IFP

The Great Refactor

How to secure critical open-source code against memory safety exploits by automating code hardening at scale | **Herbie Bradley & Girish Sastry**

The Great Refactor

How to secure critical open-source code against memory safety exploits by automating code hardening at scale | Herbie Bradley & Girish Sastry

This essay is part of The Launch Sequence, a collection of concrete, ambitious ideas to accelerate AI for science and security.

Summary

Critical systems across the country run on software riddled with vulnerabilities. AI now arms our adversaries with automated tools to find and weaponize these flaws. But it is also a transformative opportunity to strengthen our defenses.

A major source of vulnerabilities in our digital infrastructure stems from code written in languages like C and C++, which are vulnerable to exploits targeting computer memory. Estimates show that around 70% of vulnerabilities in these codebases stem from memory exploits. But AI could cheaply translate code from memory-unsafe to memory-safe languages, eliminating entire classes of vulnerabilities while modernizing legacy codebases.

We propose "The Great Refactor," an effort to systematically rewrite key open-source software libraries into the Rust programming language, which offers strong guarantees of memory safety. This project would target widely used, under-resourced libraries historically responsible for severe vulnerabilities.

The Great Refactor— structured as a Focused Research Organization (FRO) — should aim to secure 100 million lines of code before 2030, in a transformative effort to secure US critical infrastructure and our software supply chains. The US government should fund this effort with $100 million, which could save billions in cybersecurity costs. With AI advancing rapidly and Rust achieving industry maturity, the time has come for urgent action.

Motivation

Most software that we run is insecure, a fact frequently exploited by America’s adversaries.[1] But this insecurity is not an inevitable feature of code: we know how to make our digital world much more resilient. The main bottleneck is labor — translating code into safer languages, finding and fixing known vulnerabilities, and formally verifying that software meets desired properties all require skilled engineers to implement. Many software projects, especially open-source libraries depended on by thousands of American businesses, cannot afford this labor. Fundamentally, our security is limited by how much attention we can afford to give.

AI enters this picture as a force that can greatly reduce the cost of some types of labor. As a result, the cybersecurity landscape is rapidly evolving into an arms race where AI-powered offensive capabilities risk outpacing defensive measures. Nation-state actors and sophisticated criminal organizations are already experimenting with AI to help automate parts of the cyber kill chain.[2] As the cost of this new form of intelligence declines rapidly, we can shift the cybersecurity offense-defense balance towards the defender.[3]

But how can we best use this intelligence to make our world more secure? Memory safety vulnerabilities remain one of the dominant threats to our digital infrastructure.[4] These are bugs that occur when software accesses computer memory in unsafe ways, allowing attackers to inject malicious code or crash systems. Such vulnerabilities have underpinned some of the most devastating cyberattacks in recent history (including Slammer, WannaCry, and Heartbleed), with the economic damages of each measured in the billions of dollars.[5] Our

[1] The ODNI’s 2023 Annual Threat Assessment report notes that “Foreign cyber actors are increasingly bold in their malicious activity, targeting US public and private sector networks to steal data, disrupt operations, and potentially pre-position for future activity.”

[2] See, e.g. a 2024 report from Microsoft and OpenAI.

[3] A full analysis of the offense-defense balance changes from AI is beyond the scope of this piece, but we believe it could improve significantly if, before a software system is deployed, the marginal cost of removing vulnerabilities approaches 0.

[4] Some estimate that ~70% of critical CVEs (Common Vulnerabilities and Exposures, the standard identifiers for publicly known cybersecurity vulnerabilities) stem from memory corruption issues in C/C++ codebases.

[5] See, e.g., estimates of the impact of Slammer, WannaCry, and Heartbleed.

software supply chain is built upon libraries in memory-unsafe languages, often written decades ago, that provide no inherent protection against some of the most devastating attack vectors. Memory exploits are estimated to account for around 70% of all security vulnerabilities in C and C++ code.

