WiP: Assessing the Effects of Source Language on Binary Similarity Tools

Landen Doty
landoty@ku.edu
University of Kansas
Lawrence, Kansas, USA

Dr. Prasad Kulkarni
prasadk@ku.edu
University of Kansas
Lawrence, Kansas, USA

ABSTRACT

State-of-the-art binary similarity tools have been developed to account for and are evaluated against variations in compilers, compiler flags, optimization levels, architectures, and even obfuscations. Although these tools aim to measure and detect binary code segments generated from similar or identical source code segments, they have yet to be evaluated on source languages other than C/C++.

We present a **work in progress** to assess the effects of source language on modern binary similarity tools. Specifically, we provide a comparative investigation on the efficacy of BSim, a recently released component of the Ghidra framework, when comparing binaries produced by C as well as Rust. Using a benchmark of 800 binaries and more than one million functions, we investigate the overall accuracy and differentiating ability of BSim and find that the source language introduces a significant degree of imprecision not previously documented. We also provide a technical overview of the BSim utility, which provides context for our assessment results and a clear direction for addressing the shortcomings highlighted by our findings.

CCS CONCEPTS

• **Security and privacy** → **Software reverse engineering**; • **General and reference** → *Evaluation.*

KEYWORDS

Binary Code, Diffing, Similarity, Ghidra, BSim, Rust

1 INTRODUCTION

Binary code similarity is a fundamental technique to compare a pair, set, or large corpus of binary-level code segments - instruction sequences, basic blocks, functions, etc. Binary code comparison has applications in several domains, including malware classification and detection [2, 6, 10, 16], vulnerability research [8, 17, 20], and even intellectual property protection [13]. Given its real-world applicability and the growing number of programs distributed without source, binary code similarity has seen a sustained research effort to both improve existing techniques and develop novel approaches.

Early binary similarity work, such as BinDiff, leveraged structural representations of binary code, such as control flow, data flow, and call graphs [4, 5]. However, *binary*-level comparison of software components is uniquely challenging, as even small variations in the compilation environment, including the compiler, optimization levels, and various compiler flags, can result in significant changes in the generated machine code. As such, more recent approaches attempt to capture semantic understanding of binary code segments using traditional program analysis techniques such as symbolic execution and fuzzing, and have shown improved results across variations of compiler settings [7, 15, 20]. Further, the use of machine learning and data science techniques - natural language processing (NLP), approximate nearest neighbor, and deep learning - has shown promise in improving the state of the art [3, 14, 19].

Despite the vast number of research endeavors focused on this area, to our knowledge, no current work has investigated how *source*-level constructs affect the efficacy of *binary*-level similarity techniques. With the growing adoption of non-C/C++ high-level compiled languages, practitioners will certainly see an increasing amount of binary artifacts compiled from richer high-level abstractions than those seen in C/C++. Thus, if the tools and frameworks available are overfit to patterns most commonly seen in C/C++ binaries, practitioners may be at a distinct disadvantage when comparing binaries from languages like Rust and Golang.

This paper presents a first in-depth analysis of the effects of source language on binary code similarity techniques. With a specific interest in tools readily available to binary analysis practitioners, we conduct our investigation on *BSim*, a recently open-sourced binary similarity tool in the Ghidra Software Reverse Engineering Framework developed by the National Security Agency (NSA). As a **Work In Progress** submission, this paper includes 1) a baseline analysis of BSim's performance on C/C++ binaries, 2) a comparative analysis on *Rust* binaries and 3) early results and discussion towards addressing the observed effects.

Beyond this investigation, this paper also includes a high-level technical description of BSim, its components, and configurations resulting from our analysis of its open-source content. This information is important in contextualizing the results of our assessments and provides future work with a baseline of technical understanding.

In summary, this work makes the following contributions:

- We provide a novel investigation of the effects of source language on binary similarity. Using Rust as our experimental language, we show that an existing tool, BSim, is tuned to features seen in C-based languages and its performance is degraded when comparing binaries not of C origin.
- We detail the internal components of BSim, its feature representation, and available configuration and tuning capabilities. This contribution results from our manual analysis of BSim's released source code as well as extensive work using BSim to facilitate our assessment of source language effects.