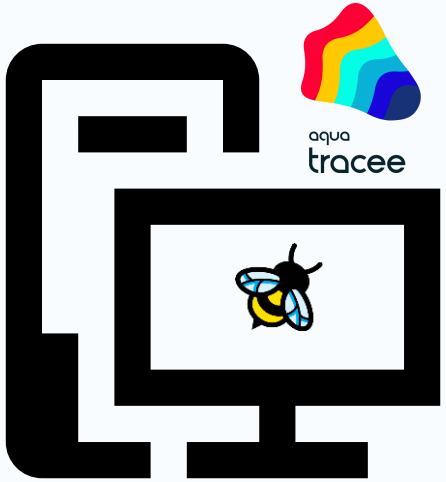
Itay Shakury itaysk

Jose Donizetti josedonizetti

“just a couple of years ago”



Tracee - Runtime Security and Forensics using eBPF



Container started

Process Executed

Over 500 event types

Network called

Reverse Shell

Container stopped

First solution – Tracee in the pipeline

```
services:
  tracee:
    image: aquasec/tracee:latest
    options: "--privileged -v /proc:/proc -v /boot:/boot -v /lib/modules/:/lib/modules/:ro -v
/usr/src:/usr/src:ro -v /tmp/tracee:/tmp/tracee"
```

```
- name: Show Tracee logs
  run: docker logs "${{ job.services.tracee.id }}"
```

```
Show Tracee logs                                                    0s
1  ▶ Run docker logs "06524c58ff7f68807198ea7d296ce3eedd164652304e7b7a22ce4a99962060e2"
7  Loaded signature(s):  [TRC-1 TRC-2 TRC-3 TRC-4 TRC-5 TRC-6 TRC-7]
```

```
Loaded signature(s):  [TRC-1 TRC-2 TRC-3 TRC-4 TRC-5 TRC-6 TRC-7]

*** Detection ***
Time: 2021-04-29T19:26:13Z
Signature ID: TRC-1
Signature: Standard Input/Output Over Socket
Data: map[ip:8.8.8.8]
Command: poc.py
Hostname: fv-az210-368
```

```
- name: Fail pipeline on malicious activity
  run: docker logs "${{ job.services.tracee.id }}" | grep -v "Detection"
```

Signatures

- ✗ New executable
- ✓ Evasion techniques
- ✓ Crypto miners
- ✗ Spawned shell
- Custom signatures

itaysk

Available Rules

Search

Getting Started | Tutorials | Docs | Contributing

Name	Description	Full Description
TRC-101	Process standard input/output over socket detected	A process has its standard input/output redirected to a socket. This behaviour is the base of a Reverse Shell attack, which is when an interactive shell being invoked from a target machine back to the attacker's machine, giving it interactive control over the target. Adversaries may use a Reverse Shell to retain control over a compromised target while bypassing security measures like network firewalls.
TRC-102	Anti-Debugging detected	A process used anti-debugging techniques to block a debugger. Malware use anti-debugging to stay invisible and inhibit analysis of their behavior.
TRC-103	Code injection detected using ptrace	Possible code injection into another process was detected. Code injection is an exploitation technique used to run malicious code, adversaries may use it in order to execute their malware.
TRC-104	Dynamic code loading detected	Possible dynamic code loading was detected as the binary's memory is both writable and executable. Writing to an executable allocated memory region could be a technique used by adversaries to run code undetected and without dropping executables.
TRC-105	Fileless execution detected	Fileless execution was detected. Executing a process from memory instead from a file in the filesystem may indicate that an adversary is trying to avoid execution detection.
TRC-106	Cgroups notify_on_release file modification	An attempt to modify Cgroup notify_on_release file was detected. Cgroups are a Linux kernel feature which limits the resource usage of a set of processes. Adversaries may use this feature for container escaping.
TRC-107	LD_PRELOAD code injection detected	LD_PRELOAD usage was detected. LD_PRELOAD lets you load your library before any other library, allowing you to hook functions in a process. Adversaries may use this technique to change your applications' behavior or load their own programs.
TRC-108	K8s service account token file read	The Kubernetes service account token file was read on your container. This token is used to communicate with the Kubernetes API Server. Adversaries may try to communicate with the API Server to steal information and/or credentials, or even run more containers and laterally extend their grip on the systems.
TRC-109	ASLR inspection detected	The ASLR (address space layout randomization) configuration was inspected. ASLR is used by Linux to prevent memory vulnerabilities. An adversary may want to inspect and change the ASLR configuration in order to avoid detection.
TRC-1010	Cgroups release agent file	An attempt to modify Cgroup release agent file was detected. Cgroups are a Linux kernel feature which limits the resource usage of a set of

