Evaluation of Ethereum Smart Contract Similarity Measures

Bachelor's Thesis in Software and Information Engineering

Author: Raphael Nußbaumer - 01526647 - nussi.rn@gmx.at

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Gernot Salzer

1 Abstract

This thesis explores fuzzy hashing methods to compute similarities between Ethereum Smart Contracts. For this purpose a set of python utilities was implemented and an data-set for evaluation generated.

2 Goal

The goal was to compare different similarity measures and pre-processing steps and evaluate how they respond to changes in the codes. This should be accomplish with a set of reusable python utilities.

3 Introduction

Starting points where the studies (He et al. 2020 [2]) and (Norvill et al. 2017 [5]).

They both use ssdeep[4] described in the paper (Kornblum 2006 [3]).

4 Terms

- **Ethereum account** An account has an eth balance an can execute transactions.
- **Smart Contract** An ethereum account with associated runnable code and data stored on the Ethereum Blockchain.
- **Runtime Code** The runnable code stored on the blockchain as a string of OP-Code bytes. Used synonymously with code or bytecode.
- **EVM** Ethereum Virtual Machine. Transaction base Stack-Machine.
- **OP-Code** An EVM instruction encoded as one byte.

5 Data sets

5.1 Solc Versions and Options

To evaluate the similarity measures I selected a set of 13 solidity smart contracts and compiled them with different solc versions and compiler options. Necessary changes where made to the source code to ensure compatibility with the various solc versions.

The following solc versions where used:

1. `0.5.16`

2. `0.6.12`
3. `0.7.6`
4. `0.8.4`

To evaluate the effect of optimization I applied the following options:

1. `{ enabled: false, runs: 200 }`
2. `{ enabled: true, runs: 0 }`
3. `{ enabled: true, runs: 200 }`
4. `{ enabled: true, runs: 999999 }`

Further the contracts where compiled with ABI encoding v1 and v2.

5.1.1 Optimization statistics

To determine the relevant optimization options I calculated the following statistic on all contracts with verified source on etherscan.io.

optimization-runs	count	proportion	optimization-enabled
0	1379	0.6 %	31.6 %
1	671	0.3 %	99.0 %
$<$ x $<$	1804	0.8 %	99.4 %
200	202289	90.3 %	50.0 %
$<$ 200	17973	8.0 %	99.2 %
total	224116	100 %	54.4 %

5.2 Wallets

5.3 Proxies

6 Pre-Processing

6.1 Segmentation

6.2 Skeletonization

6.3 Op-Code Filtering

6.3.1 F-Statistic Filter

7 Hashing Methods

TODO: reference to exact page number or figure

7.1 Signature similarity

The macro-similarity based on the contract-interface is described in ‘II.C.2) Interface Restoration‘(Di Angelo et al. 2020 [1])

7.2 ssdeep - Context Triggered Piecewise Hashes (CTPH)

ssdeep is based on spamsum[8] wich was written for email spam detection.

Context Triggered Piecewise Hashes follow the steps:

1. Compute a rolling hash of the last n bytes for every position.
2. The rolling hash is used to determine the cutoff points.
3. The resulting chunks are hashed using a traditional cryptographic hash.
4. The final hash results from concatenating part of the chunk-hashes (e.g. the last byte).

7.2.1 Modified ssdeep

The following modifications where made:

- Remove sequence-stripping. It made many hashes incomparable because of long strips of 'K' chunk-hashes.
- Remove rounding in the score calculation to differentiate between exact match and close match as well as incomparable and minor similarity.
- Remove common substring detection to make more hashes comparable.
- Handle case where first chunk is never triggered.
- Add option to use Jaccard-Index for comparison.

7.2.2 JUMP-Hash

7.3 Bytebag - Op-Code Frequency

As lower reference bound for more complex similarity detection I implemented the following measure:

1. Count every byte-value in the code forming a multiset or bag of byte-values, a bytebag.
2. Compare via Jaccard-Index for bags.

7.4 LZJD - Binary-Hashing

This is how a cite[7] looks like.

