SecureHEAT: Camera-based HVAC Control

Evaluation of programming language Odin for implementation for SecureHEAT

Amina Lazrak
University of California, Los Angeles

1. Application Background

Haversack Inc. creates products specializing in large-scale heating, ventilation and air conditioning (HVAC) systems. These systems are often very expensive to set in place and require extensive funds and labor to maintain. As a result, Haversack has put together a team to develop an alternative method to thermostats to efficiently control the temperature of buildings which is known as SecureHEAT. This system consists of multiple inexpensive cameras that implement the Human Embodied Autonomous Thermostat (HEAT) system which uses facial temperatures to regulate the buildings' air temperatures. The system works by using these inexpensive cameras as sensors to scan the faces of the buildings' occupants, then computing their facial skin temperatures and subsequently readjusting the temperature of the space, aiming for high occupant comfort by taking into consideration features such as thermal comfort zone and comfort probability [1]. This set up leads to more efficient HVAC systems through the automatic optimization of thermal environments and the potential removal of manual thermostat use that often need repeated maintenance and require high energy.

Although the HEAT system provides much greater efficiency and incurs lower costs than manual HVAC systems, a few considerations must be taken into account when setting them up. HVAC systems are often used in large-scale environments such as in the buildings for banks, prisons and intelligence systems meaning they must provide a maximum amount of security and behave in a robust, reliable manner. The use of inexpensive cameras to replace manual thermostats brings with it potential security threats, specifically in regards to the software running these cameras. In addition, these camera sensors consist of CPUs with limited processing power and memory and therefore must connect to base stations where most of the computations are done. The cameras also run completely on the building power without a backup battery and connect wirelessly to the building's base stations at secure locations where the bulk of the work is performed. All of these constraints must be accounted for when designing such a system with these inexpensive cameras.

Haversack Inc. has previously utilized a combination of the programming languages C and C++ to implement most of their programs, yet both these languages are quite vulnerable to security attacks. Therefore, the SecureHEAT team has investigated the potential use of another language, Odin, as a candidate for writing and running the cameras' software. This executive summary thoroughly examines the language's strengths and weaknesses to determine whether it is an appropriate fit for fulfilling the desired requirements of the SecureHEAT product.

2. Odin Background

On the primary page for the language documentation, Odin is described as a language created as an alternative to C with the goals of providing simplicity, high performance, and compatibility with modern systems [2]. It is inspired by Go as well as C and its syntax resembles closely to the one for Pascal [3]. Odin is a relatively new programming language with its development beginning in 2016 and the documentation still marked "in progress". The recency of this language is important to take into account because it could affect its reliability if updates are continuously occurring and the language has not yet reached stability.

3. Odin Language Features

Two distinctive features Odin presents is parametric polymorphism and implicit context. Parametric polymorphism allows for the use of generic procedures or functions as they have the ability to accept and handle multiple types of parameters. Odin offers explicit and implicit parametric polymorphism where types are either explicitly provided to the procedures or they are implicitly inferred. Because Odin has a strong typing system, it would be recommended to use explicit parametric polymorphism to ensure safety in the case of SecureHEAT. The type inference associated with strong typing could introduce uncertainty and therefore hinder reliability. Another option to increase safety of the product when using polymorphic parameters is the use of Odin's 'where' clause. This feature allows for bounds to be applied to arbitrary types and polymorphic type parameters and can be used to perform sanity checks for parameters, enforcing consistency to reduce ambiguity that comes with polymorphism [4].

The second feature that distinguishes Odin is that each scope contains an implicit local value known as 'context' [4]. This

context variable is implicitly passed through a pointer to any procedure call in its scope with the intent to intercept any third-party code and libraries and modify their functionality if need be. This is a powerful feature for implementing security and reliability and stands out as one of the advantages of using a language such as Odin, yet the program for SecureHEAT must be free-standing and therefore will not make use of any interceptions, essentially rendering this feature inapplicable in this case.