Second solution - profile

```
{
  "4026531840:/proc/1/root/usr/bin/bash:1620921980354639930": {
    "times": 1,
    "file_hash": "04a484f27a4b485b28451923605d9b528453d6c098a5a5112bec859fb5f2eea9"
  },
  "4026531840:/proc/1/root/usr/bin/curl:1620921980462639952": {
    "times": 20,
    "file_hash": "4deb8004d2a6999767df78ae441bb2fb3f95b63c265e80c96dcdfff98dc2e24a"
  },
  "4026531840:/proc/1/root/usr/bin/sleep:1620921980358639931": {
    "times": 20,
    "file_hash": "45cf3208dc6704e806bbc5d776e884b5487744bd75171a93930c94e9b9b20ebb"
  }
}
```

Stop and Check Tracee results and create a PR

```
1  ▶ Run simar7/tracee-action@ecdeeea7304dfa28c3c8ac4a84c26d079f63d20c
6  Stopping Tracee...
7  tracee-profiler
8  0
9  Checking results...
10 Differences found...
11 2c2,5
12 <   "4026531840:/proc/1/root/usr/bin/bash:1620921980354639930": {
13 ---
14 >   "/proc/1/root/usr/bin/bash": {
15 >     "mount_ns": 4026531840,
16 >     "time_stamp": 1620921980354639930,
17 >     "times": 1,
18 5,8c8,11
19 <   "4026531840:/proc/1/root/usr/bin/docker:1621826030596620581": {
20 <     "file_hash": "a03bd440d5955a104a2823638cf7f42be0f0b1e219ad442b32dab5fd601472e5"
21 <   },
22 <   "4026531840:/proc/1/root/usr/bin/sleep:1620921980358639931": {
23 ---
```

```yaml
name: My pipeline
jobs:
  my-job:
  runs-on: ubuntu-latest
  steps:
  - name: Start Tracee
    uses: aquasecurity/tracee-action@v0.3.0-start

  ...

  - name: Stop Tracee
    uses: aquasecurity/tracee-action@v0.3.0-stop
```

