CS 131 Homework 6
Odin: An Executive Summary

Puvali Chatterjee
UID: 504822474

Introduction

This executive summary presents an evaluation of the programming language 'Odin' for its use as a potential platform in Human Embodied Autonomous Thermostat (HEAT) system cameras. The cameras are attached to building power, and have no battery backup; they use a wireless network to connect to base stations at secure locations within the building. As the cameras' embedded CPUs have limited processing power and memory, the base stations do the bulk of the calculations and use the results to control the building's HVAC units. The software written in Odin needs to fulfill the following criteria: the software running inside the cameras must be cleanly written and easy to audit, it should be a freestanding program with no separate operating system, it should be as simple and stripped-down as possible with no components like interpreters or garbage collectors, and finally, the software must be capable of interfacing to low-level hardware devices, such as camera or network interfaces [1].

Effects on Security

A secure language must make it easier for a programmer to write reliable code that is invulnerable to attacks. Firstly, for safe memory management, storage allocation should be private (no other part of the program points into the newly allocated storage) and type-correct, and deallocation must be safe [2]. Since Odin uses manual memory management, all storage allocation and deallocation must be done by the programmer. Aside from the builtin dynamic array and map types, a programmer may manually allocate or deallocate storage using procedures such as *alloc, new, free, delete*, etc. along with a suitable allocator. As such, storage allocation with alloc is not type-safe or private, and deallocation with *free* or *free_all* is unsafe since it may lead to issues like dangling pointers. One way in which Odin *is* secure is that array access is bounds-checked by default, both at compile-time and runtime. Odin does, however, allow casting variables and this is also unsafe. Overall, Odin is not significantly superior to C/C++ in terms of security.

Ease of Use

One of the strengths of Odin is that it was designed with the goals of simplicity and readability in mind. One of the ways this was achieved was by attempting to offer only one way of doing something. For example, Odin does not allow infix and suffix syntax for procedural calls. Additionally, there was also an attempt to make Odin easy to learn. Odin does not offer implicit numeric type conversion despite its convenience. Odin also does not have pointer arithmetic or pre/post-increment or decrement operators. Furthermore, Odin also offers built-in types like dynamic arrays and hash maps, explicit procedure overloading, clean syntax, ranged for loops and many other functionalities that would reduce the amount of code that an Odin programmer would need to write.

Flexibility

Odin offers good support for interfacing with other languages and libraries. The *cstring* type, which is equivalent to char const * in C, is used to interface with foreign libraries written in/for C that use zero-terminated strings. The *foreign* system offers support for interfacing with external code or libraries; one can *import* (keyword) an external library into the code. In addition, Odin also supports several code calling conventions to help in interfacing with foreign code. These conventions include: *odin* (default convention used for an Odin procedure), *contextless* (same as Odin but without implicit *context* pointer), *stdcall* or *std* (same as the stdcall convention specified by Microsoft), *cdecl* or *c* (default calling convention for a procedure in C), *fastcall* or *fast* (compiler-dependent), and *none* (compiler-dependent).

Generality

Generality refers to the avoidance of special cases and the combination of related constructs and functionality into one construct. One way in which Odin is a general language is that it offers only one loop statement: the *for* loop. This is in contrast with a language like Pascal which has three loop structures: while, repeat and for [3]. However, Odin also has range-based *for* loops,

which may be considered a special case of the basic *for* loop, and this extra for statement lends itself to loss of generality. In some other ways, Odin's design goal of explicitness may have conflicted with generality. For example, *switch* statements in Odin do not require semicolons after each case (as in C/C++) because only the selected case is executed. To explicitly fall into the next case block instead of breaking out of the *switch* statement, one must use the *fallthrough* statement at the end of the block. This is an additional special case that is a part of the *switch* statement construct.

Performance

Another major design goal of Odin was high performance. It is not instantly evident how Odin offers significantly better performance than C since it is similar to C in many ways, but C is already one of the best performance languages. There are still a few ways in which Odin aims to improve its performance. In C, everything is passed by value by default but in Odin, data structures like slices, dynamic arrays and maps are passed as immutable pointers internally for performance. The *bit_set* type, which is modeled to be the equivalent of the mathematical notion of a set, is implemented as bit vectors internally for high performance. Although array access is bounds checked by default, the *#no_bounds_check* directive may be used to improve performance when the array index is known to be within range.

Reliability

The reliability of a language is strongly related to its security. The following quote by Niklaus Wirth is listed in the Odin documentation: "Reliable and transparent programs are usually not in the interest of the designer." Therefore, one can surmise that reliability was not a goal when creating Odin. Evidently, Odin trades in reliability for ease of use and performance, and leaves it up to the programmer to determine how errors are handled. According to the documentation, Odin does not have exceptions because they would complicate the understanding of the program. So Odin uses plain error handling via multiple return values. Programmers are encouraged to use the Odin type system to handle errors there and then without passing them up the stack by explicitly testing return values of calls; the alternative to this would be exceptions, which require unwinding of the stack and are much slower than the manual examination of a return value [4].

Conclusion

Odin supports the four properties stated in the introduction: Odin software can be cleanly written, it would form a freestanding program, the software could be written very simply, and the software would be extremely capable of interfacing with low-level hardware devices. However, the main aim, which was to find a programming language that can be used to write a program that would make HEAT cameras invulnerable to attacks, is not met at all. Although Odin has its strengths, it is not particularly safer than C or C++ and would therefore present the same issues of vulnerability as software written in those languages.

References

[1] Eggert, P. (2021). *Homework 6. Evaluate a language for secure camera-based HVAC control.*
Retrieved June 4, 2021. https://web.cs.ucla.edu/classes/spring21/cs131/hw/hw6.html

[2] Lee, K. (2004). *CSE 341: Unsafe languages (C)* [Lecture Notes].
Retrieved June 4, 2021. https://courses.cs.washington.edu/courses/cse341/04wi/lectures/26-unsafe-languages.html

[3] Rhodes, L. K. (2015). *Design Criteria for Programming Languages.*
Retrieved June 4, 2021. http://jcsites.juniata.edu/faculty/rhodes/lt/plcriteria.htm

[4] Ginger Bill. (2018). *Exceptions — And Why Odin Will Never Have Them.*
Retrieved June 4, 2021. https://www.gingerbill.org/article/2018/09/05/exceptions-and-why-odin-will-never-have-them/#fn:1

[5] Ginger Bill. *Odin Overview.*
Retrieved June 4, 2021. https://odin-lang.org/docs/overview/ (not cited in-text)

[6] Ginger Bill. *Frequently Asked Questions.*
Retrieved June 4, 2021. http://odin-lang.org/docs/faq/ (not cited in-text)