NGINX App Protect Documentation

Matthieu Dierick

Dec 03, 2020

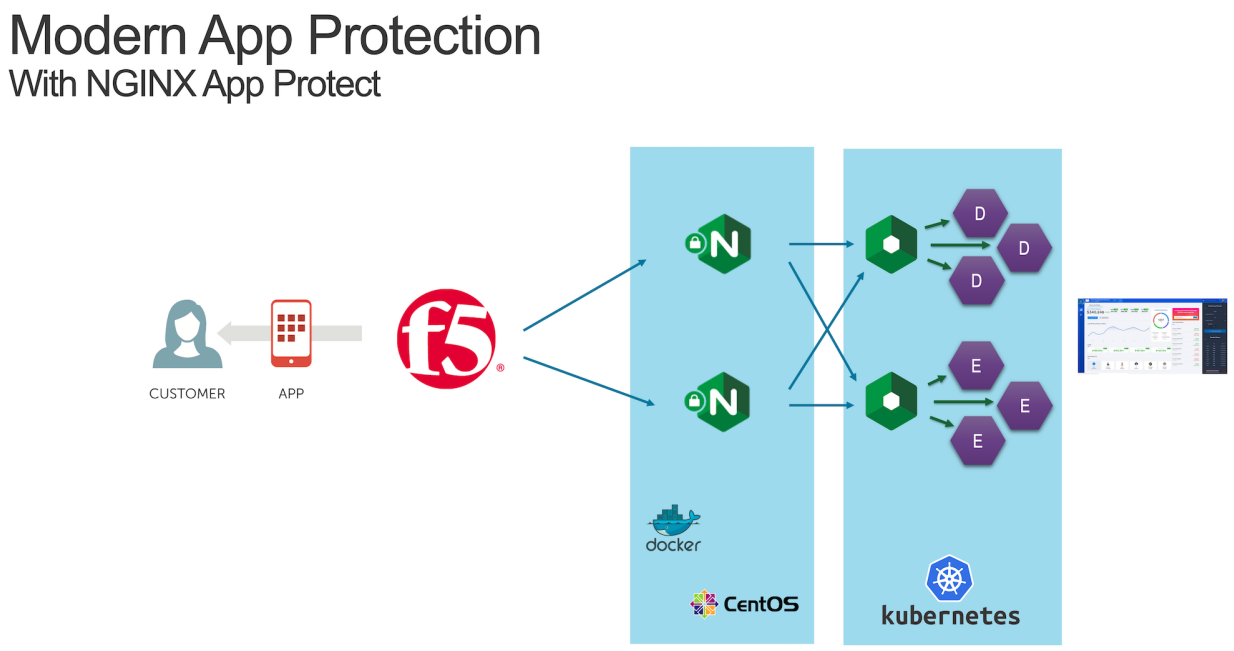
CONTENTS:

CHAPTER

ONE

PUBLISH AND PROTECT MODERN APPLICATIONS

Warning: For any remark or mistake in this lab, please send a Teams chat to Matthieu DIERICK.



1.1 Class 1 - Deploy modern application with modern tools

In this class, we will deploy a modern application (Arcadia Finance app) with modern tools in a modern environment.

What are modern tools:

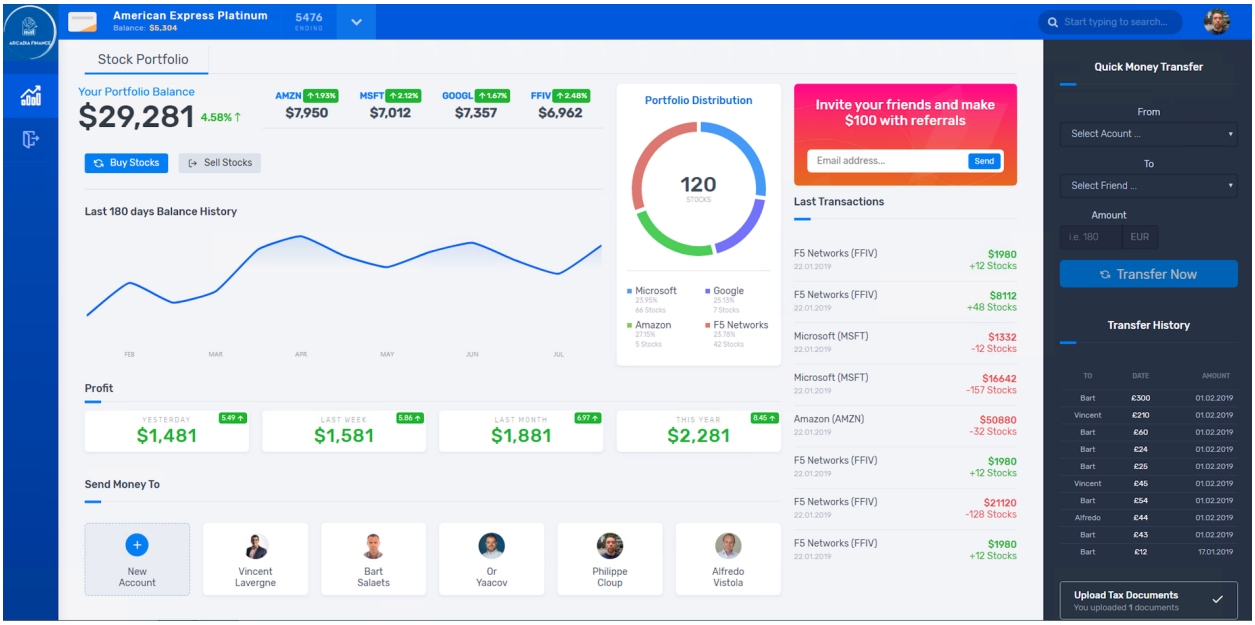
- Ansible
- Terraform
- Gitlab and Gitlab CI

What is a modern environment:

- Kubernetes
- Docker containers with docker registry

Note: Don't be afraid if you don't know those tools. The goal of the lab is not to learn how to deploy them, but how to use them.

First of all, this is Arcadia Finance application



Class 1 - All sections

1.1.1 Architecture of Arcadia Application

Note: This application is available in GitLab in case you want to build your own lab :

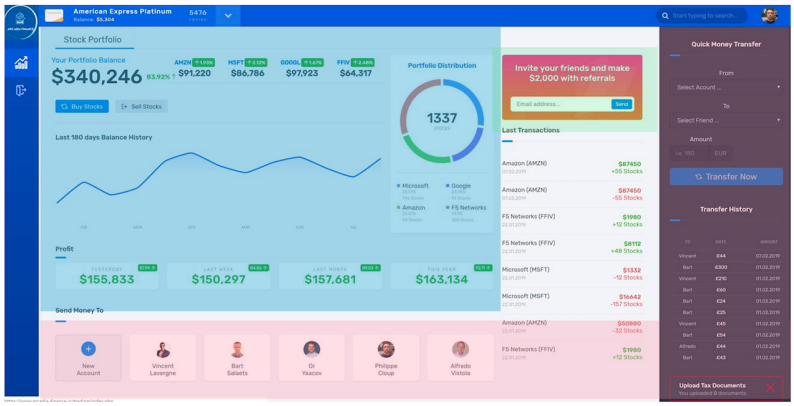
First of all, it is important to understand how Arcadia app is split between micro-services

This is what Arcadia App looks like when the 4 microservices are up and running, and you can see how traffic is routed based on URI

API schema

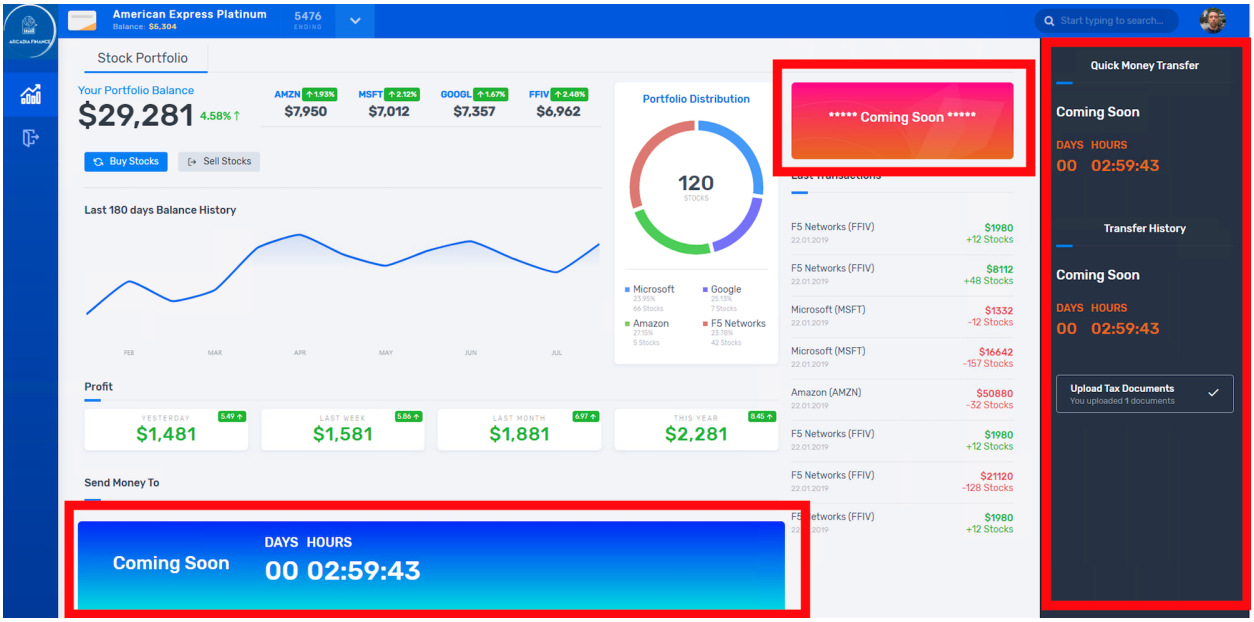
ARCADIA FINANCE

- 4 microservices deployed
  - Main App
    - /*
  - Back End
    - /files
  - App2 (Money Transfer)
    - /api
  - App3 (Refer Friend)
    - /app3



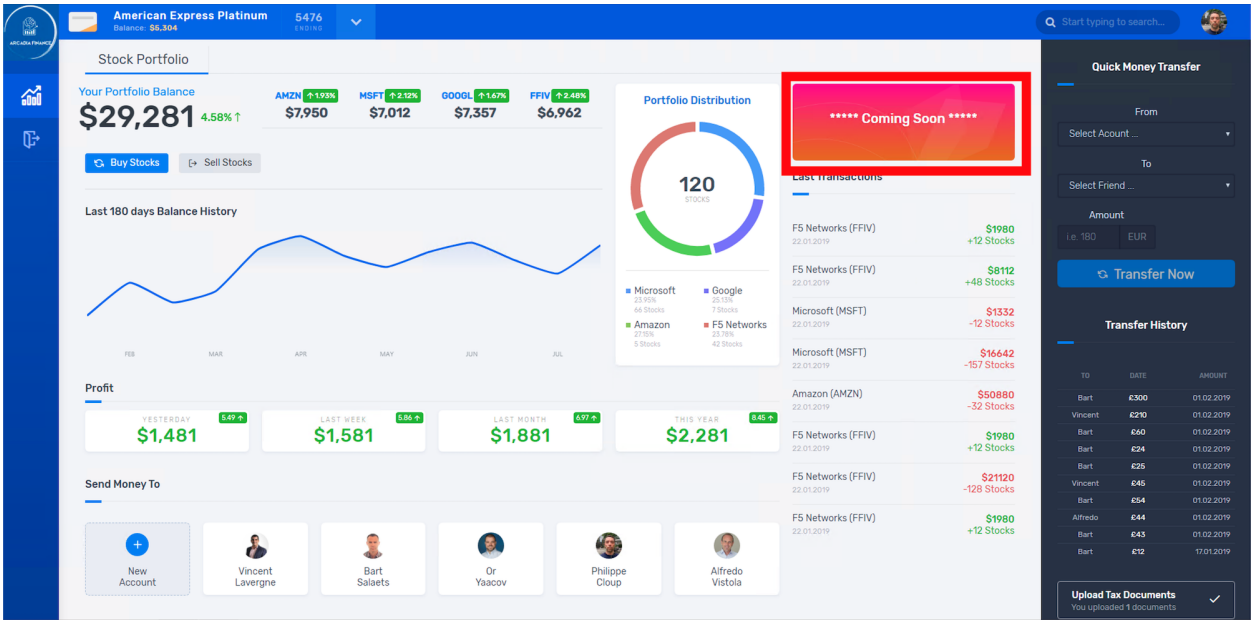
But you can deploy Arcadia Step by Step

If you deploy only `Main App` and `Back End` services.

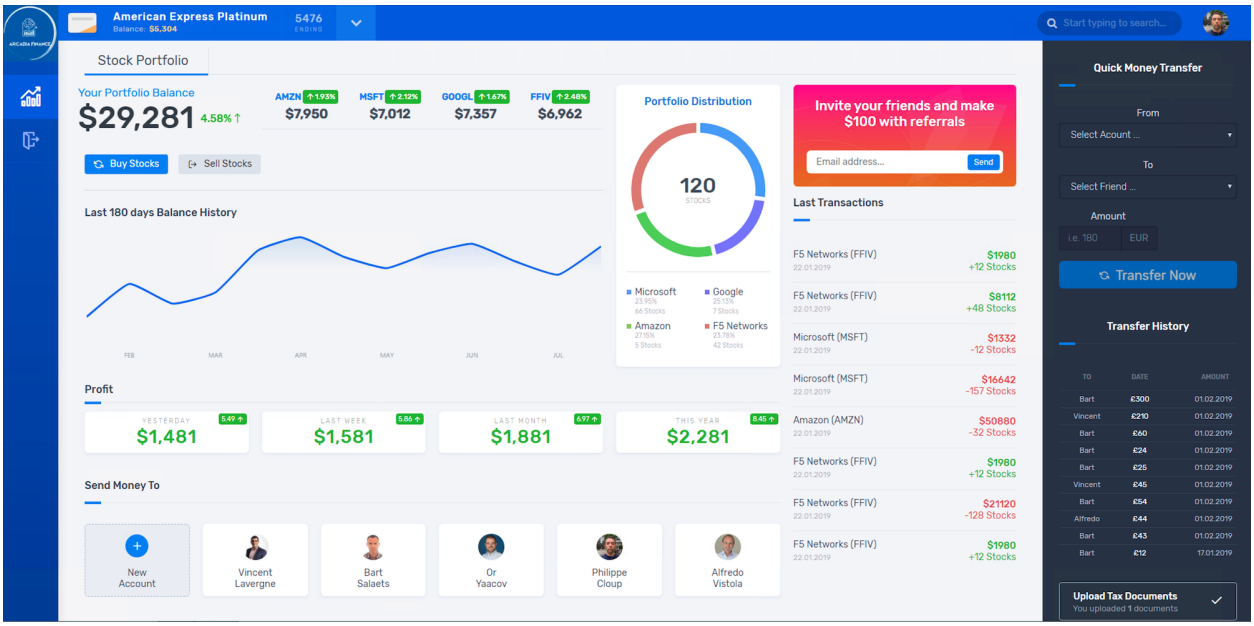


Note: You can see App2 (Money Transfer) and App3 (Refer Friend) are not available. There is dynamic content showing a WARNING instead of a 404 or blank frame.

If you deploy `Main App`, `Back End` and `Money Tranfer` services.



If you deploy `Main App`, `Back End`, `Money Tranfer` and `Refer Friend` services.



1.1.2 Workflow of this lab

The demo is split into 3 classes and 9 steps :

1. **Deploy modern application with modern tools**
    1. Deploy and publish Arcadia Finance application in Kubernetes
    2. Publish Arcadia app with an NGNIX Plus Ingress Controller
2. **Protect Arcadia with NGINX App Protect in Docker**
    1. Build your first NAP (NGINX App Protect) docker image
    2. Update this image with the latest WAF signatures
    3. Check logs in Kibana
    4. Customize the WAF policy
    5. Deploy NAP with a CI/CD a toolchain
3. **Protect Arcadia with NGINX App Protect in Linux host**
    1. Install the NGINX Plus and App Protect packages manually
    2. Deploy App Protect via CI/CD pipeline

Step 1 - Deploy and publish Arcadia Finance application in Kubernetes

Note: Goal is to deploy Arcadia Application in Kubernetes

Tasks:

1. Run a Kubernetes command (kubectl) that will download Arcadia containers from an external public repo (Gitlab.com), and run them
2. Check in Kubernetes Dashboard if Arcadia is deployed and runnning

Step 2 - Publish Arcadia app with a NGINX Plus Ingress Controller

Note: Goal is to publish Arcadia application outside the Kubernetes cluster and use NGINX Plus Ingress Controller for that

Tasks:

1. Run a Kubernetes command (kubectl) that will download and run an NGINX Plus Ingress Controller image from a private repo (Gitlab.com)
2. Check how this NIC (NGINX Ingress Controller) is set in order to route packets to the right Arcadia container (pod)

Step 3 - Build your first NAP (NGINX App Protect) docker image

Note: Goal is to build your first NAP docker image and run it

Tasks:

1. Run a docker build command using a Dockerfile
2. Run a docker run command to start this docker container in front of Arcadia application
3. Check the signature package included in this image
4. Check that Aracadia is protected

Step 4 - Update this image with the latest WAF signature

Note: Goal is to create a new NAP image with the latest Signature package.

Task:

1. Run the same Docker build command but with a new Dockerfile containing the new repo with the signatures
2. Destroy the previous NAP container and run a new one from this new image
3. Check the signature date

Step 5 - Update the Docker image with the Threat Campaign package

Note: Goal is to create a new NAP image with the latest Threat Campaign package ruleset.

Task:

1. Run the same Docker build command but with a new Dockerfile containing the new package to install
2. Destroy the previous NAP container and run a new one from this new image
3. Check the Threat Campaign ruleset date

Step 6 - Check logs in Kibana

Note: Goal is to check logs in ELK (Elastic, Logstash, Kibana)

Task:

1. Connect to Kibana and check logs

Step 7 - Customize the WAF policy

Note: Goal is to customize the WAF policy in front of Arcadia application. By default, a base policy is deployed.

Task:

1. Run NAP container with a new nginx.conf file refering to the new policies

Step 8 - Deploy NAP with a CI/CD toolchain

Note: Goal is to deploy NAP in a real environment with a CI/CD toolchain in place.

Task:

1. Upload a new signature package into the local repo (gitlab) or ask for an update
2. GitLab CI build a new version of the NAP image with this new signature package
3. Deploy and run this new version of the NAP image in front of Arcadia
4. Check the signature package date

Step 9 - Install the NGINX Plus and App Protect packages manually

Note: Goal is to deploy NAP and NGINX Plus in a CentOS linux host.

Task:

1. Install NGINX Plus r20
2. Install NGINX App Protect

3. Install NGINX App Protect Signature Package

Step 10 - Deploy App Protect via CI/CD pipeline

Note: Goal is to deploy NAP by using a CI/CD pipeline with automation toolchain packages provided by F5.