Closed **Updates to tracee profile** #28

Changes from all commits · File filter · Conversations · Jump to

412 .tracee/profile-exec.json

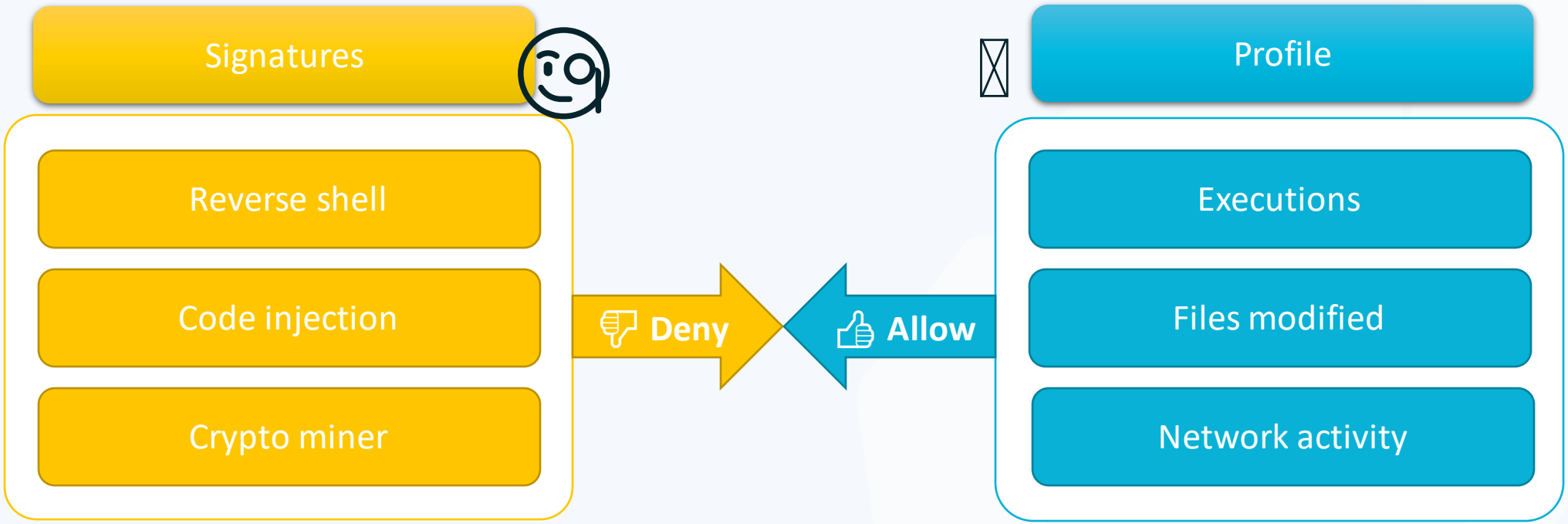
```diff
276      -  "JOURNAL_STREAM=8:16703",
277      -  "JOURNAL_STREAM=8:16703",
278      -  "JOURNAL_STREAM=8:16703",
     275 +  "JOURNAL_STREAM=8:16771",
     276 +  "JOURNAL_STREAM=8:16771",
     277 +  "JOURNAL_STREAM=8:16771",
     278 +  "JOURNAL_STREAM=8:16771",
279  279    "LANG=C.UTF-8",
280  280    "LANG=C.UTF-8",
281  281    "LANG=C.UTF-8",
@@ -336,9 +336,9 @@
336  336    "RUNNER_TOOL_CACHE=/opt/hostedtoolcache",
337  337    "RUNNER_TOOL_CACHE=/opt/hostedtoolcache",
338  338    "RUNNER_TOOL_CACHE=/opt/hostedtoolcache",
339      -  "RUNNER_TRACKING_ID=github_65067c3b-0f09-4c94-abd9-712342e462f2",
340      -  "RUNNER_TRACKING_ID=github_65067c3b-0f09-4c94-abd9-712342e462f2",
341      -  "RUNNER_TRACKING_ID=github_65067c3b-0f09-4c94-abd9-712342e462f2",
     339 +  "RUNNER_TRACKING_ID=github_8df4d698-943f-4154-be11-9d69233e8408",
     340 +  "RUNNER_TRACKING_ID=github_8df4d698-943f-4154-be11-9d69233e8408",
     341 +  "RUNNER_TRACKING_ID=github_8df4d698-943f-4154-be11-9d69233e8408",
342  342    "RUNNER_USER=runner",
343  343    "RUNNER_USER=runner",
344  344    "RUNNER_USER=runner",
@@ -363,9 +363,9 @@
363  363    "SWIFT_PATH=/usr/share/swift/usr/bin",
364  364    "SWIFT_PATH=/usr/share/swift/usr/bin",
