Asm2Vec: Boosting Static Representation Robustness for Binary Clone Search against Code Obfuscation and Compiler Optimization

Steven H. H. Ding†, Benjamin C. M. Fung∗, and Philippe Charland†

†Data Mining and Security Lab, School of Information Studies, McGill University, Montreal, Canada.
Email: steven.h.ding@mail.mcgill.ca, ben.fung@mcgill.ca

‡Mission Critical Cyber Security Section, Defence R&D Canada - Valcartier, Quebec, QC, Canada.
Email: philippe.charland@drdc-rddc.gc.ca

Abstract—Reverse engineering is a manually intensive but necessary technique for understanding the inner workings of new malware, finding vulnerabilities in existing systems, and detecting patent infringements in released software. An assembly clone search engine facilitates the work of reverse engineers by identifying those duplicated or known parts. However, it is challenging to design a robust clone search engine, since there exist various compiler optimization options and code obfuscation techniques that make logically similar assembly functions appear to be very different.

A practical clone search engine relies on a robust vector representation of assembly code. However, the existing clone search approaches, which rely on a manual feature engineering process to form a feature vector for an assembly function, fail to consider the relationships between features and identify those unique patterns that can statistically distinguish assembly functions. To address this problem, we propose to jointly learn the lexical semantic relationships and the vector representation of assembly functions based on assembly code. We have developed an assembly code representation learning model Asm2Vec. It only needs assembly code as input and does not require any prior knowledge such as the correct mapping between assembly functions. It can find and incorporate rich semantic relationships among tokens appearing in assembly code. We conduct extensive experiments and benchmark the learning model with state-of-the-art static and dynamic clone search approaches. We show that the learned representation is more robust and significantly outperforms existing methods against changes introduced by obfuscation and optimizations.

1. Introduction

Software developments mostly do not start from scratch. Due to the prevalent and commonly uncontrolled reuse of source code in the software development process [1], [2], [3], there exist a large number of clones in the underlying assembly code as well. An effective assembly clone search engine can significantly reduce the burden of the manual analysis process involved in reverse engineering. It addresses the information needs of a reverse engineer by taking advantage of existing massive binary data.

Assembly code clone search is emerging as an Information Retrieval (IR) technique that helps address security-related problems. It has been used for differing binaries to locate the changed parts [4], identifying known library functions such as encryption [5], searching for known programing bugs or zero-day vulnerabilities in existing software or Internet of Things (IoT) devices firmware [6], [7], as well as detecting software plagiarism or GNU license infringements when the source code is unavailable [8], [9]. However, designing an effective search engine is difficult, due to varieties of compiler optimizations and obfuscation techniques that make logically similar assembly functions appear to be dramatically different. Figure 1 shows an example. The optimized or obfuscated assembly function breaks control flow and basic block integrity. It is challenging to identify these semantically similar, but structurally and syntactically different assembly functions as clones.

Developing a clone search solution requires a robust vector representation of assembly code, by which one can measure the similarity between a query and the indexed functions. Based on the manually engineered features, relevant studies can be categorized into static or dynamic approaches. Dynamic approaches model the semantic similarity by dynamically analyzing the I/O behavior of assembly code [10], [11], [12], [13]. Static approaches model the similarity between assembly code by looking for their static differences with respect to the syntax or descriptive statistics [6], [7], [8], [14], [15], [16], [17], [18]. Static approaches are more scalable and provide better coverage than the dynamic approaches. Dynamic approaches are more robust against changes in syntax but less scalable. We identify two problems which can be mitigated to boost the semantic richness and robustness of static features. We show that by considering these two factors, a static approach can even achieve better performance than the state-of-the-art dynamic approaches.

PI: Existing state-of-the-art static approaches fail to consider the relationships among features. LSH-S [16], n-gram [8], n-perm [8], BinClone [15] and Kam1n0 [17] model assembly code fragments as frequency values of operations and categorized operands. Tracelet [14] models assembly code as the editing distance between instruction sequences. Discovre [7] and Genius [6] construct descriptive features, such as the ratio of arithmetic assembly instructions, the number of transfer instructions, the number of basic blocks, among others. All these approaches assume each feature or category is an independent dimension. However, a xmm0 Streaming SIMD Extensions (SSE) register is related to SSE operations such as movaps. A felen0 libc function call is related to other file-related libc calls such as fopen. A strepy libc call can be replaced with memcpy. These relationships provide more semantic information than
individual tokens or descriptive statistics.

To address this problem, we propose to incorporate lexical semantic relationship into the feature engineering process. Manually specifying all the potential relationships from prior knowledge of assembly language is time-consuming and infeasible in practice. Instead, we propose to learn these relationships directly from plain assembly code. Asm2Vec explores co-occurrence relationships among tokens and discovers rich lexical semantic relationships among tokens (see Figure 2). For example, `memcpy`, `strcpy`, `memncpy` and `mempcpy` appear to be semantically similar to each other. SSE registers relate to SSE operands. Asm2Vec does not require any prior knowledge in the training process.

P2: The existing static approaches assume that features are equally important [14], [15], [16], [17] or require a mapping of equivalent assembly functions to learn the weights [6], [7]. The chosen weights may not embrace the important patterns and diversity that distinguishes one assembly function from another. An experienced reverse engineer does not identify a known function by equally looking through the whole content or logic, but rather pinpoints critical spots and important patterns that identify a specific function based on past experience in binary analysis. One also does not need mappings of equivalent assembly code.

To solve this problem, we find that it is possible to simulate the way in which an experienced reverse engineer works. Inspired by recent development in representation learning [19], [20], we propose to train a neural network model to read many assembly code data and let the model identify the best representation that distinguishes one function from the rest. In this paper, we make the following contributions:

- We propose a novel approach for assembly clone detection. It is the first work that employs representation learning to construct a feature vector for assembly code, as a way to mitigate problems P1 and P2 in current hand-crafted features. All previous research on assembly clone search requires a manual feature engineering process. The clone search engine is part of an open source platform.
- We develop a representation learning model, namely Asm2Vec, for assembly code syntax and control flow graph. The model learns latent lexical semantics between tokens and represents an assembly function as an internally weighted mixture of collective semantics. The learning process does not require any prior knowledge about assembly code, such as compiler optimization settings or the correct mapping between assembly functions. It only needs assembly code functions as inputs.
- We show that Asm2Vec is more resilient to code obfuscation and compiler optimizations than state-of-the-art static features and dynamic approaches. Our experiment covers different configurations of compiler and a strong obfuscator which substitutes instructions, splits basic blocks, adds bogus logics, and completely destroys the original control flow graph. We also conduct a vulnerability search case study on a publicly available vulnerability dataset, where Asm2Vec achieves zero false positives and 100% recalls. It outperforms a dynamic state-of-the-art vulnerability search method.

Asm2Vec as a static approach cannot completely defeat code obfuscation. However, it is more resilient to code obfuscation than state-of-the-art static features. This paper is organized as follows: Section 2 formally defines the search problem. Section 3 systematically integrates representation learning into a clone search process. Section 4 describes the model. Section 5 presents our experiment. Section 6 discusses the literature. Section 7 discusses the limitations and concludes the paper.

1. https://github.com/McGill-DMaS/Kam1n0-Plugin-IDA-Pro