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.

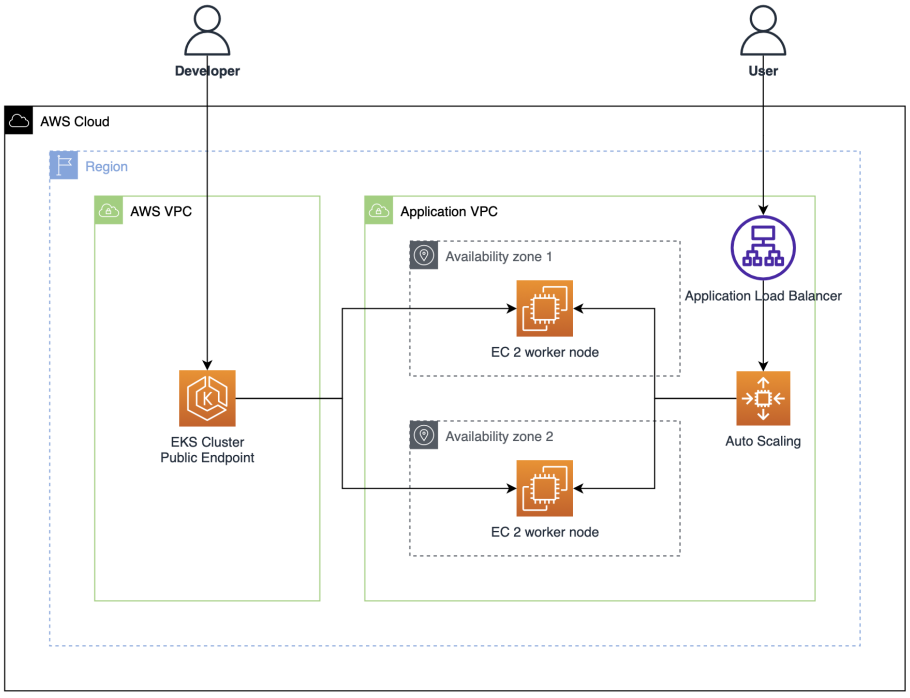

Figure 1: Architecture diagram

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.

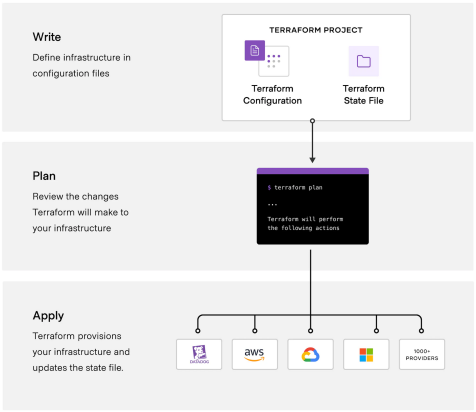

Figure 2: Terraform workflow [4]

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.

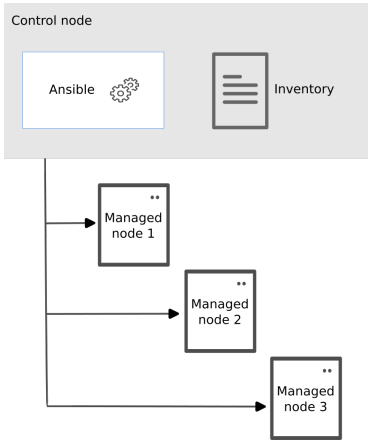

Figure 3: Ansible components [1]

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 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.

2.5 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.

2.6 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.

2.7 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.

2.7.1 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]

2.8 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.

