KCD Austria
2024

Detecting unexpected behavior and intrusions with Falco + Atomic Red Team

WORKSHOP

PRESENTER

Miguel Hernández Boza

Sr. Threat Research Engineer

PRESENTER

Vicente J. Jiménez Miras

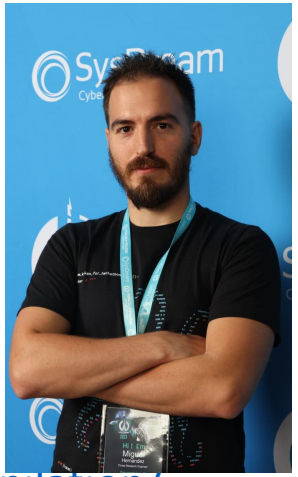
Independent Technical Trainer

Austria

KCD

GROWING CLOUD NATIVE TOGETHER

WHOAMI - Miguel



- +10 years in cybersecurity
  - Fraud detection, OSINT, ML Security, Cloud native Security.
- Speaker at cybersecurity conferences
  - HITB, HIP, CCN-CERT, RootedCon, TheStandoff, Codemotion…
- Open-Source
  - Grafscan: https://github.com/Miguel000/GraFScaN
  - Spyscrap: https://github.com/RuthGnz/SpyScrap
  - Offensive-ai-compilation: https://jiep.github.io/offensive-ai-compilation/

- https://www.linkedin.com/in/miguelhzbz/

WHOAMI - Vicente

- Different roles in IT through +20 years experience
  - System Administrator, System Engineer, Content Engineer, Instructor.
- Red Hat Certified Instructor and Examiner
  - RHEL, Ansible, Openshift Container Platform, Linux Security, etc.
- International working experience
  - Spain, Latin America, Germany.
- Passion for Cybersecurity
  - Hardening, Penetration Testing, Red Team, CTFs, etc.

Vicente J. Jiménez

Open source evangelist

Vicente J. Jiménez Miras has a background as an infrastructure engineer and open source trainer. Being a digital nomad since early in his career, he's lived or worked in almost a hundred different locations. He also shares a passion for food, especially Indian cuisine, although, as a Spaniard, he's very proud of his tortilla española.

In 2021, after 4 years working as an instructor for Red Hat, he joined Sysdig to continue his work educating technical teams and creating awareness of open source cybersecurity.

Logistics

Workshop: https://play.instruqt.com/sysdig/invite/ealaxretrxwq

https://bit.ly/4dvL1GQ

Length: 180 min

QR-Code:

Agenda

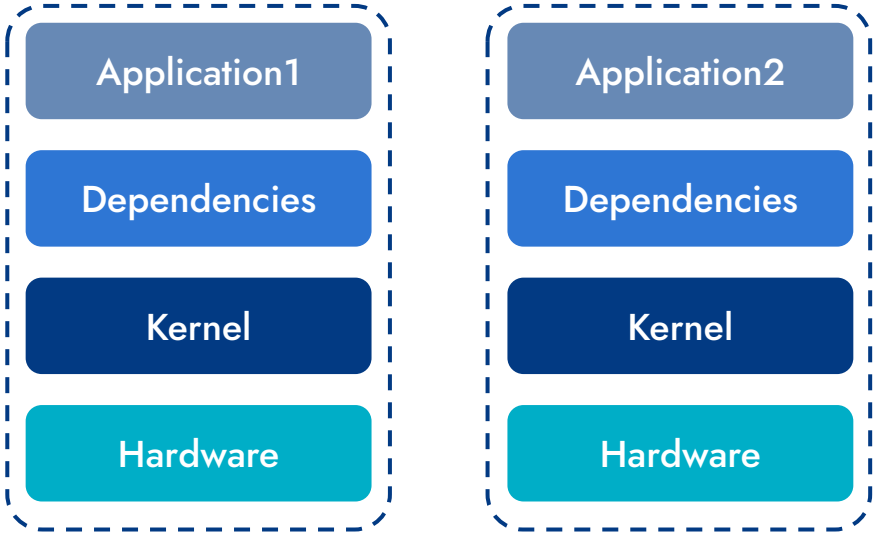
- Cloud Threat Detection and Runtime Security
- Falco engine
- Falco ecosystem
- Atomic Red Team
- Closing remarks

Cloud Threat Detection and Runtime Security

Microservices, Containers, and Kubernetes

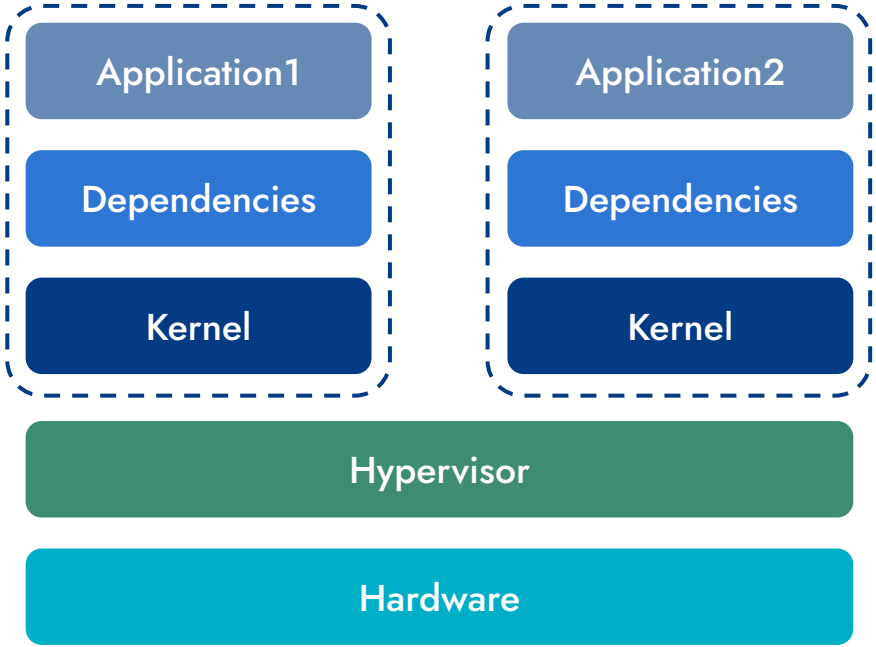
Once upon a time… dedicated servers

- Deployments took months
- Low utilization
- Not portable
- Hard to replicate



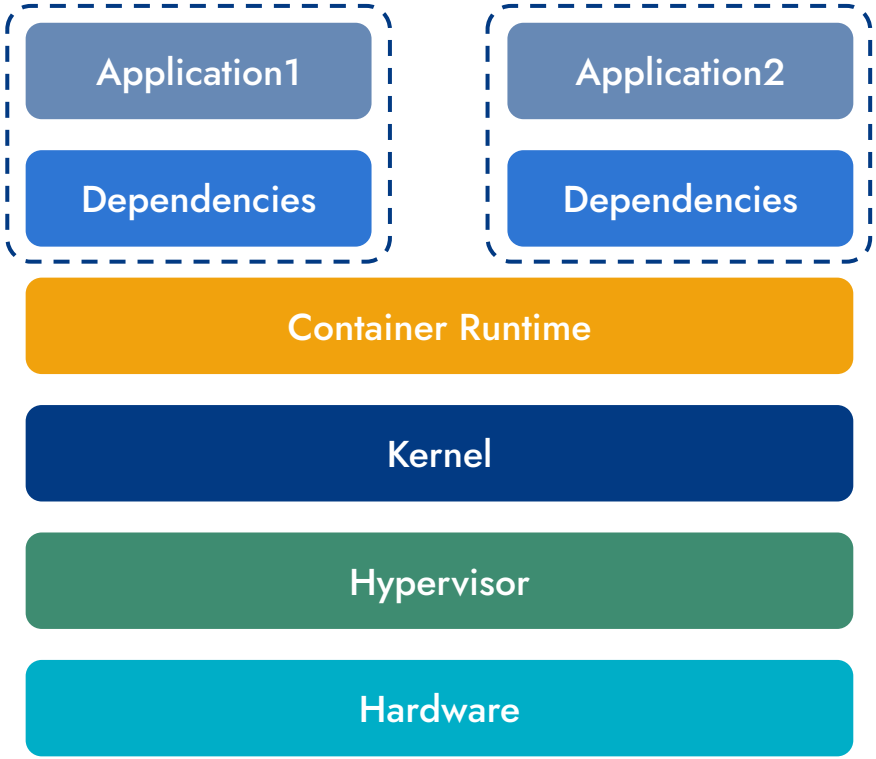
Virtualization

- Improved utilization
- Deployment took days
- Redundant kernel
- All bundled together
- Low app <--> kernel isolation



Containers

- Lightweight
- Stand-alone
- Resource efficient
- Portable execution packages
- Deploy within minutes/seconds



Rise of microservices and containers

- Need for greater flexibility, agility, scalability, and reliability
  - loosely coupled, fine-grained components.
- Microservices make it easier to:
  - build and maintain complex applications
  - scale individual services as needed, not scaling the entire system
  - adopt new technologies and tools as they become available.
  - implement changes and updates, without affecting the entire system
- Containers are a perfect fit.
- Cloud providers heavily support containers and microservices
  - easier and more cost-effective to build and scale applications.
- Containers are everywhere, but managing them at scale is challenging!

Kubernetes wins

- Kubernetes is an open-source container orchestration system
  - for automating software deployment, scaling, and management.
- Over 60% of orgs have adopted it already in 2022 (Statista research).
- Originally designed by Google,
  - the project is now maintained by the Cloud Native Computing Foundation.

