DevSecOps with OpenSource

Hello $username!

Welcome to the w00d00 workshop where you will learn (or recap) on some tools and techniques helping you to automagically identify vulnerabilities in applications and infrastructures as well as to address and track them in manageable form.

Speaking of automation, tools in this field have always been expensive or ~~crapy~~ not efficient, or both, which usually left the most of us, vulnerability hunters, with occasional OpenVAS/Nessus/Burp scans and, maybe (for millionaires) some SAST for the code. And no, SonarQube is not a SAST, unless you enable the right plugins in it, which developers usually don't do because these produce even more noise in addition to thousands of alerts they already get from it.

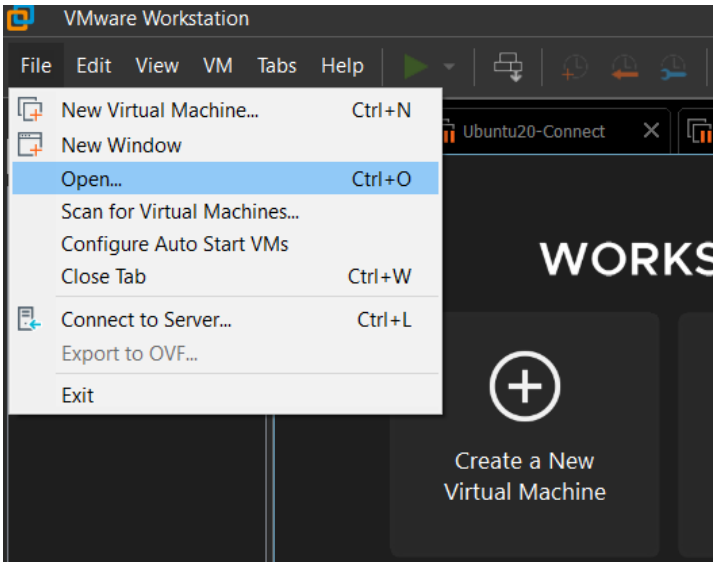
However, the time goes, and some good researchers have developed decent tools and even made them available for the public, so today I'd like to show you some ways you may want to build your vulnerability management process without spending too much time for developing own tools or money to buy expensive solutions.

1. Lab preps

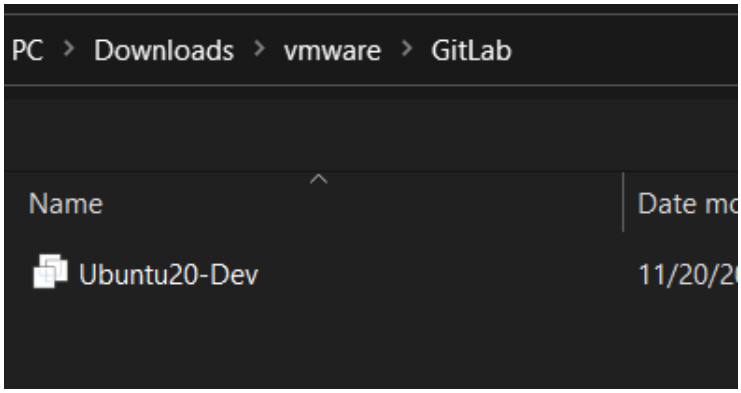
To run the labs please download **2 VMs - scansuite.zip and GitLab.zip** These are rather big images, and you may want to download them before coming to the conference as the Internet speed there may disappoint you.

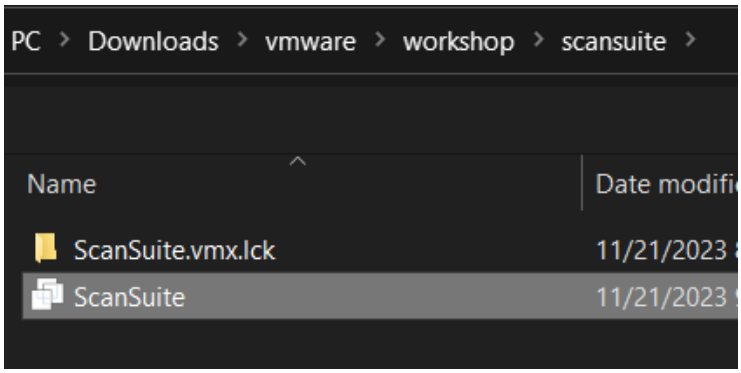
To run VMs you'd need a VMWare Workstation/Fusion/Player/ESXi – something compatible with VMWare images.

Once you have downloaded the VMs, unzip the files and open the VMs via VMWare interface:



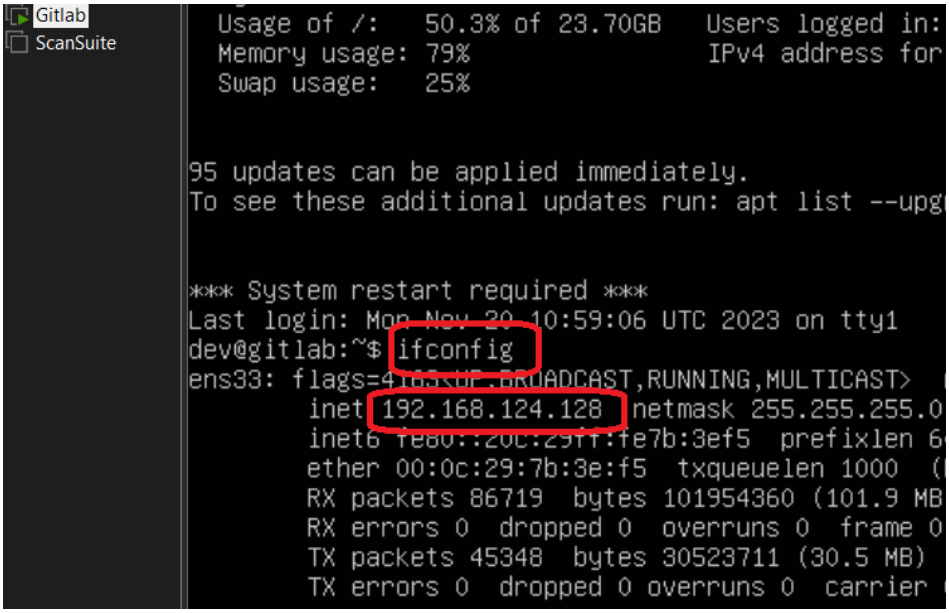
GitLab VM has a weird Ubuntu20-Dev name, that's OK just open it:



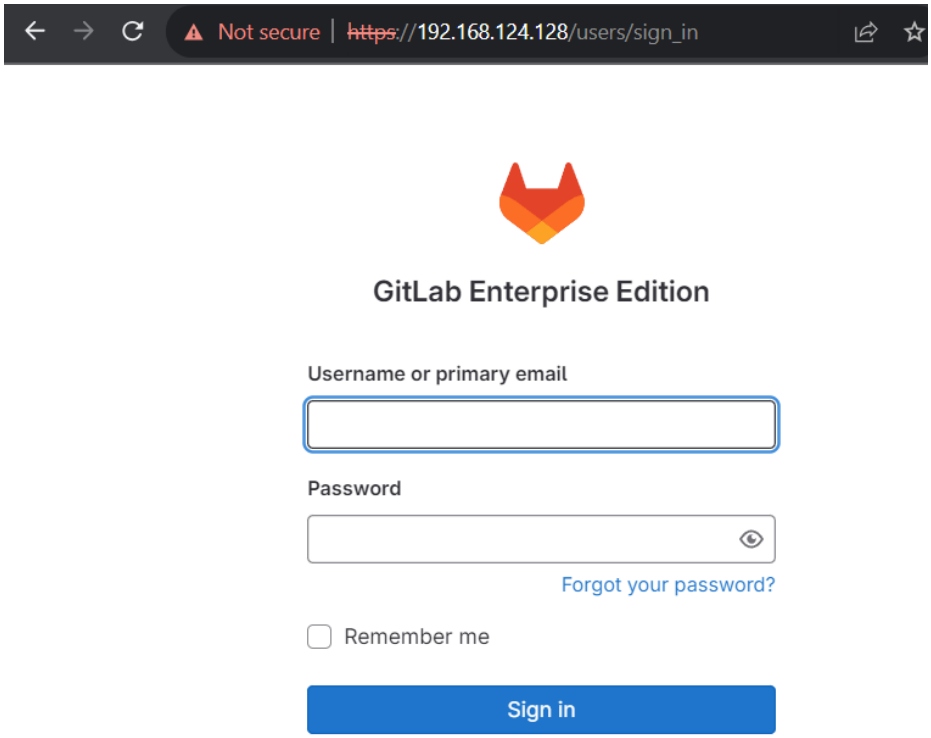


Boot the machines and select "I copied it" when asked by the VMWare.

Login to the **GitLab VM user dev with password gitlab1** over console and get it's IP address:



Open the browser and check if the GitLab server is loaded. That would be https://YOUR_GITLAB_IP :



It may load 3-5 minutes after the machine is booted, just wait a bit if you don't see a login screen.

Obviously, you need to agree with the usage of self-signed certificates in every web app we will be using.

