Practical DevSecOps

Exam Report

for

DevSecOps Professional (CDP)

By Rique Rio Orozco

Challenge 1: Create a CI/CD pipeline and embed security tools in it with the following steps (30 points)

In this challenge, you will use Gitlab CI to implement a CI/CD pipeline with the following details.

- Mature django.nv project to level 1 and level 2.
- Implement, SCA, Secret Scanning, SAST and DAST (baseline) in the pipeline. All the jobs shouldn't fail the builds and should save each tool's machine parseable output/result files on CI server for further processing.
- All the jobs must implement the applicable DevSecOps Gospel (best practices).
- Also, explain why you wouldn't want to fail the build even though security issues are found in the django.nv code.

Challenge 2: Create an Ansible Playbook to harden the prod server (20 points)

In this challenge, you will use Gitlab CI to implement a CI/CD pipeline with the following details.

- Create a job called os-hardening under the deploy stage.
- In this job, run ansible os hardening script from dev-sec (https://github.com/dev-sec/ansible-os-hardening) on the production machine, ensure this automation runs only on the master branch in the django.nv repository.
- The os-hardening job shouldn't fail the build but must still save the results on CI server for further processing in a machine-readable format like JSON, CSV, XML, etc.
- As always, test it locally on DevSecOps-box machine and then put it into the CI/CD pipeline.

Challenge 3: Scan for secrets in django.nv repository (25 points)

In this challenge, you will use Gitlab CI to implement a CI/CD pipeline with the following details.

● Create a job called secrets-scanning under the build stage.

● In this job, run trufflehog tool using docker. ● the secrets-scanning job must fail the build(do not just use exit 1 to achieve this).

● the secrets-scanning job should save the trufflehog results/output on the CI server for further processing in a machine-readable format like JSON, CSV, XML, etc.

● Get rid of false positives from the results so the next scans do not show the false positives.

● As always, test it locally on DevSecOps-box machine and then put it into the CI/CD pipeline.

Challenge 4: Run ZAP Scan against django.nv application and upload results to Defect Dojo (15 points)

In this challenge, you will run the zap baseline scan on django.nv application from the DevSecOps Box and put this task in CI pipeline.

- Run ZAP Scan on the django.nv app.
- Upload the results of the ZAP baseline scan into defect dojo's engagement id 1, using upload-results.py python script.

Challenge 5: Run inspec nginx-baseline profile on the production machine (10 points)

In this challenge, you will run nginx-baseline inspec profile on the production machine and will fix failed tests/errors on the production machine.

- Run nginx-baseline inspec profile on production machine from DevSecOps-Box machine.
- Try to fix the inspec control failures on the production machine. You can log in to the production machine using ssh prod-xxxx command.
- Explain how you manually fixed the control failures given by Inspec
- 10 Bonus points will be awarded if the fix is added to the ansible role nginx-baseline (of dev-sec).

Challenge 1 Solution:

First I will need to grab the django.nv project and cd into its folder

```
$ git clone https://gitlab.practical-devsecops.training/pdso/django.nv webapp
$ cd webapp
```

I need to install and run locally each of the security tools I will be using. It seems I don't have access to the docker hysnsec locally, so I will install each the hard way.

For SCA frontend I will use retire

```
$ sudo apt-get update && sudo apt-get upgrade
$ sudo apt-get install nodejs npm
$ npm install
$ npm install -g retire
```

For SCA backend I will use safety

```
$ pip3 install safety
```

For Secrets-Scanning I will use trufflehog

```
$ pip3 install trufflehog
```

For SAST I will use bandit

```
$ pip3 install bandit
```

For DAST I will use nikto, sslyze, nmap and zap-baseline

```
$ sudo apt-get install nikto
$ pip3 install sslyze
$ sudo apt-get install nmap
$ docker pull owasp/zap2docker-stable
```

each tool has a way to explore what options I can use with it via –help

with these options I can run each tool locally.

Retire

```
$ retire --outputformat json --outputpath retirejs-report.json --severity high --exitwith 0
```

Safety

```
$ safety check -r requirements.txt --json > oast-results.json
```

Trufflehog

```
$ trufflehog https://gitlab.practical-devsecops.training/pdso/django.nv | tee trufflehog-output.json
```

Bandit

```
$ bandit -r . -f json -o bandit-output.json
```

Nikto (also create the nikto.dtd file)

```
$ touch /usr/share/doc/nikto/nikto.dtd
```

```
$ nikto -output nikto_output.xml -h prod-yQJa8AqM
```

Sslyze

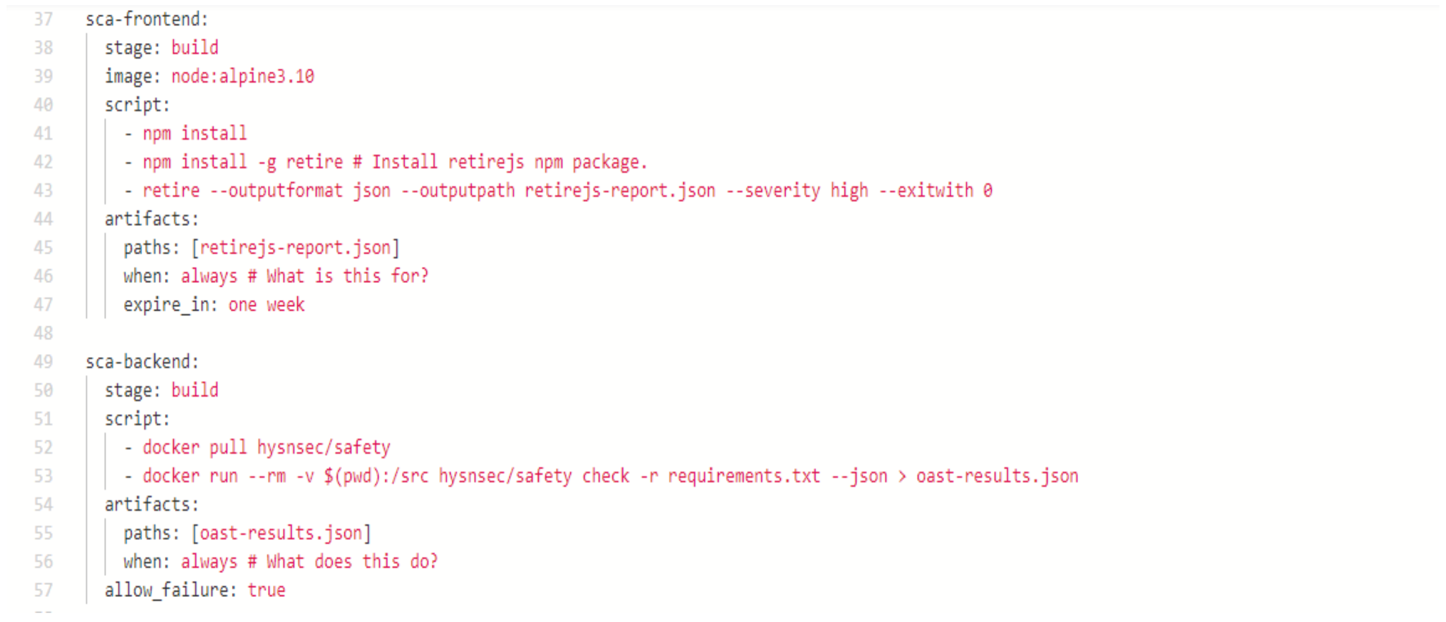
```
$ sslyze --regular --json_out sslyze-output.json prod-yQJa8AqM.lab.practical-devsecops.training:443
```

Zap-baseline

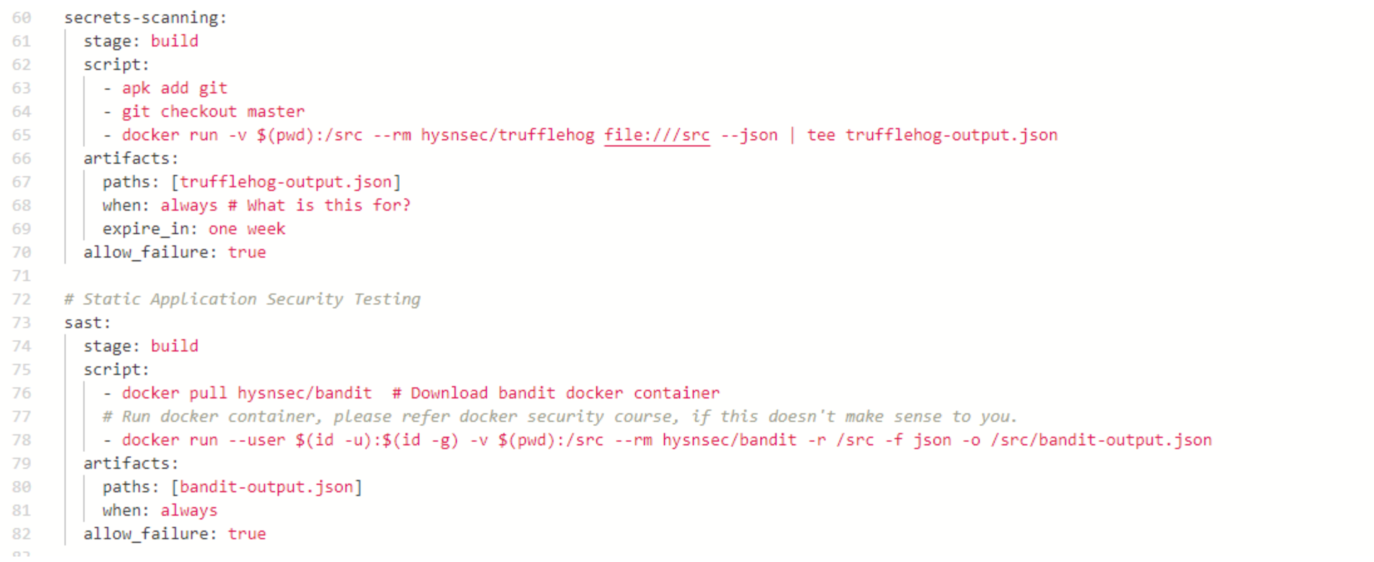
```
$ docker run --user $(id -u):$(id -g) --rm -v $(pwd):/zap/wrk:rw owasp/zap2docker-stable zap-baseline.py -t https://prod-yQJa8AqM.lab.practical-devsecops.training -J zap-output.json
```

every tool has been installed and ran successfully locally. Using all of these properly In our pipeline allows Django to reach maturity of level 2. Now to create the jobs for the .gitlab.yml for the CI/CD pipeline. For gitlab I will be using the shortcut installation with docker at hysnsec and running through docker