CNCF - Cloud Native Computing Foundation

- Founded in 2015 to advance container technology and align the tech industry around its evolution.
- Announced alongside Kubernetes 1.0
- Part of the Linux Foundation.
- Projects have a maturity level: Sandbox, Incubated, and Graduated

What is Cloud Native?

- Cloud computing environments
- Highly scalable, flexible, and resilient applications (quick updates)
- Modern tools and techniques that support development on cloud infrastructure.
- Fast and frequent changes to applications without impacting service delivery

Cloud Threat Detection and Runtime Security

RUNTIME SECURITY

Once, there was a perimeter

You had a perimeter **guarded by a firewall**

Detecting intrusions was your breach indicator

Now, there is no perimeter in the cloud

- Cloud providers own external connections
- Cloud is exposed to the outside world
- You need to control access to services your team uses
- You need to detect unusual activity

Why Runtime Security?

Runtime Security vs. other Security Approaches

- Runtime behavior vs config checks or static entities (CIEM, VM, CSPM)
- Strict inventories (libs, vulns, configs) vs pattern analysis
- “Expected attacks” vs Unexpected patterns
- It is not vs, but +

Example

- Log4j vulnerability was introduced in 2013.
- Reported and fixed: 2022.

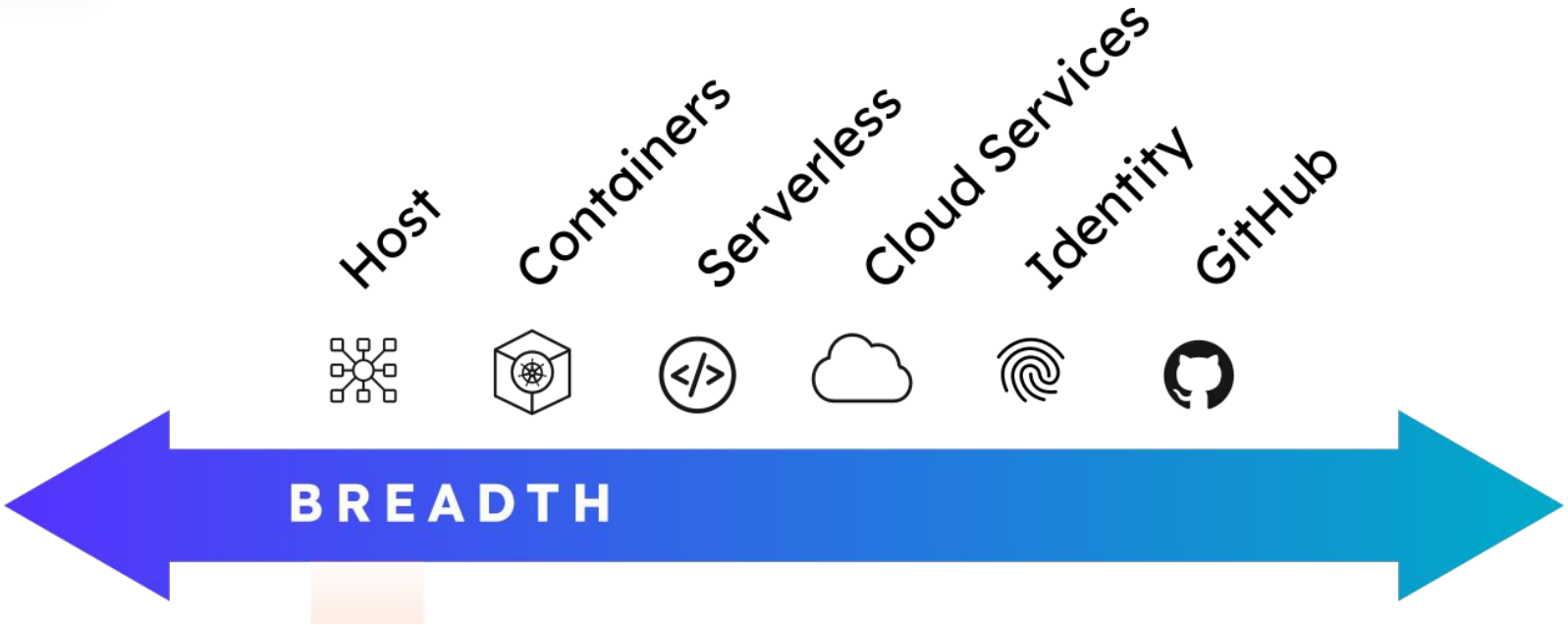
Prepare for the unexpected:

- **Runtime Nature of Attacks**
- **Zero-day Vulnerabilities**

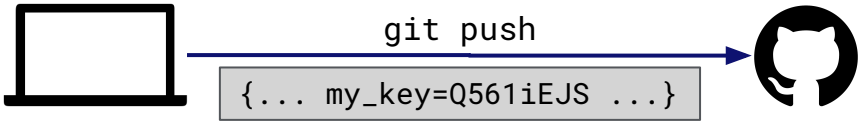
Without a perimeter, a security camera is more important than a good lock

- Watch for changes that create security gaps
- Identify intruders and suspicious insider behavior
- Send an alert and take immediate action

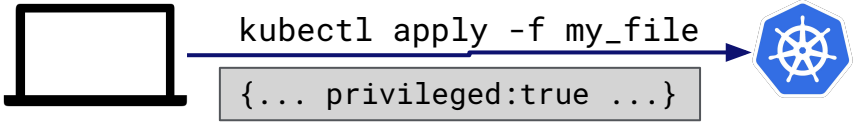
End-to-End Detection



Suspicious activities

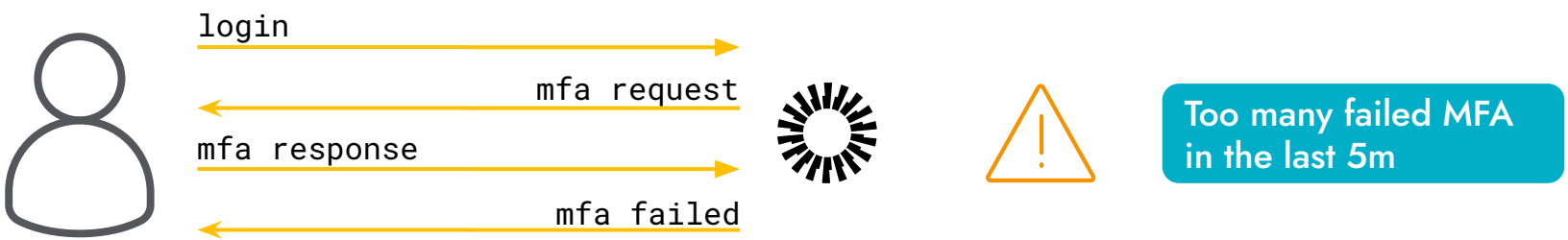


Secret pushed into a public repository.

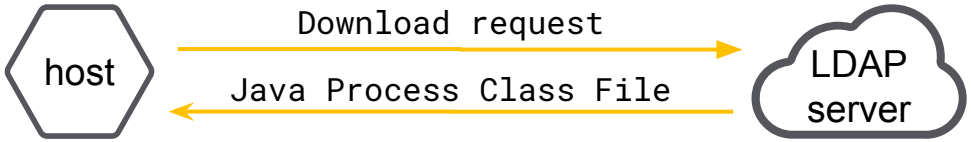


Create privileged pod.

Suspicious activities

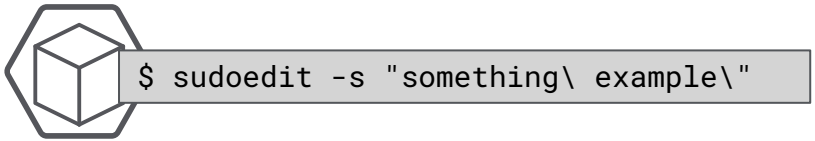


Suspicious activities



Java Process Class File Download

CVE-2021-4422



Sudo Potential Privilege Escalation

CVE-2021-3156

FALCO

v

Falco

Falco is an **open source** runtime security solution for **threat detection** across **Kubernetes**, containers, hosts and **the cloud**.

