How to Talk to Your Manager About Memory Safety

Managing computer memory securely is critical for ensuring that software functions as intended and for avoiding memory vulnerabilities that could be exploited to disrupt, intercept, or take over computer systems. However, it is also a commonly neglected aspect of enterprise security.

Memory safety vulnerabilities are everywhere, and they're dangerous. They make up between 60 and 90 percent of all software flaws across software written in memory-unsafe languages.[1] They also represent a very high fraction of zero-day vulnerabilities.[2]

To increase memory safety in the software your organization is using, wherever possible:

- use software that has been developed in a memory-safe language.
- ask developers to build your software in a memory-safe language.
- increase internal expertise in memory-safe coding.
- choose software vendors that prioritize memory safe language adoption and use.

What Is Memory Safety?

Memory safety is a term used to describe whether software or a programming language is designed to prevent memory bugs and vulnerabilities. Typically, software or a programming language is described as memory-safe or not.

[1] "What is memory safety and why does it matter?" *Prossimo,* https://www.memorysafety.org/docs/memory-safety/; "Queue the Hardening Enhancements," *Google Security Blog,* May 9, 2019, https://security.googleblog.com/2019/05/queue-hardening-enhancements.html.
[2] Examples: https://twitter.com/LazyFishBarrel/status/1129000965741404160 and https://source.android.com/docs/security/test/memory-safety.

Memory-safe languages incorporate *automatic* memory management, which means that they only allow for safe reads and writes, and they safely free memory when it is no longer needed. A memory-unsafe language does not do this; engineers must manually manage memory allocations, a slow and painstaking process that is rarely done entirely correctly, even by highly experienced programmers. Even with help from automation, it's basically impossible to identify all vulnerabilities and prevent false positives. Attempting to fix these vulnerabilities also adds the risk of inadvertently introducing further vulnerabilities.

Memory bugs and vulnerabilities arise when a program mismanages memory, including access,[3] allocation,[4] and buffer control.[5] This can lead to poor performance and application failure, and it makes software much more vulnerable to exploits.

Memory safety bugs and vulnerabilities include:

- Buffer overflow: overwriting of memory locations adjacent to a buffer boundary. Data provided by an end user may overwrite program data, and even lead to unauthorized code execution. This may allow users to read other people's data and/or take control of the machine running the unsafe code.[6] For example, the Heartbleed vulnerability in OpenSSL allowed attackers to access encrypted personal and sensitive data across a wide range of websites.[7]
- Use after free: The pointer continues to point to memory that has been deallocated. This may allow a user to read data that has been deleted.

What Are Some Common Memory-Safe Languages?

Rust, C#, Go, Java, Ruby, Python, JavaScript, Swift.

What Are Some Common Memory-*Unsafe* Languages?

C, C++, Assembly.

[3] **Memory access:** How a program reads and writes to memory.
[4] **Memory allocation:** This is the process of reserving sections of memory in a program to be used for specific purposes.
[5] **Buffer control:** This refers to how a program manages how memory is temporarily stored in the buffer before it is moved between one place and another, e.g., between data processors, or input and output devices, or being sent over the network. The buffer helps smooth out variations in the data.
[6] Fernando Diaz, "How to secure memory-safe vs. manually managed languages," *GitLab,* March 14, 2023,
https://about.gitlab.com/blog/2023/03/14/memory-safe-vs-unsafe/.
[7] "OpenSSL 'Heartbleed' vulnerability (CVE-2014-0160)," *Cybersecurity & Infrastructure Security Agency,* last revised October 5, 2016,
https://www.cisa.gov/news-events/alerts/2014/04/08/openssl-heartbleed-vulnerability-cve-2014-0160.

Why Does It Matter?

Protects against intrusion and data breaches: Memory safety bugs and vulnerabilities are exploited by attackers to access enterprise data, deploy ransomware, disrupt or deny service, and otherwise harm an organization's systems, customers, and business partners. These vulnerabilities could be avoided by switching to memory-safe code. The average cost of data breaches to enterprises is now above $4 million (USD) per incident.[8]

Demonstrates that your organization cares about security-by-design: Writing and using software in memory-safe code is a practical example of security-by-design.

Potential advantage in government contracts: Even if governments do not require the software they use to be developed using memory-safe code today, they may in the future, and it could be one of the factors they take into consideration when choosing a commercial provider.

How Do You Figure Out Where You're Exposed and How to Fix It?

The first step is to know exactly what software your organization is using, as well as understanding the whole supply chain for that software. Modern software often pulls in third-party code, such as software libraries from multiple sources. Then, the best next step is to figure out where the biggest source of risk is. That might be the component with the most vulnerabilities reported the prior year, or network and privilege boundaries. You might need to reach out to external security experts and check the results of public code reviews. Then see what the blockers are to memory-safe solutions and start knocking the barriers down.