Unlike many security flaws that require case-by-case patches, memory safety vulnerabilities offer the highest returns on investment because they can be proactively eliminated through translation to memory-safe languages.

Sample real-world memory safety vulnerabilities and their impacts

Library or Component	Maintainers at time of vulnerability	User base	Memory-safety vulnerability	Impact
OpenSSL	<5 core developers	≥ ½ million servers affected	Heartbleed (2014) – buffer over-read	Data/key leakage; cleanup & re-issuance cost estimated at ≥ $500 M
Dnsmasq	1 primary maintainer	Included in most Linux distros, home routers, Android and IoT firmware	Multiple buffer overflows in DNS and DHCP handling (e.g. heap overflow in DNS query reply, CVE-2017-14491) (2017)	Enabled remote code execution on affected devices (many routers and IoT devices)
BlueZ (Linux Bluetooth)	~1–2 developers	Default Bluetooth stack in Linux; used by PCs, smartphones, and IoT	Heap buffer overflow in L2CAP packet processing (Linux kernel CVE-2017-1000251) allowing remote code execution via Bluetooth (2017)	Complete device takeover via wireless Bluetooth (no pairing needed)
Exim MTA	~1-2 maintainers	Installed on ~57% of public email servers at the time of the 2019 disclosure, per Rapid7 scans	CVE-2019-10149 (and later) Heap overflow in string expansion logic	Remote root on mail servers; used in botnets and persistent backdoors

Patching these vulnerabilities requires maintainers to understand often-complex and subtle security bugs, and apply fixes in a timely manner. Moreover, there is a "long tail" of small libraries with vulnerabilities that are open-source passion projects. These often have a handful of maintainers who volunteer their time to develop these libraries — easy pickings for a determined adversary. For example, in 2021 the Log4j vulnerability was discovered: an attack allowing China, Iran, and ransomware outfits to compromise critical systems globally (including US state government networks) that stemmed from an open-source Java library maintained by just a handful of volunteers.[6]

Security engineers have long dreamed about the possibility of replacing insecure and brittle software with memory-safe code. Rust is the world's most used memory-safe programming language. As well as being highly performant, Rust enforces strict rules over the use of memory as soon as code is compiled, rather than when it is run, drastically reducing the potential for bugs.

Historically, the economics of rewriting code have been deeply unfavorable. Converting even a moderately sized C codebase into Rust would typically require several person-years of engineers expert in Rust — a small and expensive talent pool.[7] For comparison, the Linux kernel's partial Rust implementation required thousands of engineer-hours, with costs running into millions of dollars at standard Silicon Valley rates.

But AI could transform the economics of code translation. AI systems can now write idiomatic Rust that previously required years of experience. Translation quality has reached a tipping point where human review time, rather than translation time, becomes the primary bottleneck. And AI's programming capability is growing fast, with AI coding agents rapidly improving in their ability to autonomously fix issues and implement features.[8] This is a trend we can bet on. Meanwhile, the Rust ecosystem has matured enough to support real-world production deployments at major organizations like Microsoft, Google, and

[6] We note that Log4j was not specifically a memory-safety exploit, but the state of the library's maintenance is representative of many memory unsafe codebases.

[7] According to BairesDev, "the talent pool for C++ programmers is a thousand times bigger than Rust, at least for now."

[8] At some big tech companies like Google, more than a quarter of all new code is written by AI.

Amazon, with tangible security and maintainability benefits.[9] Securing 100 million lines of critical code could lead to saving billions of dollars.[10]

However, market forces alone cannot solve this problem. The history of cybersecurity in the Internet age makes that clear. Critical libraries that serve as foundational dependencies have minimal security resources despite their outsized impact on the digital ecosystem. While individual companies have incentives to secure their own codebases, the distributed nature of open-source maintenance creates a coordination problem preventing adoption of systematic solutions, leaving potentially transformative security gains on the table.

