Clustering Smart Contracts based on Vulnerabilities

BTP Presentation
1901CS02 . Abhinav Dutta
Under Prof. Raju Halder

Introduction

- We will look at ways of measuring similarity between smart contracts. that has a high correlation with vulnerability which would allows us to cluster them efficiently.
- Main contribution - Propose LHS amenable distance metrics and compare their suitability based on their correlation with vulnerabilities. Use the above distance metrics to cluster smart contracts and analyze their effectiveness.

General Workflow Of Vulnerability detection Tools

- Embedding – Using tools like code2vec, word2vec, fasttext (SmartEmbed) or one-hot encoding over handcrafted features(N. Jiang, Deckard)
- Similarity – Usual (cosine, euclidean)

SmartEmbed

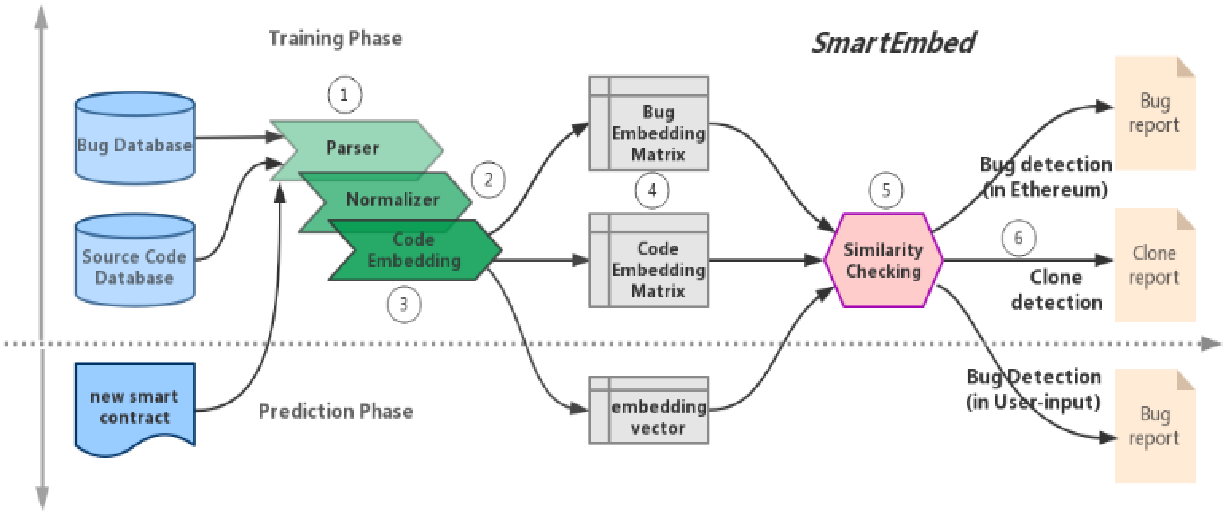


Fig. 1. Overview of Our Approach

- Normalization -> clean up the code

Replaced single-character variables, such as "i", "j", "a", "b", "k", etc., with "SimpleVar" . Eg. uint private r = 0 ; -> uint private SimpleVar = 0 ; Removed non-essential punctuations such as ', ', ";" , etc.

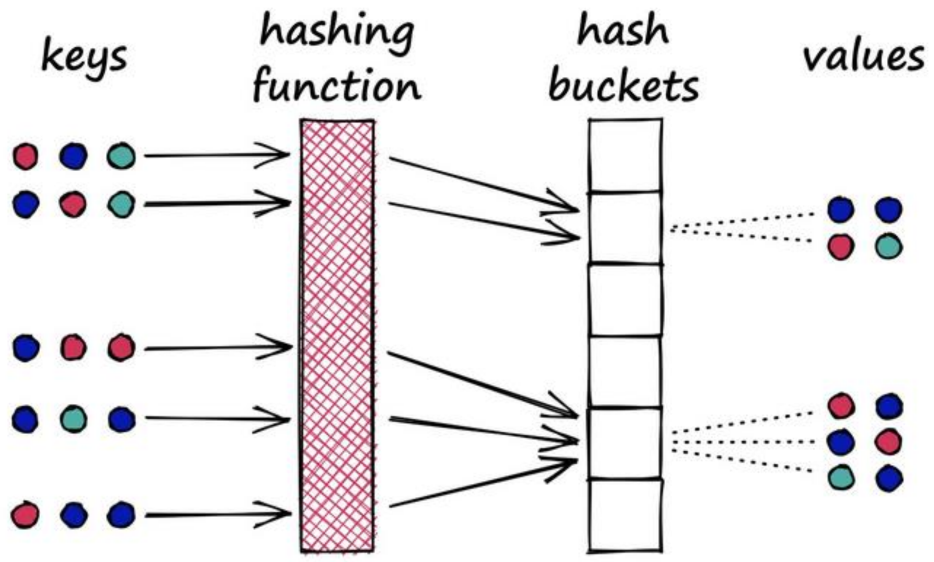
- **Embedding** -> Use FastText (better than word2vec).

