CS 131 HW6: Analysis of D, Odin, and Zig Programming Languages for Use on Secure Camera-Based HVAC Control

Ryan Wakefield
University of California, Los Angeles

Abstract

The goal of this assignment is to weigh the advantages and disadvantages of using the programming languages D, Odin, and Zig for use on a secure camera-based HVAC control system. To make this determination, we researched the strengths and weaknesses of each of the languages, emphasizing the potential security issues with the language that could cause security breaches and data theft, compromising the integrity of the system.

1. Introduction

HaverSack Inc. is working on SecureHEAT, a system intended to save money by controlling temperature more efficiently in buildings. The goal of this program is to inexpensive cameras to implement a Human Embodied Autonomous Thermostat (HEAT) system that monitors the faces of humans, calculating their facial temperatures, and then updating the buildings' air temperatures accordingly. These cameras use wireless networks to connect to base stations at secure locations within the building. The base stations do the bulk of the calculations and use the result to control the building's HVAC units. Many of the potential customers for these HVAC systems have sensitive information stored at their facilities, and thus need guarantees that these cameras and system are secure and won't be penetrated by malicious hackers. Thus, the software being used in the cameras must have the following attributes:

1) Software must be cleanly written and easy to audit.
2) Software should be a freestanding program, no separate OS.
3) Software should be as stripped down as possible.
4) Software must be capable of interfacing with low-level hardware devices.

The potential languages that are being considered for use in this project are D, Odin, and Zig. Thus, the pros and cons must be weighed to determine the viability of each of these languages satisfying the aforementioned conditions. This paper will analyze the feasibility of using these three programs for this specific project.

2. D Language

Introduction

D language was created by Walter Bright and released in 2001. The main purpose of D was to be a language that is considered a "better C++" and thus have a very similar style and syntax to C++ but be easier to use and learn.

Memory

Memory locations in D come in three different groups:

1. Thread local memory
2. Immutable memory locations (write-once)
3. Share memory (accessible to multiple threads)

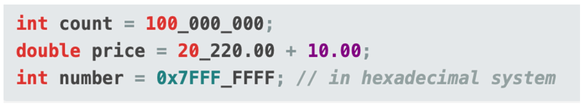
D is a system programming language which allows for manual memory management. By default, D uses a garbage collector to manage memory allocation. However, garbage collection can be turned off and manual memory management of reference counting can be used if the garbage collection software isn't trusted. Additionally, D allows for three different security levels for functions which will allow for safer programming. These three levels are:

1. @system
2. @trusted
3. @safe

The default function level is @system. The level that prevents memory bugs by design is @safe; @safe code can only call @safe or @trusted functions. Pointer arithmetic in safe functions is forbidden, which will prevent buffer overflow attacks, which is done by manipulating the stack pointer to point outside of the buffer memory and access data that shouldn’t be allowed to be accessed.

Syntax

D is a very readable and understandable language for anyone that has written in C/C++ as this language was developed to be a better version of C++. Underscores are allowed to be used in numbers to allow for ease of reading, as seen in the figure shown below.



```
int count = 100_000_000;
double price = 20_220.00 + 10.00;
int number = 0x7FFF_FFFF; // in hexadecimal system
```

Figure 1: Use of underscores in numbers to allow for better readability. Source: https://opensource.com/article/17/5/d-open-source-software-development.

There are many other instances of syntactic sugar in D that allow for improved readability of the language, such as the use of the “foreach” function to traverse a collection of objects. The syntax of this language overall makes it more readable and easier to use.

Disadvantages

D also has many features that push it towards being a “high level” language. For example, the *auto* keyword in D allows for type inferencing. While this allows for more flexibility in the code, it is not necessary for low-level programs like the one that we are trying to create, as it can create larger data type (32-bit int vs 64-bit int), where the extra size isn’t necessary and wasteful for the limited memory that the cameras have.

