CMSC430 Project Proposal

Foreign Function Interfaces (Lawbreaker)

Garrett Alessandrini & Nicholas Romano

Core Idea

In many languages, users can call functions that are defined using a programming language different from the one they are programming in. Loot currently has a foreign function interface for read_byte, peek_byte, and write_byte. However, this foreign function interface is hard-coded to match a symbol representing each function name. This project hopes to allow the user to call user-defined functions in C. By calling user-defined functions in C, our language will be able to support functionality that has not been implemented. Furthermore, we will demonstrate how using a foreign function interface can lead to security vulnerabilities. This is possible because the functions that are being called are written in C and can potentially modify memory, which was previously not possible in our langauge. See **Demonstrations of Security Vulnerabilities** for more info.

What needs to change from Loot/Expected Operation

- Syntax/AST
    - Addition of a new callable operation
        - (ccall name xs)
            - Name represents the C function that will be called
            - Xs represents a list containing our arguments
                - If Xs is the empty list, no arguments will be passed
                - To keep a reasonable scope, only Integers, Booleans, and Characters will be supported
            - Arguments will be transformed and organized on the stack/registers according to the SystemV calling convention and C type representations to be compatible with the C runtime
            - After the foreign function returns a value, we can convert the resulting C value back to our own language's representation
- Compile-Time
    - Generation of a dynamic extern list
        - Currently, we have a hand-written list of external symbols added to the top of our generated assembly code to allow function calls for read_byte/write_byte/peek_byte to link successfully

        - Instead, we will dynamically build a list of externs based on the final AST, with our three existing external calls appended manually
- Runtime
    - Functions on the C side will have to accept a variable number of val_t arguments, one for each they require
    - Then, at the beginning of the function, all val_t arguments must be unwrapped into what their respective types should be using our existing type functions in C
    - After completing its task, the function should return the final value as a val_t (the final value should be wrapped to be our own representation)
        - To keep a reasonable scope, only Integers, Booleans, Characters, and Void will be supported as return values
- Interpreter
    - Due to C functions needing to be linked separately, we will not be implementing this functionality for the interpreter

Demonstration of Security Vulnerabilities

- By providing users with a way to call foreign functions written in C, the ability to write to the stack/heap in ways not previously possible is introduced
- We will create example programs that can showcase this behavior

Timeline

1. To start, create a basic version of our CCall within the AST that can call C functions without worrying about arguments, but can process a return value
2. Add the ability to provide a varying number of arguments to our C calls
3. Create demonstrations of security vulnerabilities introduced into our language due to this new functionality

Related Resources

- SystemV Specification
    - https://www.uclibc.org/docs/psABI-x86_64.pdf
    - Data Representation is in section 3.1.2
    - Calling Convention is in section 3.2 / A.2.1
        - Parameter passing is section 3.2.3
    - Example assembly code for function calls is section 3.5.5