NOVETTA

SACS: Syntax-Agnostic Code Similarity

January 9, 2020
Lara Dedic
ldedic@novetta.com

Vulnerability Research

Challenges

- Manual investigations into codebases to find recurring vulnerabilities can take 3-4 weeks to complete with multiple analysts
- The only automated solution available is based on exact matches
- Most solutions are syntax-based

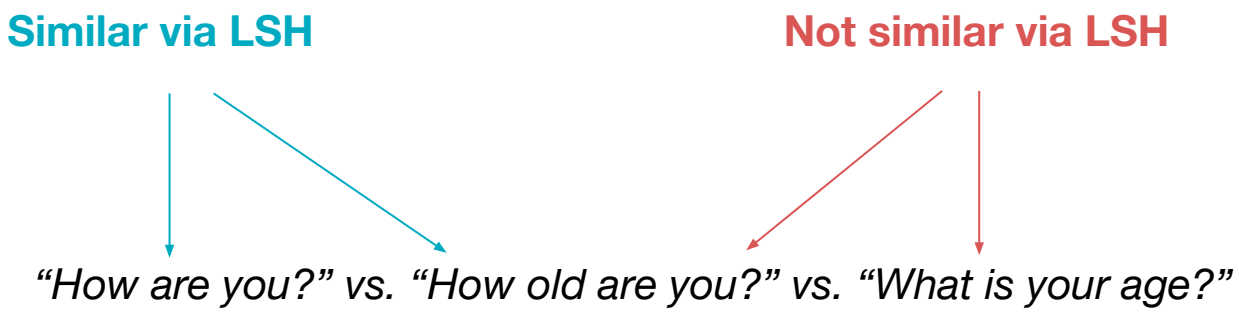
LSH for Duplicate Detection

Mining data for *textually* similar items is a fundamental big data problem

- Difficult to avoid pairwise comparisons: $O(n^2)$
- Locality Sensitive Hashing (LSH) is a solution to this: O(log n)
- LSH finds near-duplicate documents at scale

LSH is used in multiple domains:

- GoogleAI: used LSH for large-scale image searching tool, VisualRank
- Uber Engineering: used LSH to find similar trips in detecting fraudulent drivers
- Kaspersky Labs: Detecting new modifications in malware without executing it
- Plagiarism detection
- Multimedia (audio / video / text) searching
- Fingerprint comparison



Technical Challenge: Limitations of LSH

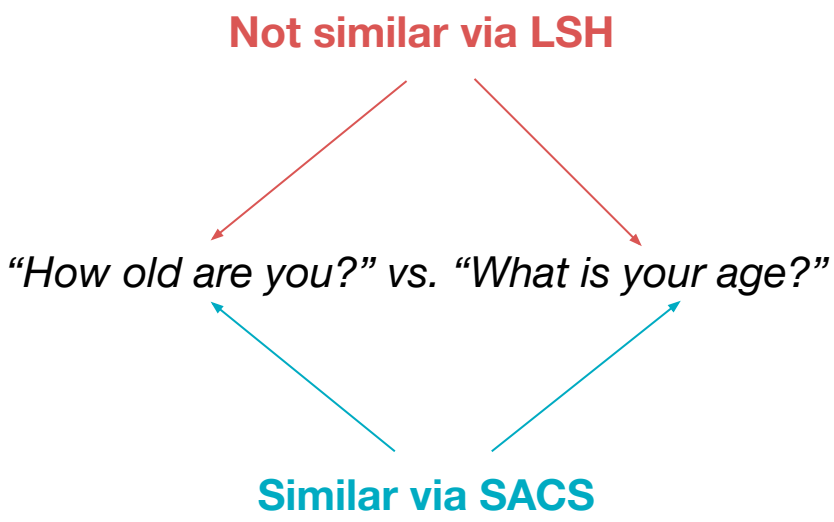
LSH is quite good at finding textually similar documents