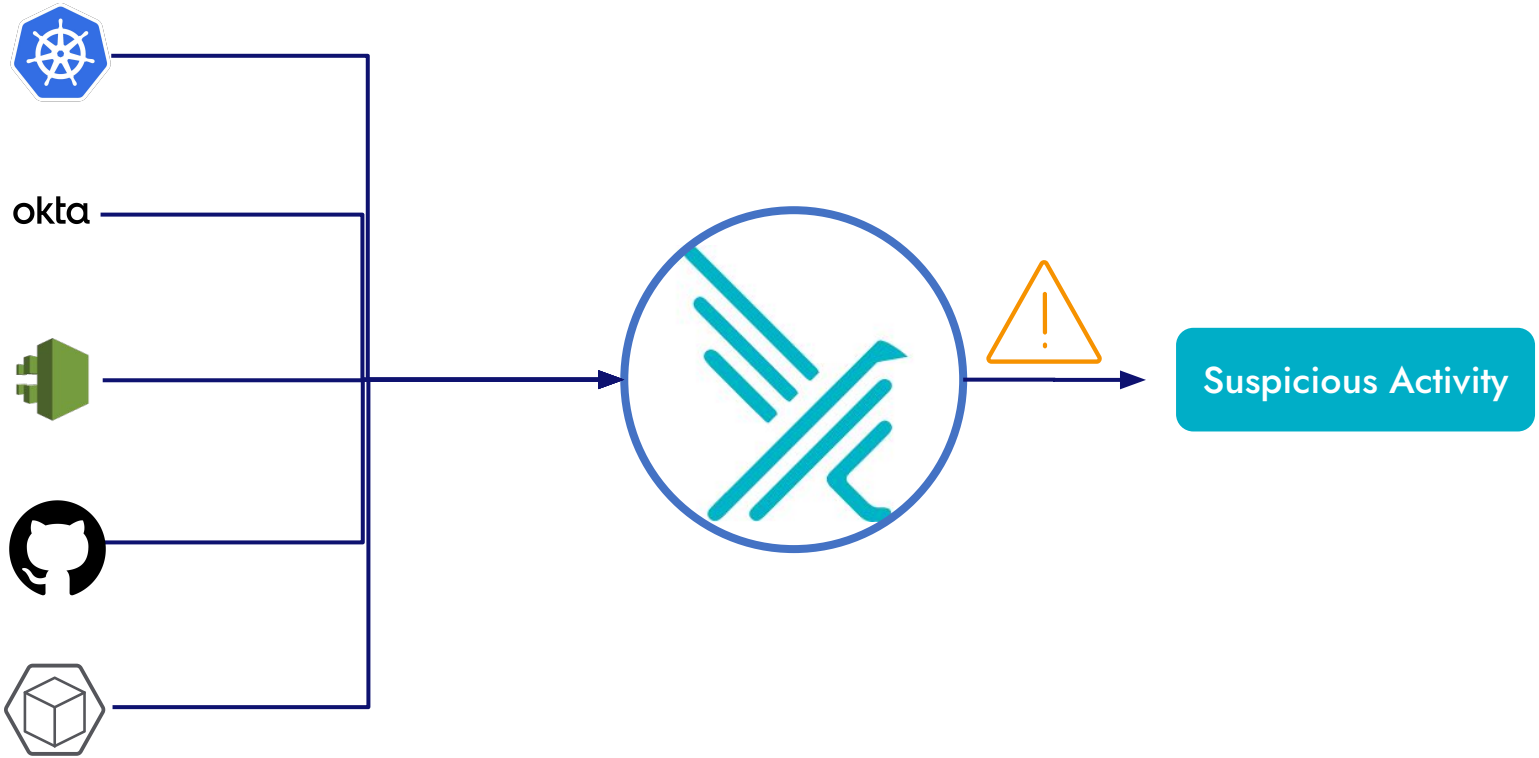
Graduated maturity level on February 29, 2024.

7.3k

60M+ pulls

CLOUD NATIVE COMPUTING FOUNDATION

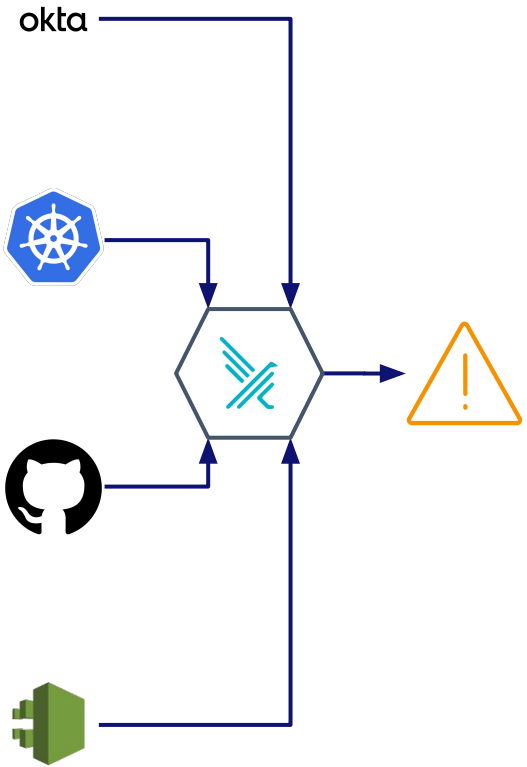
Falco Overview

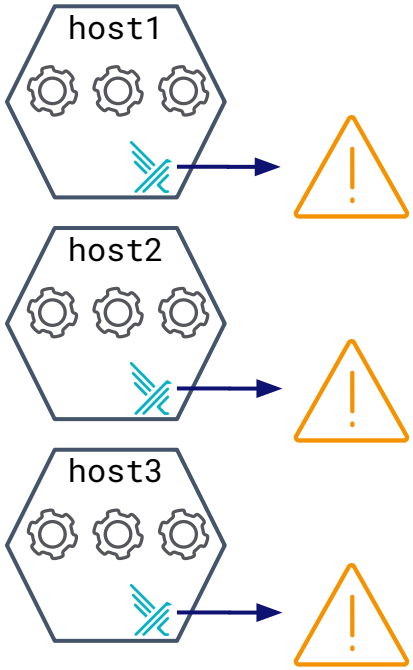


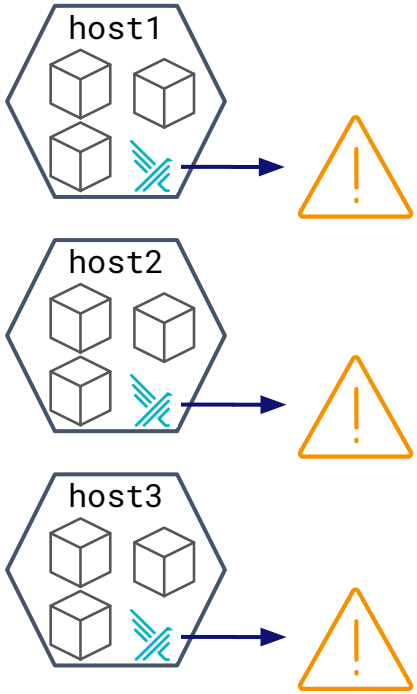
Falco Alert

```
2022-04-07T12:51:08: Notice A shell was spawned in a container with an attached terminal (user=root
user_loginuid=-1 elastic_borg (id=a10bd3b1b2a8) shell=bash parent=<NA> cmdline=bash terminal=34816
container_id=a10bd3b1b2a8 image=ubuntu)
2022-04-07T12:51:41: Warning Netcat runs inside container that allows remote code execution
(user=root user_loginuid=-1 command=nc -e container_id=a10bd3b1b2a8 container_name=elastic_borg
image=ubuntu:latest)
```

Falco deployment examples







Set up Falco

LAB TIME!