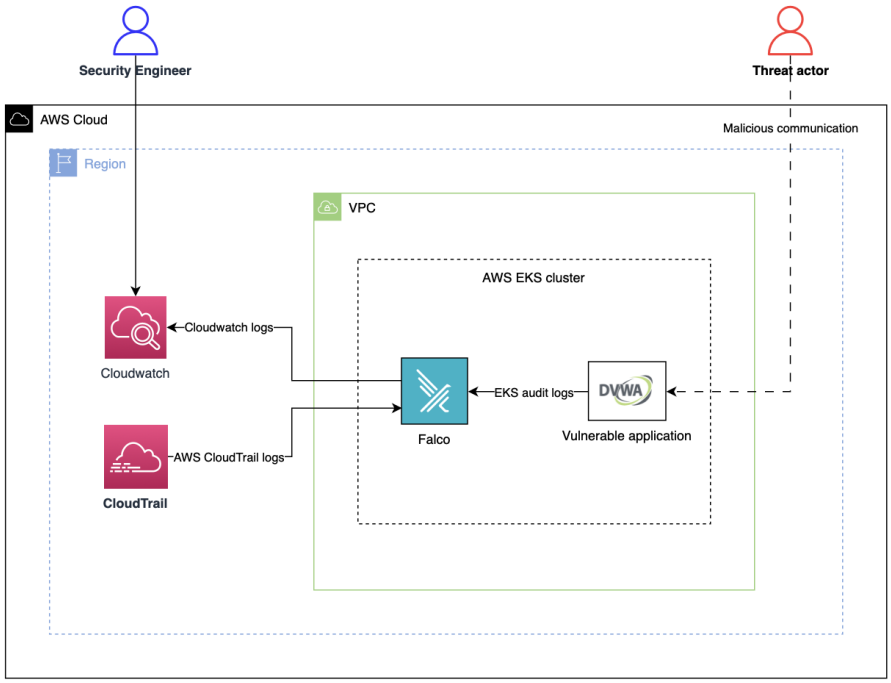

Figure 4: Demo concept diagram

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. The files will describe the configuration of an EKS cluster consisting of a single node-group, with a target size of 2 nodes.

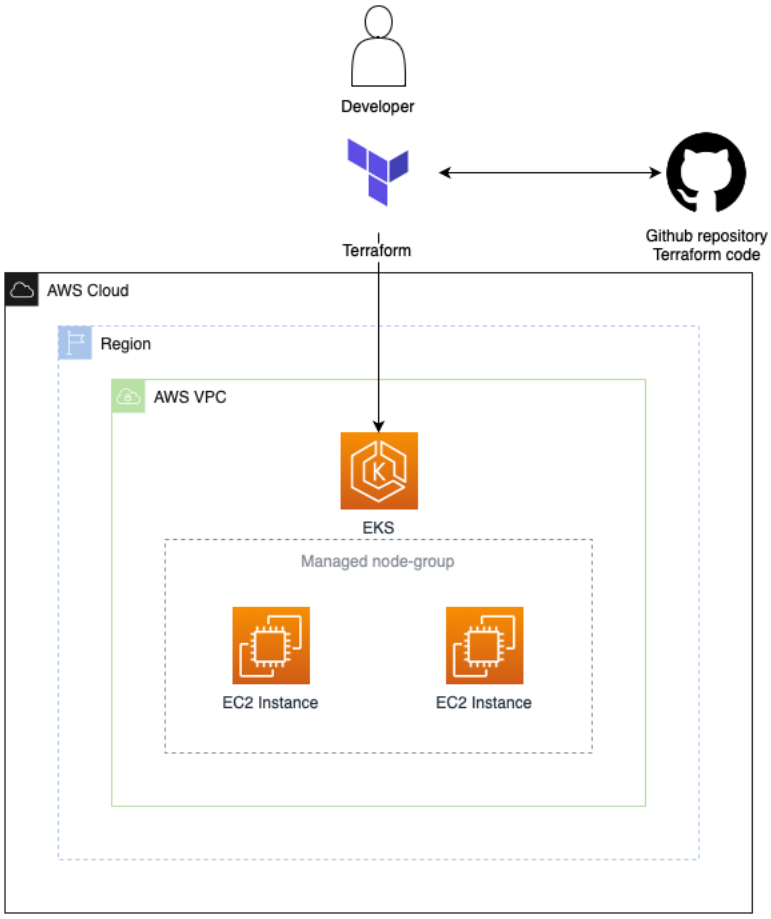

Figure 5: Configuration description

6 Installation method

6.1 Prerequisites

- Set up an AWS account and obtain access credentials (access key and secret key).
- Create EKS Cluster
- Install the AWS CLI and configure it with your AWS credentials.
- Install Ansible and Terraform on your local machine.
- Install kubectl and the AWS IAM Authenticator
- Install Helm