365  365    "SWIFT_PATH=/usr/share/swift/usr/bin",
366      -  "SYSTEMD_EXEC_PID=666",
367      -  "SYSTEMD_EXEC_PID=666",
368      -  "SYSTEMD_EXEC_PID=666",
     366 +  "SYSTEMD_EXEC_PID=675",
     367 +  "SYSTEMD_EXEC_PID=675",
     368 +  "SYSTEMD_EXEC_PID=675",
369  369    "USER=runner",
370  370    "USER=runner",
371  371    "USER=runner",
@@ -497,9 +497,9 @@
497  497    "GITHUB_ACTIONS=true",
498  498    "GITHUB_ACTIONS=true",
499  499    "GITHUB_ACTIONS=true",
500      -  "GITHUB_ACTION_PATH=/home/runner/work/_actions/itaysk/tracee-action/5dc9ca5464345a09ff20a4ea3a92936f917ad223",
501      -  "GITHUB_ACTION_PATH=/home/runner/work/_actions/itaysk/tracee-action/5dc9ca5464345a09ff20a4ea3a92936f917ad223",
502      -  "GITHUB_ACTION_PATH=/home/runner/work/_actions/itaysk/tracee-action/5dc9ca5464345a09ff20a4ea3a92936f917ad223",
     500 +  "GITHUB_ACTION_PATH=/home/runner/work/_actions/itaysk/tracee-action/f5586125e6de0da26e0a2057eeb2d8a3fbe9ce6d",
     501 +  "GITHUB_ACTION_PATH=/home/runner/work/_actions/itaysk/tracee-action/f5586125e6de0da26e0a2057eeb2d8a3fbe9ce6d",
     502 +  "GITHUB_ACTION_PATH=/home/runner/work/_actions/itaysk/tracee-action/f5586125e6de0da26e0a2057eeb2d8a3fbe9ce6d",
503  503    "GITHUB_ACTION_REF=",
504  504    "GITHUB_ACTION_REF=",
505  505    "GITHUB_ACTION_REF=",
@@ -518,9 +518,9 @@
518  518    "GITHUB_BASE_REF=",
519  519    "GITHUB_BASE_REF=",
```