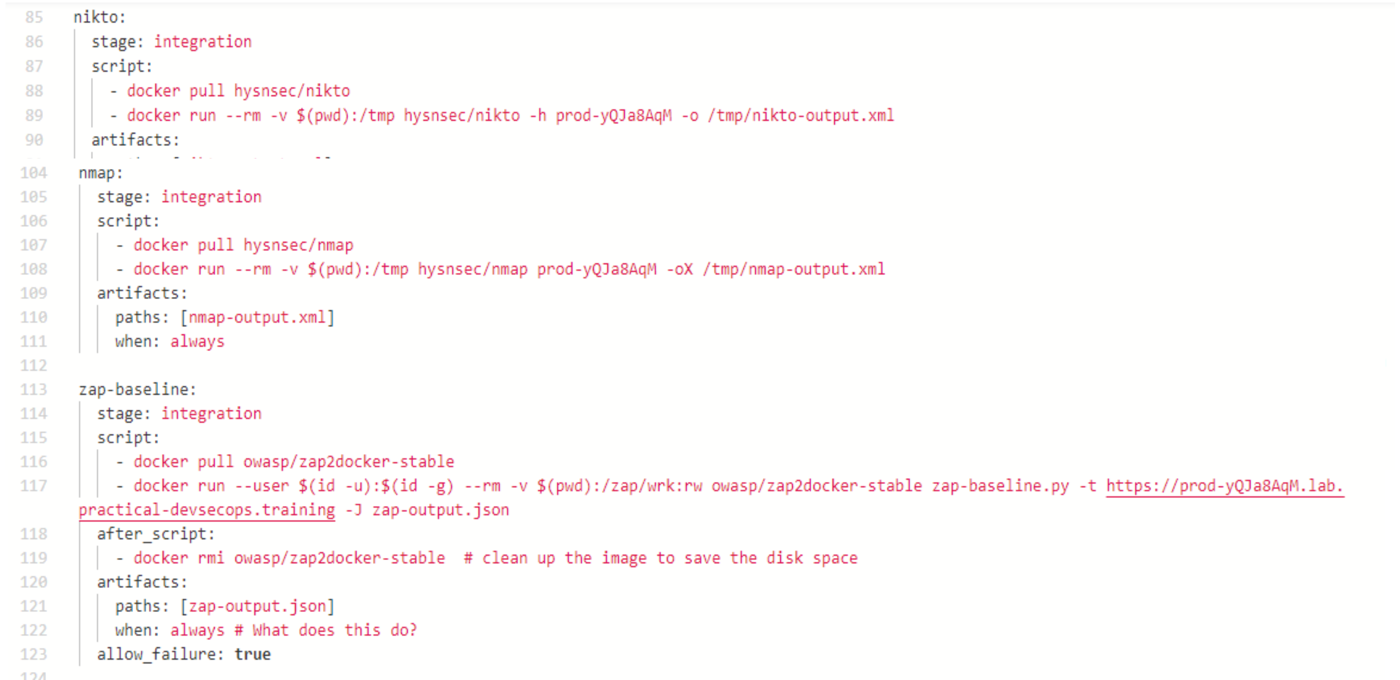
Please see gitlab.yml in challenge 1 folder for the full yml file. These are just the jobs for the above.



```yaml
sca-frontend:
  stage: build
  image: node:alpine3.10
  script:
    - npm install
    - npm install -g retire # Install retirejs npm package.
    - retire --outputformat json --outputpath retirejs-report.json --severity high --exitwith 0
  artifacts:
    paths: [retirejs-report.json]
    when: always # What is this for?
    expire_in: one week

sca-backend:
  stage: build
  script:
    - docker pull hysnsec/safety
    - docker run --rm -v $(pwd):/src hysnsec/safety check -r requirements.txt --json > oast-results.json
  artifacts:
    paths: [oast-results.json]
    when: always # What does this do?
  allow_failure: true
```

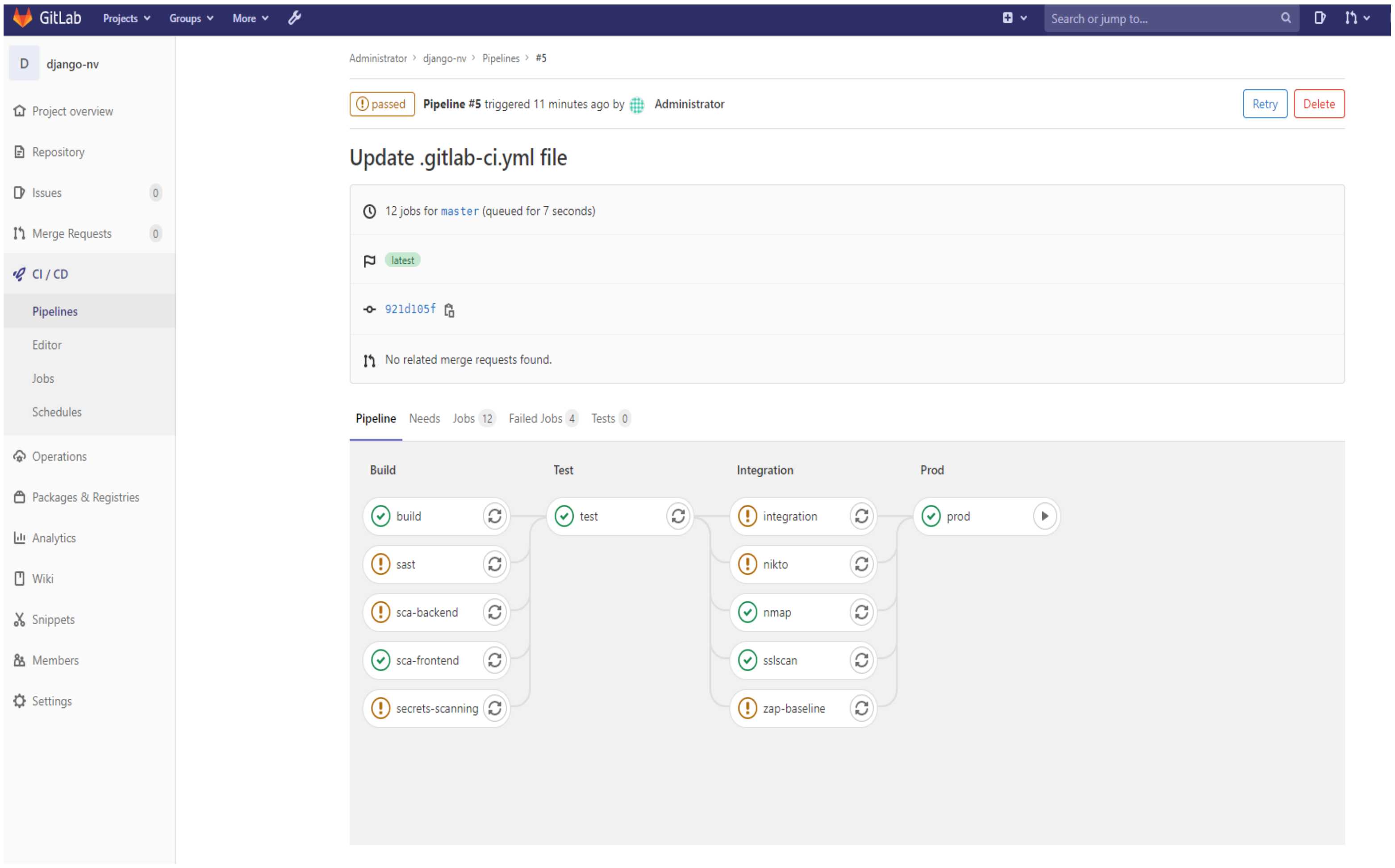


```yaml
secrets-scanning:
  stage: build
  script:
    - apk add git
    - git checkout master
    - docker run -v $(pwd):/src --rm hysnsec/trufflehog file:///src --json | tee trufflehog-output.json
  artifacts:
    paths: [trufflehog-output.json]
    when: always # What is this for?
    expire_in: one week
  allow_failure: true

# Static Application Security Testing
sast:
  stage: build
  script:
    - docker pull hysnsec/bandit  # Download bandit docker container
    # Run docker container, please refer docker security course, if this doesn't make sense to you.
    - docker run --user $(id -u):$(id -g) -v $(pwd):/src --rm hysnsec/bandit -r /src -f json -o /src/bandit-output.json
  artifacts:
    paths: [bandit-output.json]
    when: always
  allow_failure: true
```



```yaml
nikto:
  stage: integration
  script:
    - docker pull hysnsec/nikto
    - docker run --rm -v $(pwd):/tmp hysnsec/nikto -h prod-yQJa8AqM -o /tmp/nikto-output.xml
  artifacts:
nmap:
  stage: integration
  script:
    - docker pull hysnsec/nmap
    - docker run --rm -v $(pwd):/tmp hysnsec/nmap prod-yQJa8AqM -oX /tmp/nmap-output.xml
  artifacts:
    paths: [nmap-output.xml]
    when: always

zap-baseline:
  stage: integration
  script:
    - docker pull owasp/zap2docker-stable
    - docker run --user $(id -u):$(id -g) --rm -v $(pwd):/zap/wrk:rw owasp/zap2docker-stable zap-baseline.py -t https://prod-yQJa8AqM.lab.
practical-devsecops.training -J zap-output.json
  after_script:
    - docker rmi owasp/zap2docker-stable  # clean up the image to save the disk space
  artifacts:
    paths: [zap-output.json]
    when: always # What does this do?
  allow_failure: true
```

Here is the passed CI/CD pipeline in gitlab



passed Pipeline #5 triggered 11 minutes ago by Administrator

Update .gitlab-ci.yml file

12 jobs for master (queued for 7 seconds)

latest

921d105f

No related merge requests found.

Even though security issues have been found in the django.nv code, we currently don't know if these are false positives or not. We would have to do a false positive analysis in order to verify if these are true failures/issues or not.