Task:

1. Use CI/CD toolchain in order to deploy NAP automatically with the latest signature package.

Step 11 - Deploy a new version of the NGINX Plus Ingress Controller

Note: Goal is to deploy NAP in the Kubernetes Ingress Controller. Since NAP v1.3, NAP can be deployed in a KIC with NGINX+

Task:

1. Pull NGINX+ KIC image from my private Gitlab repo
2. Deploy a new Ingress configuration with NAP annotations and configuration

Step 12 - API Security with OpenAPI file import

Note: Goal is to deploy NAP in Centos to protect an API

Tasks:

1. Push OpenAPI file to a repo (swaggerhub in this lab)
2. Create a new NAP policy based on this OAS file

1.1.3 Step 1 - Deploy and publish Arcadia Finance application in Kubernetes

It's time to deploy Arcadia Finance application :)

Deploy Arcadia Application with kubectl command

With Kubernetes, there are several ways to deploy containers (pods). One way is to use `kubectl` command with a YAML deployment file. I prepared this YAML file below (this is only for the main app container). You can have a look, and see it will deploy containers from my Gitlab.com repo.

```
apiVersion: v1
kind: Service
metadata:
name: main
namespace: default
```

(continues on next page)

(continued from previous page)

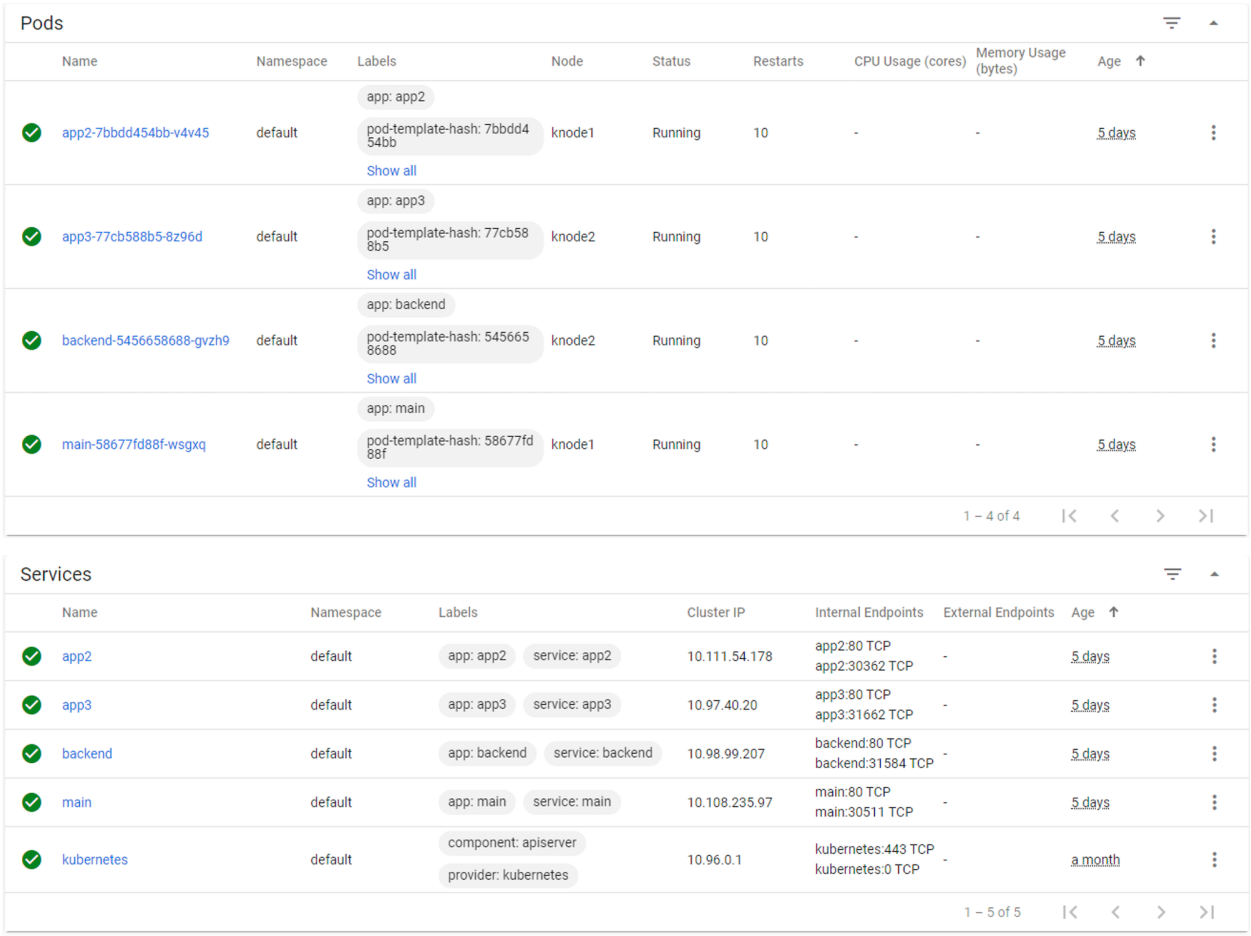
```
labels:
    app: main
spec:
type: NodePort
ports:
- name: main-80
    nodePort: 30511
    port: 80
    protocol: TCP
    targetPort: 80
selector:
    app: main
---
apiVersion: apps/v1
kind: Deployment
metadata:
name: main
namespace: default
labels:
    app: main
    version: v1
spec:
replicas: 1
selector:
    matchLabels:
    app: main
    version: v1
template:
    metadata:
    labels:
        app: main
        version: v1
    spec:
    containers:
    - env:
        - name: service_name
        value: main
        image: registry.gitlab.com/arcadia-application/main-app/mainapp:latest
        imagePullPolicy: IfNotPresent
        name: main
        ports:
        - containerPort: 80
        protocol: TCP
---
```

Note: To make it simple, it deploys the container from gitlab.com repo, and a service. The service is used later on by the NGINX Plus Ingress Controller.

Steps :

1. SSH (or WebSSH and `cd /home/ubuntu/`) to CICD Server
2. Run this command `kubectl apply -f /home/ubuntu/Arcadia_k8S/all_apps.yaml`
3. RDP to the jumphost with `user:user` as credentials
4. Open Chrome

5. Open Kubernetes Dashboard bookmark (if not already opened)
6. Click `skip` on the logon page
7. You should see the services and the pods



Pods

	Name	Namespace	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Age
✓	app2-7bbdd454bb-v4v45	default	app: app2 pod-template-hash: 7bbdd454bb Show all	knode1	Running	10	-	-	5 days
✓	app3-77cb588b5-8z96d	default	app: app3 pod-template-hash: 77cb588b5 Show all	knode2	Running	10	-	-	5 days
✓	backend-5456658688-gvzh9	default	app: backend pod-template-hash: 5456658688 Show all	knode2	Running	10	-	-	5 days
✓	main-58677fd88f-wsgxq	default	app: main pod-template-hash: 58677fd88f Show all	knode1	Running	10	-	-	5 days

1 – 4 of 4

Services

	Name	Namespace	Labels	Cluster IP	Internal Endpoints	External Endpoints	Age
✓	app2	default	app: app2 service: app2	10.111.54.178	app2:80 TCP app2:30362 TCP	-	5 days
✓	app3	default	app: app3 service: app3	10.97.40.20	app3:80 TCP app3:31662 TCP	-	5 days
✓	backend	default	app: backend service: backend	10.98.99.207	backend:80 TCP backend:31584 TCP	-	5 days
✓	main	default	app: main service: main	10.108.235.97	main:80 TCP main:30511 TCP	-	5 days
✓	kubernetes	default	component: apiserver provider: kubernetes	10.96.0.1	kubernetes:443 TCP kubernetes:0 TCP	-	a month

1 – 5 of 5

Warning: Arcadia Application is running but not yet available for the customers. We need to publish it.

Video of this module (force HD 1080p in the video settings)

1.1.4 Step 2 - Publish Arcadia app with a NGINX Plus Ingress Controller

It's time to publish Arcadia application externally from the Kubernetes cluster.

Deploy the NGINX Plus Ingress Controller

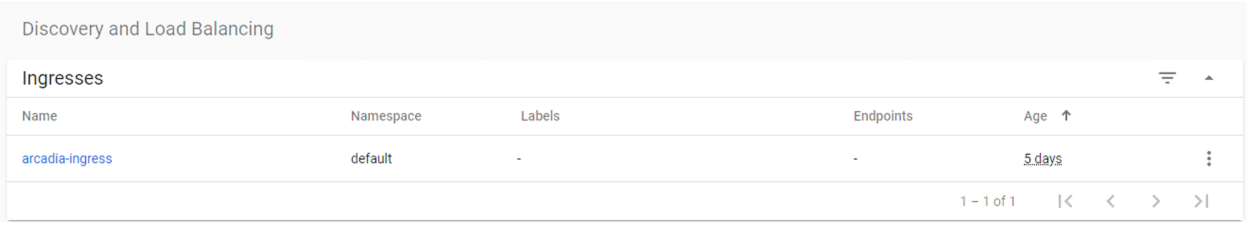
Now, Arcadia App is running in the Kubernetes Cluster. We need a solution to publish it externally (using Kubernetes front end IP addresses) and routing the packets to the right pods (main, back, app2, app3)

To do so, I prepared a `kubectl` Kubernetes Deployment in YAML.

Steps:

1. SSH (or WebSSH and `cd /home/ubuntu/`) to CICD Server

2. Run this command `kubectl apply -f /home/ubuntu/k8s_ingress/full_ingress_arcadia.yaml`
3. You should now see a new namespace `nginx-ingress` and a new ingress in the Kubernetes Dashboard on the Jumphost
4. Check the Ingress `arcadia-ingress` (in the `default` namespace) by clicking on the 3 dots on the right and `edit`
5. Scroll down and check the specs




```
spec:
rules:
    - host: k8s.arcadia-finance.io
    http:
        paths:
        - path: /
            pathType: ImplementationSpecific
            backend:
            serviceName: main
            servicePort: 80
        - path: /files
            pathType: ImplementationSpecific
            backend:
            serviceName: backend
            servicePort: 80
        - path: /api
            pathType: ImplementationSpecific
            backend:
            serviceName: app2
            servicePort: 80
        - path: /app3
            pathType: ImplementationSpecific
            backend:
            serviceName: app3
            servicePort: 80
```

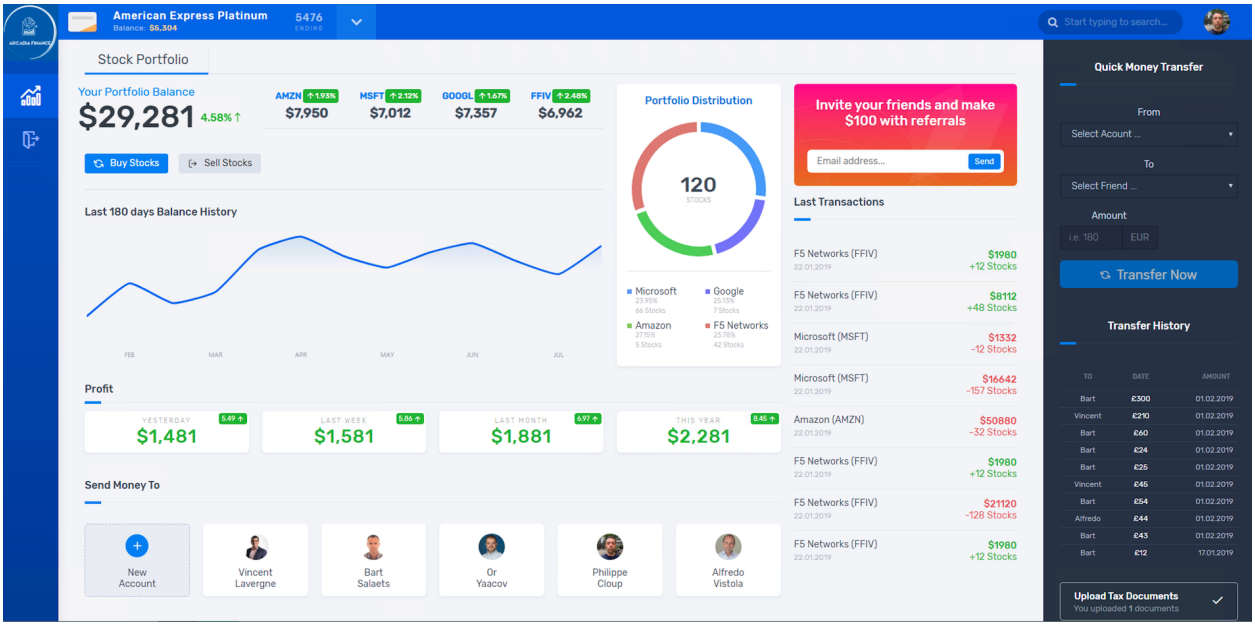
Note: You can see the Ingress is routing the packets to the right service based on the URI.

Note: Now, Arcadia is available for customers.

Steps:

1. In Chrome, click on the bookmark `Arcadia k8s`
2. Click on `Login`
3. Login with `matt:ilovef5`

4. You should see all the apps running (main, back, app2 and app3)



1.2 Class 2 - Protect Arcadia with NGINX App Protect in Docker

In this class, we will deploy App Protect with several methods. We will start with a manual method building the Docker images, and we will finish with a full CI/CD pipeline workflow.

Note: At the moment, this lab does not cover the deployment of NAP in a Linux host. It covers only Docker deployment. You can use Frida's UDF blueprint to know how to deploy NAP in Linux host.

1.2.1 Step 3 - Build your first NAP (NGINX App Protect) Docker image

In this module, we will build manually our first NAP Docker image via command line.

Follow the step below to build the Docker image:

1. SSH (or WebSSH and `cd /home/ubuntu/`) to Docker App Protect + Docker repo VM
2. Run the command `docker build -t app-protect:nosig .` <– Be careful, there is a "." (dot) at the end of the command
3. Wait until you see the message: `Successfully tagged app-protect:nosig`

Note: This command execute the Dockerfile below

```
#For CentOS 7
FROM centos:7.4.1708

# Download certificate and key from the customer portal (https://cs.nginx.
↪com)
```

(continues on next page)

(continued from previous page)

```
# and copy to the build context
COPY nginx-repo.crt nginx-repo.key /etc/ssl/nginx/

# Install prerequisite packages
RUN yum -y install wget ca-certificates epel-release

# Add NGINX Plus repo to yum
RUN wget -P /etc/yum.repos.d https://cs.nginx.com/static/files/nginx-plus-7.
↪repo

# Install NGINX App Protect
RUN yum -y install app-protect \
    && yum clean all \
    && rm -rf /var/cache/yum \
    && rm -rf /etc/ssl/nginx

# Forward request logs to Docker log collector
#RUN ln -sf /dev/stdout /var/log/nginx/access.log \
#    && ln -sf /dev/stderr /var/log/nginx/error.log

# Copy configuration files
COPY nginx.conf log-default.json /etc/nginx/
COPY entrypoint.sh  ./

CMD ["sh", "/entrypoint.sh"]
```

When Docker image is built :

1. Check if the app-protect Docker image is available locally by running `docker images`

```
ubuntu@ip-10-1-1-5:~$ docker images
REPOSITORY                TAG                 IMAGE ID            CREATED             SIZE
app-protect               nosig               41b8d1c21137        47 seconds ago      622MB
joxit/docker-registry-ui  static              7c9411ced70c        3 months ago        22.6MB
joxit/docker-registry-ui  latest              9c812b360653        3 months ago        22.6MB
registry                  2                   708bc6af7e5e        3 months ago        25.8MB
centos                    7.4.1708            9f266d35e02c        14 months ago       197MB
```

2. Run a container with this image `docker run -dit --name app-protect -p 80:80 -v /home/ubuntu/nginx.conf:/etc/nginx/nginx.conf app-protect:nosig`
3. Check that the Docker container is running `docker ps`

```
ubuntu@ip-10-1-1-5:~$ docker ps
CONTAINER ID        IMAGE                                COMMAND                  CREATED             STATUS              PORTS                    NAMES
44df893d16ec        app-protect:nosig                    "/root/entrypoint.sh"    3 seconds ago       Up 2 seconds        0.0.0.0:80->80/tcp       app-protect
a89aef04940d        joxit/docker-registry-ui:static      "/bin/sh -c entrypoi…"   23 hours ago        Up 10 hours         0.0.0.0:8080->80/tcp     registry-ui
1e0dfd08c3e8        registry:2                           "/entrypoint.sh /etc…"   29 hours ago        Up 10 hours         0.0.0.0:5000->5000/tcp   registry
ubuntu@ip-10-1-1-5:~$
```

4. Check the signature package date included in this image (by default) `docker exec -it app-protect more /var/log/nginx/error.log`

```
2020/05/19 16:59:29 [notice] 12#12: APP_PROTECT { "event": "configuration_
↪load_success", "attack_signatures_package":{"revision_datetime":"2019-07-
↪16T12:21:31Z"},"completed_successfully":true}
```

It's time to test your lab

1. In Chrome, click on the bookmark `Arcadia NAP Docker`
2. Navigate in the app, and try some attacks like injections or XSS - I let you find the attacks :)
3. You will be blocked and see the default Blocking page

```
The requested URL was rejected. Please consult with your administrator.

Your support ID is: 14609283746114744748

[Go Back]
```

Note: Did you notice the blocking page is similar to ASM and Adv. WAF ?

Video of this module (force HD 1080p in the video settings)

Warning: You can notice some differences between the video and the lab. When I did the video, the dockerfile was different. But the concept remains the same.

1.2.2 Step 4 - Update the Docker image with the latest WAF signatures

In this module, we will update the signature package in the Docker image.

Warning: There are several ways to update the signatures. All of them have pros and cons. In this lab, I decided to create a new Docker image with the new signature package to preserve immutability. And then destroy and run a new Docker container from this new image in front of Arcadia App.

The signatures are provided by F5 with an RPM package. The best way to update the image is to build a new image from a new Dockerfile referring to this signature package (and change the image tag). We will use the Dockerfile below:

```
#For CentOS 7
FROM centos:7.4.1708

# Download certificate and key from the customer portal (https://cs.nginx.com)
# and copy to the build context
COPY nginx-repo.crt nginx-repo.key /etc/ssl/nginx/

# Install prerequisite packages
RUN yum -y install wget ca-certificates epel-release

# Add NGINX Plus repo to yum
RUN wget -P /etc/yum.repos.d https://cs.nginx.com/static/files/nginx-plus-7.repo
RUN wget -P /etc/yum.repos.d https://cs.nginx.com/static/files/app-protect-signatures-
↪7.repo
```

(continues on next page)

(continued from previous page)

```
# Install NGINX App Protect
RUN yum -y install app-protect app-protect-attack-signatures\
   && yum clean all \
   && rm -rf /var/cache/yum \
   && rm -rf /etc/ssl/nginx

# Forward request logs to Docker log collector
#RUN ln -sf /dev/stdout /var/log/nginx/access.log \
#    && ln -sf /dev/stderr /var/log/nginx/error.log

# Copy configuration files
COPY nginx.conf log-default.json /etc/nginx/
COPY entrypoint.sh  ./

CMD ["sh", "/entrypoint.sh"]
```

Note: You may notice one more package versus the previous Dockerfile in Step 3. I added the package installation `app-protect-attack-signatures`

Follow the steps below to build the new Docker image:

1. SSH to Docker App Protect + Docker repo VM
2. Run the command `docker build -t app-protect:20200316 -f Dockerfile-sig .` <– Be careful, there is a "." (dot) at the end of the command
3. Wait until you see the message: `Successfully tagged app-protect:20200316`

Note: Please take time to understand what we ran. You may notice 2 changes. We ran the build with a new Dockerfile `Dockerfile-sig` and with a new tag `20200316` (date of the signature package when I built this lab). You can put any tag you want, for instance the date of today. Because we don't know the date of the latest Attack Signature package.

