Vulnerability detection in smart contracts through clone detection

Methodology:

Data collection:

The way data is collected will depend on the granularity level; for code-level analysis (analyzing a smart contract wholistically), can simply use the hugging face dataset, since it is already curated and labelled.

For function-level analysis, after collecting smart contract, whether through the hugging face dataset or crawling etherscan, extract the functions from the smart contracts, and analyze them using the slither tool to give them a label.

Reason for choosing slither analysis tool: it is the most accurate open-source static analysis tool for vulnerability detection.

Cons: slither has a false positive rate of 90%, so there may be some false classifications.

Potential alternatives:

- focus solely on clone detection (no vulnerability detection)
- Use SolidityScan. SolidityScan was the best tool found (false positive rate of 5%) for vulnerability detection. Sign up for free trial, and pay for membership ($30/month) if needed for a longer time

Data Preprocessing:

Important definitions:

- AST path: a path from one terminal node in the AST to another terminal node
- Important path: here, an "important" path in an AST is defined as how relevant a path is to the final output of a program. The more important a path is, the more relevant the code snippet that it represents

Preprocessing:

In order to train the model, we will need a vector representation of the code snippets to be able to train the model (see "Model training" section below). Here is the proposed approach, inspired by the code2vec paper:

For each code snippet

1. Parse the code into an abstract syntax tree

2. Randomly sample multiple paths from the tree.
3. For each path, embed it into a vector. A path's embedding is defined as follows:
    a. Get the word embedding of the source terminal node using the word2vec algorithm
    b. Get the word embedding of each node traversed on the path to the destination terminal node, and concatenate them into one vector
    c. Get the word embedding of the destination terminal node
    d. Concatenate the vectors obtained in a, b and c into one vector
4. Form a matrix, where each row is the vector of a path
5. Use the attention-mechanism to extract the most important paths in the matrix
6. Compress the output matrix of the attention-mechanism into one row, to obtain the final vector embedding of a code snippet

Model Training:

- Choice of model: Siamese neural network
    - Siamese neural networks have many applications, such as face recognition and signature verification
    - Because of its structure and the way it is designed, it is very well-suited for tasks that involve similarity checking
- Model structure:

    In essence, a Siamese neural network works by first forming triplets of nodes, called the positive, anchor, and negative nodes, where the model should be trained in a way such that it predicts that the anchor and positive nodes are similar, while the anchor and negative nodes are dissimilar.

    Triplet extraction:
    When forming triplets, it is important NOT to choose them randomly, because, if the anchor-positive similarity is already high, while the anchor-negative one is already low to begin with, then the model will not train properly, since it has already reached its goal.

    Therefore, main point is to choose them in a way such that they are "hard" to train, which in this case, means that their similarity calculations are approximately the same in the beginning.

    To facilitate this process of extraction, it is common for the same piece of data to be used multiple times, such that it can get picked as a positive node in a triplet set.

- Train-test split: 80%-20%
- Hyperparameters:
  - Margin: this will be used when calculating the loss in gradient descent
  - Similarity threshold in extraction of triplet sets

Model evaluation:

Calculate the testing accuracy of the model, and compare it with other models, both in clone and vulnerability detection.

Diagram of workflow: