**Source code Review / SAST Tools**

**Guidance and & Lists of tools**

* [**https://owasp.org/www-community/Source\_Code\_Analysis\_Tools**](https://owasp.org/www-community/Source_Code_Analysis_Tools)
* [**https://github.com/analysis-tools-dev/static-analysis**](https://github.com/analysis-tools-dev/static-analysis)

**Multi-Language Tools**

[**Naxus - AI-Gents**](https://www.naxusai.com/)

There is a **free package to review PRs**.

[**Semgrep**](https://github.com/returntocorp/semgrep)

It's an **Open Source tool**.

**Supported Languages**

| **Category** | **Languages** |
| --- | --- |
| GA | C# · Go · Java · JavaScript · JSX · JSON · PHP · Python · Ruby · Scala · Terraform · TypeScript · TSX |
| Beta | Kotlin · Rust |
| Experimental | Bash · C · C++ · Clojure · Dart · Dockerfile · Elixir · HTML · Julia · Jsonnet · Lisp · |

**Quick Start**

# Install https://github.com/returntocorp/semgrep#option-1-getting-started-from-the-cli

brew install semgrep

# Go to your repo code and scan

cd repo

semgrep scan --config auto

You can also use the [**semgrep VSCode Extension**](https://marketplace.visualstudio.com/items?itemName=Semgrep.semgrep) to get the findings inside VSCode.

[**SonarQube**](https://www.sonarsource.com/products/sonarqube/downloads/)

There is an installable **free version**.

**Quick Start**

# Run the paltform in docker

docker run -d --name sonarqube -e SONAR\_ES\_BOOTSTRAP\_CHECKS\_DISABLE=true -p 9000:9000 sonarqube:latest

# Install cli tool

brew install sonar-scanner

# Go to localhost:9000 and login with admin:admin or admin:sonar

# Generate a local project and then a TOKEN for it

# Using the token and from the folder with the repo, scan it

cd path/to/repo

sonar-scanner \

-Dsonar.projectKey=<project-name> \

-Dsonar.sources=. \

-Dsonar.host.url=http://localhost:9000 \

-Dsonar.token=<sonar\_project\_token>

**CodeQL**

There is an **installable free version** but according to the license you can **only use free codeQL version in Open Source projects**.

**Install**

# Download your release from https://github.com/github/codeql-action/releases

## Example

wget https://github.com/github/codeql-action/releases/download/codeql-bundle-v2.14.3/codeql-bundle-osx64.tar.gz

# Move it to the destination folder

mkdir ~/codeql

mv codeql-bundle\* ~/codeql

# Decompress it

cd ~/codeql

tar -xzvf codeql-bundle-\*.tar.gz

rm codeql-bundle-\*.tar.gz

# Add to path

echo 'export PATH="$PATH:/Users/username/codeql/codeql"' >> ~/.zshrc

# Check it's correctly installed

## Open a new terminal

codeql resolve qlpacks #Get paths to QL packs

**Quick Start - Prepare the database**

The first thing you need to do is to **prepare the database** (create the code tree) so later the queries are run over it.

* You can allow codeql to automatically identify the language of the repo and create the database

codeql database create <database> --language <language>

# Example

codeql database create /path/repo/codeql\_db --source-root /path/repo

## DB will be created in /path/repo/codeql\_db

This **will usually trigger and error** saying that more than one language was specified (or automatically detected). **Check the next options** to fix this!

* You can do this **manually indicating** the **repo** and the **language** ([list of languages](https://docs.github.com/en/code-security/codeql-cli/getting-started-with-the-codeql-cli/preparing-your-code-for-codeql-analysis#running-codeql-database-create))

codeql database create <database> --language <language> --source-root </path/to/repo>

# Example

codeql database create /path/repo/codeql\_db --language javascript --source-root /path/repo

## DB will be created in /path/repo/codeql\_db

* If your repo is using **more than 1 language**, you can also create **1 DB per language** indicating each language.

export GITHUB\_TOKEN=ghp\_32849y23hij4...

codeql database create <database> --source-root /path/to/repo --db-cluster --language "javascript,python"

# Example

export GITHUB\_TOKEN=ghp\_32849y23hij4...

codeql database create /path/repo/codeql\_db --source-root /path/to/repo --db-cluster --language "javascript,python"

## DBs will be created in /path/repo/codeql\_db/\*

* You can also allow codeql to **identify all the languages** for you and create a DB per language. You need to give it a **GITHUB\_TOKEN**.

export GITHUB\_TOKEN=ghp\_32849y23hij4...

codeql database create <database> --db-cluster --source-root </path/to/repo>

# Example

export GITHUB\_TOKEN=ghp\_32849y23hij4...

codeql database create /tmp/codeql\_db --db-cluster --source-root /path/repo

## DBs will be created in /path/repo/codeql\_db/\*

**Quick Start - Analyze the code**

Now it's finally time to analyze the code

Remember that if you used several languages, **a DB per language** would have been crated in the path you specified.

# Default analysis

codeql database analyze <database> --format=<format> --output=</out/file/path>

# Example

codeql database analyze /tmp/codeql\_db/javascript --format=sarif-latest --output=/tmp/graphql\_results.sarif

# Specify QL pack to use in the analysis

codeql database analyze <database> \

<qls pack> --sarif-category=<language> \

--sarif-add-baseline-file-info \ --format=<format> \

--output=/out/file/path>

# Example

codeql database analyze /tmp/codeql\_db \

javascript-security-extended --sarif-category=javascript \

--sarif-add-baseline-file-info --format=sarif-latest \

--output=/tmp/sec-extended.sarif

**Quick Start - Scripted**

export GITHUB\_TOKEN=ghp\_32849y23hij4...

export REPO\_PATH=/path/to/repo

export OUTPUT\_DIR\_PATH="$REPO\_PATH/codeql\_results"

mkdir -p "$OUTPUT\_DIR\_PATH"

export FINAL\_MSG="Results available in: "

echo "Creating DB"

codeql database create "$REPO\_PATH/codeql\_db" --db-cluster --source-root "$REPO\_PATH"

for db in `ls "$REPO\_PATH/codeql\_db"`; do

echo "Analyzing $db"

codeql database analyze "$REPO\_PATH/codeql\_db/$db" --format=sarif-latest --output="${OUTPUT\_DIR\_PATH}/$db).sarif"

FINAL\_MSG="$FINAL\_MSG ${OUTPUT\_DIR\_PATH}/$db.sarif ,"

echo ""

done

echo $FINAL\_MSG

You can visualize the findings in [**https://microsoft.github.io/sarif-web-component/**](https://microsoft.github.io/sarif-web-component/) or using VSCode extension [**SARIF viewer**](https://marketplace.visualstudio.com/items?itemName=MS-SarifVSCode.sarif-viewer).

You can also use the [**VSCode extension**](https://marketplace.visualstudio.com/items?itemName=GitHub.vscode-codeql) to get the findings inside VSCode. You will still need to create a database manually, but then you can select any files and click on Right Click -> CodeQL: Run Queries in Selected Files

[**Snyk**](https://snyk.io/product/snyk-code/)

There is an **installable free version**.

**Quick Start**

# Install

sudo npm install -g snyk

# Authenticate (you can use a free account)

snyk auth

# Test for open source vulns & license issues

snyk test [--all-projects]

# Test for code vulnerabilities

## This will upload your code and you need to enable this option in: Settings > Snyk Code

snyk test code

# Test for vulns in images

snyk container test [image]

# Test for IaC vulns

snyk iac test

You can also use the [**snyk VSCode Extension**](https://marketplace.visualstudio.com/items?itemName=snyk-security.snyk-vulnerability-scanner) to get findings inside VSCode.

[**Insider**](https://github.com/insidersec/insider)

It's **Open Source**, but looks **unmaintained**.

**Supported Languages**

Java (Maven and Android), Kotlin (Android), Swift (iOS), .NET Full Framework, C#, and Javascript (Node.js).