Compilers

There are three different compilers available for D, all open source. Open-source allows for increased trust that the source code isn’t malicious and prone to allowing security breaches. The three compilers for D are DMD, GDC, LDC.

Generality

This is a very good language to use for other programs and projects, not just low-level. It has a garbage collector as well as type inferencing, which makes it good for high-level programming as well.

3. Odin Programming Language

Introduction

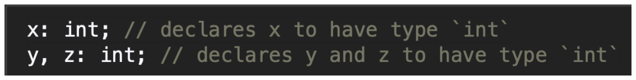
Odin is an open source programming language that was made with the intent of replacing and being a better version of C. Its goal is to achieve better simplicity while maintaining high performance.

Memory

Odin supports the use of dynamic arrays and other dynamically allocated objects. Odin is not automatically garbage-collected. The user must explicitly release the memory in the program after the program is done using it.

Syntax

Although Odin is supposed to be a similar language syntactically to C, there are noticeable differences between the two languages. For example, the type of a variable is declared after the variable name and a colon.



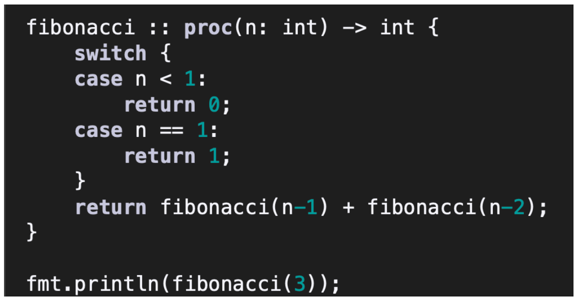
```
x: int; // declares x to have type `int`
y, z: int; // declares y and z to have type `int`
```

Figure 2: Declaring variables in Odin. Source: https://odin-lang.org/help/

Additionally, functions in Odin are called procedure, and they are declared using the format:

[ProcName] :: proc([parameters]) -> [returnType]

An example is shown in the figure below.



```
fibonacci :: proc(n: int) -> int {
    switch {
    case n < 1:
        return 0;
    case n == 1:
        return 1;
    }
    return fibonacci(n-1) + fibonacci(n-2);
}

fmt.println(fibonacci(3));
```

Figure 3: Declaration of procedures in Odin. Source: https://odin-lang.org/docs/overview/

The language overall is fairly readable and easy to maintain.

Documentation

Documentation of this language seems to be fairly sparse, with not much information regarding memory management and examples of the code. This is most likely due to the language being fairly new, with not as many people using the language compared to older languages that have been around longer.

Reliability

There is still question to the reliability of this language because of how new the language is and the possibility for bugs in the program. Additionally, there is not a lot of documentation for this language which makes it difficult to determine reliability. More research will be needed to fully understand the reliability for this language.

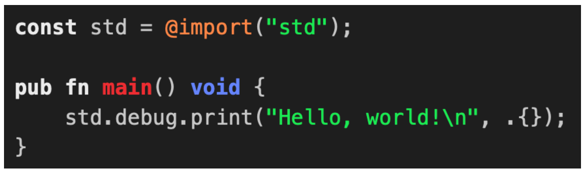
Disadvantages
Disadvantages of Odin, as explained above, are lack of documentation and potential reliability issues given how new the language is. There could be potential security flaws and bugs in the code that haven't been discovered yet due to lack of testing and less people having used the language.

4. Zig Programming Language

Introduction
Zig is a statically typed, compiled programming language designed by Andrew Kelley in 2016. Zig prides itself in being a maintainable, robust language that is easy to read and use.

Memory Management
Zig requires manual memory management, there is no garbage collection available for this language. Additionally, programmers must also handle memory allocation failure. Zig has the keywords *defer* and *errdefer* that guarantees the memory allocation and deallocation is successful and will let you know if there are any issues.

Syntax
Zig programming language has no hidden control flow, no hidden memory allocation, and no macros. The official website states "If Zig code doesn't look like it's jumping away to call a function, then it isn't" (Ziglang.org). This is different from C++ in the sense that C++ can have operator overloading in which an operator might call a separate function. This helps in making it easier to see if code jumps to another function that potentially could cause a security breach.