- However, we're interested in finding *semantically* similar code

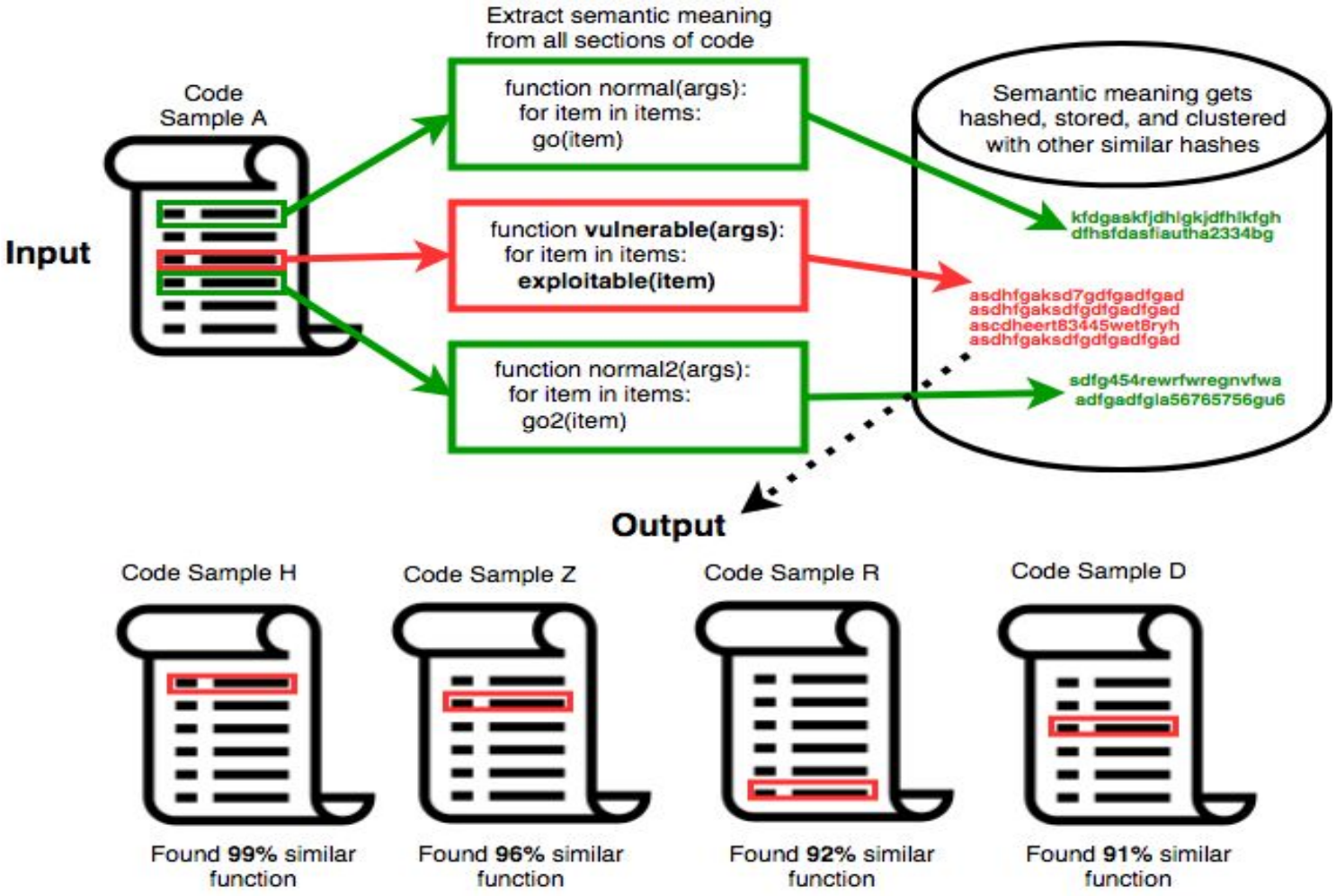
Disassembled executables can look different even if they have the same functionality

