Contents

1 Introduction

Extensible programming languages were first discussed in the 1960s. As originally discussed in Douglas McIlroy's 1960 paper, the original concept was to use a small number of macros in order to extend a compiler to accept very general extensions to the original language accepted by the compiler (CITE McIlroy's paper). This has been modernized in various ways, including the idea of extensible syntax, compilers, and even extensible runtimes (CITE? wiki has dead link).

One example of an extensible programming language is `ABLE`C (cite), an extensible version of the C programming language, built on top of the `Silver` (cite) attribute grammar system. Various extensions exist for `ABLE`C that were created to solve various problems (CITE paper containing extensions from MELT).

This paper examines the uses of extensible programming languages in a real-world application: an Internet Relay Chat (IRC) server implementation. The IRC protocol was originally designed for text-based conferencing (cite IRC RFC), and many server implementations are available. For this paper, we consider the specific implementation of `ngIRCd` (next-generation IRC daemon), written in the C programming language. This open-source server allows for end users to clone the server and modify it to suit their specific purposes. These modifications, however, open the server up to various security vulnerabilities. The C programming language, while fast, does not provide many security checks for the programmer. We desire a way of allowing users to modify the server without introducing any security vulnerabilities.

Modifying this C code provides a natural use for the `ABLE`C language. By introducing both new and existing extensions to the `ngIRCd` server, we are able to both eliminate certain existing vulnerabilities in the server that are innate to the C programming language, but also we can make certain aspects of the server easier to implement and understand for end users, as well.

2 Background

This chapter provides background information for the work that follows throughout the paper. First, we discuss the existing nonnull type qualifier extension implemented as a language extension to `ABLE`C. Next, we discuss the two new extensions written for this project: an extension allowing for easier use of asynchronous IO and an extension allowing the use of the Wuffs programming language in the parsing of messages.

2.1 Nonnull Qualifiers

One of the most common sources of bugs in the C programming language is errors related to null pointers. The dereferencing of a null pointer is considered undefined behavior in C (cite ISO/IEC 9899 clause 6.5.3.2

paragraph 4, footnote 87), and is extremely undesirable in programming. We introduce an extension to `ABLEC` to deal with null pointers (referred to herein as `ABLEC`-nonnull). This extension allows for compile-time checking for any possible null dereferences of pointers.

The fact that `ABLEC`-nonnull allows for compile-time checking of possible null pointer dereferences is incredibly valuable to a programmer and allows for the avoidance of many headaches typically associated with dealing with pointers in C.

2.2 Asynchronous I/O

Asynchronous I/O is a major part of the difficulty in creating a server implemenation in C, particularly when using threads is not a viable option. We don't want the server to be stuck while we wait for it to perform some I/O operation like reading or writing to an existing connection, when the server has other tasks it could be doing, like establishing new client connections or parsing an already-received message. In C, one of the primary ways of performing asynchronous I/O is using the `epoll` API. Using this API, we are able to keep a list of file descriptors we want the current process to monitor, as well as a list of file descriptors that are ready for I/O. However, this process is extraordinarily tedious, requiring many expensive system calls to set up and maintain the `epoll` instance. When compared with other, more modern languages, the `epoll` API is both more verbose and more difficult to use. Consider a more modern language like Go or Javascript. Both of these languages have their own facilities for asynchronous operation. In Go, we use several constructs, including the `go` and `select` keywords, to implement various aspects of I/O. In Javascript, we utilize both the `async` and `await` keywords, as well as the idea of *promises* in order to achieve some measure of asynchronous operation.

2.3 Wuffs

A final extension implemented for this project involves the Wuffs language. Wuffs is a language created by Google for memory-safe handling of untrusted file formats. Originally intended for use in security-specific portions of a program, Wuffs allows for compile-time checking of various security vulnerabilities, as well as some semi-rigorous proofs of program safety.

A simple parsing program in C is shown below. The same parsing program is also shown in Wuffs as a comparison of the two languages. Both of these programs take as input a string of numbers, for example, `"123"`, and return the associated numeric value, in this case, 123.

Leaving aside the syntactic peculiarities of Wuffs, note that the Wuffs code is signficantly more verbose than the C code in the above examples. However, we consider the edge cases of parsing some string,

Figure 1: Parsing in C

```
uint32_t parse(char * p, size_t n) {
    uint32_t ret = 0;
    for (size_t i = 0; (i < n) && p[i]; i++) {
        ret = 10 * ret + (p[i] - '0';
    }
    return ret;
}
```

Figure 2: Parsing in Wuffs

```
pub func parser.parse?(src: base.io_reader) {
    var c : base.u8
    while true {
        c = args.src.read_u8?()
        if c == 0 {
            return ok
        }
        if (c < 0x30) or (0x39 < c) { // '0' and '9' in ASCII
            return "#not a digit"
        }
        //Rebase from ASCII to numeric value
        c -= 0x30

        if this.val < 429_496729 {
            this.val = (10 * this.val) + (c as base.u32)
            continue
        } else if (this.val > 429_496729) or (c > 5) {
            return "#too large"
        }

        this.val = (10 * this.val) + (c as base.u32)
    } endwhile
}
```

particularly with regards to integer overflow. In C, an integer overflow error can occur, but it does so silently. Obviously, in a real C parser implementation, we would implement security checks to cover these overflow cases. The point, however, is less about the fact that this is possible in C and more about the fact that these checks are mandatory in Wuffs. The Wuffs compiler will not allow an addition expression that could possibly overflow the maximum integer value.

This compile-time security checking for basic security vulnerabilites is incredibly useful, particularly in contexts where we are parsing unknown code or strings that could be malicious. We don't need to worry about runtime security checks when we can use Wuffs to guarantee no buffer overflows, out-of-bounds array accessing, integer overflow, or integer underflow at compile time.

3 Non-Null Pointer Extension

Information on the `ableC` non-null pointer extension here.

4 Asynchronous I/O

Information about the `ableC` Asynchronous I/O extension

5 Wuffs Parsing

Information here about the Wuffs extension. Probably also want to include details about what Wuffs is, why we're including it

6 Discussion

Discussion of each of the extensions, it's use, etc

6.1 Non-null Pointers

Non-null discussion

6.2 Asynchronous I/O

Async IO discussion

6.3 Wuffs

Wuffs discussion

7 Future Work

Discussion of future work

8 Bibliography

- https://link.springer.com/content/pdf/10.1007/3-540-45937-5_11.pdf Forwarding in attribute grammars
- https://dl.acm.org/doi/10.1145/3138224 original AbleC paper
- https://core.ac.uk/download/pdf/429260481.pdf Aaron's thesis
- https://conservancy.umn.edu/bitstream/handle/11299/203190/technical_report.pdf?sequence=1 Also Aaron's thesis?
- https://www-users.cse.umn.edu/~evw/pubs/carlson19ppopp/carlson19ppopp.pdf Tutorial for how to build extensions
- https://www.rfc-editor.org/rfc/pdfrfc/rfc1459.txt.pdf RFC for Internet Relay Chat (IRC)