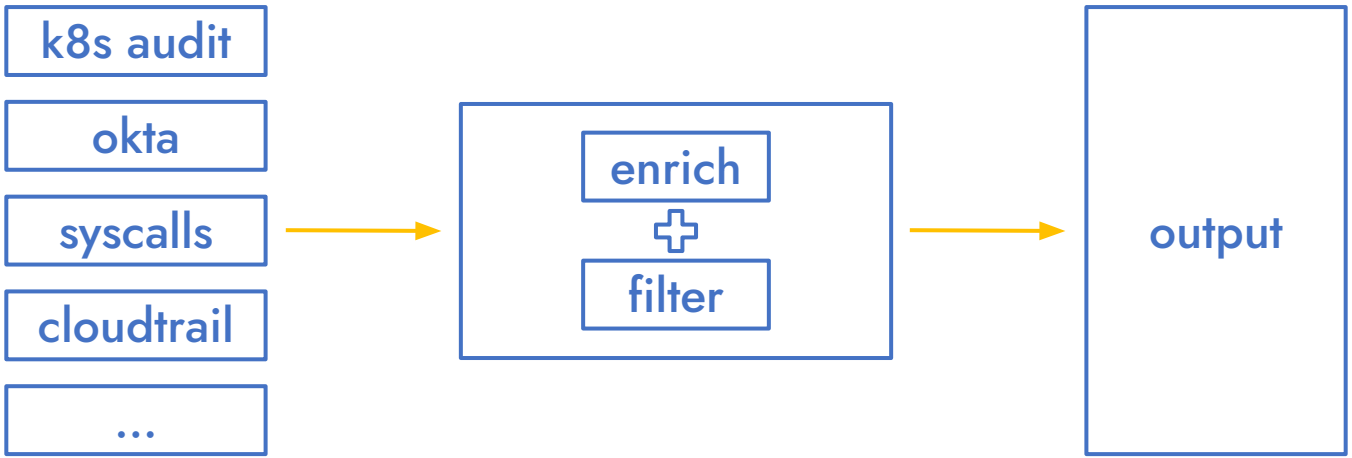
FALCO ENGINE

v

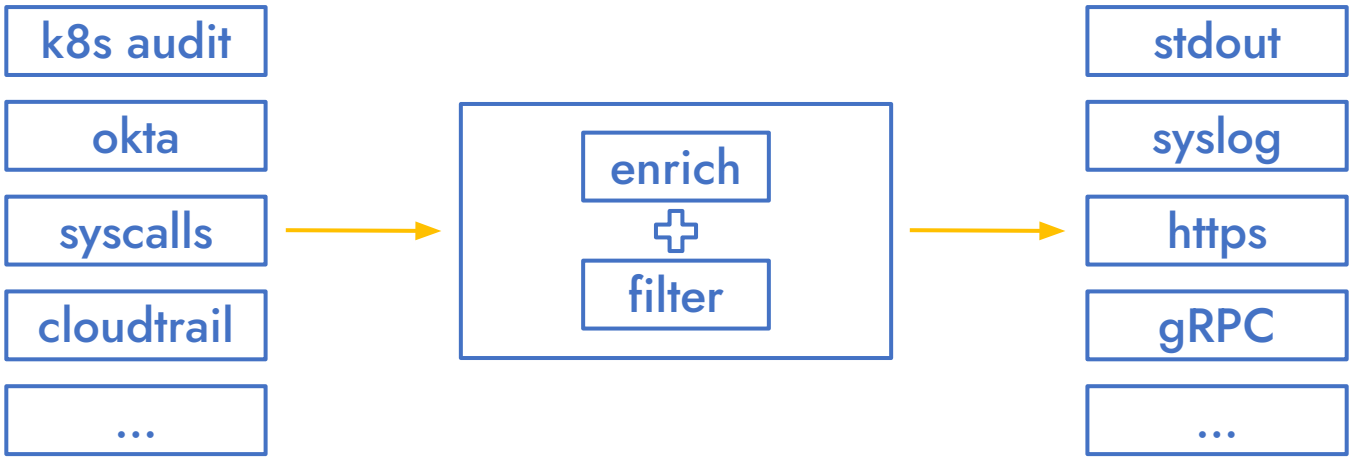
Falco architecture

Falco architecture

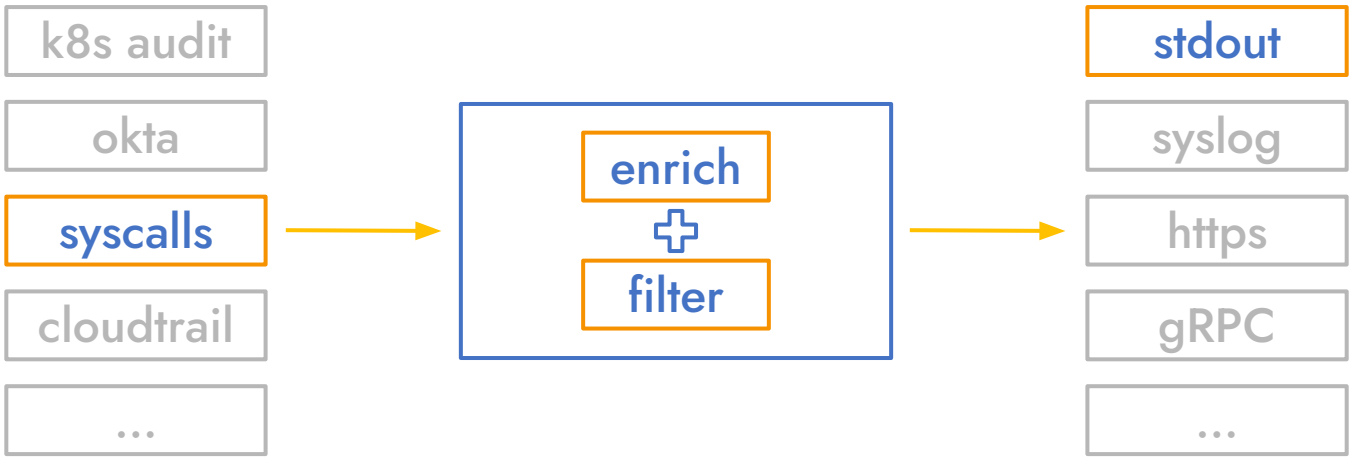
Falco architecture



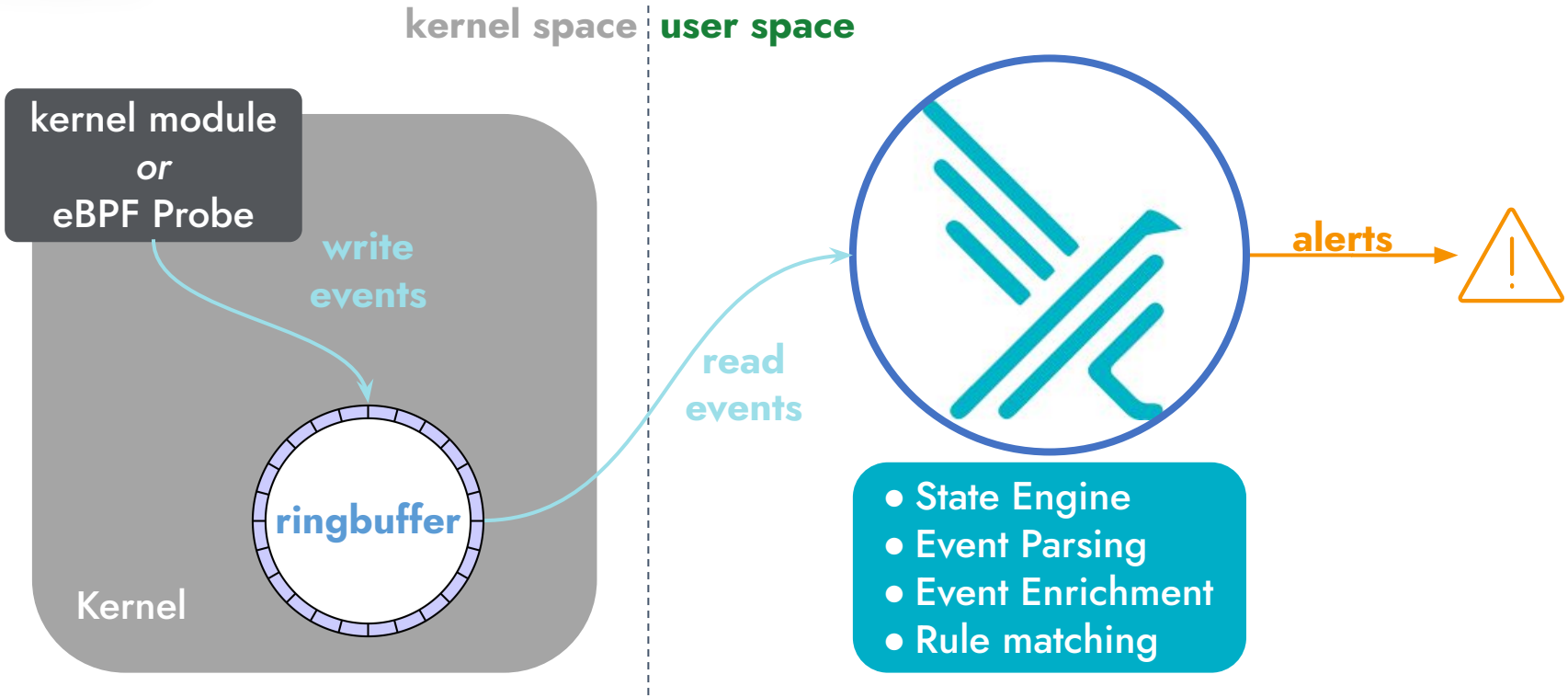
Falco architecture



Falco architecture



Falco Syscall Architecture Overview



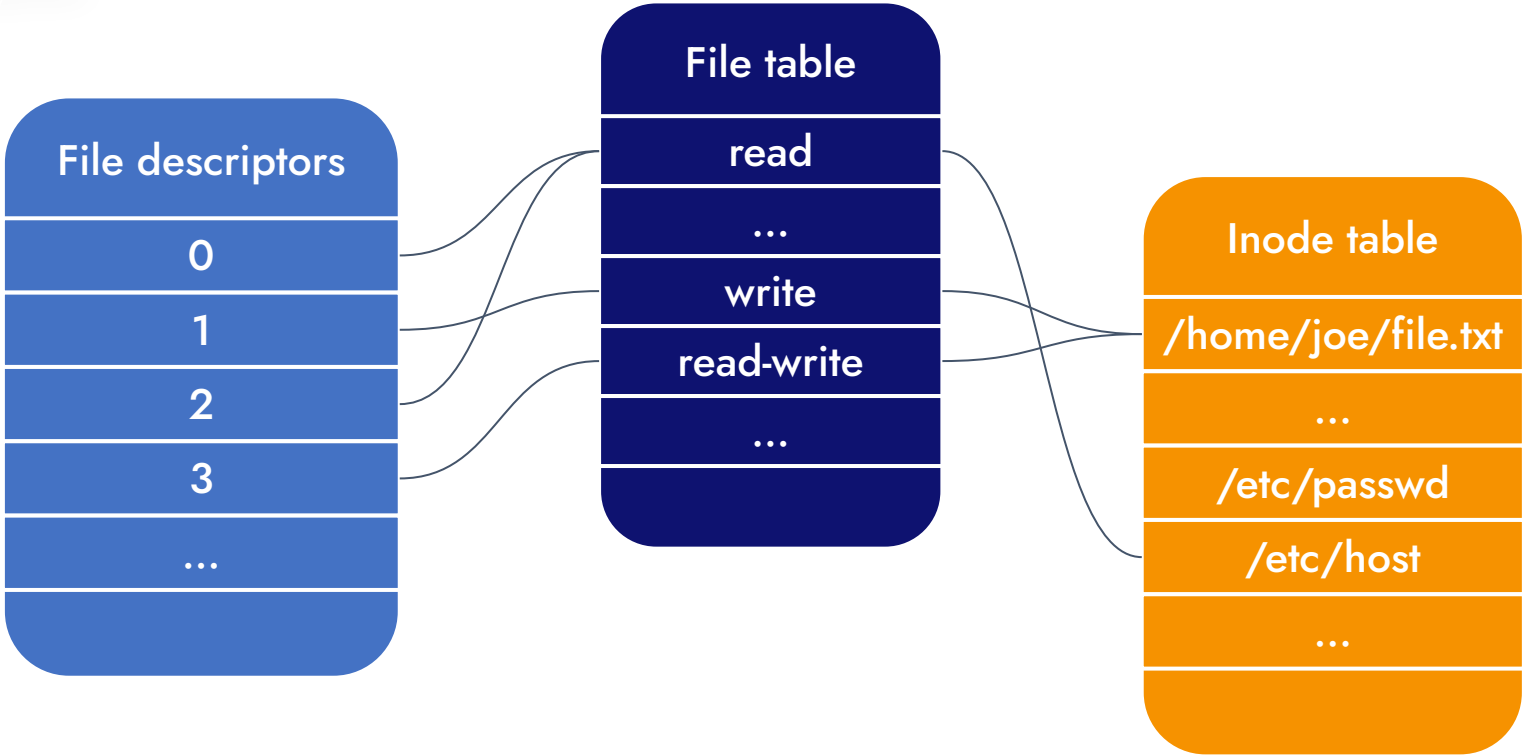
eBPF

- Extended Berkeley Packet Filter.
- Extend the kernel capabilities safely and efficiently.
  - without changing kernel source code.
  - without loading kernel modules.