Destroy the previous running NAP container and run a new one based on the new image (tag 20200316)

1. Check if the new app-protect Docker image is available locally by running `docker images`. You will notice the new image with a tag of `20200316`.

```
ubuntu@ip-10-1-1-5:~$ docker images
REPOSITORY                TAG                 IMAGE ID            CREATED             SIZE
app-protect               20200316            2628f7b9f24d        44 seconds ago      624MB
app-protect               nosig               41b8d1c21137        17 hours ago        622MB
joxit/docker-registry-ui  static              7c9411ced70c        3 months ago        22.6MB
joxit/docker-registry-ui  latest              9c812b360653        3 months ago        22.6MB
registry                  2                   708bc6af7e5e        3 months ago        25.8MB
centos                    7.4.1708            9f266d35e02c        14 months ago       197MB
ubuntu@ip-10-1-1-5:~$
```

2. Destroy the existing and running NAP container `docker rm -f app-protect`

3. Run a new container with this image `docker run -dit --name app-protect -p 80:80 -v /home/ubuntu/nginx.conf:/etc/nginx/nginx.conf app-protect:20200316`

> **Warning:** If you decided to change the tag `20200316` by another tag, change your command line accordingly

4. Check that the Docker container is running `docker ps`

```
ubuntu@ip-10-1-1-5:~$ docker ps
CONTAINER ID        IMAGE                              COMMAND                  CREATED             STATUS              PORTS                    NAMES
513d25345119        app-protect:20200316               "/root/entrypoint.sh"    3 seconds ago       Up 1 second         0.0.0.0:80->80/tcp       app-protect
a89aef04940d        joxit/docker-registry-ui:static    "/bin/sh -c entrypoi…"   40 hours ago        Up 26 hours         0.0.0.0:8080->80/tcp     registry-ui
1e0dfd08c3e8        registry:2                         "/entrypoint.sh /etc…"   45 hours ago        Up 26 hours         0.0.0.0:5000->5000/tcp   registry
ubuntu@ip-10-1-1-5:~$
```

5. Check the signature package date included in the new Docker container `docker exec -it app-protect more /var/log/nginx/error.log`

```
2020/05/20 09:30:20 [notice] 12#12: APP_PROTECT { "event": "configuration_
↪load_success", "attack_signatures_package":{"revision_datetime":"2020-03-
↪16T14:11:52Z","version":"2020.03.16"},"completed_successfully":true}
```

Note: Congrats, you are running a new version of NAP with an updated signature package.

Video of this module (force HD 1080p in the video settings)

Note: You can notice some differences between the video and the lab. When I did the video, the dockerfile was different. But the concept remains the same.

1.2.3 Step 5 - Update the Docker image with the Threat Campaign package

In this module, we will install the package Threat Campaign into a new Docker image.

Threat Campaign is a **feed** from F5 Threat Intelligence team. This team is collecting 24/7 threats from internet and darknet. They use several bots and honeypotting networks in order to know in advance what the hackers (humans or robots) will target and how.

Unlike `signatures`, Threat Campaign provides with `ruleset`. A signature uses patterns and keywords like `' or` or `1=1`. Threat Campaign uses `rules` that match perfectly an attack detected by our Threat Intelligence team.

Note: The App Protect installation does not come with a built-in Threat campaigns package like Attack Signatures. Threat campaigns Updates are released periodically whenever new campaigns and vectors are discovered, so you might want to update your Threat campaigns from time to time. You can upgrade the Threat campaigns by updating the package any time after installing App Protect. We recommend you upgrade to the latest Threat campaigns version right after installing App Protect.

For instance, if we notice a hacker managed to enter into our Struts2 system, we will do forensics and analyse the packet that used the breach. Then, this team creates the `rule` for this request. A `rule` **can** contains all the HTTP L7 payload (headers, cookies, payload . . .)

Note: Unlike signatures that can generate False Positives due to low accuracy patterns, Threat Campaign is very accurate and reduces drastically the False Positives.

Note: NAP provides with high accuracy Signatures + Threat Campaign ruleset. The best of bread to reduce FP.

Threat Campaign package is available with the `app-protect-signatures-7.repo` repository. It is provided with the NAP subscription.

In order to install this package, we need to update our `Dockerfile`. I created another Dockerfile named `Dockerfile-sig-tc`

```
#For CentOS 7
FROM centos:7.4.1708

# Download certificate and key from the customer portal (https://cs.nginx.com)
# and copy to the build context
COPY nginx-repo.crt nginx-repo.key /etc/ssl/nginx/

# Install prerequisite packages
RUN yum -y install wget ca-certificates epel-release

# Add NGINX Plus repo to yum
RUN wget -P /etc/yum.repos.d https://cs.nginx.com/static/files/nginx-plus-7.repo
RUN wget -P /etc/yum.repos.d https://cs.nginx.com/static/files/app-protect-signatures-
↪7.repo

# Install NGINX App Protect
RUN yum -y install app-protect app-protect-attack-signatures app-protect-threat-
↪campaigns\
   && yum clean all \
   && rm -rf /var/cache/yum \
   && rm -rf /etc/ssl/nginx

# Forward request logs to Docker log collector
#RUN ln -sf /dev/stdout /var/log/nginx/access.log \
#    && ln -sf /dev/stderr /var/log/nginx/error.log

# Copy configuration files
COPY nginx.conf log-default.json /etc/nginx/
COPY entrypoint.sh  ./

CMD ["sh", "/entrypoint.sh"]
```

Note: You may notice one more package versus the previous Dockerfile in Step 4. I added the package installation `app-protect-threat-campaigns`

Follow the steps below to build the new Docker image:

1. SSH to Docker App Protect + Docker repo VM
2. Run the command `docker build -t app-protect:tc -f Dockerfile-sig-tc .` <– Be careful, there is a "." (dot) at the end of the command
3. Wait until you see the message: `Successfully tagged app-protect:tc`

Note: Please take time to understand what we ran. You may notice 2 changes. We ran the build with a new Dockerfile `Dockerfile-sig-tc` and with a new tag `tc`. You can choose another tag like `tcdate` where date is the date of today. We don't know yet the date of the TC package ruleset.

Destroy the previous running NAP container and run a new one based on the new image (tag tc)

1. Check if the new app-protect Docker image is available locally by running `docker images`. You will notice the new image with a tag of `tc`.

```
ubuntu@ip-10-1-1-5:~$ docker images
REPOSITORY                  TAG                 IMAGE ID            CREATED             SIZE
app-protect                 tc                  4d539837ef35        2 minutes ago       548MB
app-protect                 20200316            06b4f496b54a        5 minutes ago       547MB
app-protect                 nosig               987fd0d594d6        7 minutes ago       542MB
joxit/docker-registry-ui    static              7c9411ced70c        5 months ago        22.6MB
joxit/docker-registry-ui    latest              9c812b360653        5 months ago        22.6MB
registry                    2                   708bc6af7e5e        5 months ago        25.8MB
centos                      7.4.1708            9f266d35e02c        15 months ago       197MB
ubuntu@ip-10-1-1-5:~$
```

2. Destroy the existing and running NAP container `docker rm -f app-protect`
3. Run a new container with this image `docker run -dit --name app-protect -p 80:80 -v /home/ubuntu/nginx.conf:/etc/nginx/nginx.conf app-protect:tc`

> **Warning:** If you decided to change the tag `tc` by another tag, change your command line accordingly

4. Check that the Docker container is running `docker ps`

```
ubuntu@ip-10-1-1-5:~$ docker ps
CONTAINER ID        IMAGE                              COMMAND                  CREATED             STATUS              PORTS                    NAMES
3a104c34e6a3        app-protect:tc                     "sh /entrypoint.sh"      7 seconds ago       Up 6 seconds        0.0.0.0:80->80/tcp       app-protect
a89aef04940d        joxit/docker-registry-ui:static    "/bin/sh -c entrypoi…"   6 weeks ago         Up 31 minutes       0.0.0.0:8080->80/tcp     registry-ui
1e0dfd08c3e8        registry:2                         "/entrypoint.sh /etc…"   6 weeks ago         Up 31 minutes       0.0.0.0:5000->5000/tcp   registry
ubuntu@ip-10-1-1-5:~$
```

5. Check the Threat Campaign ruleset date included in the new Docker container `docker exec -it app-protect cat /var/log/nginx/error.log`

> **Note:** You can notice in one line of log, you get the `Signature date` and the `Threat Campaign date`.

```
2020/07/01 17:03:14 [notice] 12#12: APP_PROTECT { "event": "configuration_
↪load_success", "software_version": "3.74.0", "attack_signatures_package":{
↪"revision_datetime":"2020-06-28T15:30:59Z","version":"2020.06.28"},
↪"completed_successfully":true,"threat_campaigns_package":{"revision_
↪datetime":"2020-06-25T19:13:36Z","version":"2020.06.25"}}
```

Simulate a Threat Campaign attack

1. RDP to the `Jumphost` (user / user)
2. Open `Postman` and select the collection `NAP - Threat Campaign`
3. Run 2 calls.
    1. Strust2 Jakarta attack
    2. Drupal attack
    3. The NMAP attack is now detected as a Bot. We added Bot Protection in NAP v2.1. Do not run it.
4. In the next module, we will check the logs in Kibana.

Note: Congrats, you are running a new version of NAP with the latest Threat Campaign package and ruleset.

Video of this module (force HD 1080p in the video settings)

1.2.4 Step 6 - Check logs in Kibana

In this module, we will check the logs in ELK (Elastic, Logstash, Kibana)

Check how logs are sent and how to set the destination syslog server

Steps:

1. SSH to Docker App Protect + Docker repo VM
2. In `/home/ubuntu` (the default home folder), list the files `ls -al`
3. You can see 2 files `log-default.json` and `nginx.conf`
4. Open log-default.json `less log-default.json`. You will notice we log all requests.

```
{
"filter": {
   "request_type": "all"
      },
"content": {
   "format": "default",
   "max_request_size": "any",
   "max_message_size": "5k"
      }
}
```

5. Open nginx.conf `less nginx.conf`

```
user nginx;

worker_processes 1;
load_module modules/ngx_http_app_protect_module.so;

error_log /var/log/nginx/error.log debug;

events {
   worker_connections  1024;
}
```

(continues on next page)

(continued from previous page)

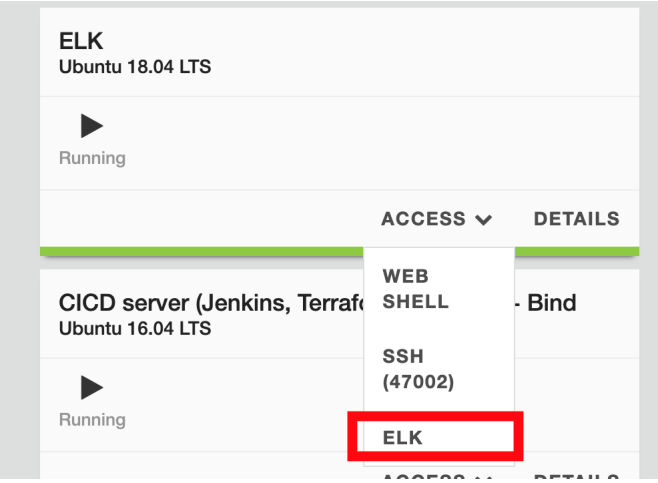
```
http {
   include       /etc/nginx/mime.types;
   default_type  application/octet-stream;
   sendfile        on;
   keepalive_timeout  65;

   server {
      listen       80;
      server_name  localhost;
      proxy_http_version 1.1;

      app_protect_enable on;
      app_protect_security_log_enable on;
      app_protect_security_log "/etc/nginx/log-default.json" syslog:server=10.1.
↪20.6:5144;

      location / {
            resolver 10.1.1.9;
            resolver_timeout 5s;
            client_max_body_size 0;
            default_type text/html;
            proxy_pass http://k8s.arcadia-finance.io:30274$request_uri;
      }
   }
}
```

Note: You will notice in the `nginx.conf` file the refererence to `log-default.json` and the remote syslog server (ELK) `10.1.20.6:5144`

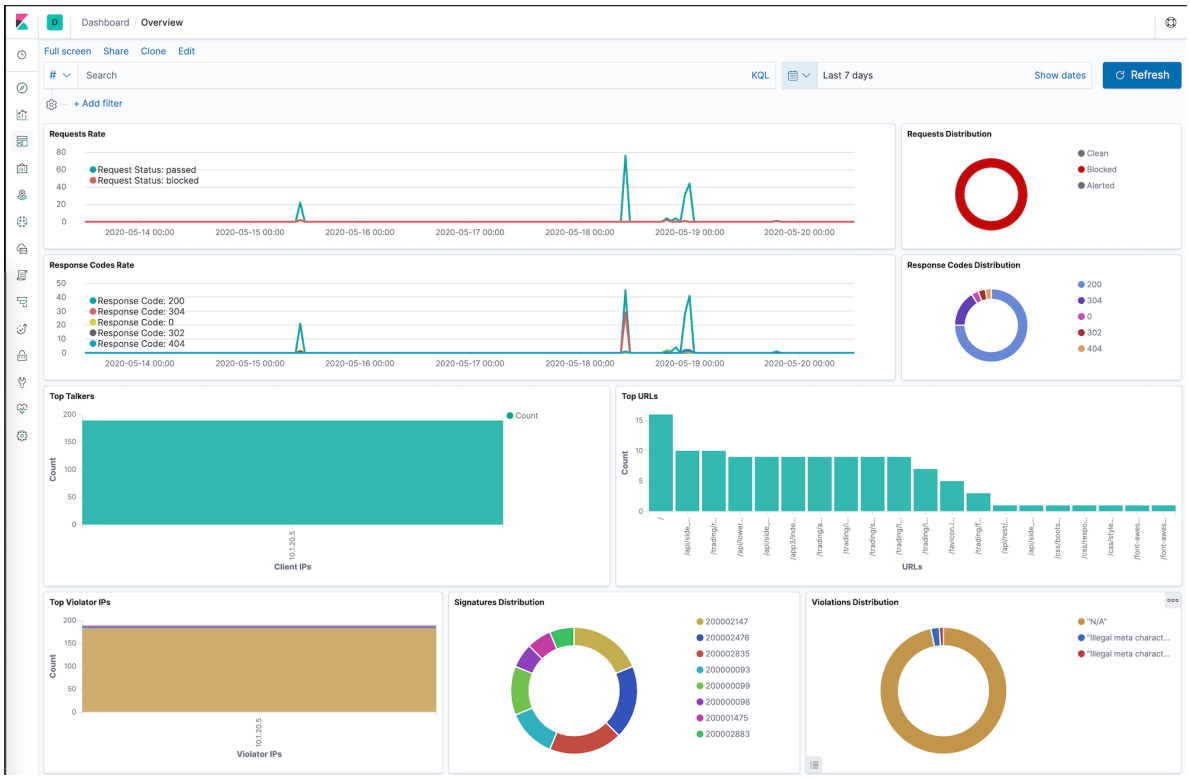
Open Kibana in the Jumphost or via UDF access

Steps:

1. In UDF, find the ELK VM and click Access > ELK



1. In Kibana, click on `Dashboard > Overview`



1. At the bottom of the dashboard, you can see the logs. Select one of the log entries and check the content

---

Note: You may notice the log content is similar to ASM and Adv. WAF

Note: The default time window in this Kibana dashboard is **Last 15 minutes**. If you do not see any requests, you may need to extend the time window to a larger setting

---

Video of this module (force HD 1080p in the video settings)

1.2.5 Step 7 - Customize the WAF policy

So far, we have been using the default NGINX App Protect policy. As you notices in the previous lab (Step 5), the `nginx.conf` does not file any reference to a WAF policy. It uses the default WAF policy.

In this lab, we will customize the policy and push a new config file to the docker container.

Use a custom WAF policy and assign it per location

Steps:

1. SSH to the Docker App Protect + Docker repo VM
2. In the `/home/ubuntu` directory, create a new folder `policy-adv`

```
mkdir policy-adv
```

3. Create a new policy file named `policy_base.json` and paste the content below

```
vi ./policy-adv/policy_base.json
```

```
{
    "name": "policy_name",
    "template": { "name": "POLICY_TEMPLATE_NGINX_BASE" },
    "applicationLanguage": "utf-8",
    "enforcementMode": "blocking"
}
```

4. Create another policy file named `policy_mongo_linux_JSON.json` and paste the content below

```
vi ./policy-adv/policy_mongo_linux_JSON.json
```

```
{
    "policy":{
    "name":"evasions_enabled",
    "template":{
        "name":"POLICY_TEMPLATE_NGINX_BASE"
    },
    "applicationLanguage":"utf-8",
    "enforcementMode":"blocking",
    "blocking-settings":{
        "violations":[
            {
                "name":"VIOL_JSON_FORMAT",
                "alarm":true,
                "block":true
            },
            {
                "name":"VIOL_EVASION",
                "alarm":true,
                "block":true
            },
            {
                "name": "VIOL_ATTACK_SIGNATURE",
                "alarm": true,
                "block": true
            }
        ],
        "evasions":[
            {
                "description":"Bad unescape",
                "enabled":true,
                "learn":false
            },
            {
```

(continues on next page)

(continued from previous page)

```
                "description":"Directory traversals",
                "enabled":true,
                "learn":false
            },
            {
                "description":"Bare byte decoding",
                "enabled":true,
                "learn":false
            },
            {
                "description":"Apache whitespace",
                "enabled":true,
                "learn":false
            },
            {
                "description":"Multiple decoding",
                "enabled":true,
                "learn":false,
                "maxDecodingPasses":2
            },
            {
                "description":"IIS Unicode codepoints",
                "enabled":true,
                "learn":false
            },
            {
                "description":"IIS backslashes",
                "enabled":true,
                "learn":false
            },
            {
                "description":"%u decoding",
                "enabled":true,
                "learn":false
            }
        ]
    },
    "json-profiles":[
            {
                "defenseAttributes":{
                    "maximumTotalLengthOfJSONData":"any",
                    "maximumArrayLength":"any",
                    "maximumStructureDepth":"any",
                    "maximumValueLength":"any",
                    "tolerateJSONParsingWarnings":true
                },
                "name":"Default",
                "handleJsonValuesAsParameters":false,
                "validationFiles":[

                ],
                "description":"Default JSON Profile"
            }
        ],
    "signature-settings": {
            "attackSignatureFalsePositiveMode": "disabled",
            "minimumAccuracyForAutoAddedSignatures": "low"
```

(continues on next page)

(continued from previous page)

```
    },
    "server-technologies": [
            {
                "serverTechnologyName": "MongoDB"
            },
            {
                "serverTechnologyName": "Unix/Linux"
            },
                    {
                "serverTechnologyName": "PHP"
            }
    ]
    }
}
```

Note: you can notice the difference between the `base` and the `advanced` policy.

