Evaluate language D for secure camera-based HVAC control

Abstract
In this project, we examine the programming language D and whether it supports a proposed application well. In order to evaluate the features of this language, we examine the security, ease of use, flexibility, generality, performance, and reliability of D.

1 Introduction

This project is aimed to evaluate D supporting a proposed application, which is a HEAT system called SecureHEAT that uses inexpensive cameras to monitor the faces of human occupants, calculates the occupants' facial temperatures, and uses these calculations to control the buildings' air temperatures. However, a problem comes up that many potential customers manage buildings that require high security, so the software in the cameras is concerned with problem of being penetrated and leak information to adversaries.
Hence, programming language D is examined as an alternative of C/C++ to support the properties of this application, which involves that the software in the cameras must be easy to audit, freestanding program, simple and stripped-down, and capable of interfacing to low-level hardware devices.

2 Features of D

2.1 Security

There are three categories of functions from the perspective of memory safety: @safe, @trusted, and @system. system functions may perform any operation legal from the perspective of the language including inherently memory unsafe operations like returning pointers to expired stackframes. These functions may not be called directly from safe functions. Also, safe functions have a number of restrictions on what they may do and are intended to disallow operations that may cause memory corruption. For example, A scope parameter of reference type must not escape the function call (e.g. by being assigned to a global variable). It has no effect for non-reference types. scope escape analysis is only done for @safe functions. For other functions scope semantics must be manually enforced. So, attacks like memory injection would be prevented by D's safety design of memory management.

2.2 Ease of use

D is a simple but functional programming language, such that it shares many similar syntax with C or C++ that can be easily learned and implemented. Also, D has many useful libraries that can save a lot of low-level redundant implementation. For example, it can import C module such as module c.stdio, which is module stdio in the c package.
Besides, D is designed as object-oriented, since it can declare classes and instantiate them with objects. Also, it has various data structures, such as associative array(hash maps) and resizable arrays, as well as many supports for web framework such as TCP and UDP connections and database connection with MongoDB, etc.

2.3 Flexibility

In D, Data can be marked as const or immutable, with the default being changeable (or mutable). Therefore, we are able to easily tell which data can be expected to not change, which data might change, and who may change that data.
Also, D uses static type check, such that basically all variables are declared with types including bool, short, int, long, cent, char, float, void, and so on.

For array, the conversion is also very flexible. For example, a dynamic array, say x, of a derived class can be implicitly converted to a dynamic array, say y, of a base class if and only of elements of x and y are qualified as being either both const or both immutable.

2.4 Generality

As its name suggests, the initial motivation for D was to improve on C and C++ while keeping the features and purposes. Therefore, the efficiency, low-level access, and Algol-style syntax are all well preserved, so D has good generality such that it can handle problems in a similar way as C or C++. Also, D make obvious improvement focused on rapid development, convenience, and simplifying the syntax without hampering expressiveness.

Specifically, D is designed to fit comfortably with a C compiler for the target system. D makes up for not having its own VM by relying on the target environment's C runtime library. Moreover, C functions can be called directly from D by using keyword ‘extern (C)’. There is no need for wrapper functions, argument swizzling, and the C functions do not need to be put into a separate DLL.

2.5 Performance

One way to improve performance significantly for massive computation is multithreading. In D, the preferred way to do multi-threading is to rely on immutable data and synchronize threads using message passing. However, the language has built-in support for synchronization primitives as well as type system support with shared to mark objects that are accessed from multiple threads. For example, library like std.concurrency can be import and class can be declared with keyword synchronized, such that it can be used safely among different threads and all access to an instance of it is automatically locked.

2.6 Reliability

One concern for reliability is memory management. For D, it can explicitly allocate memory using core.stdc.stdlib.malloc() and core.stdc.stdlib.free(), which are useful for connecting to C functions that expect malloc'd buffers. Also, D managed to ensure that memory will not be collected by the garbage collector before the C function is done with it. To achieve this goal, D always make a copy of the data with its malloc function and only passes the copy instead, then it leaves a pointer to it on the stack (as a parameter or automatic variable), as the garbage collector will scan the stack, and registers the pointer with the garbage collector with the std.gc.addRoot() or std.gc.addRange() calls. Thus, the garbage collector does not scan the stacks of threads not created by the D Thread interface, nor does it scan the data segments of other DLLs.

3 Supporting Properties of Applications

3.1 Advantages

First, D is easy to audit, since it has built-in testing and verification as a core part of this language. For example, it has unit tests which examine test cases applied to a module to determine if it is working properly. A D program can be run with unit tests enabled or disabled. Also, D has strong error handling mechanism that can handle errors including out of memory, out of disk space, invalid file name, attempting to write to a read-only file, read a non-existent file, and requesting a system service that is not supported. Since the error handling is standardized and no errors get inadvertently ignored, D programs can safely run with reasonable audit and error precautions.

Also, D is capable of interfacing to low-level hardware devices, such as camera or network interfaces, since it has functional libraries for the connections. Also, D can directly call C functions by using keyword extern (C), so the connections can also be set up in the way of relying on C libraries, while it might not be safe.

Besides, D is able to handle multithreading situations. Since our application may involve parallel tasks and I/O to the same device or objects, the library for synchronization in D may be very helpful for processing such scenario.

3.2 Disadvantages

D appears to not be a good freestanding program, since it has abundant interface to C, C++, and objective-C. Also, its libraries to set up TCP

connections and database connection are also relying on other programming languages, so such dependencies might cause safety issues for preventing attacks in our applications.
Also, D is not significantly simple and stripped-down, since it has garbage collectors that scan through all copies of objects in the stack. Although there are safety implementation for setting the access permission for the memory and garbage collector, the mechanism itself is similar to high-level languages and might be vulnerable to attacks that may leak information, which is strongly concerned by our customers that require high security.

4 References

Memory-safe D Spec:
https://dlang.org/spec/memory-safe-d.html

Multithreading, synchronization and sharing for D:
https://tour.dlang.org/tour/en/multithreading/synchronization-sharing

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

Modules of D:
https://dlang.org/spec/module.html