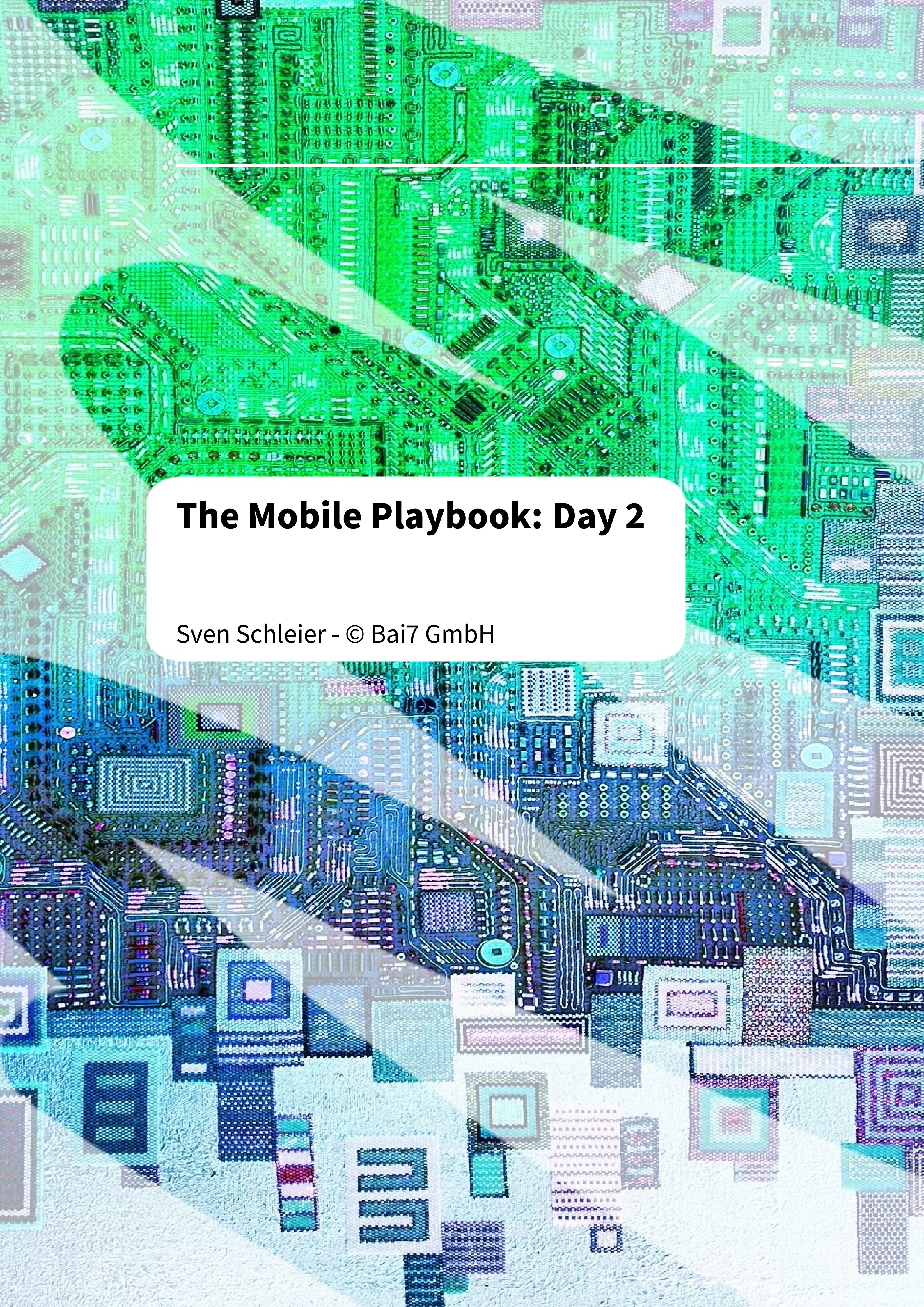


The Mobile Playbook: Day 2

Sven Schleier - © Bai7 GmbH

Contents

Introduction - WIP

All information needed for the labs is located in the VM in the directories listed below.

WIP

Lab - Fork and setup Repo

Time to finish lab	5 minutes
App used for this exercise	None

Training Objectives

In the following exercise we will fork a Github repository that we will use for some of the Android exercises.

