OSDA - Open Source Dependencies Analysis

Project Proposal

Hugo Rocha Nº 42132
A42132@alunos.isel.pt
925154475

Tiago Lopes Nº 42171
42171@alunos.isel.ipl.pt
912603315

Rui Lima Nº 42200
42200@alunos.isel.ipl.pt
916291802

Pedro Félix, ISEL
pedrofelix@cc.isel.ipl.pt

March 12, 2018

1 Introduction

The evolution of code brought Open-Source code to the table but with it also came many problems, like the use of unauthorized code and its modification. This is in great deal the result of the proliferation of build tools[1] that facilitated the use of external libraries, since it already contains a dependency graph indicating all libraries to use, directly and indirectly, the programmer doesn't need, anymore, to spend hours searching the web for all the libraries in need. To effectively do this it's needed a set of information regarding the library, e.g. version, name, *url* to where it's stored, its license and its dependencies.

This ease of usage came with a cost, especially regarding one's license, being that a project developed by a certain company might not follow the rules determined in a policy, emitted by the entity, causing safety and lawful problems. The policy serves to indicate what type of licenses are accepted in a range of projects. There can be different types of policies depending on what are the types of projects it's going to affect, for example if its for commercial or private use. Another problem is the fact that now it's extremely complicated to make sure that every library in use is not vulnerable to any known type of attack because of the amount of libraries in use, since it's extremely complicated to keep track of everything.

With that said, many companies sought out for ways to easily validate their projects according to the established restrictions and the dependencies being used.

With OSDA we aim to solve the problem of verifying a project dependencies. This concerns licenses, vulnerabilities and versions in accordance with a policy made by an entity interested in checking if its projects in development are all legal and secure.

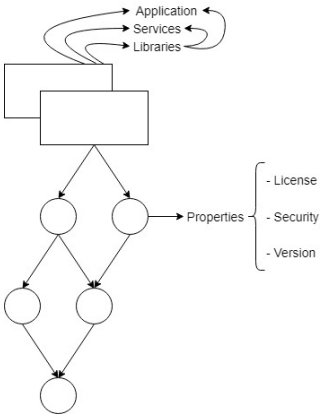

Figure 1: Example of a dependencies graph structure on a Project

To solve this problem, a build plugin is going to be developed that checks if all dependencies are valid according to the policy specifications. If there is an invalid element, the operation fails and an alarm is set notifying the administrator that something went wrong, so that it can be fixed as soon as possible. In either case, success or failure, a human readable report will be generated. The dependencies can be seen as a graph by having a direct connection to the main node(Application, Services or Libraries) and having dependencies of their own as can be seen on Figure 1.

So, this product was thought out to respond to that demand by the companies. Although many solutions to this problem already exist[2], most of those are premium services, meaning that someone with low financial resources like a small company or an individual that's using Open Source code will probably not be able to afford that service. With this in mind, OSDA provides a simple Open Source alternative to premium services.

2 Requirements

OSDA consists in a Web Application where it's possible to validate the dependencies of a given project. Therefore, the plugin running during the build process generates a report showing all modules emphasizing the ones invalidated by the policy or with vulnerabilities.

2.1 Mandatory Requirements

- Definition of the file format holding the company's policies.
  - Present the usable licenses.
  - Ability to create exceptions, where a license that does not correspond to the company policies can be used in the scenario where the module with that license is too important to the overall project to be discarded.
- Validation of a project done in **.NET**, **JVM** or **node.JS**.
  - Analyze all config files in the repository.
  - Create dependencies graph.
  - Analyze dependencies and their license regarding the compliance, creating a report.
- Develop a plugin that validates the dependencies every time a build in gradle is performed, and make its equivalence in **npm** and **.NuGet**.
- Develop a Web Application that periodically checks a set of Github[3] repositories for instance of a department, filtered by tags.
  - Application will build the projects and retrieves the reports that result from the build process.
  - The reports are stored in a data base, enabling their dynamic visualization.
  - In the event of an error raised by the plugin in build time, the project owner will be notified via an alert.

2.2 Optional Requirements

- There can be two policies, one for compilations and another for tests.
- In the report, present alternatives to the modules in which the policies are not valid for the company. The alternative modules should fulfill similar objectives as those of the module being discarded but with a valid license. The alternatives can be one or various, being presented in the report their names, version and url to their documentation.
- Give support to a different package manager for .NET named Paket[4].

