Software Assurance Tips

A product of the Software Assurance Tips Team[4]

Jon Hood

Monday 2nd September, 2024

1 Back to the Building Blocks: Codifying Complacency

Updated Wednesday 28th August, 2024

In February, the National Cyber Director, Harry Coker, presented the White House's strategy for combating cyber threats. Included in this strategy, the White House instructed technical practitioners to heed reports such as *Back to the Building Blocks: A Path Toward Secure and Measurable Software*.[1]

A core piece of the White House's strategy is to "eliminate entire categories of software vulnerabilities" by using programming languages which do not "lack traits associated with memory safety."[5, p. 7] While the White House should be applauded for recognizing the pervasive nature of memory safety vulnerabilities, there is an underlying flaw in the assertations of the recommendation. The report continues, "C and C++...are not memory safe programming languages. Rust [is] one example of a memory safe programming language..."[5, p. 9]

As a software assurance practitioner, I get to review a large amount of "memory-safe" Ada software and some Rust applications. To date, only a single project that has been provided to me for review written in either of these languages use only the memory safe features of the language. Over 99% of the projects I have reviewed either use `unchecked_conversions` in Ada[2] or raw pointers in Rust. Frequently, these projects also disable the additional memory safety checks explicitly while citing real-time and performance requirements. Contrast this to software written in C++ where it is not unusual to receive a project with enforced coding standards. Some of these coding standards can require developers to use smart pointers, RAII concepts, and memory-safe development practices. A C++ program written with such enforced coding standards will undoubtedly have fewer memory vulnerabilities than a Rust program using raw pointers.

A development organization shouldn't rely on the White House's misguided recommendations to use Rust instead of C++. There are certainly unique Rust features that afford memory safe programming design for developers. To program in Rust, a developer is required to understand its concepts of ownership. Its ownership system allows the compiler to make memory safety guarantees and enables the borrow checker. While modern C++ compilers have made large improvements to verifying lifetime ownership (such as Clang's -Wlifetime warnings), and several borrow checking implementations for C++ exist, very few projects use those features. Rust, however, was engineered around the borrow checker.

In conclusion, the White House did a tremendous disservice to the development community by not recommending the enforcement of RAII concepts[3], smart pointers, and secure coding standards. Software assurance practitioners should be prepared to evaluate Rust and other "memory safe" software applications with an additional level of scrutiny in verifying that the memory safety protections have not been bypassed. Furthermore, DoD project offices have a history of ignoring memory safety issues when software is written in a language such as Ada or Rust, citing misguided directions like this one from the White House. When we deliver a report with identified memory overflows, the project office's reflex is frequently to ignore the issue. Project offices could become complacent, and the issues may be swept under the rug.

In a worst-case scenario, thirty years from now, some software assurance team is going to get a critical, Rust-developed application riddled with assembly and raw pointer memory vulnerabilities. They will ask why the project office did not heed the software assurance scans conducted decades earlier. The project office will answer, "We were told this was a safe language and that we weren't vulnerable to those issues,"–the same excuse we receive from project offices with legacy Ada code today. Hopefully, these project offices implement the additional memory safe hardware and formal method recommendations from the White House.

References

[1] Harry Coker. National Cyber Director and Assistant Director. White House. 2024. URL: https://youtu.be/xVYSvkogoUM?si=7i5_GbuShOJ4ShCt&t=259.

[2] Jon Hood. "Ada Unchecked Conversions". In: SwATips.com (2023). URL: https://www.swatips.com/articles/20230410.html.

[3] Jon Hood. "Sticking with a RAII Standard". In: SwATips.com (2021). URL: https://www.swatips.com/articles/20210412.html.

[4] Jon Hood, ed. SwATips. https://www.SwATips.com/.

[5] The White House. Back to the Building Blocks. A Path Toward Secure and Measurable Software. 2024. URL: https://www.whitehouse.gov/wp-content/uploads/2024/02/Final-ONCD-Technical-Report.pdf.