5. Now, create a new `nginx.conf` in the `policy-adv` folder. Do not overwrite the existing `/etc/nginx/nginx.conf` file, we need it for the next labs.

```
vi ./policy-adv/nginx.conf
```

```
user nginx;

worker_processes 1;
load_module modules/ngx_http_app_protect_module.so;

error_log /var/log/nginx/error.log debug;

events {
    worker_connections  1024;
}

http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;
    sendfile        on;
    keepalive_timeout  65;

    server {
        listen       80;
        server_name  localhost;
        proxy_http_version 1.1;

        app_protect_enable on;
        app_protect_security_log_enable on;
        app_protect_security_log "/etc/nginx/log-default.json"␣
↪syslog:server=10.1.20.6:5144;

        location / {
            resolver 10.1.1.9;
            resolver_timeout 5s;
            client_max_body_size 0;
            default_type text/html;
```

(continues on next page)

(continued from previous page)

```
            app_protect_policy_file "/etc/nginx/policy/policy_base.json";
            proxy_pass http://k8s.arcadia-finance.io:30274$request_uri;
        }
        location /files {
            resolver 10.1.1.9;
            resolver_timeout 5s;
            client_max_body_size 0;
            default_type text/html;
            app_protect_policy_file "/etc/nginx/policy/policy_mongo_linux_
↪JSON.json";
            proxy_pass http://k8s.arcadia-finance.io:30274$request_uri;
        }
        location /api {
            resolver 10.1.1.9;
            resolver_timeout 5s;
            client_max_body_size 0;
            default_type text/html;
            app_protect_policy_file "/etc/nginx/policy/policy_mongo_linux_
↪JSON.json";
            proxy_pass http://k8s.arcadia-finance.io:30274$request_uri;
        }
        location /app3 {
            resolver 10.1.1.9;
            resolver_timeout 5s;
            client_max_body_size 0;
            default_type text/html;
            app_protect_policy_file "/etc/nginx/policy/policy_mongo_linux_
↪JSON.json";
            proxy_pass http://k8s.arcadia-finance.io:30274$request_uri;
        }

    }
}
```

6. Last step is to run a new container (and delete the previous one) referring to these 3 files.

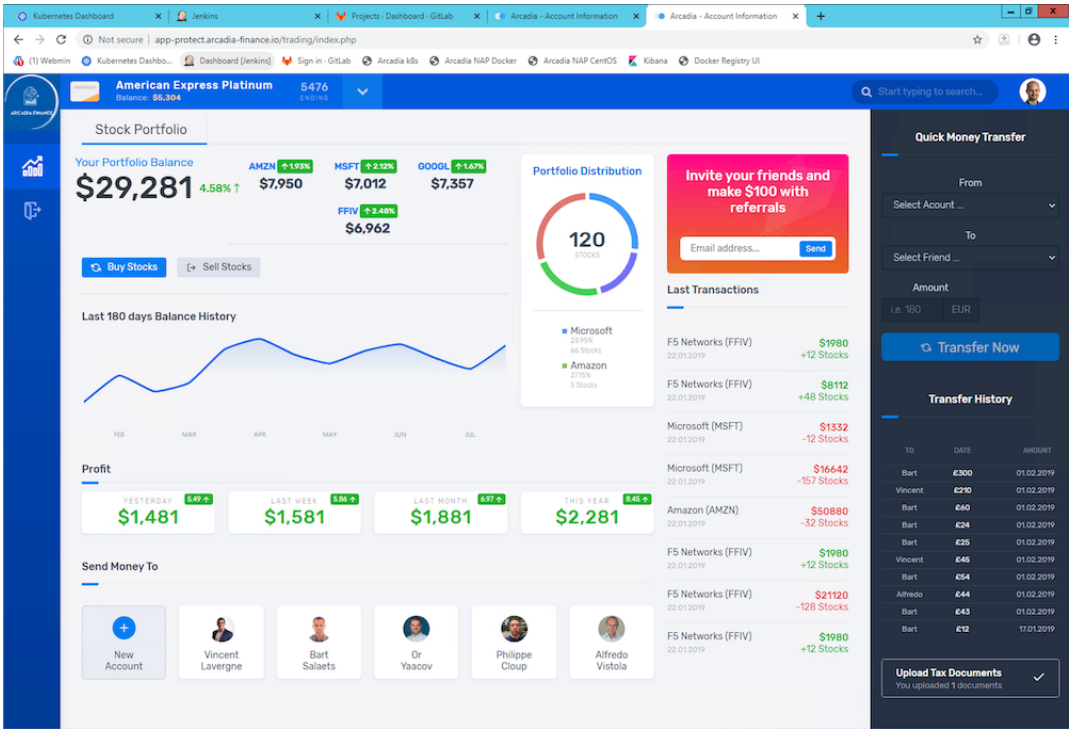
```
docker rm -f app-protect
docker run -dit --name app-protect -p 80:80 -v /home/ubuntu/policy-adv/
↪nginx.conf:/etc/nginx/nginx.conf -v /home/ubuntu/policy-adv/policy_base.
↪json:/etc/nginx/policy/policy_base.json -v /home/ubuntu/policy-adv/
↪policy_mongo_linux_JSON.json:/etc/nginx/policy/policy_mongo_linux_JSON.
↪json  app-protect:20200316
```

7. Check that the `app-protect:20200316` container is running

```
docker ps
```

```
ubuntu@ip-10-1-1-5:~$ docker ps
CONTAINER ID        IMAGE                                COMMAND                  CREATED             STATUS              PORTS                    NAMES
8e88795c3a14        app-protect:20200316                 "/root/entrypoint.sh"    4 minutes ago       Up 4 minutes        0.0.0.0:80->80/tcp       app-protect
a89aef04940d        joxit/docker-registry-ui:static      "/bin/sh -c entrypoi…"   4 days ago          Up 56 minutes       0.0.0.0:8080->80/tcp     registry-ui
1e0dfd08c3e8        registry:2                           "/entrypoint.sh /etc…"   4 days ago          Up 56 minutes       0.0.0.0:5000->5000/tcp   registry
```

8. RDP to the Jumhost as `user:user` and click on bookmark `Arcadia NAP Docker`



Note: From this point on, NAP is using a different WAF policy based on the requested URI:

1. policy_base for `/` (the main app)
2. policy_mongo_linux_JSON for `/files` (the back end)
3. policy_mongo_linux_JSON for `/api` (the Money Transfer service)
4. policy_mongo_linux_JSON for `/app3` (the Refer Friend service)

Use External References to make your policy dynamic

External references in policy are defined as any code blocks that can be used as part of the policy without being explicitly pasted within the policy file. This means that you can have a set of pre-defined configurations for parts of the policy, and you can incorporate them as part of the policy by simply referencing them. This would save a lot of overhead having to concentrate everything into a single policy file.

A perfect use case for external references is when you wish to build a dynamic policy that depends on moving parts. You can have code create and populate specific files with the configuration relevant to your policy, and then compile the policy to include the latest version of these files, ensuring that your policy is always up-to-date when it comes to a constantly changing environment.

Note: To use the external references capability, in the policy file the direct property is replaced by “xxxReference” property, where xxx defines the replacement text for the property. For example, “modifications” section is replaced by “modificationsReference”.

In this lab, we will create a `custom blocking page` and host this page in Gitlab.

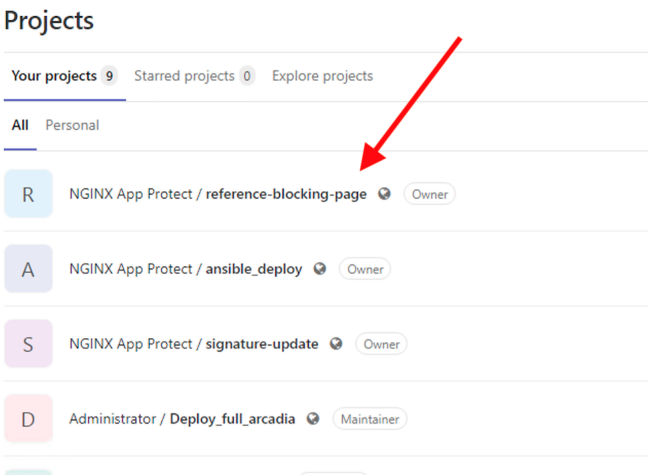
Note: In this configuration, we are completely satisfied with the basic base policy we created previously / `policy-adv/policy_base.json`, and we wish to use it as is. However, we wish to define a custom response page using an external file located on an HTTP web server (Gitlab). The external reference file contains our custom response page configuration.

As a reminder, this is the base policy we created:

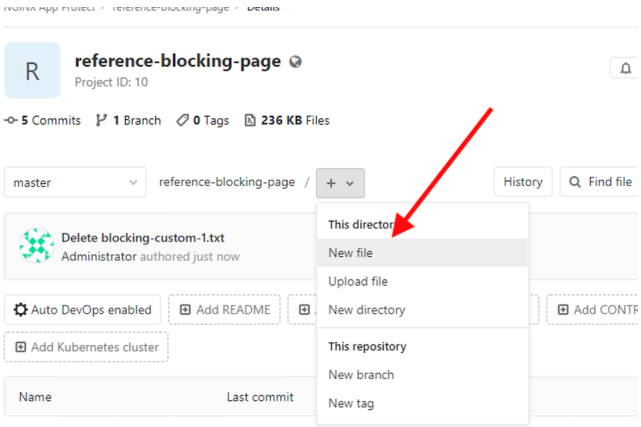
```
{
    "name": "policy_name",
    "template": { "name": "POLICY_TEMPLATE_NGINX_BASE" },
    "applicationLanguage": "utf-8",
    "enforcementMode": "blocking"
}
```

Steps :

1. RDP to `Jumphost` and connect to `GitLab` (root / F5twister$)
2. Click on the project named `NGINX App Protect / reference-blocking-page`



3. Add a new file and name it `blocking-custom-1.txt`



4. Paste the text below

```
[
    {
        "responseContent": "<html><head><title>Custom Reject Page</title>
↪</head><body><p>This is a <strong>custom response page</strong>, it is␣
↪supposed to overwrite the default page for the <strong>base NAP policy.&
↪nbsp;</strong></p><p>This page can be <strong>modified</strong> by a
↪<strong>dedicated</strong> team, which does not have access to the WAF␣
↪policy.<br /><br /></p><p><img src=https://media.giphy.com/media/
↪12NUbkX6p4xOO4/giphy.gif></p><br>Your support ID is: <%TS.request.ID()%>
↪<br><br><a href='javascript:history.back();'>[Go Back]</a></body></html>
↪",
        "responseHeader": "HTTP/1.1 302 OK\\r\\nCache-Control: no-cache\\
↪r\\nPragma: no-cache\\r\\nConnection: close",
        "responseActionType": "custom",
        "responsePageType": "default"
    }
]
```

5. Click `Commit Changes`
6. SSH to `Docker App Protect + Docker repo` VM
7. Delete the running docker

```
docker rm -f app-protect
```

8. Modify the base policy created previously

```
vi ./policy-adv/policy_base.json
```

9. Modify the JSON as below

```
{
    "name": "policy_name",
    "template": { "name": "POLICY_TEMPLATE_NGINX_BASE" },
    "applicationLanguage": "utf-8",
    "enforcementMode": "blocking",
    "responsePageReference": {
        "link": "http://10.1.20.4/nginx-app-protect/reference-blocking-
↪page/-/raw/master/blocking-custom-1.txt"
    }
}
```

Note: You can notice the reference to the TXT file in Gitlab

10. Run a new docker refering to this new JSON policy

```
docker run -dit --name app-protect -p 80:80 -v /home/ubuntu/policy-adv/
↪nginx.conf:/etc/nginx/nginx.conf -v /home/ubuntu/policy-adv/policy_base.
↪json:/etc/nginx/policy/policy_base.json -v /home/ubuntu/policy-adv/
↪policy_mongo_linux_JSON.json:/etc/nginx/policy/policy_mongo_linux_JSON.
↪json  app-protect:tc
```

11. In the `Jumphost`, open `Chrome` and connect to `Arcadia NAP Docker` bookmark
12. Enter this URL with a XSS attack `http://app-protect.arcadia-finance.io/?a=<script>`

13. You can see your new custom blocking page
14. Extra lab if you have time - modify this page in Gitlab and run a new docker. The policy is modified accordingly without modifying the `./policy-adv/policy_base.json` file.

Create an OWASP Top 10 policy for NAP

So far, we created basic and custom policy (per location) and used external references. Now it is time to deploy an OWASP Top 10 policy. The policy not 100% OWASP Top 10 as several attacks can't be blocked just with a negative policy, we will cover a big part of OWASP Top 10.

Steps:

1. SSH to the Docker App Protect + Docker repo VM
2. In the `/home/ubuntu` directory, create a new folder `policy_owasp_top10`

```
mkdir policy_owasp_top10
```

3. Create a new policy file named `policy_owasp_top10.json` and paste the content below

```
vi ./policy_owasp_top10/policy_owasp_top10.json
```

```
{
"policy": {
    "name": "Complete_OWASP_Top_Ten",
    "description": "A generic, OWASP Top 10 protection items v1.0",
    "template": {
    "name": "POLICY_TEMPLATE_NGINX_BASE"
    },
    "enforcementMode":"blocking",
    "signature-settings":{
        "signatureStaging": false,
        "minimumAccuracyForAutoAddedSignatures": "high"
    },
    "caseInsensitive": true,
    "general": {
    "trustXff": true
    },
    "data-guard": {
    "enabled": true
    },
    "blocking-settings": {
    "violations": [
        {
        "alarm": true,
        "block": true,
        "description": "Modified NAP cookie",
        "name": "VIOL_ASM_COOKIE_MODIFIED"
        },
        {
        "alarm": true,
        "block": true,
        "description": "XML data does not comply with format settings",
```

(continues on next page)

(continued from previous page)

```
            "name": "VIOL_XML_FORMAT"
            },
            {
            "name": "VIOL_FILETYPE",
            "alarm": true,
            "block": true
            }
        ],
        "evasions": [
            {
            "description": "Bad unescape",
            "enabled": true
            },
            {
            "description": "Apache whitespace",
            "enabled": true
            },
            {
            "description": "Bare byte decoding",
            "enabled": true
            },
            {
            "description": "IIS Unicode codepoints",
            "enabled": true
            },
            {
            "description": "IIS backslashes",
            "enabled": true
            },
            {
            "description": "%u decoding",
            "enabled": true
            },
            {
            "description": "Multiple decoding",
            "enabled": true,
            "maxDecodingPasses": 3
            },
            {
            "description": "Directory traversals",
            "enabled": true
            }
        ]
        },
        "xml-profiles": [
        {
            "name": "Default",
            "defenseAttributes": {
            "allowDTDs": false,
            "allowExternalReferences": false
            }
        }
        ]
}
}
```

Note: Please have a quick look on this policy. You can notice several violations are enabled in order to cover the different OWASP categories

4. Now, create a new `nginx.conf` in the `policy_owasp_top10` folder. Do not overwrite the existing `/etc/nginx/nginx.conf` file, we need it for the next labs.

```
vi ./policy_owasp_top10/nginx.conf
```

```
user nginx;

worker_processes 1;
load_module modules/ngx_http_app_protect_module.so;

error_log /var/log/nginx/error.log debug;

events {
    worker_connections  1024;
}

http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;
    sendfile        on;
    keepalive_timeout  65;

    server {
        listen       80;
        server_name  localhost;
        proxy_http_version 1.1;

        app_protect_enable on;
        app_protect_security_log_enable on;
        app_protect_policy_file "/etc/nginx/policy/policy_owasp_top10.json
↪";
        app_protect_security_log "/etc/nginx/log-default.json"␣
↪syslog:server=10.1.20.6:5144;

        location / {
            resolver 10.1.1.9;
            resolver_timeout 5s;
            client_max_body_size 0;
            default_type text/html;
            proxy_pass http://k8s.arcadia-finance.io:30274$request_uri;
        }
    }
}
```

Note: You can notice we get back to a very simple policy. This is what DevOps and DevSecOps expect when they deploy NAP. Simple policy for OWASP Top10 attacks.

5. Last step is to run a new container (and delete the previous one) referring to these new files for OWASP Top 10 protection.

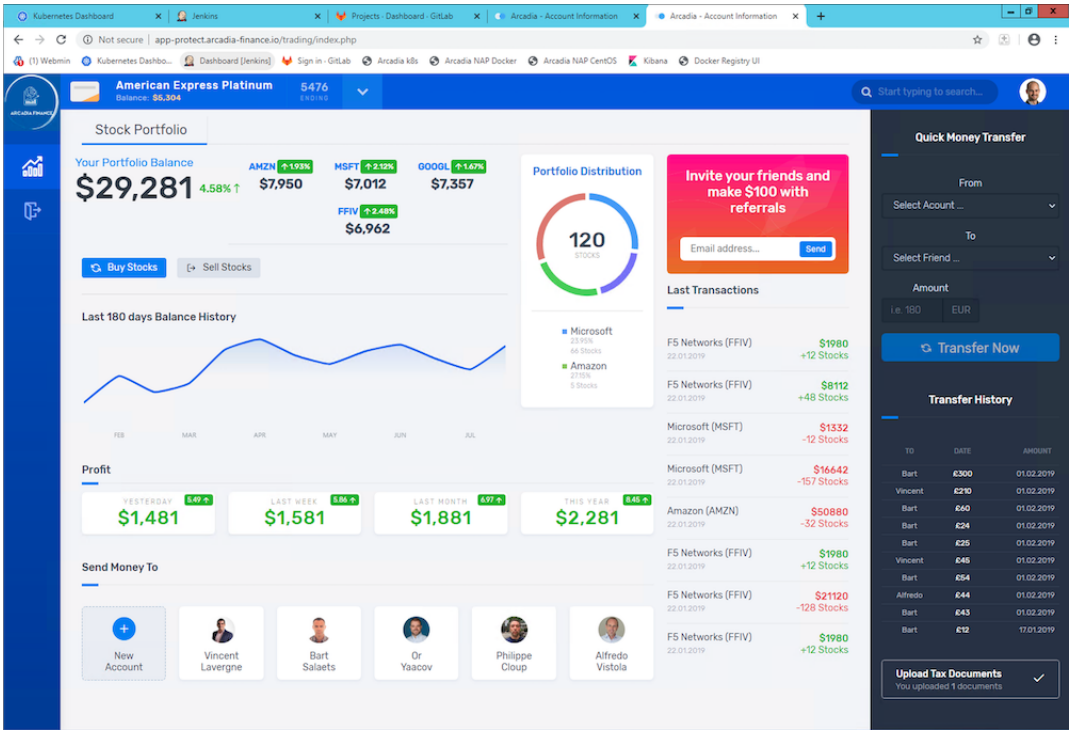
```
docker rm -f app-protect
docker run -dit --name app-protect -p 80:80 -v /home/ubuntu/policy_owasp_
↪top10/nginx.conf:/etc/nginx/nginx.conf -v /home/ubuntu/policy_owasp_
↪top10/policy_owasp_top10.json:/etc/nginx/policy/policy_owasp_top10.json␣
↪app-protect:20200316
```

6. Check that the `app-protect:20200316` container is running

```
docker ps
```

```
ubuntu@ip-10-1-1-5:~$ docker ps
CONTAINER ID        IMAGE                          COMMAND                  CREATED             STATUS              PORTS                    NAMES
ed07a846e470        app-protect:20200316           "sh /entrypoint.sh"      6 seconds ago       Up 5 seconds        0.0.0.0:80->80/tcp       app-protect
a89aef04940d        joxit/docker-registry-ui:static "/bin/sh -c entrypoi…" 4 months ago        Up 4 hours          0.0.0.0:8080->80/tcp     registry-ui
1e0dfd08c3e8        registry:2                     "/entrypoint.sh /etc…"   4 months ago        Up 4 hours          0.0.0.0:5000->5000/tcp   registry
ubuntu@ip-10-1-1-5:~$
```

7. RDP to the Jumhost as `user:user` and click on bookmark `Arcadia NAP Docker`



Video of this module (force HD 1080p in the video settings)

1.2.6 Step 8 - Deploy NAP with a CI/CD toolchain

In this module, we will deploy deploy NAP with a CI/CD pipeline. NAP is tied to the app, so when DevOps commits a new app (or a new version), the CI/CD pipeline has to deploy a new NAP component in front. In order to avoid repeating what we did previously, we will use a Signature package update as a trigger.

Note: When a new signature package is available, the CI/CD pipeline will build a new version of the Docker image and run it in front of Arcadia Application

---

This is the workflow we will run

1. Check if a new Signature Package is available
2. Simulate a Commit in GitLab (Goal is to simulate a full automated process checking Signature Package date every day)
3. This commit triggers a webhook in Gitlab CI
4. Gitlab CI runs the pipeline
    1. Build a new Docker NAP image with a new tag `date of the signature package`
    2. Destroy the previous running NAP container
    3. Run a new NAP container with this new Signature Package

---

Note: Goal of this module is not to learn how to do it, but understand how I did it.