It treats each word as the aggregation of its subwords.

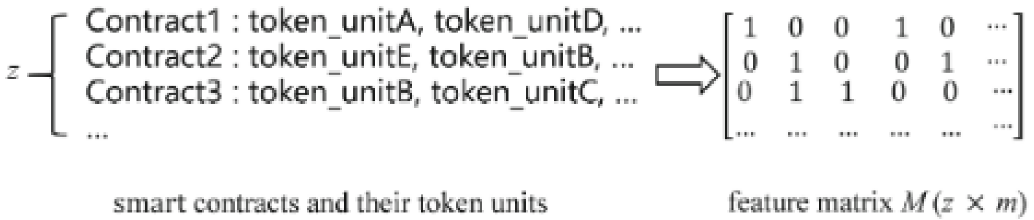
Subwords are taken to be the n-gram of the word, and the vector for a word with FastText is the sum of all n-gram vectors of its component.

Similarity -> Euclidean

Locality Sensitive Hashing (LSH)

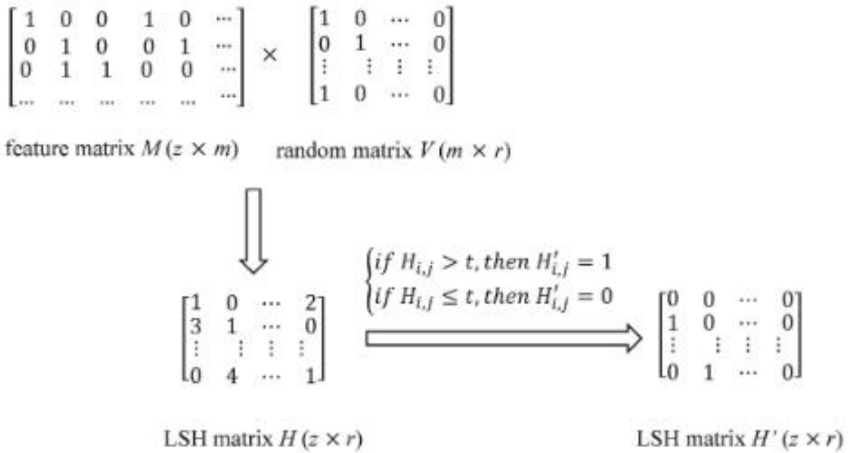


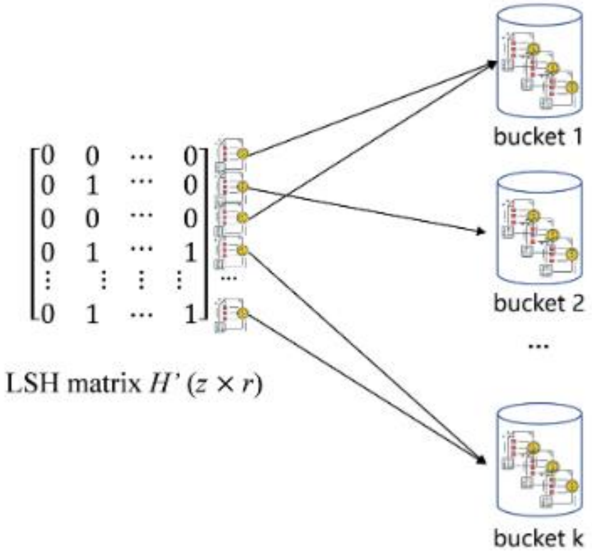
- ((p1, p2, r, c)-Sensitive Hashing) A family
  F of hash functions $h : V \to U$ is called (p1, p2, r, c)-
  sensitive ($c \geq 1$), if $\forall v_i, v_j \in V$,
  if $D(v_i, v_j) < r$ then $P[h(v_i) = h(v_j)] > p_1$
  if $D(v_i, v_j) > cr$ then $P[h(v_i) = h(v_j)] < p_2$
- Basically items similar (ie within some threshold) should hash to same values



N. Jiang

- Embedding – One hot encoding of 90 possible features. Eg. if(to == address(this))" contains three types of syntax tokens - IfStatement, BinaryExpression, and CallExpression.
- Similarity - LSH