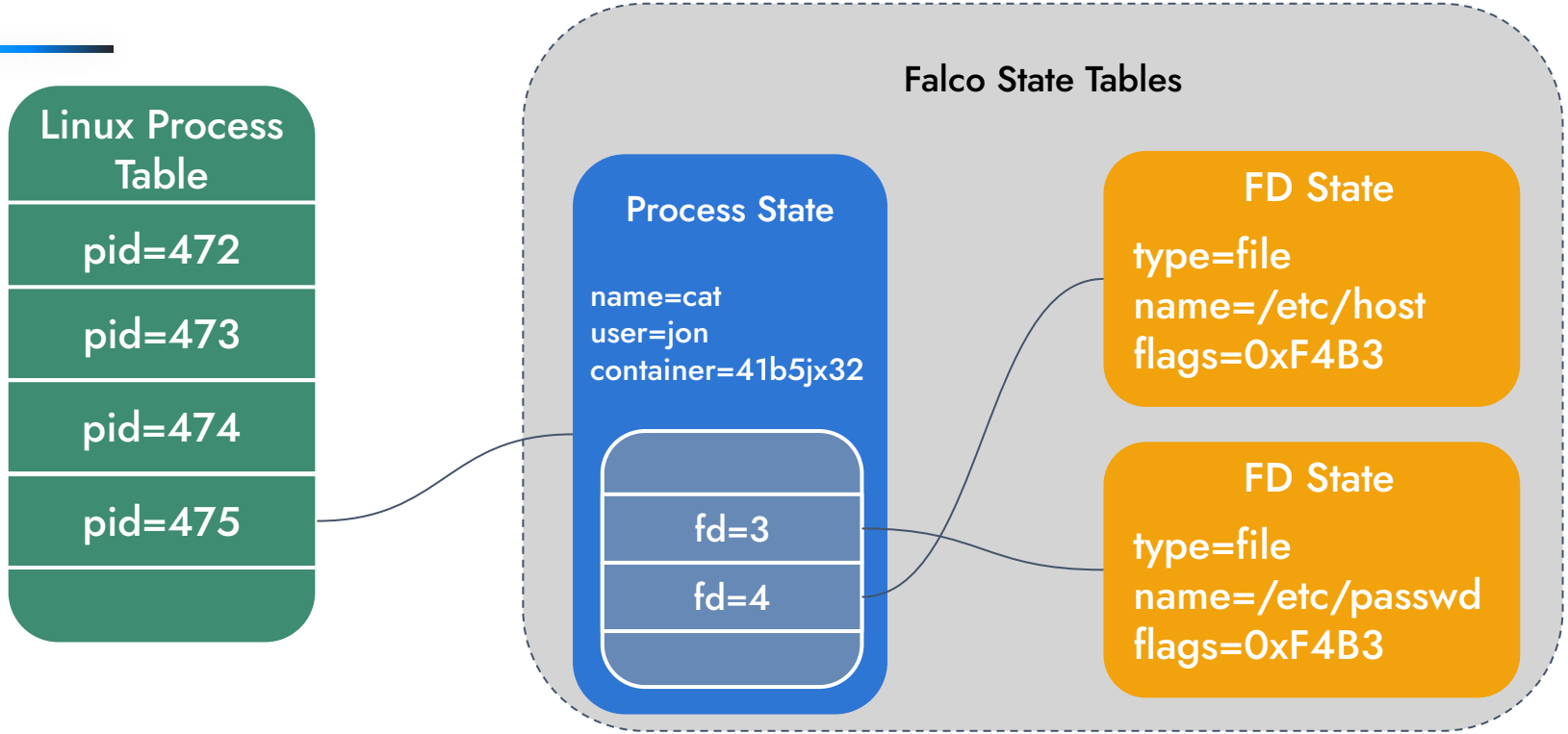
Modern, low-overhead, production-ready observability for monitoring and security in containers and Linux systems.

Kernel instrumentation made simple!

Navigating the Linux File System



Falco state


Linux Process Table
pid=472
pid=473
pid=474
pid=475
Falco State Tables
Process State
name=cat
user=jon
container=41b5jx32
fd=3
fd=4
FD State
type=file
name=/etc/host
flags=0xF4B3
FD State
type=file
name=/etc/passwd
flags=0xF4B3

Falco default rules

- Falco ships with more than 80 default rules:
  - Privilege escalation
  - R/W to sensitive directories
  - Executing shell
  - Execute SSH binaries
  - Mutating binaries
  - Creating symlinks
  - …

Falco syntax

```
- rule: Terminal shell in container
  desc: A shell has been spawned in a container.
  condition: >
    spawned_process and container and shell_procs
  output: >
    A shell was spawned in a container (user=%user.name
user_loginuid=%user.loginuid %container.info shell=%proc.name
parent=%proc.pname cmdline=%proc.cmdline container_id=%container.id)
```

https://falco.org/docs/reference/rules/rule-fields/

Falco syntax

```
- rule: Terminal shell in container
  desc: A shell has been spawned in
  condition: >
    spawned_process and container an
  output: >
    A shell was spawned in a contain
user_loginuid=%user.loginuid %contai
parent=%proc.pname cmdline=%proc.cmd
```

```
- list: shell_binaries
  items: [ash, bash, csh, ksh, sh,
tcsh, zsh, dash]

- macro: shell_procs
  condition: proc.name in
(shell_binaries)

- macro: container
  condition: (container.id != host)

- macro: spawned_process
  condition: >
        evt.type in (execve, execveat)
        and evt.dir=<
```

https://falco.org/docs/reference/rules/rule-fields/

Macros and Lists

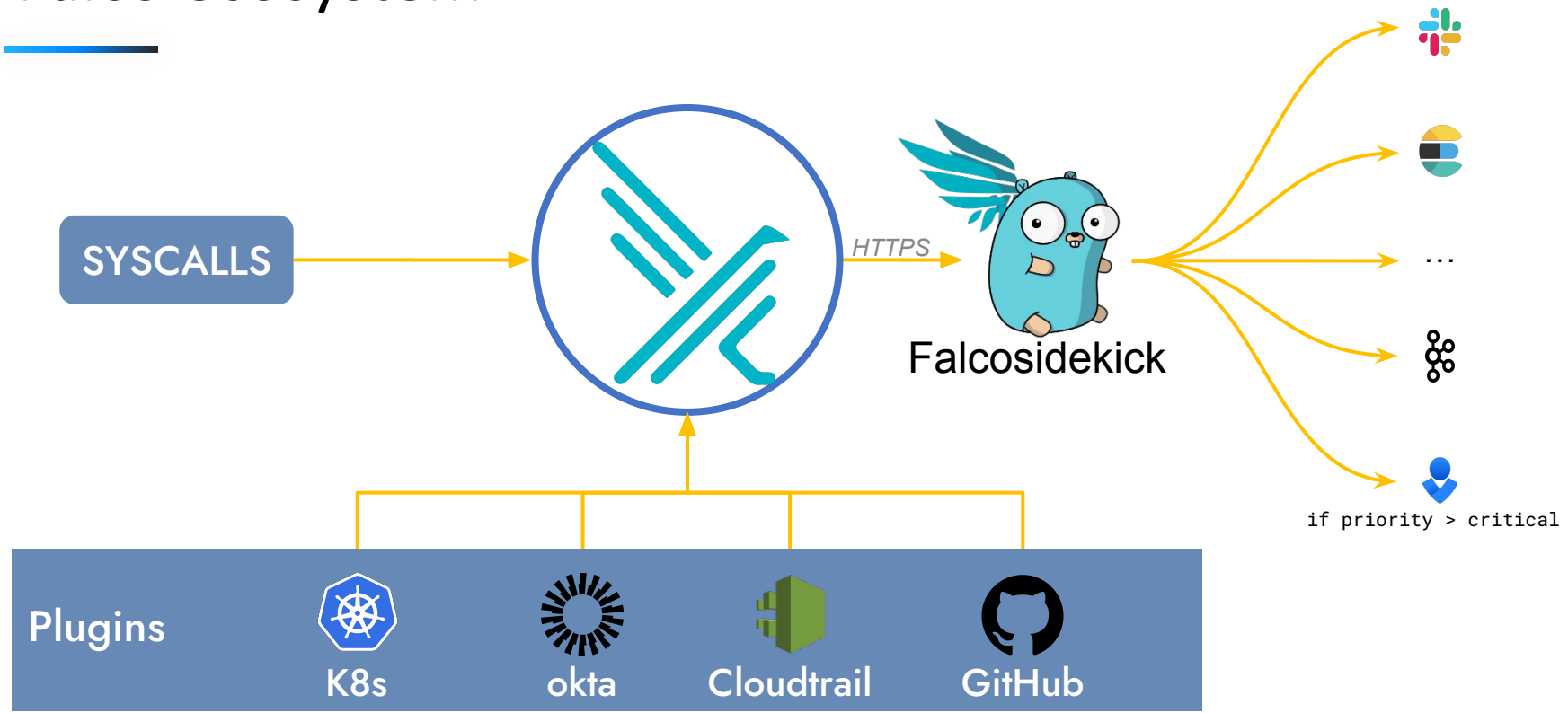
- **Macros** allow you to define conditions and reuse them wherever you want.
- **Lists** help you organize your rules files with naming and segmentation.
- They have four main benefits:
  - code reuse
  - avoid long strings of conditions
  - rules are easier to understand
  - easier to extend

Set up Falco

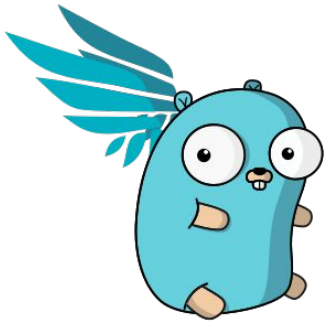
LAB TIME!

