| **Article** | https://www.techbang.com/posts/119558-the-us-governments-call-for-the-deprecation-of-c-c-will-usher |
|---|---|
| **Litany** | ● C and C++ are not "memory safe" programming languages, if not written carefully will cause a lot of problems.<br>● The companies and open-source projects are suggested by the CISA to switch from memory-unsafe languages (C/C++) to memory-safe languages. |
| **Systematic causes** | ● Memory safety vulnerabilities account for 70 percent of security vulnerabilities.<br>● C/C++ has been the prevailing programming language for a long time.<br>● C/C++ requires the developer to handle the memory manually. (you could do whatever you want with the system resources programming in C/C++, it's essentially chaos language)<br>● The memory-safe languages such as Rust, Java, Go, have built-in memory-safety (preventing the programs or programmers from creating common bugs).<br>● Memory-safe languages reduce the possibility of creating common errors with the built-in memory-safety (by putting restrictions in the programming language, not allowing the programmer to do whatever).<br>● It could take a lot of money, resources, time, and meticulous planning to re-write years of C/C++ codebase (proprietary or free) to memory-safe languages.<br>● Switching programming language means investing in and switching to new developing tools, environment, and debugger.<br>● Poorly written code, doesn't matter the programming language, leads to vulnerabilities in programs, fixing them or someone exploiting them are loss of money and time (sometimes trust if not dealt with correctly). |
| **Worldview** | ● Making the cyber world as a whole safer.<br>● Having an authoritative entity push the change could perhaps move things along faster.<br>● Less errors to correct equals less resource strain.<br>● Difficulty to reach mastery in programming and prevent all kinds of serious bugs.<br>● Memory-safe languages may not be as fast and efficient as C/C++.<br>● Senior C/C++ programmers don't really have a reason to switch language, especially the operating system kernel maintainers, since they know how to write good code.<br>● Steep learning curve, the memory-safe languages are very different from C/C++.<br>● Confronted with cost-effectivity and security in a short developing timeframe, security is typically an after-thought.<br>  ■ Programmers make programs with the required functions first, then wrap it with layers of protection. It's hardly a good practice, the protection could fail when someone finds a way to bypass it, or the protection fails to cover everything.<br>● The increase of long-term profit may be larger than the initial investment when it comes to switching to memory-safe languages.<br>● The change is not going to happen fast, maybe not in the 2020s, perhaps 2030s or 40s |
| **Metaphors** | ● Rome wasn't built in a day.<br>● You throw a rock out there, you hit a memory safety vulnerability. |

(not part of the table)

About the whole memory-safe languages and memory-unsafe languages, there's always a saying that "It's your skill issue not being able to write good and safe and secure code." Sure, why learn a higher-level language if you can code with 0 and 1, or assembly language (a level higher than 0 and 1), or C/C++ skillfully. They all can achieve the same result. However, it's been years, and the majority of programmers still haven't managed to write secure code, the memory safety issues still prevail.

I'd argue it's not incompetence, but the difficulty of learning all the possible scenario of bug (the program still runs), and the difficulty of debugging the bugs manually (checking bounds of memory ensuring that it doesn't overflow). Why not just build a common bug checker inside of a language (or compiler) and have it stop programmers from creating such bugs or tell programmers where such bug could be a serious problem. And that's one of the points of memory-safe languages, check/solve/prevent pre-existing problems for you so you don't have to do it manually.

(not part of the topic)

Aside from languages, it's sometimes comical how trivial (as in easy to exploit and get sensitive information) a CVE (common vulnerabilities and exposures) is, it makes you wonder who the heck coded this or did they test it in production. Thus the "Security is an after-thought" saying.

CVE-2024-10914 (https://netsecfish.notion.site/Command-Injection-Vulnerability-in-name-parameter-for-D-Link-NAS-12d6b683e67c80c49ffcc9214c239a07) This recent one happened in D-link devices has a critical 9.2/10 score, and all you have to do is type a one line command in the terminal with the right parameters, and voila! the machine is yours. D-link refuses to fix it and says those are end-of-life devices, asking the customers to buy newer models. It's not a memory safety vulnerability, but still a vulnerability. It's kinda funny but probably pretty scary to the shareholders and customers.