Fuzzing the C Compiler: Discovering Modern Syntactic and Semantic Errors

Names: Jarod Keene and Logan Stratton

Background of the Problem and Motivations:

There has been a great deal of defensive progress against the front of exploitation. Modern exploitative techniques are difficult to discover and often short lived. This has encouraged the development of robust fuzzing frameworks. With growing code bases, and complexifying backends, emulation and fuzzing have allowed for experts to examine a much greater attack surface area in a fraction of the time. However, most modern fuzzing models are interested only in runtime examination. This priority leaves a gap of understanding within the build process, a gap of understanding that we hope to fill.

We believe that there is a great deal of possible environment vulnerability that has yet to be examined. Within the C programming language, there are many dangerous syntactic and semantic decisions that the programmer can make, and we believe the freedom allows the compiler to do things with a level of uncertainty, and we hope to to discover where those limits are found and how they can be utilized.

By bypassing the syntactic and semantic crossroad malicious applications appear, such as, subverting execution in a large codebase. We have seen how C code can be used improperly with confused execution happening through side challenges called codegolf. Where the purpose is to create a program utilizing the least number of characters or instructions possible, these programs are syntactically and semantically correct but are disturbing in appearance. With fuzzing, we create syntactically correct code but is awkward in the semantic sense. Utilizing flaws, such as these in the compiler logic, may result in a compiled program with hidden backdoors for attackers to use in the future.

Objective:

In this work, we aim to explore the possibilities of generating convoluted code. Through the use of Damato, a DOM focused fuzzer**,** we expect to find logic flaws in the implementation of certain compilers such as GCC and Clang. Damato is also very versatile, and only requires a working Context Free Grammar (CFG) for a given language to begin to fuzz against that language’s compiler/interpreter. With this in mind, if successful in finding flaws within C compilation, we hope to extend our research to other domains (Rust, Go, etc). We will also create a possible malicious example of bypassing any protections that the compiler may implement. Our findings will then be used to fix the errors and create a possibly safer future!