7.5 Normalized Compression Ratio

8 Evaluation Framework

To compare the similarity measures and evaluate there efficacy a package including test-sets, similarity-measures, python utils for exploration and evaluate was created.

For reuse it is available publicly[6].

9 Results

`TODO:` artifacts, hypothesis

9.1 Bytebag

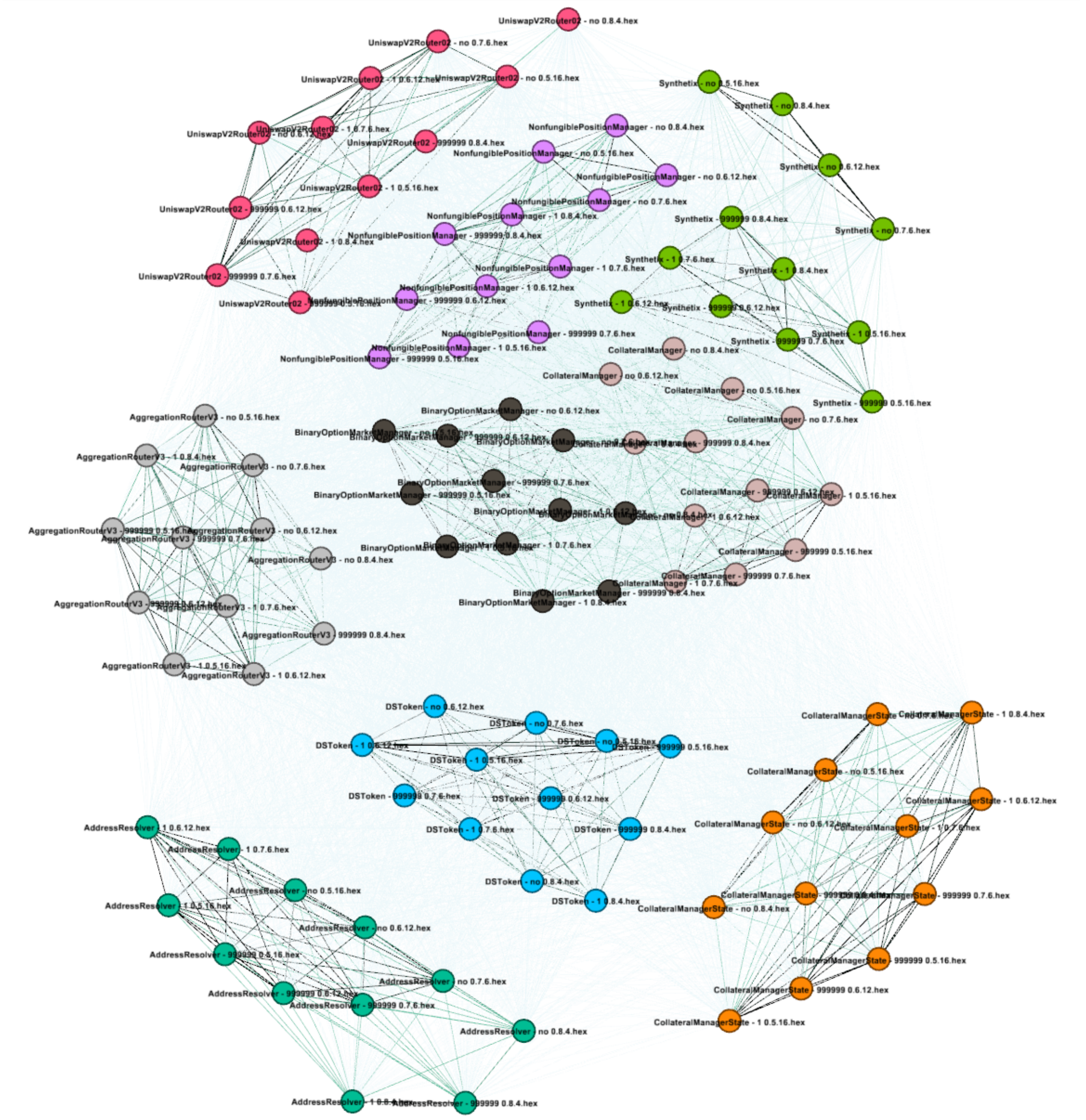
Combined with skeletonization and OP-Code filtering Bytebag performs better than ssdeep in some scenarios.