6.2 Infrastructure as Code (IaC)

The Infrastructure as Code (IaC) approach in the described project involves using Ansible and Terraform to deploy Falco and an application on an Amazon Elastic Kubernetes Service (EKS) cluster. Here's an overview of the IaC approach in this project:

Terraform: Terraform is used as the IaC tool to define, provision, and manage the infrastructure resources required for the EKS cluster. This includes defining the networking components, security groups, EKS cluster, and any other necessary resources. Terraform provides a declarative syntax to define the desired state of the infrastructure and automates the provisioning of resources on AWS.

Ansible: Ansible is used as the configuration management tool to deploy Falco and the application on the EKS cluster. Ansible provides a simple and agentless approach to manage the configuration of servers, applications, and services. With Ansible, you can define tasks and playbooks to automate the deployment and configuration of Falco and the application on the EKS cluster.

Elastic Kubernetes Service (EKS): EKS is the managed Kubernetes service provided by AWS. It simplifies the management of the underlying Kubernetes infrastructure, such as control plane operations and scaling, allowing developers to focus on deploying and running their applications. Terraform is used to create and manage the EKS cluster resources, while Ansible is used to configure and deploy Falco and the application on the EKS cluster.

Infrastructure Automation: Terraform scripts are used to define the infrastructure components, including the networking setup, security groups, and the EKS cluster. These scripts specify the desired state of the infrastructure, such as the number of nodes in the EKS cluster, network configurations, and security settings. Terraform will then provision and configure these resources on AWS based on the defined specifications.

DVWA and Falco Deployment with Ansible: Ansible playbooks and roles are used to define the deployment and configuration steps for Falco and the

application on the EKS cluster. Ansible tasks can include actions such as creating Kubernetes deployment and service manifests, deploying Docker containers, configuring environment variables, and managing dependencies. Ansible allows for idempotent deployments, meaning the playbooks can be run multiple times without causing issues or inconsistencies.

By adopting the IaC approach with Terraform and Ansible, the project enables infrastructure provisioning and application deployment to be defined as code. This approach brings several benefits, including repeatability, version control, scalability, and easier maintenance. It allows for efficient collaboration among team members and provides a reliable and consistent way to deploy and manage Falco and the application on the EKS cluster.

7 How to reproduce - step by step

7.1 AWS

1. Log into AWS Management Console
2. Get your credentials and onfigure AWS on your local machine using `aws configure` or copying your acces keys and session token to *credentials* file (it should be in */.aws/* folder

7.2 Infrastructure as Code approach

7.2.1 Terraform

1. Go to */terraform folder*
2. Initialize your Terraform working directory by running command: `terraform init`
3. Change defaults in *variables.tf* file:
   - in "subnet_id_1" variable paste the id of first subnet
   - in "subnet_id_2" variable paste the id of second subnet
   - in "subnet_id_3" variable paste the id of third subnet
   - in "role_arn" variable paste ARN role (in our case LabRole)
   - in "region" variable leave "us-east-1" or change it if needed

```
variable "subnet_id_1" {
  description = "First subnet ID"
  type    = string
  default = "subnet-0ff2bf41e8cbfdf25"
}
variable "subnet_id_2" {
  description = "Second subnet ID"
  type    = string
  default = "subnet-036d22cdce0fae025"
}
variable "subnet_id_3" {
  description = "Third subnet ID"
  type    = string
  default = "subnet-01026c365f2f42dce"
}

variable "role_arn" {
  description = "LabRole arn"
  type    = string
  default = "arn:aws:iam::448323382078:role/LabRole"
}

variable "region" {
  description = "AWS region"
  type        = string
  default     = "us-east-1"
}
```

Figure 6: Filled variables.tf file

4. Now you can use `terraform plan` command in order to preview the changes that Terraform plans to make to your infrastructure

5. Execute `terraform apply` to execute actions proposed in terraform plan - it might take a while to create all the infrastructure. When it is done, you can go to the Amazon EKS Service in AWS Management Console and check if your Cluster (named 'My Cluster') is set.

