Feasibility Study on
Memory Access Permissions for Stack Frames to Reduce Vulnerabilities to Buffer Overflow Attacks

(Technical Topic)

What Level of Freedom is Ethical to Provide Developers in Making Their Own Decisions About Performance and Security

(STS Topic)

A Thesis Prospectus
In STS 4500
Presented to
The Faculty of the
School of Engineering and Applied Science
University of Virginia
In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science in Computer Science

By
Ratik Mathur

On my honor as a University student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments.

ADVISOR
Kent A. Wayland, Department of Engineering and Society
Nada Basit, Department of Computer Science

General Research Problem

Securing Systems with Modern Performance Demands

As the global landscape has shifted to a cyber-age, the internet has become a crucial component to global infrastructure. Malicious actors have turned their focus to exploiting any gap or oversight in software applications to achieve their means. Thus, defending against cyber-attacks has become an upmost priority for the modern-day software developer… or at least it should be.

While the system being highly secure should be mandatory, both developers and project managers often push security to the side as something "nice to have" or as a "great add-on" (Lim, et. al. 2009). From the development perspective, implementing security requires significantly higher technical expertise than just simply creating an app. Often developers are not even aware of the current security landscape and thus don't even know what types of security they should be implementing (McGregor 2019). Additionally, as developers have security knowledge gaps (often because they are either self-trained or their degree didn't teach them), app development teams often need security specialists if they want to secure their app which would result in a higher cost (Naiakshina, et. al. 2020). As far as managers are concerned, they want an app that runs smoothly with negligible latency; not unlike developers, they may also be blissfully unaware of the security needs of their app (Shreeve, et. al. 2023).

The lack of awareness and understanding of the security landscape is highly common among developers leading to insecure applications. Looking at UVA, all security focused classes are electives; cybersecurity is not required for any computing program at UVA.

Baking in security by default into computer systems takes the choice away from developers regarding speed and security and intentionally slows down the computer. Some types of applications can choose to be less secure for speed, especially in a testing environment or for personal projects.

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 developers and managers, as well as the overall security landscape?

To explore this question, I will be undertaking a technical Feasibility Study as well as conduct a sociotechnical analysis of the performance vs security debate in an STS context. My feasibility study is about implementing security close to or within the hardware to address a common security vulnerability which is typically solved in software but slows applications. My analysis will consider what would be considered an ethical level of freedom to give developers regarding implementing security given that they oversee critical global infrastructure.

Technical Topic

Feasibility Study on Memory Access Permissions for Stack Frames to Reduce Vulnerabilities to Buffer Overflow Attacks

Older languages, such as C and C++, are still very relevant for today's developers. Most computer systems are built with backwards compatibility in mind such that systems from decades ago may run as intended on a modern system. These programming languages work rather closely to the underlying hardware and provide a convenient abstraction for the hardware. This means that they require the developer to tell the computer exactly what to do and thus make the developer responsible for securing their code properly to prevent vulnerabilities.

A notable vulnerability with this model is the "buffer overflow" attack. Buffer overflows occur when the amount of data written to a location (the buffer) exceeds the expected storage space the computer has allocated for that data (overflowing it). For an example: if a program wants to take the user's name as input and then display it to the user, the developer might allocate 10 bytes on the server to store that name; but without extra precautions taken, an end user of the program could type in 1 MB of data and this would all get written to the server which would likely overwrite important data.

All (useful) programming languages will let you place lines of codes into a "function" that can then be referenced by the name given to it; this is called "calling a function" and a section of code that performs this is referred to as a "caller." This simultaneously helps introduce abstraction but also enables code reuse without having to type it out again. Every time the code calls a function, as the function is likely in a different section of memory, the code must make sure to save all important data about the function's caller before "jumping" to the memory location where the desired function is. This data is placed into what is called a "stack frame" within a part of the computer's memory called the "stack." Notable among this data is the caller's "return address," which specifies exactly where in the computer's memory the caller is located. The return address is very important as it is used to return to the caller after the called function has finished its execution.

