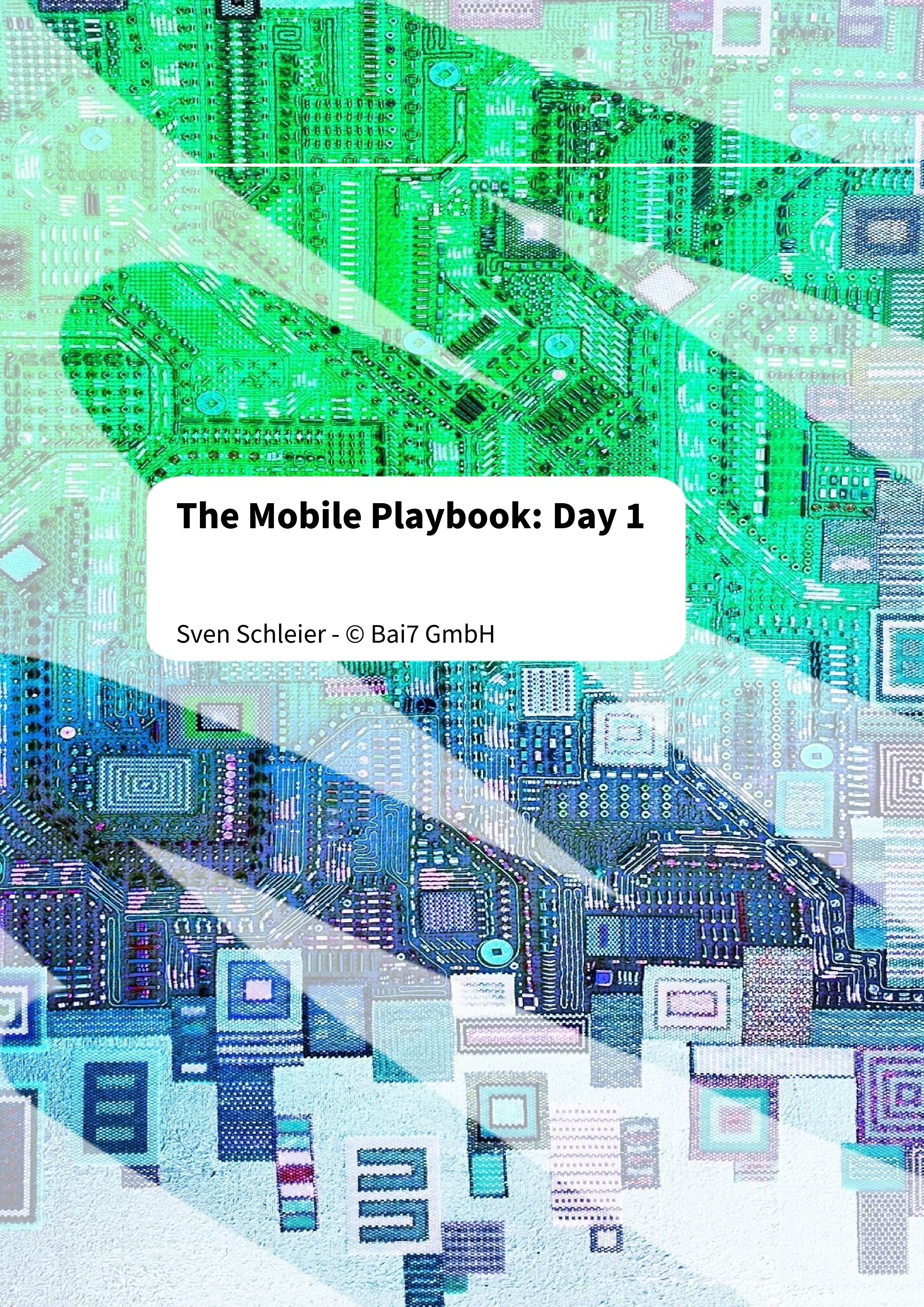


The Mobile Playbook: Day 1

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-android

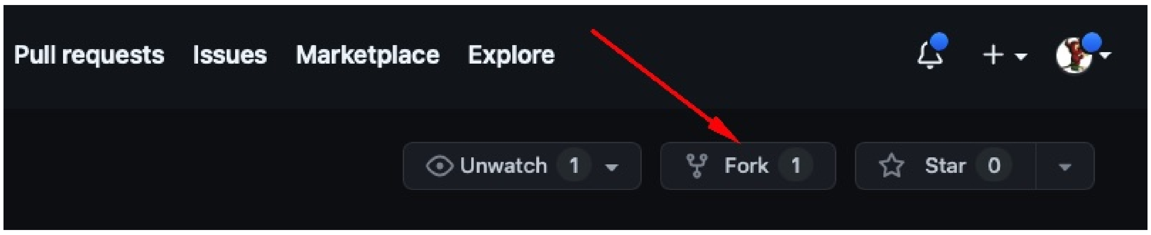
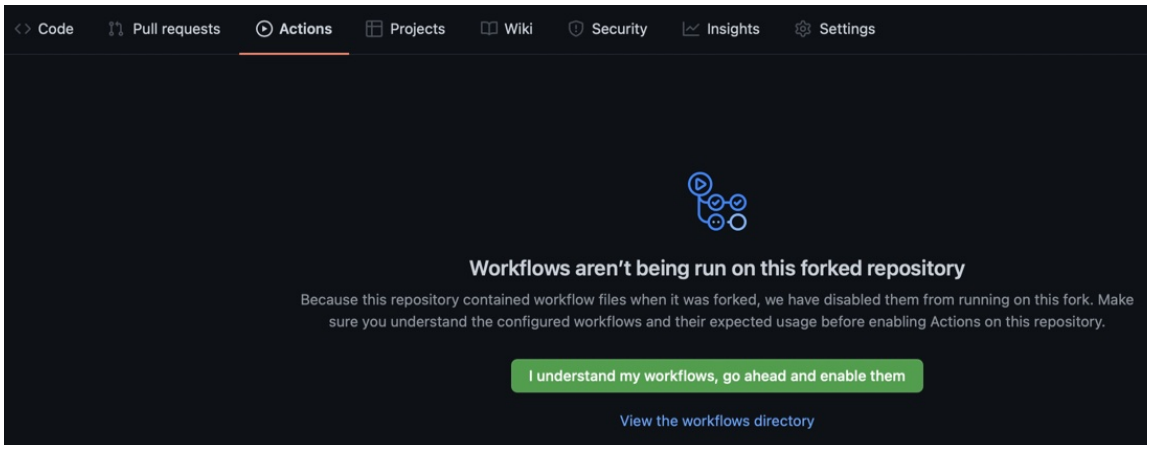


Figure 1: Fork repo

- 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.

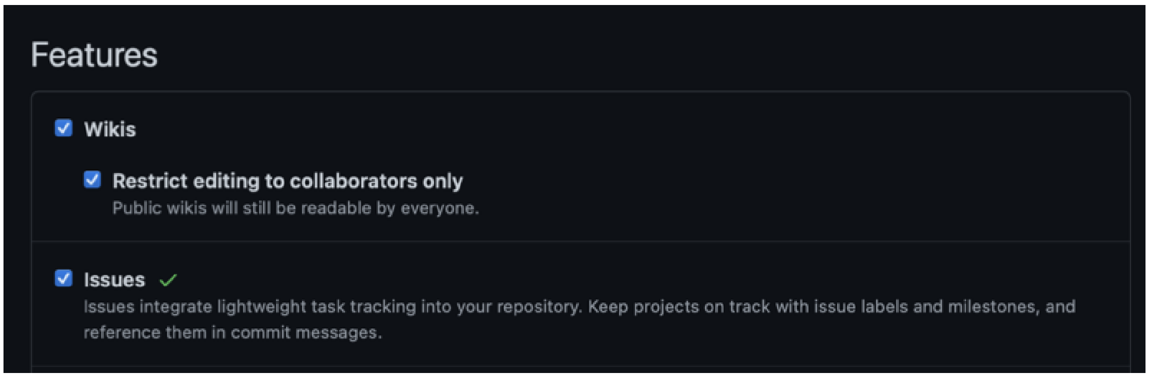

Code
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
I understand my workflows, go ahead and enable them
View the workflows directory

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

☑ Wikis

☑ Restrict editing to collaborators only

Public wikis will still be readable by everyone.