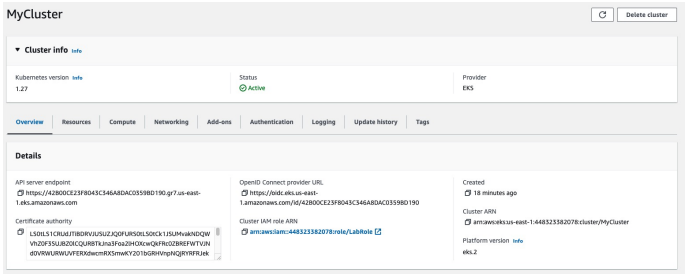

Figure 7: Running cluster

6. Next configure kubectl using `aws eks --region $(terraform output -raw region) update-kubeconfig --name $(terraform output -raw`

```
cluster_name)

zbrzezin@krk-mpgwp terraform % aws eks --region $(terraform output -raw region) update-kubeconfig
--name $(terraform output -raw cluster_name)
Added new context arn:aws:eks:us-east-1:448323382078:cluster/MyCluster to /Users/zbrzezin/.kube/co
nfig
```

Figure 8: Running cluster

7.3 Elasticsearch

1. Add elasticsearch repository in Helm using
   `helm repo add elastic https://helm.elastic.co`
2. Now, use the curl command to download the values.yaml file containing configuration information:
   `hcurl -O https://raw.githubusercontent.com/elastic/helm-charts/master/elasticsearch/examples/minikube/values.yaml`
3. Use the helm install command and the values.yaml file to install the Elasticsearch helm chart: `helm install elasticsearch elastic/elasticsearch -f ./values.yaml`

7.4 Kibana

To install Kibana on top of Elasticsearch, type the following command: `helm install kibana elastic/kibana`

7.5 Fluent Bit

1. To add the fluent helm repo, run:
   `helm repo add fluent https://fluent.github.io/helm-charts`
2. To install a release named fluent-bit, run:
   `helm install fluent-bit fluent/fluent-bit`

8 Demo deployment steps

The previous chapters described how to prepare an AWS account and deploy the necessary infrastructure. Now we will use Ansible to run Falco and test application called DVWA. Then we will carry out attacks that will be captured in Falco logs.

8.1 Configuration set-up

There are 5 folders in the Ansible folder, three main ones that we will use during demo:

- *falco* - used to deploy Falco
- *plain-dvwa* - used to deploy test application called DVWA
- *nginx-webserver* - to deploy NGINX web server

The other two are:

- *dvwa* - shows another way of deploying application using Dockerfile
- *efk* - used to deploy Elasticsearch, Fluent-bit and Kibana

In each of three main folders there is one file that will be used as Ansible playbook. To use them, execute:

ansible-playbook ./falco/playbook-falco.yaml

ansible-playbook ./plain-dvwa/playbook-plain-dvwa.yaml

ansible-playbook ./nginx-webserver/playbook-nginx.yaml

Figure 9 presents the resources that will appear in Falco namespace after successful Ansible deployment.

```
patryk@Air-Patryk ansible % k get all -n falco
NAME              READY   STATUS    RESTARTS   AGE
pod/falco-pttxl   2/2     Running   0          4m25s
pod/falco-th7xj   2/2     Running   0          4m24s

NAME                   DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
daemonset.apps/falco   2         2         2       2            2           <none>          4m38s
```

Figure 9: Falco namespace resources

Similarly Figure 10 shows those in default namespace.

