Secure Camera-based HVAC Control: Language Evaluation

805291212
Matthew Chan

Abstract

I am tasked with the assignment of evaluating three different languages in order to consider how compatible they are with implementing a secure bare metal camera based heating system. There are different constraints and specifications given to me in order to determine which language to seriously consider; the languages are D, Odin, and Zig.

Introduction

This HVAC system will take place where the privacy of the users is paramount. Thus, we have to keep in mind security features suitable for places like banks, military bases, federal intelligence locations, prisons and other various high security sites. From this, we determine that absolutely no OS should be implemented on our devices, as they pose a security risk. In addition, the language used needs to be as bare bones and reduced to basic functionality as possible. This eliminates features like interpreters as well as garbage collectors.

Furthermore, this code will need to be inspected and audited. In other words, language syntax needs to be simply written and unambiguous, to better find errors in logic or language specific bugs. Thus, it becomes clear that transparency of the written source code is incredibly important.

To conclude the specifications and constraints, the language must of course interface well towards low level hardware devices. This is the most obvious, yet one of the most important. With all these rules and regulations in mind, we turn to the three languages we were tasked to investigate.

D Lang

To be blunt, Dlang is not even worth consideration. No matter what merits it has, the language violates an important rule we have set in that it comes with an automatic garbage collector. This instantly removes itself as a viable candidate for our secure HVAC system.

Odin Advantages

Odin, on the other hand, has some merits to it that make it seem like an acceptable candidate for what we have in mind. To start, Odin comes with manual garbage collection, one of the necessities we look for yet was conspicuously absent from dlang. Furthermore, the tenets of Odin are grounded in simplicity and unambiguous code; it is documented that there is only one way to write an expression (though there are different ways to type cast). Ginger Bill, the creator of Odin, created this language with the goal of achieving simplicity and functionality stripped down to its core components. As an aside, he has so much faith that he provides only the most basic and core features of a language that should you be presented with a missing feature, he asks you to “please investigate what features Odin does have and...find intriguing ways to solve your problem in the lack of [the] feature”[2].

The author also states compatibility with systems programming. As it is similar to C, which is a top pick for bare metal programming, the fact that this also does well is of no surprise. Another feature of note is its explicit procedure overloading for a sort of polymorphism, though it doesn’t support implicit procedure nor operator overloading. And to conclude, Odin features plain error handling, which makes debugging easier as you know exactly which procedure caused the program to fail.

Odin Disadvantages

Although Odin comes with several favorable features, there are still disadvantages when it comes to using this language to implement our service. First and foremost is that Odin is an incredibly new language. The language debut was just last year in March, however the documentation is nowhere near complete; there are still lots of missing entries in the website’s documentation. In addition, since it is a new language, some features and libraries may be unavailable at the given moment. To be frank, it even lacks its own Wikipedia page. Although the author did state before to think of “intriguing ways” solving our problems, there exist problems creating those intriguing ways. For example, there are no methods and classes, as a core Odin belief lies in the separation of code and data. Furthermore, there is a lack of C++ style copy, move, and regular constructors. All these things may make it difficult to implement the exact service we desire.

In conclusion, Odin is an acceptable choice to investigate, however we must be prepared to do our procedure in a roundabout way.

Zig Advantages

From my investigations of Zig, this seems to also be an acceptable language for our HVAC service. Zig is classified as an imperative statically typed systems programming language. Thus, it interfaces well to lower level hardware and programming. It was designed to be a competitor to C. Further, the aim of Zig is to “focus on debugging your application rather than debugging your programming language knowledge”[3]. To that end, you would not be wrong to deduce that their syntax is clear, unambiguous, and intuitive; although this is debatable. These desirable qualities, in addition to the language requiring manual data management, beg further attention and investigation for our purposes. The clear and concise manner of the language is exhibited by its notion of no hidden control flow. This, in effect, means that what you see is what you get. There is no operator overloading, functions don’t have throw and catch exceptions, basically everything the program does is plain to see and there are no hidden mechanisms like preprocessors or hidden memory

allocation. In addition, the language has four different build modes, each one tailored to different needs like trading speed for safety and vice versa.

Zig seems to be commonly used for bare metal programming and a few projects have already made their way on GitHub, quite more than Odin. A few things that make Zig desirable are it's inclusion of assembly code manipulation and error return trace. For the former, although the Zig compiler generates machine code, it is possible for the developer to have direct access and to manipulate the generated machine code. As for the error return trace, it enumerates the control flow the system took when it crashed, making it easier to see which conditions failed and which steps were taken following that. Regular stack traces do not provide the same level of detail and therefore make debugging harder than when used with error return traces.

