Suitability of V for Secure Camera-Based HVAC Control Software
CS 131 Homework 6

Joseph Picchi
UID: 605-124-511

1 Ease of Use, Flexibility, and Generality

The V language is highly convenient for development in team settings like SecureHEAT. The syntax is very simple, and complex features that compromise readability are discouraged. V largely emphasizes readability at the expense of flexibility, all while preserving generality.

Stylistically, there is one way to perform an operation in most cases [1]. For example, variables must be declared with the syntax `var := value`, where the variable type is inferred from the value on the right hand side, and the variable is forced to be initialized to some value [2]. Another example is that naming conventions are enforced [3]. Variable and function names must use `snake_case`, and type names must use `PascalCase`. Also, `for` is the only looping keyword, and it has specialized forms for different purposes (e.g. `map for`, `condition for`). This is a pro because it improves readability by reducing stylistic variation between developers on the SecureHEAT team, while still providing constructs that are expressive enough to perform practically any operation that can be performed in other languages. The downside is that it sacrifices flexibility by confining implementation choices.

V also avoids complex behaviors that are difficult to trace when reading code. For example, function overloading is largely prohibited, with the exception of certain operators [3]. The keyword `mut` must precede mutable function arguments in both function definitions and calls so as to warn of function call side-effects. Elements of arrays must have the same type, and the user can explicitly specify that type, which limits confusion about array contents. Module prefixes must be specified every time you call external functions and variables, thus clarifying their external origins. The pro of this is that it drastically improves readability and ease-of-use because developers can more easily understand program behavior at first glance. The con is that it again sacrifices flexibility because programmers are forced to follow specific syntactical and structural paradigms.

Another pro for ease of use is V's interoperability with C [3]. C functions can be called from V, C code can be directly embedded in V code, and C code can be translated into V code. This allows developers to use versatile, well-tested C libraries with the safety and performance guarantees of V.

Finally, V can automatically format code to satisfy standardized vfmt style guidelines while preserving semantics [1]. This is a huge pro for readability and flexibility because developers can write code in their own styles, and team members can understand them by reading the vfmt converted files.

2 Reliability

V prioritizes reliability at the expense of implementation convenience and flexibility.

In most cases, error-prone behaviors are prohibited by default, but they can be enabled. Functions, constants, and types are private by default, requiring the keyword `pub` to use them in other modules [3]. Functions are pure by default, having no side-effects aside from I/O unless the `mut` keyword precedes arguments in function definitions and calls. There exists no global state by default, requiring the compiler flag `-enable-globals` to add globals. And almost all typecasting must be explicit (with minor exceptions). These features are major pros for reliability because they require programmers to acknowledge the addition of error-prone constructs each time they are used, thus reducing the probability of hard-to-catch bugs arising from function side-effects, global state modification, typecasting, and other phenomena. A con is that these constructs are often useful, so manual enabling may be inconvenient when performed frequently.

In other cases, V completely prohibits problematic behaviors. Variables must always be initialized [2]. Function arguments with primitive types are always immutable [3]. Variable shadowing is never allowed [1]. Strings are read-only arrays [3]. And struct fields are always initialized when created. These are pros for reliability because all of these prohibited behaviors are highly error-prone. They are cons for performance because initialization can be costly, particularly for memory-intensive objects whose fields are not all utilized (e.g. large arrays, large structs). Lack of variable shadowing is a con for ease-of-use because it requires variety in variable names.

Furthermore, V often mandates error handling. Matching `enum`, `sum`, and other types must be exhaustive or have an `else` branch, thus forcing programmers to explicitly handle all possibilities [3]. Option/result types indicate errors, and those errors cannot be left unhandled. There are 4 ways to handle optional types: (1) propagate errors to the caller (not allowed in `main()`), (2) break execution early, (3) provide a default value to return when errors occur, and (4) check for errors with `if` unwrapping. In all 4 cases, errors must be checked for explicitly. This is a reliability pro because errors cannot occur silently and produce illusive bugs, as they can in other languages (e.g. C). This is also a security pro because silent errors cannot cultivate undesired, security-compromising effects. It is a relatively insignificant con for ease-of-use.

3 Security

Most of the reliability pros mentioned in 2 also improve security because they discourage runtime errors that may cultivate unforeseen security vulnerabilities. For example, immutable function arguments may prevent sensitive data from being inadvertently altered or written to memory as a result of obscure function side-effects.