Additionally, in Figure 10 there is external IP next to dvwa-service. After opening this link in the browser, the main page of the DVWA application will appear, as presented in Figure 11. Log in using random string of characters and click the "Create/Reset Database" button. After logging in again, the main menu of the application will appear.

```
patryk@Air-Patryk ansible % k get all
NAME                         READY   STATUS    RESTARTS   AGE
pod/dvwa-7974d7c79f-l2v5z    1/1     Running   0          65s
pod/nginx2-795d4df889-l5cbz  1/1     Running   0          87s

NAME                  TYPE           CLUSTER-IP      EXTERNAL-IP                                                                PORT(S)        AGE
service/dvwa-service  LoadBalancer   10.100.39.138   a225cc85427a24cc486c7ada6161d93d-845455609.us-east-1.elb.amazonaws.com   80:31029/TCP   64s
service/kubernetes    ClusterIP      10.100.0.1      <none>                                                                     443/TCP        18m

NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/dvwa    1/1     1            1           66s
deployment.apps/nginx2  1/1     1            1           88s

NAME                               DESIRED   CURRENT   READY   AGE
replicaset.apps/dvwa-7974d7c79f    1         1         1       66s
replicaset.apps/nginx2-795d4df889  1         1         1       88s
```

Figure 10: Default namespace resources

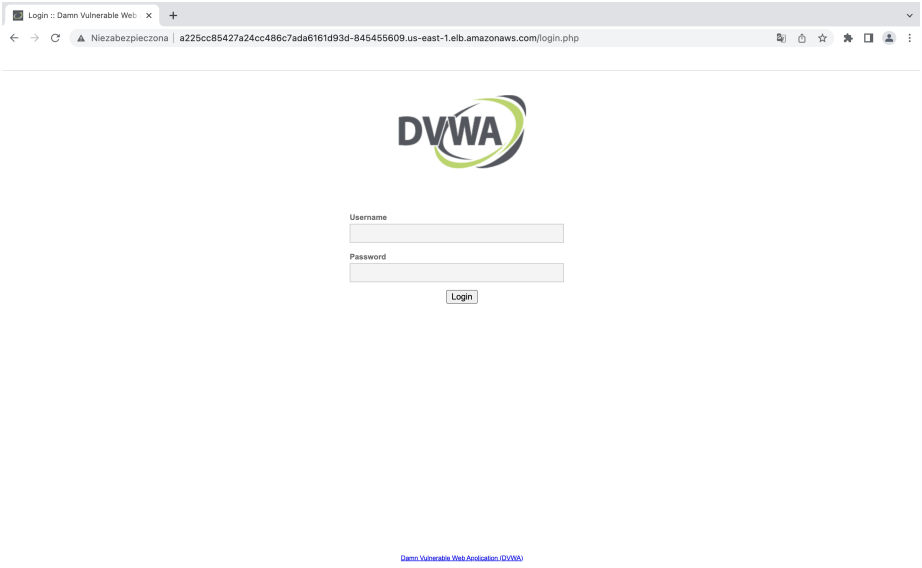

Figure 11: DVWA main page

8.2 Data preparation

In the case of our project, one should prepare the attacks that will be performed on the application. Using the application interface, we will read data from a sensitive file:

ping 127.0.0.1; cat /etc/passwd

Then the attack will be simulated inside the container, using the following commands:

curl -k https://kubernetes.default.svc

apt-get update

apt install -y openssh-client openssh-server

scp

touch /dev/malware_variables

cat /proc/1489/environ

cat /etc/shadow

rm /.bash_history

8.3 Execution procedure

In our procedure, we have presented methods to perform attacks that will later be observed in Falco.

8.3.1 Attack using the application interface

In the first part, the attacker exploits the interface of the application we connected to in the previous chapter. For this purpose, malicious user will use the Command Injection tab. By design, one should only be able to ping a specific IP address. However, it is possible to run any command on the server. Figure 12 shows the result of the attack. The attacker was able to read all the information from the sensitive /etc/passwd file.