Login to the GitLab **user root** with **password NOaQYCqYlOwUzGGs9+v0Wogmy5bVmfA2bmlxa/GTBJQ=**

Now, let's move to the second VM. Boot it and login as **scanbot / scanbot**

Execute the script in the home folder to specify the **GitLab VM IP** address with sudo:

sudo ./conn-gitlab.sh

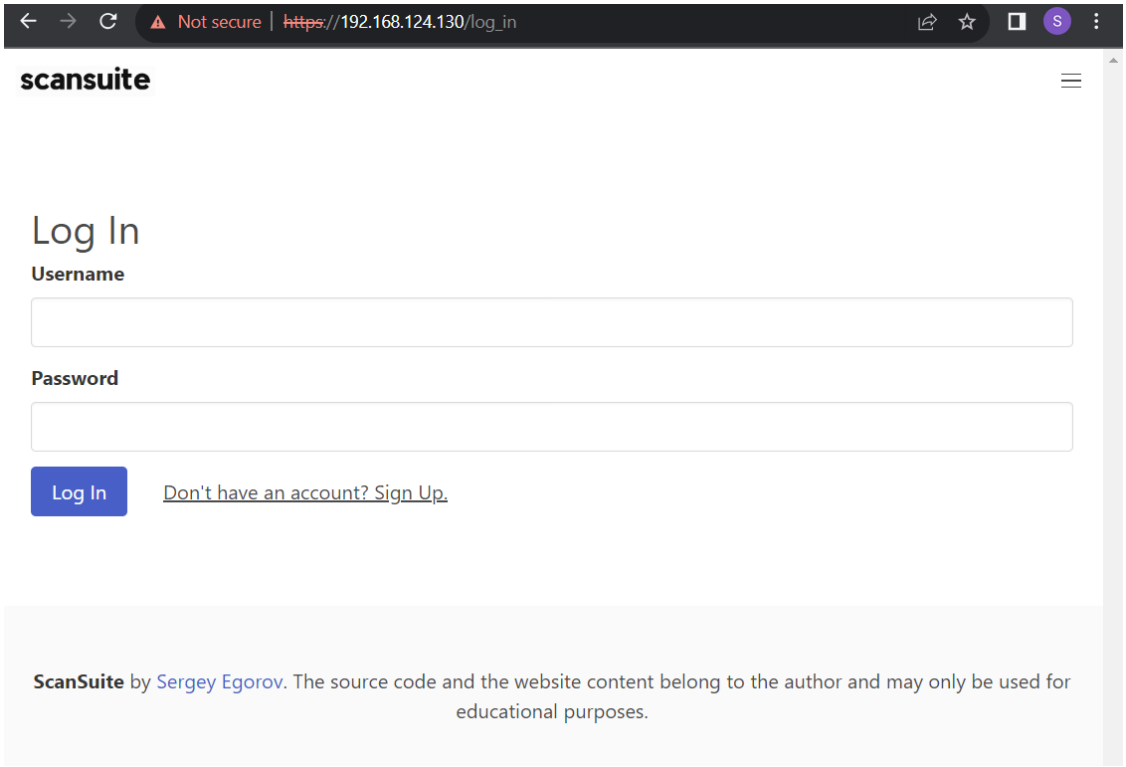
```
scanbot@scansuite:~$ sudo ./conn-gitlab.sh
[sudo] password for scanbot:
Enter GitLab IP address: 192.168.124.128
PING gitlab-local (192.168.124.128) 56(84) bytes of data.
64 bytes from gitlab-local (192.168.124.128): icmp_seq=1 ttl=64 time=0.307 ms
64 bytes from gitlab-local (192.168.124.128): icmp_seq=2 ttl=64 time=0.275 ms
64 bytes from gitlab-local (192.168.124.128): icmp_seq=3 ttl=64 time=1.23 ms
64 bytes from gitlab-local (192.168.124.128): icmp_seq=4 ttl=64 time=0.256 ms

--- gitlab-local ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3031ms
rtt min/avg/max/mdev = 0.256/0.516/1.227/0.410 ms
scanbot@scansuite:~$
```

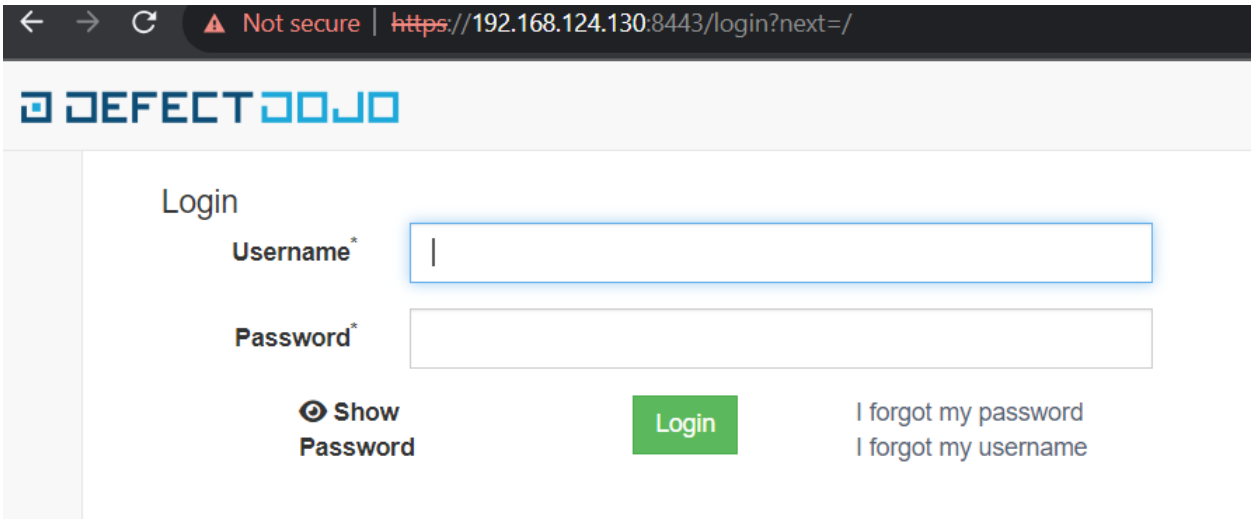
At the end it should successfully ping the gitlab-local host, which is your GitLab VM.

If gitlab-local is not reachable, try to troubleshoot by separately executing commands in the script with sudo privs

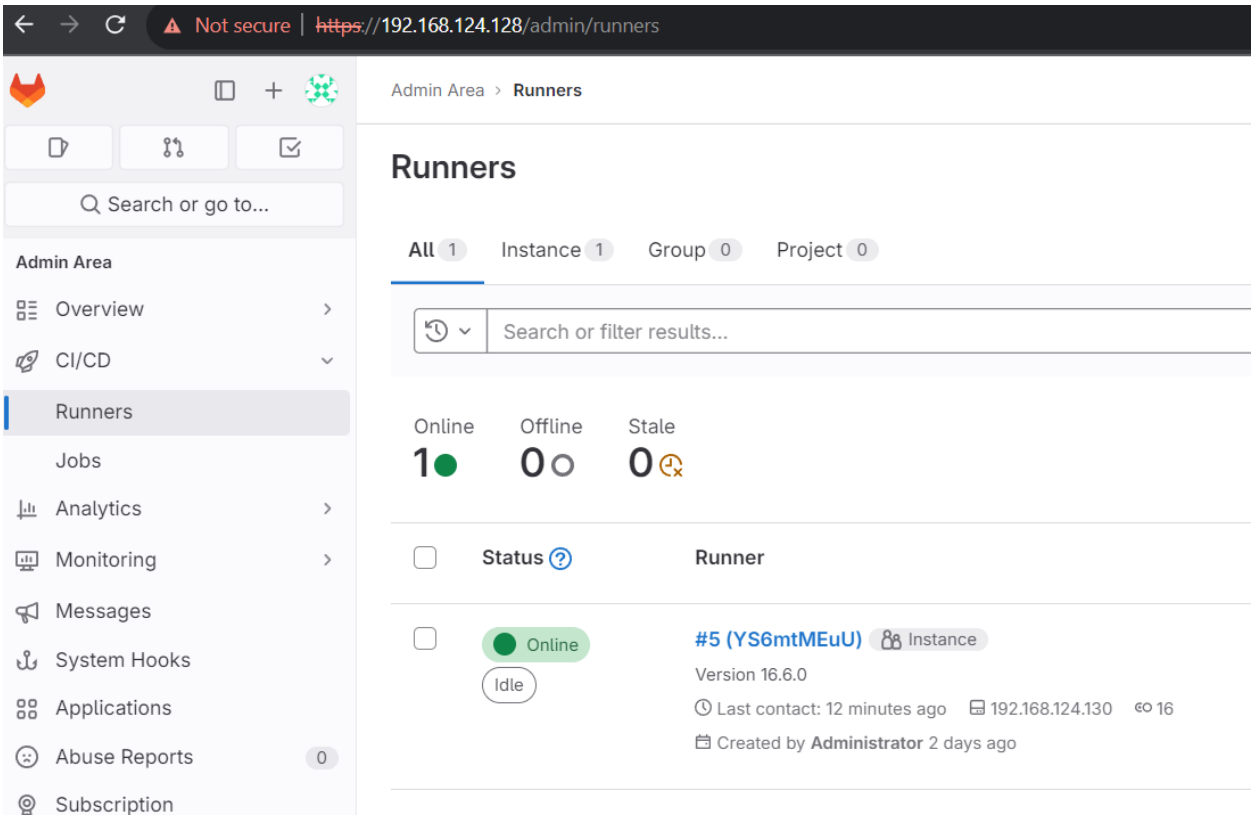
Now, open the ScanSuite web interface via https://YOUR_SCANSUITE_IP . Login with **user admin and password COzAVQwewpBVAMHNj**



Also open the DefectDojo via https://YOUR_SCANSUITE_IP:8443 The **username** here would be **admin** and the **password YUcfb8qUM6yCecD6C1PnS6**



Now, get back to the GitLab web console and follow to the https://YOUR_GITLAB_IP/admin/runners page:



You should see one online / green runner. If you don't, try to reboot your scansuite (runner is there) host and check again.

Please ensure all above works before you start the labs. Otherwise approach me (contacts are on the last page) and we troubleshoot together.

In addition, please download the application **source codes vulnado.zip and django.nV.zip** These are just zipped repos for 2 well known vulnerable apps. You do not need to unzip them, just save them somewhere and upload the zip when requested.

