# How the Adoption of Rust has Impacted Memory Related CVE's in IoT Devices

Schulman, Derek
*NYU Cyber Fellows (of Aff.)*
*Tandon School of Engineering(of Aff.)*
Brooklyn, USA
derek.schulman@nyu.edu

***Abstract*—In this paper we will investigate memory vulnerabilities of IOT devices, and the impact memory safe languages have on their Common Vulnerabilities and Exposures (CVEs). We have pulled data from the National Institute of Standards and Statistics National Vulnerability Database (NVD), and cross-referencing this with an estimated adoption rate of the Rust programming language based on Google queries of Rust tutorials over time. Based on our analysis, we can see a significant drop in memory vulnerability CVEs for IOT devices.**

***Index Terms*—Memory Vulnerabilities, Buffer Overflows, IOT Devices, CVE, NIST NVD,**

## I. Introduction

Currently, there are an estimated 21.5 billion Internet of Things (IoT) devices in the world [1]. In North America, 70% of homes have a network connected device. [2]. The firmware for these devices has traditionally been written in C/C++ because of the low footprint and performance benefits [3], but memory unsafe languages leave them open to memory vulnerabilities including buffer overflow attacks.

To try and mitigate these vulnerabilities some companies have turned to using memory safe compiled languages like Rust. The question then arises; Does the use of memory safe languages have a measurable effect on the overall population of IoT devices? This paper hypothesizes that IOT devices that have micro-kernels/firmware written in Rust (or other modern memory safe languages) will have less buffer overflows or other memory management vulnerabilities when compared to micro-kernels/firmware written in more traditional languages like C/C++.

While the hypothesis is not new, we propose a novel approach to measuring this anticipated decline in memory vulnerabilities. We are taking a data analysis approach to this question, and looking to measure the decline in memory vulnerability NIST CVE's reported, and compare that to the adoption rate of Rust year over year. We anticipate this will show a reduction rate at least proportionate to the adoption rate.

To examine this hypothesis, we will adhere to the following organizational structure: Section 2 will look at some of the current literature related to memory vulnerabilities in IoT devices and how others have chosen to measure the impact of memory safe languages on them. Section 3 will recount the methodology used to investigate the hypothesis. Section 4 will review the results of the study. Section 5 will introduce our conclusion and outline future work that we believe would further our work.

## II. Related Research

Authors Kelsey et. al. conduct surveys to understand the benefits of implementing Rust. Their team surveyed 16 (mostly senior) developers who actively worked with Rust on their teams [4]. While this is a good way to get direct feedback from those implementing Rust, it is limited to those willing to participate in surveys. The sample size is not enough to make larger scale summaries of the state of Rust adoption or its impact on memory issues. Our approach looks directly at a much larger dataset, and is able to infer general benefits as a result of data analysis.

Authors Xu et. al. looks at 168 bug report/ CVEs ending in 2020-12-31. This type of analysis is what we are looking to continue. By looking at CVE's from 2020 through 2024, we are looking at trends that extend beyond the scope of Xu et. al.'s work, taking it into the present day. Xu et. al. also goes on to do manual analysis of the individual bugs identified in the CVE's [5]. While this is a valuable method for deeper understanding of the memory issues that affect code written in Rust, their work focuses on all the bugs they found for memory issues implementing "unsafe" code, we are looking more holistically at the improvements that Rust offers versus traditional unsafe languages like C/C++.

Authors Sharma et. al. look at a dataset of 6,408 Rust embedded software packages in addition to conducting a survey with 225 developers. They conduct code analysis on the packages to determine vulnerabilities, and used the surveys to understand slow adoption of Rust for embedded applications [6]. While this is a very thorough approach, as close to 6.5K packages is not insignificant, it is still limited to the efforts of their team. Our approach of pulling CVE data from 2020 to 2024 will allow us to see the effects of Rust on memory vulnerabilities at a still larger scale.

## III. Methodology

The hypothesis of this paper is that IOT devices that have firmware written in Rust will have less memory vulnerabilities when compared to firmware written in C/C++. To measure this, we chose to look at the number of CVEs reported

in the NIST National Vulnerabilities Database (NVD) and compare this with the adoption rate of Rust. To determine the adoption rate of Rust, we relied on data pulled from PopularitY of Programming Language[1] which chooses to measure the popularity of a programming language by how often tutorials are searched on Google [7].

Our primary data set was the NIST NVD. Using Python, we pulled all the CVEs from 2020 through the end of July 2025 (approximately 168,000 records). Our next step was to filter the records, which was done using Python in Google's colab environment [2]. It was determined that any CVE determined to be "Rejected" was to be removed from the dataset. Additionally, we only wanted CVEs related to Internet of Things, and later we would also filter down to only items that were related to IOT and contained reference to Memory Vulnerabilities, this would be done by choosing keywords and removing entries that did not contain these keywords. Of note, "IOT" was one of our keywords, and while filtering, it was realized that a number of false positives were captured due to the word "bibliotheca"; these would be further filtered from our results.