MOSS

- Breaks a document as a set of n-grams. Selects a subset of n-grams (using a 'local' scheme called minimizers) and puts it in a hash table. Detects similar pieces of code using chaining + dynamic programming to get alignment.
- Serves as our inspiration to use Jaccard Distance as a distance metric

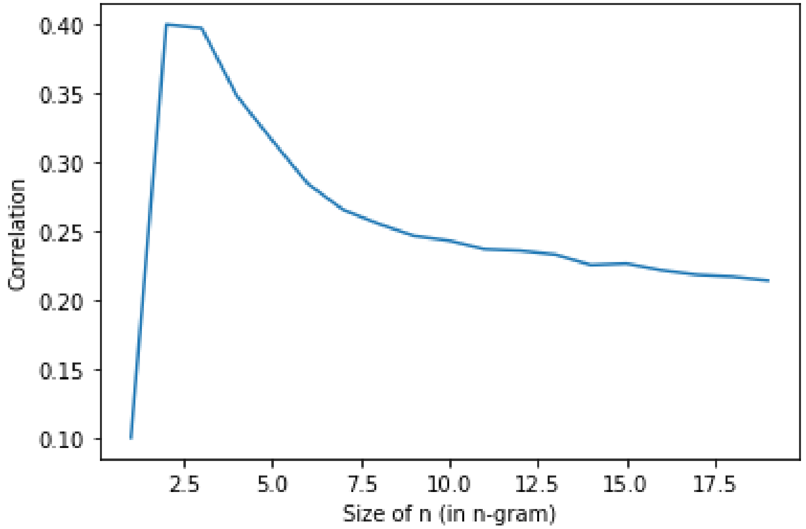
Experiments

Dataset -> 61k , SmartCheck

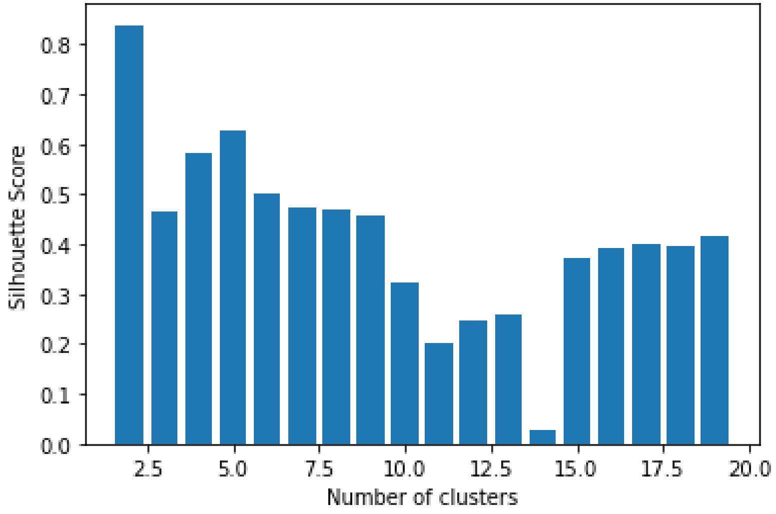
S.No	Basis	Distance Metric	Correlation (all P-val < 0.001)	Clustering perf. (Silhouette)
1	Source code, N-grams	Jaccard	0.256	-0.0301
2	Source Code	Edit Distance	0.573	0.626
3	Bytecode, N-grams	Jaccard	0.399	0.471
4	Bytecode	Edit Distance	0.136	0.839

