FALCO-8

Zuzanna Brzezińska
Patryk Cetnar
Adrianna Łysik
Tomasz Ukowski

March 2023

Contents

1 Introduction

The increasing popularisation of cloud-native environments and container orchestration systems such as Kubernetes presents new challenges for security professionals. Traditional security tools might not be suitable for detecting and preventing attacks on these modern systems. This project aims to demonstrate the capabilities of Falco, an open-source cloud-native security tool designed to detect anomalous behavior and potential threats in Kubernetes, containers, and serverless functions.

In this tech demo, we will deploy a Kubernetes cluster on Amazon Web Services (AWS) and set up Falco to monitor the cluster. We will also deploy a vulnerable web application using Ansible and conduct a series of simulated attacks on the application. Throughout the demonstration, we will analyze Falco's output to assess its effectiveness in detecting and alerting on these security events.

1.1 Project Relevance and Target Audience

In the context of rapidly evolving cloud-native technologies, ensuring the security of these systems is of significant importance. With the emergence of new threats and vulnerabilities, it is essential for security professionals to be knowledgeable about cutting-edge tools and techniques to safeguard their infrastructure. This project aims to provide valuable insights and hands-on experience in using Falco for security monitoring in a Kubernetes environment.

The target audience for this project includes security professionals, DevOps engineers, and anyone responsible for maintaining the security of containerized applications or Kubernetes clusters. By following the demonstration and replicating the setup, readers can learn to deploy and configure Falco in their own environments, assess its effectiveness in detecting security events, and gain a better understanding of the challenges and potential solutions in cloud-native security.

1.2 Division of duties

The project tasks have been divided among the team members as follows:

1. Infrastructure deployment and provisioning (AWS EKS, Terraform)
    - Adrianna, Patryk
2. Falco (with plugins) deployment and configuration (Ansible)
    - It may also be required to implement *falcosidekick* to export events to external platform
    - Adrianna, Patryk
3. Vulnerable app deployment (DVWA, Ansible)

- Zuzanna, Tomasz

4. Vulnerability exploitation and investigation (bash scripts, documentation)
   - Zuzanna, Tomasz
5. Demo preparation

2 Theoretical background/technology stack

Diagram shows the high-level architecture of the planned solution. It's based on services provided by Amazon Web Services (AWS). To deploy and maintain the infrastructure, developers will use Terraform. An Amazon Elastic Kubernetes Service (EKS) will be deployed and used to run the Falco and monitored application. To provide external application communication, an application load balancer with a proper configuration will be prepared.