## IV. Results

TABLE I
MEMORY PLUS IOT CVES COMPARED TO IOT ONLY CVES

| Year | IOT CVE | IOT and Memory CVE | Memory to Total |
|---|---|---|---|
| Year 2020 | 341 | 134 | 39.30% |
| Year 2021 | 295 | 112 | 37.97% |
| Year 2022 | 295 | 118 | 40.00% |
| Year 2023 | 65 | 13 | 20.00% |
| Year 2024 | 95 | 14 | 14.74% |
| Year 2025 | 54 | 7 | 12.96% |

Looking at the data in Table I, we can see a decrease in CVEs with a significant drop corresponding to 2023. We also see a significant reduction in the ratio of memory- and IOT-related CVEs to just IOT CVEs in 2023. We next took the PYPL index values for Rust and C/C++ for June of each year. We were looking at the data and graphed this (on a logarithmic scale) along with the percentage of memory and IOT CVEs to just IOT CVEs. Figure 1 shows that the popularity of searches for Rust tutorials has increased significantly between 2022 and 2023, and this corresponds to a decrease in the ratio of IOT CVEs related to memory. We also noted that the index value of C/C++ remained mostly flat during the period of this study. Our research has shown an inverse relationship between the number of CVEs related to IoT and Memory and the popularity of Rust programming, which would seem to indicate that our hypothesis was correct.

## V. Conclusion and Future Work

Our work here is more in line with a preliminary study of the viability of this kind of research and analysis. While it does seem to indicate that there is a relationship to examine more closely, we know that much more work would need to be done to achieve more impactful results using this method. One of our limiting factors was trying to calculate an adoption rate for Rust. When we initially started our research, we had hopes that CVE entries would include programming language used by the developers, thus allowing us to see the percentage of devices firmware coded with each language. This proved to be an incorrect assumption, so we had to find an alternative to way to understand the adoption rate. We settled on looking at the PyPL method, as its novel choice of measurement seemed like a good technique to understand the popularity of Rust's use. Future work would include more investigation into alternative methods of measuring Rust's adoption rate. Additionally, more time should be spent refining the keywords used in identifying the CVEs used in the calculations.

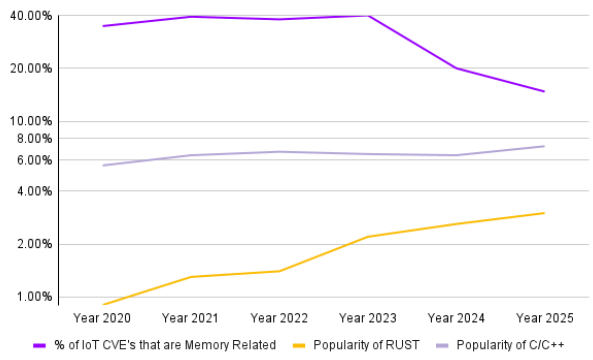


Fig. 1. Rust Adoption vs IOT Memory CVEs

[1]https://pypl.github.io/PYPL.html

[2]Colab notebooks are available via github repository: https://github.com/ds7389/CS_6233-Introduction_to_OS-Rearch_Project

## References


[1] S. Sinha, "State of iot 2024: Number of connected iot devices growing 13% to 18.8 billion globally," *IoT Analytics*, 2024.

[2] D. Kumar, K. Shen, B. Case, D. Garg, G. Alperovich, D. Kuznetsov, R. Gupta, and Z. Durumeric, "All things considered: An analysis of IoT devices on home networks," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 1169–1185. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/kumar-deepak

[3] T. Vandervelden, R. De Smet, D. Deac, K. Steenhaut, and A. Braeken, "Overview of embedded rust operating systems and frameworks," *Sensors*, vol. 24, no. 17, 2024. [Online]. Available: https://www.mdpi.com/1424-8220/24/17/5818

[4] K. R. Fulton, A. Chan, D. Votipka, M. Hicks, and M. L. Mazurek, "Benefits and drawbacks of adopting a secure programming language: Rust as a case study," in *Seventeenth Symposium on Usable Privacy and Security (SOUPS 2021)*. USENIX Association, Aug. 2021, pp. 597–616. [Online]. Available: https://www.usenix.org/conference/soups2021/presentation/fulton

[5] H. Xu, Z. Chen, M. Sun, Y. Zhou, and M. R. Lyu, "Memory-safety challenge considered solved? an in-depth study with all rust cves," *ACM Trans. Softw. Eng. Methodol.*, vol. 31, no. 1, Sep. 2021. [Online]. Available: https://doi.org/10.1145/3466642

[6] A. Sharma, S. Sharma, S. Torres-Arias, and A. Machiry, "Rust for embedded systems: Current state, challenges and open problems," *arXiv preprint arXiv:2311.05063*, 2023.

[7] PYPL, "PYPL PopularitY of Programming Language Index," https://pypl.github.io/PYPL.html, 2025, accessed: 2025-07-20. [Online]. Available: https://pypl.github.io/PYPL.html