WHATHEHACK

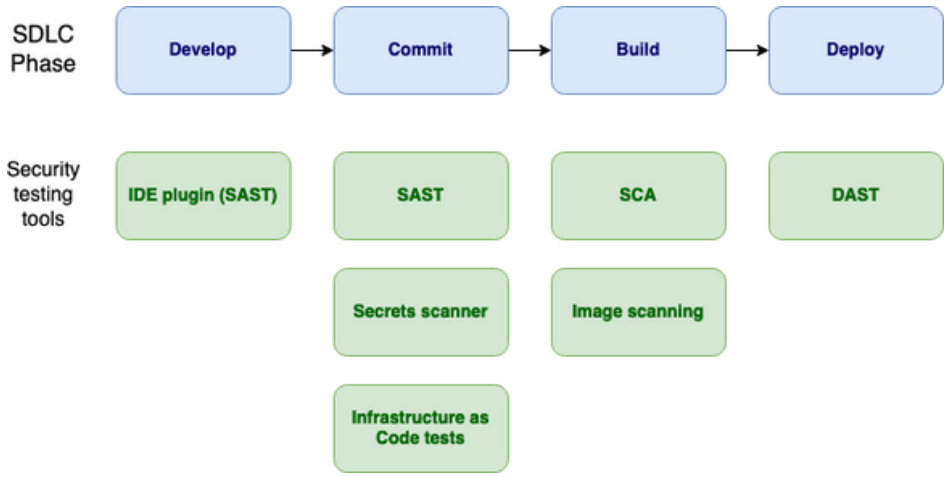
0$ SEPT 2022

AUTOMATIC VULNERABILITY SEARCH

Introduction

Welcome to Whathehack, your favorite monthly cybersecurity and development magazine. The first Wednesday of each month on whathehack.com.

In this topic we will talk about tools that help us finding security issues during the software development lifecycle (SDLC). More specifically, we will talk about Software Composition Analysis (SCA), Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST).



Security Testing Tools by SDLC Phase Diagram from: https://owasp.org/www-project-security-culture/stable/7-Security_Testing/

As long as you cite this magazine and the article author you can freely use this content to teach other people without asking permission . If you find a typo or something that you think is wrong, open an issue for us at github.com/wth-news/magazine. We are not native speakers so we will probably make some mistakes. Thank you very much.

Remember to subscribe to the magazine RSS feed with your favorite reader.

Greetings from Spain!

The articles of this magazine have been written by Samuel López Saura and Guillermo Martínez Esteban.

Software Composition Analysis (SCA)

Summary

Software Composition Analysis (SCA) checks third-party components used within your application.

Use cases

Analyze third party components to:

- Find malicious third party components.
- Find vulnerabilities in third party components.
- Avoid legal issues with problematic licenses of third party components.

Description

SCA tools find security issues by searching reported vulnerabilities for the components used in your application. SCA-type tools have a very low percentage of false positives compared to SAST and DAST tools. SCA tools only checks third-party components. They don't search vulnerabilities included in your own code. Some SCAs also checks licenses of components in order to avoid legal issues by warning you about problematic licenses. The most common ones compare dependency files with known vulnerabilities for the libraries specified in these files.

Example dependency files: *requirements.txt, Gemfile, package.json*

The following file has the content of a dependency file from an application made with python and django. Each line belongs to a external dependency. We can see here how the application uses the django version 3.0.7.

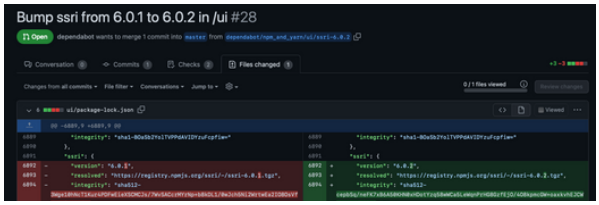
```
django==3.0.7
djangorestframework==3.11.0
psycopg2-binary==2.8.5
django-cors-headers==3.4.0
```

An SCA tool will search security vulnerabilities associated to each library+version specified in this file.

Proof of concept

Safety is a SCA designed for Python that can be freely installed by running pip install safety.
Safety can be run against our project dependencies by specifying the dependency file: *safety check -r requirements.txt.*

Advanced SCA tools could try to fix vulnerabilities automatically. Github dependabot is and example of this. After finding vulnerabilities in dependencies it creates a pull request to fix them by upgrading the component to a higher version where the vulnerability it is expected to be resolved.



Other tools enumeration

- *OWASP dependency-check*
- *Snyk*
- *Sonatype Nexus Lifecycle*
- *Veracode Software Composition Analysis.*

Conclusion

- SCA tools checks third party components vulnerabilities. Some of them could also check problematic licenses.
- SCA tools don't search for vulnerabilities introduced in your code.
- SCA tools have a very low false positives ratio.
- SCA execution is fast. If the team has some kind of development pipeline it is desirable to add a SCA tool to discover third party vulnerabilities and resolve them during development.