**Quick Start**

# Check the correct release for your environment

$ wget https://github.com/insidersec/insider/releases/download/2.1.0/insider\_2.1.0\_linux\_x86\_64.tar.gz

$ tar -xf insider\_2.1.0\_linux\_x86\_64.tar.gz

$ chmod +x insider

$ ./insider --tech javascript --target <projectfolder>

[**DeepSource**](https://deepsource.com/pricing)

Free for **public repos**.

**NodeJS**

* **yarn**

# Install

brew install yarn

# Run

cd /path/to/repo

yarn audit

npm audit

* **pnpm**

# Install

npm install -g pnpm

# Run

cd /path/to/repo

pnpm audit

* [**nodejsscan**](https://github.com/ajinabraham/nodejsscan)**:** Static security code scanner (SAST) for Node.js applications powered by [libsast](https://github.com/ajinabraham/libsast) and [semgrep](https://github.com/returntocorp/semgrep).

# Install & run

docker run -it -p 9090:9090 opensecurity/nodejsscan:latest

# Got to localhost:9090

# Upload a zip file with the code

* [**RetireJS**](https://github.com/RetireJS/retire.js)**:** The goal of Retire.js is to help you detect the use of JS-library versions with known vulnerabilities.

# Install

npm install -g retire

# Run

cd /path/to/repo

retire --colors

**Electron**

* [**electronegativity**](https://github.com/doyensec/electronegativity)**:** It's a tool to identify misconfigurations and security anti-patterns in Electron-based applications.

**Python**

* [**Bandit**](https://github.com/PyCQA/bandit)**:** Bandit is a tool designed to find common security issues in Python code. To do this Bandit processes each file, builds an AST from it, and runs appropriate plugins against the AST nodes. Once Bandit has finished scanning all the files it generates a report.

# Install

pip3 install bandit

# Run

bandit -r <path to folder>

* [**safety**](https://github.com/pyupio/safety): Safety checks Python dependencies for known security vulnerabilities and suggests the proper remediations for vulnerabilities detected. Safety can be run on developer machines, in CI/CD pipelines and on production systems.

# Install

pip install safety

# Run

safety check

* [**~~Pyt~~**](https://github.com/python-security/pyt): Unmaintained.

**.NET**

# dnSpy

https://github.com/0xd4d/dnSpy

# .NET compilation

C:\Windows\Microsoft.NET\Framework64\v4.0.30319\csc.exe test.cs

**RUST**

# Install

cargo install cargo-audit

# Run

cargo audit

#Update the Advisory Database

cargo audit fetch

**Java**

# JD-Gui

https://github.com/java-decompiler/jd-gui

# Java compilation step-by-step

javac -source 1.8 -target 1.8 test.java

mkdir META-INF

echo "Main-Class: test" > META-INF/MANIFEST.MF

jar cmvf META-INF/MANIFEST.MF test.jar test.class

| **Task** | **Command** |
| --- | --- |
| Execute Jar | java -jar [jar] |
| Unzip Jar | unzip -d [output directory] [jar] |
| Create Jar | jar -cmf META-INF/MANIFEST.MF [output jar] \* |
| Base64 SHA256 | sha256sum [file] | cut -d' ' -f1 | xxd -r -p | base64 |
| Remove Signing | rm META-INF/*.SF META-INF/*.RSA META-INF/\*.DSA |
| Delete from Jar | zip -d [jar] [file to remove] |
| Decompile class | procyon -o . [path to class] |
| Decompile Jar | procyon -jar [jar] -o [output directory] |
| Compile class | javac [path to .java file] |

**Go**

<https://github.com/securego/gosec>

**PHP**

[Psalm](https://phpmagazine.net/2018/12/find-errors-in-your-php-applications-with-psalm.html) and [PHPStan](https://phpmagazine.net/2020/09/phpstan-pro-edition-launched.html).

**Wordpress Plugins**

<https://www.pluginvulnerabilities.com/plugin-security-checker/>

**Solidity**

* <https://www.npmjs.com/package/solium>

**JavaScript**

**Discovery**

1. Burp:
   1. Spider and discover content
   2. Sitemap > filter
   3. Sitemap > right-click domain > Engagement tools > Find scripts
2. [WaybackURLs](https://github.com/tomnomnom/waybackurls):
   1. waybackurls <domain> |grep -i "\.js" |sort -u

**Static Analysis**

**Unminimize/Beautify/Prettify**

* <https://prettier.io/playground/>
* <https://beautifier.io/>
* See some of the tools mentioned in 'Deobfuscate/Unpack' below as well.