2. Lab overview

Here is how our stack looks like. We have got a few open-source scanners, they can produce the results, exportable to the vulnerability management system, DefectDojo, which is also free to use.

Although it not necessary, one may want to have some orchestration engine for the scanners, as the single point of interaction with some interfaces to it, like web, cli or callable via CI/CD pipeline. I have developed one and called it ScanSuite, you may use it exclusively for these labs because it nicely glues

the scanners with the respective systems and helps you to learn the tools without getting into the syntax details of every individual tool.

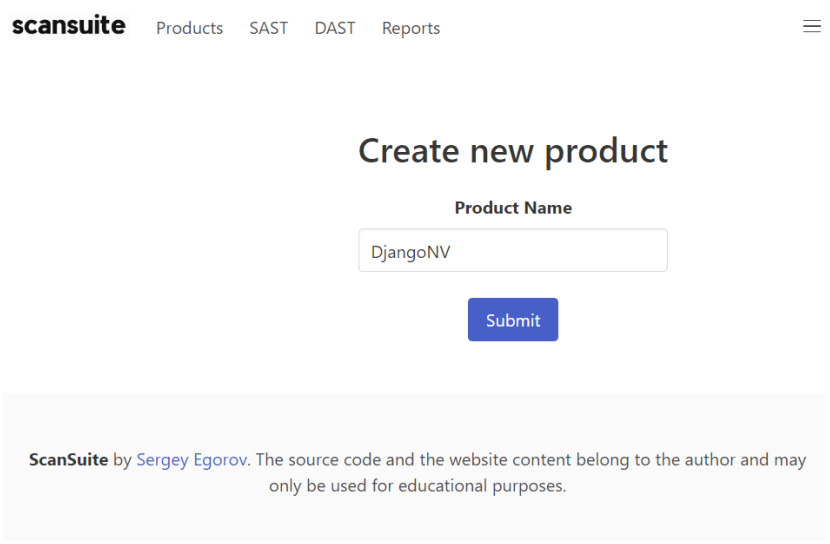
We won't be exporting issues to Jira this time, although this can be done from DefectDojo out of box.

So, let's warm up a bit.

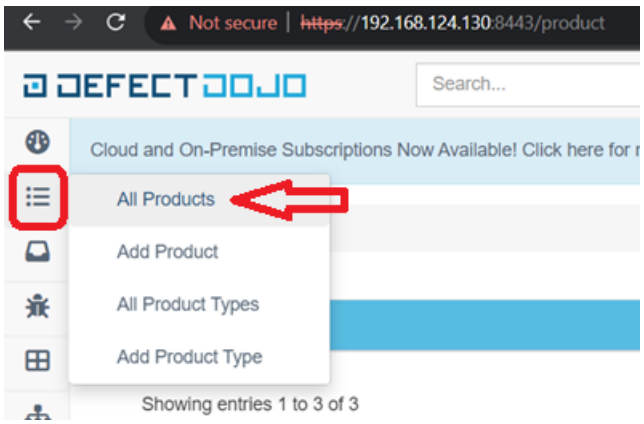
Lab 1 Vulnerable python application

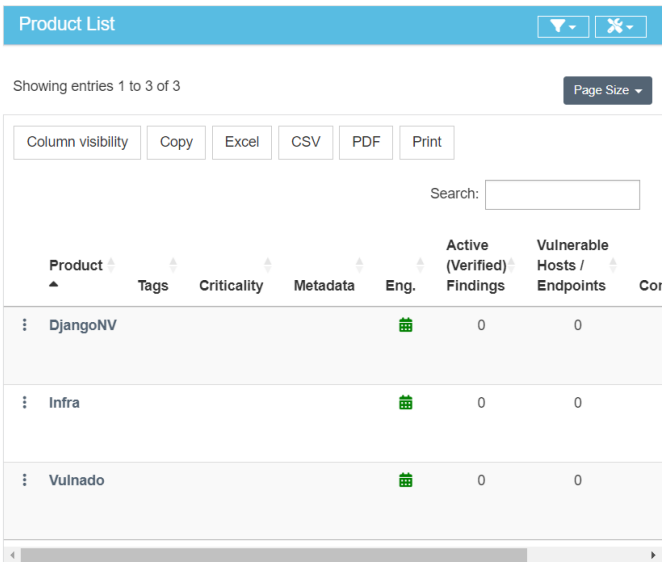
As your developer peers may approach you and ask you to check the bugs in their code because "you are the security expert", you can send them back to their salt mines OR you can run some checks and see what comes up.

First, we need to create a "Product" in terms of DefectDojo. That would be a "folder", which will contain one or many "subfolders" called "Engagements". Let's open the "Products" Tab in ScanSuite and create 3 products one by one: **DjangoNV, Vulnado** and **Infra**

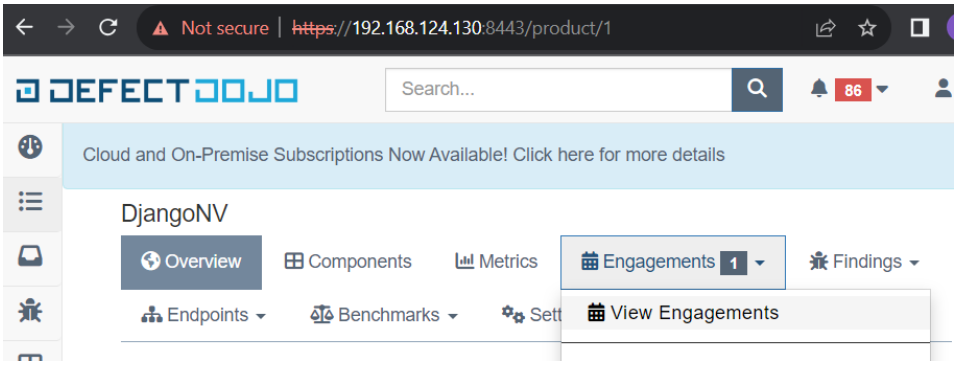


You won't get a result in ScanSuite because it just shoots the command to DefectDojo, so let's open it and check the created Products:

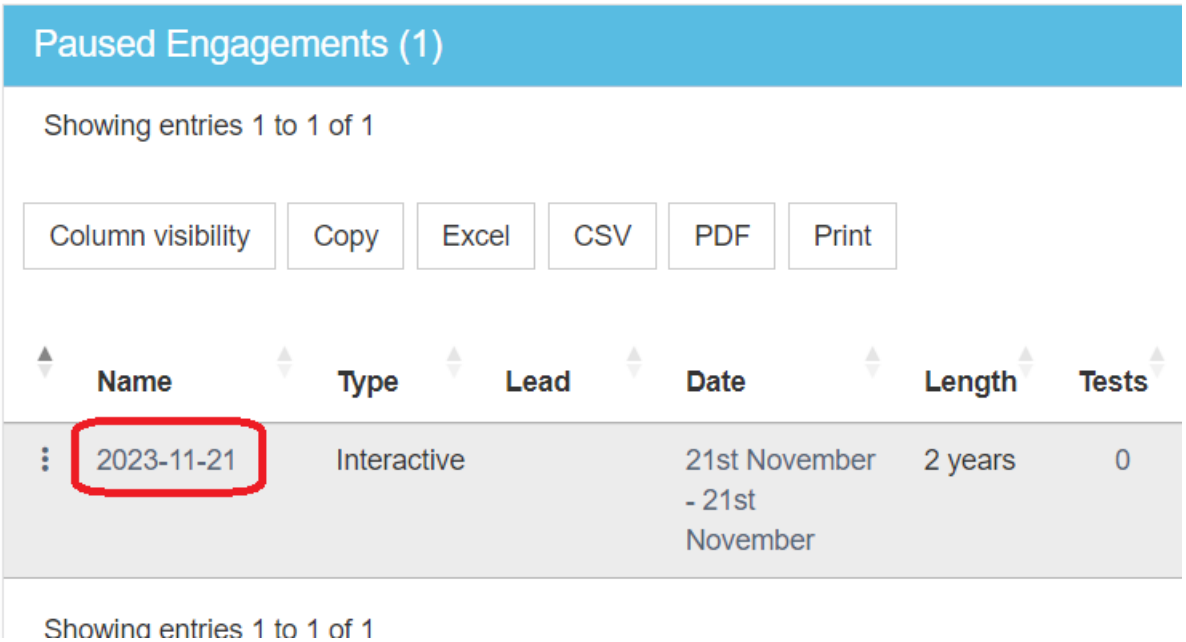




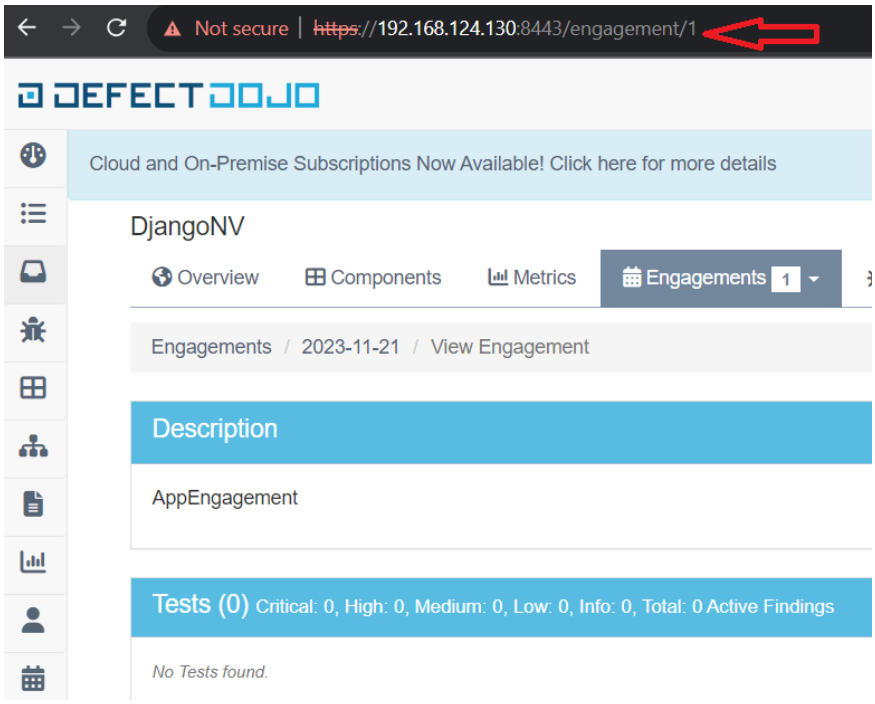
Click on the DjangoNV one and open “Engagements - View Engagements” in it:



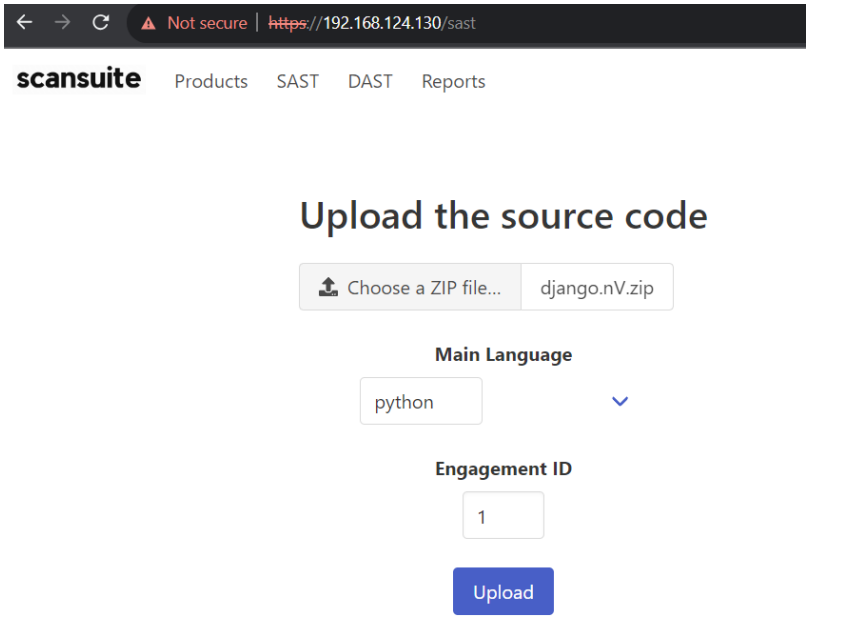
You will see the created Engagement with the current date as the name so just select it:



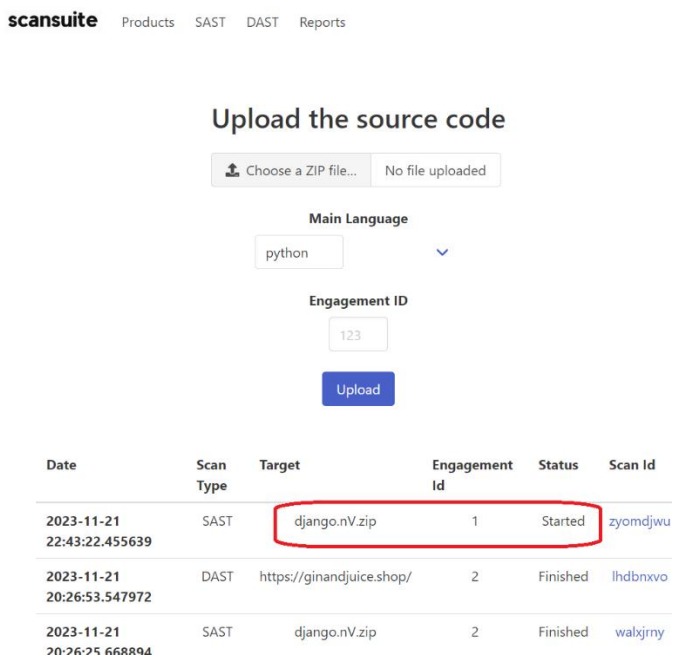
Finally, we are inside the engagement where the results of our Python scans will come to. Check the number in the URL (most likely it will be 1 or 2). That’s an Engagement ID you will need to provide during the scans:



Let's get back to ScanSuite and upload the django.nV.zip file. You should have it downloaded already during the prep stage. Go to the SAST page, select the file and specify Engagement ID. In my case it's 1:

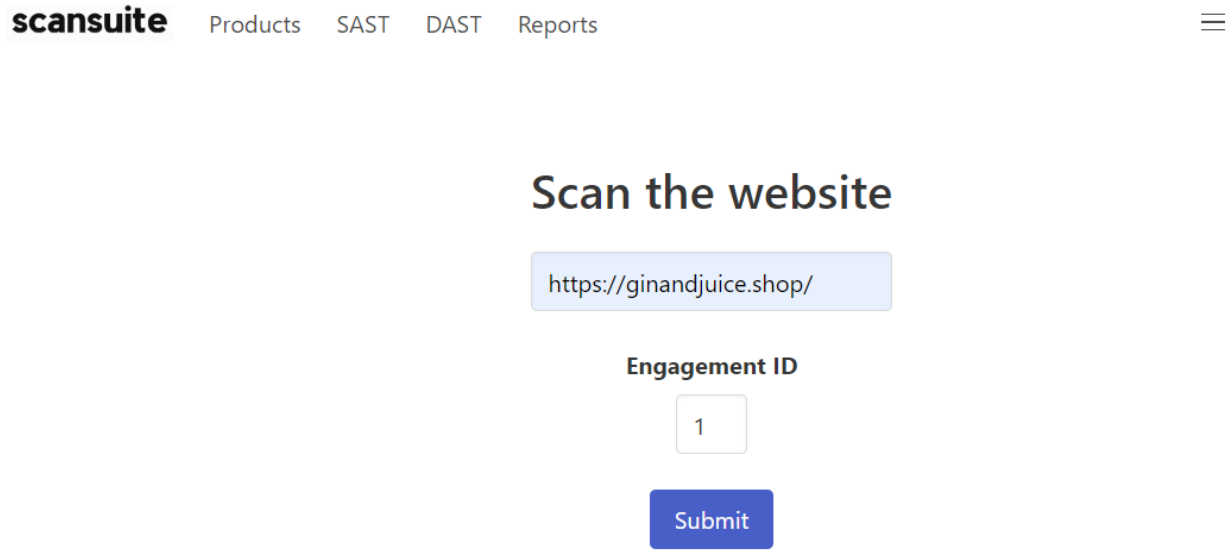


Click "Upload" and select the SAST tab again. You should see the scan was started:

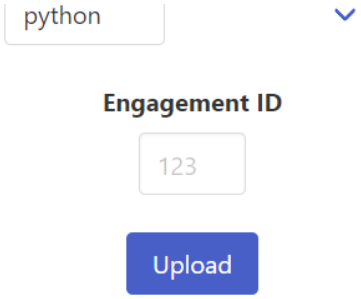


Next you may want to run a DAST scan for the ready website so see some typical server misconfigs or exposed files to the public, so select the DAST tab and execute the scan.

You can target the https://ginandjuice.shop/ site, which was purposely created for vulnerability scanning or you can scan the website, which belongs to me https://myappsec.eu/ providing the same Engagement Id as you did for SAST:



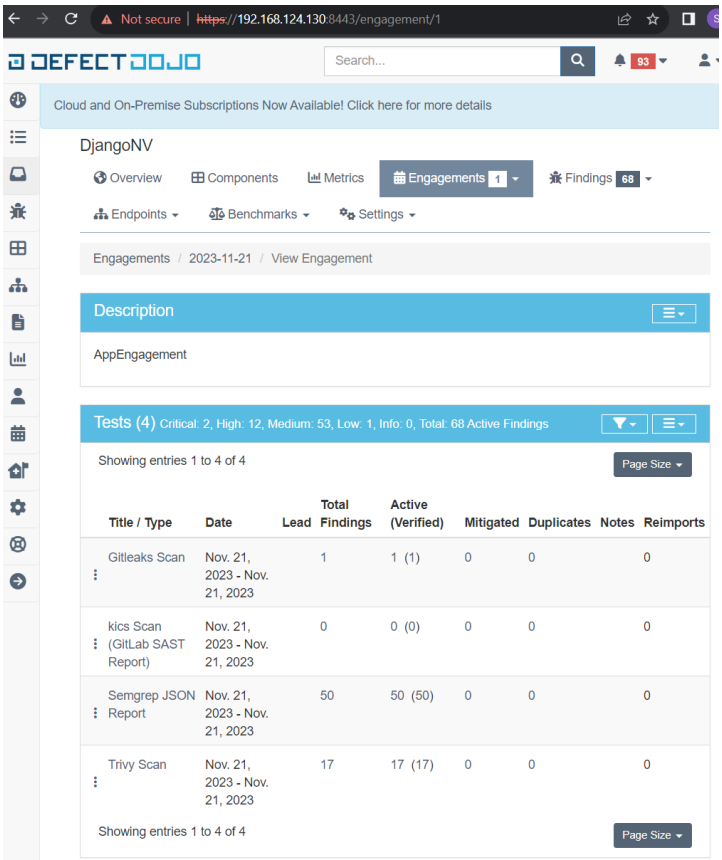
Coming back to SAST tab we see another scan has been started:



Date	Scan Type	Target	Engagement Id	Status	Scan Id
2023-11-22 21:07:11.694027	DAST	https://ginandjuice.shop/	1	Started	wedqjfkg
2023-11-21 22:43:22.455639	SAST	django.nV.zip	1	Finished	zyomdjwu

DAST scan is quite time consuming by its nature so for the purposes of this lab the quick / compliance ZAP scan is configured to run.