Zig Disadvantages

To be blunt, Zig is not complete language. The amount of users who actually use this language is actually pretty small, though still larger than Odin. However, version 1.0.0 has not even made a release yet. This means that features are liable to change. Undocumented features we previously took advantage of might suddenly not work and overhauls are liable security risks. In addition, though the language is meant to be clear and concise, the actual syntax of the language is none too pleasing. Those unfamiliar with the language will have difficulty learning the control flow of the language.

To conclude Zig's assessment, it appears to be a viable choice for our secure HVAC system; both Zig and Odin should be looked further into.

With all that said and done, I must bring about an important topic to discuss: the ethics of our secure camera based HVAC system.

Ethical Discussion

The internet of things served as a great opener for the discussion of the ethics of future devices. In this great information age, we see companies take advantage of their users all the time without their consent, just like in the Cambridge Analytica case. We need to discuss ground rules and introduce a system of trust between employer and worker through everyday, inconspicuous, yet vital control systems. Our systems have three core components: the sensors, informational processors, and actuators. The internet of things introduced relevant and current issues to these three core components as we face the advent of technology today: informed consent, privacy, informational security, decision making and trust [5]. It is important to note, however, that trust is a ubiquitous factor and can only be achieved through the fulfillment of all other issues. Trust, then, is the end all be all we should strive to achieve.

Sensors

For our service, these are the cameras collecting facial temperature readings of those in the building. The main issue surrounding sensors are informed consent and privacy. You may be able to tell those residing in the secure location that a new, advanced heating system will be implemented, however they will not know that their facial temperatures will be used throughout the day for climate control. They have no idea that their personal information is being collected like this, a violation of privacy, although they have been 'technically' warned that a new system will put into place, informed consent. This information can be harvested, analyzed, and then used against them all without their informed consent.

Those working in the secure location have no idea what this new implementation entails, thus they are unable to give informed consent and have their privacy violated. They need to be educated on what implementing the new system means and should be fully aware of how the system achieves its goal and the risk it causes them. Only then can there be informed consent and a lack of violation of privacy.

Informational Processors

These informational processors would be the wireless data receivers connecting the cameras to a central unit in charge of analyzing the data and controlling the temperature. The issue here is with informational security. Given that a high security site will use our technology, it is not unreasonable to assume any data accessible to the outside world will be hacked. Thus, we must look to how secure our system is. For example, the app Strava had a global heatmap that allowed enemies of the state to track jogging patterns and supply routes of military personnel [5]. Even seemingly harmless apps and data can be abused and turned into weapons, given the opportunity. We must be proactive in our endeavors to disallow our data and technology to be used against us in such a manner.

Actuators

And finally, the last component that needs to be addressed are the actuators. These are the base stations in charge of analyzing the camera's harvested data and manipulating the heating based on the people's facial readings. The main issue connected with the actuators is their decision making capability. Given the condition of the people inside, what will the system do? I would say this is the start of a trusting machine worker relationship. There needs to be trust that the system will have your best interests in mind and adjust the temperature to what you and everyone else needs. It becomes all too easy for the employers to manipulate the machines to save as much money as possible or make it hot enough to crave cold beverages.

Conclusion

There are a few ethical concerns we must first address before fully realizing our secure HVAC system. The actuators of the system should be programmed with the users' best interests in mind. The communications should be secure enough to not leak and threaten the privacy of those whom the data belongs to. And finally the sensors need to be completely explained in their method, to let the users know what exactly the system entails and what it requires of them. Further, we investigated three languages for our service. The first, Dlang, was completely unsuitable as it had an automatic garbage collector. Odin and Zig are far more acceptable choices and both are still relatively new, though Zig has had its debut three years earlier than Odin. They both feature manual garbage collection and have concise syntax. And finally, they both can interface to low level hardware, such as cameras, which make them desirable for our service.

References

[1] https://dlang.org/spec/garbage.html
[2] https://odin-lang.org/
[3] https://ziglang.org/
[4] https://en.wikipedia.org/wiki/Zig_(programming_language)
[5] Fritz Allhoff, Adam Henschke,The Internet of Things: Foundational ethical issues, Internet of Things, Volumes 1–2, 2018, Pages 55-66, ISSN 2542-6605, https://doi.org/10.1016/j.iot.2018.08.005. (http://www.sciencedirect.com/science/article/pii/S2542660518300532)