Challenge 2 Solution:

First I will need to grab the django.nv project and cd into its folder

```
$ git clone https://gitlab.practical-devsecops.training/pdso/django.nv webapp
```

```
$ cd webapp
```

Now I need to install ansible locally

```
$ pip3 install ansible==2.10.4 ansible-lint==4.3.7
```

Next I will need to create an inventory file for ansible

```
$ cat > inventory.ini <<EOL
```

```
# DevSecOps Inventory
[devsecops]
devsecops-box-yQJa8AqM
[prod]
prod-yQJa8AqM
EOL
```

Now I will need to ensure I disable the SSH's yes/no prompts while running ansible

```
$ ssh-keyscan -t rsa prod-yQJa8AqM devsecops-box-yQJa8AqM gitlab-ce-yQJa8Aqm >>
~/.ssh/known_hosts
```

The next step is to create the playbook which will be used to run ansible

```
$ cat > playbook.yml << EOL
```

```
---
 - name: Playbook
  hosts: prod
  become: yes
  become_user: root
  tasks:
   - name: ensure nginix is at the latest version
    apt:
     name: nginx
     state: latest
  roles:
    - dev-sec.os-hardening
```

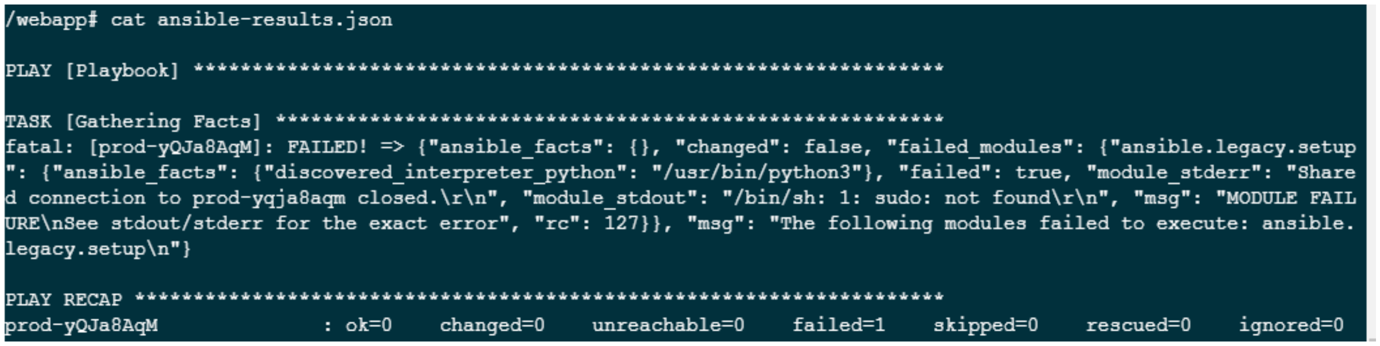
```
EOL
```

We need to ensure we have installed the proper role for ansible with ansible galaxy

```
$ ansible-galaxy install dev-sec.os-hardening
```

We can now run ansible locally with the following cmd

$ ansible-playbook -i inventory.ini playbook.yml



```
/webapp# cat ansible-results.json

PLAY [Playbook] ****************************************************************

TASK [Gathering Facts] *********************************************************
fatal: [prod-yQJa8AqM]: FAILED! => {"ansible_facts": {}, "changed": false, "failed_modules": {"ansible.legacy.setup
": {"ansible_facts": {"discovered_interpreter_python": "/usr/bin/python3"}, "failed": true, "module_stderr": "Share
d connection to prod-yqja8aqm closed.\r\n", "module_stdout": "/bin/sh: 1: sudo: not found\r\n", "msg": "MODULE FAIL
URE\nSee stdout/stderr for the exact error", "rc": 127}}, "msg": "The following modules failed to execute: ansible.
legacy.setup\n"}

PLAY RECAP *********************************************************************
prod-yQJa8AqM              : ok=0    changed=0    unreachable=0    failed=1    skipped=0    rescued=0    ignored=0
```

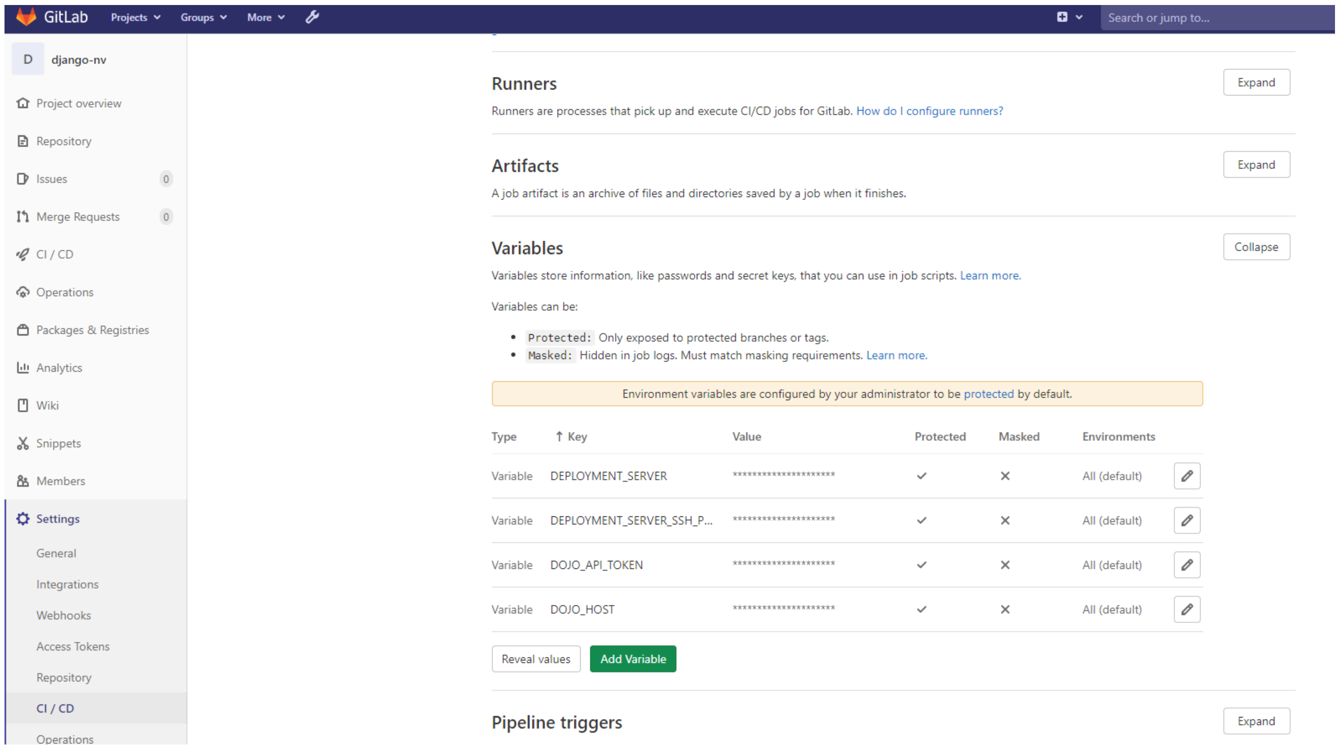
Ansible has ran successfully, we now need to install the environment variables DEPLOYMENT_SERVER and DEPLOYMENT_SERVER_SSH_PRIVKEY to our gitlab settings.

DEPLOYMENT_SERVER

prod-yQJa8AqM

DEPLOYMENT_SERVER_SSH_PRIVKEY

obtained via cmd. $ cat /root/.ssh/id_rsa



Runners

Runners are processes that pick up and execute CI/CD jobs for GitLab. How do I configure runners?

Artifacts

A job artifact is an archive of files and directories saved by a job when it finishes.

Variables

Variables store information, like passwords and secret keys, that you can use in job scripts. Learn more.

Variables can be:

- `Protected:` Only exposed to protected branches or tags.
- `Masked:` Hidden in job logs. Must match masking requirements. Learn more.

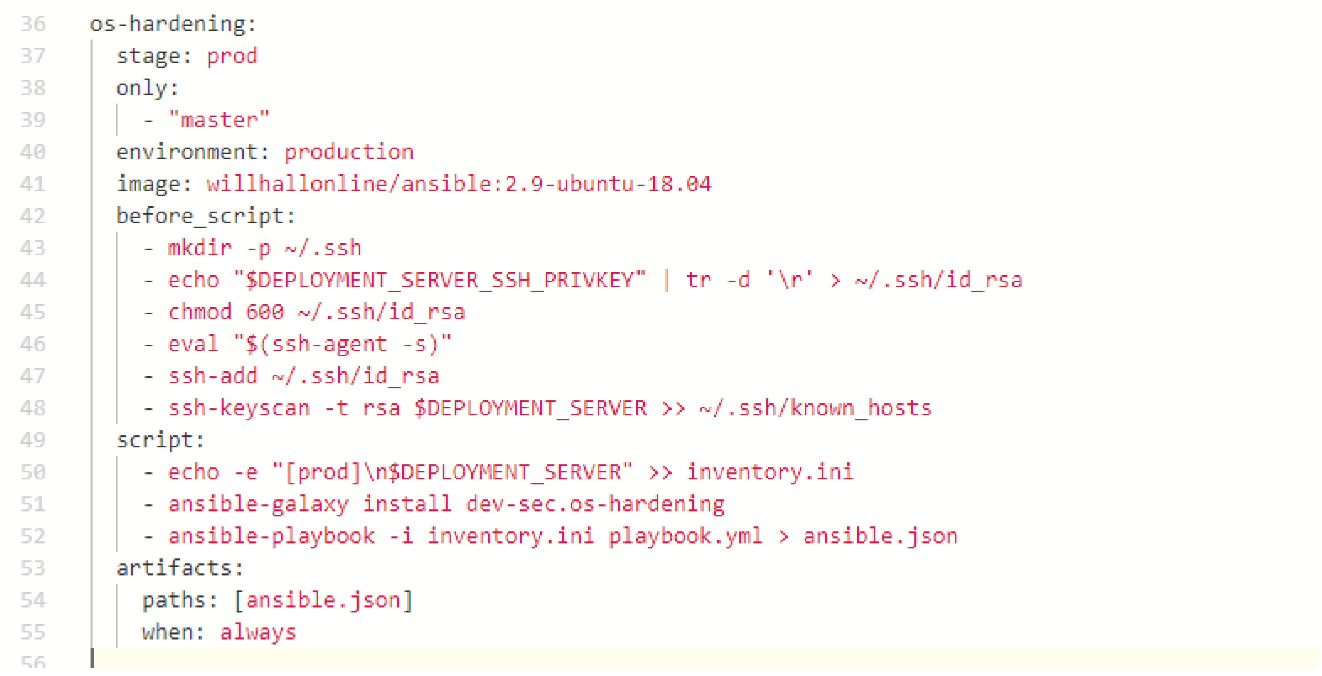
Environment variables are configured by your administrator to be protected by default.