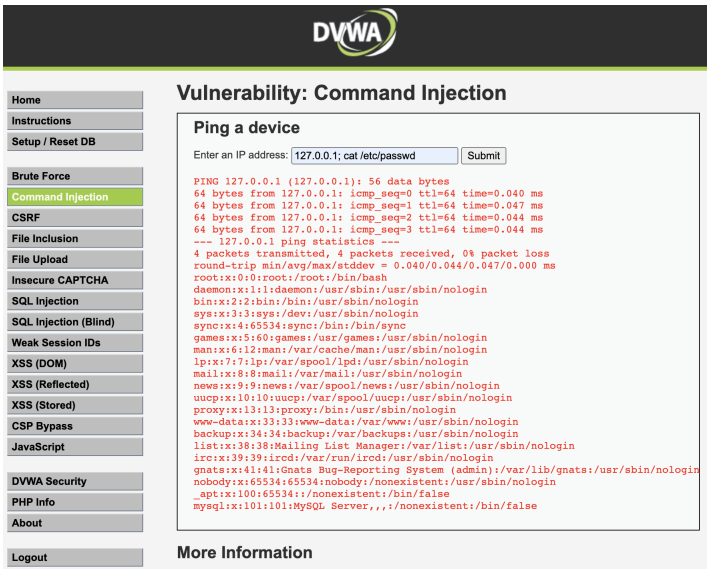

Figure 12: DVWA Command Injection attack

8.3.2 Attacks inside the container

The next part of the demo is to simulate the actions of an adversary who has already gained access to the vulnerable container. We will use the following command:

kubectl exec -it '[pod_name]' /bin/bash

where [pod_name] represents the name of the pod in our cluster e.g. pod/nginx2-795d4df889-l5cbz.

After gaining access to the container, we begin with reconnaissance. For this purpose, we are trying to connect to the Kubernetes api-server:

curl -k https://kubernetes.default.svc

Then we install tools that will help us maintain the control and transfer files:

apt-get update

apt install -y openssh-client openssh-server

We can use the SCP tool to copy the files:

scp

Then we create a configuration file that will be used remotely:

touch /dev/malware_variables

And read sensitive data that may be useful later:

cat /proc/30/environ

cat /etc/shadow

The results of these command are presented in Figure 13 and 14.

At the end, we clean the traces left:

rm /.bash_history

```
root@nginx2-795d4df889-xdxwj:/# cat /etc/shadow
root:*:18513:0:99999:7:::
daemon:*:18513:0:99999:7:::
bin:*:18513:0:99999:7:::
sys:*:18513:0:99999:7:::
sync:*:18513:0:99999:7:::
games:*:18513:0:99999:7:::
man:*:18513:0:99999:7:::
lp:*:18513:0:99999:7:::
mail:*:18513:0:99999:7:::
news:*:18513:0:99999:7:::
uucp:*:18513:0:99999:7:::
proxy:*:18513:0:99999:7:::
www-data:*:18513:0:99999:7:::
backup:*:18513:0:99999:7:::
list:*:18513:0:99999:7:::
irc:*:18513:0:99999:7:::
gnats:*:18513:0:99999:7:::
nobody:*:18513:0:99999:7:::
_apt:*:18513:0:99999:7:::
nginx:!:18515:0:99999:7:::
systemd-timesync:*:19523:0:99999:7:::
systemd-network:*:19523:0:99999:7:::
systemd-resolve:*:19523:0:99999:7:::
messagebus:*:19523:0:99999:7:::
sshd:*:19523:0:99999:7:::
```

Figure 13: /etc/shadow contents

```
root@nginx2-795d4df889-xdxwj:/# cat /proc/30/environ
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/binHOSTNAME=nginx2-795d4df889-xdxwjNGINX_VERSION=1.19.2NJS_VERSION=0.4.3PKG_RELEASE=1~busterKUBERNETES_PO
RT_443_TCP_PROTO=tcpKUBERNETES_PORT_443_TCP_PORT=443KUBERNETES_PORT_443_TCP_ADDR=10.100.0.1KUBERNETES_SERVICE_HOST=10.100.0.1KUBERNETES_SERVICE_PORT=443KUBERNETES_SERV
ICE_PORT_HTTPS=443KUBERNETES_PORT=tcp://10.100.0.1:443KUBERNETES_PORT_443_TCP=tcp://10.100.0.1:443TERM=xtermHOME=/rootroot@nginx2-795d4df889-xdxwj:/#
```