3 Architecture

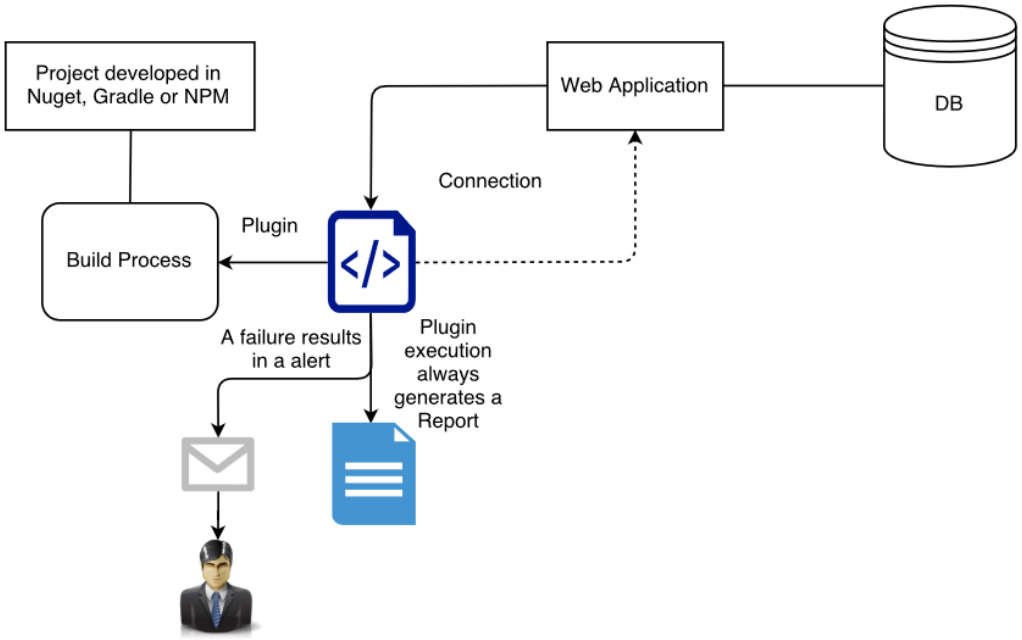

Figure 2: Product architecture

This product is divided in two phases. The first one is the development of a plugin that will be added to the build process of the project in order to validate the dependencies properties(license, vulnerabilities, versions) and generate a report based in the analysis of the project with its relevant information.

The other phase involves a Web Application which identifies repositories stored in GitHub by a tag chosen by the user. In order to access this repositories, the user needs to be logged in into their corresponding GitHub account and consent access. The Application builds each project periodically, stores the report in a database and presents dynamically all the stored information. The build operation is performed with the use of the plugin developed in the previous phase.

4 Risks

One of the risks resides in the fact that external sources, used by the plugin to check dependencies vulnerabilities, might not be kept up to date or accessible at a given time. In this case the plugin won't be able to execute properly and important information might not gonna be available or even wrong by being outdated.

On the possibility of a member of the group not being able to finish one of their tasks, the product development might be delayed and some obligatory requirements could not be done in due time. It's possible that, on the delivery of the product, plugins have different levels of development.

5 Plan

Task	Begin Date	End Date	Duration(weeks)
Proposal Delivery	19/03/2018		
Policy Structure Definition	19/03/2018	26/03/2018	1
Plugin Development	26/03/2018	28/05/2018	9
Find configuration files in project	26/03/2018	2/04/2018	1
Validate the project regarding the policy	02/04/2018	16/04/2018	2
Progress Report	30/04/2018		
Validate the project regarding its vulnerabilities using an external API	16/04/2018	07/05/2018	3
Produce a report with all information	07/05/2018	21/05/2018	2
Validate plugin and add it to build process	21/05/2018	28/05/2018	1
Beta Version	28/05/2018		
Web Application Development	28/05/2018	14/07/2018	7
Periodically build project from github repository	28/05/2018	11/06/2018	2
Data base storage of the reports produce by the build process	11/06/2018	18/06/2018	1
Presentation of the information in the reports	18/06/2018	02/07/2018	2
Validation, bug correction, deployment	02/07/2018	14/07/2018	2
Final Version	14/07/2018		

References

[1] Build tools, 2011. http://www.lihaoyi.com/post/WhatsinaBuildTool.html. Last accessed 06/03/2018.

[2] Blackduck by synopsys, 2003. https://www.blackducksoftware.com.

[3] Github, 2008. https://github.com.

[4] Paket, 2014. https://fsprojects.github.io/Paket/.