Solution

The Great Refactor is our proposal to systematically identify and refactor key open-source software libraries into Rust using AI. If successful, this project could significantly reduce a key class of vulnerabilities that remain endemic to our software supply chains. This project also provides a golden opportunity to significantly improve the efficiency and maintainability of our software supply chain. Where older translation tools would attempt the most direct translation possible, we can use AI to simultaneously translate and refactor our code into better architectures, where appropriate.

We think the best vehicle for this ambitious goal is a **Focused Research Organization (FRO):**[11] a purpose-built, time-limited non-profit organization designed to tackle a clear technical objective that requires coordination across disciplines. Unlike traditional academic labs or startups, a FRO can marshal long-term funding and top-tier engineering talent while staying mission-focused

[9] Android successfully cut the percentage of memory safety vulnerabilities from 76% to 24% over 6 years.

[10] $2B in savings assuming a cost per exploited incident of $4.8M on average, a cost to fix a vulnerability at $3000, that ~0.5% of vulnerabilities are exploited, and 70% rate of vulnerabilities stemming from memory-safety. See appendix for detailed costs & savings.

[11] We examined other org design options, including a further DARPA project, a new team within government, an FFRDC housed within a nonprofit, etc. We settled on an FRO because we believe independence (to more easily attract top talent), the ability to draw on private sector funding, and flexibility of work direction (given the rapidly shifting capabilities of AI) are crucial to success.

and driven by the public interest.[12] A successful effort will need to integrate expertise across security engineering and AI to resolve the engineering and logistical hurdles. The FRO model is uniquely suited to deliver on this kind of complex, interdisciplinary project.

Existing efforts in the Rust translation space are part of the puzzle. DARPA's TRACTOR program funds translation research projects starting from the C language and encompassing both AI tools and older methods, aiming to develop novel techniques for more effective translation. Meanwhile, large tech companies like Microsoft are using tools internally, including AI, to translate key code into Rust.

This proposal would complement these efforts. Its mission: to apply the state of the art in Rust AI translation to critical open-source libraries (particularly in C and C++), and ensure their widespread adoption. Success requires more than just working translation tools — the FRO should do the work to ensure that the newly-translated Rust libraries have the support they need to flourish, rather than simply being dropped from above. This will involve engaging with maintainers of existing libraries, building AI tools and documentation to help migration, and putting careful thought into the structure of translated code to make new libraries a delight to maintain.

A further benefit is in building infrastructure for **formal verification** efforts. Formal verification involves creating a mathematical model of a program and its intended behavior, then using logical proofs to demonstrate that the implementation matches the specification, which has many security benefits. Rust is an ideal intermediate step,[13] so this initiative can help significantly improve our software security in the long run.

Program roadmap

The FRO should be tasked with applying the state-of-the-art AI models and Rust translation tools to key open-source libraries, and ensuring their widespread

[12] Successful examples include FutureHouse and the Lean FRO.

[13] This is because the Rust compiler mathematically enforces strong guarantees of memory safety when the program is compiled.

adoption. We anticipate that due to the rapidly improving software engineering ability of AI, this will require surprisingly little fundamental R&D effort.[14] Most of the technical work is in deeply understanding the target codebase for translation, using AI tools to design the new Rust libraries, and in verifying that the new code has the correct functionality.

A $100 million investment would empower the FRO to hire a team of tens of security engineers, AI researchers, and administrators.[15] Over a 3–5 year timeline, the FRO would:

- **Identify the most important libraries to secure.** The FRO should develop a systematic approach to finding high-value targets, combining metrics like library popularity, maintenance status, security importance, and translation complexity. Ideal candidates share several common properties: they are a key dependency for widely-used software, have minimal security or maintenance resources, and possess sufficient test coverage to validate translation correctness. Promising domains for initial work include parsing libraries, network protocol implementations, and system-level utilities—areas historically responsible for some of the most severe vulnerabilities.
- **Ensure robust validation of translated code.** Translated code needs to be correct in function, free of bugs, and idiomatic & maintainable Rust. The FRO should develop a robust validation engine that goes beyond unit tests and AI-assisted fuzzing to automatically assess the output code for correct functional equivalence with the original library. Security and Rust experts should be employed to review the output as a final validation layer—assisted by AI tooling.
- **Build a basis for defense-dominant cybersecurity.** Beyond simply translating code, the Great Refactor should aim to create more maintainable, well-documented Rust code. This could be the launching point for future security efforts, including formally verifying the properties of code, building

[14] Even in cases where current AI systems struggle, we think it makes little sense to throw large amounts of resources into building complex scaffolding for specialized translation tasks, when trends should lead us to expect general AI capabilities to improve and increasingly solve some parts of this problem by default.

[15] A rough estimate that assumes ~70% of the costs going towards expert staff and the remaining 30% to services and administrative expenses.

toward even stronger security guarantees. AI systems can handle the drudgery of documentation and leverage their vast knowledge of programming idioms to create highly maintainable code.

- **Develop a repeatable playbook for adoption.** Close engagement with the maintainers and major users of existing open-source libraries is key to ensure smooth migration. The FRO should aim to ensure the widest possible adoption of translated libraries. This should include developing AI tools and documentation to ease transition for existing developers, and advocating the use of the Rust libraries to key stakeholders like Linux distribution maintainers.

Success metrics would aim to track the long-term sustainability of translated libraries. This means not just the number of completed translations, but also adoption rates, security impact measurements, and ecosystem health indicators.

Recommended government actions

The US Government should:

- **Establish dedicated FRO funding.** DARPA, NSF, or a new interagency initiative should allocate $50–75 million over 5 years specifically for The Great Refactor FRO. This funding could be tied to specific milestones, like successfully hardening critical codebases depended on by government software.
- **Create an oversight and advisory structure.** Assemble a world-class technical advisory board with representatives from the NSA, CISA, the DoD, and industry to provide strategic guidance and ensure that the FRO's work targets national security priorities.
- **Integrate with existing procurement and compliance frameworks.** Work with GSA and other agencies to update software procurement guidelines to favor memory-safe alternatives, creating market incentives for adoption of translated libraries.
- **Leverage industry co-investment.** Partner with leading American tech companies like Google, Microsoft, and Amazon — who have already invested heavily in Rust adoption — to provide both funding and technical expertise.

Industry partners could contribute approximately 30–40% of total funding while gaining early access to secure libraries critical to their infrastructure.

By accelerating this effort, we can use AI to break the cyber arms race and fundamentally shift the economics of cybersecurity from perpetual patching toward systemic prevention, creating more resilient digital infrastructure for everyone.

Appendix

Risks and limitations

The capability of AI in software engineering is improving rapidly, so we believe this ambitious proposal to be technically very promising. However, we'd be remiss not to mention some important potential limitations:

- **Eliminating memory-safety shifts attacks to other vulnerabilities.** While memory-safety vulnerabilities are a large share of common exploits, many attacks in practice often leverage other methods, particularly social engineering. Securing against all possible attack vectors is beyond the scope of this project; as the attack surface area from memory unsafety is reduced, we expect the focus of attackers to shift elsewhere.

- **Navigating the AI reliability frontier.** AI is advancing rapidly at software engineering tasks, but there are still questions about its reliability, especially at potentially sensitive tasks like writing secure code. Even if a codebase has extensive automated tests, they may not be sufficient to have confidence in the equivalence of a full Rust translation. However, AI research (i.e., the discriminator-generator gap)[16] suggests that it's possible to automatically verify that produced code does not introduce new vulnerabilities and matches the original codebase in function. By developing rigorous testing protocols, relying on expert human oversight at key development milestones, and validating

[16] The discriminator-generator gap is the observed difference in performance between a model's ability to generate high-quality outputs (generator) and its ability to discriminate, critique, or effectively evaluate those outputs (discriminator).

translations incrementally, we can mitigate these risks while capitalizing on AI's rapid improvements in code generation and analysis.

