COM SCI 131 - SecureHEAT Language Options

Gabriella Long
University of California, Los Angeles

Abstract

The purpose of this report is to find a suitable language to fit the needs of Haversack Inc. Currently, Haversack is working on a new system called SecureHEAT, a more efficient way to control temperature by using cameras to implement the Human Embodied Autonomous Thermostat (HEAT) System which determines temperature control based on human occupants facial temperatures. However, the cameras must record temperature data and send the data over a wireless network to a secure base station; this causes security concerns, especially for clientele in high security buildings. To overcome this Haversack currently uses C/C++, but wishes to switch to a newer programming language to overcome the previous vulnerabilities.

1. Introduction

There are three (3) new languages that are considered as candidates for the Haversack SecureHEAT system: Zig, Odin, and D. Compared to C/C++ these languages are relatively new with D being released in 2001, Odin in 2019, and Zig in 2016. This summary seeks to explore each language, their advantages, disadvantages, usability, and implementation in regards to the SecureHEAT project. In order for a language to be a good candidate for the project it must fit with the existing architecture, be secure, written cleanly, easy to audit, freestanding with no separate operating system, simple, and able to communicate with low-level hardware [2, 5, 6].

2. Zig

2.1 Background

On the developer's website for Zig, it is marketed as a "general-purpose programming language … for maintaining robust, optimal, and reusable software." From that statement Zig already seems promising, however, to get a full understanding of the language we have to delve deeper. From the documentation and capabilities of Zig, it shows that its developers are attempting to dethrone C. Zig boasts its clarity and simplicity which are important for readability and auditing purposes. To expand on the clarity aspect, Zig favors explicitness over abstractions, yet does not suffer from losses in the performance category. Zig does not have macros nor does it have metaprogramming, but it has enough power to tackle complex programs in a clear fashion. Zig's clean code promises extend to error handling and it seeks to make error handling a language-expressible concern. One of the benefits of the "mainstream" languages is the portability, and C is famous for it. However, Zig is attempting to achieve library portability; it can be relatively straightforward to use the C ABI for required external functions. In addition, Zig's language design and safety precautions can reduce the number of common debugging issues [1, 8].

2.2 Compared to C

Zig showcases clarity and simplicity; it does not add complications or abstractions like C/C++. Another advantage Zig has over C is its memory safety. Similarly to C, when a variable's scope ends the variable should end as well to avoid dangling references. Zig has runtime safety checks to catch mistakes. It is memory-safe as long as the runtime checks remain enabled. On the other hand, C requires explicit memory management which can make it memory unsafe in inexperienced hands. Another feature of Zig is the speed compared to C, this makes Zig a fitting language for systems requiring real-time devices. One of Zig's biggest disadvantages is the lesser control over code as opposed to C; just how much of a disadvantage this is comes down to the project at hand, and the programmer's preference. Unfortunately there is a slightly more glaring problem with Zig, the comptime function's leakage. A comptime function may leak its implementation details to the interface; in the scope of SecureHEAT this is a problem because maintaining backwards compatibility becomes more difficult than it would with C. To avoid this, the programmer has to rid other comptime functions of related asserts which is a concern in regards to maintenance. Lastly, Zig uses automatic heap allocation which could cause the system to run out of memory especially when having an influx of data coming in [1, 8, 9].

2.3 Candidacy

Referring back to SecureHEAT's constraints, we want a clean, freestanding, simple, and interfacing language. Zig is a clean and simple language due to its goal of high code reusability and lack of abstractions, yet performance is not sacrificed; this makes it easily auditable. The SecureHEAT software also needs to be freestanding as a separate OS can be a security leak; Zig is not yet a freestanding language, however an experimental OS for Zig, Zen, is in development. Additionally, Zig could be considered a "stripped down" language because of lack of garbage collection. Lastly, Zig works well with real-time software and lower level hardware devices, which is required as SecureHEAT utilizes cameras to capture temperatures. Unfortunately not much is known regarding Zig's security, but what is known is that Zig is a reliable, flexible, and simple language that does not sacrifice performance. [9]

3. Odin

3.1 Background

On the front page of Odin's website it describes the statically-typed, procedural language as an alternative to C with the goals of achieving simplicity, high performance, and adaptability for modern systems. The general design principles for Odin are as follows: simple, minimal, orthogonality, data transformation, and algorithmic expressions over type systems. Delving a bit deeper into those statements, the main programming design feature for Odin is the simple and minimal approaches, the developer of Odin mentions that there should only be "one way to write something." Likely referring to more fully-featured languages that have no shortage of approaches to any given problem. As a language Odin wants to limit undefined behaviour; this way any optimizations can make clear assumptions about the given code. It seeks to force programmers to not rely on bloated APIs, but rather to have a language that does not limit you nor your abilities, nor does it help you. Some of Odin's highlights are: the packaging system, "when" statements for conditional compilation, less error propagation, custom allocations with its context system (allowing for more programming flexibility). Additionally, it allows for easier tracking (and control) of third-party libraries [2, 13, 14].