Let's get back to the DefectDojo and refresh the page. Scan results are coming:



Overall, that should be 4 SAST tests. The DAST one will come up later (10-20 mins depending on the website you scan and the Internet speed). Let's check the results by clicking the scan names or the corresponding active findings:

1. Secrets scan – GitLeaks Scan

   The scan identified secrets in the code and git history, which is a big headache for many orgs, especially the cloud-based ones. One leaked API key to the cloud resources may lead to a full compromise of the cloud project or complete organization. So, keep an eye on these things and scan the code for these issues as often as needed.

   In our case, however, this is truly a false positive, so we follow to the next scan.

2. Infrastructure as Code Scan (IACS) kics Scan

   A neat way to identify vulnerabilities in your infra even before it gets deployed. Checks Terraform, Ansible, Docker & Kubernetes etc configs which we don't have in this repo, moving to the next one.

3. SAST – Semgrep

   Here we see some issues in the code itself like XSS, SQLi, CSRF etc. Not a big deal compared to the number of issues the application has.

4. Software Composition Analysis (SCA) or just the dependency checks – Trivy Scan

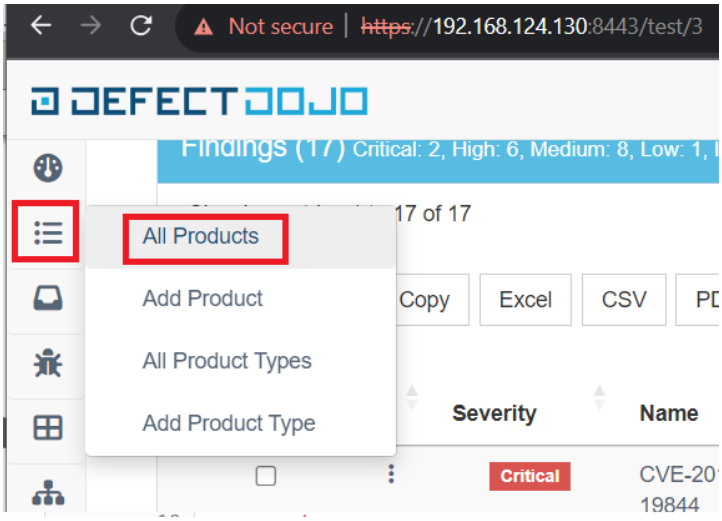
   The dependency scan shows us known CVEs for the libs and imports used by the project.

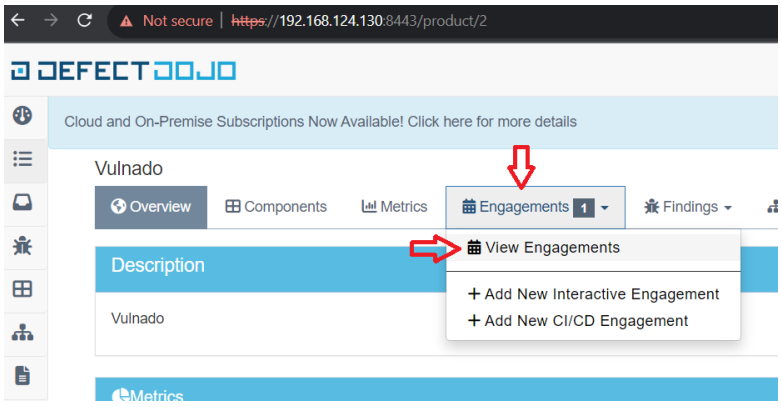
Hope you have enjoyed this warm up, let's move to the second lab.

Lab 2. CI/CD pipelines

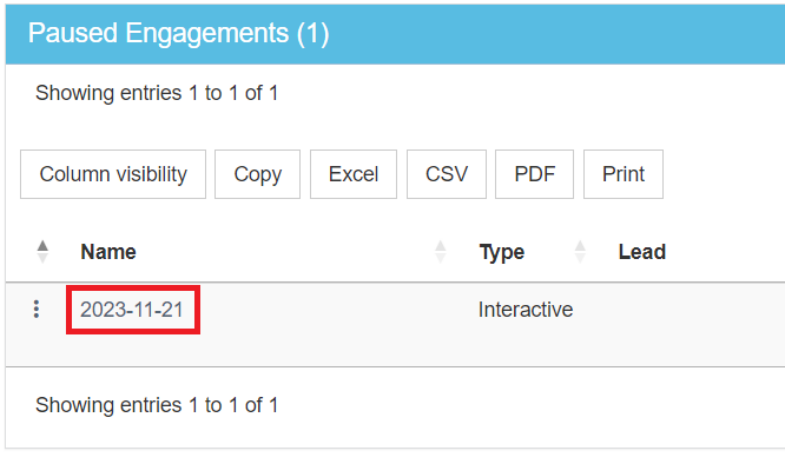
Your dev team uses the CI/CD rails to test and deploy the code which is great because you can integrate the scans right into the pipeline. Let's see how to do that.

First, we switch to Vulnado product in DefectDojo to locate its Engagement Id:

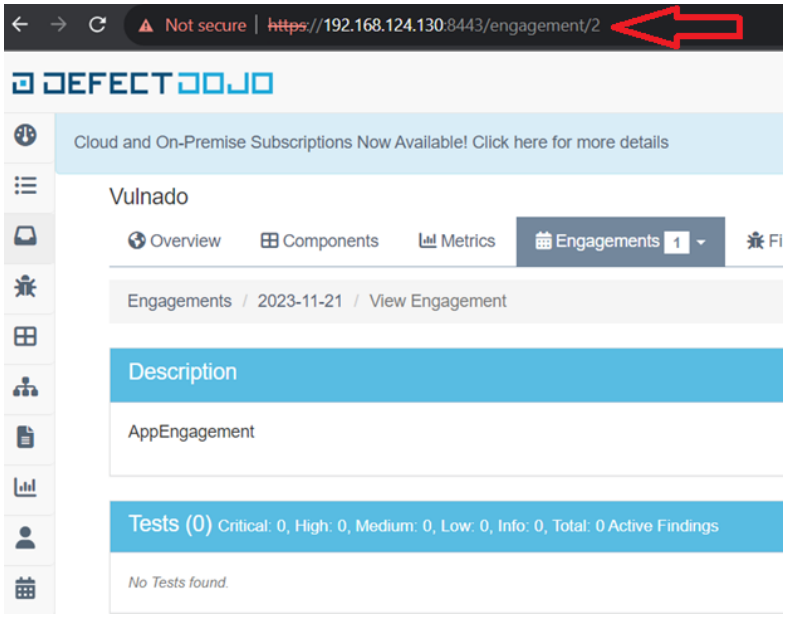




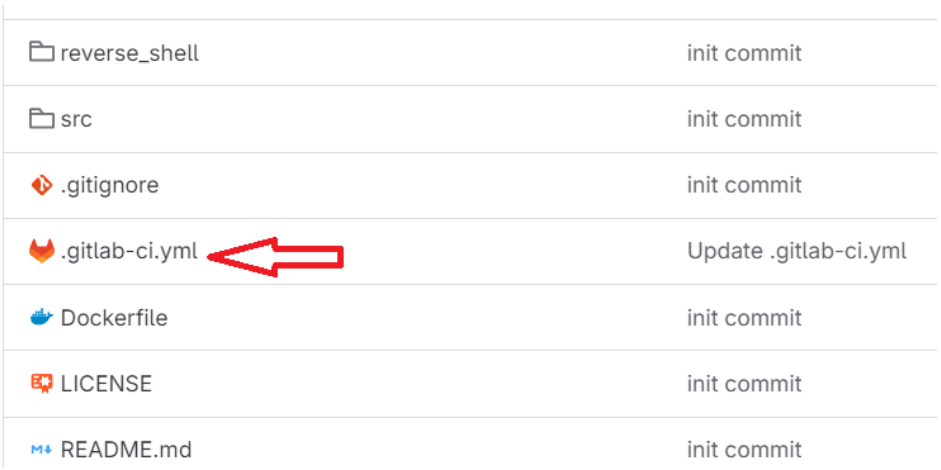
Click of the Engagement with the current or the most recent date:



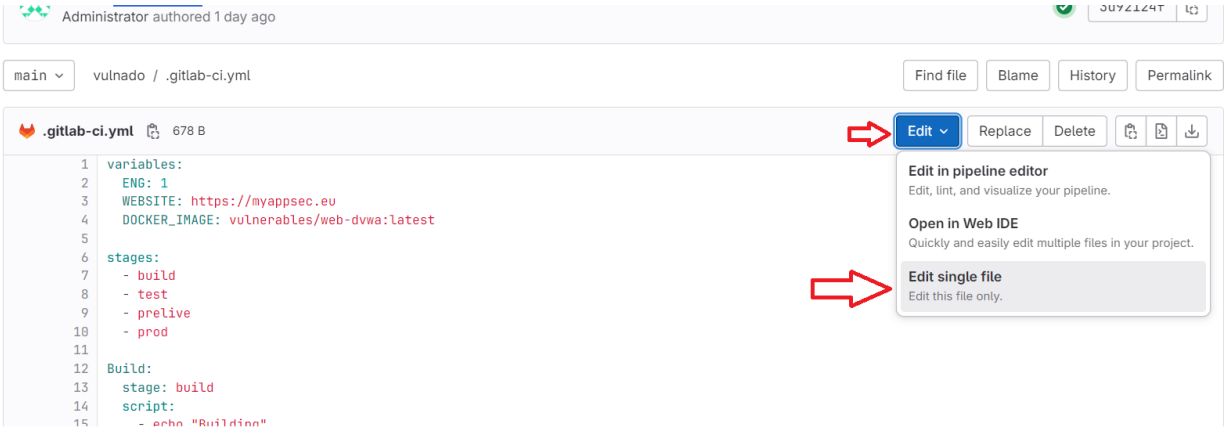
Check the number in the URL and note it down:



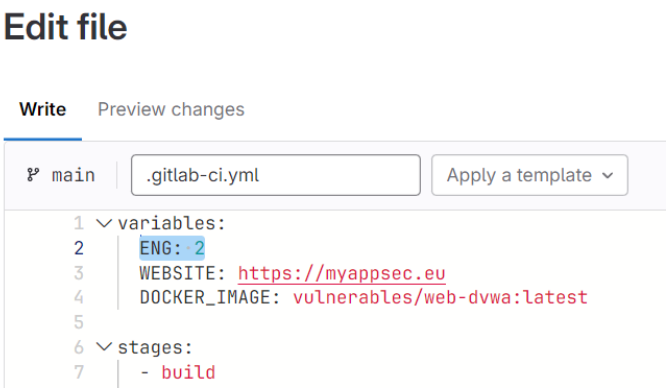
Open the https://YOUR_GITLAB_IP/root/vulnado page where you will find the vulnerable Java application code. Click on .gitlab-ci.yml file:



And then Edit – Edit single file:

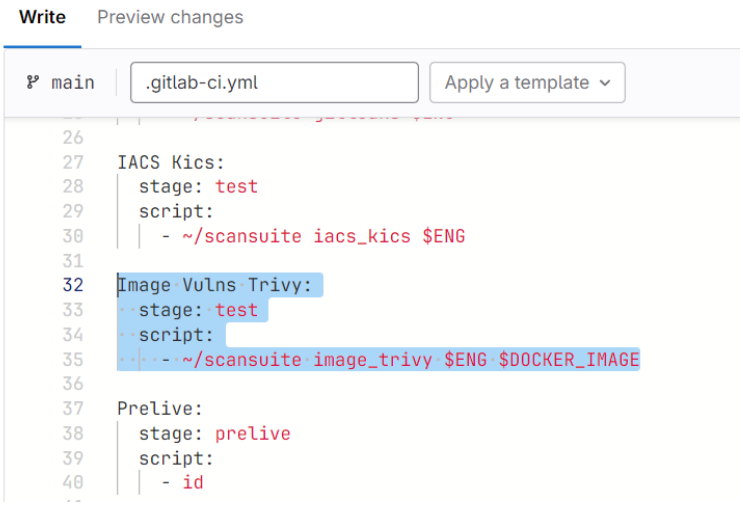


This is a yaml config file which describes the pipeline to be executed. GitLab is monitoring code commits to the git main/master branch and autoruns the pipeline described in this file. Contents of this file are easily readable. Let's change the ENG variable value to the Engagement Id we just got in DefectDojo, will be 2 in my case:

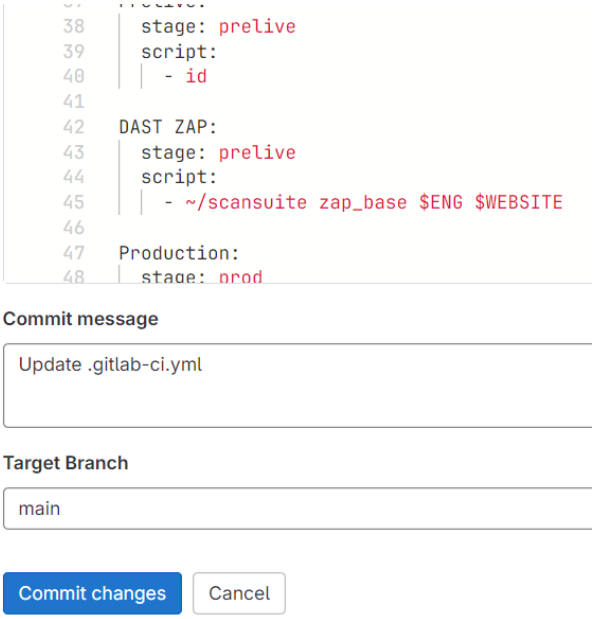


And add another test to the config right after the IACS Kics scan. That would be a docker image scan, just Ctrl+C – Ctrl+V the text below and paste into your web page:

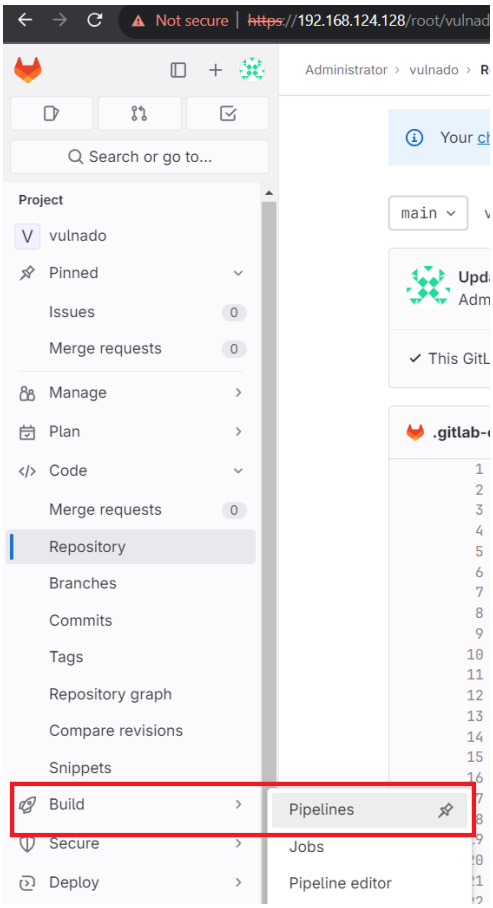
```
Image Vulns Trivy:
  stage: test
  script:
    - ~/scansuite image_trivy $ENG $DOCKER_IMAGE
```



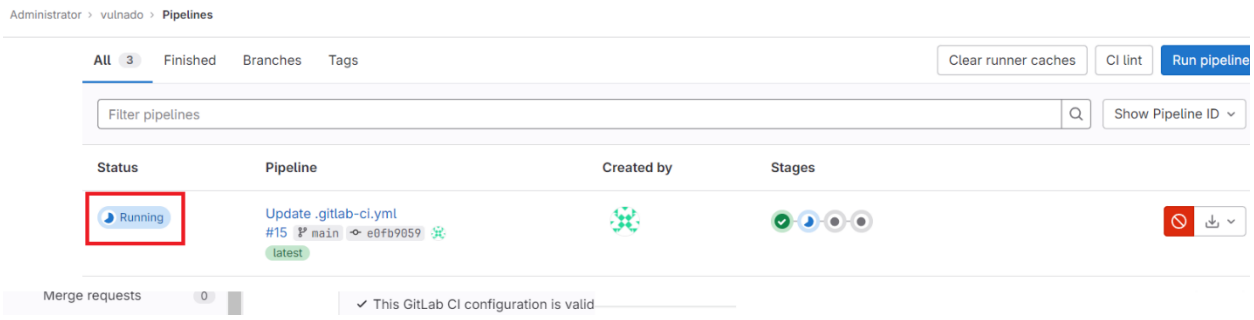
Then scroll down and commit the changes:



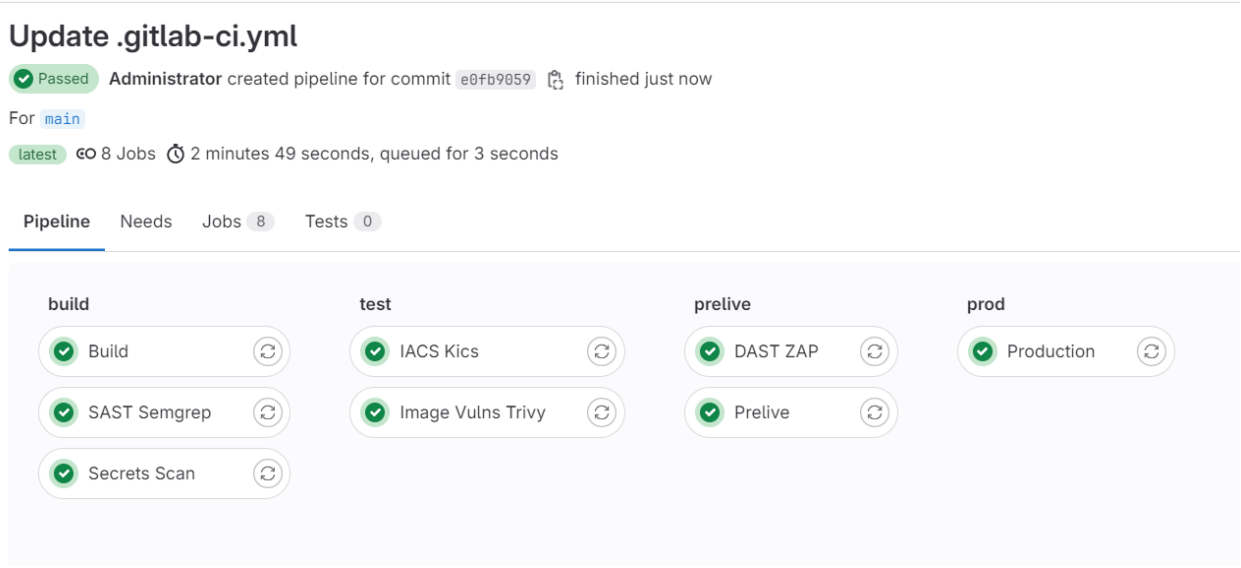
Now moving to the Build – Pipelines:



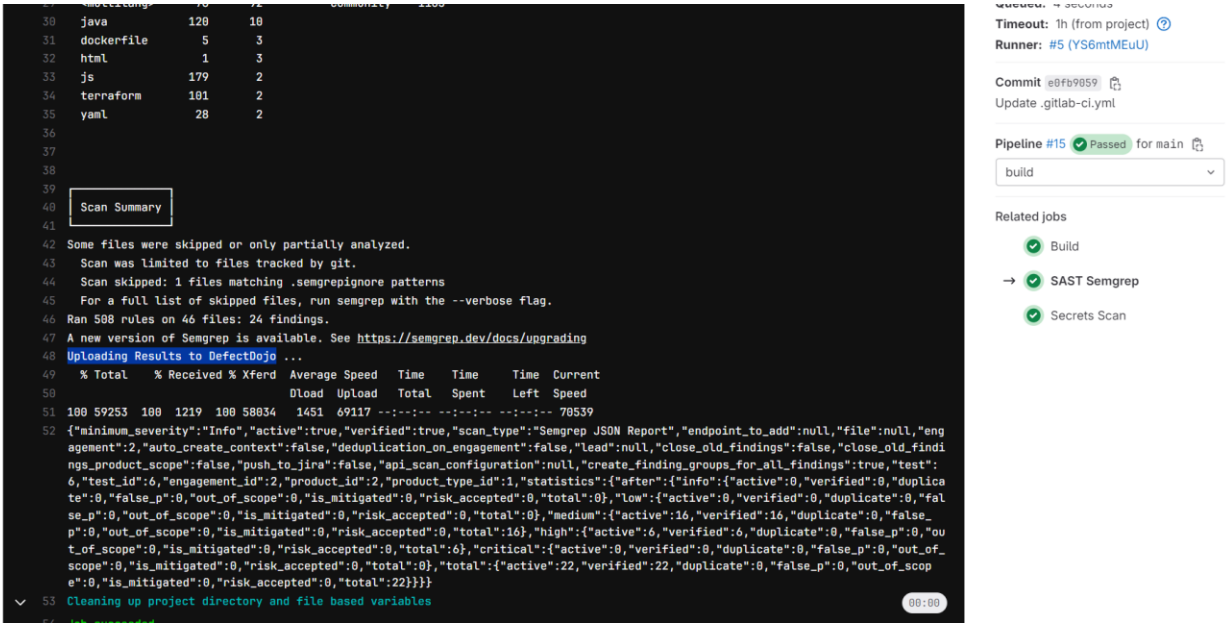
The pipeline is executing, click on the Running logo to see the details of it:



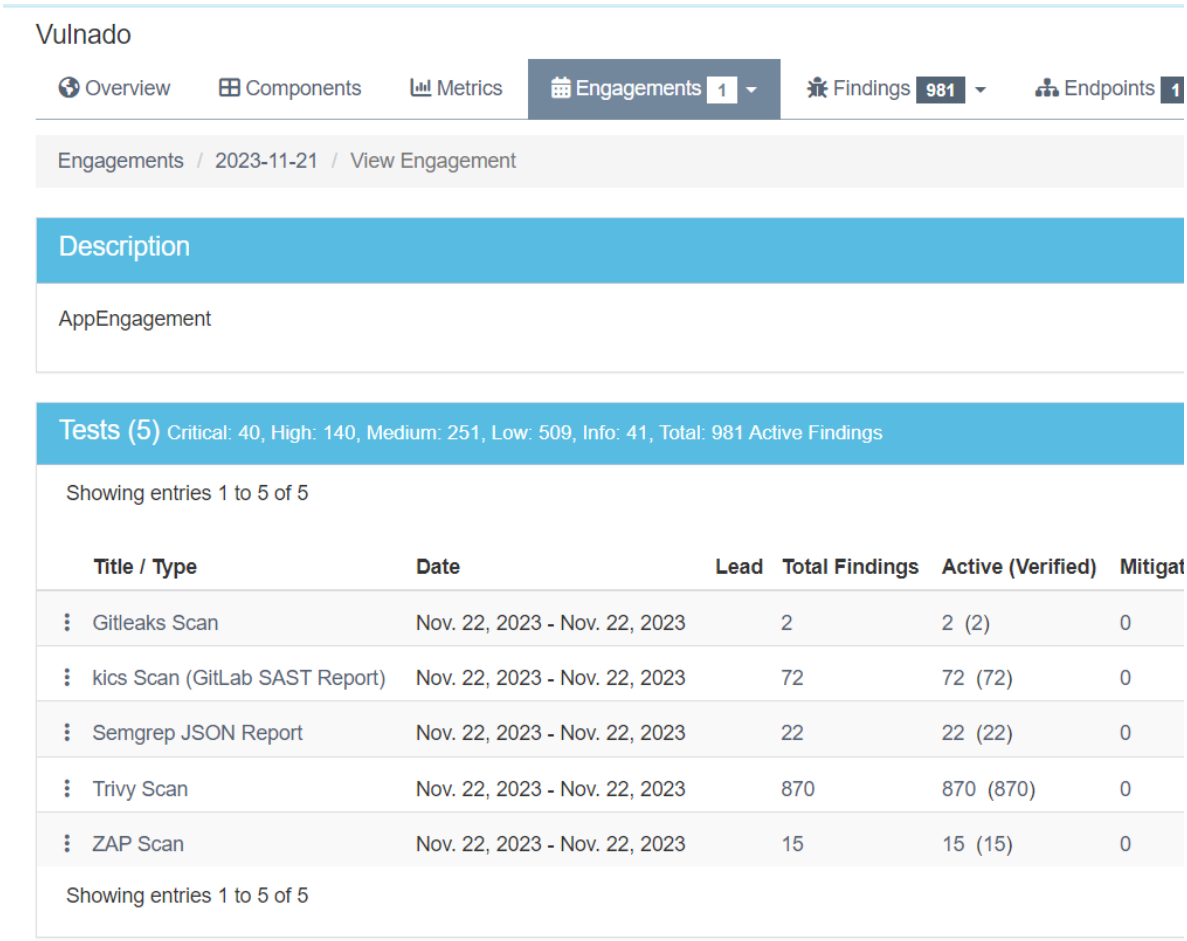
Here we have stages and tasks, executed by our Runner, installed on the scansuite VM host. If tasks are not picked by the runner, try to reboot the **scansuite** machine, hopefully it comes back.



Feel free to click on any of the tasks where you will see the terminal output of the executed command.



Let's check the results in DefectDojo. Here we have 5 scan results, both static and (eventually) dynamic:



Check the results and reach me if you have any questions. This time results should be more actionable with much less false positives.

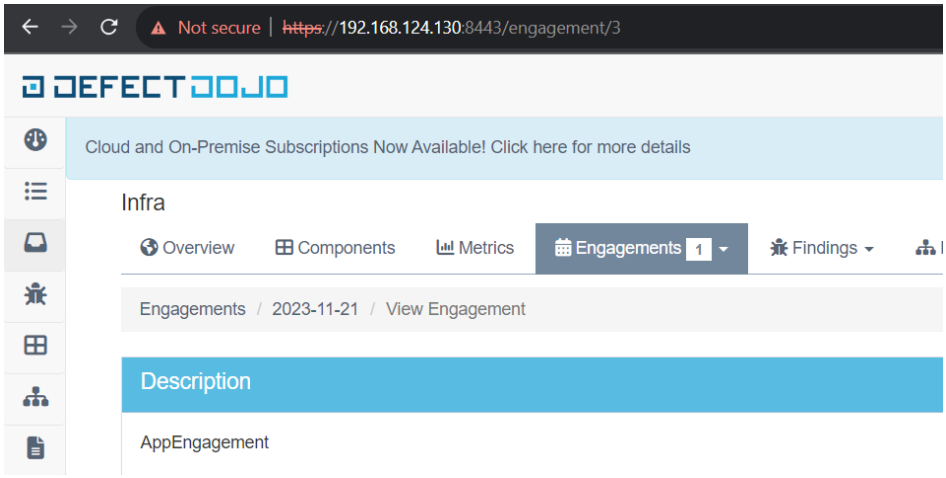
Worth noting, that Trivy Image scan brought 870+ findings as the examined Docker image “web-dvwa” is old and vulnerable, as it should be.

Let’s switch the gears and examine how we can check local systems for infrastructure vulnerabilities.

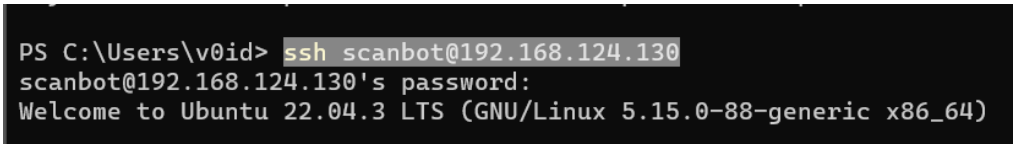
Lab 3 Infrastructure scanning

During the previous labs we have used IACS (Infra as Code Scanning) and Docker image scanning tools which allowed us to gather some environmental vulnerabilities in which applications are supposed to be executed.