**Deobfuscate/Unpack**

**Note**: It may not be possible to fully deobfuscate.

1. Find and use .map files:
   1. If the .map files are exposed, they can be used to easily deobfuscate.
   2. Commonly, foo.js.map maps to foo.js. Manually look for them.
   3. Use [JS Miner](https://github.com/PortSwigger/js-miner) to look for them.
   4. Ensure active scan is conducted.
   5. Read '[Tips/Notes](https://github.com/minamo7sen/burp-JS-Miner/wiki#tips--notes)'
   6. If found, use [Maximize](https://www.npmjs.com/package/maximize) to deobfuscate.
2. Without .map files, try JSnice:
   1. References: <http://jsnice.org/> & <https://www.npmjs.com/package/jsnice>
   2. Tips:
      * If using jsnice.org, click on the options button next to the "Nicify JavaScript" button, and de-select "Infer types" to reduce cluttering the code with comments.
      * Ensure you do not leave any empty lines before the script, as it may affect the deobfuscation process and give inaccurate results.
3. For some more modern alternatives to JSNice, you might like to look at the following:

* <https://github.com/pionxzh/wakaru>
  + Javascript decompiler, unpacker and unminify toolkit Wakaru is the Javascript decompiler for modern frontend. It brings back the original code from a bundled and transpiled source.
* <https://github.com/j4k0xb/webcrack>
  + Deobfuscate obfuscator.io, unminify and unpack bundled javascript
* <https://github.com/jehna/humanify>
  + Un-minify Javascript code using ChatGPT This tool uses large language modeles (like ChatGPT & llama2) and other tools to un-minify Javascript code. Note that LLMs don't perform any structural changes – they only provide hints to rename variables and functions. The heavy lifting is done by Babel on AST level to ensure code stays 1-1 equivalent.
  + <https://thejunkland.com/blog/using-llms-to-reverse-javascript-minification.html>
    - Using LLMs to reverse JavaScript variable name minification

1. Use console.log();
   1. Find the return value at the end and change it to console.log(<packerReturnVariable>); so the deobfuscated js is printed instead of being executing.
   2. Then, paste the modified (and still obfuscated) js into <https://jsconsole.com/> to see the deobfuscated js logged to the console.
   3. Finally, paste the deobfuscated output into <https://prettier.io/playground/> to beautify it for analysis.
   4. **Note**: If you are still seeing packed (but different) js, it may be recursively packed. Repeat the process.

**References**

* [YouTube: DAST - Javascript Dynamic Analysis](https://www.youtube.com/watch?v=_v8r_t4v6hQ)
* [https://blog.nvisium.com/angular-for-pentesters-part-1](https://web.archive.org/web/20221226054137/https:/blog.nvisium.com/angular-for-pentesters-part-1)
* [https://blog.nvisium.com/angular-for-pentesters-part-2](https://web.archive.org/web/20230204012439/https:/blog.nvisium.com/angular-for-pentesters-part-2)
* [devalias](https://twitter.com/_devalias)'s [GitHub Gists](https://gist.github.com/0xdevalias):
  + [Deobfuscating / Unminifying Obfuscated Web App Code](https://gist.github.com/0xdevalias/d8b743efb82c0e9406fc69da0d6c6581#deobfuscating--unminifying-obfuscated-web-app-code)
  + [Reverse Engineering Webpack Apps](https://gist.github.com/0xdevalias/8c621c5d09d780b1d321bfdb86d67cdd#reverse-engineering-webpack-apps)
  + [etc](https://gist.github.com/search?q=user:0xdevalias+javascript)

**Tools**

* <https://portswigger.net/burp/documentation/desktop/tools/dom-invader>

**Less Used References**

* <https://cyberchef.org/>
* <https://olajs.com/javascript-prettifier>
* <https://jshint.com/>
* <https://github.com/jshint/jshint/>