Deep Learning Approaches for Identifying Compiler and Optimization Options from Binary Code.

Mistry Unnat, 20BCE515

Student

Computer Science and Engineering Department

Nirma University, A’bad, India

Neel Prajapati, 20BCE518

Student

Computer Science and Engineering Department

Nirma University, A’bad, India

Prof. Tarjni Vyas

Computer Science and Engineering Department

Nirma University, A’bad, India

*Abstract*—When compiling a source file, several flags can be passed to the compiler. These flags, however, can vary between debug and release compilation. In the release compilation, in fact, smaller or faster executables are usually preferred, whereas for a debug one, ease-of-debug is preferred over speed and no optimization is involved. After the compilation, however, most of the flags used cannot be inferred from the compiled file. These flags could be useful in case we want to classify if an older build was made for release or debug purposes, or to check if the file was compiled with flags that could expose vulnerabilities. In this paper we present a deep learning network capable of automatically detecting, with function granularity, the compiler used and the presence of optimization with 99% accuracy. We also analyze the change in accuracy when submitting increasingly shorter amounts of data, from 2048 up to a single byte, obtaining competitive results with less than 100 bytes. We also present our process in the huge dataset creation and manipulation, along with a comparison with other less successful networks using functions of varying size. Index Terms—Static Analysis, Binary Analysis, Deep Learning, Compilers

Keywords—Static Analysis, Binary Analysis, Deep Learning, Compilers

# Introduction

During the software development life-cycle of a natively compiled application, the process of converting source code to binary code performed by a compiler happens quite often. While performing this transformation, several flags are given to the compiler, signaling the developer’s intention to keep or drop some information or to modify the original code in a more optimized version. These flags can be used to optimize towards faster executables, smaller size or lower energy consumption. However, they are not explicitly recorded in the binary file itself as they are completely unnecessary by the machine in order to execute the binary code.

Moreover, also the compiler itself is not easy to identify. There is no standard way to record this information, and although some compilers write a comment in the binary itself, this is easily patchable and not guaranteed to be parsable. In fact, for example, if a file compiled with the clang compiler is linked with a library compiled with gcc, the comment will contain both signatures