Type	↑ Key	Value	Protected	Masked	Environments
Variable	DEPLOYMENT_SERVER	********************	✓	✕	All (default)
Variable	DEPLOYMENT_SERVER_SSH_P...	********************	✓	✕	All (default)
Variable	DOJO_API_TOKEN	********************	✓	✕	All (default)
Variable	DOJO_HOST	********************	✓	✕	All (default)

Pipeline triggers

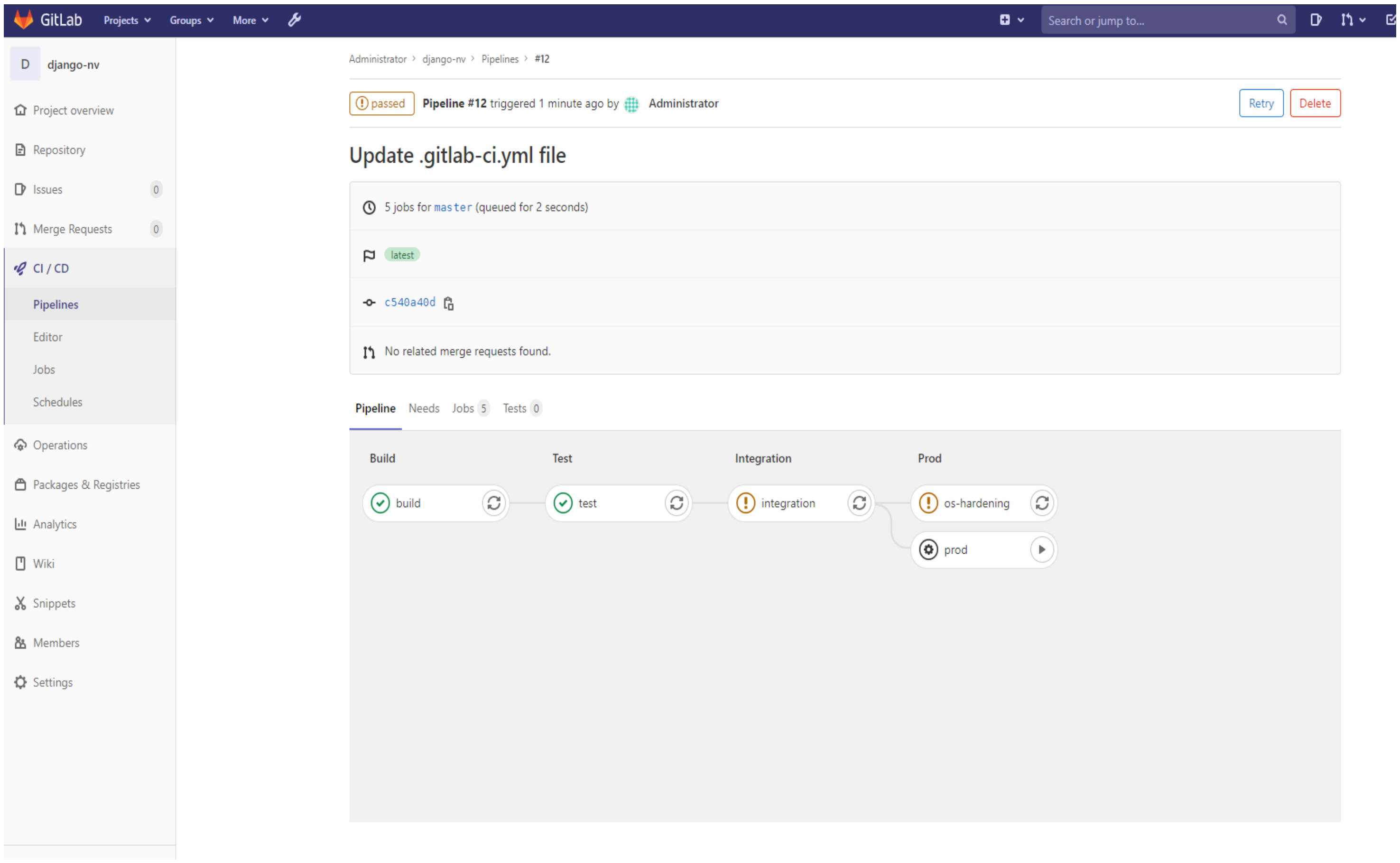
Be sure to upload your playbook to gitlab before running the os-hardening job on gitlab. This can be done via git commands. Or just create new file on the django.nv project and copy/paste the playbook code into the new file. And name it playbook.yml

Please see gitlab.yml in challenge 2 folder for the full yml file. This is just the job for the above.



```yaml
os-hardening:
  stage: prod
  only:
    - "master"
  environment: production
  image: willhallonline/ansible:2.9-ubuntu-18.04
  before_script:
    - mkdir -p ~/.ssh
    - echo "$DEPLOYMENT_SERVER_SSH_PRIVKEY" | tr -d '\r' > ~/.ssh/id_rsa
    - chmod 600 ~/.ssh/id_rsa
    - eval "$(ssh-agent -s)"
    - ssh-add ~/.ssh/id_rsa
    - ssh-keyscan -t rsa $DEPLOYMENT_SERVER >> ~/.ssh/known_hosts
  script:
    - echo -e "[prod]\n$DEPLOYMENT_SERVER" >> inventory.ini
    - ansible-galaxy install dev-sec.os-hardening
    - ansible-playbook -i inventory.ini playbook.yml > ansible.json
  artifacts:
    paths: [ansible.json]
    when: always
```

Here is the passed challenge 2 pipeline



Administrator › django-nv › Pipelines › #12

passed **Pipeline #12** triggered 1 minute ago by Administrator

Update .gitlab-ci.yml file

5 jobs for master (queued for 2 seconds)

latest

c540a40d

No related merge requests found.

Build	Test	Integration	Prod
build	test	integration	os-hardening
			prod

Challenge 3 Solution:

First I will need to grab the django.nv project and cd into its folder

$ git clone https://gitlab.practical-devsecops.training/pdso/django.nv webapp

$ cd webapp

Now I need to install trufflehog

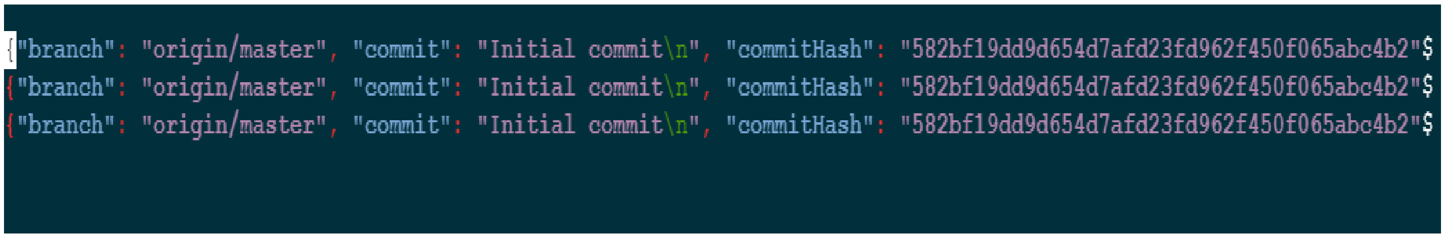
$ pip3 install trufflehog

We can run trufflehog using docker like so

$ docker run -v $(pwd):/src --rm hysnsec/trufflehog file:///src --json | tee trufflehog-output.json

We can perform a false positive analysis using nano to remove false positives.

$ nano trufflehog-output.json

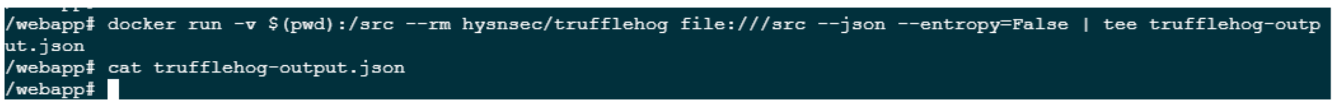


```
{"branch": "origin/master", "commit": "Initial commit\n", "commitHash": "582bf19dd9d654d7afd23fd962f450f065abc4b2"$
{"branch": "origin/master", "commit": "Initial commit\n", "commitHash": "582bf19dd9d654d7afd23fd962f450f065abc4b2"$
{"branch": "origin/master", "commit": "Initial commit\n", "commitHash": "582bf19dd9d654d7afd23fd962f450f065abc4b2"$
```

There were 3 issues found. Yet all of them are false positives via high entropy. Trufflehog actually has a cmd that removes these type of false positives.

Re-run trufflehog after performing the false positive analysis using the –entropy=False flag.

$ docker run -v $(pwd):/src --rm hysnsec/trufflehog file:///src –json | tee trufflehog-output.json

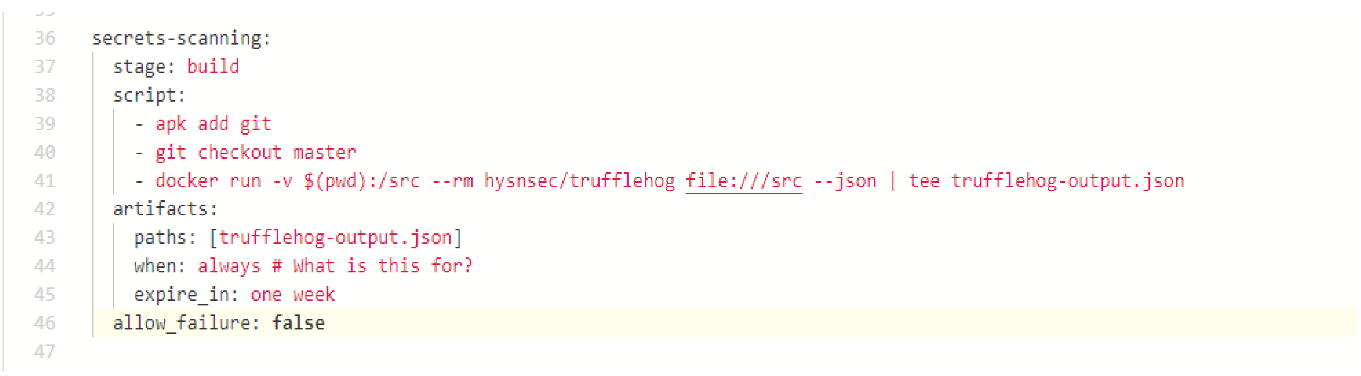


```
/webapp# docker run -v $(pwd):/src --rm hysnsec/trufflehog file:///src --json --entropy=False | tee trufflehog-outp
ut.json
/webapp# cat trufflehog-output.json
/webapp# 
```

We see that these new results no longer show the previous false positives.