Buffer overflows, like the example mentioned above, work by overwriting the data in other stack frames. If an attacker can determine how the stack frames are laid out, they could attempt to overflow exactly enough data that they overwrite the return address with malware that opens a backdoor. When a function is done executing, the computer will look at whatever data is in the slot the caller's return address **should** be in, unconditionally jump to that memory address, and begin executing any code it locates there.

To defend against such attacks, developers either need to add additional instructions to properly check the data before writing it to a buffer or use a language that will insert these instructions for you. This slows the application down as there is literally more code to be run, and the solution is through software instructions which are much slower to execute than hardware instructions.

To mitigate these vulnerabilities, I will be increasing the granularity of permissions for data within stack frames. If a buffer overflow attempts to overwrite the return address, which is in the previous stack frame, it will fail to do so as it doesn't have the permissions. More specifically, the hardware will crash the entire program since the program attempted to access invalid memory. This idea is like the concept of "Page Faults" which effectively does this but for entire programs: they prevent other programs from touching memory they shouldn't have. I wish to introduce "Stack Faults" that crash the program if a stack frame modifies memory it shouldn't have. However, while the return address should never be modified by a buffer write, it is rather common for functions to modify other data (including other buffers) from different stack frames (this is called pass-by-reference). A simple example would be an update_username function, which should modify the memory address at which the username lives, which would be in a previous frame.

To address this issue, I will specifically be concerning myself with the scenario of writing across stack frames. That is, beginning to write to a buffer in one stack frame and then end up writing to memory in another stack frame. This way, valid use cases for passing by reference will be allowed as the writing would occur within the appropriate frame.

To achieve this, I will be modifying the source code for an operating system to keep track of which frame is being written to anytime data is placed on the stack. If a change in frame is detected, this would almost certainly be due to a buffer overflow as the stack frames should have adequate space allocated in them to support all the data a buffer would require. I will edit the code to send a signal to the operating system, referred to as "interrupting" the OS, to crash the current process when a change in stack frame is detected.

STS Topic

What Level of Freedom is Ethical to Provide Developers in Making Their Own Decisions About Performance and Security

I will be attempting to unravel the ethical considerations associated with giving developers the autonomy to make decisions about performance and security, all while considering the cultural and organizational influences shaping these choices.

Implementing security within computer systems by default, such as how I am in my technical topic, offers several pros and cons. Primarily, it would ensure that developers not familiar with the current security landscape still develop secure applications especially when dealing with industries such as healthcare or finance. It ensures basic security across the board which can help protect against common vulnerabilities. However, when security is integrated in the system, developers may find their applications running slower or be limited by how much optimization they can do. These tradeoffs between speed and security may not align with the specific needs of an application, especially if in a test environment or in a personal project.

I will be analyzing my chosen topic with the aid of Pinch and Bijker's STS theory known as SCOT: Social Construction of Technology. SCOT surmises that "culture and society shape technological development" as opposed to the traditional notion of "technology shaping society" (Pinch & Bjiker, 1987). Technological artifacts, according to SCOT, embody their cultural context, making this theory a useful lens for examining the security landscape. This is especially relevant here as I will be using the cultures and attitudes of developers, managers, and organizations as a whole to evaluate what level of security should be implemented by default in platforms used for app development.

Incorporating the lens of SCOT, the sociotechnical analysis will explore how organizational culture and developer communities affect the attitudes of developers. SCOT provides a framework to understand the development of technology *with* society, emphasizing the idea that these two influence each other.

The ongoing debate surrounding the freedom for developers to tailor security based on their application's unique requirements versus enforcing standardized security at the system level warrants an exploration. Useful to consider would be an assessment of the security skills and knowledge gap among developers, and that they are socially constructed and influenced by higher education (namely computer science degrees), online development communities, and the attitudes of developers' organizations.

An interesting concept presented by Lim, Joo S., et al. (2009) is that of OC: Organizational Culture. "OC refers to the systems of shared beliefs and values that develops within an organization and guides the behaviors of its members" and is typically formed by "the behaviors of dominant organizations members like founders and top management" (Lim, Joo S., et al. 2009; Schein 1992). They discuss how their literature exploration suggests that developers' attitudes are largely developed by the culture of the organization they work for and how that culture often does not include a security culture. This can be for various reasons like the ones I have mentioned earlier. This results in an environment that does not actively encourage security, which should be unacceptable for modern Internet-Scale Applications. This directly draws on the principles of society shaping technology that are prevalent in SCOT.