- **Software deployment and update adoption remains a bottleneck.** Even the most secure code provides limited protection if organizations fail to deploy updates promptly. This proposal addresses the creation of memory-safe alternatives but doesn't directly solve the constant effort to ensure widespread adoption of newer, more secure versions. Critical infrastructure often runs on outdated software due to compatibility concerns, organizational inertia, or fear of disrupting stable systems. Incentives to encourage timely updating could help increase the odds that more-secure dependencies are actually used.
- **Adoption may be hindered by lack of widespread Rust expertise.** Rust translations may be harder to adopt if the maintainers of existing libraries do not have sufficient Rust expertise, which is often a current bottleneck for Rust translation efforts. This can be mitigated with a coordinated effort to engage with library maintainers early in the process and provide comprehensive migration support, including AI-powered developer tools. While some resistance exists due to programmer preferences for familiar languages, the growing momentum in the Rust ecosystem and clear security benefits of memory-safe alternatives suggest this cultural barrier will diminish as adoption increases.[17]

Funding requirements

We estimate that this FRO will need roughly **$100 million over 5 years** to achieve its ambitions, broken down as follows:

Cost Bucket	Salary/Assumptions	Annual $M	Total $M (5Y)
Personnel			
5x AI engineers	$350k–$500k	2.125	

[17] Rust's rules about how data is handled force programmers to organize their code in ways that veteran C programmers may find restrictive. Unlike C, which lets programmers freely manipulate memory however they want, Rust requires clearer boundaries between different parts of a program. However, these same restrictions that cause initial frustration are what make Rust programs more reliable, easier to update, better at handling multiple tasks simultaneously, and much less prone to security flaws.

15x Rust systems engineers to validate translations	$200k	3	
5x Security engineers	$220k	1.1	
3x Formal verification experts	$220k	0.66	
5x DevRel to smooth adoption	$200k	1	
5x DevOps/infra	$220k	1.1	
5x Exec + admin	$200k	1	
1.4x multiplier for benefits, taxes		13.98	69.9
AI services	$1m/yr in token costs	1	5
Grants to OSS maintainers	Stipends + integration support for a subset of projects	2	10
Third-party red-teaming & bug-bounties		1	5
General & Administrative	Rent, insurance, legal, outreach, board, travel	1.65	8.25
Total		**19.63**	**98.15**

Funding sources should include a mix of public and private funding. US Government funding could be tied to specific objectives that are particularly desirable for the public interest — for example, specifically to carry out Rust translation work on open-source codebases relevant for government use, or to translate internal government legacy code.

This project can produce a high ROI; the cost of memory-safety incidents is significant, and we estimate savings of ~$2B from securing 100 million lines of code. This assumes a cost per exploited incident of $4.8M on average, a cost to fix a vulnerability at $3000, that ~0.5% of vulnerabilities are exploited, and 70% rate of vulnerabilities stemming from memory-safety.

Further resources

- MemorySafety.org, Memory safety issues, n.d.

  Memory safety issues remain one of the most prevalent and dangerous security vulnerabilities in modern software. See also this explainer by the NSA and CISA.

- DARPA, Translating All C To Rust (TRACTOR) program, n.d.

  The DARPA Translating All C To Rust (TRACTOR) program aims to research tools and techniques that can automatically translate C code to memory-safe Rust while preserving functionality, addressing one of the root causes of software vulnerabilities in critical systems used throughout government and industry.

- Convergent Research, Focused Research Organizations (FROs), n.d.

  Focused Research Organizations (FROs) represent a new model for conducting scientific research that fills the gap between academia and industry, tackling important technical problems that don't fit neatly into existing research structures by bringing together dedicated teams with focused missions and specific deliverables.