- Profile
  - ✗ Timestamp
  - ✓ User ID
  - ✓ Binary path
  - ✗ Process ID

itaysk

aqua

Third solution - both



DEMO

tracee-action

Executions

Signatures

- dynamic load
- code injection
- cron
- ld_preload
- default ld
- hidden exec
- rcd
- process_vm_writev
- kernel module
- /proc/mem
- fileless exec

Profile

- Binary Path
- Binary Hash
- Process UID
- Process Arguments
- Process Environment
- ~~Binary ctime~~

Triggers

- ~~execve~~
- sched_process_exec

ignore rules

- /home/runner/work/_temp/
- GITHUB_
- RUNNER_TRACKING_ID
- SYSTEMD_EXEC_PID
- INVOCATION_ID
- JOURNAL_STREAM

Files modified

Signatures

- ASLR inspection
- Proc kcore
- Core dump
- System request key
- Docker socket
- Proc mem
- sudoers
- sched_debug

Profile

- File Path
- Attributes*

Triggers

- ~~write~~
- Open write

ignore rules

- !GitHub Workspace

Network activity

Signatures

- Bare IP*
- Crypo miner
- Reverse shell
- SSH*

Profile

- Domain name

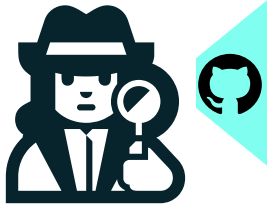
Triggers

- ~~network events~~
- DNS resolution

ignore rules

- pipelines.actions.githubusercontent.com

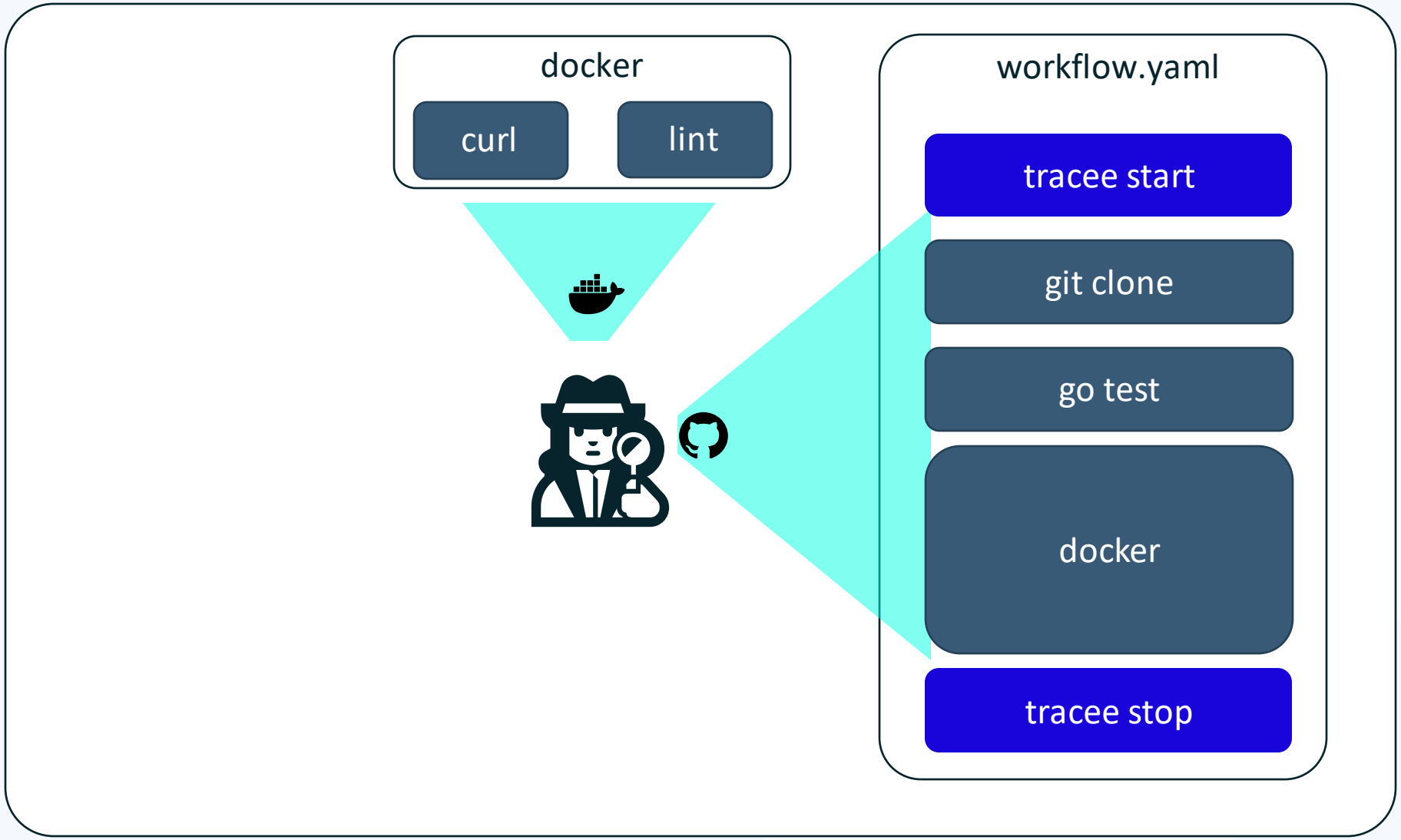
Scope



workflow.yaml

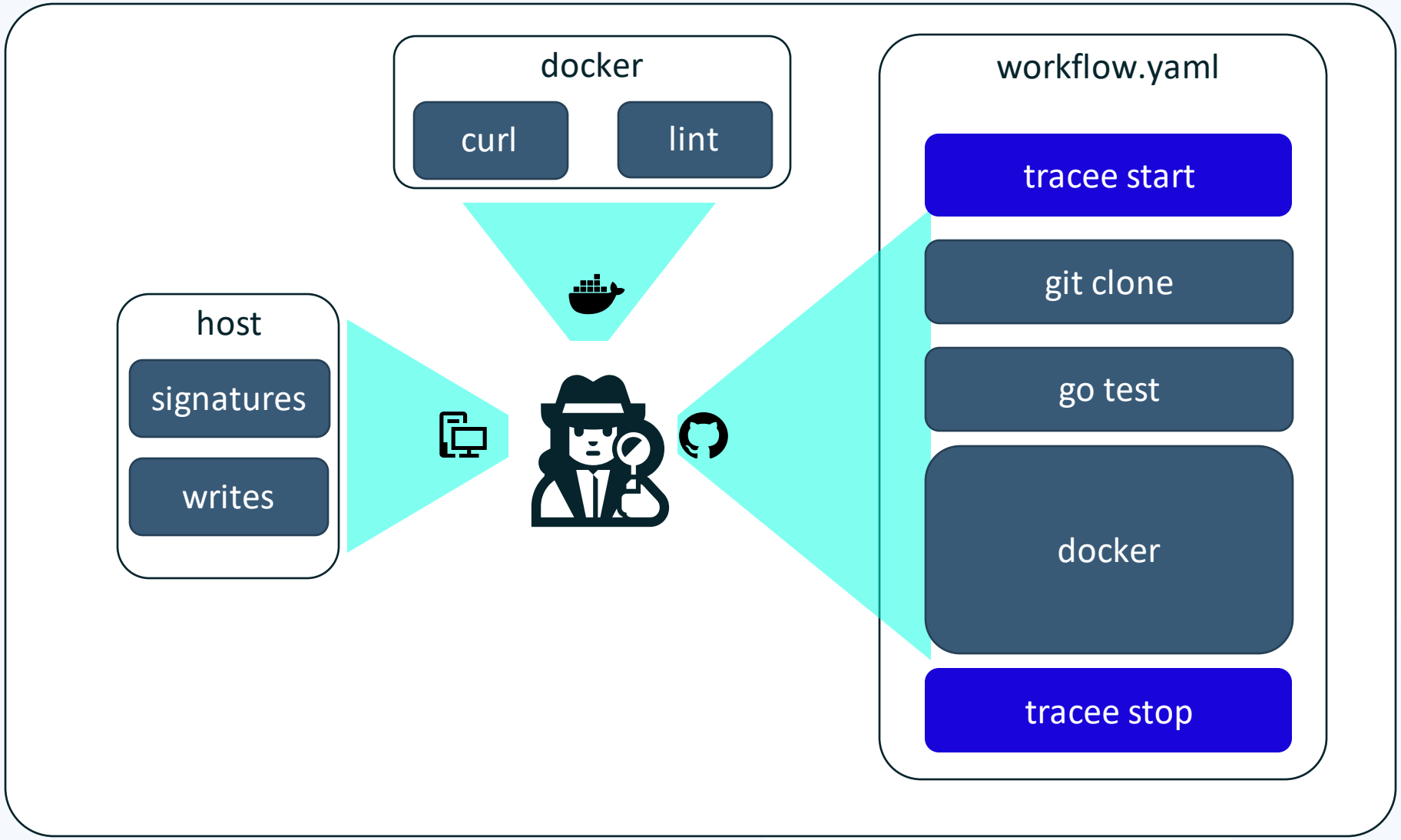
- tracee start
- git clone
- go test
- docker
- tracee stop