3.2 Compared to C

Since Odin and C are quite similar they share many of the same traits. However, Odin seeks to cover the shortcomings of C, and improve upon procedural, statically typed languages. Odin, as a language, provides a high control over memory layout, has a strong library system, has the ability to track allocations by third party libraries, and provides support for data structures such as arrays and slices (and much more). One clear advantage over C is parametric polymorphism or a dynamically sized array type, Odin does. There shockingly are not many glaring disadvantages to Odin apart from the obvious lack of community. I was also unable to find any information regarding the operating system it runs on, but it is quite new so that is to be expected [2].

3.3 Candidacy

In regards to the SecureHEAT project, Odin is a very promising language. It is clean, simple, and stripped-down language which already satisfies 2/4 of the requirements. To expand a little more why Odin is a good candidate is its lack of garbage collection, freedom, readability, and high performance. Additionally, Odin has a strong foreign system that interacts with C and vice versa which allows it to interface with low-level hardware devices. Like C, Odin's manual memory management allows for flexibility and control. One of the downsides to Odin is its current lack of operating system which is one of the free-standing aspects desired by the project. However, Odin is a very new language and an operating system written in Odin may not be too far away.

4. D

4.1 Background

On the landing page of D's website, it is described as a "general-purpose programming language with static typing, systems-level access, and C-like syntax." Developed in 2001 D was created with the goal of improving upon C and C++. D seeks to preserve the efficiency, low-level access, and Algol-style syntax of C/C++. D's creator wanted to gather the best aspects of the two languages and merge them together to create a new one. It allowed better memory allocation so previous memory-based constraints became a thing of the past. However, currently D is colloquially referred to as C++ with garbage collection. This is a good thing as it allows D to have safe memory, something that C/C++ lack. D has the goal of simplicity without sacrificing performance, and that design philosophy has helped D gain traction among professional and hobbyist programmers; and while D is not nearly as popular as C/C++ it has its uses in the industry, academia, systems programming, and research. It is worth noting that D is used by large companies such as Facebook, Ebay, and Netflix, just to name a few [3, 4, 5, 10].

4.2 Compared to C/C++

In comparison to C++, D has more powerful templates; a D programmer is able to do more with less effort, error messages also help to make D's templates easier to use and debug. Another of D's strengths when compared to C++ is handling of arrays. They can easily be appended to, resized, concatenated, etc, which for the amount of data that would be handled for SecureHEAT is a great attribute. D's arrays have slicing enabled as well which increase the efficiency for processing the arrays. D also allows 100% compatibility with existing C code. D allows for both low-level control, as well as higher level abstractions [3, 5, 7].

4.3 Candidacy

D is the near perfect candidate for the SecureHEAT project if it was not held back by its garbage collection. D is a clean, concise, and readable language which makes it a good fit for projects that need to be audited and regularly maintained. D is a free standing language; there are already 3 examples of operating systems, kernels, and compilers written in D, thus reducing the possibility of the operating system becoming a security concern. While D's front-end is open source, its backend was donated by Symantec in 2017. As briefly mentioned previously, D prides itself on its simple yet powerful design; it is a high performance language which suits real-time data collection such as the type required when recording and adjusting temperatures. D is written fast, reads fast, and runs fast. Finally, D is used in embedded applications so it would work well with the lower-level cameras required to make SecureHEAT [4, 5].