☑ 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 - Secret Scanning

Time to finish lab	15 minutes
App used for this exercise	NYT decompiled

Training Objectives

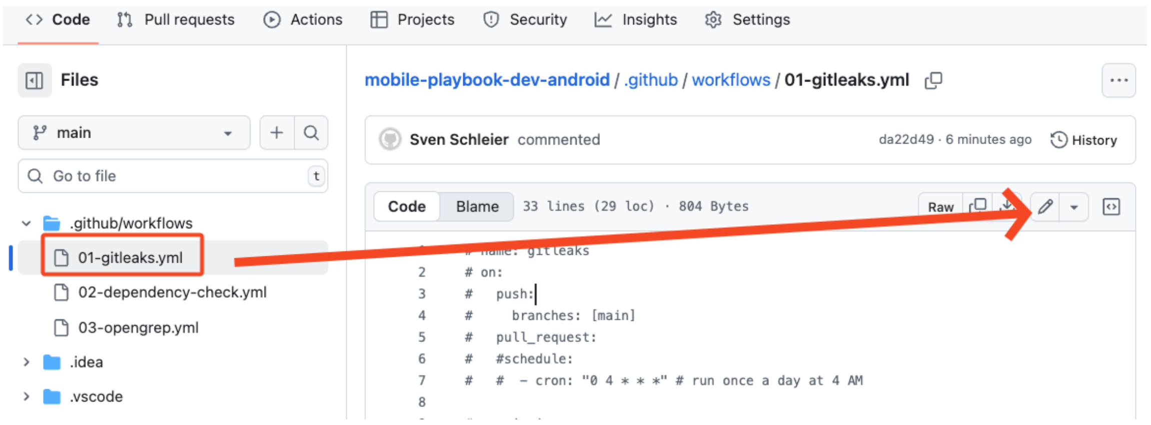
Learn what an attacker would do after decompiling an APK and use the decompiled code to scan it for secrets with the tool `gitleaks`. In our lab I already decompiled the New York Times Android App for you and we simplified it by executing gitleaks through a Github Action.

Tools used in this section

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

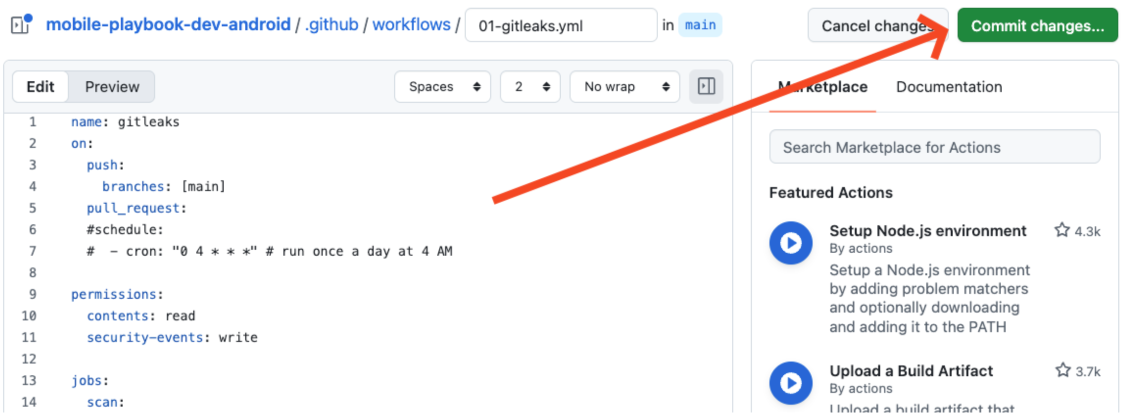
Preparation

- Go to your forked repo and to the main directory in your browser. Select the `.github/workflows` folder and the file `01-gitleaks.yml` and uncomment everything by selecting the whole text and pressing "CTRL" + "/" (Windows) or "Command" + "/" (macOS):



```
# name: gitleaks
# on:
#   push:
#     branches: [main]
#   pull_request:
#   #schedule:
#   #  - cron: "0 4 * * *" # run once a day at 4 AM

```

Figure 4: Edit Gitleaks Github Action

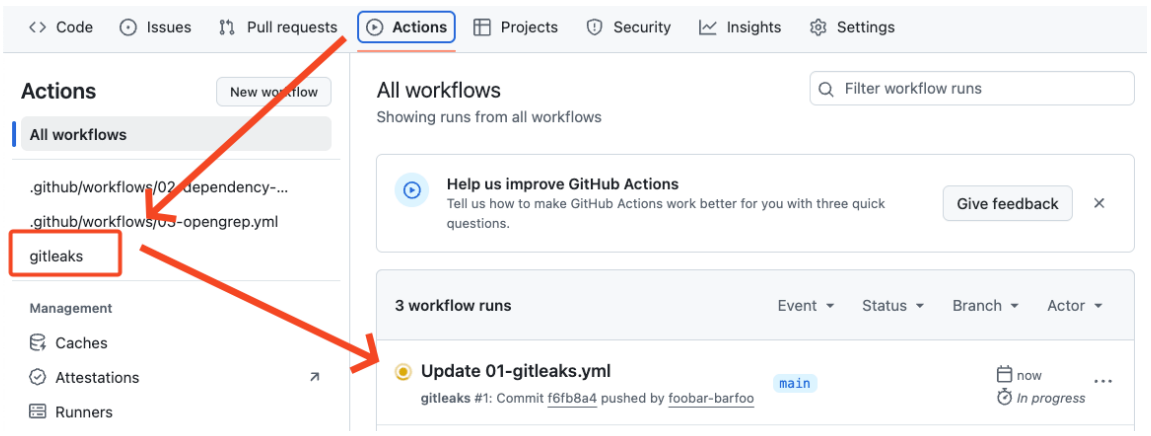


```
name: gitleaks
on:
  push:
    branches: [main]
  pull_request:
  #schedule:
  #  - cron: "0 4 * * *" # run once a day at 4 AM

permissions:
  contents: read
  security-events: write

jobs:
  scan:
```

Setup a Node.js environment by adding problem matchers and optionally downloading and adding it to the PATH

Figure 5: Commit changes

- Click on "Actions" and "gitleaks", you will see the workflows that were just triggered due to the commit. Click on the `Gitleaks` worflow run.

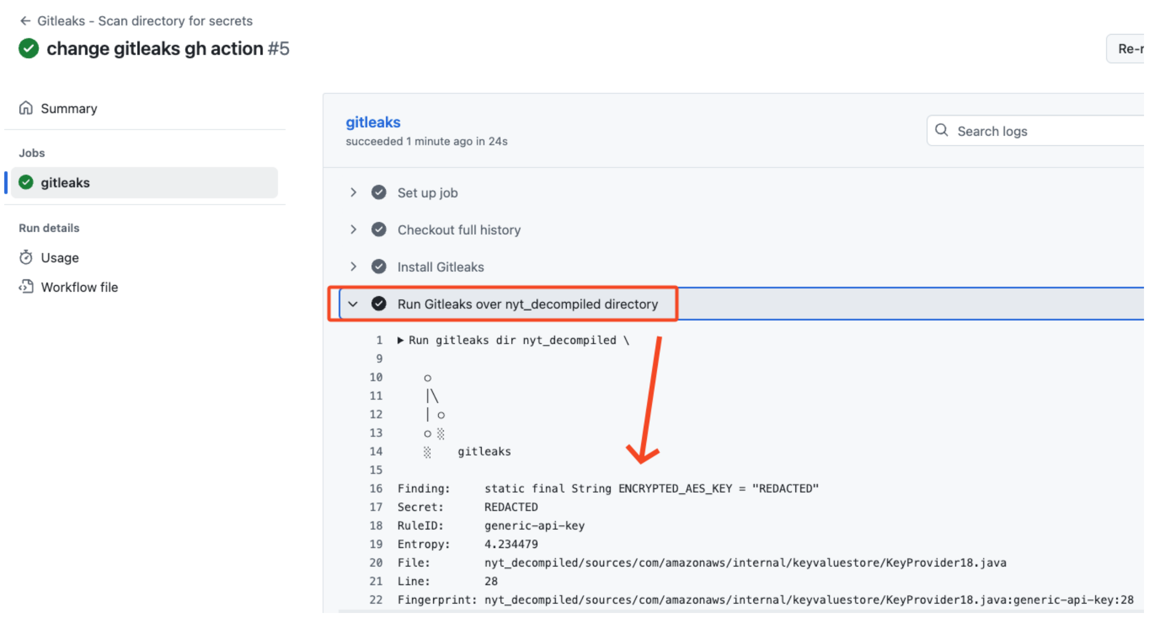


Help us improve GitHub Actions
Tell us how to make GitHub Actions work better for you with three quick questions.

Update 01-gitleaks.yml
gitleaks #1: Commit f6fb8a4 pushed by foobar-barfoo

Figure 6: Actions running

- Click on the step "Run Gitleaks over nyt_decompiled directory".