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

pub fn main() void {
    std.debug.print("Hello, world!\n", .{});
}
```

Figure 4: Hello World in Zig. Source: https://ziglang.org

As can be seen in the "Hello World" program for Zig above, Zig has a very similar syntax to C.

Generality
This language has good generality for many different purposes, as it is very similar to C with better security and other minor features. This would be a good language to use for purposes other than this project, as it has many good features to use.

5. Security and Ethical Concerns

A main concern for the development of this program is the potential security risks involved with using security cameras to determine people's face temperatures. If the software or the language it is written in has security flaws, they can be exploited to obtain sensitive data if some of the locations it is being installed are private or hold confidential information. To prevent security breaches, we need to make sure that all the code being written doesn't have vulnerabilities that can be exposed. Someone that gains control of these cameras could potentially use it to either track people in the building or to use it to see classified information that is stored in the secure building. Thus, there needs to be assurances that there are no vulnerabilities in the code and no potential for these cameras to be hacked.

It is good to know the source code for languages when determining if they are fully secure or not, and a good way to do this is to use an open-source software so everyone can see the actual code that is being used. Open-source software allows for transparency between the company and the user, ensuring that the engineers that created the program did not add anything into their code that could potentially create security problems and cause data leaks.

A large ethical concern that pertains to this project is the invasion of privacy to individuals in the building. The cameras being used are storing information on individuals inside the building, tracking their moves and using facial features and temperature to determine whether the HVAC system should be turned on to regulate the temperature inside the building. This could be considered a violation of privacy, especially if the people in the building aren't aware that they are being tracked.

While the SecureHEAT technology could create a potential security concern, it should be noted that often times apps on your smartphone are already tracking and storing your personal information, so this would not be much different than what is already happening on your phone. However, there are ways to mitigate these ethical issues. For example, you can inform all workers and customers of the technology that is being used in the cameras, and make sure they understand what data it is collecting and using. Informing people and making them aware of what is happening gives them the power to decide whether they want to enter the building where this technology is being used. Making them aware of the technology and allowing them to give informed consent, as discussed in Allhoff's and Henschke's paper *The Internet of Things: foundational ethical issues,* gives power back to the individuals, rather than leaving them unaware of the situation and leaving them in the dark.

Further action can be taken to reduce the risk of security breaches and potential misuse of information and violations of people's privacy. For example, ensuring that the code is secure and free of bugs and security issues, as mentioned above, along with giving a limited number of people access to the system will reduce the risk of misuse of the system and help maintain users privacy while keeping the company's data secure.

6. Performance

All of these languages have decent performance for the purpose of this project. These languages are decently efficient and good for using on the cameras for SecureHEAT.

7. Language Recommendation

After looking at the specifications and examining each of the languages, it is my recommendation that Zig be used for this project. D is too high level for this project, with having a garbage collector and type inferencing. While these are useful to have, for low-level programming these are unnecessary and can cause security and memory issues that will create difficulties implementing this project. While Odin is a good low-level language, it is very new and is not very well documented. Additionally, because of how recent the language is, there could be potential bugs that could lead to security breaches, and because of this, it is my recommendation that Odin should not be used for the security system. Zig is a good low-level language that is well-documented, no garbage collector and manual memory allocation, all of which are ideal for this project. Additionally, Zig has safe memory and other security measures in effect to make this a good language for SecureHEAT. For these reasons, I recommend Zig be used for the project.

8. References

[1] 5 reasons the D programming language is a great choice for development https://opensource.com/article/17/5/d-open-source-software-development

[2] D https://dlang.org/foundation/about.html

[3] Introduction to the Zig Programming Language https://andrewkelley.me/post/intro-to-zig.html

[4] Zig https://ziglearn.org

[5] Zig Introduction https://ziglang.org

[6] Odin https://odin-lang.org