Homework 6 Report:
Evaluating Zig for secure camera-based HVAC control

Carson Kim - Winter 2021

1 Abstract

For this assignment, we were tasked with researching and evaluating one of three programming languages and its utility for Haversack Inc., an HVAC-systems company. The company's latest technology involves using cameras to monitor heat signatures of a building's human occupants and adjusting air temperatures accordingly. This brief report will discuss the prospects of Zig, a fairly new imperative, general-purpose language.

2 Introduction

Zig seems to be a strong candidate for Haversack Inc.'s facial-recognition based thermostat applications. It is reminiscent of C/C++ with some differences in its type-checking, memory management system, and compile-time operations. Zig is a small, simple language that supports both high performance and safety through its four build modes. In addition, its lack of implicitness allows for "calls to be calls" and little to no opaqueness in determining variable types. It also requires manual memory management, giving more control to the programmer in how allocation and deallocation is handled. Of course, programmers may encounter difficulties when dealing with memory-failure related edge cases in their code. It is important to note that Zig does offer robust error-handling features to curb these types of problems.

3 Effects on Security

There are security concerns with the implementation of a C/C++ program, as they have certain vulnerabilities. Of course, sensitive information within such programs needs to be protected. Luckily, Zig is a cleanly written language that would be rather easy to audit. Although it is a relatively new language, it offers a robust selection of build modes. These build modes allow for the prioritization of safety over performance, or vice versa.

The 4 build modes, `Debug`, `ReleaseFast`, `ReleaseSafe`, and `ReleaseSmall` all offer varying levels of safety checks, runtime performance levels, compilation speed, binary size, and reproducible build requirements. Of course, `ReleaseSafe` would be the preferred option for the programmers at Haversack during initial development, as it guarantees more program safety over performance.

4 Ease of Use

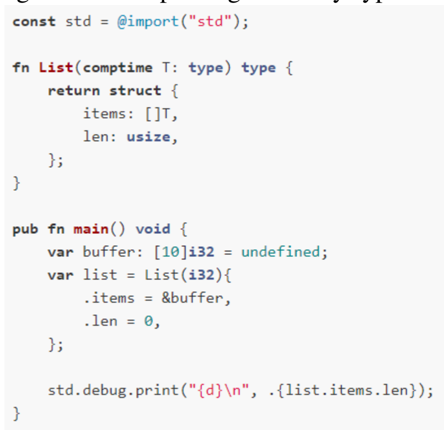
Because Zig is a relatively small, stripped-down language (especially when compared to other high-level languages such as Python and Java), it certainly meets expectations relating to ease of use. Its syntax is also more simple than C/C++ and aims for maximum readability. It also disallows hidden control flow. For example, since C++ has operator overloading, the + operator might call a function; Zig avoids this problem since its control flow is only managed with language keywords and function calls and it prohibits operator overloading as well.

Since SecureHEAT programmers are readily familiar with C and C++, they should have no problem using Zig. The language can interact with C in a number of different ways. For example, Zig can seamlessly integrate key C libraries without significant overhead or entanglements. In addition, it can compile C code and export functions / build libraries that C code can utilize. It accomplishes of this without depending on the C standard library.

5 Flexibility and Generality

In addition to the four build modes mentioned earlier, Zig also offers flexibility in a number of other areas. The SecureHEAT system must be able to communicate with low-level interfaces such as cameras and network interfaces, and Zig appears to support such interactions. Packed structs, multiple pointer types, and arbitrary integer width are just a few of Zig's supported low-level features. One can also directly

Figure 1: Example of generically-typed data structure in Zig



```
const std = @import("std");

fn List(comptime T: type) type {
    return struct {
        items: []T,
        len: usize,
    };
}

pub fn main() void {
    var buffer: [10]i32 = undefined;
    var list = List(i32){
        .items = &buffer,
        .len = 0,
    };

    std.debug.print("{d}\n", .{list.items.len});
}
```

inline assembly code in Zig if necessary. However, because Zig is a "smaller" language, it contains less language features than a language such as C or Java. This may feel somewhat constricting to programmers on the SecureHEAT team, especially since Zig relies so much on its powerful comptime methods to achieve explicit type behavior.

Zig also emphasizes generality in its design, as it supports generic data structures and functions. Compile time parameters allow for duck-typing in Zig, which may prove to be superior to C / C++'s system of handling templated objects. Figure 1 contains a snippet of code from a Zig program that contains a data structure that returns a type that is determined at compile time (with the help of the `comptime` keyword).

6 Memory Management and Performance

Since code written in Zig can have wide applications, programmers must manage their own memory - including the failure of memory allocation. The documentation mentions that the Zig Standard Library is compatible with freestanding programs so long as memory is allocated with an allocation parameter. Since the programmer is responsible for various aspects of the memory management system, such as preventing memory leaks, accessing null pointers, or corrupting the heap, dynamic allocation is a bit riskier. It should go without saying that the team developing the SecureHEAT system should be able to swiftly detect and eliminate any problems that arise out of non-automatic memory management systems.

In terms of performance, Zig's 4 build modes offer a wide range of improving / optimizing program speed. The `Optimizations` parameter improves speed at the cost of debugging and compile time. In addition, Zig limits runtime overhead through `comptime` variables, where types can be manipulated as values.

7 Conclusion

In general, Zig's design seems to be suited for Haversack's requirements. It is lightweight, generic, and easy enough to learn - especially for those familiar with C and C++ syntax. Of course, there is a trade-off; Zig sacrifices some tools for its low-abstraction and explicitness. In addition, programmers will have to worry about managing memory themselves when writing code in Zig. Despite certain drawbacks, it seems like Zig could be a good fit for writing safe, secure applications in the SecureHEAT system.

8 References

https://ziglang.org/documentation/0.4.0/

https://jaxenter.com/replace-c-zig-language-138242.html

https://jaxenter.com/zig-language-kelley-interview-138517.html