Check the Gitlab CI file

```
stages:
    - Build_image
    - Push_image
    - Run_docker

before_script:
    - docker info

Build_image:
    stage: Build_image
    script:
        - TAG=`yum info app-protect-attack-signatures | grep Version | cut -d':' -f2`
        - echo $TAG
        - docker build -t 10.1.20.7:5000/app-protect:`echo $TAG` .
        - echo export TAG=`echo $TAG` > $CI_PROJECT_DIR/variables
    artifacts:
        paths:
        - variables

Push_image:
    stage: Push_image
    script:
        - source $CI_PROJECT_DIR/variables
        - echo $TAG
        - docker push 10.1.20.7:5000/app-protect:`echo $TAG`

Run_docker:
```

(continues on next page)

(continued from previous page)

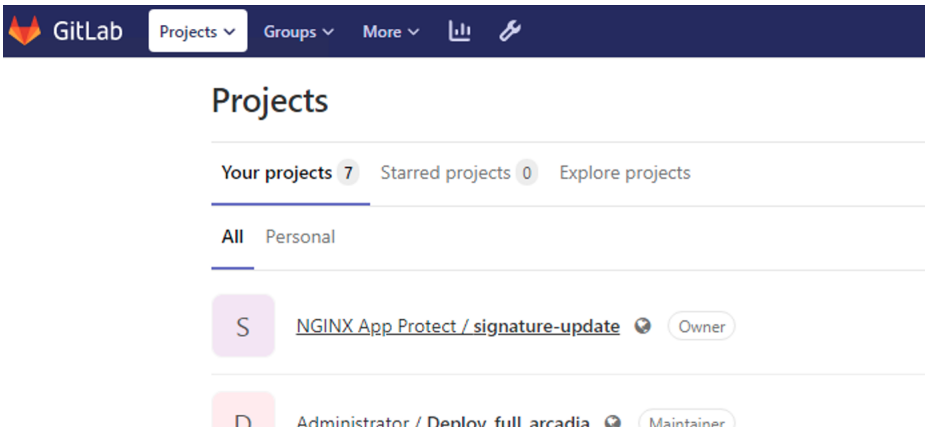
```
    stage: Run_docker
    script:
        - source $CI_PROJECT_DIR/variables
        - echo $TAG
        - ansible-playbook -i hosts playbook.yaml --extra-var dockertag=`echo $TAG`
```

Note: The challenge here was to retrieve the date of the package and tag the image with this date in order to have one image per signature package date. This is useful if you need to roll back to a previous version of the signatures.

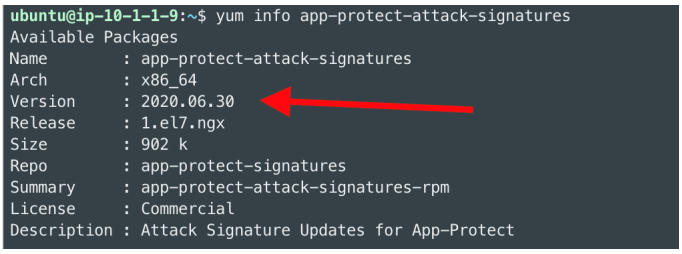
Simulate an automated task detecting a new Signature Package has been release by F5

Steps:

1. RDP to the Jumphost and open `Chrome`
2. Open `Gitlab`
    1. If Gitlab is not available (502 error), restart the GitLab Docker container. SSH to the GitLab VM and run `docker restart gitlab`
3. In GitLab, open `NGINX App Protect / signature-update` project



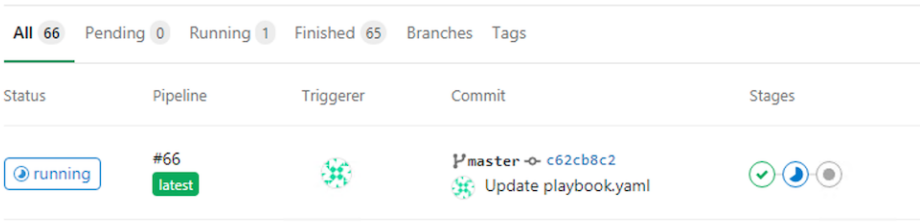
4. SSH (or WebSSH) to `CICD server (Gitlab runner, Terraform, Ansible)`
    1. Run this command in order to determine the latest Signature Package date: `yum info app-protect-attack-signatures`
    2. You may notice the version date. In my case, when I write this lab `2020.06.30` was the most recent version of the signatures package. We will use this date as a Docker tag, but this will be done automatically by the CI/CD pipeline.



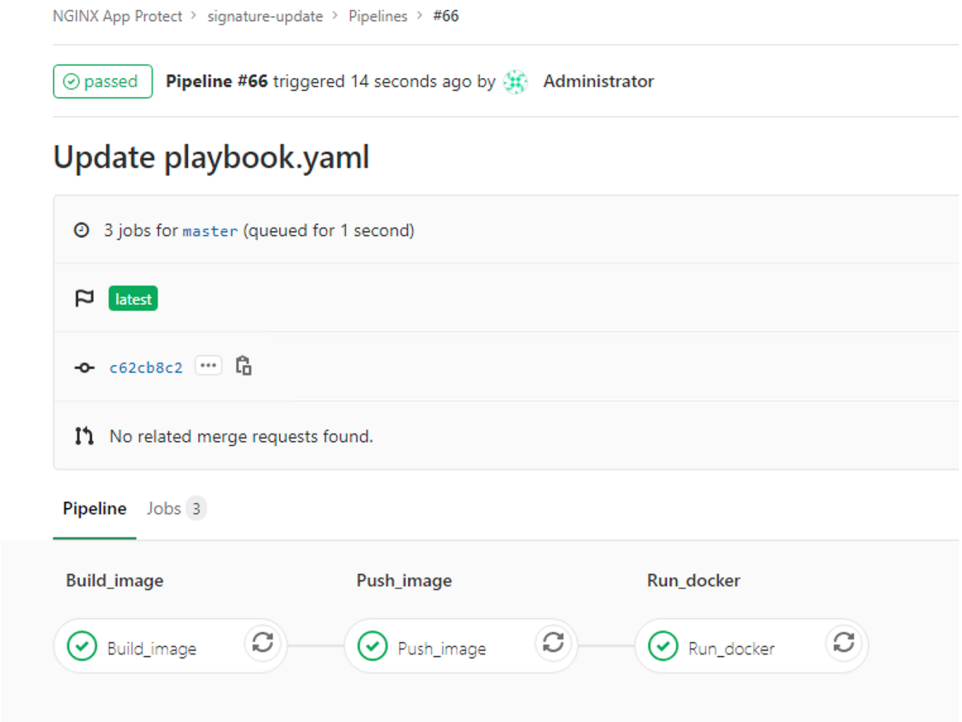
Trigger the CI/CD pipeline

Steps :

1. In GitLab, click on `Repository` and `Tags` in the left menu
2. Create a new tag and give it a name like `Sig-<version date>` where ideally `<version_date>` should be replaced by the package version information found in the result of the `yum info` step above. But it does not matter, you can put anything you want in this tag.
3. Click `Create tag`
4. At this moment, the `Gitlab CI` pipeline starts
5. In Gitlab, in the `signature-update` repository, click `CI / CD` > `Pipelines`



6. Enter into the pipeline by clicking on the `running or passed` button. And wait for the pipeline to finish. You can click on every job/stage to check the steps



7. **Check if the new image created and pushed by the pipeline is available in the Docker Registry.**

    1. In `Chrome` open bookmark `Docker Registry UI`
    2. Click on `App Protect` Repository
    3. You can see your new image with the tag `2020.06.30` - or any other tag based on the latest package date.

8. Connect in SSH to the Docker App Protect + Docker repo VM, and check the signature package date running `docker exec -it app-protect more /var/log/nginx/error.log`

```
2020/07/06 09:32:05 [notice] 12#12: APP_PROTECT { "event": "configuration_
↪load_success", "software_version": "3.74.0", "attack_signatures_package":{
↪"revision_datetime":"2020-06-30T10:08:35Z","version":"2020.06.30"},
↪"completed_successfully":true,"threat_campaigns_package":{}}
```

Note: Congratulations, you ran a CI/CD pipeline with a GitLab CI.

1.3 Class 3 - Protect Arcadia with NGINX App Protect in Linux host

In this class, we will deploy NGINX App Protect in CentOS host by installing the RPM packages from the official NGINX Plus Repo.

Warning: In this lab, there are cert + keys to download the packages from official NGINX repo. It is forbidden to share them with anyone.

1.3.1 Step 9 - Install the NGINX Plus and App Protect packages manually

In this module, we will manually install the NGINX Plus and NGINX App Protect modules in CentOS from the official repository.

Warning: NGINX Plus private key and cert are already installed on the CentOS. Don't share them.

Steps:

1. SSH to the App Protect in CentOS VM
2. Add NGINX Plus repository by downloading the file `nginx-plus-7.repo` to `/etc/yum.repos.d`:

```
sudo wget -P /etc/yum.repos.d https://cs.nginx.com/static/files/nginx-
↪plus-7.repo
```

3. Install the most recent version of the NGINX Plus App Protect package (which includes NGINX Plus):

```
sudo yum install -y app-protect
```

4. Check the NGINX binary version to ensure that you have NGINX Plus installed correctly:

```
sudo nginx -v
```

5. Configure the `nginx.conf` file. Rename the existing `nginx.conf` to `nginx.conf.old` and create a new one.

```
cd /etc/nginx/
sudo mv nginx.conf nginx.conf.old
sudo vi nginx.conf
```

Paste the below configuration into `nginx.conf` and save it

```
user  nginx;
worker_processes  auto;

error_log  /var/log/nginx/error.log notice;
pid        /var/run/nginx.pid;

load_module modules/ngx_http_app_protect_module.so;

events {
    worker_connections 1024;
}

http {
    include          /etc/nginx/mime.types;
    default_type  application/octet-stream;
    sendfile        on;
    keepalive_timeout  65;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request
↪" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    server {
    listen       80;
        server_name  localhost;
        proxy_http_version 1.1;

        app_protect_enable on;
        app_protect_policy_file "/etc/nginx/NginxDefaultPolicy.json";
        app_protect_security_log_enable on;
        app_protect_security_log "/etc/nginx/log-default.json"␣
↪syslog:server=10.1.20.6:5144;

        location / {
            resolver 10.1.1.9;
            resolver_timeout 5s;
            client_max_body_size 0;
            default_type text/html;
            proxy_pass http://k8s.arcadia-finance.io:30274$request_uri;
        }
    }
}
```

6. Create a log configuration file `log_default.json` (still in `/etc/nginx/`)

```
sudo vi log-default.json
```

Paste the configuration below into `log-default.json` and save it

```
{
    "filter": {
        "request_type": "all"
    },
    "content": {
        "format": "default",
        "max_request_size": "any",
        "max_message_size": "5k"
    }
}
```

7. Temporarily make SELinux permissive globally (https://www.nginx.com/blog/using-nginx-plus-with-selinux).

```
sudo setenforce 0
```

8. Start the NGINX service:

```
sudo systemctl start nginx
```

9. Check everything is running

```
less /var/log/nginx/error.log
```

```
2020/05/22 09:13:20 [notice] 6195#6195: APP_PROTECT { "event":
↪"configuration_load_start", "configSetFile": "/opt/app_protect/config/
↪config_set.json" }
2020/05/22 09:13:20 [notice] 6195#6195: APP_PROTECT policy 'app_protect_
↪default_policy' from: /etc/nginx/NginxDefaultPolicy.json compiled␣
↪successfully
2020/05/22 09:13:20 [notice] 6195#6195: APP_PROTECT { "event":
↪"configuration_load_success", "software_version": "2.52.1", "attack_
↪signatures_package":{"revision_datetime":"2019-07-16T12:21:31Z"},
↪"completed_successfully":true}
2020/05/22 09:13:20 [notice] 6195#6195: using the "epoll" event method
2020/05/22 09:13:20 [notice] 6195#6195: nginx/1.17.9 (nginx-plus-r21)
2020/05/22 09:13:20 [notice] 6195#6195: built by gcc 4.8.5 20150623 (Red␣
↪Hat 4.8.5-39) (GCC)
2020/05/22 09:13:20 [notice] 6195#6195: OS: Linux 3.10.0-1127.8.2.el7.x86_
↪64
2020/05/22 09:13:20 [notice] 6195#6195: getrlimit(RLIMIT_NOFILE):␣
↪1024:4096
2020/05/22 09:13:20 [notice] 6203#6203: start worker processes
2020/05/22 09:13:20 [notice] 6203#6203: start worker process 6205
2020/05/22 09:13:26 [notice] 6205#6205: APP_PROTECT { "event": "waf_
↪connected", "enforcer_thread_id": 0, "worker_pid": 6205, "mode":
↪"operational", "mode_changed": false}
```

Note: Congrats, now your CentOS instance is protecting the Arcadia application

Note: You may notice we used exactly the same `log-default.json` and `nginx.conf` files as in the Docker

lab.

Now, try in the Jumphost

Steps:

1. RDP to the Jumphost with credentials `user:user`
2. Open Chrome and click `Arcadia NAP CentOS`
3. Run the same tests as the Docker lab and check the logs in Kibana

Next step is to install the latest Signature Package

Steps:

1. To add NGINX Plus App Protect signatures repository, download the file app-protect-signatures-7.repo to /etc/yum.repos.d:

```
sudo wget -P /etc/yum.repos.d https://cs.nginx.com/static/files/app-
↪protect-signatures-7.repo
```

2. Update attack signatures:

```
sudo yum install -y app-protect-attack-signatures
```

   To install a specific version, list the available versions:

```
sudo yum --showduplicates list app-protect-attack-signatures
```

   To upgrade to a specific version:

```
sudo yum install -y app-protect-attack-signatures-2020.04.30
```

   To downgrade to a specific version:

```
sudo yum downgrade app-protect-attack-signatures-2019.07.16
```

3. Reload NGINX process to apply the new signatures:

```
sudo nginx -s reload
```

4. Check the **new** signatures package date:

```
less /var/log/nginx/error.log
```

Note: Upgrading App Protect does not install new Attack Signatures. You will get the same Attack Signature release after upgrading App Protect. If you want to also upgrade the Attack Signatures, you will have to explicitly update them by the respective command above.

Last step is to install the Threat Campaign package

Note: The App Protect installation does not come with a built-in Threat campaigns package like Attack Signatures. Threat campaigns Updates are released periodically whenever new campaigns and vectors are discovered, so you might want to update your Threat campaigns from time to time. You can upgrade the Threat campaigns by updating the package any time after installing App Protect. We recommend you upgrade to the latest Threat campaigns version right after installing App Protect.

Note: After having updated the Threat campaigns package you have to reload the configuration in order for the new version of the Threat campaigns to take effect. Until then App Protect will run with the old version, if exists. This is useful when creating an environment with a specific tested version of the Threat campaigns.

Steps :

1. As the repo has been already added, no need to add it. TC and Signatures use the same repo `https://cs.nginx.com/static/files/app-protect-signatures-7.repo`
2. Install the package

```
sudo yum install app-protect-threat-campaigns
```

3. Reload NGINX process to apply the new signatures:

```
sudo sudo nginx -s reload
```

4. Check the **new** Threat Campaign package date:

```
less /var/log/nginx/error.log
```

Note: We don't spend more time on Threat Campaign in this lab as we did it already in the Docker lab (Class 2 - Step 5)

Video of this module (force HD 1080p in the video settings)

1.3.2 Step 10 - Deploy App Protect via CI/CD pipeline

In this module, we will install NGINX Plus and App Protect packages on CentOS with a CI/CD toolchain. NGINX teams created Ansible modules to deploy it easily in a few seconds.

Note: The official Ansible NAP role is available here https://github.com/nginxinc/ansible-role-nginx-app-protect and the NGINX Plus role here https://github.com/nginxinc/ansible-role-nginx

Uninstall the previous running NAP

1. SSH to the App Protect in CentOS VM
2. Uninstall NAP in order to start from scratch

```
sudo yum remove -y app-protect*
```

```
Dependencies Resolved

================================================================================
 Package                                                        Arch
================================================================================
Removing:
 app-protect                                                    x86_64
 app-protect-compiler                                           x86_64
 app-protect-engine                                             x86_64
 app-protect-plugin                                             x86_64
Removing for dependencies:
 nginx-plus-module-appprotect                                   x86_64

Transaction Summary
================================================================================
Remove  4 Packages (+1 Dependent package)
```

3. Uninstall NGINX Plus packages

```
sudo yum remove -y nginx-plus*
```

```
Dependencies Resolved

================================================================================
 Package                                                        Arch
================================================================================
Removing:
 nginx-plus                                                     x86_64

Transaction Summary
================================================================================
Remove  1 Package
```

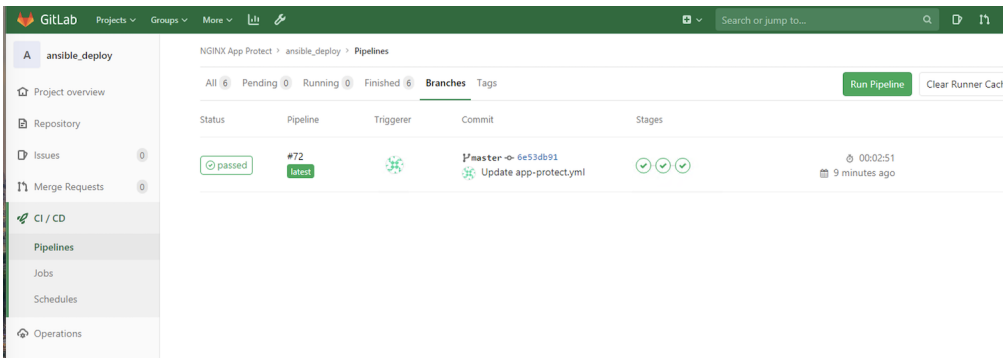
4. Delete/rename the directories from the existing deployment

```
sudo rm -rf /etc/nginx
sudo rm -rf /var/log/nginx
```

Run the CI/CD pipeline from Jenkins

Steps:

1. RDP to the Jumphost with credentials `user:user`
2. Open `Chrome` and open `Gitlab` (if not already opened)
3. Select the repository `ansible-deploy` and go to `CI /CD`



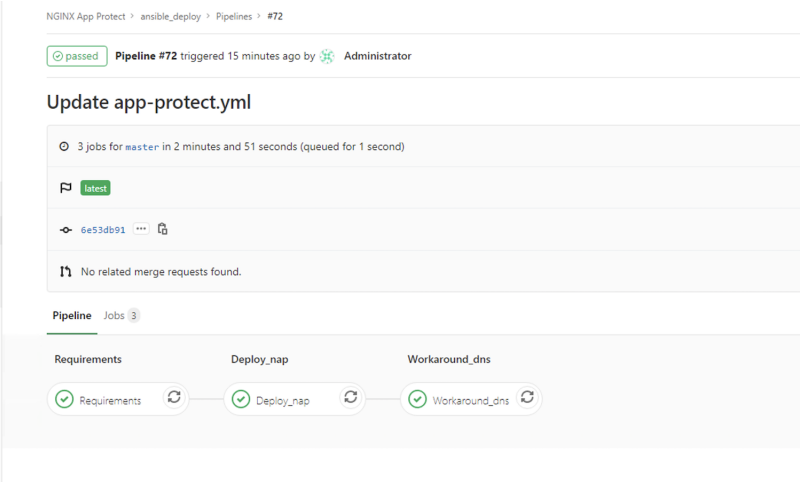
The pipeline is as below:

```
stages:
    - Requirements
    - Deploy_nap
    - Workaround_dns

Requirements:
    stage: Requirements
    script:
        - ansible-galaxy install -r requirements.yml --force

Deploy_nap:
    stage: Deploy_nap
    script:
        - ansible-playbook -i hosts app-protect.yml

Workaround_dns:
    stage: Workaround_dns
    script:
        - ansible-playbook -i hosts copy-nginx-conf.yml
```

Note: As you can notice, the `Requirements` stage installs the `requirements`. We use the parameter `--force` in order to be sure we download and install the latest version of the module.

Note: This pipeline executes 2 Ansible playbooks.

1. One playbook to install NAP (Nginx Plus included)
2. The last playbook is just there to fix an issue in UDF for the DNS resolver



When the pipeline is finished executing, perform a browser test within `Chrome` using the `Arcadia NAP CentOS` bookmark

Note: Congrats, you deployed `NGINX Plus` and `NAP` with a CI/CD pipeline. You can check the pipelines in `GitLab` if you are interested to see what has been coded behind the scenes. But it is straight forward as the Ansible modules are provided by F5/NGINX.

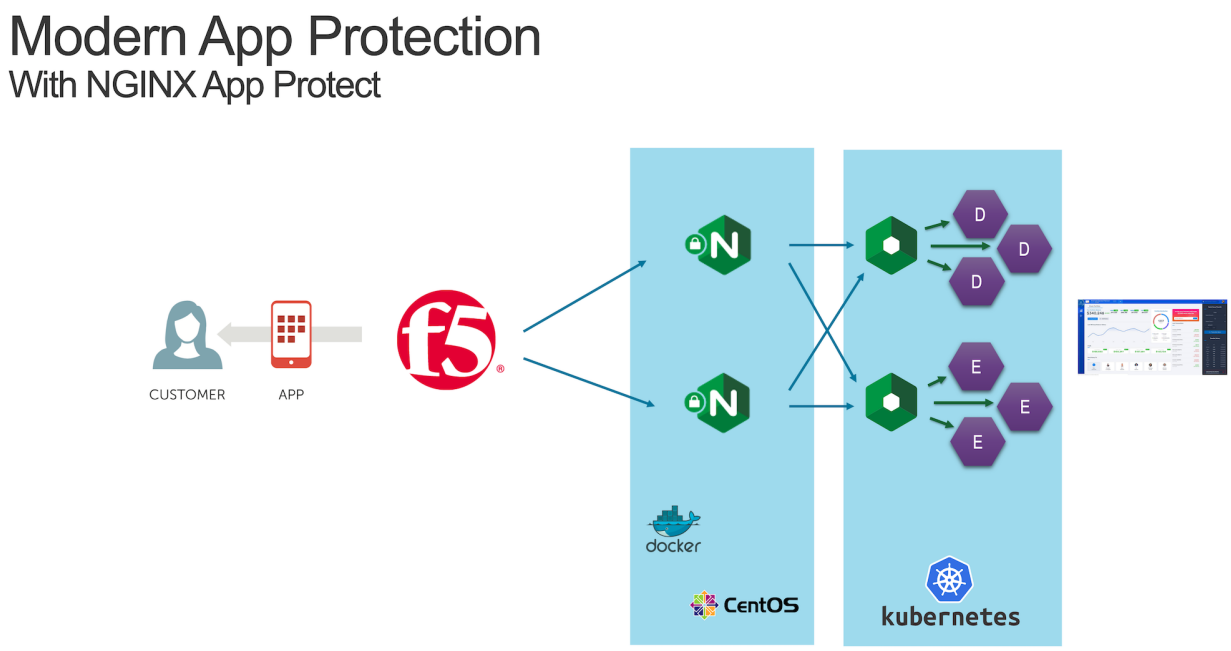
1.4 Class 4 - Protect Arcadia with NGINX App Protect in Kubernetes Ingress Controller

In this class, we will deploy NGINX App Protect in KIC. To do so, we will deploy an NGINX+ instance in Kubernetes with NAP installed.

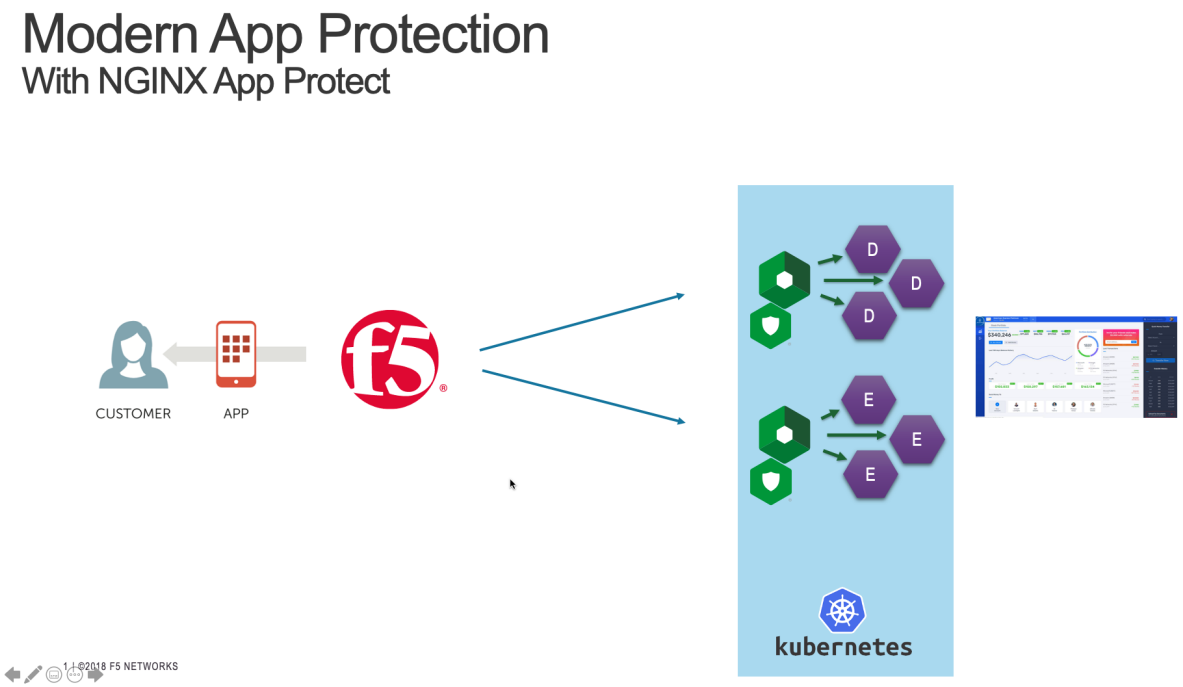
> **Warning:** In this lab, there are cert + keys to download the packages from official NGINX repo. It is forbidden to share them with anyone.

1.4.1 Step 11 - Deploy a new version of the NGINX Plus Ingress Controller

As a reminder, in `Class 1 - Step 2 - Publish Arcadia App with a NGINX Plus Ingress Controller` we deployed a NGINX Plus instance as an Ingress Controller in our Kubernetes cluster.



Now, with NAP v1.3, we can deploy this NGINX Plus instance with the NAP module enabled.



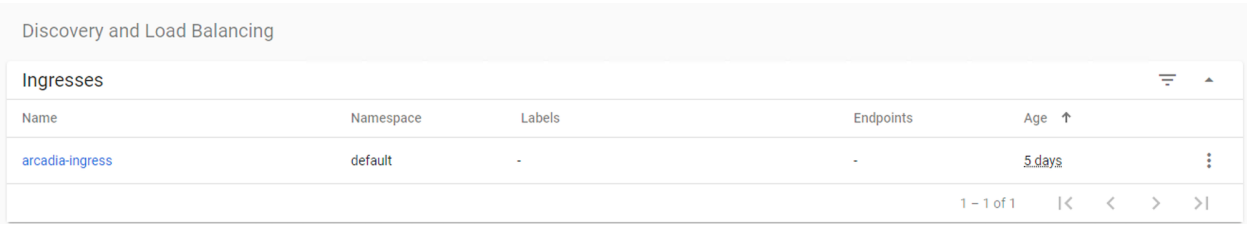
To do so, we will:

1. Deploy a new version of the Pod (NGINX r22 + NAP v1.3)
2. Deploy a new Ingress configuration template (with NAP configuration files)

> **Warning:** The NGINX Plus Ingress Controller image is available on my private Gitlab repo. Don't share the key.

Steps

1. SSH (or WebSSH and `cd /home/ubuntu/`) to CICD Server
2. Run this command in order to delete the previous KIC `kubectl delete -f /home/ubuntu/k8s_ingress/full_ingress_arcadia.yaml`
3. Run this command in order to push the new version of the KIC `kubectl apply -f /home/ubuntu/k8s_ingress/full_ingress_arcadia_nap.yaml`
4. Check the Ingress `arcadia-ingress` (in the `default` namespace) by clicking on the 3 dots on the right and `edit`
5. Scroll down and check the specs



As you can notice, we added few lines in our Ingress declaration. To do so, I followed the guide (https://docs.nginx.com/nginx-ingress-controller/app-protect/installation/)

1. I added NAP specifications (from the guide)

2. I added NAP annotations for Arcadia app (see below)

```
---
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
name: arcadia-ingress
annotations:
    appprotect.f5.com/app-protect-policy: "default/dataguard-blocking"
    appprotect.f5.com/app-protect-enable: "True"
    appprotect.f5.com/app-protect-security-log-enable: "True"
    appprotect.f5.com/app-protect-security-log: "default/logconf"
    appprotect.f5.com/app-protect-security-log-destination: "syslog:server=10.1.20.
↪6:5144"

spec:
rules:
- host: k8s.arcadia-finance.io
    http:
    paths:
    - path: /
        backend:
        serviceName: main
        servicePort: 80
    - path: /files
        backend:
        serviceName: backend
        servicePort: 80
    - path: /api
        backend:
        serviceName: app2
        servicePort: 80
    - path: /app3
        backend:
        serviceName: app3
        servicePort: 80
```

Please a make a new test by clicking on `Arcadia k8s` Chrome bookmark.

1. Open `Chrome`
2. Click on `Arcadia k8s` bookmark
3. Now, you are connecting to Arcadia App from a new KIC with NAP enabled
4. Send an attack (like a XSS in the address bar) by appending `?a=<script>`
5. Attack is blocked
6. Open ELK and check your logs

1.5 Class 5 - Advanced features

In this class, we will use the NAP in Centos and will deploy advanced features offered by the latest NAP release.

1. Bot Protection
2. Cryptonice integration
3. API Security with OpenAPI file import

Warning: In this lab, there are cert + keys to download the packages from official NGINX repo. It is forbidden to share them with anyone.

1.5.1 Step 12 - Bot Protection

Bot signatures provide basic bot protection by detecting bot signatures in the `User-Agent` header and `URI`. The bot-defense section in the policy is `enabled` by default.

Each bot signature belongs to a bot class. Search engine signatures such as `googlebot` are under the trusted_bots class, but App-Protect performs additional checks of the trusted bot's authenticity. If these checks fail, it means that the respective client impersonated the search engine in the signature and it will be classified as `class - malicous_bot`, `anomaly - Search engine verification failed`, and the request will be blocked, irrespective of the class's mitigation actions configuration.

An action can be configured for each bot class, or may also be configured per each bot signature individually:

1. `ignore` - bot signature is ignored (disabled)
2. `detected` - only report without raising the violation - VIOL_BOT_CLIENT. The request is considered legal unless another violation is triggered.
3. `alarm` - report, raise the violation, but pass the request. The request is marked as illegal.
4. `block` - report, raise the violation, and block the request

Note: We could stop the lab here, and run Bot requests. As Bot protection is `enabled` by `default`, the default protection will apply. But in order to understand to customise the config, let's create a new Policy JSON file.

Steps for the lab

1. SSH (or WebSSH) to `App Protect in CentOS`
2. Go to `cd /etc/nginx`
3. `ls` and check the files created during the previous CI/CD pipeline job (steps 10)

```
[centos@ip-10-1-1-7 nginx]$ ls
app-protect-log-policy.json       conf.d          koi-utf  mime.types
↪NginxApiSecurityPolicy.json  nginx.conf.orig          NginxStrictPolicy.
↪json  uwsgi_params
app-protect-security-policy.json  fastcgi_params  koi-win  modules
↪nginx.conf                   NginxDefaultPolicy.json  scgi_params
↪      win-utf
```

4. Create a new NAP policy JSON file with Bot

Note: The default actions for classes are: `detect for trusted-bot`, `alarm for untrusted-bot`, and `block for malicious-bot`. In this example, we enabled bot defense and specified that we want to raise a violation for trusted-bot, and block for untrusted-bot.

```
sudo vi /etc/nginx/policy_bots.json
```

```
{
    "policy": {
        "name": "bot_defense_policy",
        "template": {
            "name": "POLICY_TEMPLATE_NGINX_BASE"
        },
        "applicationLanguage": "utf-8",
        "enforcementMode": "blocking",
        "bot-defense": {
            "settings": {
                "isEnabled": true
            },
            "mitigations": {
                "classes": [
                    {
                        "name": "trusted-bot",
                        "action": "alarm"
                    },
                    {
                        "name": "untrusted-bot",
                        "action": "block"
                    },
                    {
                        "name": "malicious-bot",
                        "action": "block"
                    }
                ]
            }
        }
    }
}
```

5. Modify the `nginx.conf` file is order to reference to this new policy json file. Just a new line to add.

```
sudo vi /etc/nginx/nginx.conf
```

```
user nginx;

worker_processes 1;
load_module modules/ngx_http_app_protect_module.so;

error_log /var/log/nginx/error.log debug;

events {
    worker_connections  1024;
}

http {
    include       /etc/nginx/mime.types;
```

(continues on next page)

(continued from previous page)

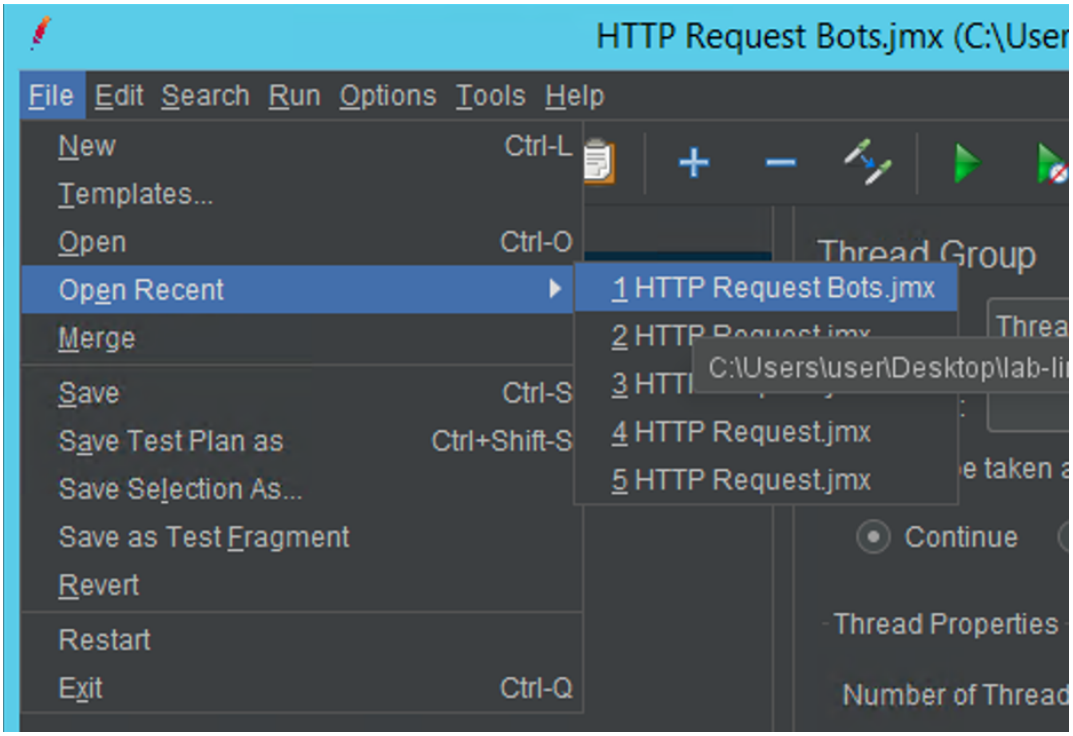
```
    default_type  application/octet-stream;
    sendfile        on;
    keepalive_timeout  65;

    server {
        listen       80;
        server_name  localhost;
        proxy_http_version 1.1;

        app_protect_enable on;
        app_protect_policy_file "/etc/nginx/policy_bots.json";
        app_protect_security_log_enable on;
        app_protect_security_log "/etc/nginx/app-protect-log-policy.json"␣
↪syslog:server=10.1.20.6:5144;

        location / {
            resolver 10.1.1.9;
            resolver_timeout 5s;
            client_max_body_size 0;
            default_type text/html;
            proxy_pass http://k8s.arcadia-finance.io:30274$request_uri;
        }
    }
}
```

6. Reload Nginx

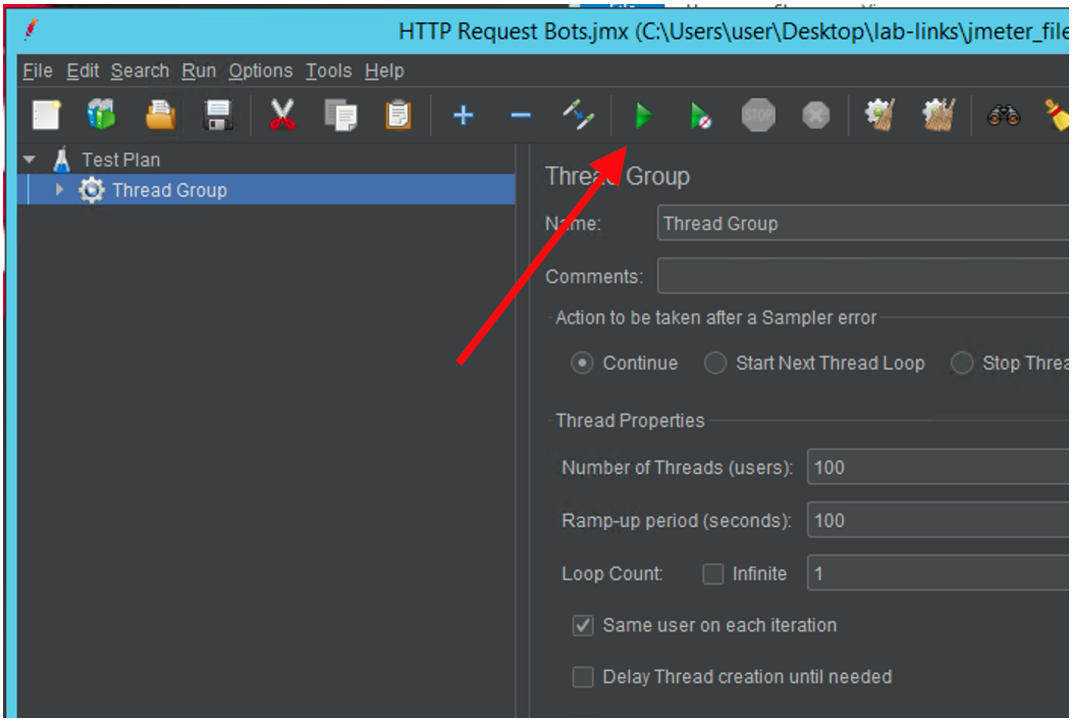
```
sudo nginx -s reload
```

Generate simulated Bot traffic

1. RDP to Windows `Jumphost` as `user`:`user`
2. Open `Chrome` and check your can acces Arcadia Web Application via the Bookmark `Arcadia NAP CentOS`
3. Now, on the `Desktop`, launch `Jmeter`
4. In Jmeter, open the project in `File` >> `Open Recent` >> `HTTP Request Bots.jmx`. This file is located in folder Desktop > lab-links > jmeter_files