← Gitleaks - Scan directory for secrets
change gitleaks gh action #5
Re-r
Summary
Jobs
gitleaks
Run details
Usage
Workflow file
gitleaks
succeeded 1 minute ago in 24s
Search logs
Set up job
Checkout full history
Install Gitleaks
Run Gitleaks over nyt_decompiled directory

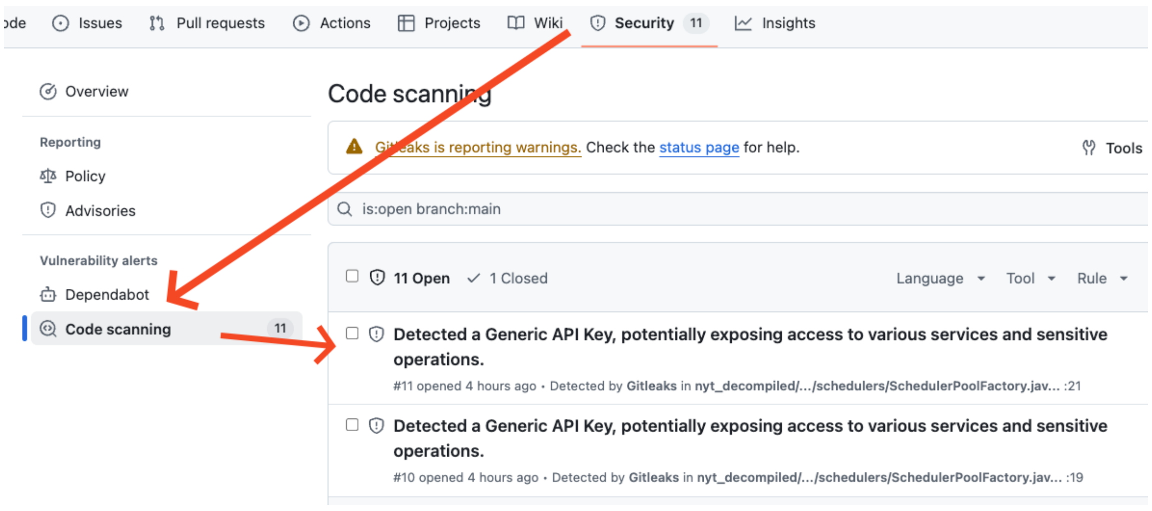
```
▶ Run gitleaks dir nyt_decompiled \


    ○
    |\
    | ○
    ○ ░
    ░    gitleaks

Finding:     static final String ENCRYPTED_AES_KEY = "REDACTED"
Secret:      REDACTED
RuleID:      generic-api-key
Entropy:     4.234479
File:        nyt_decompiled/sources/com/amazonaws/internal/keyvaluestore/KeyProvider18.java
Line:        28
Fingerprint: nyt_decompiled/sources/com/amazonaws/internal/keyvaluestore/KeyProvider18.java:generic-api-key:28
```

Figure 7: Gitleaks run completed

- You will get the detailed output of the whole `gitleaks` job including all steps.
- The `gitleaks` Github Action will now be executed every time we are doing a commit or pull request and will raise now an alert if sensitive information is being detected. But detecting is only half the job, as we need to manage the amount of detected secrets in an efficient way. Luckily this was already being taken care of automatically through our Github action and the SARIF file that was being created and imported into Github Code Scanning. This is the relevant snippet from the `01-gitleaks.yml` file:

```
      - name: Upload SARIF to GitHub Security
        uses: github/codeql-action/upload-sarif@v3
        with:
          sarif_file: gitleaks.sarif
```

- Go to the “Security” tab in your repo and click on “Code scanning” on the left side. All secrets that could be identified by `gitleaks` are all automatically imported into the code scanning alerts.



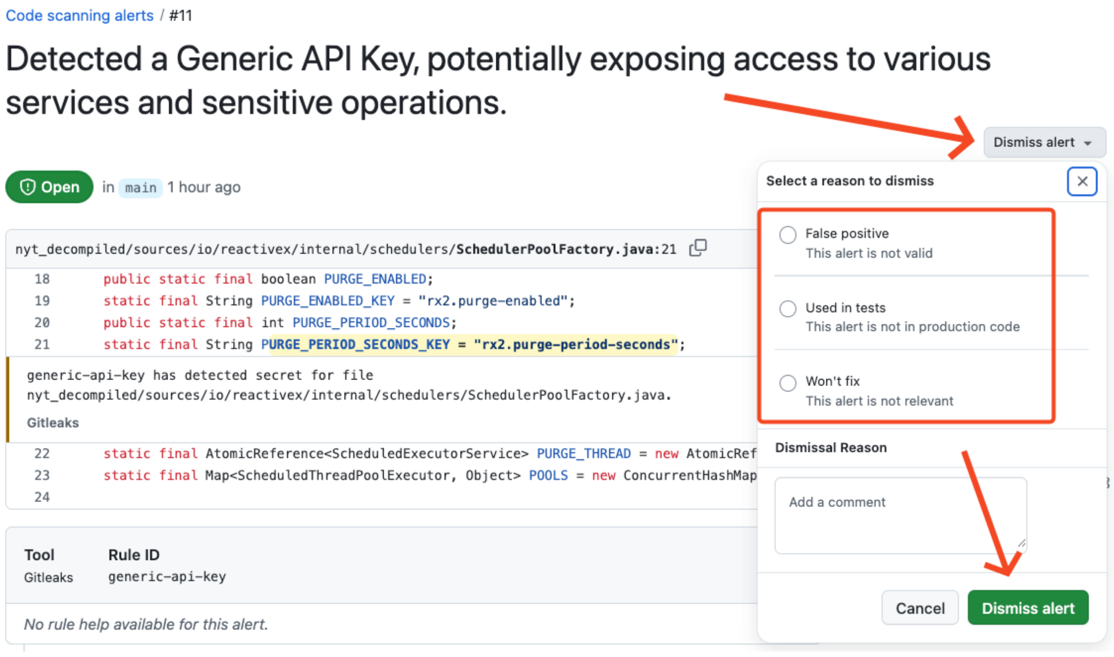
Gitleaks is reporting warnings. Check the status page for help.

Detected a Generic API Key, potentially exposing access to various services and sensitive operations.
#11 opened 4 hours ago • Detected by Gitleaks in nyt_decompiled/.../schedulers/SchedulerPoolFactory.jav... :21

Detected a Generic API Key, potentially exposing access to various services and sensitive operations.
#10 opened 4 hours ago • Detected by Gitleaks in nyt_decompiled/.../schedulers/SchedulerPoolFactory.jav... :19

Figure 8: Gitleaks results

- This allows us to manage the identified alerts. Github takes care of de-duplication of findings automatically through fingerprinting, so Github will not flag out the same finding again in consecutive scans. Click on the first finding and you can get all the details around it.



Detected a Generic API Key, potentially exposing access to various services and sensitive operations.

nyt_decompiled/sources/io/reactivex/internal/schedulers/**SchedulerPoolFactory.java**:21

```
    public static final boolean PURGE_ENABLED;
    static final String PURGE_ENABLED_KEY = "rx2.purge-enabled";
    public static final int PURGE_PERIOD_SECONDS;
    static final String PURGE_PERIOD_SECONDS_KEY = "rx2.purge-period-seconds";
generic-api-key has detected secret for file
nyt_decompiled/sources/io/reactivex/internal/schedulers/SchedulerPoolFactory.java.
Gitleaks
    static final AtomicReference<ScheduledExecutorService> PURGE_THREAD = new AtomicRef
    static final Map<ScheduledThreadPoolExecutor, Object> POOLS = new ConcurrentHashMap
```

Tool	Rule ID
Gitleaks	generic-api-key

No rule help available for this alert.

Figure 9: Gitleaks Triage

- Your job is to triage the identified findings and decide if they are a:
  - valid finding (then keep the finding),
  - False Positive or
  - if you won't fix it.

Once you are done with: Congratulations, you are now able to detect secrets!

Appendix

Rules

The rules in `giteaks` are defined as regular expressions, which can always trigger false positives. Here is an example for AWS keys.

You can add your own configuration and rules, or ignore secets.

Lab - Scan an APK for Secrets

Time to finish lab	15 minutes
App used for this exercise	com.nytimes.android.apk

Training Objectives

In this exercise you are scanning the New York Times app (the APK) for secrets. Your goal is to check if you would report them or if they are a false positive.

Tools used in this section

- `apkleaks` - https://github.com/dwisiswant0/apkleaks
- `jadx` - https://github.com/skylot/jadx

Exercise - Android Analyse Local Storage

The scan with `apkleaks` on the New York Times app has already been done for you and was stored in the file `nyt-apkleaks.txt`.

Your job is now to validate the secrets detected by `apkleaks` by opening the text file `nyt-apkleaks.txt` in your editor of choice.

Make a decision if they are a false positive or a finding that you would report and share with the developer of the app!

Open in another terminal the decompiled Java Code in your editor of choice (like VS Studio Code). It is available in the directory of the repo you cloned earlier:

```
$ cd ~/mobile-playbook-dev-android/nyt_decompiled
```

> With the command `jadx -d com.nytimes.android.rebuild.apk` it was already decompiled for you to save you some time.

In the decompiled code that you opened in your editor you can verify where specific secrets are used and identify it's context of use.

Which secrets are false positives and which secrets would you flag out to a developer if this is a real penetration test and why?