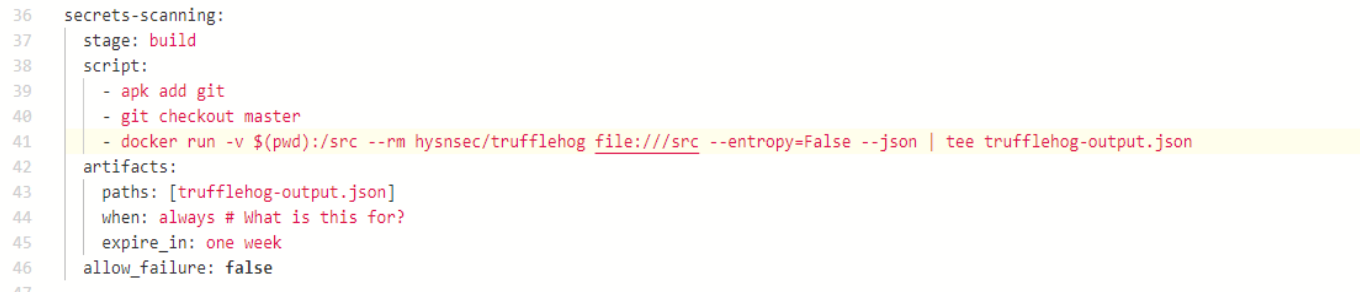
Please see gitlab.yml in challenge 3 folder for the full yml file. This is just the job for the above. [NOTE the full gitlab has code with the false positives, as challenge directions did not indicate to update it. Images are below for the jobs with and without false positives same for the pipeline with and without the FP]

With false positives



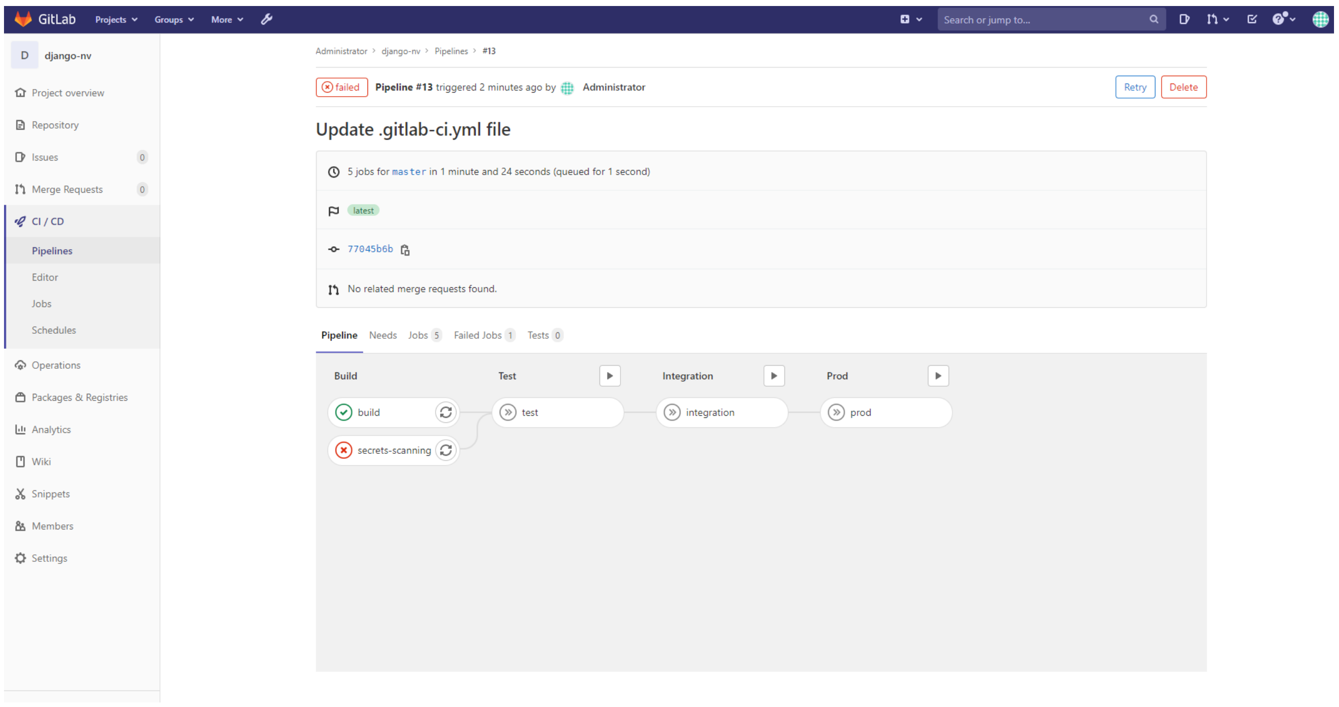
```yaml
secrets-scanning:
  stage: build
  script:
    - apk add git
    - git checkout master
    - docker run -v $(pwd):/src --rm hysnsec/trufflehog file:///src --json | tee trufflehog-output.json
  artifacts:
    paths: [trufflehog-output.json]
    when: always # What is this for?
    expire_in: one week
  allow_failure: false
```

without false positives



```yaml
secrets-scanning:
  stage: build
  script:
    - apk add git
    - git checkout master
    - docker run -v $(pwd):/src --rm hysnsec/trufflehog file:///src --entropy=False --json | tee trufflehog-output.json
  artifacts:
    paths: [trufflehog-output.json]
    when: always # What is this for?
    expire_in: one week
  allow_failure: false
```

Here is the failed challenge 3 pipeline (we wanted it to fail, and not by using exit 1)



failed **Pipeline #13** triggered 2 minutes ago by Administrator

Update .gitlab-ci.yml file

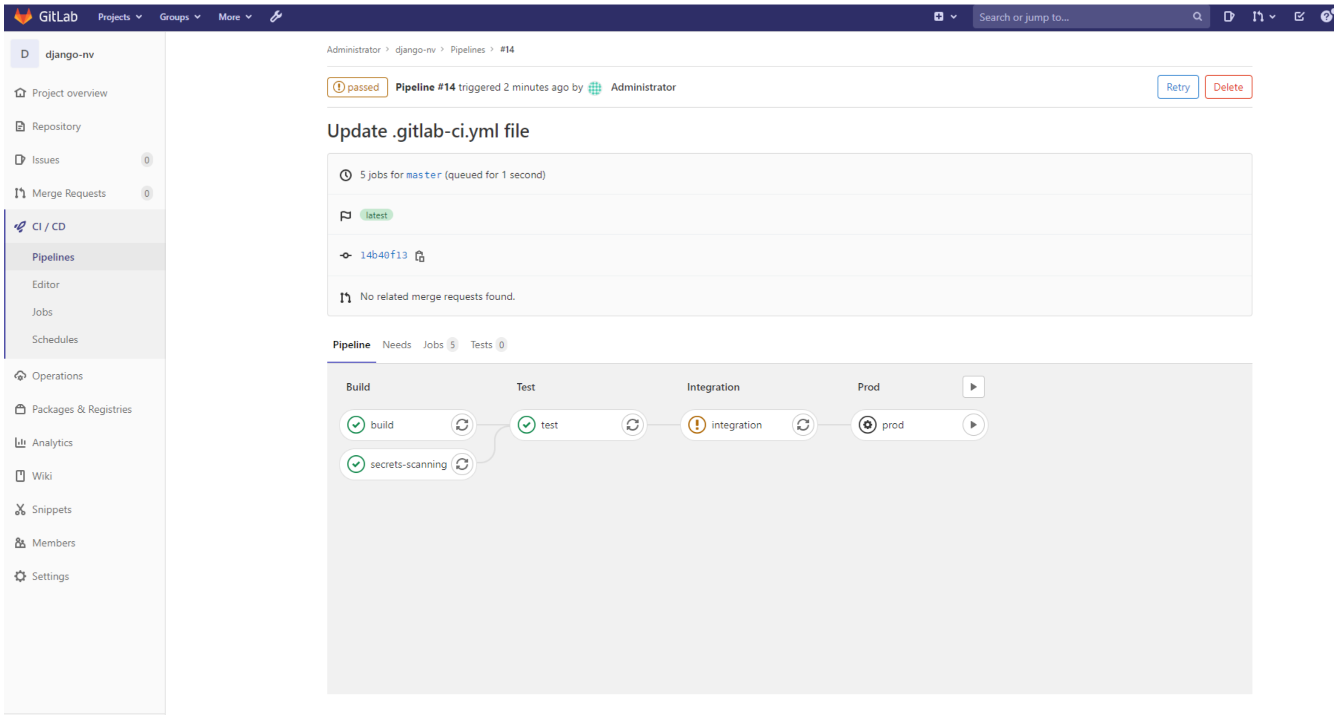
5 jobs for master in 1 minute and 24 seconds (queued for 1 second)

latest

77045b6b

No related merge requests found.

and the passed pipeline once we cleaned up the false positives



passed Pipeline #14 triggered 2 minutes ago by Administrator

Update .gitlab-ci.yml file

5 jobs for master (queued for 1 second)

latest

14b40f13

No related merge requests found.

Challenge 4 Solution:

First I will need to grab the django.nv project and cd into its folder

```
$ git clone https://gitlab.practical-devsecops.training/pdso/django.nv webapp
```

```
$ cd webapp
```

The next step would be to install Zap via docker

```
$ docker pull owasp/zap2docker-stable
```

We can run zap with the following command

```
$ docker run --user $(id -u):$(id -g) --rm -v $(pwd):/zap/wrk:rw owasp/zap2docker-stable zap-baseline.py -t https://prod-yQJa8AqM.lab.practical-devsecops.training -J zap-output.json
```

We want to upload the results to defect dojo. To do that, we will need to grab the upload-results.py python script using curl, like so.

```
$ curl https://gitlab.practical-devsecops.training/-/snippets/3/raw -o upload-results.py
```

One more step before we can upload. Our python script requires the use of requests module. So we just need to quickly install it.

```
$ pip3 install requests
```

Now to Upload it to Defect Dojo!

```
$ python3 upload-results.py --host $DOJO_HOST --api_key $DOJO_API_TOKEN --engagement_id 1 --product_id 1 --lead_id 1 --environment "Production" --result_file zap-output.json --scanner "ZAP Scan"
```

We can now add this into a CI/CD pipline. Before we update our gitlab yml file. We need to first add 2 environment variables. DOJO_HOST and DOJO_API_TOKEN as they are needed for our upload-results.py

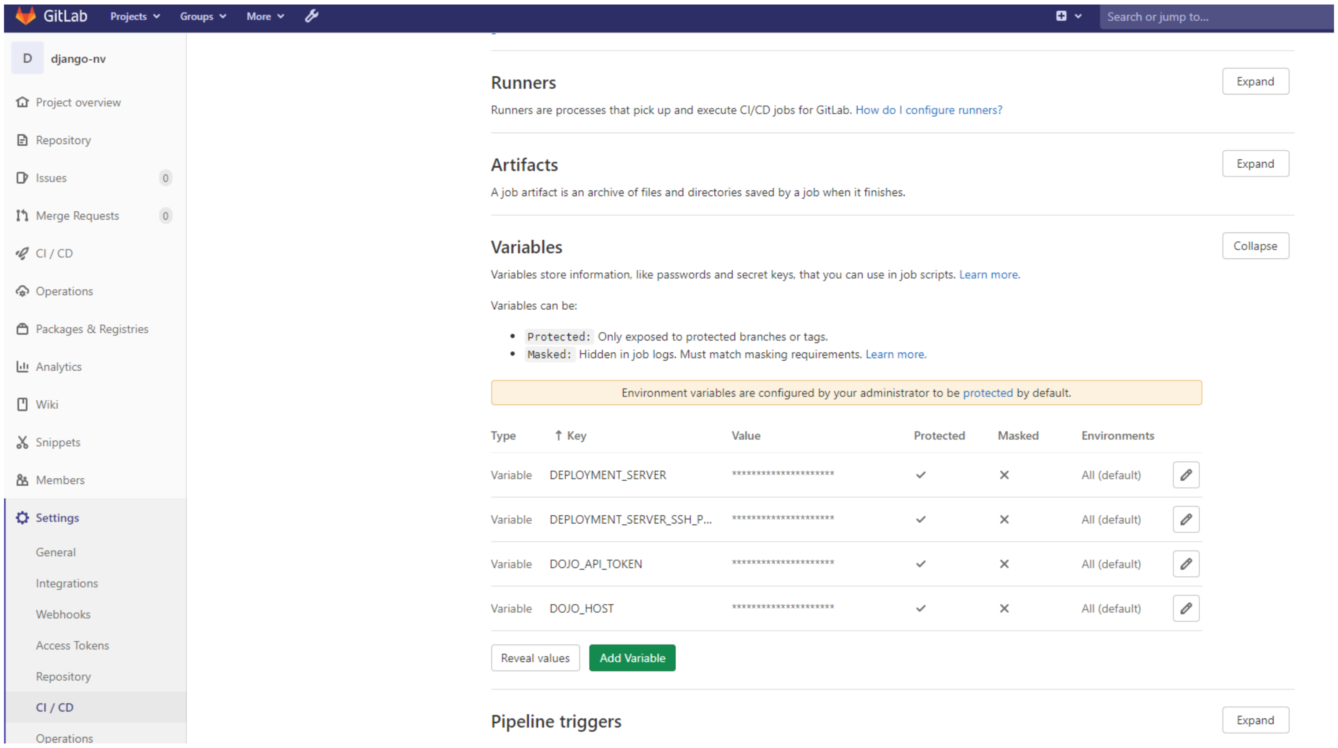
DOJO_HOST

dojo-yQJa8AqM.lab.practical-devsecops.training

DOJO_API_TOKEN

found at url https://dojo-yqja8aqm.lab.practical-devsecops.training/api/key-v2

ours is: 77bd94a91cc12dea7a3323cd9099b831e42ced05



Runners

Runners are processes that pick up and execute CI/CD jobs for GitLab. How do I configure runners?

Artifacts

A job artifact is an archive of files and directories saved by a job when it finishes.

Variables

Variables store information, like passwords and secret keys, that you can use in job scripts. Learn more.

Variables can be:

- `Protected:` Only exposed to protected branches or tags.
- `Masked:` Hidden in job logs. Must match masking requirements. Learn more.

Environment variables are configured by your administrator to be protected by default.

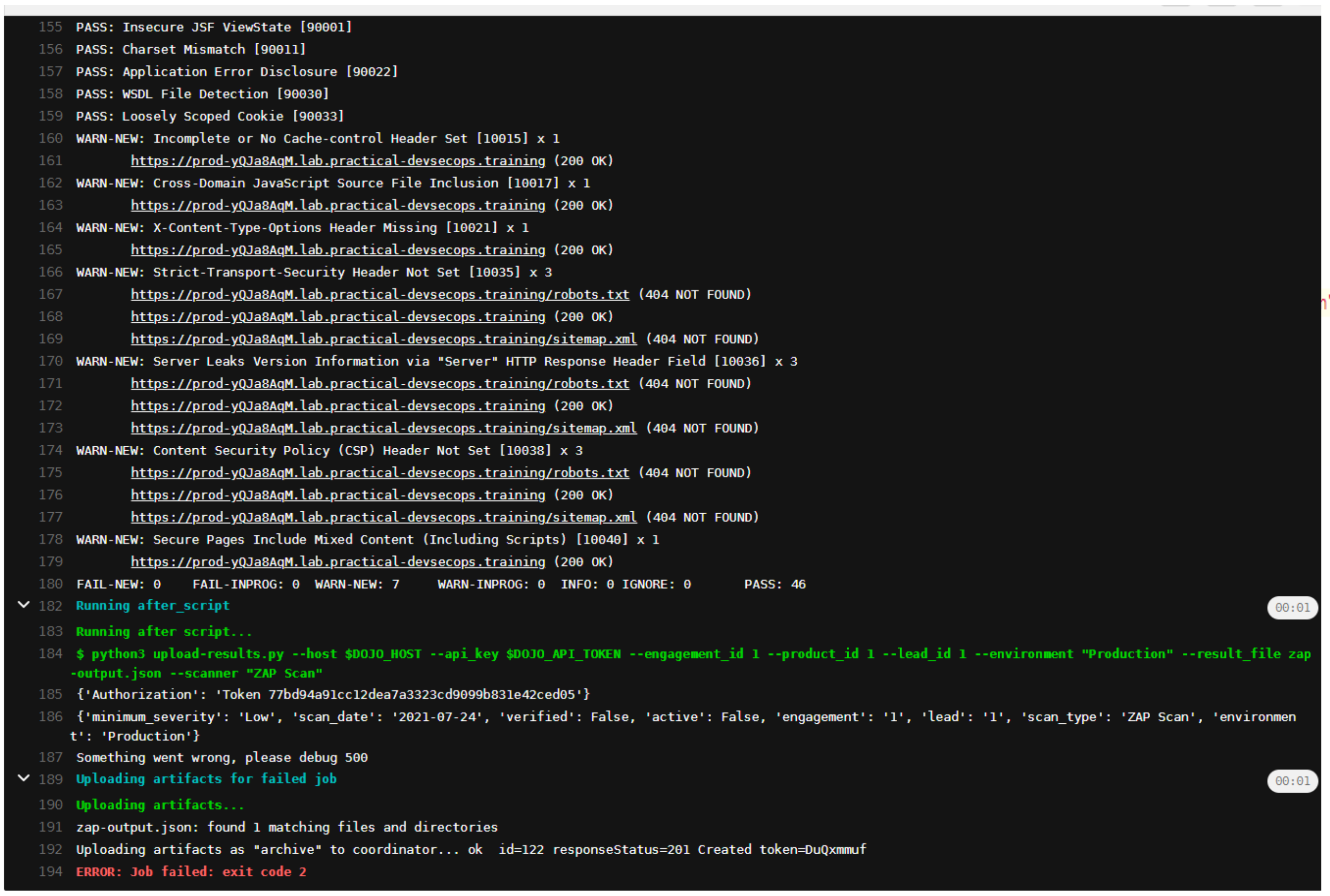
Type	↑ Key	Value	Protected	Masked	Environments
Variable	DEPLOYMENT_SERVER	********************	✓	✕	All (default)
Variable	DEPLOYMENT_SERVER_SSH_P...	********************	✓	✕	All (default)
Variable	DOJO_API_TOKEN	********************	✓	✕	All (default)
Variable	DOJO_HOST	********************	✓	✕	All (default)

Pipeline triggers

We also need to be sure we add the upload-results.py to our Gitlab server. This can be done via git command like so.

```
git config --global user.email "student@pdevsecops.com"
git config --global user.name "student"
git clone http://root:pdso-training@gitlab-ce-yQJa8AqM.lab.practical-devsecops.training/root/django-nv.git

curl https://gitlab.practical-devsecops.training/-/snippets/3/raw -o upload-results.py
git add upload-results.py
git commit -m "adding upload python file"
git push origin master
```

Please see gitlab.yml in challenge 4 folder for the full yml file. This is just the job for the above.



```
PASS: Insecure JSF ViewState [90001]
PASS: Charset Mismatch [90011]
PASS: Application Error Disclosure [90022]
PASS: WSDL File Detection [90030]
PASS: Loosely Scoped Cookie [90033]
WARN-NEW: Incomplete or No Cache-control Header Set [10015] x 1
        https://prod-yQJa8AqM.lab.practical-devsecops.training (200 OK)
WARN-NEW: Cross-Domain JavaScript Source File Inclusion [10017] x 1
        https://prod-yQJa8AqM.lab.practical-devsecops.training (200 OK)
WARN-NEW: X-Content-Type-Options Header Missing [10021] x 1
        https://prod-yQJa8AqM.lab.practical-devsecops.training (200 OK)
WARN-NEW: Strict-Transport-Security Header Not Set [10035] x 3
        https://prod-yQJa8AqM.lab.practical-devsecops.training/robots.txt (404 NOT FOUND)
        https://prod-yQJa8AqM.lab.practical-devsecops.training (200 OK)
        https://prod-yQJa8AqM.lab.practical-devsecops.training/sitemap.xml (404 NOT FOUND)
WARN-NEW: Server Leaks Version Information via "Server" HTTP Response Header Field [10036] x 3
        https://prod-yQJa8AqM.lab.practical-devsecops.training/robots.txt (404 NOT FOUND)
        https://prod-yQJa8AqM.lab.practical-devsecops.training (200 OK)
        https://prod-yQJa8AqM.lab.practical-devsecops.training/sitemap.xml (404 NOT FOUND)
WARN-NEW: Content Security Policy (CSP) Header Not Set [10038] x 3
        https://prod-yQJa8AqM.lab.practical-devsecops.training/robots.txt (404 NOT FOUND)
        https://prod-yQJa8AqM.lab.practical-devsecops.training (200 OK)
        https://prod-yQJa8AqM.lab.practical-devsecops.training/sitemap.xml (404 NOT FOUND)
WARN-NEW: Secure Pages Include Mixed Content (Including Scripts) [10040] x 1
        https://prod-yQJa8AqM.lab.practical-devsecops.training (200 OK)
FAIL-NEW: 0    FAIL-INPROG: 0  WARN-NEW: 7     WARN-INPROG: 0  INFO: 0 IGNORE: 0       PASS: 46
Running after_script                                                                          00:01
Running after script...
$ python3 upload-results.py --host $DOJO_HOST --api_key $DOJO_API_TOKEN --engagement_id 1 --product_id 1 --lead_id 1 --environment "Production" --result_file zap
-output.json --scanner "ZAP Scan"
{'Authorization': 'Token 77bd94a91cc12dea7a3323cd9099b831e42ced05'}
{'minimum_severity': 'Low', 'scan_date': '2021-07-24', 'verified': False, 'active': False, 'engagement': '1', 'lead': '1', 'scan_type': 'ZAP Scan', 'environmen
t': 'Production'}
Something went wrong, please debug 500
Uploading artifacts for failed job                                                            00:01
Uploading artifacts...
zap-output.json: found 1 matching files and directories
Uploading artifacts as "archive" to coordinator... ok  id=122 responseStatus=201 Created token=DuQxmmuf
ERROR: Job failed: exit code 2
```

There are no screenshots of the passing pipeline. Something is off with DefectDojo. Following all notes and documentation in the course, I am only receiving error code 400 and error code 500 on upload. The command is correct. The parameters for upload-results are all correct. Host/apikey/scanner etc is correct. So im unsure of whats going on.