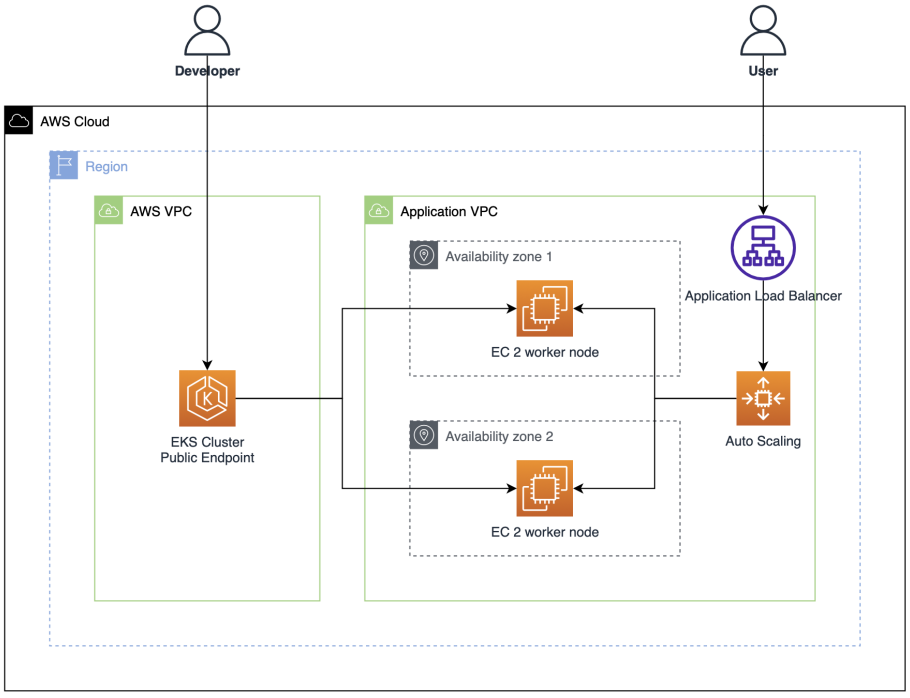

Figure 1: Architecture diagram

2.1 Terraform

HashiCorp Terraform is an infrastructure as code tool that lets you define both cloud and on-prem resources in human-readable configuration files that you can version, reuse, and share. You can then use a consistent workflow to provision and manage all of your infrastructure throughout its lifecycle. Terraform can manage low-level components like compute, storage, and networking resources, as well as high-level components like DNS entries and SaaS features. Terraform creates and manages resources on cloud platforms and other services through their application programming interfaces (APIs). Providers enable Terraform to work with virtually any platform or service with an accessible API.

The core Terraform workflow consists of three stages:

- Write: You define resources, which may be across multiple cloud providers and services. For example, you might create a configuration to deploy an application on virtual machines in a Virtual Private Cloud (VPC) network with security groups and a load balancer.
- Plan: Terraform creates an execution plan describing the infrastructure it will create, update, or destroy based on the existing infrastructure and your configuration.
- Apply: On approval, Terraform performs the proposed operations in the correct order, respecting any resource dependencies. For example, if you update the properties of a VPC and change the number of virtual machines in that VPC, Terraform will recreate the VPC before scaling the virtual machines.

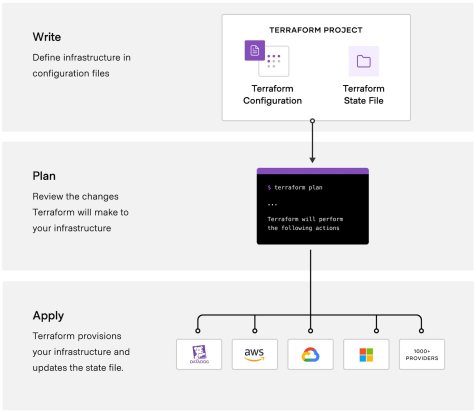

Figure 2: Terraform workflow [4]

2.2 Ansible

Ansible automates the management of remote systems and controls their desired state. A basic Ansible environment has three main components:

- Control node - A system on which Ansible is installed. You run Ansible commands such as ansible or ansible-inventory on a control node.
- Managed node - A remote system, or host, that Ansible controls.
- Inventory - A list of managed nodes that are logically organized. You create an inventory on the control node to describe host deployments to Ansible.

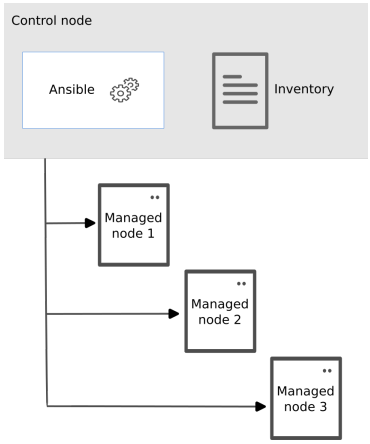

Figure 3: Ansible components [1]

2.3 AWS Elastic Kubernetes Service

Amazon Elastic Kubernetes Service (Amazon EKS) is a managed service that you can use to run Kubernetes on AWS without needing to install, operate, and maintain your own Kubernetes control plane or nodes. Kubernetes is an open-source system for automating the deployment, scaling, and management of containerized applications.

An Amazon EKS cluster consists of two primary components:

- The Amazon EKS control plane
- Amazon EKS nodes that are registered with the control plane

The Amazon EKS control plane consists of control plane nodes that run the Kubernetes software, such as etcd and the Kubernetes API server. The

control plane runs in an account managed by AWS, and the Kubernetes API is exposed via the Amazon EKS endpoint associated with your cluster. Each Amazon EKS cluster control plane is single-tenant and unique, and runs on its own set of Amazon EC2 instances.