Scope


docker
curl
lint
workflow.yaml
tracee start
git clone
go test
docker
tracee stop

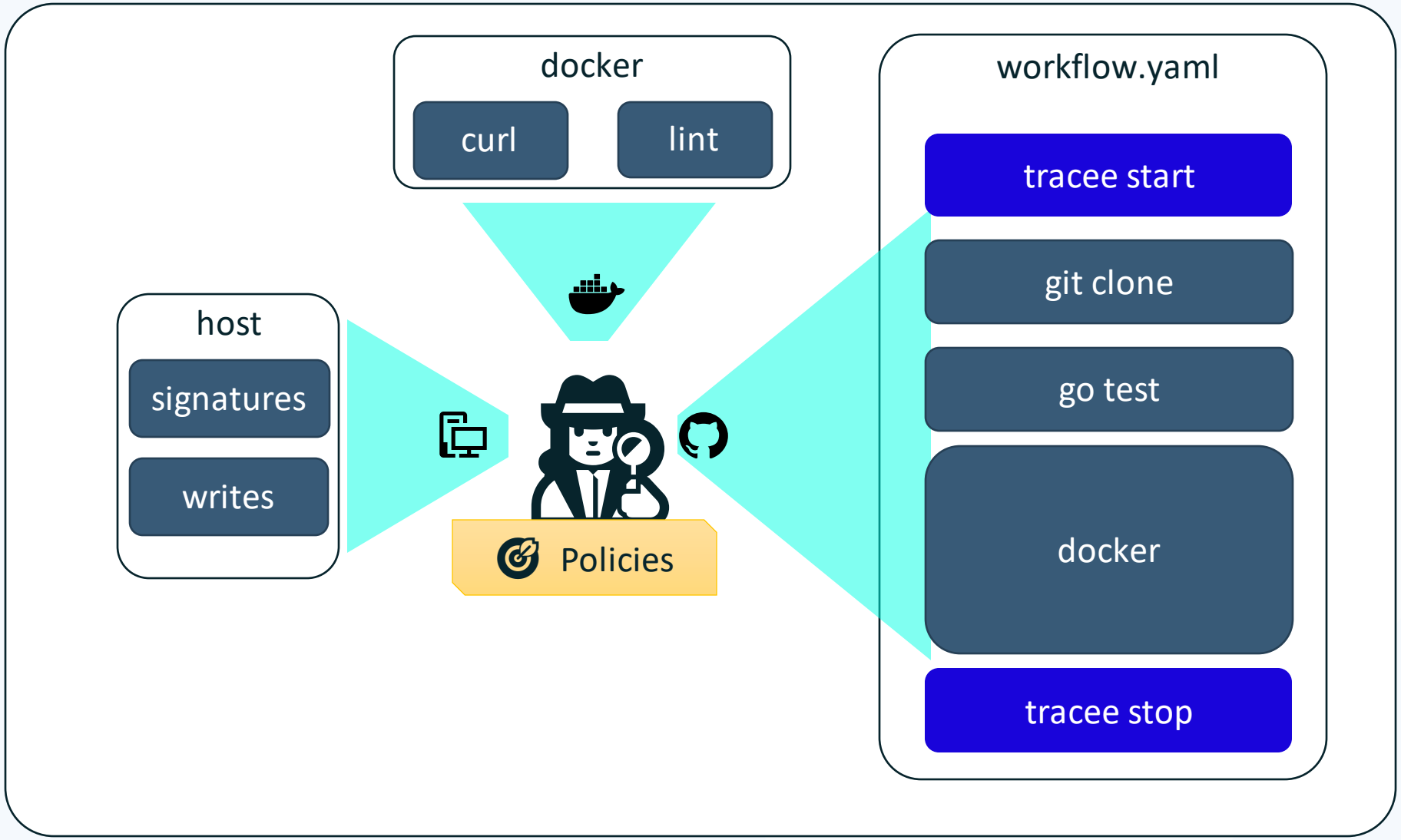
aqua

Scope


docker
curl
lint
workflow.yaml
tracee start
git clone
go test
docker
tracee stop
host
signatures
writes

aqua

Scope


docker
curl
lint
workflow.yaml
tracee start
git clone
go test
docker
tracee stop
host
signatures
writes
Policies

aqua

Runner process tree - policy

```
name: runner
description: trace runner process tree
defaultAction: log
scope:
- tree=$RUNNER_PID
rules:
- event: sched_process_exec
- event: net_packet_dns
```