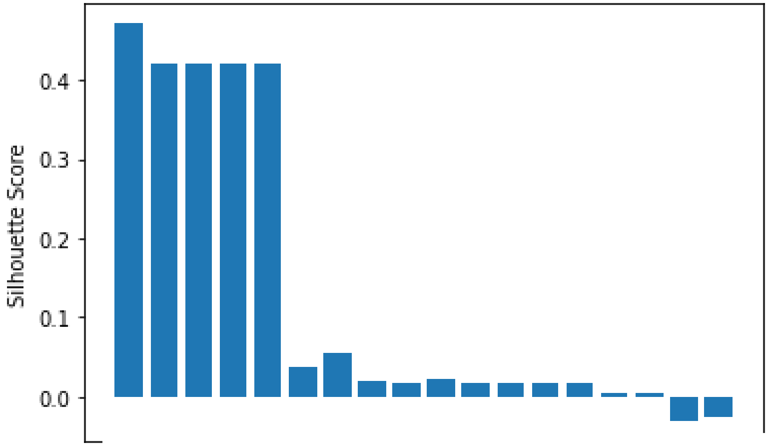
Bytecode

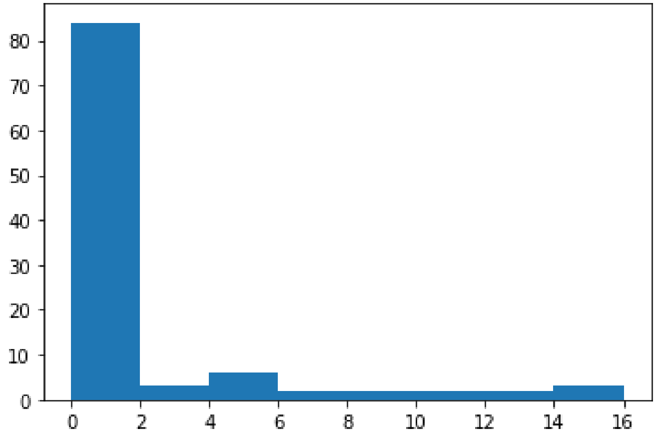
Jaccard



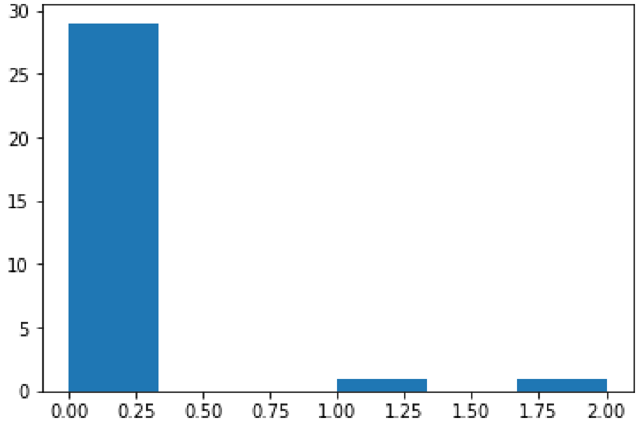
Edit Distance





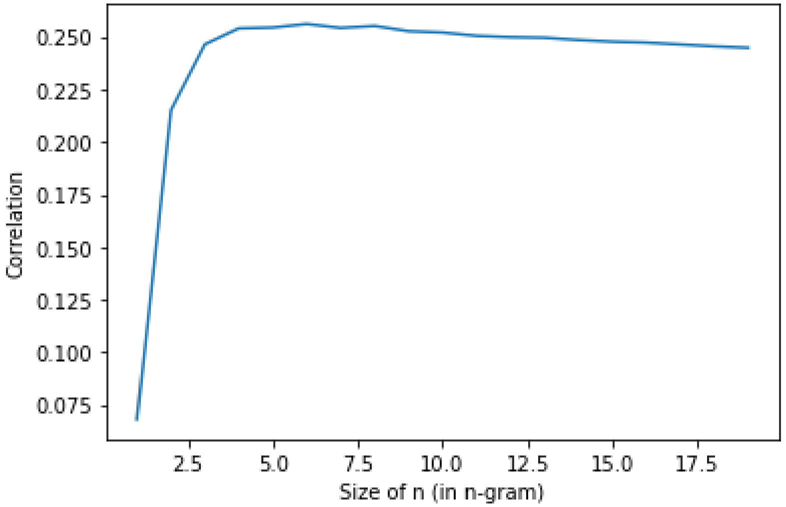


<-Histogram of the cluster ->

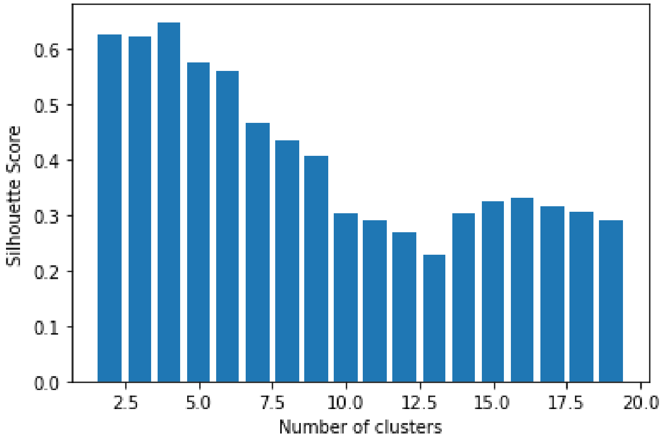


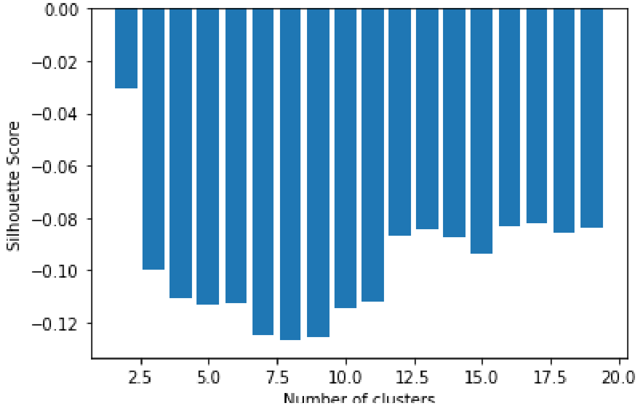
Source Code

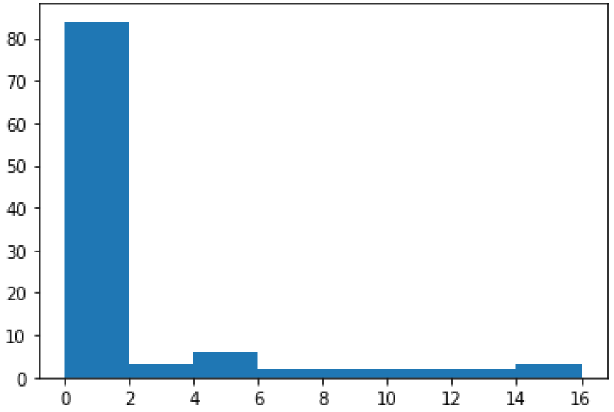
Jaccard



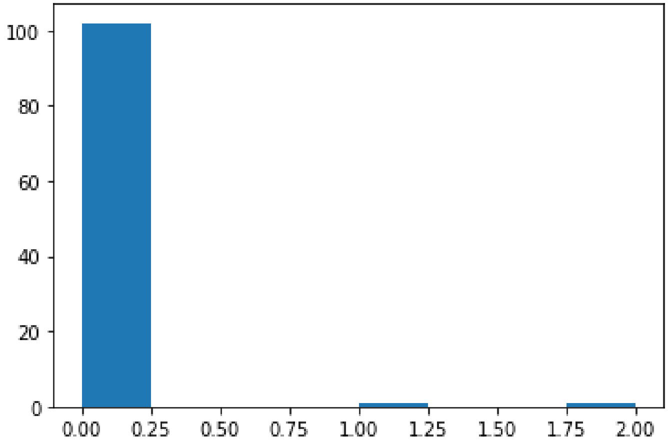
Edit Distance







<-Histogram of the cluster ->



Key Takeaways

• Clustering on the basis of bytecode is more effective than doing so on the basis of plain syntax. Currently, it seems that using the edit distance metric
on bytecode of the smart-contract is the most efficient distance metric.
• Better correlation with vulnerability always does not lead to better clustering
(as can be seen from the Silhouette scores).

Edit Distance on the smart contract source code has the best correlation with vulnerability.
• Edit Distance on smart contract bytecode results in the best clusterin

Future Scope

- Problems with SmartCheck. Need better tools to evaluate ground truth.