All of the data stored by the etcd nodes and associated Amazon EBS volumes is encrypted using AWS KMS. The cluster control plane is provisioned across multiple Availability Zones and fronted by an Elastic Load Balancing Network Load Balancer. Amazon EKS also provisions elastic network interfaces in your VPC subnets to provide connectivity from the control plane instances to the nodes (for example, to support kubectl exec logs proxy data flows). [2]

A Kubernetes node is a machine that runs containerized applications. Each node has the following components:

- Container runtime – Software that's responsible for running the containers.
- kubelet – Makes sure that containers are healthy and running within their associated pod.
- kube-proxy – Maintains network rules that allow communication to your pods.

2.4 Falco

Falco is an open-source Cloud Native Security tool that is designed to detect anomalous or potentially malicious behavior in Cloud Native environments such as Kubernetes, containers, and serverless functions.

Falco uses system calls to secure and monitor a system, by:

- Parsing the Linux system calls from the kernel at runtime
- Asserting the stream against a powerful rules engine
- Alerting when a rule is violated

Falco ships with a default set of rules that check the kernel for unusual behavior such as:

- Privilege escalation using privileged containers
- Namespace changes using tools like setns
- Read/Writes to well-known directories such as /etc, /usr/bin, /usr/sbin, etc
- Creating symlinks
- Ownership and Mode changes
- Unexpected network connections or socket mutations
- Spawned processes using execve

- Executing shell binaries such as sh, bash, csh, zsh, etc
- Executing SSH binaries such as ssh, scp, sftp, etc
- Mutating Linux coreutils executables
- Mutating login binaries
- Mutating shadowutil or passwd executables such as shadowconfig, pwck, chpasswd, getpasswd, change, useradd, etc, and others.

2.4.1 Falco plugins

The Falco libraries and Falco itself can be extended by using Plugins. Plugins are shared libraries that conform to a documented API and allow for:

- Adding new event sources that can be evaluated using filtering expressions/Falco rules.
- Adding the ability to define new fields that can extract information from events.

Falco has two plugins specific to AWS. The CloudTrail plugin can read AWS CloudTrail logs and emit events for each CloudTrail log entry. It includes out-of-the-box rules that can be used to identify potential threats in CloudTrail logs, including:

- Console logins that do not use multi-factor authentication.
- Disabling multi-factor authentication for users.
- Disabling encryption for S3 buckets.

Falco has extended its capability to read Kubernetes audit logs through a plugin for CloudWatch, where it can read the EKS audit logs. [3]

2.5 DVWA

DVWA is a PHP/MySQL web application, its main goal is to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and to aid both students and teachers to learn about web application security in a controlled class room environment. The aim of DVWA is to practice some of the most common web vulnerabilities, with various levels of difficulty, with a simple straightforward interface.

3 Case study concept description

To demonstrate the correct operation of the Falco tool, we will deploy it in a Kubernetes cluster along with a vulnerable application. The demo will feature a threat actor and a member of the security engineering team. The first one will perform various operations to exploit multiple vulnerabilities in the application. During and after the attack, the person responsible for application security will be able to observe the steps taken by the attacker. To provide this information, we will use Falco - Kubernetes threat detection engine.