FALCO ECOSYSTEM

Falco ecosystem



Falcosidekick



chat

logs

queue/streaming

faas

metrics

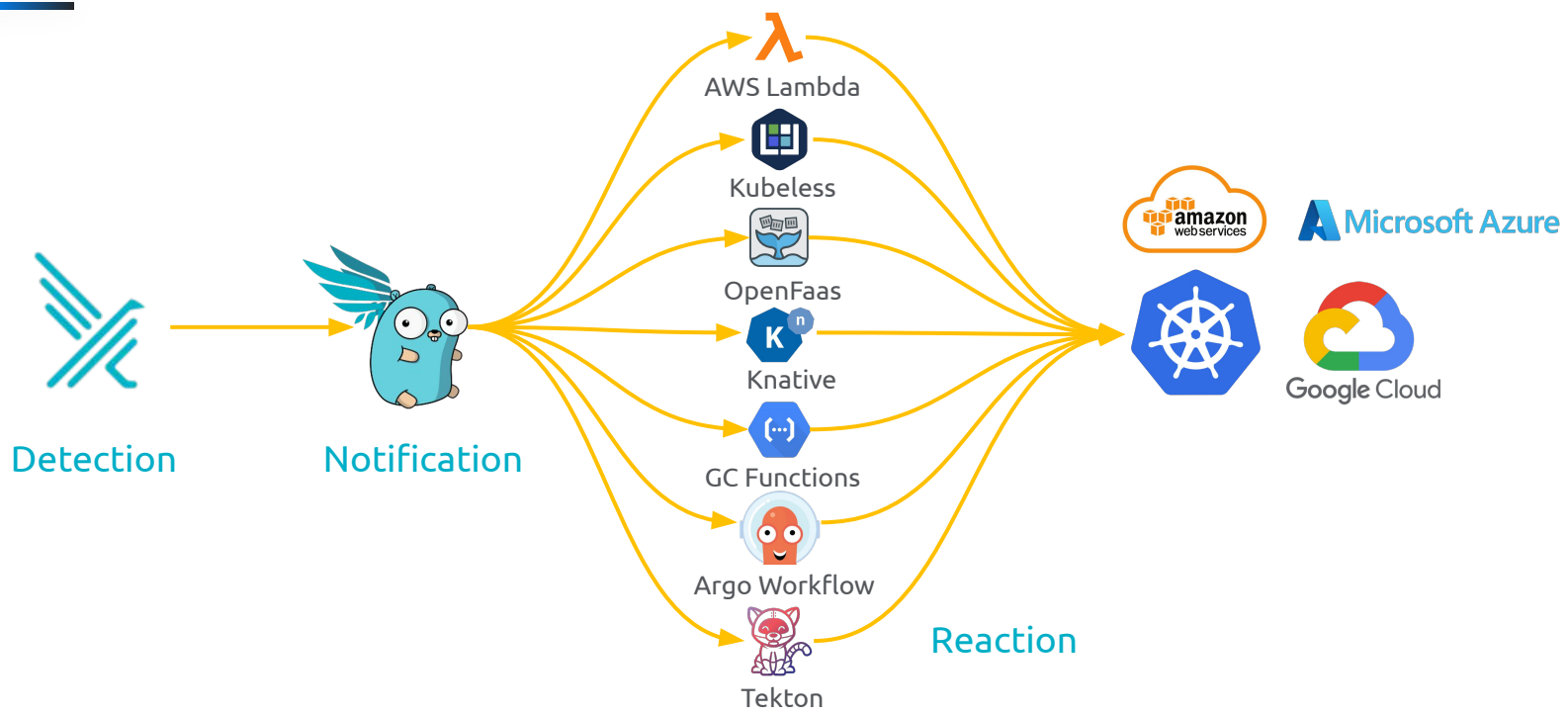
alerting

storage

and more ...

https://github.com/falcosecurity/falcosidekick

Event management



Falcosidekick install

Standalone:

```
helm repo add falcosecurity https://falcosecurity.github.io/charts
helm repo update

helm install falcosidekick -n falco --set config.debug=true falcosecurity/falcosidekick
```

With Falco:

```
helm install falco -n falco falcosecurity/falco \
  --set falcosidekick.enabled=true \
  --set falcosidekick.webui.enabled=true
```

Event generator

- Generates a variety of suspect actions that are detected by Falco rulesets.
- Good to test Falco rulesets:
  - syscalls
  - kubernetes audit
- Run it within Docker or Kubernetes,
  - as some commands might alter your system.

```
docker run -it --rm falcosecurity/event-generator run
```

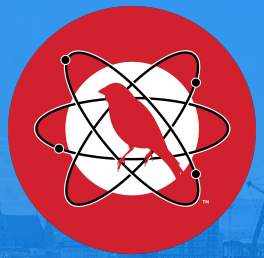
```
helm install event-generator falcosecurity/event-generator \
  --namespace event-generator \
  --create-namespace \
  --set config.actions=""
```

https://github.com/falcosecurity/event-generator

Set up Falco

LAB TIME!

Atomic Red Team



The Atomic Red Team Project

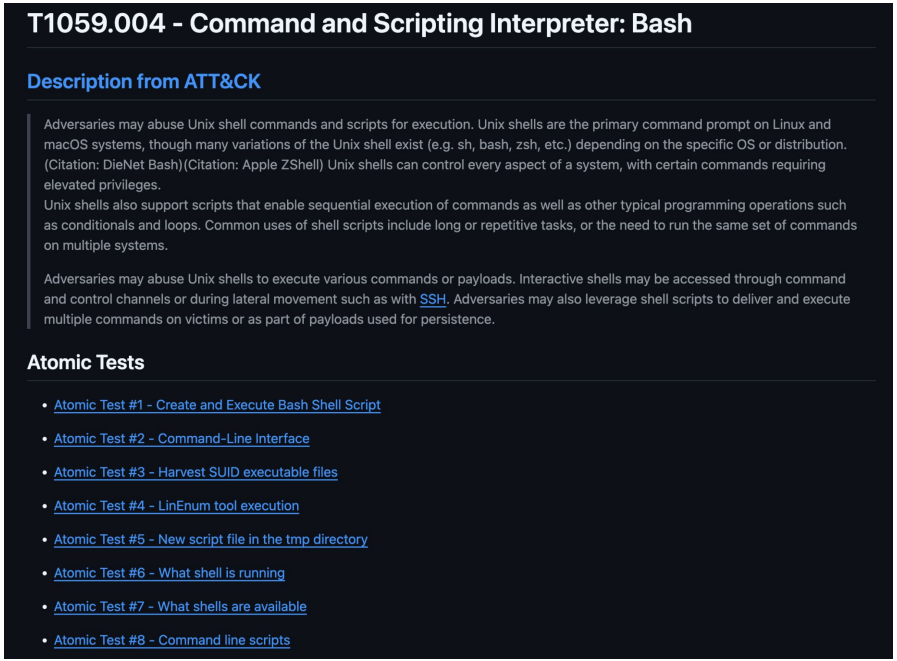
- Library of Scripted Attacks
- Started by Red Canary in 2017
- Free and Open Source
- Community Developed (over 190 contributors)