Figure 14: Process environment variables

8.4 Results presentation

During the demo, we simulated the attacker's behaviour, now we will look at what information the Security Engineer received thanks to Falco. Execute the command below:

k -n falco logs [falco_pod_name]

8.4.1 Attack using the application interface

Figure 15 shows what Falco recorded during the first attack. These are:

- *time*
- *executed command*
- *container ID*
- *image*
- *Kubernetes namespace*
- *Kubernetes pod name*

Using this information, the Cybersecurity Specialist can determine what information the malicious user managed to obtain.

```
15:58:04.908458918: Debug Shell spawned by untrusted binary (user=www-data user_loginuid=-1 shell=sh parent=apache2 cmdline=sh -c ping  -c 4 127.0.0.1; cat /etc/passwd
 pid=20880 pcmdline=apache2 -k start gparent=apache2 ggparent=main.sh aname[4]=<NA> aname[5]=<NA> aname[6]=<NA> aname[7]=<NA> container_id=b8dc213729fd image=docker.io
/vulnerables/web-dvwa) k8s.ns=default k8s.pod=dvwa-7974d7c79f-bptlc container=b8dc213729fd
```

Figure 15: Falco logs - attack using the application interface

8.4.2 Attacks inside the container

When it comes to the second attempt in which the attacker used many commands, Falco generated more events. They are shown in the Figures 16 and 17.

```
16:28:16.094237340: Notice A shell was spawned in a container with an attached terminal (user=root user_loginuid=-1 k8s.ns=default k8s.pod=nginx2-795d4df889-xdxwj container=796a8051730e
 shell=bash parent=runc cmdline=bash pid=30405 terminal=34816 container_id=796a8051730e image=docker.io/library/nginx)
16:28:31.207248863: Notice Unexpected connection to K8s API Server from container (command=curl -k https://kubernetes.default.svc pid=30478 k8s.ns=default k8s.pod=nginx2-795d4df889-xdxw
j container=796a8051730e image=docker.io/library/nginx:1.19.2 connection=172.31.82.108:39686->10.100.0.1:443)
16:28:38.803763601: Error Package management process launched in container (user=root user_loginuid=-1 command=apt-get update pid=30507 container_id=796a8051730e container_name=nginx im
age=docker.io/library/nginx:1.19.2) k8s.ns=default k8s.pod=nginx2-795d4df889-xdxwj container=796a8051730e
16:28:45.104245644: Error Package management process launched in container (user=root user_loginuid=-1 command=apt install -y openssh-client openssh-server pid=30802 container_id=796a80
51730e container_name=nginx image=docker.io/library/nginx:1.19.2) k8s.ns=default k8s.pod=nginx2-795d4df889-xdxwj container=796a8051730e
16:28:47.994974119: Critical Executing binary not part of base image (user=root user_loginuid=-1 user_uid=0 comm=systemctl enable gettytty1.service exe=systemctl container_id=796a80517
30e image=docker.io/library/nginx proc.name=systemctl proc.sname=dpkg proc.pname=systemd.postins proc.aname[2]=dpkg exe_flags=EXE_WRITABLE|EXE_UPPER_LAYER proc.exe_ino=10599306 proc.exe
_ino.ctime=1686846652783304l499 proc.exe_ino.mtime=1680204751000000000 proc.exe_ino.ctime_duration_proc_start=161781770 proc.exepath=/bin/systemctl proc.cwd=/ proc.tty=34817 container.st
art_ts=1686836649060071991 proc.sid=445 proc.vpgid=445 evt.res=SUCCESS) k8s.ns=default k8s.pod=nginx2-795d4df889-xdxwj container=796a8051730e
```