- Architectural differences
- Compilation optimizations

LSH, on its own, won't meet customer needs

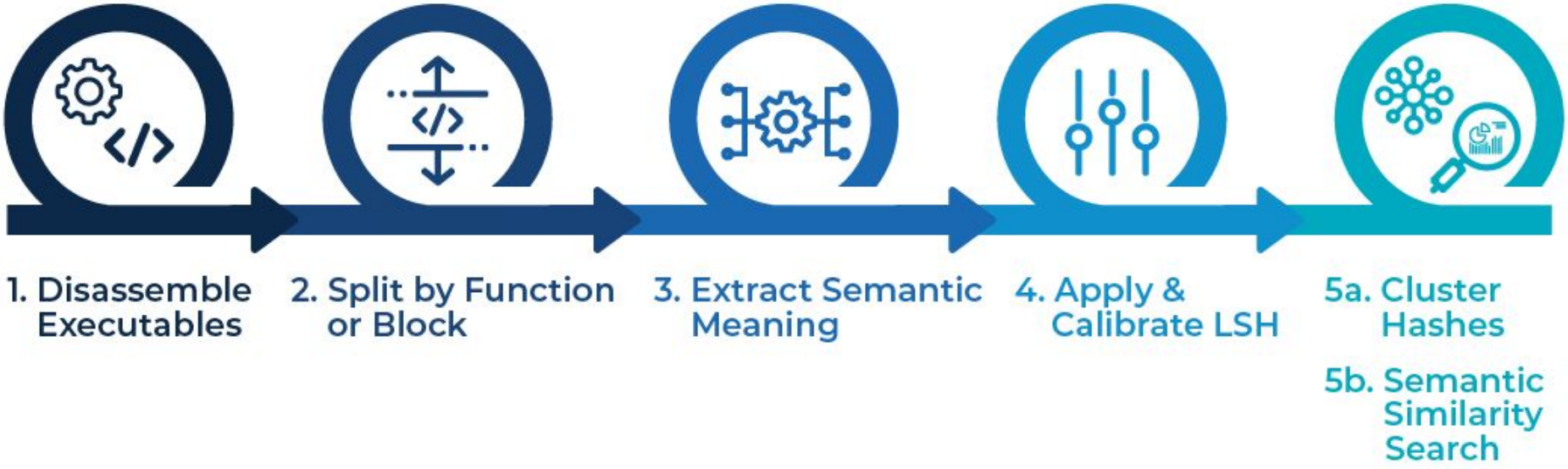


Syntax-Agnostic Code Similarity (SACS)



This type of hashing allows for **exact** and **similar** sections of code to be detected.

SACS Overview



1. Disassemble Executables

Convert executables from machine code to assembly

- Different assembly languages exist for each CPU architecture
- angr
  - Free, open source binary analysis tool
  - Disassembly is one of the many functionalities angr has

2. Split by Function or Block

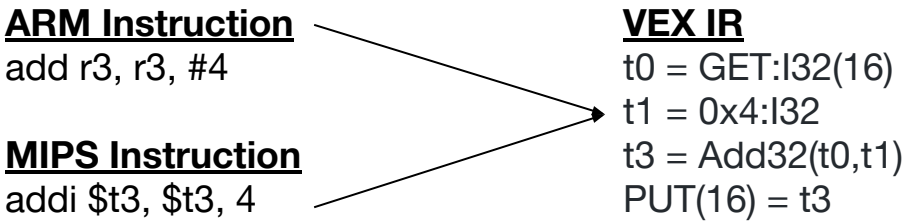
User can choose to split by function or block before ingestion begins

- angr used to extract functions and blocks
  - control flow graphs
- basic block
  - code sequence
  - one entry, one exit
- functions can contain multiple blocks

3. Extract Semantic Meaning

Semantic meaning from assembly code is extracted using Intermediate Representation (IR)

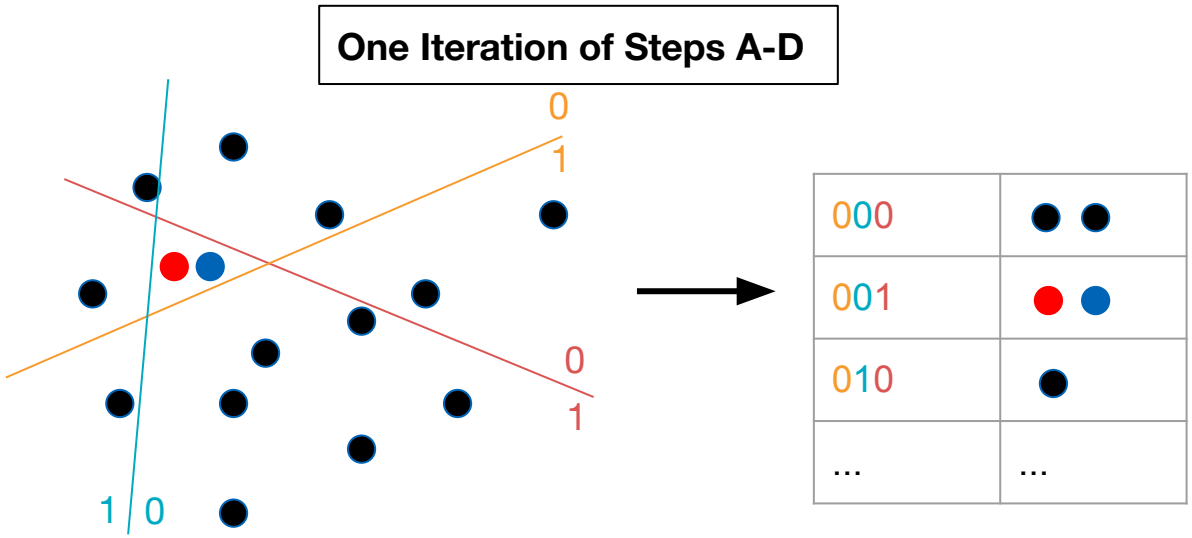
- Abstract machine language
- Originally used for compilers
  - compiler internal representation between source language and machine code
  - machine independent code
- Known as the semantic representation of assembly code
- Angr uses VEX for the IR
  - According to angr, VEX is a “architecture-agnostic, side-effects-free representation of a number of target machine languages”