In case you find a S3 or GCS Bucket, you can try the following to check if it's publicly accessible:

```
$ curl <bucket-URL>
```

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

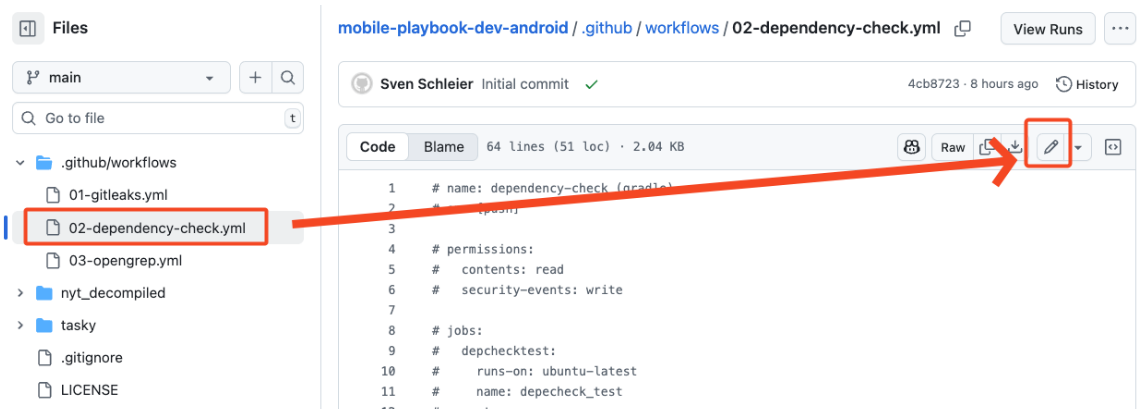
- `dependency-check` - https://jeremylong.github.io/DependencyCheck/
- `dependency-check` Gradle Plugin - https://jeremylong.github.io/DependencyCheck/dependency-check-gradle/index.html

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

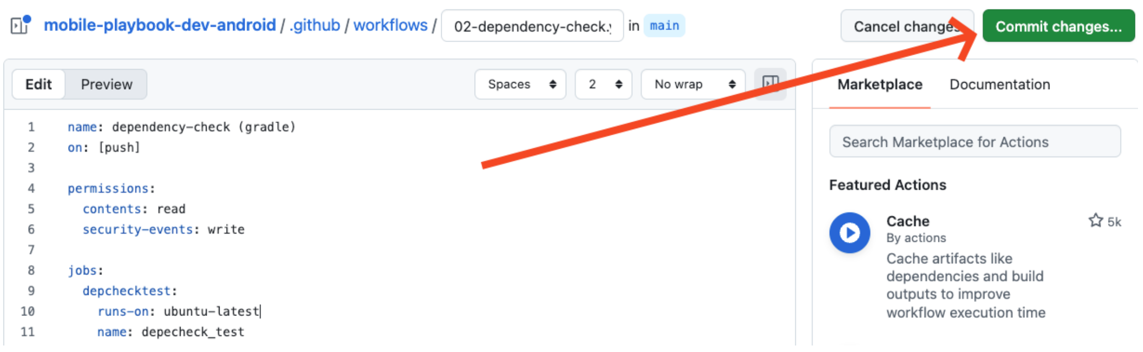
- Go to your forked repo and to the main directory in your browser. Select the `.github/workflows` folder and the file `02-dependency-check.yml` and edit it. and uncomment everything by selecting the whole text and pressing CTRL + "/" and commit your changes.



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

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

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

Figure 10: Edit Yaml file

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

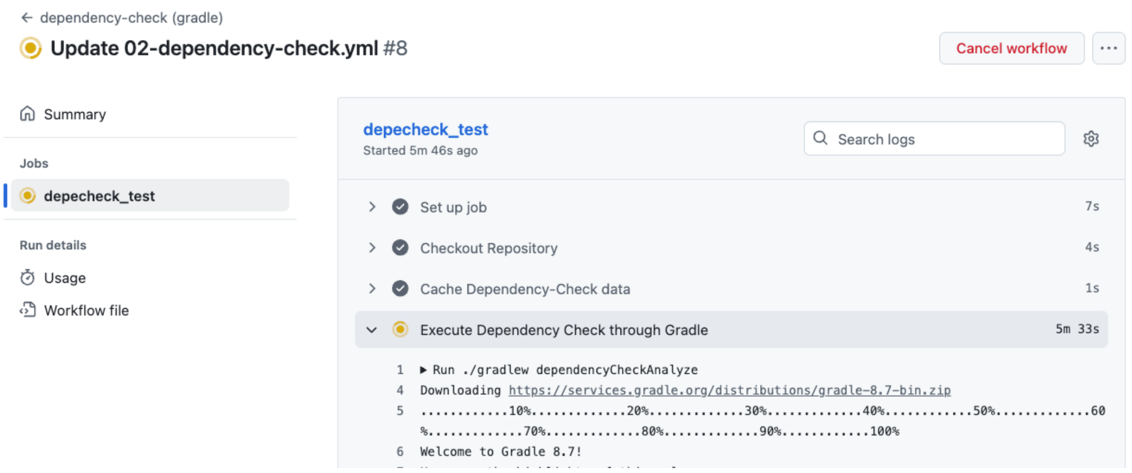

mobile-playbook-dev-android / .github / workflows / 02-dependency-check.y in main
Cancel changes
Commit changes...
Edit
Preview
Spaces
2
No wrap
Marketplace
Documentation
Search Marketplace for Actions
Featured Actions
Cache
By actions
Cache artifacts like dependencies and build outputs to improve workflow 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 11: 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.

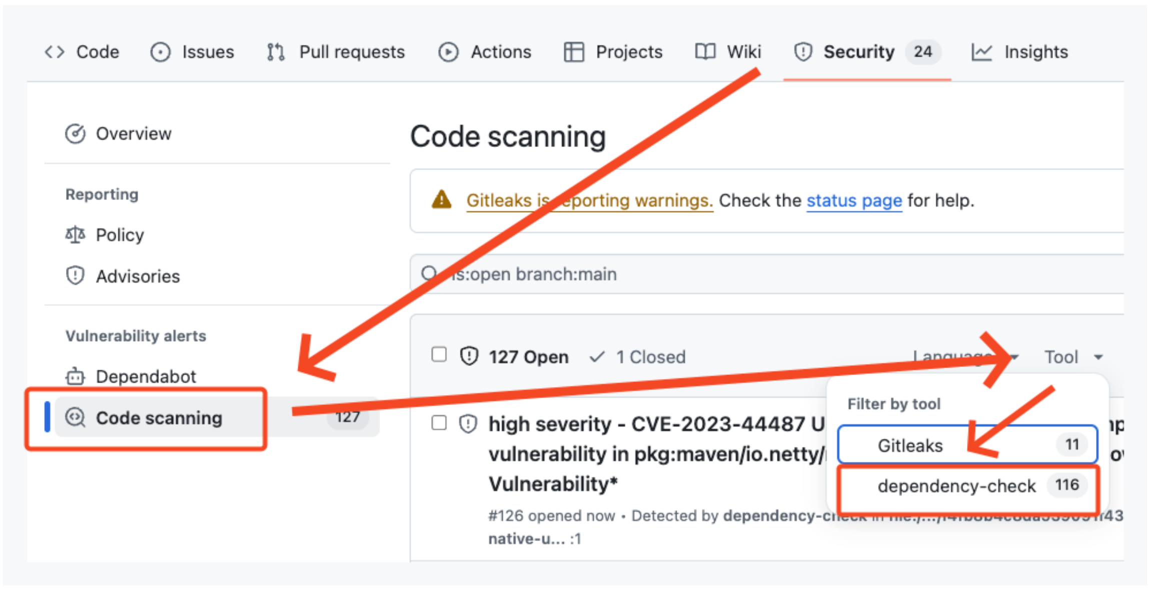
> The first scan can take over 10 minutes, as dependency check need to build up a vulnerability database and download data. This scan was executed already earlier when you did the first lab to speed up the process. In consecutive runs like now, this database is cached in your fork and the workflow will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



```
1  ▸ Run ./gradlew dependencyCheckAnalyze
4  Downloading https://services.gradle.org/distributions/gradle-8.7-bin.zip
5  ............10%.............20%.............30%.............40%............50%.............60
   %.............70%.............80%.............90%............100%
6  Welcome to Gradle 8.7!
```

Figure 12: 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`:



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

is:open branch:main

127 Open ✓ 1 Closed Language Tool

high severity - CVE-2023-44487 U… vulnerability in pkg:maven/io.netty/… Vulnerability*

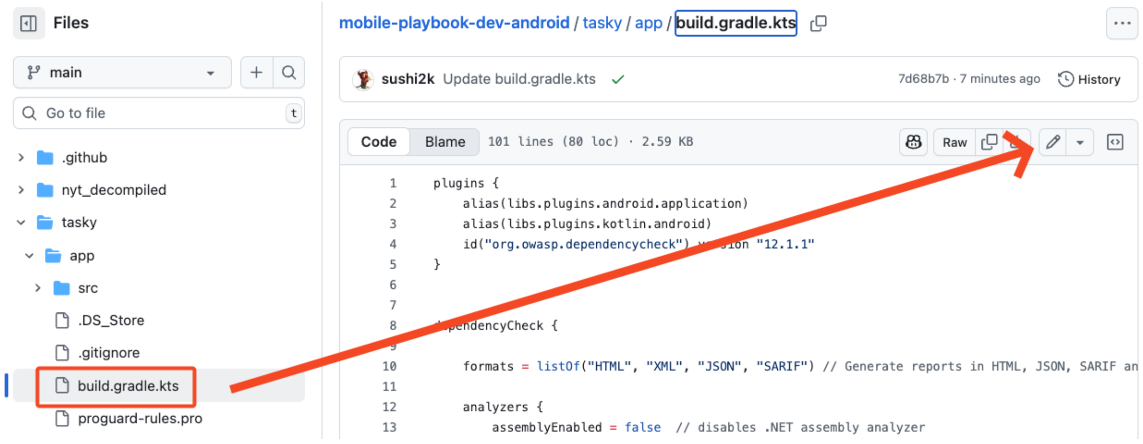
#126 opened now • Detected by dependency-check in file:/…/f4fb8b4e8da59909f43 native-u… :1

Filter by tool

Gitleaks	11
dependency-check	116

Figure 13: Findings

- We have now a lot of false positives through `netty` or `grpc` libraries, that are used to build the project and for testing but are not ending up in our Android app. Therefore we need to find a way to "suppress" them.
- 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:

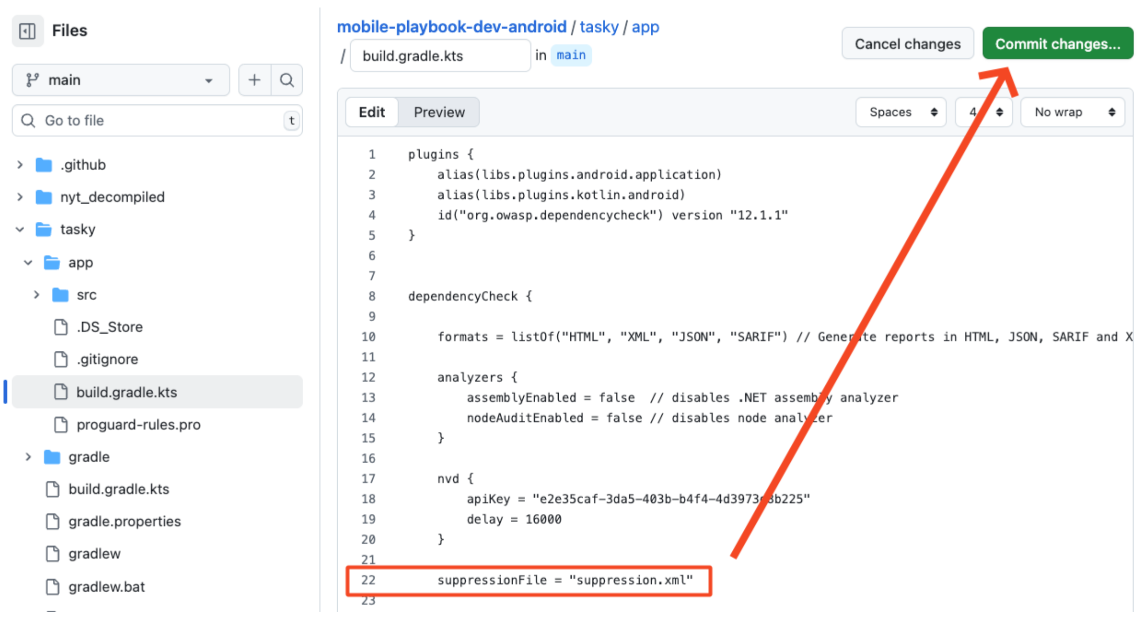


101 lines (80 loc) · 2.59 KB

```
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 14: Edit build.gradle.kts

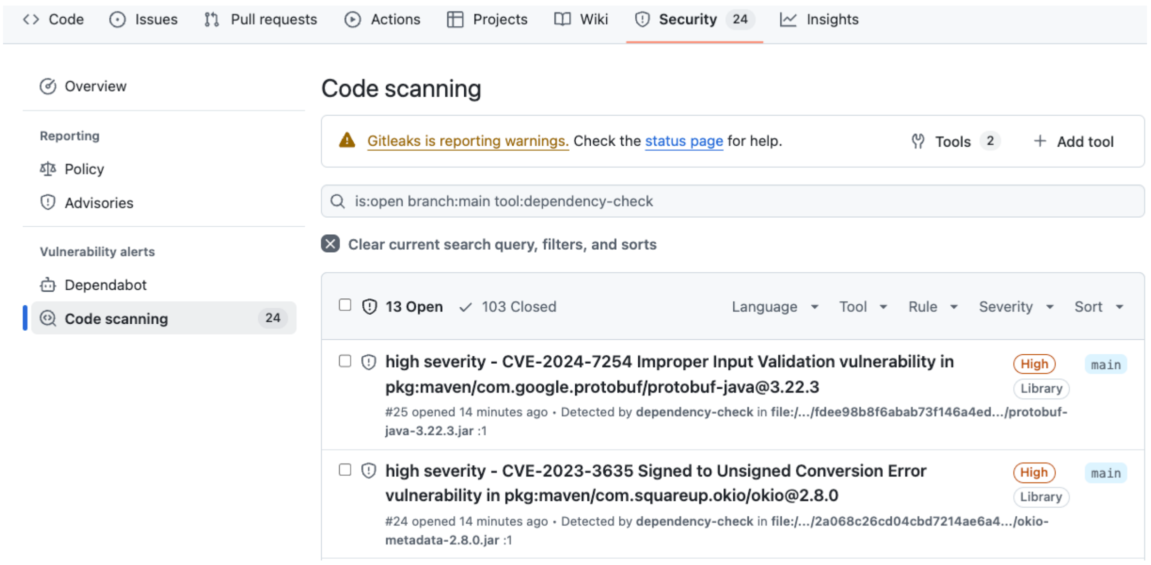
- Remove the comments in line 21 to include the `suppressionFile`.



```
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 15: 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

Clear current search query, filters, and sorts

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

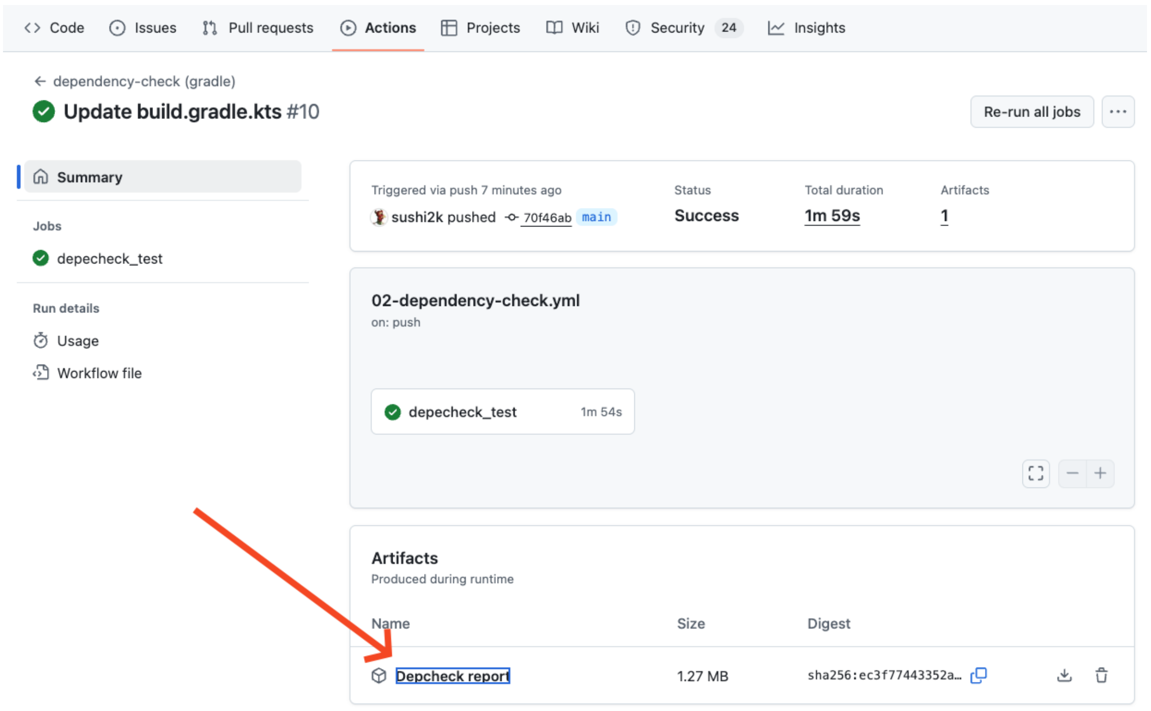
high severity - CVE-2023-3635 Signed to Unsigned Conversion Error vulnerability in pkg:maven/com.squareup.okio/okio@2.8.0 — High — Library — main
#24 opened 14 minutes ago • Detected by dependency-check in file:/.../2a068c26cd04cbd7214ae6a4.../okio-metadata-2.8.0.jar :1

Figure 16: 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.



Update build.gradle.kts #10

Triggered via push 7 minutes ago	Status	Total duration	Artifacts
sushi2k pushed 70f46ab main	Success	1m 59s	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 17: 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!

Lab - Static Scanning with semgrep

Time to finish lab	20 minutes
App used for this exercise	Finstergram.apk and Kotlin-Shack.apk

Training Objectives

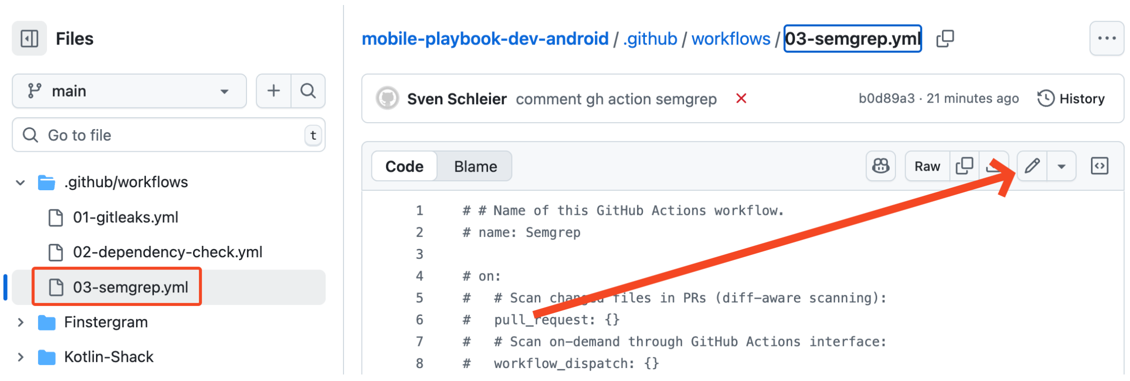
In this lab we will be scanning the decompiled code base of two Android apps for vulnerabilities and misconfiguration with `semgrep` by using a Github Action.

Tools used in this section

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

Preparation

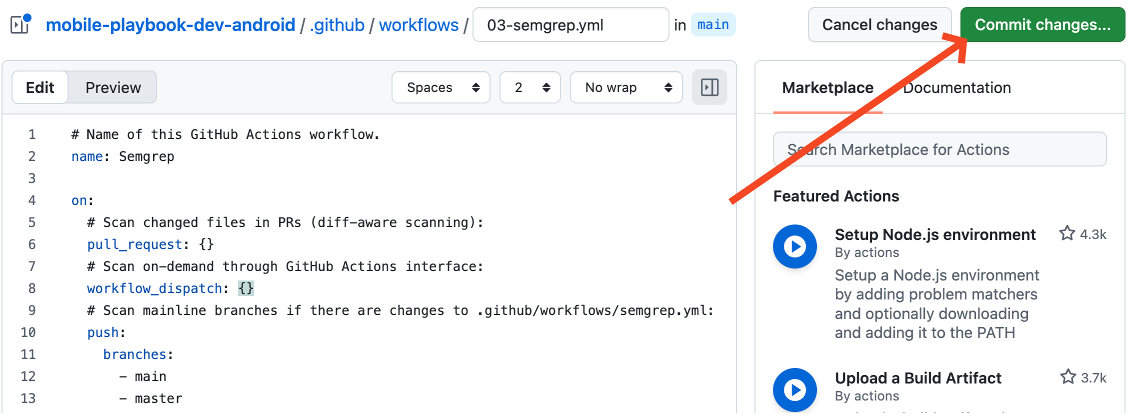
- Go to your forked repo and to the main directory in your browser. Select the `.github/workflows` folder and the file `03-semgrep.yml`.



```
# # Name of this GitHub Actions workflow.
# name: Semgrep

# on:
#   # Scan changed files in PRs (diff-aware scanning):
#   pull_request: {}
#   # Scan on-demand through GitHub Actions interface:
#   workflow_dispatch: {}
```

Figure 18: semgrep workflow

- In the next screen press either the key "e" on your keybord or press the edit button top right. In the editor mode uncomment everything by selecting the whole text and pressing "/" and commit the changes directly into the main branch.


mobile-playbook-dev-android / .github / workflows / 03-semgrep.yml in main
Cancel changes
Commit changes...
Edit
Preview
Spaces
2
No wrap
Marketplace
Documentation
Search Marketplace for Actions
Featured Actions
Setup Node.js environment
By actions
4.3k
Setup a Node.js environment by adding problem matchers and optionally downloading and adding it to the PATH
Upload a Build Artifact
By actions
3.7k

```yaml
# Name of this GitHub Actions workflow.
name: Semgrep

on:
  # Scan changed files in PRs (diff-aware scanning):
  pull_request: {}
  # Scan on-demand through GitHub Actions interface:
  workflow_dispatch: {}
  # Scan mainline branches if there are changes to .github/workflows/semgrep.yml:
  push:
    branches:
      - main
      - master
```

Figure 19: Edit semgrep workflow

Semgrep Github Action

- The app `Finstergram` and `Kotlin-Shack` was already decompiled for you with `jadx` and saved into the repo.
- Click on "Actions" and you will see the workflows that were just triggered due to the commit. Click on the `semgrep` worflow run. The scan might still be running.

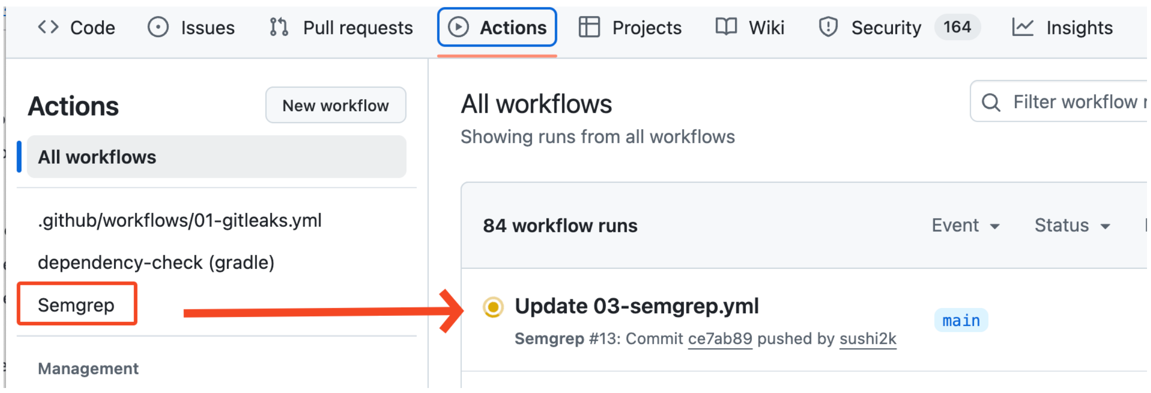


Figure 20: Running semgrep workflow

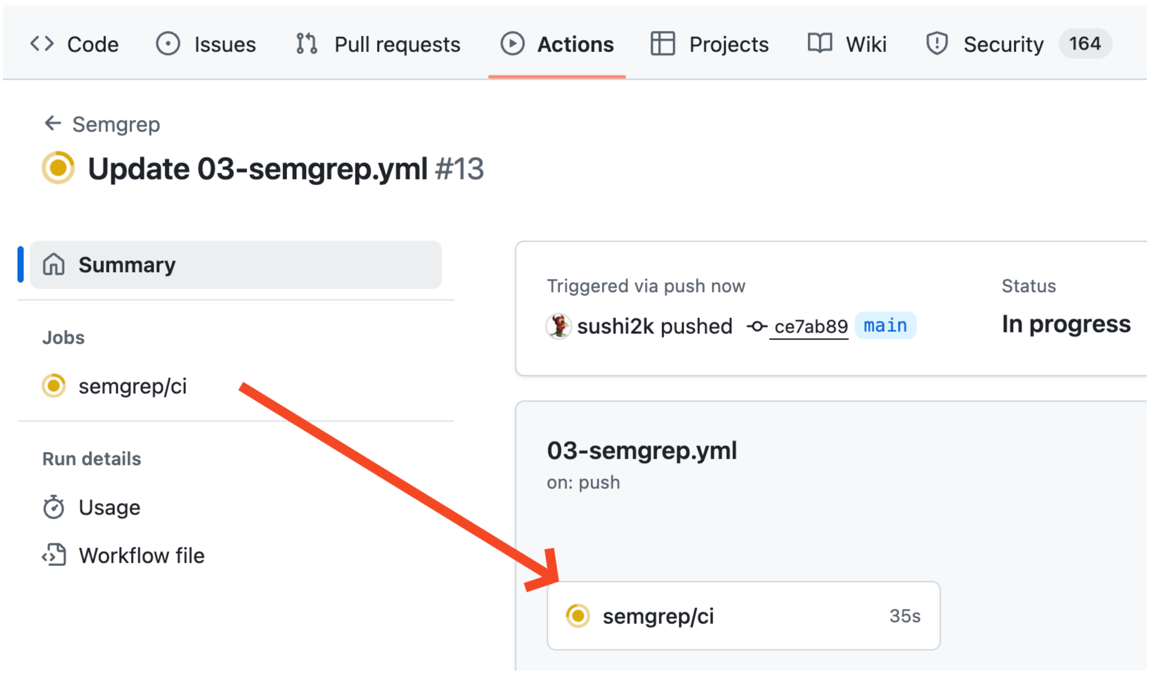
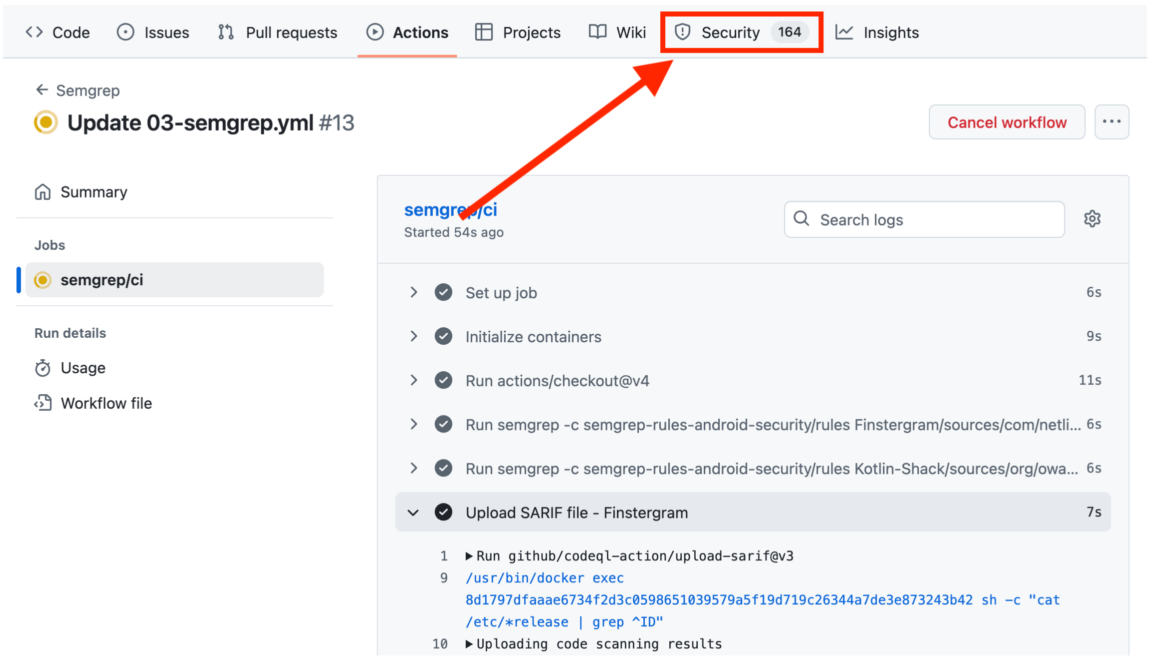


Figure 21: Running semgrep workflow

- Once the semgrep job is done you will have identified new vulnerabilities that were added to the "Security" tab, click on it.



```
Set up job  6s
Initialize containers  9s
Run actions/checkout@v4  11s
Run semgrep -c semgrep-rules-android-security/rules Finstergram/sources/com/netli... 6s
Run semgrep -c semgrep-rules-android-security/rules Kotlin-Shack/sources/org/owa... 6s
Upload SARIF file - Finstergram  7s

▶Run github/codeql-action/upload-sarif@v3
/usr/bin/docker exec
8d1797dfaaae6734f2d3c0598651039579a5f19d719c26344a7de3e873243b42 sh -c "cat
/etc/*release | grep ^ID"
▶Uploading code scanning results
```

Figure 22: Security Tab

- The `semgrep` Github Action will now be executed every time we are doing a commit or pull request and will raise now an alert if sensitive information is being detected. But detecting is only half the job, as we need to manage the amount of detected vulnerabilities in an efficient way. Luckily this was already being taken care of automatically through our Github action and the SARIF file that was being created and imported into Github Code Scanning. This is the relevant snippet from the `semgrep.yml` file:

```
    # Upload the SARIF file to Github, so the findings show up in "
      Security / Code Scanning alerts"
    - name: Upload semgrep report
      uses: github/codeql-action/upload-sarif@v1
      with:
        sarif_file: results.sarif
```

- Go to “Code Scanning Alerts”. This allows us to manage the identified vulnerabilities. Github takes care of de-duplication of findings automatically through fingerprinting, so Github will not flag out the same finding again in consecutive scans.
- 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 (then create an issue),
  - false positive,
  - used in tests, or
  - if you won’t fix it (and why).

WIP

Triage findings

Figure 23: Triage findings

Go through each finding and make a decision! Once done with the first rule, choose another one.

> In case a finding is identified but no code is highlighted, this means the absence of code patterns might be a vulnerability.

Congratulations, you are now able to scan vulnerabilities with `semgrep` and detect and triage vulnerabilities and manage code scanning alerts with Github Code Scanning!

Resources

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

Lab - Frida 101 (Frida-Server)

Training Objectives

1. Learn how to use Frida
2. Use a Frida script that is bypassing root detection

Tools used in this section

- Frida - https://www.frida.re/

App

Use the following app for this exercise:

- **Kotlin-Shack.apk**

Description

Frida supports interaction with the Android runtime. You'll be able to hook and call both Kotlin/Java and native functions inside the process and its (native) libraries. Your JavaScript snippets have full access to memory, e.g. to read and/or write any structured data.

We will demonstrate you basic usage of Frida and how you can overload functions and bypass client side security controls.

Exercise - Frida Usage

Preparation

- Login into Corellium: https://bsides.enterprise.corellium.com/ and open your Android device instance.
- Click on "Apps" on the menu on the left side and then "Install App". Select the app "Kotlin-Shack.apk" from the preparation files you downloaded earlier.
- Start the app `Kotlin-Shack` on the Android device.
- Start the Frida Server by clicking on the Frida tab in Corellium.
- Click on "Select a Process" in Frida and search for "Kotlin" and you will find the running app (your process id, the PID, will be different in your case):

- Once selected, attach to the process and make sure that the app Kotlin-Shack is running in the foreground.
- You can see now the Frida console.

```
     ____
    / _  |   Frida 16.3.3 - A world-class dynamic instrumentation
       toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://frida.re/docs/home/
   . . . .
   . . . .   Connected to 127.0.0.1:27042 (id=socket@127
      .0.0.1:27042)

[Remote::PID::2438 ]->
```

Frida CLI (console) is now connecting to the Frida-Server running on the Android instance and the Frida Core Framework is injecting the Frida-Agent into the specified identifier (`Kotlin-Shack` or the package name `org.owasp.mstgkotlin`).

Click on the multiple root detection button in the app. This will give you a Toast message on the bottom of the UI that the app has detected that the Android instance is rooted.

Our mission is to bypass this check during runtime with Frida and make the app think that this Android phone is NOT rooted.

So what can we do with the interactive Frida console? You can write commands to Frida using it's JavaScript API, just press the `tab` key on your keyboard in the CLI to see the available commands. For example you can query if Java is available or what Android OS version is used:

```
[Remote::PID::2194 ]-> Java.available
true
[Remote::PID::2194 ]-> Java.androidVersion
"14"
```

The Frida CLI allows you to do rapid prototyping to create Frida scripts and also debugging.

Frida CLI + Scripts

Besides using the CLI of `frida`, we can also load scripts. This will load pre-defined JavaScript calls as script that Frida will execute in the context of the targeted app. Click in Corellium on the top right on “Scripts” in the Frida window.

You can find the `Frida-Scripts` directory in the directory where you unzipped the preparation file. Upload the script `test.js` from this directory and execute the script:

Go back to the console and type in `y` and confirm.

In the Frida console, type in `makeToast()` and trigger the function. You can also only type `make` and wait for a second and you should see the dropdown menu with the available functions. You can just select it and then by pressing Enter, but need to add the round brackets.

```
Remote::PID::2194 ]-> makeToast()
```

Look what happened in your Android Instance! Type in `makeToast()` again and trigger the function again. You will see a so called "Toast" message in the bottom of the screen for a few seconds.

So what is happening here (you can see the code also when selecting the script and click on the three doted lines and then "edit")?

- Line 6 is defining a function called `makeToast`, which we just called in Frida.
- Line 8 is checking if the Java runtime is available, so this script will only work for Android.
- Line 10 contains the `Java.perform` wrapper, which is used to attach this function to the current thread, in our case of the Kotlin Shack App.
- Line 11 is getting the application `context` of the app we are hooking into.
- Line 13 is running the function specified on the main thread of the VM by using `Java.scheduleOnMainThread`
- Line 14 is specifying the class (`android.widget.Toast`) Frida should be using and is assigning it to the variable `toast`.
- Line 15 is calling the function `makeText` that will show the string "This works!" as Toast message in the app.

This proofs that we basically can do whatever we want with the Android app by using Frida, including changing the behavior during runtime by injecting code!

Bypass "Multiple Root Detection Check"

In order to bypass the multiple root detection check, we need to know now the class and function that we should hook into. Usually we would decompile the APK with `jadx-gui` to have a look at the decompiled classes. This was already done for you. In the files you downloaded today, go to `Apps/Kotlin-Shack/sources/org/owasp/mstgkotlin/` and open the class `MainActivity.java` in your favourite editor. This file contains a method that is combining several root detection checks.

Your task to find the multiple root detection method and it's name in this class!

Once you have the method name upload another Frida script called `bypass-multi-root.js`. Click on the 3 dots and edit the file.

In line 7 you will see "FUNCTION". Replace this with the function name you identified for the multiple root detection check.

Once done, save it and execute the script and confirm it again in the console.

```
[Remote::PID::2196 ]-> %load /data/corellium/frida/scripts/bypass-
    multi-root.js
Are you sure you want to load a new script and discard all current
    state? [y/N]
y
[*] Script loaded
```

Go back to the app, and click on the multiple root detection button. The Frida script will be triggered but with an error:

Add the **`return`** command with the right boolean value into the Frida Script after the `console.log` command, so that the App thinks the device is not rooted and that the error vanishes.