Figure 16: Falco logs - attacks inside the container

```
16:28:56.061599712: Critical Executing binary not part of base image (user=root user_loginuid=-1 user_uid=0 comm=scp exe=scp container_id=796a8051730e image=docker.io/library/nginx proc
.name=scp proc.sname=bash proc.pname=bash proc.aname[2]=<NA> exe_flags=EXE_WRITABLE|EXE_UPPER_LAYER proc.exe_ino=11751462 proc.exe_ino.ctime=1686846529489052220 proc.exe_ino.mtime=15805
04134000000000 proc.exe_ino.ctime_duration_proc_start=6571928781 proc.exepath=/usr/bin/scp proc.cwd=/ proc.tty=34816 container.start_ts=1686836649060071991 proc.sid=30 proc.vpgid=1037 e
vt.res=SUCCESS) k8s.ns=default k8s.pod=nginx2-795d4df889-xdxwj container=796a8051730e
16:29:01.219311799: Error File created below /dev by untrusted program (user=root user_loginuid=-1 command=touch /dev/malware_variables pid=31616 file=/dev/malware_variables container_i
d=796a8051730e image=docker.io/library/nginx) k8s.ns=default k8s.pod=nginx2-795d4df889-xdxwj container=796a8051730e
16:29:22.604549671: Warning Environment variables were retrieved from /proc files (user=root user_loginuid=-1 program=cat command=cat /proc/30/environ pid=31708 file=/proc/30/environ pa
rent=bash gparent=<NA> ggparent=<NA> gggparent=<NA> container_id=796a8051730e image=docker.io/library/nginx) k8s.ns=default k8s.pod=nginx2-795d4df889-xdxwj container=796a8051730e
16:29:35.640911355: Warning Sensitive file opened for reading by non-trusted program (user=root user_loginuid=-1 program=cat command=cat /etc/shadow pid=31776 file=/etc/shadow parent=ba
sh gparent=<NA> ggparent=<NA> gggparent=<NA> container_id=796a8051730e image=docker.io/library/nginx) k8s.ns=default k8s.pod=nginx2-795d4df889-xdxwj container=796a8051730e
16:29:42.723915041: Warning Shell history had been deleted or renamed (user=root user_loginuid=-1 type=execve command=rm /root/.bash_history pid=31802 fd.name=<NA> name=<NA> path=<NA> o
ldpath=<NA> k8s.ns=default k8s.pod=nginx2-795d4df889-xdxwj container=796a8051730e)
```

Figure 17: Falco logs - attacks inside the container

In this case, the Cybersecurity Specialist can trace the sequence of actions of the opponent:

- *16:28:16.094237340* A shell was spawned in a container with an attached terminal (the attacker gained access to the container)
- *16:28:31.207248863* Unexpected connection to K8S Api Server from container (the attacker performed reconnaissance and connected to the Kubernetes api-server)
- *16:28:38.803763601* Package management process launched in container (the attacker downloaded new packages)
- *16:28:56.061599712* Executing binary not part of base image (the attacker used the downloaded tool)
- *16:29:01.219311799* File created below /dev by untrusted program (the attacker left behind the configuration file)
- *16:29:22.604549671* Environment variables were retrieved from /proc files (the attacker read sensitive information from the variables)

- *16:29:35.640911355* Sensitive file opened for reading by non-trusted program (the attacker read sensitive information from the /etc/shadow file)
- *16:29:42.723915041* Shell history had been deleted or renamed (the attacker removed his traces)

It is complete information about dangerous actions the malicious user has taken. With their use, one can determine data that has leaked and systems that have been compromised.

9 Summary – conclusions

In our project, we presented the method of deploying the Elastic Kubernetes Service infrastructure using Terraform. The software implemented using Ansible enabled us to conduct a demo. During it we carried out two attacks, thanks to Falco, we were able to notice, gather information and reproduce the attacker's activities. With its use, the Security Engineer is able to assess which systems have been compromised, what data could have been leaked and what tools were used for the attack. Such knowledge is crucial in assessing the costs incurred. Additional functionalities such as creating custom rules and the ability to send alerts to various systems (such as Slack or e-mail) make it a tool that brings great benefits for Security teams.

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.