5. Ethical Concerns

Since SecureHEAT uses human faces to determine the "ideal" temperature within a contained space, security should be of the utmost importance especially when it comes to high security sites where sensitive information is housed. If the monitoring cameras become compromised then the sensitive information can be leaked. Of the three proposed options for an improved version of SecureHEAT all are open source languages; this does not make them an inherently poor choice, but there must be a consideration into the ethical implications of it. For SecureHEAT, as it is a larger scale project that will be implemented via open source code, and will be used for monitoring purposes there should be an open governance project for the language of choice. There are a few different models such as: "do-ocracy," founder-leader, self-appointing council or board, electoral, corporate-backed, and foundation-backed. For SecureHEAT, having an open governance project adds a layer of safety because it limits the contributions and also limits the possibility of a contributor adding malware or simply breaking parts of code which open it up to many vulnerabilities. Assuming Haversack Inc. is a large enough company to sponsor the developer(s) of the chosen programming language then they can choose the corporate-backed open governance model. Under this model, a company can limit, or completely stop, contributions from the language's community (anyone outside the company itself); contributor agreements are an option for trusted contributors but are not required to exist under this governance model. However, this takes away the community aspect of open sourced languages, but it is a good option to keep SecureHEAT safe for its clients' sensitive data. Additionally, as SecureHEAT fully utilizes humans and the environment to function, it is crucial for Haversack Inc. to understand the rules behind data collection required for SecureHEAT to function properly. Depending on where SecureHEAT is used the laws of data collection differ. Since SecureHEAT uses cameras to gauge and adjust temperature, they are filming occupants of the area in which it operates in; in regards to California laws, the privacy laws are quite robust. Because of this both Haversack Inc. and the clients must have legal documentation to inform users and clients (and others involved) of the data collection process taking place. However, that can be dealt with at a later date, but the open source open governance must be in place prior to reworking the SecureHEAT source code within the cameras [11].

6. Zig vs Odin vs D

While each language was previously explored more in-depth they will now be compared to each other in regards to their applicability to the SecureHEAT project with the guidelines set forth by the project manager. The software's criteria is as follows: must be cleanly written and easy to audit, a freestanding program (no separate operating system), simple and stripped-down, and capable of interfacing to low-level hardware devices.

Each of the three languages make claims of being clean and simple, and from reading code snippets from the respective languages' documentation pages, the developers are mostly truthful when referring to their code as being easy to read. However, D's syntax seems to be the closest to C which can be quickly and easily understood by a software auditor without too much prior exposure to the language.

Both Zig and D meet the freestanding program criteria and have operating systems written in their languages. More specifically, D has more than one, and Zig has a prototype. Odin, the newest language, does not have a working operating system written but it is extremely likely that one could be developed given it is marketed as a C alternative, and there are many examples of operating systems in C; Odin might have to mature more before an OS is developed in it.

All three languages have a stripped down design about them as they follow a C-like design and structure. However, neither Zig nor Odin have garbage collection. As mentioned in the specs, some clients may not trust the garbage collection aspect of a language. D has garbage collection thus it has a disadvantage in this category.

Finally, given SecureHEAT requires cameras and networks to properly function, the language chosen must be able to connect to said low-level hardware devices. All three of these languages are theoretically able to interface with hardware devices; Zig works well with real-time data connections as well, Odin mentions it works with hardware, and D has research and clients to back up its ability to interface [4, 6].

7. Recommendation

Despite D's garbage collection it is the best option (of the three) for SecureHEAT. It is a more mature language with a backend donated by Symantec owned by NortonLifeLock; its backend is strengthened by Norton's security research which, for software utilizing sensitive data collection in possibly high security areas, is an advantage over the other two languages. Additionally, the research that has gone into D provides a better understanding of the language itself and its limitations as opposed to the other two. Neither of which have been used in industry nor have much research done regarding their performance, and are typically used by programming hobbyists. D is a high performance, readable, free-standing language that should not be discounted by its garbage collector.

8. Conclusion

One of the most important aspects of programming is understanding the importance of trade-offs since there will never be a "perfect" programming language; each language has its strengths and weaknesses and it is the programmers' jobs to determine the best fit for the project at hand.

References

[1] *The Zig Programming Language*
https://ziglang.org/

[2] *Odin Programming Language*
https://odin-lang.org/

[3] *D Programming Language: Home*
https://dlang.org/

[4] *Areas of D Usage*
https://dlang.org/areas-of-d-usage.html

[5] *Origins of the D programming language*
https://dl.acm.org/doi/abs/10.1145/3386323

[6] *Homework 6*
http://web.cs.ucla.edu/classes/fall20/cs131/hw/hw6.html

[7] *D and C*
https://dlang.org/blog/the-d-and-c-series/

[8] *Zig Documentation*
https://ziglang.org/documentation/master/#comptime

[9] *Zig Documentation*
https://ziglang.org/documentation/master/#Alignment

[10] *Organizations using the D Language*
https://dlang.org/orgs-using-d.html

[11] *Understanding open source governance models*
https://www.redhat.com/en/blog/understanding-open-source-governance-models