You can edit the script as follows. Once you edit it, click save and the script will be automatically reloaded.

Once you added the right return value, the output in the app should look like this:

How did we trick now the app into thinking that Android is not rooted? The script contains the following `overload` function:

```
MainActivity.isDeviceRooted.overload().implementation = function()
    {
    console.log("Root detection script triggered")
    return false
}
```

This script is now telling Frida that once the identified function is called from the `MainActivity` class, to overwrite the function with the **`return false`** value. So every time the function is called, it will now only return the boolean value **`false`**.

Congratulations: You bypassed the multiple root detection check with Frida!

You know now 3 different way to bypass root detection:

- Smali patching
- Magisk
- Dynamic Instrumentation with Frida

> Question: What are the benefits and downsides of each technique?

References

- Install Frida on Android - https://www.frida.re/docs/
- Frida Example Script - https://www.frida.re/docs/examples/
- Frida Codeshare - https://codeshare.frida.re/browse
- Frida JavaScript API (Java) - https://www.frida.re/docs/javascript-api/#java

Lab - Bypassing SSL Pinning

Training Objectives

- Bypass different SSL Pinning implementations with Frida

Time for completing this lab: 15 min

Tools used in this section

- Frida

App

Use the following app for this exercise:

- **MSTG-Hands-on.apk**

Description

Certificate pinning is the process of associating the server with a particular X.509 certificate or public key instead of accepting any certificate signed by a trusted certificate authority. After storing ("pinning") the server certificate or public key, the mobile app will subsequently connect to the known server only.

The implementation can be done in various ways:

- By using the TrustManager class;
- By using network libraries such as OkHttp and their build in SSL pinning functions;

This exercise will demonstrate you how to bypass SSL Pinning when implemented with both methods.

Exercise 1 - Bypassing SSL Pinning with Frida

Preparation - Frida

- Install the app `MSTG Hands-on.apk` in the Android device and start it.
- Start the Frida Server by clicking on the Frida tab in Corellium.
- Click on "Select a Process" in Frida and search for "MSTG" and you will find the running app (your process id, the PID, will be different in your case):
- Once selected, attach to the process and make sure that the app MSTG Hands-on is running in the foreground.
- You can see now the Frida console.

```
     ____
    / _  |   Frida 16.3.3 - A world-class dynamic instrumentation
      toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://frida.re/docs/home/
   . . . .
   . . . .   Connected to 127.0.0.1:27042 (id=socket@127
     .0.0.1:27042)

[Remote::PID::10277 ]->
```

Frida CLI (console) is now connecting to the Frida-Server running on the Android instance and the Frida Core Framework is injecting the Frida-Agent into the specified identifier (`MSTG` or the package name `org.owasp.mstg.android`).

MSTG Android App - TrustManager

- Open the Network Tab and start monitoring.
- Click on "HTTPS Request with SSL Pinning TM".
- Due to SSL Pinning we can not intercept and don't see the request in Network Monitor and it will remain empty, but we can try to bypass it with Frida now!
- Select the Frida script `ssl_pinning.js` and click on execute.
- Acknowledge the loading of the script in the "Console" Tab:

```
[Remote::PID::1960 ]-> %load /data/corellium/frida/scripts/
    ssl_pinning.js
Are you sure you want to load a new script and discard all current
    state? [y/N] y
[+] Hook SSL pinning...
```

- Click on the "Back" button and click again on "HTTPS Request with SSL Pinning TM".
- Switch to the networking tab. Which domain was the request sent to?

Exercise 2 - Bypassing SSL Pinning when using OkHttp3

MSTG Hands-on App - OkHttp3

- Go back and click on the button `HTTPS Request with SSL Pinning OkHttp3`
- Verify in Network Monitor if you can intercept the request.
- Your new skill you just learned in Example 1 to bypass SSL Pinning will not be working here, as this function is not using the TrustManager, but a common network library for Android called OkHttp3.
- Find another Frida script on https://codeshare.frida.re/browse that is able to bypass SSLPinning for `OkHTTPv3` and that allows you to intercept the request. Once you found a script (there are quite a few) you can edit the file `ssl_pinning.js`, you can either replace the existing content with your new script that you found and save it, or comment the existing content and append your new script:
- The script will be automatically reloaded. Go back to the network monitor and click again on the button `HTTPS Request with SSL Pinning OkHttp3`, which domain was the request sent to? If you don't see a HTTP request, try another script.

> Is it possible to make the SSL Pinning implementation more secure, so it cannot (easily) be bypassed?

References

- Various SSL Unpinning Frida Scripts - https://codeshare.frida.re
- Bypass SSL Pinning on Android in the MSTG - https://mas.owasp.org/MASTG/techniques/android/MASTG-TECH-0012/#bypass-custom-certificate-pinning-statically
- Bypass SSL Pinning in a Flutter App - https://blog.nviso.be/2019/08/13/intercepting-traffic-from-android-flutter-applications/

Lab - Anti-Frida

Training Objectives

Use a Frida script that is bypassing client side controls

Time for completing this lab: 20 min

Tools used in this section

- Frida - https://www.frida.re/

App

Use the following app for this exercise:

- **Anti-Frida.apk**

Description

There are different ways of detecting Frida that is becoming more common in Android Apps.

In this exercise we will be bypassing common Anti-Frida detection mechanism with.... FRIDA!

We will demonstrate you again basic usage of Frida and how you can overload functions and bypass basic client side security controls.

Preparation - Installation

Install the app `Anti-Frida.apk` in Corellium by going to the apps menu and clicking on "Install App". Afterwards start the app. You will see the following screen:

You can see this app has three different checks available two of them are green. The Frida server should still be running in the background on your Android instance, therefore the "Check Frida Server" will be red. We will explore the checks later in more detail.

Frida CLI (REPL)

- Make sure the app "Anti-Frida" is running in the foreground.

- Click on "Select a Process" in Frida and search for "Frida" and you will find the running app (your process id, the PID, will be different in your case).
- Attach to the process and you can see now the Frida console.

```
     ____
    / _  |   Frida 16.3.3 - A world-class dynamic instrumentation
       toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://frida.re/docs/home/
   . . . .
   . . . .   Connected to 127.0.0.1:27042 (id=socket@127
      .0.0.1:27042)

[Remote::PID::11533 ]->
```

Frida CLI (console) is now connecting to the Frida-Server running on the Android instance and the Frida Core Framework is injecting the Frida-Agent into the specified identifier (`AntiFrida`).

If you go back to Corellium in the Anti-Frida app and click the button "Check Frida in memory" the app could now also detect that we were injecting the Frida-Agent into the app and it turns now also red. No worries, we will turn this around soon!

Your mission is now to hock into this function that is checking the memory for Frida and turn it green again, even though that Frida is injected into the memory.

Identify Classes and Functions

Before we can bypass any of the checks for Frida and inject code, we need to identify the classes and functions we want to manipulate!

In order to identify the classes and methods of the app, it is usually the easiest to decompile the APK to it's Java Source Code and browse through it.

The app `Anti-Frida.apk` was already decompiled by using the tool `jadx-gui`. Open the folder `Apps/Anti-Frida/sources/org/owasp/mstg/antifirda` which is the directory you downloaded earlier today. Open the file `MainActivity.java` in `VS Code` or your editor of choice.

Go through the source code of `MainActivity.java` and you should be able to see three different functions that are the implementation of the three different checks in the app.

Bypass "Check Frida in Memory"

In the folder `Frida-Scripts` of the preparation pack you downloaded earlier is a script called `frida-bypass-AntiFrida.js`.

Find the function name that is executing the memory check for Frida and replace `FUNCTION-NAME` in line 5 of the Frida-script `frida-bypass-AntiFrida.js` with it.

```
    MainActivity.FUNCTION-NAME.overload().implementation = function
        () {
```

Safe the script and upload `frida-bypass-AntiFrida.js` to Corellium and execute it. Switch to the Console and accept the script execution.

Press the button "Check Frida in Memory". The script is not throwing an error, but the check is still red.

```
[Remote::PID::11533 ]-> [*] Script loaded
[*] Script loaded
```

You're mission is now to bypass the memory check! So it looks like this again:

For this, you need to modify the script `frida-bypass-AntiFrida.js`. Check the return value in the script and make the app and Frida work without any errors.

> You can open a 2nd tab in your browser to have the console and scripts screen always open. The display of the Android device can ONLY be open in one tab!

The script is automatically reloaded in the Frida console. Then you can just click the "Check Frida in Memory" button again to see if the bypass is working, there is no need restart Frida or the app. This feature is very handy for creation of Frida scripts.

References

- Install Frida on Android - https://www.frida.re/docs/
- Frida Codeshare - https://codeshare.frida.re/browse
- Frida JavaScript API (Java) - https://www.frida.re/docs/javascript-api/#java
- Frida - https://frida.re
    - `frida-ps` - https://www.frida.re/docs/frida-ps
    - `frida` - https://www.frida.re/docs/frida-cli