This is great, but we can do better. Let’s investigate the local Docker setup deficiencies and how to analyze them using docker-bench-security tool. First, goto DefectDojo and locate the Engagement ID of Infra product. You should be already familiar how to do that:



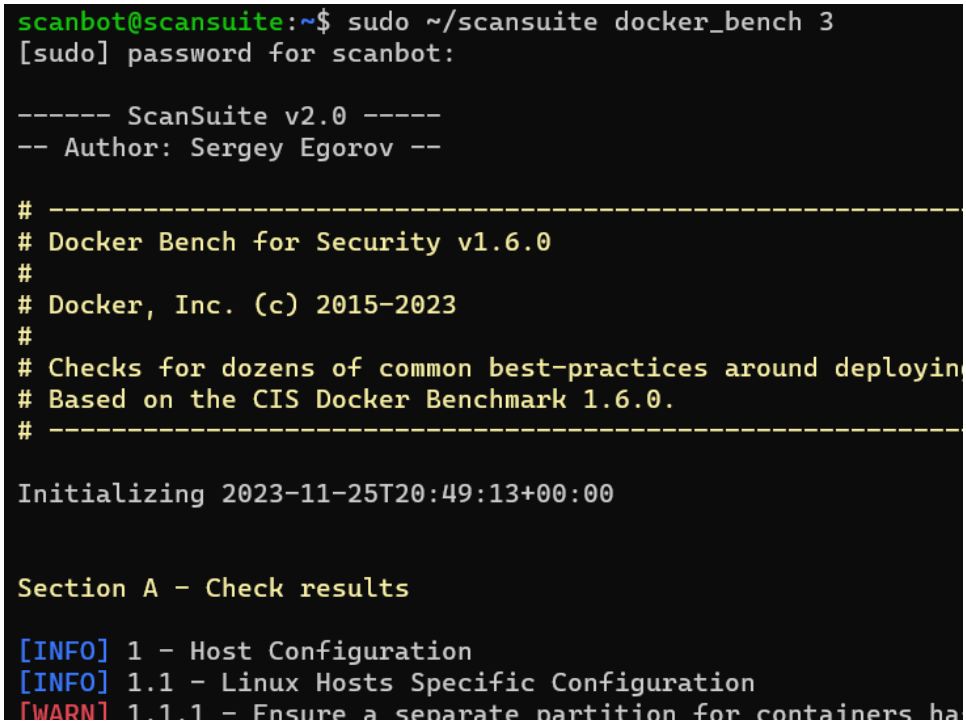
Now, login to the scansuite host over SSH (**login scanbot** with **password scanbot**) or via VMWare console:



Execute the following command:

sudo ~/scansuite docker_bench 3

Here 3 is my Engagement ID, yours might be different (but most likely will be the same)



The scan was executed, and report uploaded to DefectDojo.

Before analyzing the results let's run a couple of more local infra scans. One would be compliance check of the OS hardening to CIS Benchmarks. We will be using popular Open SCAP analyzer for it:

sudo ~/scansuite oscap_ubuntu 3 22

Again, 3 is an Engagement ID

```
scanbot@scansuite:~$ sudo ./scansuite oscap_ubuntu 3 22

------ ScanSuite v2.0 -----
-- Author: Sergey Egorov --

Title   Package "prelink" Must not be Installed
Rule    xccdf_org.ssgproject.content_rule_package_prelink_removed
Result  pass

Title   Install AIDE
Rule    xccdf_org.ssgproject.content_rule_package_aide_installed
Result  fail

Title   Build and Test AIDE Database
Rule    xccdf_org.ssgproject.content_rule_aide_build_database
Result  fail
```

The scan will take 3-5 minutes to complete, some checks will take a while. At the end the results will be uploaded to DefectDojo.

We can also check local packages for vulnerabilities using OVAL checks of the same Open SCAP tool. sudo is not required for these checks:

~/scansuite oscap_ubuntu_vuln

It will download the latest OVAL definitions from Ubuntu and check the local setup.

```
scanbot@scansuite:~$ ~/scansuite oscap_ubuntu_vuln

------ ScanSuite v2.0 -----
-- Author: Sergey Egorov --

--2023-11-25 19:02:58--  https://security-metadata.canonical.com/oval/com.ubuntu.jammy.usn.oval.xml.bz2
Resolving security-metadata.canonical.com (security-metadata.canonical.com)... 185.125.190.21, 185.125.190.29, 185.125.1
90.20, ...
Connecting to security-metadata.canonical.com (security-metadata.canonical.com)|185.125.190.21|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 263942 (258K) [application/x-bzip2]
Saving to: 'com.ubuntu.jammy.usn.oval.xml.bz2'

com.ubuntu.jammy.usn.oval.xml 100%[=================================================>] 257.76K  --.-KB/s    in 0.07s

2023-11-25 19:02:58 (3.60 MB/s) - 'com.ubuntu.jammy.usn.oval.xml.bz2' saved [263942/263942]

Definition oval:com.ubuntu.jammy:def:981000000: false
Definition oval:com.ubuntu.jammy:def:971000000: false
Definition oval:com.ubuntu.jammy:def:961000000: false
Definition oval:com.ubuntu.jammy:def:951000000: false
```

Results will not be uploaded to DefectDojo due to the absence of parser and will be saved to **oscap_scansuite_vuln.html** in the same folder. Transfer the file to your local machine and open it:

```
PS C:\Users\v0id> scp scanbot@192.168.124.130:oscap_scansuite_vuln.html .
scanbot@192.168.124.130's password:
oscap_scansuite_vuln.html                                     100%  653KB  47.7MB/s   00:00
```

Some packages need to be patched:

ID	Result	Class	Reference ID	Title
oval:com.ubuntu.jammy:def:65051000000	true	patch	[USN-6505-1], [CVE-2023-44487]	USN-6505-1 -- nghttp2 vulnerability
oval:com.ubuntu.jammy:def:64961000000	true	patch	[USN-6496-1], [CVE-2023-25775], [CVE-2023-31085], [CVE-2023-45871]	USN-6496-1 -- Linux kernel vulnerabilities
oval:com.ubuntu.jammy:def:100	true	inventory		Check that Ubuntu 22.04 LTS (jammy) is installed.
oval:com.ubuntu.jammy:def:981000000	false	patch	[LSN-0098-1], [CVE-2023-3776], [CVE-2023-3609], [CVE-2023-21400], [CVE-2023-4004], [CVE-2023-3777], [CVE-2023-40283], [CVE-2023-3090], [CVE-2023-3995], [CVE-2023-3567]	LSN-0098-1 -- Kernel Live Patch Security Notice

Switch to the DefectDojo and see how we can do better in Docker and OS hardening:

Tests (2) Critical: 0, High: 37, Medium: 90, Low: 22, Info: 7, Total: 156 Active Findings

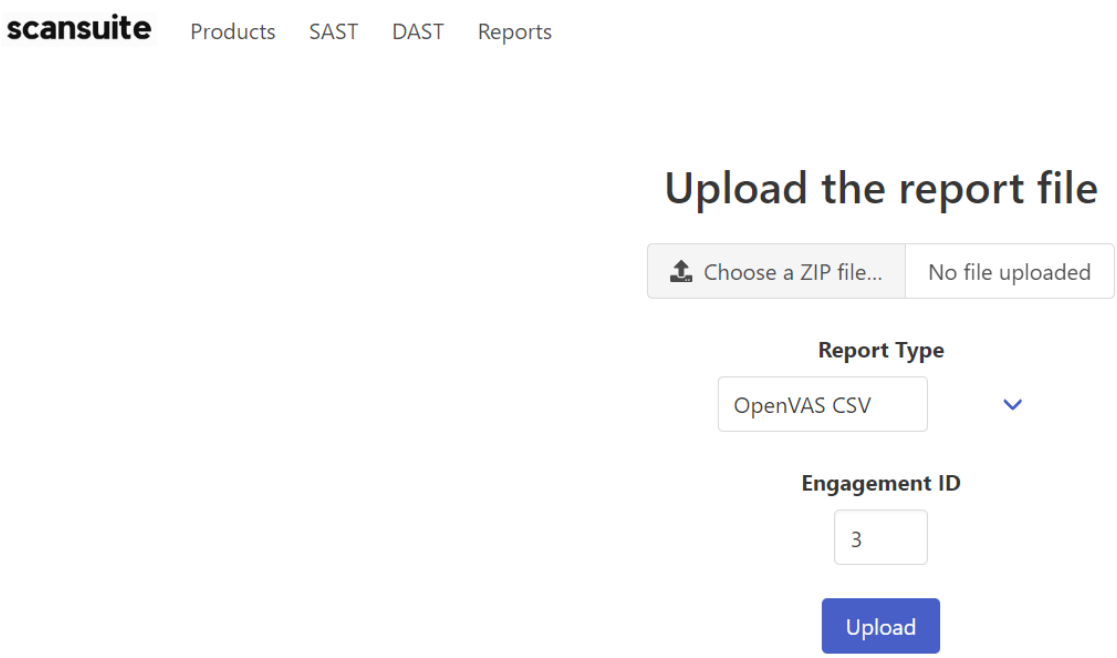
Showing entries 1 to 2 of 2 Page Size

Title / Type	Date	Lead	Total Findings	Active (Verified)	Mitigated	Duplicates	Notes	Reimports
docker-bench-security Scan	Nov. 25, 2023 - Nov. 25, 2023		54	54 (54)	0	0		0
Openscap Vulnerability Scan	Nov. 25, 2023 - Nov. 25, 2023		102	102 (102)	0	0		0

Showing entries 1 to 2 of 2 Page Size

In this lab we did not investigate the remote infrastructure scanning, which should be a mandatory part of any vulnerability management program. This is due to most tools in this area are commercial, and OpenVAS is pretty resource consuming to set for these labs.

Feel free to set it up in your spare time (using for example this guide) and upload the results using the **Reports** tab of ScanSuite interface so you can aggregate such findings in DefectDojo too:



3. Wrap-up

This is it, hope you enjoyed this intro to vulnerability scanning. Obviously, there are more scanners one can use to get better results for analyzing specific language or framework. You can check the complete list of scanners I am using along with other ScanSuite functionality here.

Try to clone some repos to **scansuite** host and run these scanners using the syntax suggested in Readme. See if you can identify new vulnerabilities in open source projects.

Please reach out to me if you have any questions about the lab walkthrough, the DevSecOps topic or would like to share the worthwhile scanners I am missing. Also, if you want to collaborate on ScanSuite or use it for your commercial projects. Will be happy to learn from your experience too.

Otherwise enjoy your days at the conference and good luck with battling vulnerabilities!

My contacts:

Sergey Egorov

cepxeo@gmail.com

Threema STR295ER