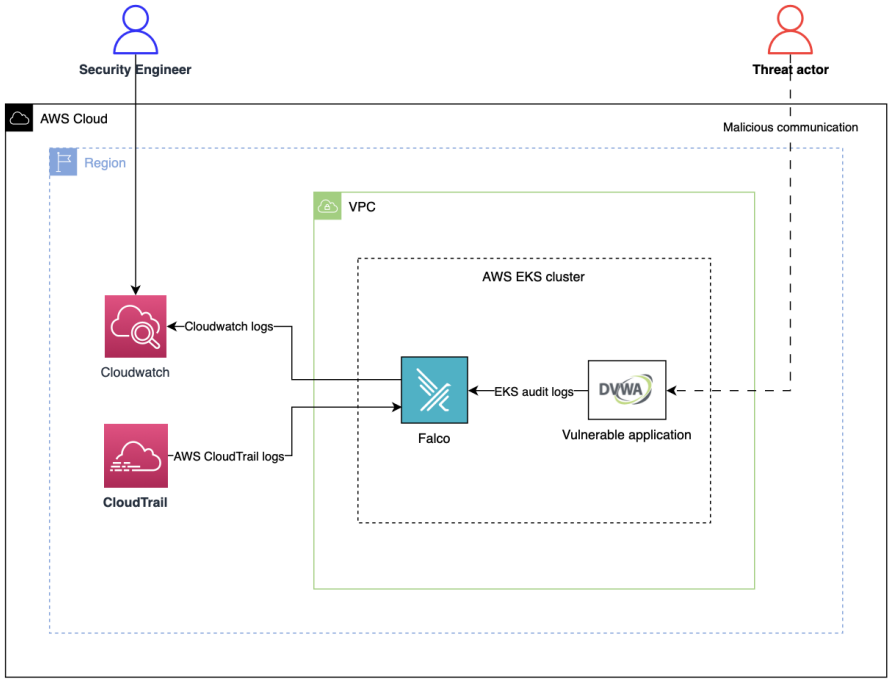

Figure 4: Demo concept diagram

4 Solution architecture

The attacker will use the vulnerability of the deployed DVWA application and the gained SSH access via a leaked key to carry out malicious operations. Falco will enable detection of the attack at various stages of its execution. Example rules that will be used are listed below:

- Reconnaissance / Discovery
  - Contact EC2 Instance Metadata Service From Container
  - Contact K8S API Server From Container
- Delivery
  - Launch Package Management Process in Container
- Exploitation
  - Redirect STDOUT/STDIN to Network Connection in Container
- Installation
  - Create files below dev
- Command & Control
  - Execution from /dev/shm
- Exfiltration
  - Launch Remote File Copy Tools in Container
  - Read environment variable from /proc files
  - Search Private Keys or Passwords
- Hiding traces
  - Clear Log Activities
  - Delete Bash History
  - Delete or rename shell history

5 Environment configuration description

The configuration will be defined using Terraform, its state will be stored in the S3 bucket so that any developer can access it. The files will describe the configuration of an EKS cluster consisting of a single node-group, with a target size of 2 nodes.

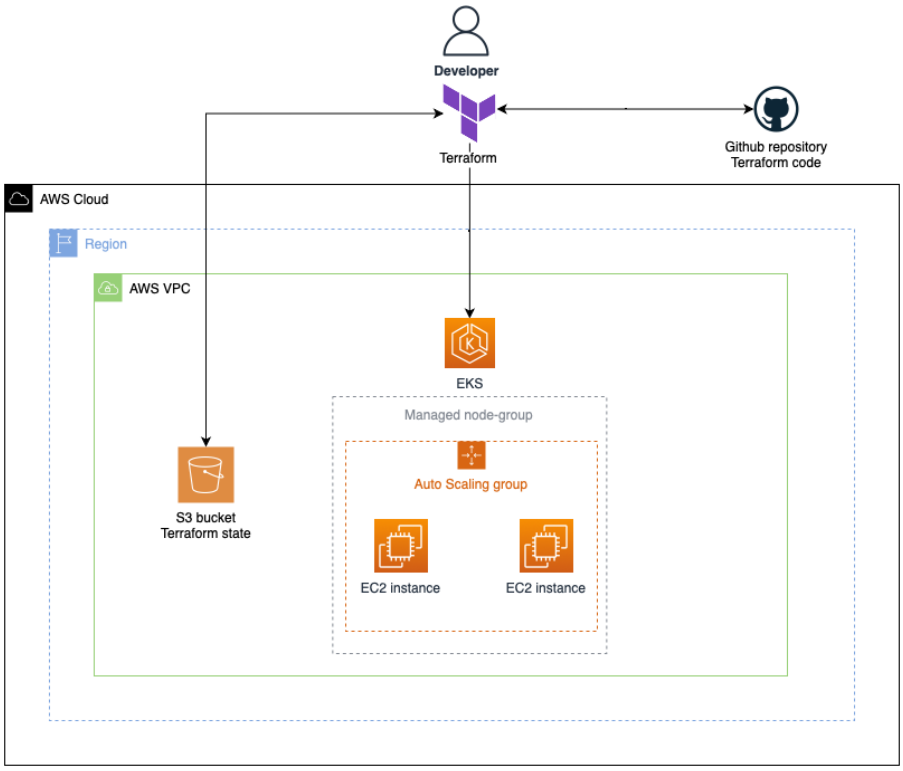

Figure 5: Configuration description

5.1 Fluent Bit

Fluent Bit is a lightweight and flexible open-source log and metric collector and forwarder. It is commonly used in log and metric processing architectures to gather, process, and transmit log and metric data between various components of a system.