Scope

Events

Container build - policy

```
name: container_build
description: trace container build process tree
defaultAction: log
scope:
- binary=/usr/bin/containerd-shim-runc-v2
- follow
rules:
- event: sched_process_exec
- event: net_packet_dns
```

Scope

Events

aqua

Signatures policy

Scope

Events

```
name: signatures
description: trace signature events
defaultAction: log
scope:
- global
rules:
- event: fileless_execution
- event: illegitimate_shell
- event: proc_mem_code_injection
- event: docker_abuse
- event: scheduled_task_mod
- event: ld_preload
- event: cgroup_notify_on_release
- event: default_loader_mod
- event: sudoers_modification
- event: sched_debug_recon
- event: proc_kcore_read
- event: proc_mem_access
- event: hidden_file_created
- event: anti_debugging
- event: ptrace_code_injection
...
```

File write – new event

```
package tracee.TRC_1001
import data.tracee.helpers

__rego_metadoc__ := {
"id": "TRC_1001",
"version": "0.1.0",
"eventName": "file_write",
"description": "A file is being written to",
}

tracee_selected_events[eventSelector] {
  eventSelector := {
    "source": "tracee",
    "name": "security_file_open",
  }
}

tracee_match = res {
  helpers.is_file_write(helpers.get_tracee_argument("flags"))
  pathname := helpers.get_tracee_argument("pathname")
  res := {
    "pathname": pathname,
  }
}
```

File write - policy

Scope

Event

```
name: file_writes
description: tracee file writes under {{github.workspace}}
defaultAction: log
scope:
- global
rules:
- event: file_write
  filter:
    - args.pathname=$WORKSPACE
```


aqua

Provenance attestation

"[Provenance is] the verifiable information about software artifacts describing where, when and **how** something was produced." – SLSA

"The runtime trace can prove the build was invoked via a script, that the build was executed in a hermetic environment with no network access, and so on." - adityasaki

Schema

```
{
    "_type": "https://in-toto.io/Statement/v1",
    "subject": [{ ... }],
    "predicateType": "https://in-toto.io/attestation/runtime-trace/v0.1",
    "predicate": {
        "monitor": {
            "type": "<TypeURI>",
            "configSource": "<ResourceDescriptor>",
            "tracePolicy": { /* object */ }
        },
        "monitoredProcess": {
            "hostID": "<URI>",
            "type": "<URI>",
            "event": "<STRING>"
        },
        "monitorLog": {
            "process": [
                { /* object */ }
            ],
            "network": [
                { /* object */ }
            ],
            "fileAccess": ["<ResourceDescriptor>", ...]
        },
        "metadata": {
            "buildStartedOn": "<TIMESTAMP>",
            "buildFinishedOn": "<TIMESTAMP>"
        }
    }
}
```

https://github.com/in-toto/attestation/blob/main/spec/predicates/runtime-trace.md

Lessons learned

- Runtime is not buildtime
- Profiles are volatile
- Signatures = deny, profile = allow
- How to write portable eBPF programs
- Trace tools might have blindspots
- Avoid noise with contextual tracing
- System call tracing is problematic
- Process arguments are important, but adds flakiness
- Environment variables might leak secrets
- Sometimes trace intention instead of the activity
- Using profile as attestation?

aqua

Resources

- aquasecurity/tracee
- aquasecurity/tracee-action
- AquaTracee
- itaysk
- josedonizetti

aqua

Verfiable GitHub Actions using eBPF

Itay Shakury itaysk

Jose Donizetti josedonizetti