To actually go about conducting this analysis, I will research how security has been integrated into applications of varying scales ranging from healthcare apps to minor tools written by independent developers hosted on GitHub. By nature of publishing on GitHub, the developer likely wishes for the tool to be adopted by the development community and should thus ensure a secure application is hosted (or at least has documentation on security vulnerabilities. GitHub supports a feature known as "Pull Requests" which allow anyone to modify **a copy** of the source code and then request to merge their changes in with the original source code; the owner of the project would then review the proposed changes and either accept or deny the merge. Looking at security-focused pull requests. As well as security-related issues raised, will serve as some of the more important statistics as I hope they will reveal a common set of security vulnerabilities that developers have to be reminded of by others. More specifically, they may also reveal whether developers simply have an awareness of the issue (issues raised) or the actual skills to address

them (pull requests). Commonalities among issues raised without pull requests may be a good candidate for enforced security.

This addresses the impact of the development community. Moving on to higher education, I will interview professors at UVA, including security professors and core curriculum professors. This will help reveal what forms of security professors choose to emphasize in their classes and which ones they don't. Of the ones that aren't, some will be meant to be integrated into code and thus should be the responsibility of the developer; these may be potential candidates for enforced security. There may be other forms of security not emphasized that are meant to be for the system and not actively integrated into code, these are the focus of system administrators and security specialists and thus aren't the focus of this analysis.

I will also interview fourth-year CS majors, including those who do and do not wish to pursue security careers to assess how they feel about their understanding of the security landscape. This would provide perspective on whether rising developers would prefer to have security taken care of or would rather implement it manually. I would ask them not only about specific vulnerabilities but also to identify vulnerabilities in source code that seem to satisfy all functionality requirements. Permitting that I get enough volunteers, I may ask students to implement simple coding tasks that may introduce vulnerabilities if not properly addressed. These would again be good candidates for enforced security.

The third important group that influences security attitudes to consider is, as mentioned above, organizational culture. Unfortunately obtaining information on organizational culture within companies and how that impacts security-driven development is rather challenging to obtain. For this I will analyze publicly available information about Amazon through the lens of my personal experience working at Amazon, a company that chants the mantra "security is job zero," as well as from gathering the experiences of other students who have held tech internships.

By gathering such data, I will be able to make conclusions about how security and speed are prioritized at varying levels of application scale and importance. This would help reveal which areas of security are lacking in developers' skills. I am aiming to understand what kind of trade-offs are often made for performance vs security, including those made either subconsciously or due to a lack of understanding. These latter cases are especially important and are what would warrant enforced security in the first place. What transforms this into a debate of what forms of security should be enforced into one about the ethics of doing so, is that I am simultaneously talking about actively stifling developer freedom and creativity while keeping in mind that there is mission critical software at play (such as in the healthcare or finance industries). It can be considered unethical to actively limit the code that can be written, but also unethical to even leave open the possibility of unsecured critical software. And thus analyzing this will help me answer what is an acceptable level of freedom to give developers regarding making their own decisions about speed vs security.

Conclusion

The general research problem, which revolves around implementing security in software development while considering modern demands, remains a pressing concern in the ever-evolving digital landscape. As technology continues to advance and malicious actors target software vulnerabilities, addressing these issues becomes paramount.

To address this, a feasibility study and sociotechnical analysis will be conducted. The technical topic focuses on a feasibility study to address memory access permissions for stack frames, aimed at reducing vulnerabilities to buffer overflow attacks. Doing this will further compound my understanding of operating systems and, in particular, OS security. Gaining such an understanding will help increase my awareness of the security landscape as a developer. It will also help me understand how operating systems function, which would enable me to better leverage computer systems in a secure manner. This is also very useful for both academia and industry. Security-focused researchers with more experience and resources would be able to expand on the work I will do and possibly explore other (possibly faster) implementations of stack faults. Industry would, consciously or not, welcome a more efficient method of protecting against buffer overflows. To the average industry app developer, such a change may not even be noticeable other than updating the OS which makes this rather important for those who lack an understanding of the security landscape.