Challenge 5 Solution:

First I will need to grab the django.nv project and cd into its folder

$ git clone https://gitlab.practical-devsecops.training/pdso/django.nv webapp

$ cd webapp

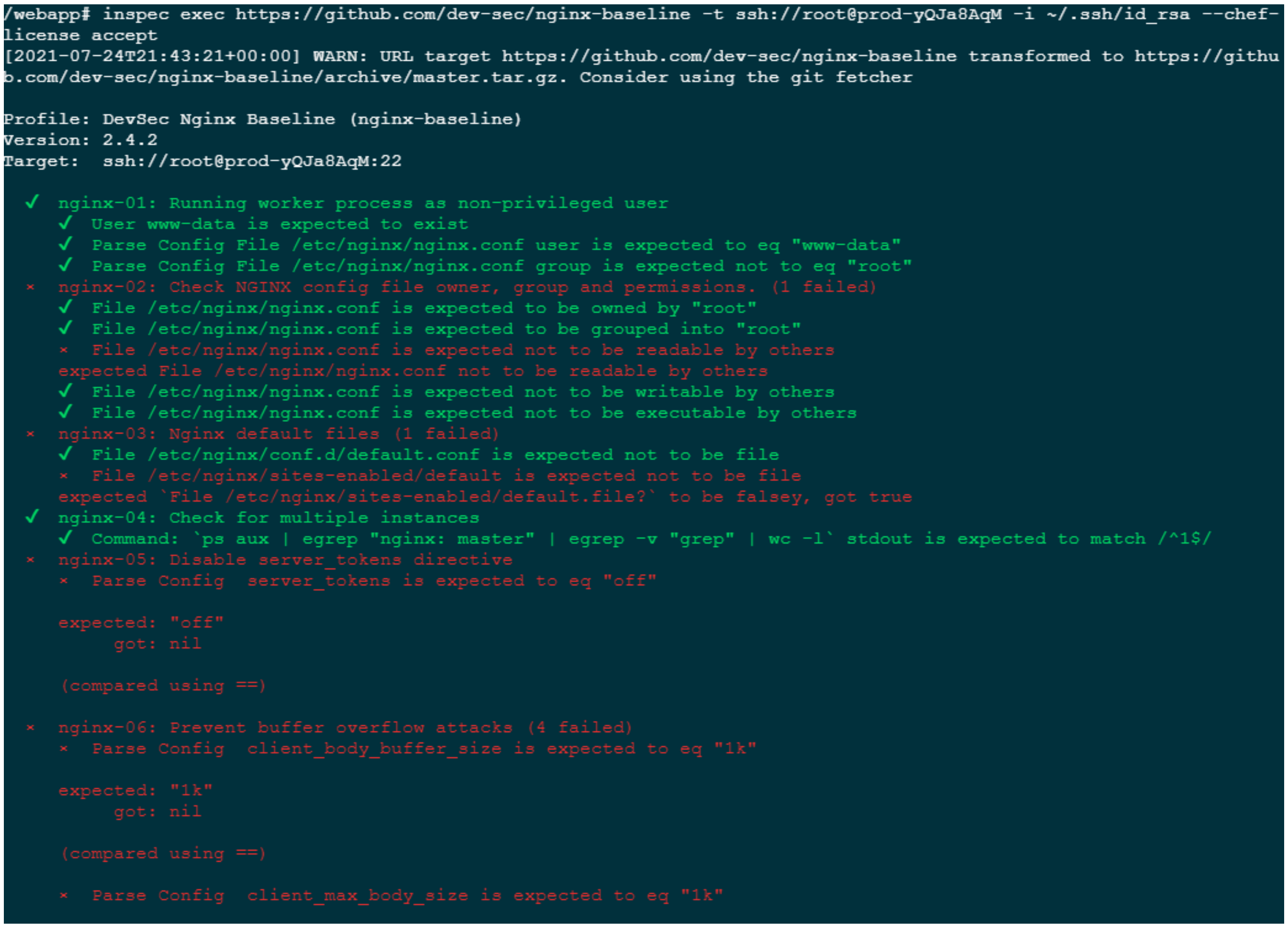
Now to download and then install inspec

$ wget https://packages.chef.io/files/stable/inspec/4.37.8/ubuntu/18.04/inspec_4.37.8-1_amd64.deb

$ dpkg -i inspec_4.37.8-1_amd64.deb

We want to run the nginx baseline profile. Lets run it on our production machine.

$ inspec exec https://github.com/dev-sec/nginx-baseline -t ssh://root@prod-yQJa8AqM -i ~/.ssh/id_rsa --chef-license accept



```
/webapp# inspec exec https://github.com/dev-sec/nginx-baseline -t ssh://root@prod-yQJa8AqM -i ~/.ssh/id_rsa --chef-
license accept
[2021-07-24T21:43:21+00:00] WARN: URL target https://github.com/dev-sec/nginx-baseline transformed to https://githu
b.com/dev-sec/nginx-baseline/archive/master.tar.gz. Consider using the git fetcher

Profile: DevSec Nginx Baseline (nginx-baseline)
Version: 2.4.2
Target:  ssh://root@prod-yQJa8AqM:22

  ✓  nginx-01: Running worker process as non-privileged user
     ✓  User www-data is expected to exist
     ✓  Parse Config File /etc/nginx/nginx.conf user is expected to eq "www-data"
     ✓  Parse Config File /etc/nginx/nginx.conf group is expected not to eq "root"
  ×  nginx-02: Check NGINX config file owner, group and permissions. (1 failed)
     ✓  File /etc/nginx/nginx.conf is expected to be owned by "root"
     ✓  File /etc/nginx/nginx.conf is expected to be grouped into "root"
     ×  File /etc/nginx/nginx.conf is expected not to be readable by others
     expected File /etc/nginx/nginx.conf not to be readable by others
     ✓  File /etc/nginx/nginx.conf is expected not to be writable by others
     ✓  File /etc/nginx/nginx.conf is expected not to be executable by others
  ×  nginx-03: Nginx default files (1 failed)
     ✓  File /etc/nginx/conf.d/default.conf is expected not to be file
     ×  File /etc/nginx/sites-enabled/default is expected not to be file
     expected `File /etc/nginx/sites-enabled/default.file?` to be falsey, got true
  ✓  nginx-04: Check for multiple instances
     ✓  Command: `ps aux | egrep "nginx: master" | egrep -v "grep" | wc -l` stdout is expected to match /^1$/
  ×  nginx-05: Disable server_tokens directive
     ×  Parse Config  server_tokens is expected to eq "off"

     expected: "off"
          got: nil

     (compared using ==)

  ×  nginx-06: Prevent buffer overflow attacks (4 failed)
     ×  Parse Config  client_body_buffer_size is expected to eq "1k"

     expected: "1k"
          got: nil

     (compared using ==)

     ×  Parse Config  client_max_body_size is expected to eq "1k"
```

We see we have some issues popping up. Lets login to the production machine and try to fix some of these issues

$ ssh root@prod-yQJa8AqM

The first issue is simply a groups/perms issue. Located at /etc/nginx/nginx.conf We want the file to only be readable by the owner. Here's how to fix it.

```
$ chmod 600 /etc/nginx/nginx.conf
```

We have another issue location at /etc/nginx/sites-enabled/default, this file is not supposed to exit. So delete it

```
$ rm /etc/nginx/sites-enabled/default
```