Static Application Security Testing (SAST)

Summary

Static Application Security Testing (SCA) analyse the source code of your application searching for programming errors and insecure code.

Use cases

Analyze our source code to:

- Find programming errors.
- Discover vulnerabilities.
- Detect the usage of dangerous methods.

Description

SAST are tools that perform static code analysis in order to find vulnerabilities.

While SAST could be a bit slow and don't check code in runtime, they are good solutions to discover bad habits in source code. Each line is checked and useful information is showed if something is not properly written.

SAST tools should be used frequently during the development. That way, potential security breaches can be quickly fixed, even before new version deploy.

Most of SAST use Abstract Syntax Tree (AST). AST transform programming code in models which could be easily follow by a analyzer. Once SAST has a model, it could be tested to found known issues.

Results of static analysis must be manually confirmed. SAST could detect false positives that can not be automatically discarded. This could be mitigated analizying the code frequently. Some SAST could be integrated with IDE, which is a really good help for developers.

Proof of concept

One SAST solution is bandit, an open source static analyzer for Python. It is very popular and easy to install in Linux machines.

The following code has an important flaw because a malicious user could execute arbitrary coommands.

```
import os
ip = input('Insert IP: ')
os.system('ping %s' % ip)
```

Executing bandit agains our source code:

```
$ bandit -r test.py -o output.csv
```

```
>> Issue: [B605:start_process_with_a_shell] Starting a process with a shell, pos
sible injection detected, security issue.
   Severity: High   Confidence: High
   Location: /code/code/test.py:5
   More Info: https://bandit.readthedocs.io/en/latest/plugins/b605_start_process
_with_a_shell.html
4
5       os.system('ping %s' % ip)
```

Other tools enumeration

- *Brakeman*
- *Checkmarx*
- *AppSweep*
- *Kiwuan*

Conclusion

- SAST tools can find programming errors and discover vulnerabilities in our source code.
- Static analysis could have important costs of time, so each execution have to be carefully managed.
- It is very useful to execute a SAST tool before making a release or a merge request.
- Results have to be checked in order to discard false positives.
- SAST tools can have a high number of false positives.

Dynamic Application Security Testing (DAST)

Summary

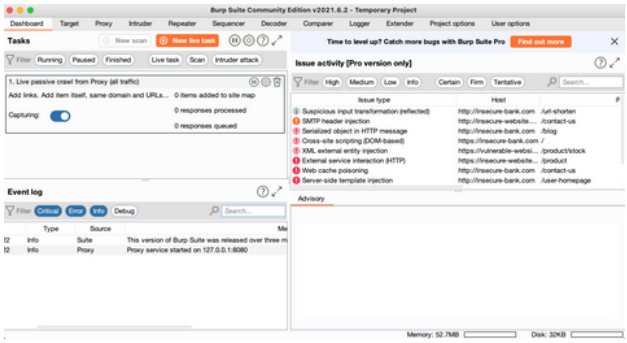
Dynamic Application Security Testing (DAST) is a manual and automated approach to discover vulnerabilities by testing an application during its execution.

Use cases

- Security testing in similar to real world scenarios.
- Discover vulnerabilities simulating a real attack.
- Penetration testing.

Description

DAST tools simulate attacks trying to exploit vulnerabilities in a deployed application. Their execution can be automated but even automation saves penetration testers a lot of time this doesn't replace them. DAST tools test application workflows searching vulnerabilities through different techniques like enumeration, fuzzing or execution of exploits for known vulnerabilities.



DAST tools can have different execution modes in order to search for vulnerabilities in an automated way or manually. Both modes should be used in conjunction but they don't need to be used at the same time. Manual mode can be used in penetration testing when a release is going to be made and automated mode can be executed periodically every night.

DAST tools can be integrated inside continuous deployment pipelines to scan each version of the deployed software.

Proof of concept

Using BURP we can perform a guided security assessment. In the next picture, a web request has been sent to the *repeater* section of BURP and it will be sent modifying some parameters to test a SQLi vulnerability in the login params. The modified content its called a payload. A lot of payloads can be tested in an automatic way for each param using the *intruder* section of BURP suite. The usage of this specific tool goes beyond this article. BURP can also works by itself crawling a web site and testing security vulnerabilities in the discovered attack surface.

OWASP ZAP, which is another DAST tool, has a Github action that allows anyone integrate ZAP scanner in his GitHub continuous deployment pipeline.

Other tools enumeration

- Arachnni-scanner
- Acunnetix
- OWASP ZAP

Conclusion

- Unlike SCA and SAST, DAST tools are executed against running applications.
- Automatic DAST tools helps finding vulnerabilities in deployed scenarios but doesn't replace penetration testing. DAST tools are also used in penetration pesting.
- DAST tools can be added to continue deployment pipelines after the deployment of the software.

WHATHEHACK

THE FIRST WEDNESDAY OF EACH MONTH