9.2 LZJD parameters

	Measure	Separation
1	raw ncd	0.8921856639247944
2	skel ncd	0.7532314923619271
3	raw lzjd256	0.7223854289071681
4	raw lzjd512	0.7159224441833137
5	raw lzjd128	0.699177438307873
6	raw lzjd64	0.6401292596944771
7	skel lzjd32	0.5925381903642774
8	skel lzjd256	0.5801997649823737
9	raw lzjd32	0.5655111633372503
10	skel lzjd64	0.5655111633372503
11	skel lzjd512	0.559048178613396
12	skel lzjd128	0.5581668625146886
13	raw lzjd1K	0.5455346650998825
14	skel lzjd1K	0.5

9.3 ssdeep variants

9.4 F-Stat filter



UniswapV2Router02 - no 0.8.4.hex
UniswapV2Router02 - no 0.7.6.hex
UniswapV2Router02 - 1 0.6.12.hex
UniswapV2Router02 - no 0.5.16.hex
UniswapV2Router02 - 1 0.7.6.hex
UniswapV2Router02 - no 0.6.12.hex
UniswapV2Router02 - 999999 0.8.4.hex
UniswapV2Router02 - 1 0.5.16.hex
UniswapV2Router02 - 999999 0.6.12.hex
UniswapV2Router02 - 1 0.8.4.hex
UniswapV2Router02 - 999999 0.7.6.hex
UniswapV2Router02 - 999999 0.5.16.hex
Synthetix - no 0.5.16.hex
Synthetix - no 0.8.4.hex
Synthetix - no 0.6.12.hex
Synthetix - no 0.7.6.hex
Synthetix - 999999 0.8.4.hex
Synthetix - 1 0.7.6.hex
Synthetix - 1 0.8.4.hex
Synthetix - 1 0.6.12.hex
Synthetix - 999999 0.6.12.hex
Synthetix - 999999 0.7.6.hex
Synthetix - 1 0.5.16.hex
Synthetix - 999999 0.5.16.hex
NonfungiblePositionManager - no 0.8.4.hex
NonfungiblePositionManager - no 0.5.16.hex
NonfungiblePositionManager - no 0.6.12.hex
NonfungiblePositionManager - no 0.7.6.hex
NonfungiblePositionManager - 1 0.8.4.hex
NonfungiblePositionManager - 999999 0.8.4.hex
NonfungiblePositionManager - 1 0.7.6.hex
NonfungiblePositionManager - 1 0.6.12.hex
NonfungiblePositionManager - 999999 0.6.12.hex
NonfungiblePositionManager - 999999 0.7.6.hex
NonfungiblePositionManager - 1 0.5.16.hex
NonfungiblePositionManager - 999999 0.5.16.hex
CollateralManager - no 0.8.4.hex
CollateralManager - no 0.6.12.hex
CollateralManager - no 0.5.16.hex
CollateralManager - no 0.7.6.hex
CollateralManager - 999999 0.8.4.hex
CollateralManager - 1 0.5.16.hex
CollateralManager - 1 0.6.12.hex
CollateralManager - 999999 0.5.16.hex
CollateralManager - 999999 0.7.6.hex
CollateralManager - 1 0.7.6.hex
BinaryOptionMarketManager - no 0.6.12.hex
BinaryOptionMarketManager - 999999 0.7.6.hex
BinaryOptionMarketManager - 999999 0.5.16.hex
BinaryOptionMarketManager - 1 0.7.6.hex
BinaryOptionMarketManager - 999999 0.8.4.hex
BinaryOptionMarketManager - 1 0.8.4.hex
AggregationRouterV3 - no 0.5.16.hex
AggregationRouterV3 - 1 0.8.4.hex
AggregationRouterV3 - no 0.7.6.hex
AggregationRouterV3 - 999999 0.5.16.hex
AggregationRouterV3 - 999999 0.7.6.hex
AggregationRouterV3 - no 0.6.12.hex
AggregationRouterV3 - no 0.8.4.hex
AggregationRouterV3 - 999999 0.6.12.hex
AggregationRouterV3 - 1 0.7.6.hex
AggregationRouterV3 - 999999 0.8.4.hex
AggregationRouterV3 - 1 0.5.16.hex
AggregationRouterV3 - 1 0.6.12.hex
DSToken - no 0.6.12.hex
DSToken - no 0.7.6.hex
DSToken - 1 0.6.12.hex
DSToken - no 0.5.16.hex
DSToken - 999999 0.5.16.hex
DSToken - 1 0.5.16.hex
DSToken - 999999 0.7.6.hex
DSToken - 999999 0.6.12.hex
DSToken - 1 0.7.6.hex
DSToken - 999999 0.8.4.hex
DSToken - no 0.8.4.hex
DSToken - 1 0.8.4.hex
CollateralManagerState - 1 0.8.4.hex
CollateralManagerState - no 0.5.16.hex
CollateralManagerState - 1 0.6.12.hex
CollateralManagerState - no 0.6.12.hex
CollateralManagerState - 1 0.7.6.hex
CollateralManagerState - 999999 0.7.6.hex
CollateralManagerState - no 0.8.4.hex
CollateralManagerState - 999999 0.5.16.hex
CollateralManagerState - 999999 0.6.12.hex
CollateralManagerState - 1 0.5.16.hex
AddressResolver - 1 0.6.12.hex
AddressResolver - 1 0.7.6.hex
AddressResolver - no 0.5.16.hex
AddressResolver - 1 0.5.16.hex
AddressResolver - no 0.6.12.hex
AddressResolver - 999999 0.5.16.hex
AddressResolver - no 0.7.6.hex
AddressResolver - 999999 0.6.12.hex
AddressResolver - 999999 0.7.6.hex
AddressResolver - no 0.8.4.hex
AddressResolver - 1 0.8.4.hex
AddressResolver - 999999 0.8.4.hex