4. Apply LSH

Objective: Similar Documents -> Similar Hashes

A. Represent documents in n-dimensional vector space based off n words or grams
B. Generate random hyperplanes to cut the space, group documents together
C. Generate hash of each document based on location relative to hyperplanes
D. Store hash as key and document id as the value in a hash table
E. Repeat 1-4 multiple times to reduce the likelihood of similar documents not being grouped together by increasing hash collisions



4. Calibrate LSH

Two documents are "similar" if they have the same hash value in any hash table generated

- But how similar are they?
- We need to minimize the likelihood that a non-similar document also gets the same hash

"Dials" in LSH can be adjusted so that:

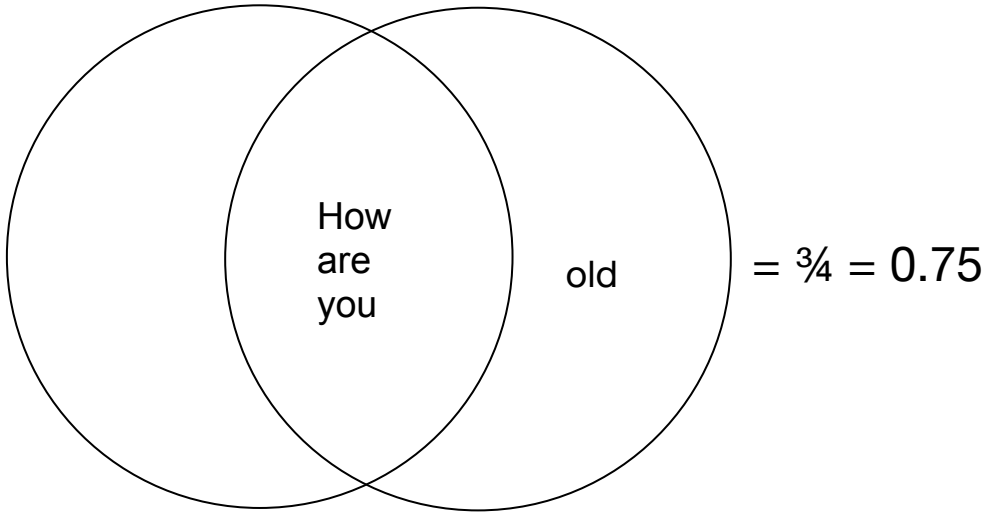
- Documents with the same hash value have a similarity threshold we desire
- False positive rates are minimized
- Dials include:
  - The vector space in which we plot our documents
    - Dictated by how we split our documents (n-grams)
  - How many hyperplanes are added
  - How many hash tables we generate
  - etc.

4. LSH Calibration Continued

How do we define “similar”?

- Approximation of Jaccard Similarity
- Jaccard (A,B) = $\frac{|A \cap B|}{|A \cup B|}$

Jaccard(“How are you”, “How old are you”)



5. Querying

5a. Cluster Hashes

- Cluster documents by a given threshold of similarity
- Cluster exploration, rather than having a prior code segment of interest
- Cluster Labeling
  - User can label clusters based on the functionality of one document
  - Use this labelled data for other tasks
    - Deep Learning: Code Summarization

5b. Semantic Similarity Search

- Top-k query
- Retrieve the most k-similar items of a code segment of interest

Unique Aspects of SACS

Why SACS is unique:

- Semantic search
- No labelled data needed
- No training required
- No need for domain expert for implementation
- No need for manual inspection of code

Applications

Data Engineering

- Reduce the time needed to investigate functionally similar code
- Autonomous code organization by functionality
- Scalable Data Labeling: knowledge of only one code fragment is sufficient to label the entire parent cluster

Autonomous Cyber Defense

- Allows for autonomous technical and logical vulnerability detection
- Can be the vulnerability detection component of an automatic code patching pipeline
- Encourages reuse of patched/secure code, rather than re-implementing a functionality
- Reduces time, effort, cost of debugging and maintaining software security

Summary

- SACS = IR + LSH
- SACS stores the semantic meaning of code
- SACS enables multiple approaches to quickly query executables on a big data scale
- Applications include data engineering and autonomous cyber defense