Preparation

Github

- Fork the following repo: https://github.com/bai7-at/mobile-playbook-dev-ios

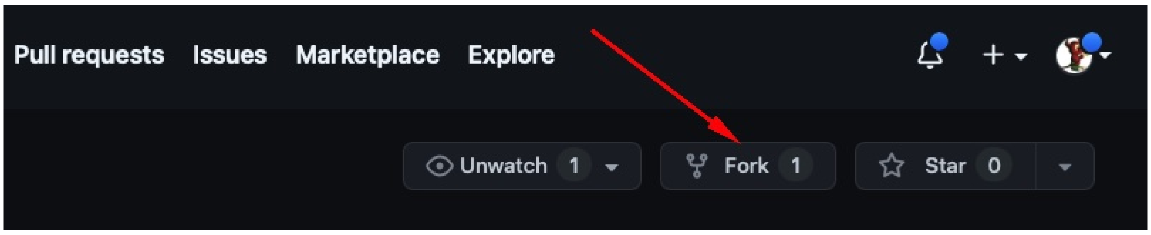
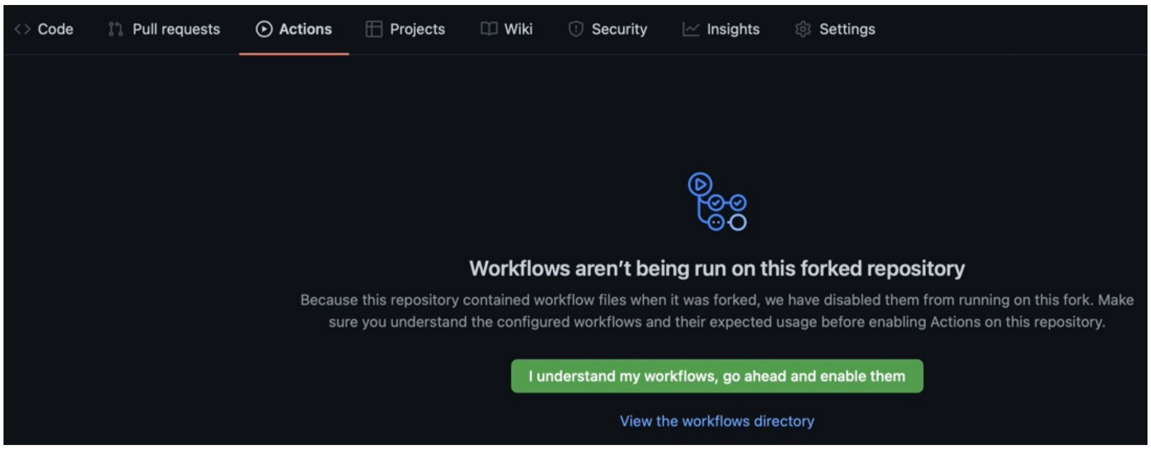


Figure 1: Fork repo

- In the next screen simply click "Create fork" and it will be forked to your Github account.
- Go to "Actions" in your newly forked repo and click on the green button to enable Github workflows.

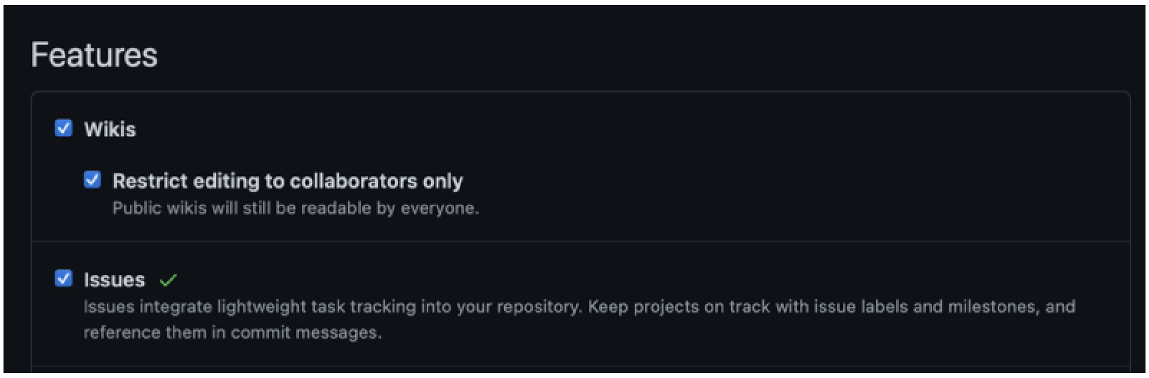

Code
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
I understand my workflows, go ahead and enable them
View the workflows directory

Workflows aren't being run on this forked repository

Because this repository contained workflow files when it was forked, we have disabled them from running on this fork. Make sure you understand the configured workflows and their expected usage before enabling Actions on this repository.

Figure 2: Enable Workflows

- Enable “Issues” in your forked repo. You can find it in the “Settings” tab under “Features”.



Features

- [x] Wikis
  - [x] Restrict editing to collaborators only
    Public wikis will still be readable by everyone.
- [x] Issues ✓
  Issues integrate lightweight task tracking into your repository. Keep projects on track with issue labels and milestones, and reference them in commit messages.

Figure 3: Enable Issues

Congratulations, you forked the repo! Later we will be activating some Github Actions to execute automated security checks.

Lab - Static Scanning with mobsfscan

Time to finish lab	20 minutes
App used for this exercise	

Training Objectives

In this lab we will be scanning the decompiled code base of two Android apps for vulnerabilities and misconfiguration with `semgrep`.

Tools used in this section

- `semgrep` - https://github.com/semgrep/semgrep

Execute the semgrep scan - Finstergram

The app `Finstergram` was already decompiled for you with `jadx`.

Execute the scan with `semgrep` in the directory of the decompiled Finstergram app and use customized semgrep rules:

```
# Either type the command, or copy the first row and then the
   second, as these need to be executed together.
$ cd ~/Mobile-Playbook/Android/Apps/Finstergram/sources/com/
   netlight/sec/finstergram

# Either type the command, or copy the first row and then the
   second, as these need to be executed together.
$ semgrep -c ../../../../../../../semgrep-rules-android-security/
   rules . --sarif-output=finstergram.sarif --sarif
```

> The original semgrep rules are build by MindedSecurity, see https://github.com/mindedsecurity/semgrep-rules-android-security, but were adjusted by me for this training in a fork that we are using https://github.com/sushi2k/semgrep-rules-android-security

Open the current directory in `VS Code` and execute the command in the terminal where you executed the semgrep scan:

```
# by using the "." the current directory will be opened
$ code --new-window . &
```

Click on the file `finstergram.sarif` and you will see that a new Window opens on the right that summarizes the results.

In case the SARIF window on the right doesn't open, go to "View/Command Palette" and type in `sarif.showpanel` and click enter. In the new window open the SARIF file by selecting the path and the file.

You can close the side panel with the folder structure and files on the left side by pressing CTRL+B, to have more space for the analysis. In the screen on the right, click on "Rules", select the first rule and click on the first entry. Once clicked you will be navigated to the decompiled source code on the left in a new window and the identified vulnerability.

Now the work starts! Your job is to triage the identified findings, go through them one by one and decide if they are either a:

- valid finding,
- false positive,
- if it shouldn't be fixed (and why).

Go through each finding and make a decision!

Execute the semgrep scan - Kotlin-Shack

The app `Kotlin-Shack` was already decompiled for you with `jadx`.

Execute the scan with `semgrep` and customized rules:

```
# Either type the command, or copy the first row and then the
   second, as these need to be executed together.
$ cd ~/Mobile-Playbook/Android/Apps/Kotlin-Shack/sources/org/owasp/
   mstgkotlin/

# Either type the command, or copy the first row and then the
   second, as these need to be executed together.
$ semgrep -c ../../../../../../semgrep-rules-android-security/rules
    .  --sarif-output=kotlin-shack.sarif --sarif
```

Open the current directory in `VS Code` and execute the command in the terminal where you executed the semgrep scan:

```
# by using the "." the current directory will be opened
$ code --new-window . &
```

- Click on the file `kotlin-shack.sarif` and you will see that a new Window opens on the right that summarizes the results.
- Now the work starts! Your job is to triage the identified findings, go through them one by one and decide if they are either a:
  - valid finding,
  - false positive,

    - if it shouldn’t be fixed (and why).

Go through each finding and make a decision!

Resources

- Mobile App Finstergram - https://github.com/netlight/finstergram

Lab - Software Composition Analysis

Time to finish lab	15 minutes
App used for this exercise	Tasky

Training Objectives

Learn how to scan Android dependencies with the tool `dependency-check` by using Github Actions.