Meanwhile, the STS topic explores the ethical considerations surrounding the level of freedom provided to developers in making decisions about performance and security by using SCOT to analyze the current state of the dynamics between developer culture, organizational culture, and security choices.

Discussions of this nature are necessary as they underscore the lack of security integration in computer science education and the often-underestimated importance of security awareness for developers. They also highlight the challenges of balancing speed and security in programming languages, offering a glimpse into the decisions faced by developers when choosing the right tools and approaches.

References

Alhozaimy, S., Menascé, D. A. (2022). A formal analysis of performance-security tradeoffs under frequent task reconfigurations. *Future Generation Computer Systems, 127, 252-262*. https://doi.org/10.1016/j.future.2021.09.005.

Hess, A., Sengupta, S., & Kumar, V. P. (2008). Joint Traffic Routing and Distribution of Security Services in High Speed Networks. *IEEE INFOCOM 2008 - The 27th Conference on Computer Communications, 2279-2287*. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4509891.

Jordan, T. B., Johnson, B., Witschey, J., Murphy-Hill, E. (2014). Designing Interventions to Persuade Software Developers to Adopt Security Tools: Proceedings of the 2014 ACM Workshop on Security Information Workers. *Association for Computing Machinery (ACM), SIW '14: Proceedings of the 2014 ACM Workshop on Security Information Workers, 35-38*. dl.acm.org/doi/10.1145/2663887.2663900.

Lim, J S., et al. (2009) Exploring the Relationship between Organizational Culture And Information Security Culture. *Proceedings of the 7th Australian Information Security Management Conference*, https://ro.ecu.edu.au/cgi/viewcontent.cgi?article=1011&context=ism.

Mcgregor, L (2019). Gamification and Collaboration to Evaluate and Improve the Security Mindset of Developers. *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, 342-343. https://dl.acm.org/doi/pdf/10.1145/3304221.3325593.

Naiakshina, A., Danilova, A., Gerlitz, E.,, and Smith, M. 2020. On Conducting Security Developer Studies with CS Students: Examining a Password-Storage Study with CS Students, Freelancers, and Company Developers. *In Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems (CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3313831.3376791.

Narayan, S & Kolahi, S & Waiariki, R & Reid, M. (2008). Performance analysis of network operating systems in local area networks. *2nd WSEAS International Conference on Computer Engineering and Applications*. https://www.researchgate.net/publication/228740465_Performance_analysis_of_network_operating_systems_in_local_area_networks.

Pinch, T. F., & Bijker, W. E. (1987). The Social Construction of Facts and Artifacts: Or How the Sociology of Science and the Sociology of Technology Might Benefit Each Other. In W. E. Bijker, T. P. Hughes, & T. F. Pinch (Eds.), *The Social Construction of Technological Systems: New Directions in the Sociology and History of Technology* (pp. 17–50). MIT Press.

Schein, E. H. (1992). Review of *Organizational Culture and Leadership*. *The Academy of Management Review, 11*(3), 677–680. https://doi.org/10.2307/258322.

Shreeve, B., Gralha, C., Rashid, A., Araújo, J., and Goulão, M. 2023. Making Sense of the Unknown: How Managers Make Cyber Security Decisions. *ACM Trans. Softw. Eng. Methodol*. 32, 4, Article 83 (May 2023), 33 pages. https://doi.org/10.1145/3548682.

Tahaei, M., Vaniea, K. E., Beznosov, K., & Wolters, M. K. (2021). Security Notifications

in Static Analysis Tools: Developers' Attitudes, Comprehension, and Ability to Act on Them. *Association for Computing Machinery (ACM), The ACM CHI Conference on Human Factors in Computing Systems 2021, 691, 1-17*. https://doi.org/10.1145/3411764.3445616.

Yasumatsu, T., Watanabe, T., Kanei, F., Shioji, E., Akiyama, M., Mori, T. (2019). Understanding the Responsiveness of Mobile App Developers to Software Library Updates. *Association for Computing Machinery (ACM), 13-24*. https://doi.org/10.1145/3292006.3300020.