May 2022

Finding a closest match for an ELF file based on proximity matching of extracted identifiers

Armijn Hemel

Finding a closest match for an ELF file based on proximity matching of extracted identifiers

Abstract

A common task for companies is to find the provenance of binary files that they receive from an upstream source. Several methods exist to fingerprint a binary, including string searches[1], looking at function names, using code clone detection, and so on. Some of these methods use a scoring mechanism or chop a file up in smaller pieces, compute a hash and look hashes up in a database.

This document describes a method to turn a sequence of strings and other identifiers into a locality sensitive hash using TLSH[2], which can then be searched in a special datastructure called Vantage Point Tree (VPT) to quickly find a closest match in a collection of TLSH hashes of known files[3][4]. A close match (where “close” is indicated by a threshold value) means that the file that is found has a (near) identical set of identifiers and is therefore likely to be made from the same source code software.

Keywords

proximity matching, open source license compliance, provenance, tlsh

Method

The method conists of first extracting identifiers from known ELF files, computing a TLSH hash over the extracted identifiers and putting the hash values into a Vantage Point Tree and then searching the Vantage Point Tree for close matches and determining if the found match is a good match based on a TLSH distance threshold.

Extracting ELF symbols is described in [5]. Vantage Point Trees and how they are used with TLSH are described in [3][4]. Another use of TLSH and Vantage Point Trees is described in [6].

First compute the TLSH hash for each known ELF file:

1. extract identifiers such as function names, variable names and strings from the ELF file. Optionally the identifiers can be cleaned up (leaving out very short identifiers or very generic ones) or the choice can be made to only use a subset of the identifers (example: only the strings).
2. sort the identifiers
3. concatenate the identifiers, possibly separated by a character such as a newline, carriage return, tab or space, or directly without a separator
4. compute the TLSH hash of the concatenated identifiers

The TLSH hashes are then stored in a Vantage Point Tree.

For an ELF file for which a closest match needs to be found the following is done:

1. extract identifiers such as function names, variable names and strings from the ELF file. If a subset is chosen or the identifiers are first cleaned up, then the same criteria should be used as when computing the TLSH values of the known files.
2. sort the identifiers
3. concatenate the identifiers, possibly separated by a character such as a newline, carriage return, tab or space, or directly without a separator. It is important that if a separator is used it is the same that was used as when generating the TLSH hashes for the known files.
4. compute the TLSH hash of the concatenated identifiers
5. search the VPT tree for a closest match
6. determine whether or not the match returned is a good match based on a TLSH distance threshold.

References

[1] Finding Software License Violations Through Binary Code Clone Detection - https://dl.acm.org/doi/10.1145/1985441.1985453, https://dl.acm.org/doi/10.1145/3468744.3468752

[2] https://tlsh.org/

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

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

[5] Using ELF symbols extracted from dynamically linked ELF binaries for fingerprinting - https://www.tdcommons.org/dpubs_series/4441/

[6] Automated clearing of software source code files using proximity matching and parsing file contents - https://www.tdcommons.org/dpubs_series/4965/