

# Assessing the Potential Impact of Memory-Safe Languages for System Software Security

**Investigators:** Daniel Molsbarger, George Kotti, Kenna Henkel, Kaneesha Moore,  
Prathyusha Mustiyala, Wilson Patterson

**Faculty Mentor:** Dr. Sudip Mittal  
Mississippi State University

**Problem Mentor:** Neal Ziring, National Security Agency

## Executive Summary

Much emphasis in today's software development efforts has been placed on the inherent systemic vulnerabilities found in popular programming languages such as C and C++. A fresh generation of "memory-safe languages" has been created to address some of the systemic flaws in popular programming languages.

The goal of this research is to search the computer science literature for crucial elements of these memory-safe languages. Understand the vulnerabilities listed in the Common Weakness Enumeration (CWE) and the National Vulnerability Database (NVD). To see if the same security vulnerabilities exist in memory-safe languages like Rust, Go, Kotlin, and Python, compare particular code written in C to analogous code written in memory-safe languages like Rust, Go, Kotlin, and Python.

## Introduction & Motivation

In today's software development efforts, much attention has been focused on the intrinsic systematic vulnerabilities present in popular programming languages like C, C++. A newer generation of "memory-safe languages" have been developed that overcome some of the systematic vulnerabilities present in popular programming languages. The development of these newer generation of languages like Rust, Go and Python have greatly augmented the field of *Secure Software Development*.

A "memory-safe language" is a language where memory allocation and garbage collection are abstracted away from the developer and handled by the programming language itself [1]. Low-level system programmers are being urged to adopt newer "memory-safe" programming languages, stemming from the ongoing battle in memory-related issues due to current development tactics. According to Meza et al. [1], memory-related vulnerabilities have remained a pressing issue for decades. Memory-related vulnerabilities include memory leaks, buffer overflows, dangling pointers, pointer arithmetic, stack overflow, memory fragmentation, and many more. Remarkably, despite the ongoing concerns related to memory vulnerabilities, programming languages like C and C++—which have been notorious breeding grounds for such issues—remain steadfastly positioned among the top three most utilized languages as of August 2023. System programmers still gravitate towards C/C++ due to their familiarity and perceived ease of use. However, a recent empirical study by Pirelli et al. challenges this notion, suggesting that it's feasible to attain safety without introducing significant runtime overhead [2]. The study indicates that memory-safe languages, reminiscent of C/C++, can offer a promising alternative. Hence, there is a crucial need to examine and further develop memory-safe languages by exploring their ability to detect and prevent said vulnerabilities, to counteract this worldwide problem in technology.

**Research Aims:** In this research, the investigators aim to survey common security flaws found in the C, C++ languages and present how memory-safe languages like Rust, Go, Kotlin, and Python overcome these flaws. The objective of this work is to examine the computer science literature for key aspects of these memory-safe languages. Understand the relevant vulnerabilities posted on the Common Weakness Enumeration (CWE), National Vulnerability Database (NVD). Compare specific code written in C to equivalent code in the memory-safe languages Rust, Go, Kotlin and Python to evaluate if the same security vulnerabilities appear in the memory-safe languages. To do this, the team will need to write that equivalent code for each language, and that code will be presented in the findings. The additional research delivered by our team strives to aid in mitigating unnecessary memory-related vulnerabilities by utilizing weaknesses found within the CWE and evaluating equivalent code across all languages chosen for this study.

To achieve this research goal, we have divided our methodology in *two research tasks*.

## Research Tasks & Plan

We break our approach in the following two tasks:

**Task 1: Literature Review and Survey of Memory-Safe Languages** The team will choose 2 popular memory-safe languages and examine the literature in computer science to find important characteristics of these languages. Look through the National Vulnerability Database to get a sample of software vulnerabilities in C/C++. For this survey, we will also present some of the bugs and exploits commonly found in C-programmed code using sources such as the Common Weakness Enumeration (CWE), and briefly discuss how these security flaws have affected the software industry. Afterwards, a minor exposition on the memory-safe languages will be covered, followed by an analysis on the behavior of these languages when exposed to similar code.

**Task 2: Recreate, Code and Test Vulnerabilities in C and Memory-Safe languages** Our

initial plan is to gather sources of information regarding common security flaws found in the C and C++ languages. While doing so, the team members will also train themselves in Rust, Go, Kotlin, and Python to write the equivalent, flawed code in those four memory-safe languages. From here, the team will test for the success of the compilation and for the flaws found in the C language. It is expected for the same exploits in the C language to not be found in any of the stated memory-safe programs.

## **Intellectual Merit**

Over time, an increasing number of security threats have arisen from poor execution of programming languages with manual memory management, but traditionally, the C language has been substantially used in the industry despite these same shortcomings. This has often led to sustained damage from outside influences/actors when there are unexpected events in interactions with the program, and has resulted in high financial harm for many product owners/companies. With this, it seems worth investigating the potential of memory-safe languages in preventing these issues. By exploiting flaws discovered in the CWE and assessing equivalent code across all languages selected for this study, the research presented by our team aims to help mitigate unnecessary memory-related software security issues.

## **Broader Impacts**

By investigating and promoting memory-safe programming languages, the research has the potential to enhance software security by reducing memory-related vulnerabilities such as memory leaks and buffer overflows. The implementations could improve software development practices, advance technology, raise public awareness, and create safer and more reliable software systems. Ultimately, this research contributes to the overarching goal of creating a more secure and resilient digital landscape, benefiting individuals, organizations, and society as a whole.