MITRE ATT&CK

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Exfiltration	Command And Control
10 items	31 items	56 items	28 items	59 items	20 items	19 items	17 items	13 items	9 items	21 items
Drive-by Compromise	AppleScript	.bash_profile and .bashrc	Access Token Manipulation	Access Token Manipulation	Account Manipulation	Account Discovery	AppleScript	Audio Capture	Automated Exfiltration	Commonly Used Port
Exploit Public-Facing Application	CMSTP	Accessibility Features	Accessibility Features	Binary Padding	Bash History	Application Window Discovery	Application Deployment Software	Automated Collection	Data Compressed	Communication Through Removable Media
Hardware Additions	Command-Line Interface	AppCert DLLs	AppCert DLLs	BITS Jobs	Brute Force	Browser Bookmark Discovery	Distributed Component Object Model	Clipboard Data	Data Encrypted	Connection Proxy
Replication Through Removable Media	Control Panel Items	AppInit DLLs	AppInit DLLs	Bypass User Account Control	Credential Dumping	File and Directory Discovery	Exploitation of Remote Services	Data from Information Repositories	Data Transfer Size Limits	Custom Command and Control Protocol
Spearphishing Attachment	Dynamic Data Exchange	Application Shimming	Application Shimming	Clear Command History	Credentials in Files	Network Service Scanning	Logon Scripts	Data from Local System	Exfiltration Over Alternative Protocol	Custom Cryptographic Protocol
Spearphishing Link	Execution through API	Authentication Package	Bypass User Account Control	CMSTP	Credentials in Registry	Network Share Discovery	Pass the Hash	Data from Network Shared Drive	Exfiltration Over Command and Control Channel	Data Encoding
Spearphishing via Service	Execution through Module Load	BITS Jobs	DLL Search Order Hijacking	Code Signing	Exploitation for Credential Access	Password Policy Discovery	Pass the Ticket	Data from Removable Media	Exfiltration Over Other Network Medium	Data Obfuscation
Supply Chain Compromise	Exploitation for Client Execution	Bootkit	Dylib Hijacking	Component Firmware	Forced Authentication	Peripheral Device Discovery	Remote Desktop Protocol	Data Staged	Exfiltration Over Physical Medium	Domain Fronting
Trusted Relationship	Graphical User Interface	Browser Extensions	Exploitation for Privilege Escalation	Component Object Model Hijacking	Hooking	Permission Groups Discovery	Remote File Copy	Email Collection	Scheduled Transfer	Fallback Channels
Valid Accounts	InstallUtil	Change Default File Association	Extra Window Memory Injection	Control Panel Items	Input Capture	Process Discovery	Remote Services	Input Capture		Multi-hop Proxy
	Launchctl	Component Firmware	File System Permissions Weakness	DCShadow	Input Prompt	Query Registry	Replication Through Removable Media	Man in the Browser		Multi-Stage Channels
	Local Job Scheduling	Component Object Model Hijacking	Hooking	Deobfuscate/Decode Files or Information	Kerberoasting	Remote System Discovery	Shared Webroot	Screen Capture		Multiband Communication
	LSASS Driver	Create Account	Image File Execution Options Injection	Disabling Security Tools	Keychain	Security Software Discovery	SSH Hijacking	Video Capture		Multilayer Encryption
	Mshta	DLL Search Order Hijacking	Launch Daemon	DLL Search Order Hijacking	LLMNR/NBT-NS Poisoning	System Information Discovery	Taint Shared Content			Port Knocking
	PowerShell	Dylib Hijacking	New Service	DLL Side-Loading	Network Sniffing	System Network Configuration Discovery	Third-party Software			Remote Access Tools
	Regsvcs/Regasm	External Remote Services	Path Interception	Exploitation for Defense Evasion	Password Filter DLL	System Network Connections Discovery	Windows Admin Shares			Remote File Copy
	Regsvr32	File System Permissions Weakness	Plist Modification	Extra Window Memory Injection	Private Keys	System Owner/User Discovery	Windows Remote Management			Standard Application Layer Protocol
	Rundll32	Hidden Files and Directories	Port Monitors	File Deletion	Replication Through Removable Media	System Service Discovery				Standard Cryptographic Protocol
	Scheduled Task	Hooking	Process Injection	File System Logical Offsets	Securityd Memory					Standard Non-Application Layer Protocol
	Scripting	Hypervisor	Scheduled Task	Gatekeeper Bypass	Two-Factor Authentication Interception					Uncommonly Used Port
	Service Execution	Image File Execution Options Injection	Service Registry Permissions Weakness	Hidden Files and Directories						Web Service
	Signed Binary Proxy Execution	Kernel Modules and Extensions	Setuid and Setgid	Hidden Users						
	Signed Script Proxy Execution	Launch Agent		Hidden Window						
	Source			HISTCONTROL						
	Space after Filename			Image File Execution Options Injection						

Repository

- https://github.com/redcanaryco/atomic-red-team



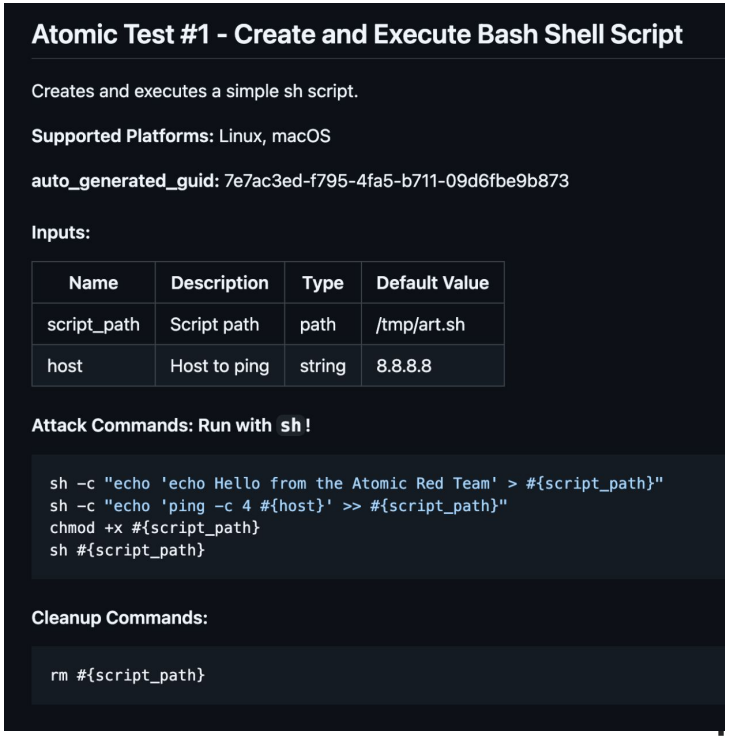
T1059.004 - Command and Scripting Interpreter: Bash

Description from ATT&CK

> Adversaries may abuse Unix shell commands and scripts for execution. Unix shells are the primary command prompt on Linux and macOS systems, though many variations of the Unix shell exist (e.g. sh, bash, zsh, etc.) depending on the specific OS or distribution. (Citation: DieNet Bash)(Citation: Apple ZShell) Unix shells can control every aspect of a system, with certain commands requiring elevated privileges.
> Unix shells also support scripts that enable sequential execution of commands as well as other typical programming operations such as conditionals and loops. Common uses of shell scripts include long or repetitive tasks, or the need to run the same set of commands on multiple systems.
>
> Adversaries may abuse Unix shells to execute various commands or payloads. Interactive shells may be accessed through command and control channels or during lateral movement such as with SSH. Adversaries may also leverage shell scripts to deliver and execute multiple commands on victims or as part of payloads used for persistence.

Atomic Tests

- Atomic Test #1 - Create and Execute Bash Shell Script
- Atomic Test #2 - Command-Line Interface
- Atomic Test #3 - Harvest SUID executable files
- Atomic Test #4 - LinEnum tool execution
- Atomic Test #5 - New script file in the tmp directory
- Atomic Test #6 - What shell is running
- Atomic Test #7 - What shells are available
- Atomic Test #8 - Command line scripts



Atomic Test #1 - Create and Execute Bash Shell Script

Creates and executes a simple sh script.

Supported Platforms: Linux, macOS

auto_generated_guid: 7e7ac3ed-f795-4fa5-b711-09d6fbe9b873

Inputs:

Name	Description	Type	Default Value
script_path	Script path	path	/tmp/art.sh
host	Host to ping	string	8.8.8.8

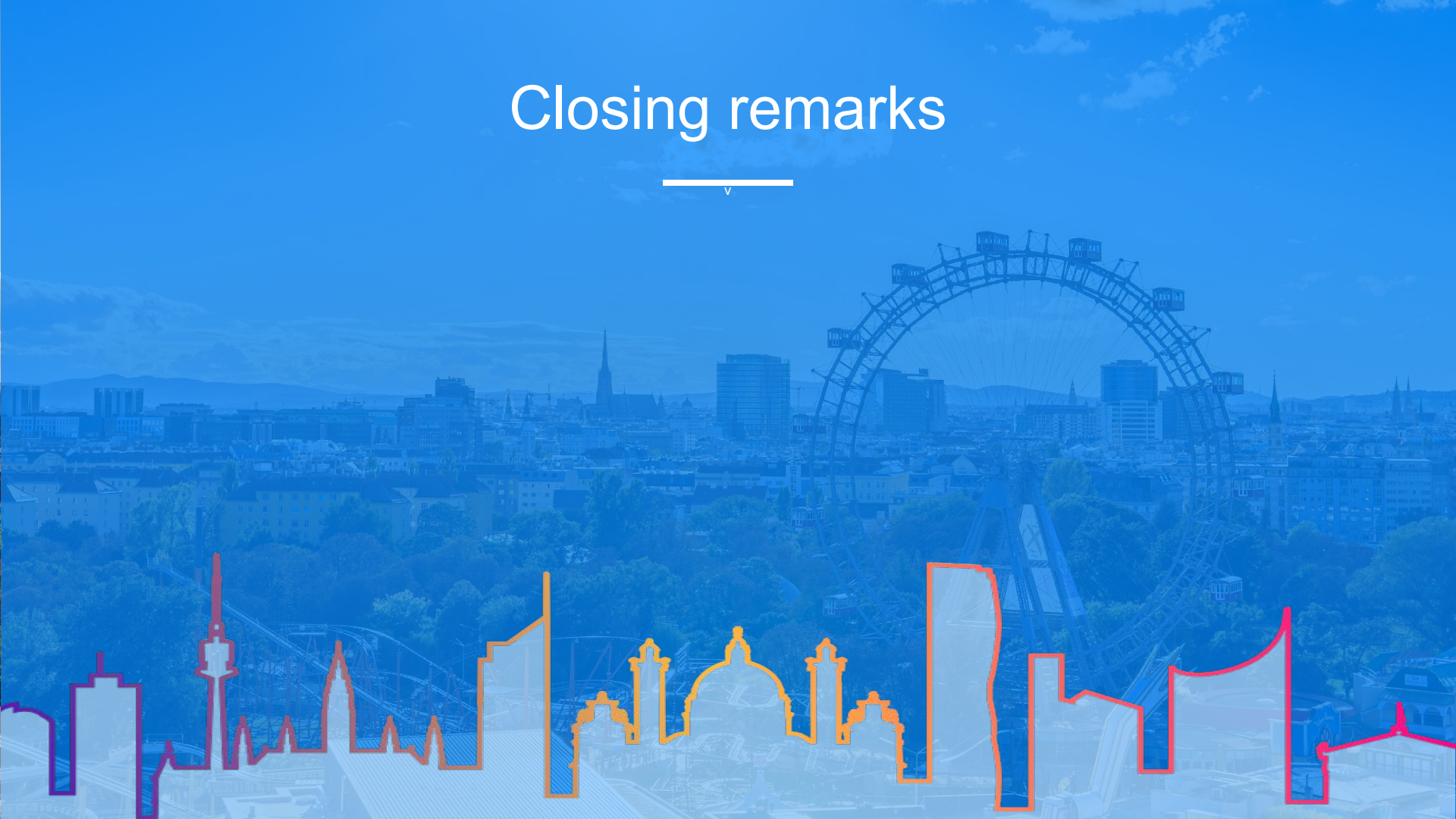
Attack Commands: Run with `sh`!

```
sh -c "echo 'echo Hello from the Atomic Red Team' > #{script_path}"
sh -c "echo 'ping -c 4 #{host}' >> #{script_path}"
chmod +x #{script_path}
sh #{script_path}
```