10 Remarks

Section um alles zu notieren, was Ihnen an Absonderlichkeiten oder Schwierigkeiten untergekommen ist, inklusive Lessons learned (also was Sie beachten würden, wenn Sie nochmals beginnen würden); wenn es da viel zu berichten gibt, können Sie es natürlich weglassen. Der Sinn einer eigenen Section ist, dass Sie hier Ihre Eindrücke informell ohne tiefere Begründungen wiedergeben können, während die Aussagen in den anderen Abschnitten begründet sein sollten.

11 Conclusion

next steps

goals for followup papers

References

[1] Monika Di Angelo and Gernot Salzer. “Characteristics of Wallet Contracts on Ethereum”. In: *2020 2nd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*. IEEE. 2020, pp. 232–239.

[2] Ningyu He et al. “Characterizing code clones in the ethereum smart contract ecosystem”. In: *International Conference on Financial Cryptography and Data Security*. Springer. 2020, pp. 654–675.

[3] Jesse Kornblum. “Identifying almost identical files using context triggered piecewise hashing”. In: *Digital investigation* 3 (2006), pp. 91–97.

[4] Jesse Kornblum. *ssdeep - Fuzzy hashing program*. 2017. URL: `https://ssdeep-project.github.io/ssdeep/index.html` (visited on 03/09/2022).

[5] Robert Norvill et al. “Automated labeling of unknown contracts in ethereum”. In: *2017 26th International Conference on Computer Communication and Networks (ICCCN)*. IEEE. 2017, pp. 1–6.

[6] Raphael Nußbaumer. *ethereum-contract-similarity*. 2022. URL: `https://github.com/mrNuTz/ethereum-contract-similarity` (visited on 03/07/2022).

[7] Andrew H Sung et al. “Static analyzer of vicious executables (save)”. In: *20th Annual Computer Security Applications Conference*. IEEE. 2004, pp. 326–334.

[8] Andrew Tridgell. *Spamsum*. 2002. URL: `http://samba.org/ftp/unpacked/junkcode/spamsum/README` (visited on 03/09/2022).