How Machine Learning Is Solving the Binary Function Similarity Problem

Andrea Marcelli¹, Mariano Graziano¹, Xabier Ugarte-Pedrero¹, Yanick Fratantonio¹, Mohamad Mansouri² and Davide Balzarotti²

¹ Cisco Systems, Inc.
² EURECOM

31st USENIX Security Symposium, Aug. 2022
The binary function similarity problem

Binary function similarity is the problem of taking as input the binary representation of a pair of functions, and producing as output a numeric value that captures the similarity between them.

Two functions are similar if they are compiled from the same source code.

Several practical applications (e.g., reverse engineering, detect vulnerabilities, malware clustering).
Researchers have published an astonishing number of papers:

- Different communities publish in different venues
- The field is extremely fragmented

Main challenges:

- Inability to neither reproduce nor replicate previous results
  - Artifacts are often incomplete or incorrect
- Evaluation results are often opaque
  - Different objectives, settings, concept of similarity, and granularities
  - Different datasets, metrics, and implementations

It is unclear in which direction binary similarity research is heading and why.
Our contribution

Perform the first measurement study on the state of the art of binary function similarity:

- We explore existing research and group each solution based on the adopted approach
  - We particularly focus on recent successful techniques based on machine learning

- We select, compare, and implement the ten most representative approaches and their variants
  - Approaches vs. papers
  - Our implementations are built on top of a common framework

- We build a new dataset that we use as a common benchmark.
Measuring function similarity

Direct vs. indirect comparison:

- Direct comparison methods always require two functions in input
- Indirect comparison methods map the input features to a low-dimensional representation

Indirect comparison methods include:

- Fuzzy hashes intentionally map similar input values to similar hashes
- Code embeddings take in input a stream of tokens (e.g., Word2vec, Seq2seq, Transformer)
- Graph embeddings take in input the function control flow graph (e.g., Graph Neural Network).
Function representations

- Raw bytes
- Assembly
- Normalized assembly
- Intermediate Representations

- Control Flow Graph
- Annotated Control Flow Graph

- Data flow analysis
- Dynamic analysis
- Symbolic execution and analysis

higher abstraction
Implementation and Dataset

We implemented each phase of the evaluation in an uniform way:

- IDA Pro 7.3 for the **binary analysis**, IDA Pro scripts for the **feature extraction**
- Tensorflow 1.14 for the **neural network models** in (except for Trex)
- Gensim 3.8 for the **word2vec models** (including Asm2vec)

We created two new datasets, Dataset-1 and Dataset-2. They include:

- Multiple compiler families and versions (GCC and Clang)
- Multiple compiler optimizations (O0, O1, O2, O3, and Os)
- Multiple architectures and bitnesses (x86-64, ARM, and MIPS, in 32 and 64 bit versions)
- Software of different nature (command line utilities vs. GUI applications)
Experiments

Selected approaches are evaluated on six tasks: \( \text{XO, XC, XC+XB, XA, XA+XO, XM} \)

- E.g., \( \text{XC} \): the function pairs have different compiler, compiler versions, and optimizations, but same architecture and bitness.

Each task is evaluated according to two different tests:

- The area under curve (AUC)
- The mean reciprocal rank (MRR) and the recall (Recall@K) at different K thresholds

The function pair selection aspect is crucial for a proper evaluation.
Results

Recall at different K for different models on the XM task (Dataset-1)
Results

Recall at different K for different models on the XM task (Dataset-1)
Main contribution of the novel machine learning solutions

Deep-learning models provide an effective way of learning a function representation.

The Siamese architecture in combination with a margin based loss introduced significant improvements.

GNNs are effective encoders and can be used in combination with others.
Takeaways (2)

The role of different set of features

Using basic block features (e.g., ACFG) provides better results

Minimal difference between manually engineered features and simpler ones (e.g., BoW opcodes)

Dataflow information can boost the results, especially for large functions.
Takeaways (3)

**Cross-architecture vs single architecture**

Most of the machine-learning models perform very similarly on all the evaluated tasks.

**Training on generic task data** allows to achieve performances close to the best for each task.

Asm2Vec and Catalog1 are limited to comparisons to a single architecture.
Takeaways (4)

Future directions of research

GNN models provide the best results, but there are tens of different variants that need to be tested.

Combining GNNs with intermediate representations and data flow information must be studied.

Training strategy and loss functions have been barely discussed in the past and only recently explored.
Thank You

anmarcel@cisco.com

https://github.com/Cisco-Talos/binary_function_similarity/