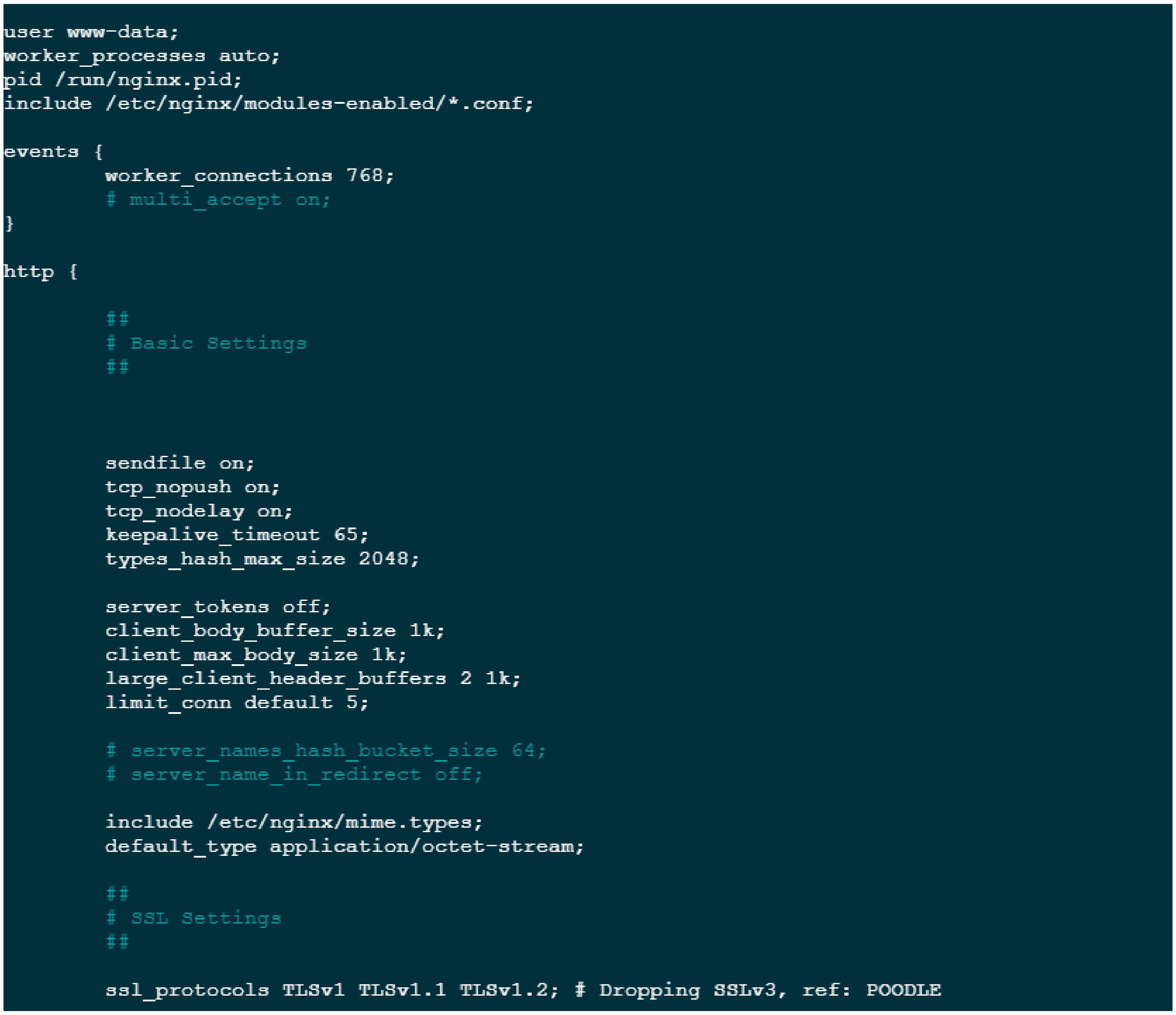
Now the remainder of the issues exist only in a single file /etc/nginx/nginx.conf.

Simply put, its looking for settings that don't exist in this file. Go through the control failures and it'll show you the names of the settings and what value you should put them as.

Add them to the http block inside of the /etc/nginx/nginx.conf.

Here are just a few that should be added [Note: there are many control failures so im only adding the fixes to a few as to not clutter, the rest are exact same format to fix. Just have name and value]

```
server_tokens off
client_body_buffer_size 1k
client_max_body_size 1k
large_client_header_buffers 2 1k
limit_conn default 5
server {
        add_header X-Frame-Options SAMEORIGIN
}
```



```
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events {
        worker_connections 768;
        # multi_accept on;
}

http {

        ##
        # Basic Settings
        ##



        sendfile on;
        tcp_nopush on;
        tcp_nodelay on;
        keepalive_timeout 65;
        types_hash_max_size 2048;

        server_tokens off;
        client_body_buffer_size 1k;
        client_max_body_size 1k;
        large_client_header_buffers 2 1k;
        limit_conn default 5;

        # server_names_hash_bucket_size 64;
        # server_name_in_redirect off;

        include /etc/nginx/mime.types;
        default_type application/octet-stream;

        ##
        # SSL Settings
        ##

        ssl_protocols TLSv1 TLSv1.1 TLSv1.2; # Dropping SSLv3, ref: POODLE
```

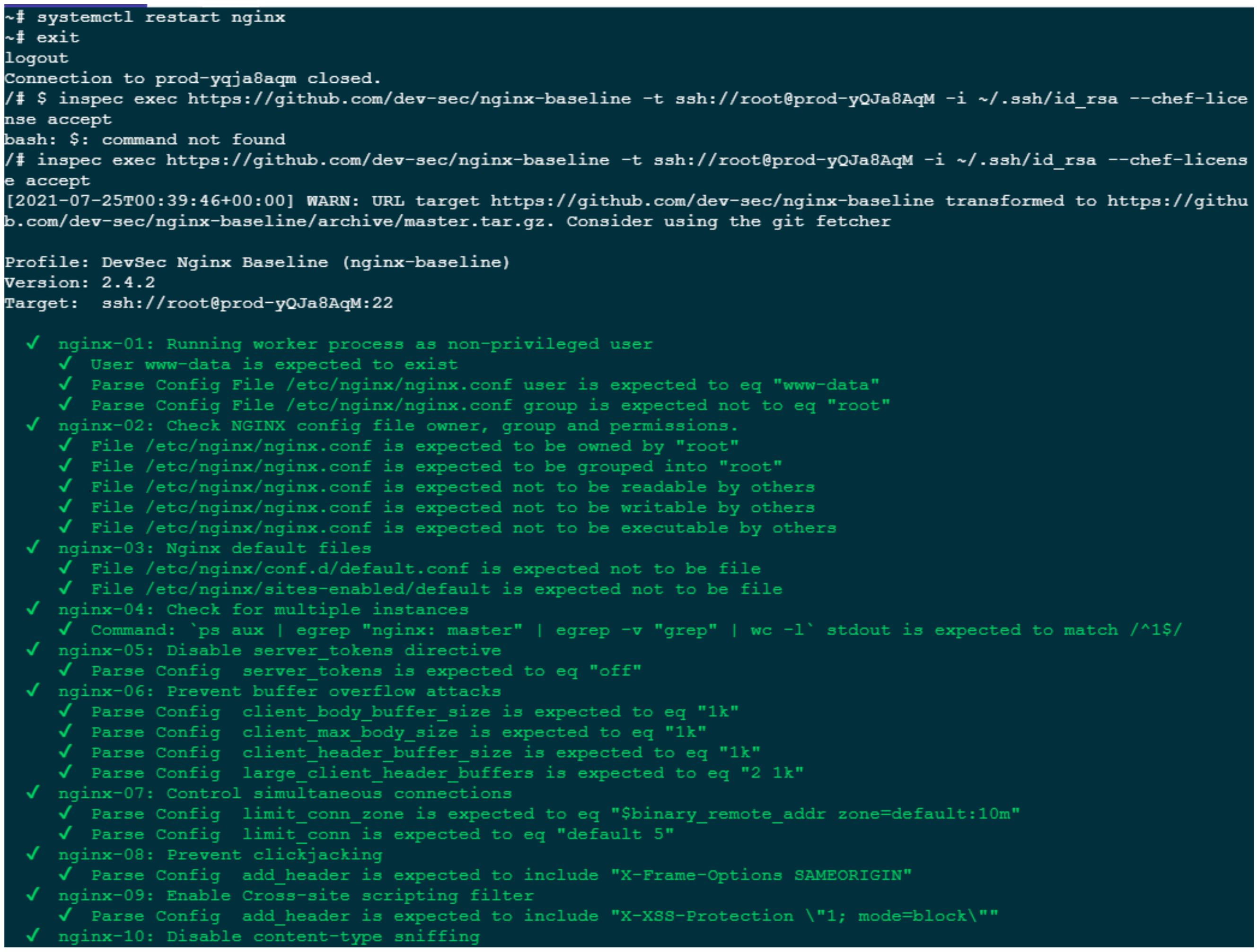
Finish going through and adding the remainder of those control failures to the http block inside of /etc/nginx/nginx.conf. Be sure to restart your nginx server for these changes to take effect.

```
$ systemctl restart nginx
```

Exit production machine and run the inspec profile again

```
$ exit
```

```
$ inspec exec https://github.com/dev-sec/nginx-baseline -t ssh://root@prod-yQJa8AqM -i ~/.ssh/id_rsa
--chef-license accept
```



```
~# systemctl restart nginx
~# exit
logout
Connection to prod-yqja8aqm closed.
/# $ inspec exec https://github.com/dev-sec/nginx-baseline -t ssh://root@prod-yQJa8AqM -i ~/.ssh/id_rsa --chef-lice
nse accept
bash: $: command not found
/# inspec exec https://github.com/dev-sec/nginx-baseline -t ssh://root@prod-yQJa8AqM -i ~/.ssh/id_rsa --chef-licens
e accept
[2021-07-25T00:39:46+00:00] WARN: URL target https://github.com/dev-sec/nginx-baseline transformed to https://githu
b.com/dev-sec/nginx-baseline/archive/master.tar.gz. Consider using the git fetcher

Profile: DevSec Nginx Baseline (nginx-baseline)
Version: 2.4.2
Target:  ssh://root@prod-yQJa8AqM:22

  ✓  nginx-01: Running worker process as non-privileged user
     ✓  User www-data is expected to exist
     ✓  Parse Config File /etc/nginx/nginx.conf user is expected to eq "www-data"
     ✓  Parse Config File /etc/nginx/nginx.conf group is expected not to eq "root"
  ✓  nginx-02: Check NGINX config file owner, group and permissions.
     ✓  File /etc/nginx/nginx.conf is expected to be owned by "root"
     ✓  File /etc/nginx/nginx.conf is expected to be grouped into "root"
     ✓  File /etc/nginx/nginx.conf is expected not to be readable by others
     ✓  File /etc/nginx/nginx.conf is expected not to be writable by others
     ✓  File /etc/nginx/nginx.conf is expected not to be executable by others
  ✓  nginx-03: Nginx default files
     ✓  File /etc/nginx/conf.d/default.conf is expected not to be file
     ✓  File /etc/nginx/sites-enabled/default is expected not to be file
  ✓  nginx-04: Check for multiple instances
     ✓  Command: `ps aux | egrep "nginx: master" | egrep -v "grep" | wc -l` stdout is expected to match /^1$/
  ✓  nginx-05: Disable server_tokens directive
     ✓  Parse Config  server_tokens is expected to eq "off"
  ✓  nginx-06: Prevent buffer overflow attacks
     ✓  Parse Config  client_body_buffer_size is expected to eq "1k"
     ✓  Parse Config  client_max_body_size is expected to eq "1k"
     ✓  Parse Config  client_header_buffer_size is expected to eq "1k"
     ✓  Parse Config  large_client_header_buffers is expected to eq "2 1k"
  ✓  nginx-07: Control simultaneous connections
     ✓  Parse Config  limit_conn_zone is expected to eq "$binary_remote_addr zone=default:10m"
     ✓  Parse Config  limit_conn is expected to eq "default 5"
  ✓  nginx-08: Prevent clickjacking
     ✓  Parse Config  add_header is expected to include "X-Frame-Options SAMEORIGIN"
  ✓  nginx-09: Enable Cross-site scripting filter
     ✓  Parse Config  add_header is expected to include "X-XSS-Protection \"1; mode=block\""
  ✓  nginx-10: Disable content-type sniffing
```