Cleanup Commands:

```
rm #{script_path}
```

Closing remarks

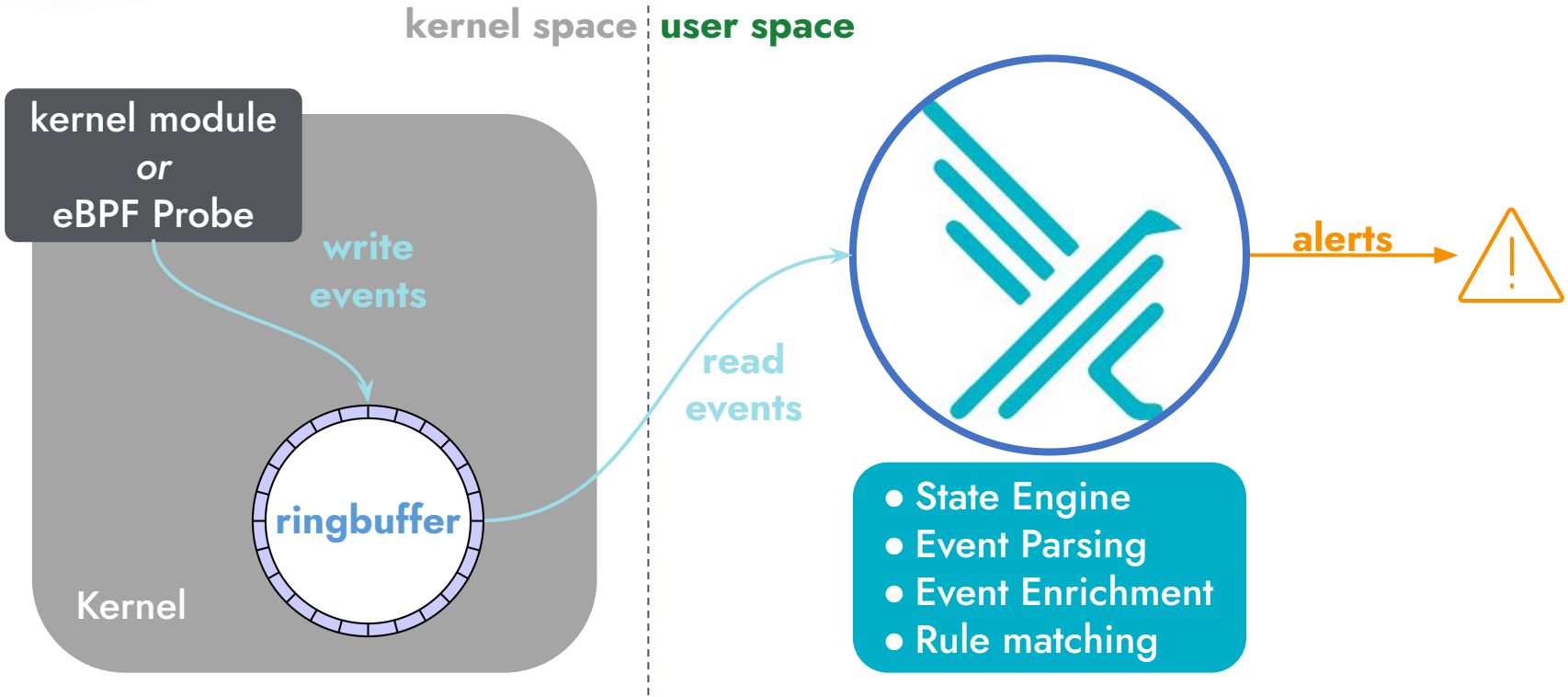


Runtime security

Guard perimeter

Detect unusual activities

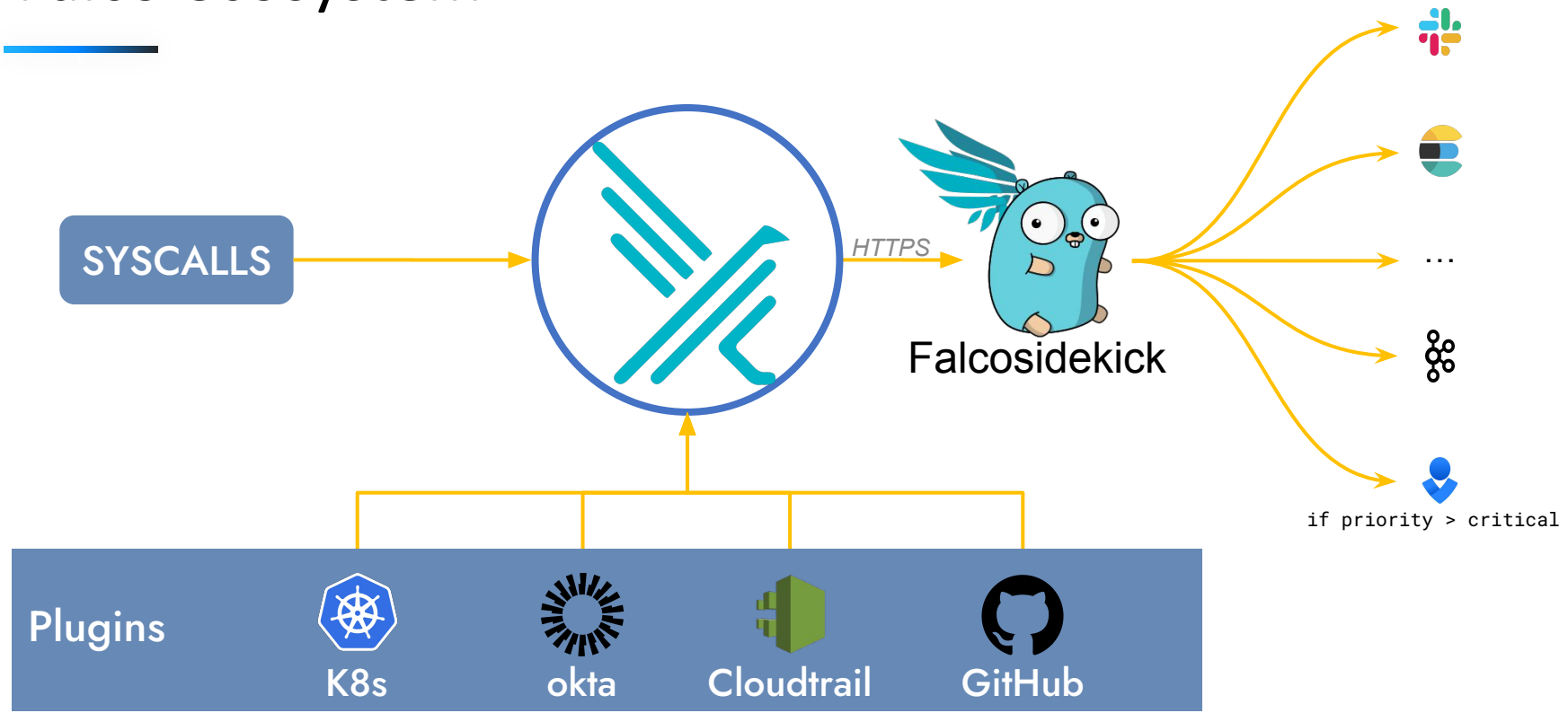
Falco Syscall Architecture Overview



Falco rules

```
- rule: Terminal shell in container
  desc: A shell has been spawned in a container.
  condition: >
    spawned_process and container and shell_procs
  output: >
    A shell was spawned in a container (user=%user.name
user_loginuid=%user.loginuid %container.info shell=%proc.name
parent=%proc.pname cmdline=%proc.cmdline container_id=%container.id)
  priority: WARNING
  tags: [container, shell, mitre_execution]
```

Falco ecosystem



References

"Practical Cloud Native Security with Falco", O'Reilly book

https://falco.org/docs

https://falco.org/training

https://falco.org/blog/extend-falco-outputs-with-falcosidekick/

KCD Austria
2024

Detecting unexpected behavior and intrusions with Falco + Atomic Red Team

WORKSHOP

PRESENTER

Miguel Hernández Boza

Sr. Threat Research Engineer

PRESENTER

Vicente J. Jiménez Miras

Independent Technical Trainer

Austria

KCD

GROWING CLOUD NATIVE TOGETHER

Survey

https://bit.ly/falcosurveypdx