Regarding the syntax of the language, Odin provides flexibility through the data structures of slices whose lengths do not necessarily need to be known at compile time as well as dynamic arrays that can change lengths during runtime [4]. Yet once again, given the constraints of the inexpensive cameras, SecureHEAT does not depend on a runtime which makes this added flexibility obsolete. Odin offers safety features for variables through the fact that allocated variables are by default initialized to zero. In addition, the symbol '---' can be utilized for null initialization and is beneficial for maintaining readability by making it explicit that certain variables are not initialized. Odin also exploits a similar feature to higher-level language Java by allowing variables to be declared as "private" and cannot be accessed by external packages. This adds to both the safety and the readability of the program as variables are limited to their respective scopes.

Procedure arguments in Odin are immutable meaning that variables must be copied explicitly before a copy is passed to a function. This is an important feature as it guarantees that additional pieces of code will not be implicitly called when variables are passed in as procedure parameters, adding another layer of transparency. In addition, this allows Odin to optimize the way in which procedure values are passed (either by value or as an internal pointer) to improve performance. Odin also allows the use of the 'defer' keyword that guarantees that a block of code will be executed before a function is returned. Unlike the C++ destructors, 'defer' is explicit and more flexible in Odin as it can be used to provide this assurance for anything. This layer of guarantee is important for safety checks before a function exits [3].

Odin's use of manual memory management means the programmer is responsible for designing their own methods of memory allocations as well as a system for tracking memory in use. As a result, Odin has a substantial support for custom allocators especially through the implicit context system. This does require more effort on the part of the programmer if memory allocation is needed but it once again increases compatibility with low-level programming making the code more portable as it does not rely on hardware memory space.
Similar to C, all variables in Odin are passed by-value where procedures receive a copy of the variables [4]. This could potentially lead to an issue with space complexity as the many copies of the same variable may be created and then stored in memory. In the case where the memory reaches capacity faster than the ability of the memory management to deallocate, a stack overflow could be encountered and would cause the SecureHEAT system to crash if error handling is not in place, markedly reducing the reliability of the program.

In terms of error handling, Odin does not use exceptions as is usually done with try and catch blocks in other languages [4]. This is because Odin does not allow non-local control flow to ensure a better understanding of the program and a more readable code as there is no jumping around the code. Odin uses 'plain error handling' in which values are returned from the procedures where the error occurs to ensure that it is clear where errors are happening. This is advantageous for safety as any bug in the program can be easily traced back and addressed right away.

4. Recommendation

The SecureHEAT product requires the design of a low-level program that is flexible and easy to use while ensuring reliability and safety. Given the features of Odin and its outlined strengths and weakness, it is reasonable to implement SecureHEAT in this language, yet a few precautions must be taken as it does not completely satisfy all of these requirements. Odin allows for features such as immutable function parameters, manual memory management and 'undefined' types which make it well suited to fulfill the flexible and low-level requirements of SecureHEAT, yet one area where it lacks support is in error handling. Preventing the program from crashes relies on the programmer's design which reduces its robustness and given the method of variable passing that requires copying variables and pointers, it is possible that not all errors are enumerated and caught manually one by one. Therefore, it is recommended that Haversack Inc. implement a backup procedure for handling program crashes to avoid building evacuation due to drastic temperature changes. Yet altogether, Odin meets the majority of the requirements of SecureHEAT and is therefore an appropriate choice if used with necessary caution.

References

[1] Li, Da, et al. "HEAT - Human Embodied Autonomous Thermostat." *ScienceDirect*, Elsevier, 2020, ebyon.engin.umich.edu/wp-content/uploads/sites/162/2020/05/2020.HEAT-Human-Embodied-Autonomous-Thermostat.pdf.
[2] "The Odin Programming Language" *Odin*, odin-lang.org
[3] "Frequently Asked Questions." *Odin*, odin-lang.org/docs/faq/.
[4] "Odin Overview." *Odin*, odin-lang.org/docs/overview/.