March 2022

Automated clearing of software source code files using proximity matching and parsing file contents

Armijn Hemel

Automated clearing of software source code files using proximity matching and parsing file contents

Abstract

A common task for companies is to clear software source code files for legal or security reasons before they can be used by the software developers. The clearing process is tool driven, using tools such as code clone detectors/snippet matchers, license scanners and security scanners. Typically the clearning process starts from 0 for each new file that is analyzed and the fact that open source software is changed incrementally most of the time, and the software being scanned will likely be nearly identical to previously seen software, is not used.

For a (large) subset of files it is possible to use this characteristic to (semi-)automate this process. When scanning a new file, first find a closest file in a set of known files, compute the difference to the known file, checking where the difference in the file is and use rules to determine what action to take depending on where the difference in the file is.

Keywords

proximity matching, open source license compliance, clearing, tlsh, srcml

Background

An increasing amount of products is built with open source software components. These software components come with certain conditions (namely the open source license conditions) and might not be free of defects, leading to security breaches. Many large companies therefore have a process in place to first clear these software components before they can be used.

Clearing typically means:

1. scanning the code to determine the exact license of the code and extract information about authors and/or copyright holders
2. determining for unknown software what the contents of the file are, mostly done using a code clone detector/snippet matcher, which searches for snippets of known code in files using a certain match algorithm (for example “winnowing”[1][2]) and a knowledgebase of information extracted from known open source code
3. making a decision based on the results of steps 1. and 2.

Whenever files that have not been scanned before are scanned, then the process repeats itself.

This method does not use the characteristic of open source that it only tends to be changed in an incremental way and in existing files there are usually no big changes. Instead, small changes are much more likely. This document describes a method to fix this.

Proposed method

To take advantage of the fact that open source is typically only changed in a few places and that earlier decisions can be copied, a few things are needed:

1. a baseline scan with information about files (license, security information, and so on). The collection of these files will be called “known files”.
2. knowledge about the individual parts of each of the known files (which parts are comments, which parts are code, and so on), as well as the new file that is being examined
3. an efficient search mechanism to do proximity matching of the new file to the known files, the result being a closest file, or nothing if there is no closest file, or if the distance between the new file and a closest file is too large for it to be considered close
4. a mechanism to compute the difference between a new file and a closest file in the set of known files
5. decision rules that specify what action needs to be taken using the data from the previous steps

Decision rules could be different depending on what is being looked at. For example, for license extraction it doesn't matter if instructions are changed, but the license text itself is not: the license assessment will be the same. For security scanning it doesn't matter if comments are changed, but actual code changes can be extremely relevant.

Baseline scan

The baseline scan is something that tends to be very company specific. It could be a scan with a tool from a commercial tool vendor, an in house tool or an open source tool.

Extracting knowledge about the structure of a file

Extracting knowledge about the structure of a file can be done by tools such as SrcML[3] which parse a source code file and return information about the structure of the file.

Efficient search mechanism for proximity matching

Proximity matching can be done by locality sensitive hashes, such as TLSH[4]. TLSH hashes of two different files can be compared to eachother. The result is a distance between the two files: low numbers indicate that the difference is small, while a big number indicates that there is a big difference.

TLSH hashes can be put into a data structure called a Vantage Point Tree (VPT), which can then be searched very efficiently[5][6] to find a closest match (there could be multiple best matches with the same distance).

Computing the difference between files

Differences between two files can be efficiently computed using the well known Unix command “diff”[7][8]. There are various implementations and modes for this utility. Which exact implementation should be used is up to whoever implements these steps, as that is dependent on programming langiage and situation.

Decision rules

Decision rules would, like the baseline scan, be company specific. Example rules are: “if there are no changes in the comments copy the license decision from the known file”, or “rescan the file with a security scanner if there is a change in the code”.

Steps

The first step in the process is to create a knowledgebase. The following steps need to be taken:

1. scan the known files with a tool to extract information (license, security, etc.) and store this. Store the original file contents as well.
2. extract information about the structure of each file using SrcML or a similar tool and store this information together with the information of step 1
3. compute the TLSH hash for each of the known files and store this information in such a way that with the TLSH the associated information from steps 1 and 2 can be retrieved
4. construct a Vantage Point Tree using the TLSH hashes from all files from step 3

When scanning a new file the following steps should be taken:

1. compute the TLSH hash of the file
2. search a closest match using the Vantage Point Tree with the TLSH hashes of the known files. The result is a TLSH of a closest match
3. retrieve the license/security/etc. information from the knowledgebase using the TLSH of the match found in step 2
4. compute the difference between the original contents of the match found in step 2 and the new file
5. parse the result from step 4, determine where in the original file corresponding to the match in step 2 and the new file the changes are made
6. (optionally) extract information about the new file using SrcML or a similar tool
7. determine where in the original file corresponding to the match in step 2 and (optionally) in the new file were found
8. apply rules depending on the defined rules on whether or not to automatically accept or reject the decision made for the original file corresponding to the match in step 2

References

[1] https://theory.stanford.edu/~aiken/publications/papers/sigmod03.pdf

[2] https://patents.google.com/patent/US6757675B2/

[3] https://www.srcml.org/

[4] https://tlsh.org/

[5] https://tlsh.org/papersDir/COINS_2020_camera_ready.pdf

[6] https://tlsh.org/papersDir/n21_opt_cluster.pdf

[7] https://pubs.opengroup.org/onlinepubs/9699919799/utilities/diff.html

[8] https://en.wikipedia.org/wiki/Diff