## **Expected Outcomes & Deliverables**

Upon the completion of this survey, our team anticipates consolidating our results in a comprehensive report detailing our research findings, in-depth vulnerability analyses, and supportive examples of vulnerability mitigation from the adoption of a memory-safe language. In all, our team will have demonstrated the benefits of reducing memory-based vulnerabilities through the use of memory-safe languages, aiding in the overall goal of improving our cybersecurity posture. In the completion of this project, the efforts exerted by our team of researchers will further the advancement of the overall project by delivering a presentation of our findings.

The list of expected outcomes would be:

- A comprehensive report that presents research findings and analysis related to the use of memory-safe programming languages
- An insight into the potential benefits of adopting memory-safe programming languages for reducing memory-based vulnerabilities in critical software
- A vulnerability analysis: in-depth understanding of the various memory-based vulnerabilities which arise from using less memory-safe programming languages (C/C++)
- A comparative analysis of Rust, Go and Python (at-least 2) memory-safe programming languages, based upon their effectiveness in eliminating the identified vulnerabilities

- Detailed examples illustrating how said vulnerabilities can arise from unsafe programming languages and how they can be mitigated using the selected memory-safe programming languages

## Logistics

The research will begin with an initial literature review. Next, the team will then familiarize ourselves with the CWE category system and identify relevant memory-associated CWEs; we will also determine our target languages (memory-safe and manually managed). In the middle of the term, we will deliver a presentation reporting on our research progress.

For the remainder of the term, we will implement and demonstrate our research empirically. We will begin to extract data from the National Vulnerability Database, and vulnerabilities will be recreated and tested in memory-safe and manually managed languages. We will analyze the findings and prepare a report. The research will conclude with a final presentation in which we communicate and discuss our findings.

| Date                              | Objective                                               |
|-----------------------------------|---------------------------------------------------------|
| August 31 - September 6, 2023     | Complete research proposal                              |
| <b>September 7, 2023</b>          | <b>Submit research proposal</b>                         |
| September 8 - September 15, 2023  | Literature Review, Familiarize with CWE category system |
| September 18 - September 22, 2023 | Identify memory-associated CWEs                         |
| September 20 - October 12, 2023   | Identify target languages                               |
| October 2 - October 12, 2023      | Prepare and refine midterm presentation material        |
| <b>October 13, 2023</b>           | <b>Present midterm research progress</b>                |
| October 16 - October 27, 2023     | Extract data from National Vulnerability Database       |
| October 30 - November 10, 2023    | Recreate, Code and Test vulnerabilities                 |
| November 13 - November 24, 2023   | Analyze performance and results                         |
| November 13 - December 4, 2023    | Draft comprehensive report on the findings              |
| November 30 - December 7, 2023    | Prepare and refine final presentation material          |
| <b>December 08, 2023</b>          | <b>Deliver final presentation</b>                       |

## Conclusion

This research initiative, focused on evaluating the potential impact of memory-safe languages on system software security, addresses a critical need in the face of escalating threats tied to memory management. With this research plan and a clear objective to systematically compare vulnerabilities identified in the CWE, we anticipate producing results that provide critical insights into the potential benefits of embracing memory-safe programming for enhanced system software security. By exploring the effectiveness of memory-safe languages for preventing memory-related vulnerabilities, this research stands to significantly advance the field of software security and strengthen the overall integrity of system software.

## References

- 
- [1] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, “Revisiting memory errors in large-scale production data centers: Analysis and modeling of new trends from the field,” in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2015, pp. 415–426.

- [2] S. Pirelli and G. Candea, “Safe low-level code without overhead is practical,” in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, 2023, pp. 2173–2184.
- [3] S. Wan, M. Sun, K. Sun, N. Zhang, and X. He, “Rustee: Developing memory-safe arm trustzone applications,” in *Annual Computer Security Applications Conference*, ser. ACSAC ’20. New York, NY, USA: Association for Computing Machinery, 2020, p. 442–453. [Online]. Available: <https://doi.org/10.1145/3427228.3427262>
- [4] H. Xu, Z. Chen, M. Sun, Y. Zhou, and M. Lyu, “Memory-safety challenge considered solved? an in-depth study with all rust cves,” 2021.
- [5] S. Nagarakatte, J. Zhao, M. M. Martin, and S. Zdancewic, “Softbound: Highly compatible and complete spatial memory safety for c,” in *Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI ’09. New York, NY, USA: Association for Computing Machinery, 2009, p. 245–258. [Online]. Available: <https://doi.org/10.1145/1542476.1542504>
- [6] —, “Softbound: Highly compatible and complete spatial memory safety for c,” *SIGPLAN Not.*, vol. 44, no. 6, p. 245–258, jun 2009. [Online]. Available: <https://doi.org/10.1145/1543135.1542504>
- [7] A. Mambretti, A. Sandulescu, A. Sorniotti, W. Robertson, E. Kirda, and A. Kurmus, “Bypassing memory safety mechanisms through speculative control flow hijacks,” in *2021 IEEE European Symposium on Security and Privacy (EuroSP)*, 2021, pp. 633–649.
- [8] A. E. Michael, A. Gollamudi, J. Bosamiya, E. Johnson, A. Denlinger, C. Disselkoen, C. Watt, B. Parno, M. Patrignani, M. Vassena, and D. Stefan, “Mswasm: Soundly enforcing memory-safe execution of unsafe code,” *Proc. ACM Program. Lang.*, vol. 7, no. POPL, jan 2023. [Online]. Available: <https://doi.org/10.1145/3571208>