Tools used in this section

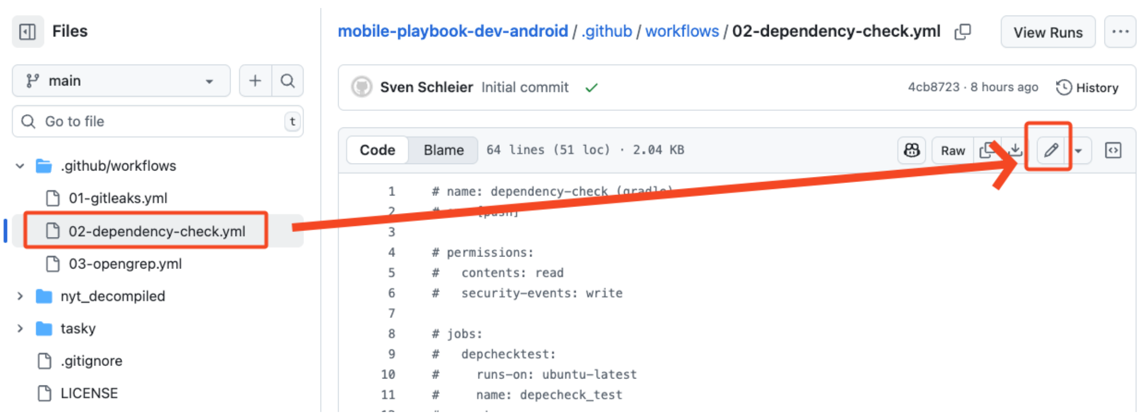
- `dependency-check` - https://jeremylong.github.io/DependencyCheck/
- `dependency-check` Gradle Plugin - https://jeremylong.github.io/DependencyCheck/dependency-check-gradle/index.html

Description

In this lab we will be scanning Android dependencies in each commit and Pull Request for known vulnerabilities with the `dependency-check` Gradle Plugin.

Preparation

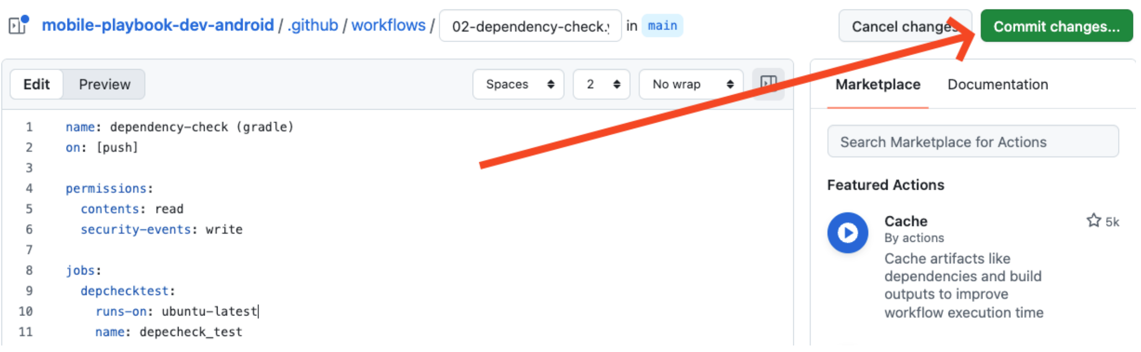
- Go to your forked repo and to the main directory in your browser. Select the `.github/workflows` folder and the file `02-dependency-check.yml` and edit it. and uncomment everything by selecting the whole text and pressing CTRL + "/" and commit your changes.



```
# name: dependency-check (gradle)
# on: [push]

# permissions:
#   contents: read
#   security-events: write

# jobs:
#   depchecktest:
#     runs-on: ubuntu-latest
#     name: depecheck_test
```

Figure 4: Edit Yaml file

- Uncomment everything by selecting the whole text and pressing "CTRL" + "/" (Windows) or "Command" + "/" (macOS):