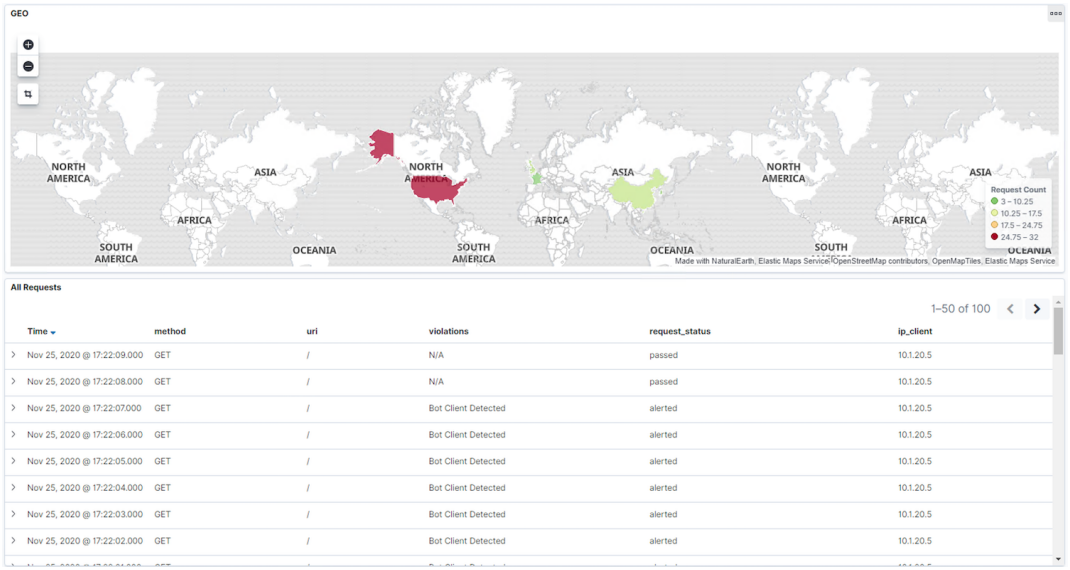


5. Now, run the project by click on the `GREEN PLAY BUTTON`



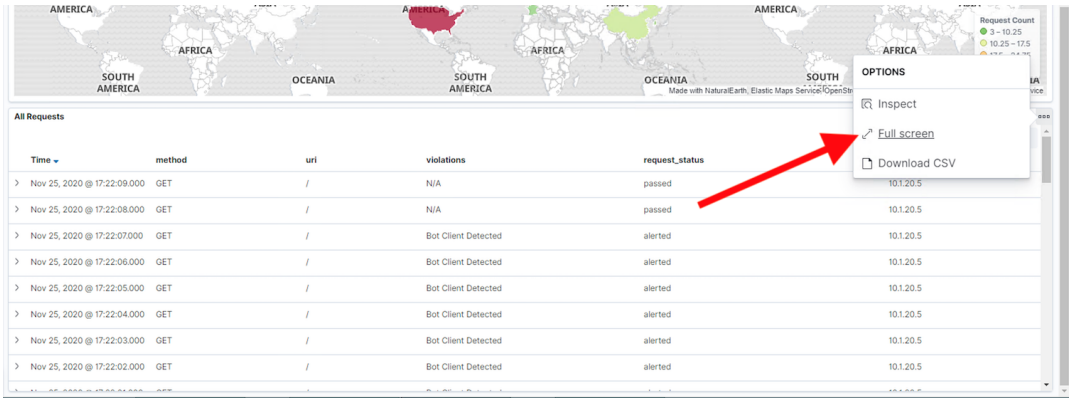
6. THe project is sending HTTP requests to the NAP with a public IP address (known as `bad reputation`) and with a Bot `User-Agent`. We will simulate bots by changing the user agent.
7. You can expand `Thread Group` and click on `View Results Tree` to see each request sent.
8. Now, go to `ELK - Kibana` from `Chrome`, and look at your `Overview` dashboard.

9. You can notice the widget `Bots Categories` is now populated.
10. You can notice Good and Bad request in the widgets, but let's focus on the logs at the bottom of the dashboard

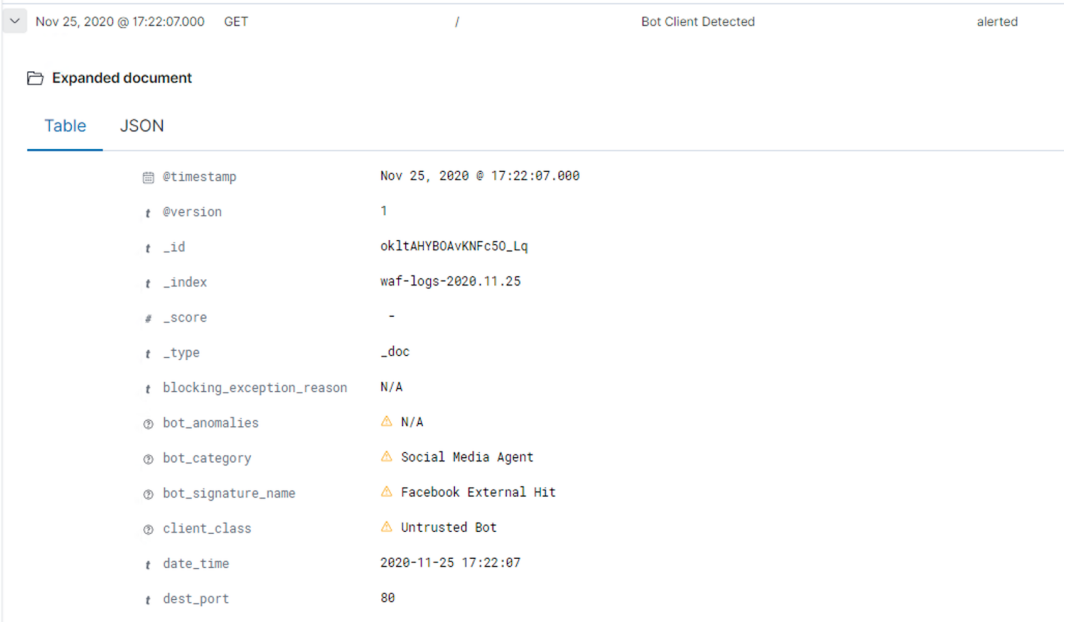


Note: You can notice we were able to `locate` the source of the request because jmeter inject an XFF header.

11. Open the logs in full screen



12. Look at the logs, and open up one or two logs `alerted` or `blocked`. You can notice the `Bot Category`, the `violation` ...



Note: Now, your NAP is protecting against `known bots` and you can customize your policy in order to make it more strick or not.

1.5.2 Step 13 - Cryptonice

What is cryptonice ?

Cryptonice is a command-line tool and Python library that allows a user to `examine` for one or more supplied domain names:

1. the TLS configuration
2. certificate information
3. web application headers
4. DNS records

Cryptonice is built heavily upon the excellent SSLyze and Cryptography Python libraries.

You can find more information on how to use Cryptonice here : https://cryptonice.readthedocs.io/en/latest/

```
RESULTS
-----------------------------------
Hostname:                         untrusted-root.badssl.com

Selected Cipher Suite:            ECDHE-RSA-AES128-GCM-SHA256
Selected TLS Version:             TLS_1_2

Supported protocols:
TLS 1.2:                          Yes
TLS 1.1:                          Yes
TLS 1.0:                          Yes


CERTIFICATE
Common Name:                      *.badssl.com
Public Key Algorithm:             RSA
Public Key Size:                  2048
Signature Algorithm:              sha256

Certificate is trusted:           False (Mozilla not trusted)
Hostname Validation:              OK - Certificate matches server hostname
Extended Validation:              False
Certificate is in date:           True
Days until expiry:                441
Valid From:                       2019-10-09 23:08:50
Valid Until:                      2021-10-08 23:08:50

OCSP Response:                    Unsuccessful
Must Staple Extension:            False

Subject Alternative Names:
          *.badssl.com
          badssl.com

Vulnerability Tests:
No vulnerability tests were run

HTTP to HTTPS redirect:           True
HTTP Strict Transport Security:   False
HTTP Public Key Pinning:          False

Secure Cookies:                   False

None

RECOMMENDATIONS
-----------------------------------
HIGH - TLSv1.0 Major browsers are disabling TLS 1.0 imminently. Carefully monitor if clients still use this protocol.
HIGH - 3DES The 3DES symmetric cipher is vulnerable to the Sweet32 attack
HIGH - TLSv1.1 Major browsers are disabling this TLS 1.1 immenently. Carefully monitor if clients still use this protocol.
Low - CAA Consider creating DNS CAA records to prevent accidental or malicious certificate issuance.

Scans complete
-----------------------------------
Total run time: 0:00:34.651568
```

Note: The goal here, with NAP, is to examine all websites published by our CI/CD pipeline.

Steps for the lab

Warning: As we have only one website published, **Arcadia Finance** website, we will run tests with real public websites.

To do so, Cryptonice will run as a Docker container, and we will run a command inside this container. The command is

```
docker exec -dit cryptonice-gitlab sh -c "cd /home && cryptonice {{ fqdn }}"
```

The variable `fqdn` will be replaced by the FQDN you will set in the pipeline. For the demo, you will set manually this variable, but in a real world, this variable is set by the pipeline itself.

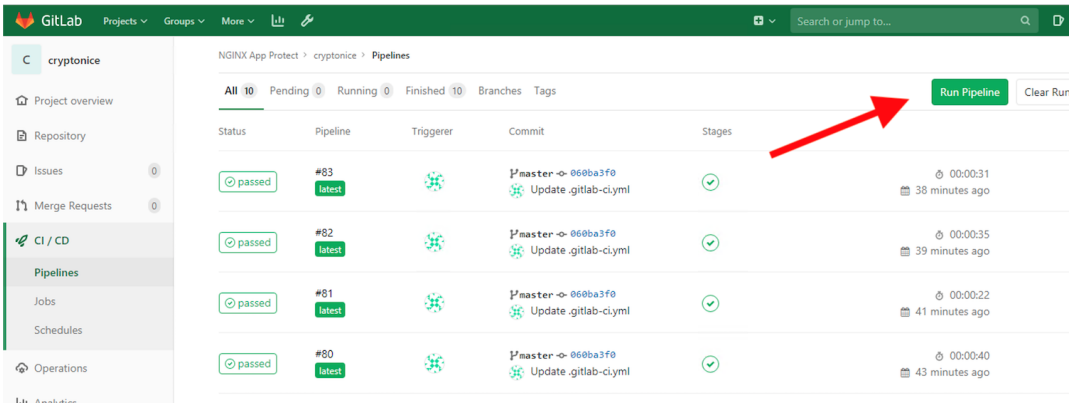
Steps

1. RDP to Windows Jumphost with credentials `user:user`
2. In `Chrome`, open `Gitlab` tab or bookmark, and click on `NGINX App Protect` > `Cryptonice` repository
3. If you want, you can check the Gitlab CI pipeline and the Ansible playbook. To make it simple, Gitlab CI pipeline runs the ansible playbook

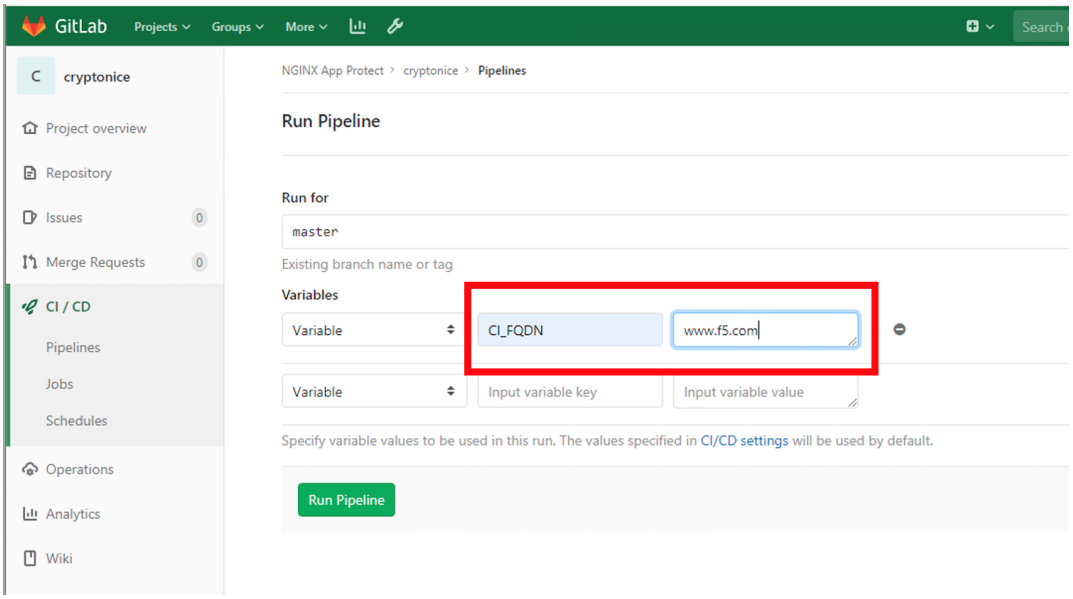
```
---
- name: copy content to ELK
  hosts: elk

  tasks:
  - name: delete all JSON in ELK
    shell: rm -f /home/gitlab-runner/crypto/*

- name: run Cryptonice
  hosts: localhost

  tasks:
  - name: Delete existing tests
    shell: rm -f /var/lib/gitlab-runner/crypto/*

  - name: Run cryptonice
    command: docker exec -dit cryptonice-gitlab sh -c "cd /home &&␣
↪cryptonice {{ fqdn }}"

  - name: WAIT
    wait_for:
    path: /var/lib/gitlab-runner/crypto/{{ fqdn }}.json

  - name: rename file
    shell: mv /var/lib/gitlab-runner/crypto/{{ fqdn }}.json /var/lib/
↪gitlab-runner/crypto/{{ fqdn }}.bck

  - name: add EOL
    shell: awk '{printf "%s\r\n\r\n", $0}' /var/lib/gitlab-runner/crypto/{
↪{ fqdn }}.bck > /var/lib/gitlab-runner/crypto/{{ fqdn }}.json

- name: copy content to ELK
  hosts: elk

  tasks:
  - name: copy JSON to ELK
    copy:
    src: /var/lib/gitlab-runner/crypto/{{ fqdn }}.json
    dest: /home/gitlab-runner/crypto/{{ fqdn }}.json
```

Note: As you can notice, running the command is not enough, we had to cleanup the environment and do some tricks so that ELK can read the outcomes. YES, all the outcomes will be readable in an ELK dashboard.

4. In the left menu, click on `CI / CD` and `Pipelines`
5. Click `Run Pipeline`



6. Define the variable `CI_FQDN` with any FQDN you want to test. Some websites like www.f5.com, or Facebook will raise some recommandations.

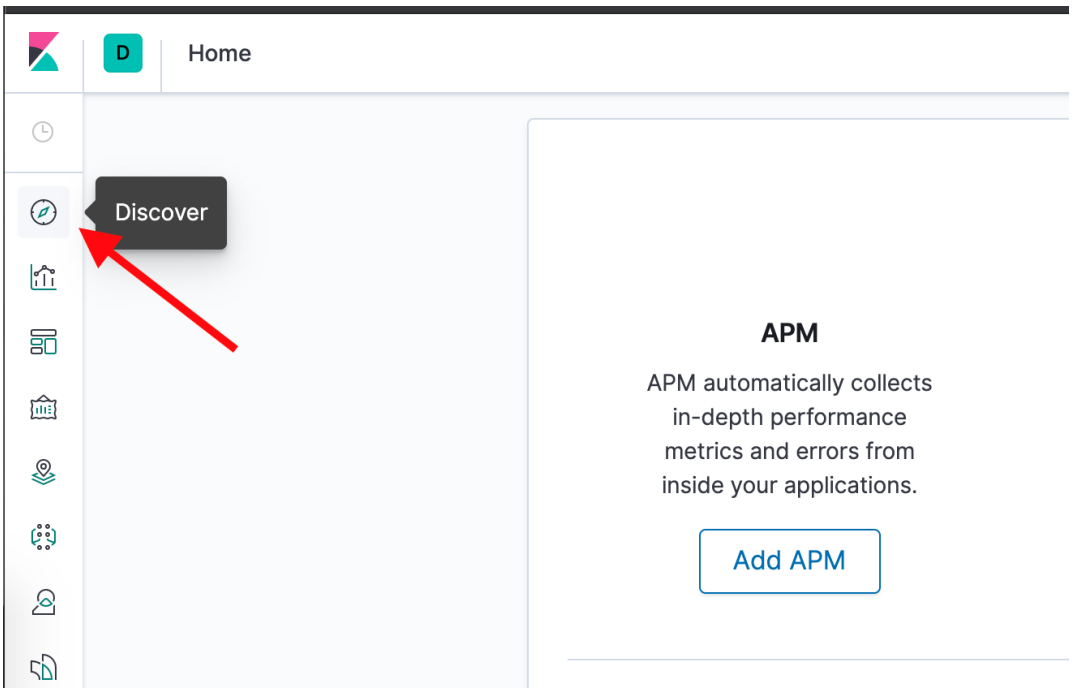


7. Click `Run Pipeline` and Wait :)

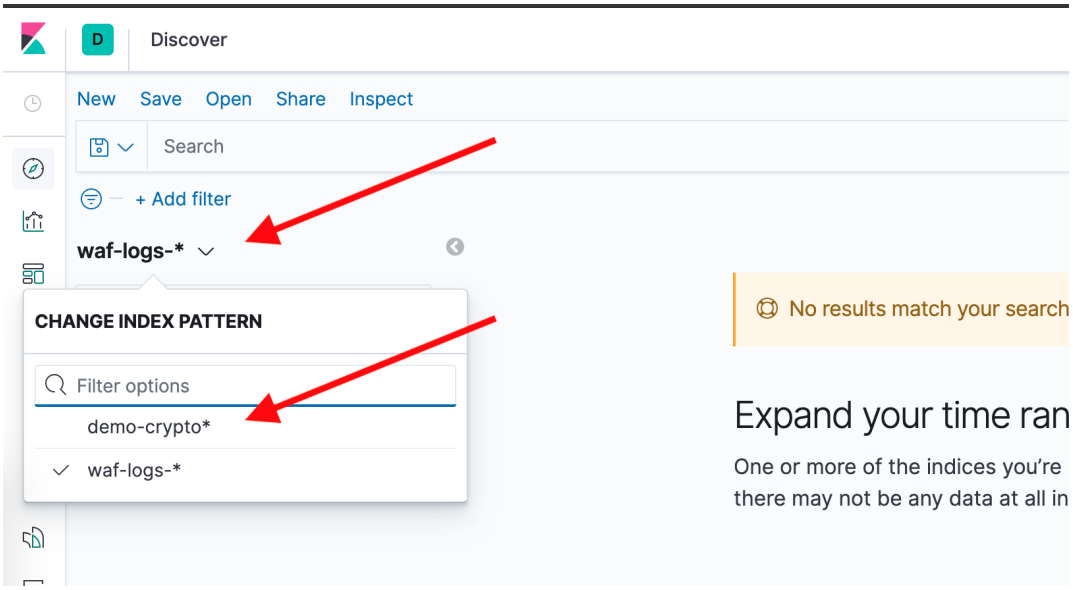
The outcomes

Now, it is time to see the results and what we can do with the information provided by `Cryptonice`

1. You should still be connected to the Jumphost RDP
2. In `Chrome`, open `Kibana` or use the Remote Access `ELK` in UDF if you prefer to connect from your laptop.
3. In `ELK` left menu, click on `Discover`