Do You Have to Do Everything All at Once?

In a word, no. A good way to get started is by focusing on your most critical libraries and packages and directly exposed attack surface. In addition to rewriting some code in a memory-safe language, you may also want to sandbox, that is, software-isolate unsafe code that can't be easily rewritten.

What Is a Simple Step to Get Started?

One way to get started is to add a new memory-safe component to an existing C/C++ project.

Some examples of this approach include Rust components shipped in Firefox 56 and 57, AWS building critical services in Rust, Chrome adding limited support for Rust components, Windows adding Rust components to the kernel, the Linux kernel adding support for Rust, and the majority of Android 13's new code being memory-safe.

[8] "Cost of a Data Breach Report 2023," *IBM,* https://www.ibm.com/reports/data-breach.

You can also work on incremental migrations or migrations of individual components, such as replacing a single Apache module using Rustls, which you can do without learning how to code in Rust.

You can also commit to making sure all new code is memory-safe.

Why Is This Worth the Cost?

Memory safety is an up-front investment that will reduce your long-term support costs. Having fewer vulnerabilities will reduce an expensive triage process, and you'll have fewer stability problems and nonsecurity crashes as well as performance improvements due to concurrency.

Where Can I Find Existing Open Source Memory-Safe Code?

Google Rust open source
Mozilla open source
ISRG Prossimo
Rustls

Audits of Rust Open Source Crates

Google and Mozilla publish their audits of their open source Rust crates on GitHub at https://github.com/google/rust-crate-audits and https://github.com/mozilla/supply-chain. You can use these audits to decide whether the "crates meet the security, correctness, and testing requirements for your projects."[9]

What Can We Do While We Are Making the Shift to Memory-Safe Code?

- Code review, code review, code review.
- Use some modern C++ idioms that can help produce more safe and reliable code.
- Use fuzzers and sanitizers to help find bugs before they make it into production.
- Use exploit mitigations to help increase the difficulty of exploiting vulnerabilities (such as addressing space layout randomization, orASLR, to mitigate buffer overflow attacks).
- Use privilege separation so that even when a vulnerability is exploited, the attacker has limited access.[10]

But remember, though these steps may help, they do not result in memory safe code or memory safety.

[9] "Open sourcing our Rust crate audits," *Google Open Source Blog,* May 23, 2023, https://opensource.googleblog.com/2023/05/open-sourcing-our-rust-crate-audits.html.
[10] "What is memory safety and why does it matter?" *Prossimo,* https://www.memorysafety.org/docs/memory-safety/.

What Are Other Ways Our Organization Can Support Memory Safety?

1. Join other organizations in funding open source initiatives, such as Internet Security Research Group's Prossimo.[11]
2. Share your memory-safe coding expertise with open source projects. Give your software developers time to volunteer their skills and effort.
3. Inform your customers when your products use memory-safe code.
4. Ask your software suppliers to use memory-safe code in their products.

[11] *Prossimo,* https://www.memorysafety.org/.

APPENDIX

Government Guidance

National Security Agency (November 2022):[12]

- Make a strategic shift to memory safety.
  "NSA advises organizations to consider making a strategic shift from programming languages that provide little or no inherent memory protection, such as C/C++, to a memory safe language when possible."
- Use available code hardening defenses.
  "Memory safe languages provide differing degrees of memory usage protections, so available code hardening defenses, such as compiler options, tool analysis, and operating system configurations, should be used for their protections as well."

Cybersecurity and Infrastructure Security Agency (CISA) (April 2023) in collaboration with the NSA, the FBI, Australian Cyber Security Centre (ACSC), Canadian Centre for Cyber Security (CCCS), the United Kingdom's National Cyber Security Centre (NCSC-UK), Germany's Federal Office for Information Security (BSI), the Netherlands' National Cyber Security Centre (NCSC-NL), Computer Emergency Response Team New Zealand (CERT NZ), and New Zealand's National Cyber Security Centre (NCSC-NZ):[13]

- Prioritize the use of memory safe languages wherever it is possible.
- Address space layout randomization (ASLR), control-flow integrity (CFI), and fuzzing can be helpful for legacy codebases.

[12] "Cybersecurity Information Sheet Software Memory Safety U/OO/219936-22 | PP-22-1723 | NOV 2022 Ver. 1.0, *National Security Agency,* https://media.defense.gov/2022/Nov/10/2003112742/-1/-1/0/csi_software_memory_safety_pdf (PDF).
[13] "Shifting the Balance of Cybersecurity Risk: Principles and Approaches for Security-by- Design and -Default," CISA | NSA | FBI | ACSC | NCSC-UK | CCCS | BSI | NCSC-NL | CERT NZ | NCSC-NZ, https://www.cisa.gov/sites/default/files/2023-04/principles_approaches_for_security-by-design-default_508_0.pdf (PDF).