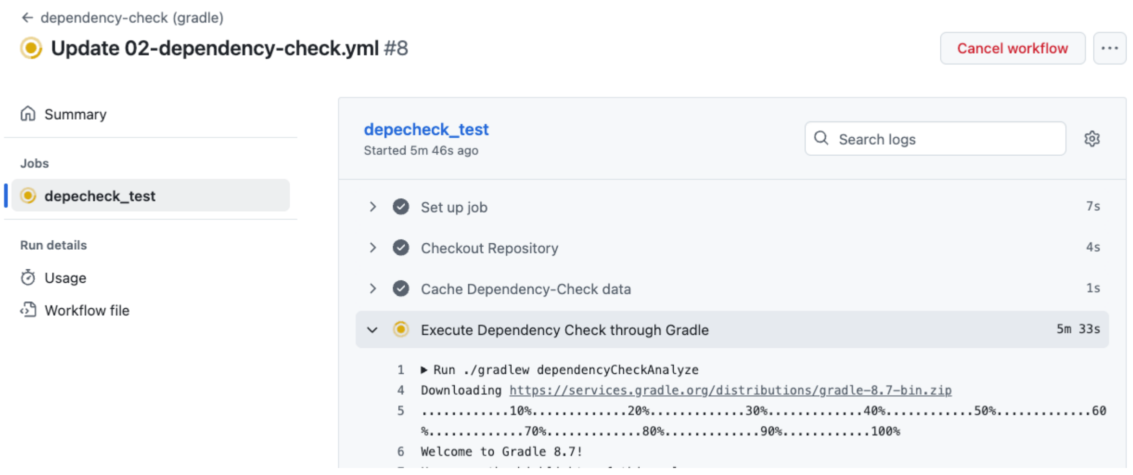

mobile-playbook-dev-android / .github / workflows / 02-dependency-check.y in main
Cancel changes
Commit changes...
Edit
Preview
Spaces
2
No wrap
Marketplace
Documentation
Search Marketplace for Actions
Featured Actions
Cache
By actions
Cache artifacts like dependencies and build outputs to improve workflow execution time
5k

```
name: dependency-check (gradle)
on: [push]

permissions:
  contents: read
  security-events: write

jobs:
  depchecktest:
    runs-on: ubuntu-latest
    name: depecheck_test
```

Figure 5: Commit changes

- In your forked repo click on "Actions" and you will see the workflows that were just triggered due to the commit. Click on the "dependency-check (gradle)" workflow run. The scan might still be running. Once the job is done you will see the "Depcheck report" at the bottom as new artifact.

> The first scan can take over 10 minutes, as dependency check need to build up a vulnerability database and download data. This scan was executed already earlier when you did the first lab to speed up the process. In consecutive runs like now, this database is cached in your fork and the workflow will take around 2-3 minutes.



```
▶ Run ./gradlew dependencyCheckAnalyze
Downloading https://services.gradle.org/distributions/gradle-8.7-bin.zip
............10%.............20%.............30%.............40%............50%.............60
%............70%.............80%.............90%............100%
Welcome to Gradle 8.7!
```

Figure 6: Worklow run

- Go to the "Security" tab in your repo and click on "Code scanning" on the left side. New vulnerabilities could be identified by `dependency check` and are all automatically imported into the code scanning alerts. Filter the new vulnerabilities by selecting the tool `dependency-check`:

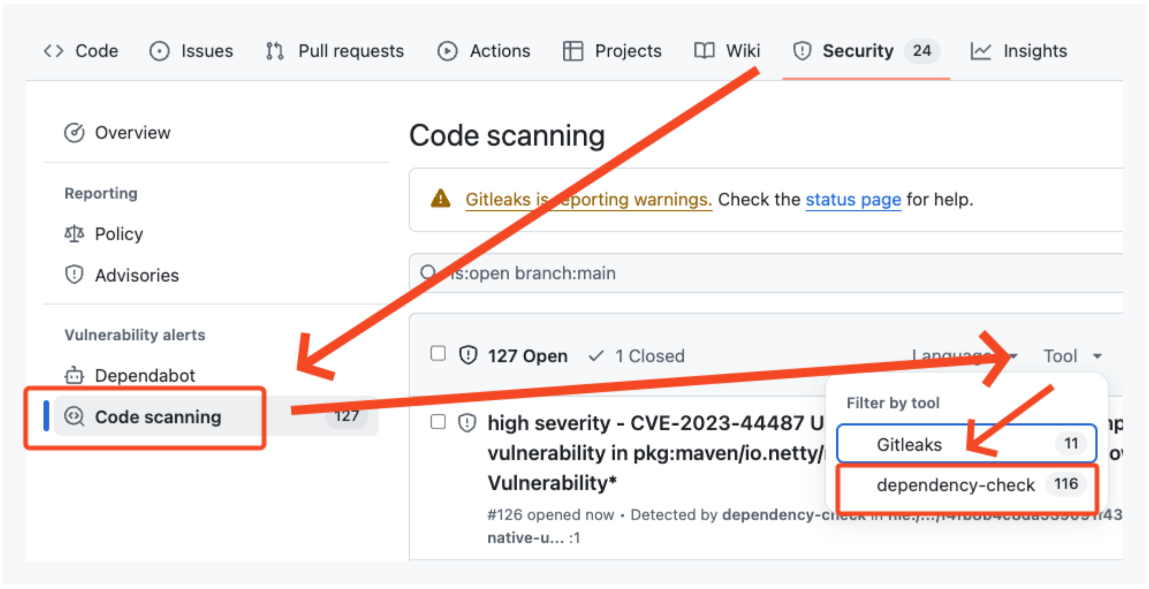
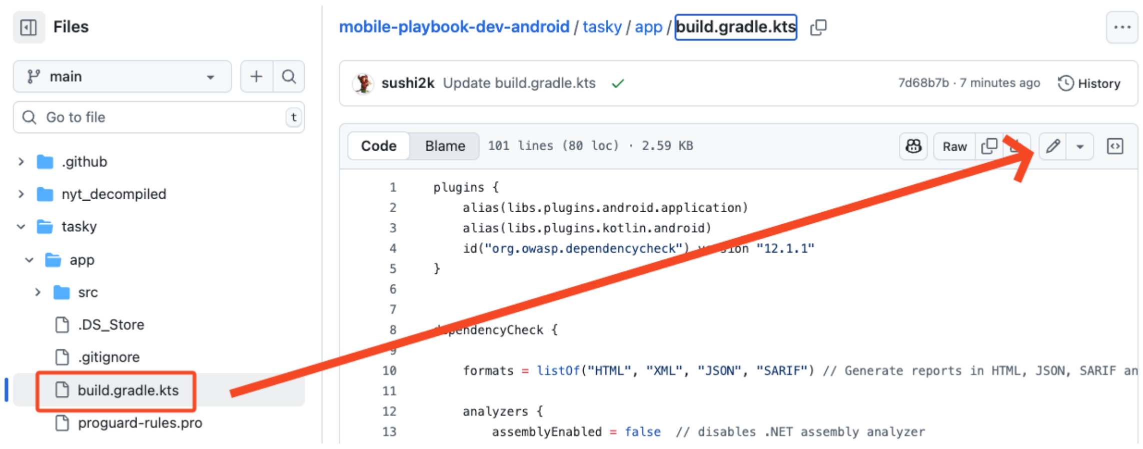


Figure 7: Findings

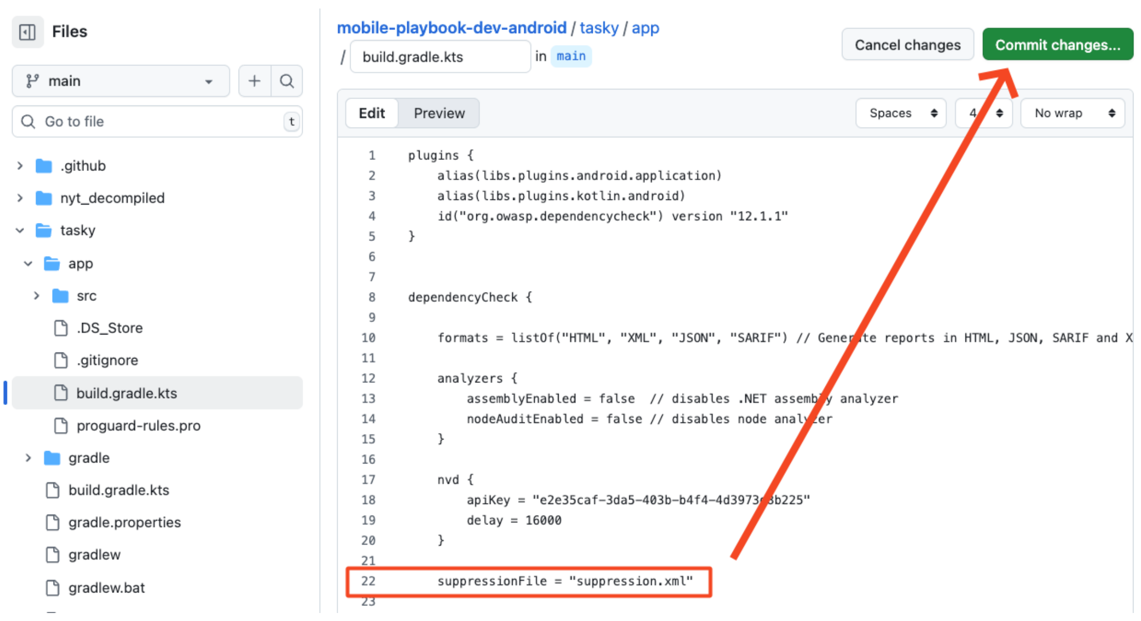
- We have now a lot of false positives through `netty` or `grpc` libraries. Let's include the file `suppresssion.xml` for `dependency-check` to remove these false positives from our scan result. Open the file `tasky/app/build.gradle.kts` and edit it:



```
plugins {
    alias(libs.plugins.android.application)
    alias(libs.plugins.kotlin.android)
    id("org.owasp.dependencycheck") version "12.1.1"
}


dependencyCheck {

    formats = listOf("HTML", "XML", "JSON", "SARIF") // Generate reports in HTML, JSON, SARIF an

    analyzers {
        assemblyEnabled = false  // disables .NET assembly analyzer
```

