Thesis Project Portfolio

Stack Faults: Memory Access Permissions for Stack Frames

(Technical Report)

Rust’s Role in Developing a Security Mindset While Preserving Developer Autonomy

(STS Research Paper)

An Undergraduate Thesis

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Ratik Mathur

Spring, 2024

Department of Computer Science

Developers are generally unaware of security vulnerabilities and/or lack the skills to defend their applications against them. Thus, modern tools such as operating systems and programming languages have baked in security; that is, when executing a program or application, they insert additional instructions to enforce security. As there is literally more code to be executed, this slows the performance of the code. Thus, developers sometimes feel frustrated that the code is not doing exactly what they want it to and that it limits their own optimizations they might choose to make to pick between speed and security. Keeping that in mind, the question to ask is how can the trade-offs between performance and security in software development be navigated to address both technical and societal concerns, considering the impact on developer autonomy, as well as the overall security landscape? My technical report is a proposal for inducing a hardware crash when unauthorized data is accessed by a portion of the program that shouldn't be. My STS thesis conducts a literature review to analyze the development of Rust, a recent systems programming language with the option to disable security checks, and see if it helps generally bring about a security mindset with limited hindrance on developer autonomy.

Buffer overflows are a common security vulnerability that plague many code bases, notably those written in C/C++, by writing data into a buffer past its capacity which may replace important data on the stack. For my technical report, I propose a stack frame-level memory permission check for data on the stack, inspired by page faults that are process-level memory permission checks. When a function call is made, an identifier for the new frame is pushed onto the stack. This identifier is checked whenever we write to the stack and, if a change in identifier is detected, this must be due to a buffer overflow and thus the program "faults" with a memory permission error. This would serve as a method of mitigating Buffer Overflow attacks with minimal performance degradation, ideally similar to bounds checking on the buffer. Future work would include extending the idea of numbering frames to get much faster than bounds checking (such as through the hardware), as well as expanding this concept of faulting the program to other areas that are vulnerable to unauthorized memory access.

Older languages such as C require the developer to explicitly manage what memory is allocated and deallocated for data, however, this is rather challenging to do properly even for skilled developers which leads to most application crashes being due to memory errors. Successor languages often have systems to manage memory under the hood, but this significantly slows the program down. Rust introduces an ownership system which assigns each piece of memory an explicit owner and simplifies memory management by rejecting to build programs that *might* be unsafe. The creators of Rust claim that this helps eliminate entire classes of errors. However, this overconservative approach limits the programs that can be written including operating systems. Thus, Rust also contains an unsafe mode that relaxes many of these restrictions to allow developers to build more robust applications. Developers wish to have autonomy in making their own tradeoffs regarding speed and security, but they still have a technomoral duty to protect the interest of society by developing secure applications. This creates a self-conflicting sociotechnical system and creates a niche in the development ecosystem that needs to be fulfilled. Using the lens of SCOT to view the development of Rust (and programming languages in general), I evaluate whether Rust shows promise in fulfilling this niche.

In conclusion, my technical report was approved as a sufficient proposal for buffer overflow mitigation, and I also presented future work in the area such as exploring other methods of adding stack-based permissions or extending this idea to beyond the stack. My STS thesis concluded that Rust has been shown to reduce memory vulnerabilities in live industrial applications and generally seems to be increasing security awareness among developers despite many claiming that their primary reason for using unsafe Rust was to achieve performance speedups. This further establishes that Rust is a promising solution to the niche created from the intersection between developer autonomy and securing the interests of society.