- Also, the smart contracts in the wild are heavily unbalanced which has been noticed by earlier studies well (this might affect the validity of our conclusions)

Table 7.1: Accuracy of tools on SmartBugs Curated Dataset

Category	**SmartCheck**	**Slither**	**SmartEmbed**
Re-entrancy	60%[9/15]	100%[15/15]	0%[0/15]
Integer Overflow	6%[1/15]	6%[1/15]	6%[1/15]
Bad PRNG	0%[0/8]	25%[2/8]	25%[2/8]
All	26%[10/38]	47%[18/38]	8%[3/38]
Popular Attacks	**SmartCheck**	**Slither**	**SmartEmbed**
DAO Example	Yes	Yes	No
Spankchain Example	No	Yes	No

Table 6: Total number of detected vulnerabilities by each tool, including vulnerabilities not tagged in the dataset.

Category	HoneyBadger	Maian	Manticore	Mythril	Osiris	Oyente	Securify	Slither	Smartcheck	Total
Access Control	0	10	28	24	0	0	6	20	3	91
Arithmetic	0	0	11	92	62	69	0	0	23	257
Denial of Service	0	0	0	0	27	11	0	2	19	59
Front Running	0	0	0	21	0	0	55	0	0	76
Reentrancy	0	0	4	16	5	5	32	15	7	84
Time Manipulation	0	0	4	0	4	5	0	5	2	20
Unchecked Low Level Calls	0	0	4	30	0	0	21	13	14	82
Unknown Unknowns	5	2	25	32	0	0	0	28	8	100
Total	5	12	76	215	98	90	114	83	76	769