Figure 8: Edit build.gradle.kts

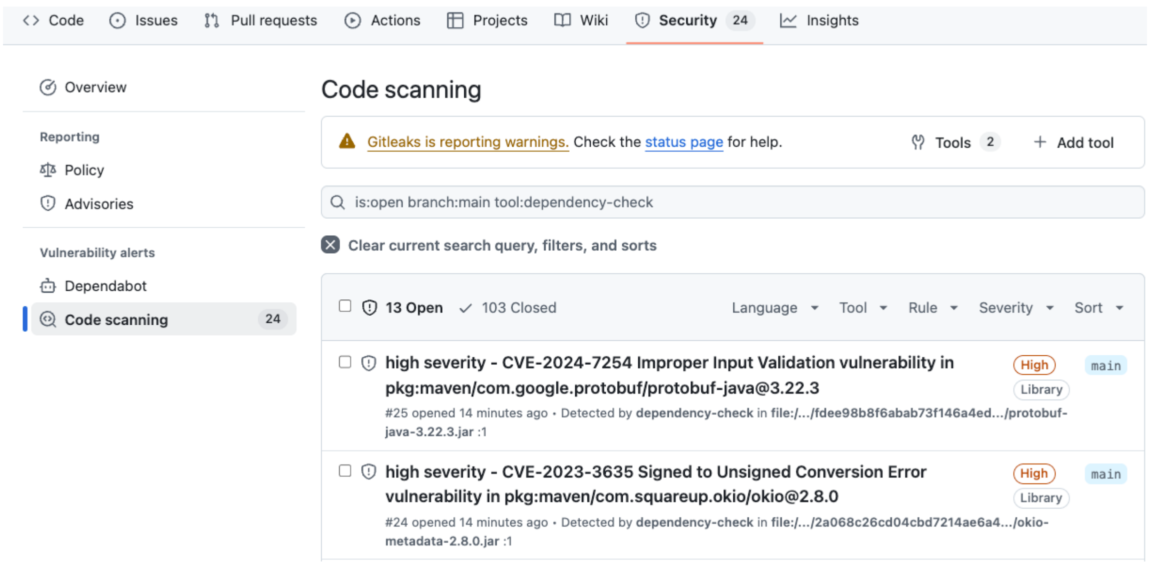
- Remove the comments in line 21 to include the `suppressionFile`.



```
plugins {
    alias(libs.plugins.android.application)
    alias(libs.plugins.kotlin.android)
    id("org.owasp.dependencycheck") version "12.1.1"
}


dependencyCheck {

    formats = listOf("HTML", "XML", "JSON", "SARIF") // Generate reports in HTML, JSON, SARIF and X

    analyzers {
        assemblyEnabled = false  // disables .NET assembly analyzer
        nodeAuditEnabled = false // disables node analyzer
    }

    nvd {
        apiKey = "e2e35caf-3da5-403b-b4f4-4d3973[illegible]3b225"
        delay = 16000
    }

    suppressionFile = "suppression.xml"
```

Figure 9: Include suppression.xml

- The Github Action is triggered again and the dependency check scan should be done in a few minutes. Now all the false positives are automatically removed. Select the tool `dependency-check` and you will have 13 open findings.