V requires that potentially memory-unsafe operations be labeled [3]. These include low-level code that might corrupt memory or present security vulnerabilities, such as pointer arithmetic, pointer indexing, pointer conversion, C functions, and `goto` statements. All such operations must be enclosed in an `unsafe` block. This practice has many pros. It warns readers that improper use of the unsafe code may make the program susceptible to adversarial attacks. It also protects developers from unintentionally using unsafe code, since the program fails to compile unless memory-unsafe code is placed inside `unsafe` blocks. Lastly, programs suspected of security vulnerability can be more efficiently debugged because all memory-unsafe operations are marked.

Furthermore, V performs bounds checks every time you access arrays. This protects against buffer overflows and other cases of writing or reading unauthorized memory addresses.

V has various other security pros. Static typing with explicit typecast statements enforces compile-time checks against variable misuse that mishandle unauthorized information, though they sacrifice the clean simplicity of a dynamic typing language like Python. V also prohibits undefined behavior, thus avoiding unforseen mishandling of sensitive information. Lastly, V can translate C code to V code, which adds the aforementioned safety precautions of the V language.

4 Performance

For memory management, V tries to avoid unecessary allocations in the first place with buffers and value types [2]. The documentation claims that 90-100% of objects are freed by V's autofree engine, which statically inserts free calls into the compiled binary during compilation. This is a pro because it makes program performance at runtime predictable. A small percentage of objects are freed using reference counting, which is a cheap form of garbage collection. The documentation does not state what qualifies an object to be freed by autofree or the GC, so the GC could be an significant cost for SecureHEAT's weak camera processors. Developers can disable autofree with a compilation flag and manually free objects when they are no longer needed. This is a pro because it enables fine-grained memory optimization [3].

For pure functions, the compiler automatically decides whether to pass by value or reference [3]. This could be a con if the compiler decides to pass by value, in which case it would have to copy the argument object.

Another con is that normal arrays in V grow dynamically, allocating a capacity of memory and reallocating more memory when the array contents exceed that capacity [3]. Reallocation incurs significant computational costs. A workaround is that V supports fixed-sized arrays with more efficient access, a capacity equivalent to array content requirements, and allocation on the stack. However, most builtin V array methods do not support them.

A final pro is that V has builtin profiling that can be enabled with a command line flag [3]. This conveniently places profiling results in a `txt` file to facilitate convenient identification of program bottlenecks.

5 Other Considerations for SecureHEAT

V would not cross compile for the SecureHEAT cameras without a custom compiler designed by the SecureHEAT team. V's compiler is written in V, so the compiler source code could be edited to generate a binary for SecureHEAT cameras. V's support for inline C and Assembly code would permit fine-grained, low-level control of registers and memory addresses on the cameras [4]. As is, the V compiler already creates reasonably sized native binaries without dependencies, which is what we require for our embedded system [1].

Since the cameras have limited processing and no OS, some capabilities of V are limited. For example, concurrency in V requires spawning threads, which V itself does not support without an OS [5]. Details about the reference counting garbage collector (i.e. number of objects it manages and its computational costs) are not available in the V documentation, so the additional costs associated with the GC may place a computational burden on our limited camera CPUs. Also, V does not yet support custom allocators, so fine-grained memory allocation would have to be performed by altering the V compiler source code.

Perhaps the biggest issue with V is that the language is not very mature. All documentation and source code is stored in a quickly evolving Github repository, and the language is not yet widely used [6] [1]. Furthermore, many functionalities are still under development, such as thread-based concurrency and translation between C and V [5]. V libraries are limited because most development is still being performed on the core of the language, so most library support would have to come from C libraries or implementations from scratch. Another issue with the lack of maturity and widespread use is that errors in the language are more likely. Since our cameras handle sensitive information, errors that compromise security or reliability would be extremely costly.

Overall, the simplicity, enforcement of standardized style, and discouragement of vulnerable and error-prone practices makes V a very attractive choice for SecureHEAT. The most significant drawback is V's lack of maturity and the implications that presents for error potential and framework support.

References

[1] V. D. Team, "The v programming language." https://vlang.io.

[2] A. Mondal, "Cs 131 week 9 discussion," May 2021.

[3] V. D. Team, "V documentation," Jun 2021. https://github.com/vlang/v/blob/master/doc/docs.md.

[4] V. D. Team, "Inline assembly code example," May 2021. https://github.com/vlang/v/blob/master/vlib/v/tests/assembly/asm_test.amd64.v.

[5] D. Angelov, "Faq," May 2021. https://github.com/vlang/v/wiki/FAQ.

[6] V. D. Team, "Github pull requests," June 2021. https://github.com/vlang/v/pulls.