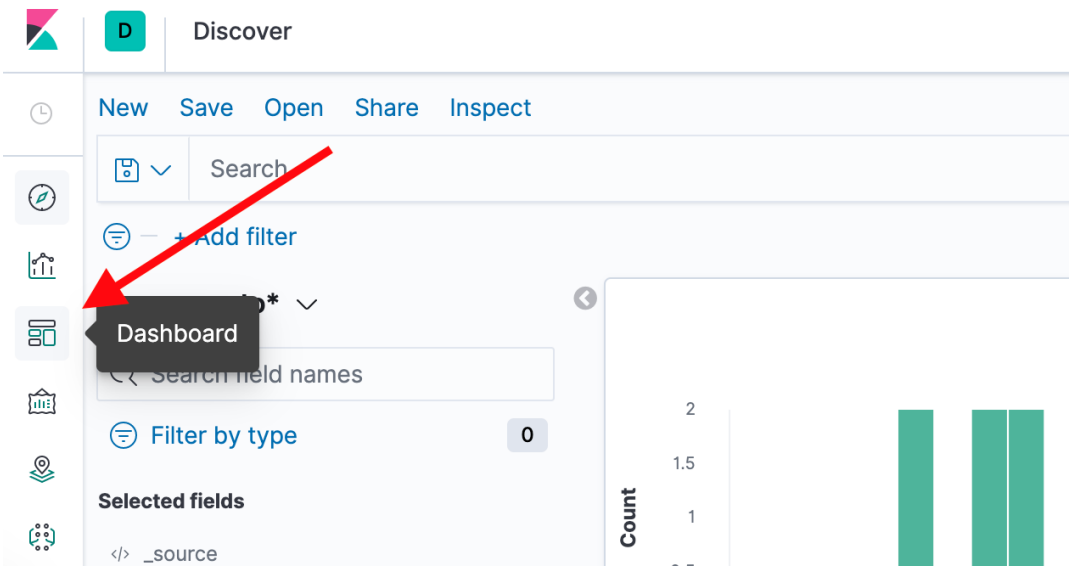


4. Then select `Demo-crypto*`

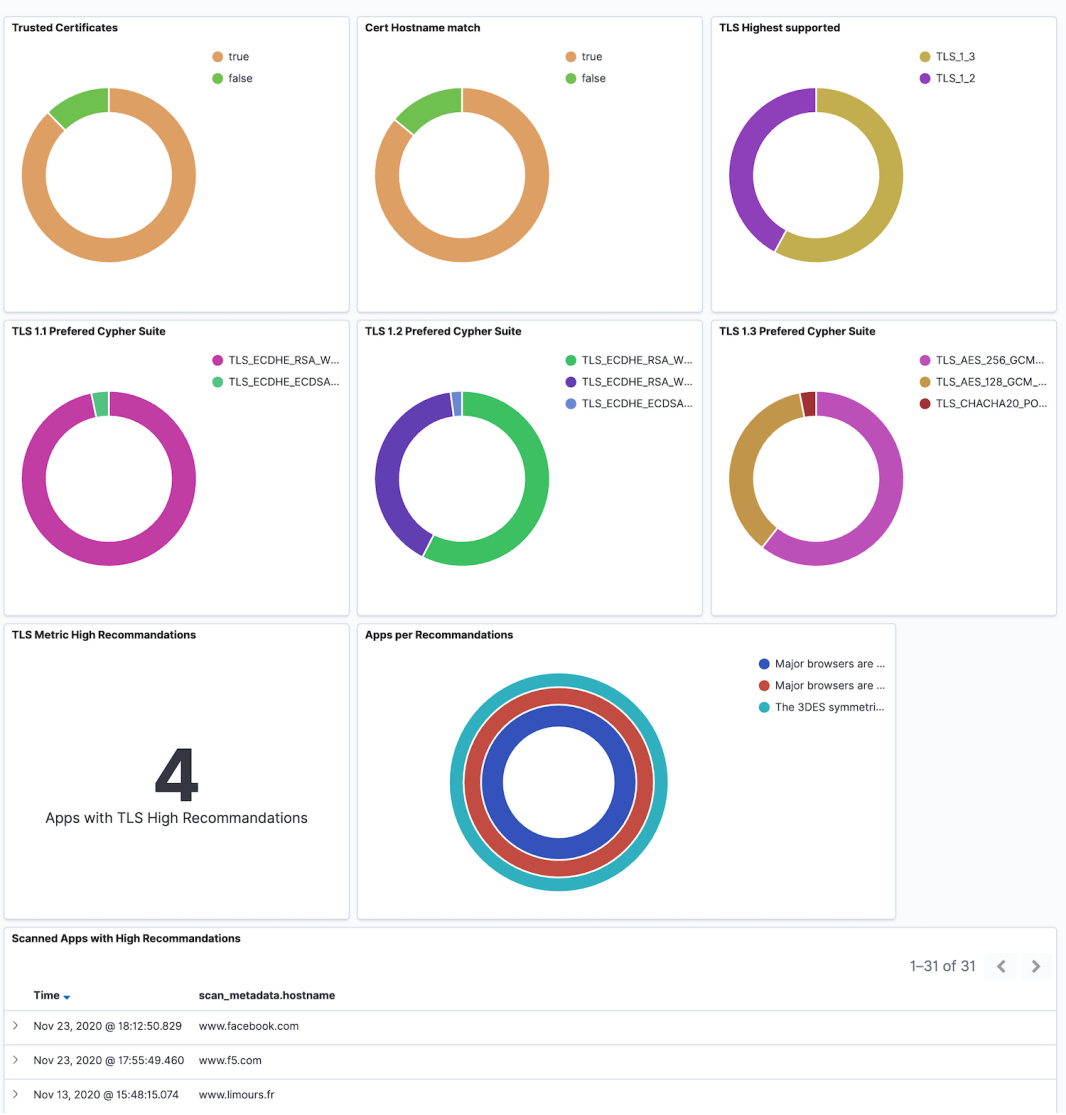


5. You should now see some logs. If not, change the time range on the top right corner. You can open a log and look at the content.

6. Now, go to the `Dashbboards` and click on `Cryptonice` dashboard



7. Change the time range to `Last 1 year` so that you can see all tests done so far (I did some for you)
8. You can see now an example of a `Cryptonice` dashboard. Feel free to create your own.



Note: In this Dashboard, you can see several information collected by `Cryptonice`. If the report contents **High Recommandations**, the website appears at the bottom and the widget is updated accordingly.

Note: Goal is to provide an easy and automated way for SecOps and DevOps to see their level of Security for TLS/HTTP/DNS Layers.

1.5.3 Step 14 - Protect Arcadia API

Context

As a reminder, in `Steps 9 and 10`, we deployed NAP in CentOS.

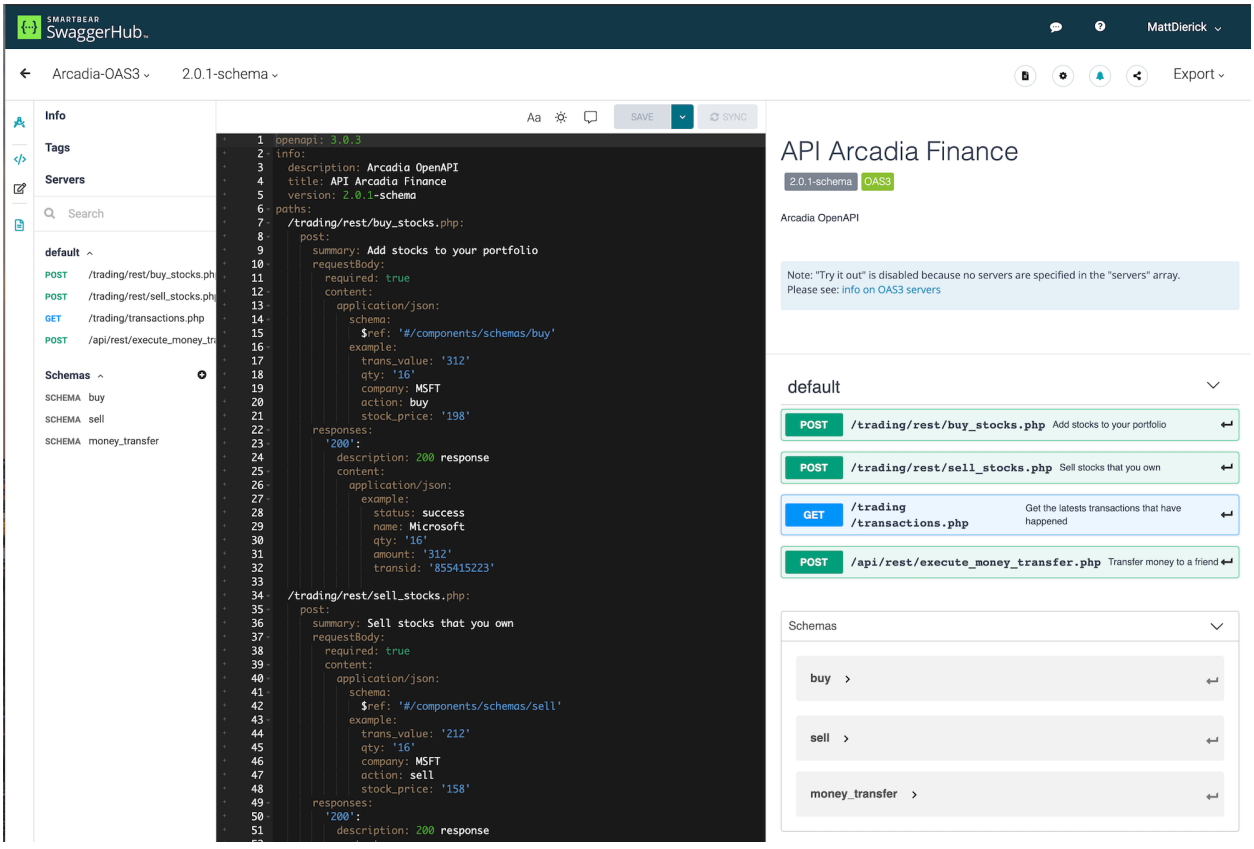
1. Step 9 manually
2. Step 10 via CI/CD pipelines

The Arcadia web application has several APIs in order to:

1. Buy stocks
2. Sell stocks
3. Transfer money to friends

In order to protect these APIs, we will push (or pull) an `OpenAPI specification file` into NAP so that it can build the WAF policy from this file.

You can find the `Arcadia Application OAS3` file here : https://app.swaggerhub.com/apis/F5EMEASSA/Arcadia-OAS3/2.0.1-schema



Note: As you can notice, there are 4 URLs in this API. And a JSON schema has been created so that every JSON parameter is known.

Steps for the lab

1. SSH (or WebSSH) to `App Protect in CentOS`
2. Go to `cd /etc/nginx`
3. `ls` and check the files created during the previous CI/CD pipeline job

```
[centos@ip-10-1-1-7 nginx]$ ls
app-protect-log-policy.json       conf.d          koi-utf  mime.types
↪NginxApiSecurityPolicy.json  nginx.conf.orig          NginxStrictPolicy.
↪json  uwsgi_params
app-protect-security-policy.json  fastcgi_params  koi-win  modules
↪nginx.conf                   NginxDefaultPolicy.json  scgi_params
↪      win-utf
```

Note: You can notice a NAP policy `NginxApiSecurityPolicy.json` exists. This is template for API Security. We will use it.

4. Edit `sudo vi NginxApiSecurityPolicy.json` and modify it with the `link` to the OAS file for Arcadia API. This file resides in SwaggerHub. Don't forget the {}

```
{
"policy" : {
    "name" : "app_protect_api_security_policy",
    "description" : "NGINX App Protect API Security Policy. The policy is
↪intended to be used with an OpenAPI file",
    "template": {
        "name": "POLICY_TEMPLATE_NGINX_BASE"
    },

    "open-api-files" : [
        {
            "link": "https://api.swaggerhub.com/apis/F5EMEASSA/Arcadia-
↪OAS3/2.0.1-schema/swagger.json"
        }
    ],

    "blocking-settings" : {
        "violations" : [
            {
...
```

5. Now, edit `sudo vi nginx.conf` and modify it as below. We refer to the new WAF policy created previously

```
user  nginx;
worker_processes  auto;

error_log  /var/log/nginx/error.log notice;
pid        /var/run/nginx.pid;

load_module modules/ngx_http_app_protect_module.so;

events {
    worker_connections 1024;
}
```

(continues on next page)

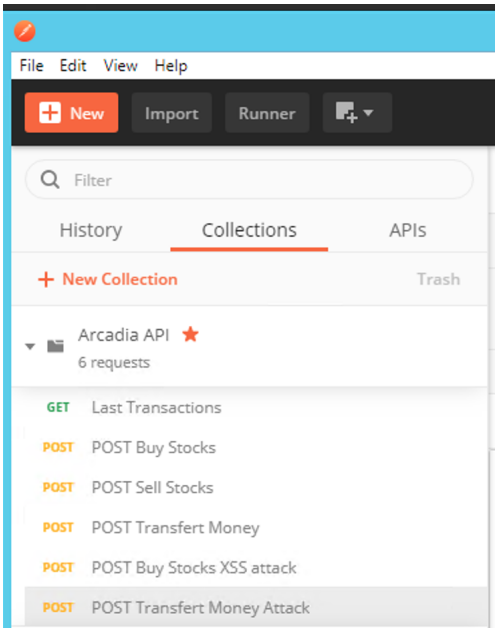
(continued from previous page)

```
http {
    include          /etc/nginx/mime.types;
    default_type  application/octet-stream;
    sendfile        on;
    keepalive_timeout  65;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request
↪" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    server {
        listen 80;
        server_name localhost;
        proxy_http_version 1.1;

        app_protect_enable on;
        app_protect_policy_file "/etc/nginx/NginxApiSecurityPolicy.json";
        app_protect_security_log_enable on;
        app_protect_security_log "/etc/nginx/log-default.json"␣
↪syslog:server=10.1.20.6:5144;

        location / {
            resolver 10.1.1.9;
            resolver_timeout 5s;
            client_max_body_size 0;
            default_type text/html;
            proxy_pass http://k8s.arcadia-finance.io:30274$request_uri;
        }
    }
}
```

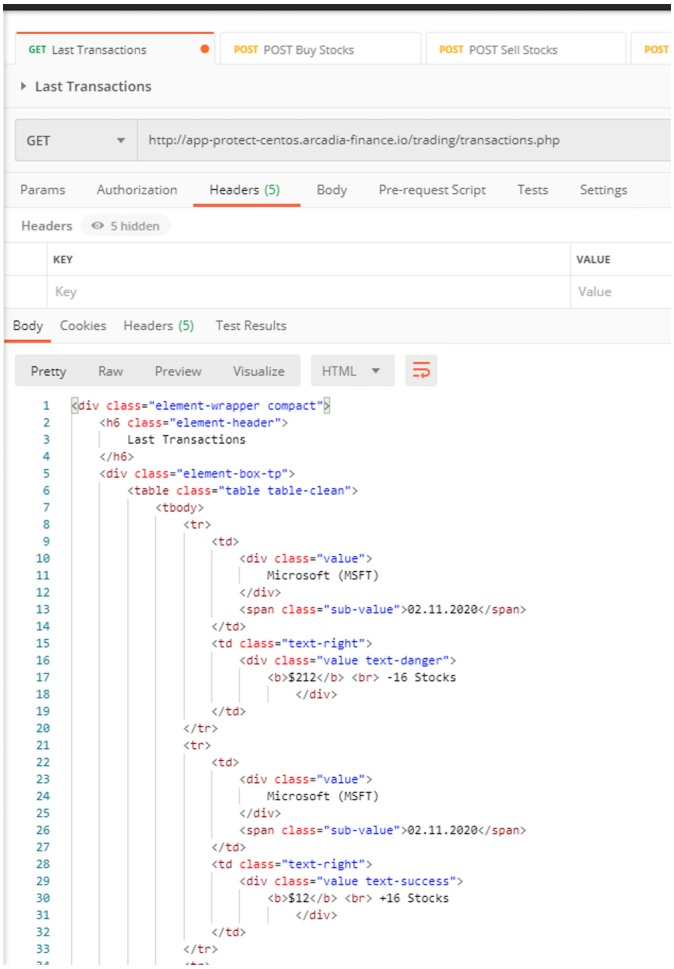
6. Now, restart the NGINX service `sudo systemctl restart nginx`

Test your API

1. RDP to Windows Jumphost with credentials `user:user`
2. Open `Postman`
3. Open Collection `Arcadia API`

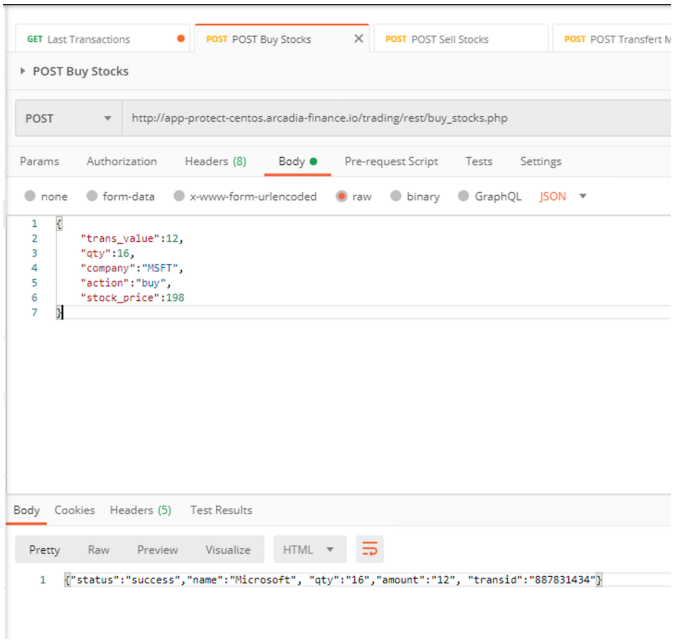


4. Send your first API Call with `Last Transactions`. You should see the last transactions. This is just a GET.

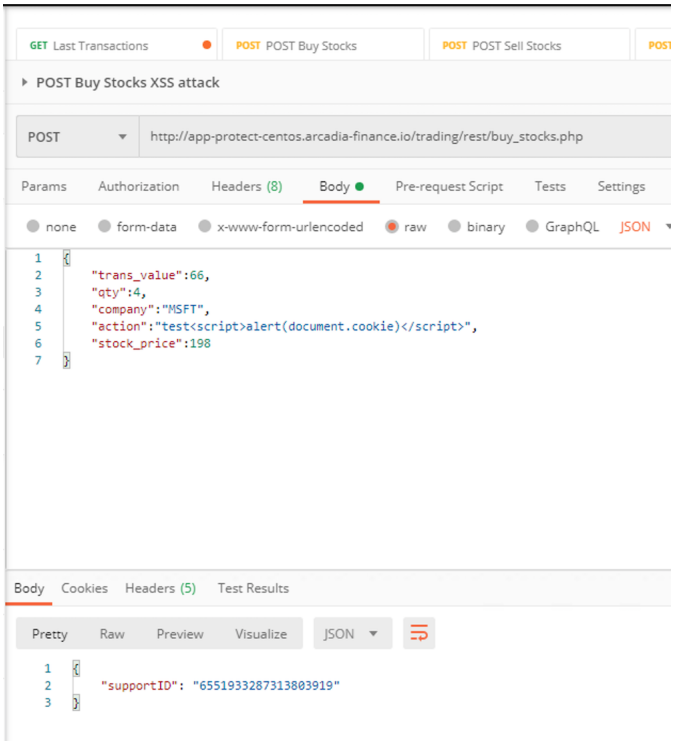


5. Now, send a POST, with `POST Buy Stocks`. Check the request content (headers, body), and compare with

the OAS3 file in SwaggerHub.



6. Last test, send an attack. Send `POST Buy Stocks XSS attack`. Your request will be blocked.



7. Check in ELK the violation.
8. You can make more tests with the other `API calls`