Gitleaks is reporting warnings. Check the status page for help.

is:open branch:main tool:dependency-check

13 Open ✓ 103 Closed

high severity - CVE-2024-7254 Improper Input Validation vulnerability in pkg:maven/com.google.protobuf/protobuf-java@3.22.3 — High — Library — main
#25 opened 14 minutes ago • Detected by dependency-check in file:/.../fdee98b8f6abab73f146a4ed.../protobuf-java-3.22.3.jar :1

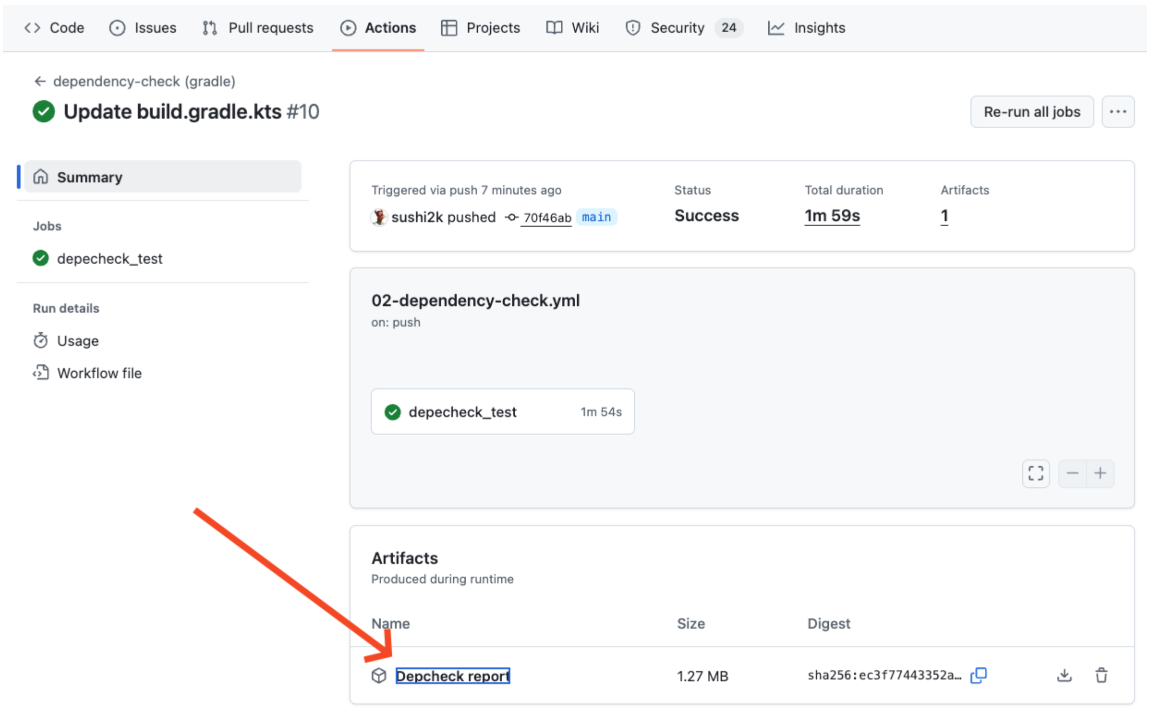
high severity - CVE-2023-3635 Signed to Unsigned Conversion Error vulnerability in pkg:maven/com.squareup.okio/okio@2.8.0 — High — Library — main
#24 opened 14 minutes ago • Detected by dependency-check in file:/.../2a068c26cd04cbd7214ae6a4.../okio-metadata-2.8.0.jar :1

Figure 10: Include suppression.xml

- Your job is to triage the identified findings and decide if they are a:
  - valid finding,
  - false positive,
  - used in tests, or
  - if you won't fix it (and why).

Go through each finding and make a decision!

You can look into the HTML report that is being generated for each workflow run.



dependency-check (gradle)

Update build.gradle.kts #10

Jobs

depecheck_test

Run details

Triggered via push 7 minutes ago

sushi2k pushed 70f46ab main

Status

Success

Total duration

1m 59s

Artifacts

1

02-dependency-check.yml

on: push

depecheck_test 1m 54s

Artifacts

Produced during runtime

Name	Size	Digest
Depcheck report	1.27 MB	sha256:ec3f77443352a…

Figure 11: Include suppression.xml

Congratulations, you are now able to scan, detect and triage known vulnerabilities in your libraries and manage them with code scanning alerts!