The main features and functions of Fluent Bit include:

- Log data collection: Fluent Bit can receive log data from various sources

such as log files, standard input streams, syslog, TCP, UDP, serial ports, and more. It can also read data from other systems like Docker or Kubernetes.

- Data processing: Upon receiving log data, Fluent Bit can process it using different processing modules. Filters, parsers, field mappings, aggregations, and other processing operations can be applied to customize and structure log data according to requirements.
- Data forwarding: After processing log data, Fluent Bit can forward it to various destination points. It can send data to storage systems such as Elasticsearch, InfluxDB, Amazon S3, Apache Kafka, MongoDB, and many others. It can also send data to monitoring and analysis services like Prometheus, Grafana, or Kibana.
- Flexibility and extensibility: Fluent Bit is designed with flexibility and extensibility in mind. It has a modular architecture that allows for adding new input, processing, and output modules.

5.2 Elasticsearch

Elasticsearch is a highly scalable and distributed open-source search and analytics engine. It is built on top of Apache Lucene, a powerful full-text search library, and is designed to handle large volumes of data and provide fast and real-time search capabilities.

Key features of Elasticsearch include:

- Full-text search: Elasticsearch enables full-text search across large volumes of structured and unstructured data. It supports various types of queries, including fuzzy searches, phrase matching, proximity searches, and more.
- Distributed and scalable: Elasticsearch is designed to be distributed, allowing data to be distributed across multiple nodes in a cluster. This provides scalability and high availability, as data can be replicated and distributed for load balancing and fault tolerance.
- Real-time analytics: Elasticsearch provides near real-time indexing and analytics capabilities, allowing users to perform complex aggregations, filtering, and analysis on large datasets quickly. It supports various aggregations, such as metrics, histograms, geolocation, and more.
- Schemaless and flexible: Elasticsearch is schemaless, which means you can index and search any kind of data without the need for predefined schemas. It automatically detects and indexes fields, making it easy to work with evolving and dynamic data.

5.3 Kibana

Kibana is an open-source data visualization and exploration tool that works alongside Elasticsearch. It provides a user-friendly interface for visualizing, analyzing, and exploring data stored in Elasticsearch, allowing users to gain insights and make data-driven decisions.

Key features of Kibana include:

- Data Visualization: Kibana allows users to create a wide range of visualizations such as charts, graphs, histograms, maps, and tables to represent data in a clear and interactive manner. These visualizations can be customized and configured to showcase different aspects and metrics of the data.
- Dashboard Creation: Kibana enables the creation of interactive dashboards by combining multiple visualizations and saved searches into a single view. Dashboards provide a consolidated overview of data, allowing users to monitor key metrics and trends at a glance.
- Data Exploration and Search: Kibana provides a powerful search and query interface, allowing users to explore and retrieve data from Elasticsearch using a query language called Elasticsearch Query DSL. Users can apply filters, aggregations, and perform complex searches to find specific information within their data.
- Real-time Monitoring: Kibana offers real-time monitoring capabilities to track and visualize live data streams. It can display metrics, logs, and events in real-time, enabling users to monitor system performance, detect anomalies, and respond to issues promptly.
- Alerting and Reporting: Kibana allows users to configure alerts and notifications based on specific conditions or thresholds. It can generate alerts for critical events, trigger actions, and send notifications via various channels. Additionally, Kibana offers reporting capabilities to generate scheduled reports and share them with stakeholders.

10 References

References

[1] "Ansible Getting Started". In: (). URL: https://docs.ansible.com/ansible/latest/getting_started/index.html.

[2] "EKS User Guide". In: (). URL: https://docs.aws.amazon.com/eks/latest/userguide.

[3] "Falco on AWS Cloud". In: (). URL: https://falco.org/blog/falco-on-aws/.

[4] "Terraform Intro". In: (). URL: https://developer.hashicorp.com/terraform/intro.