Secure by Design Alert: Eliminating Buffer Overflow Vulnerabilities

Malicious Cyber Actors Use Buffer Overflow Vulnerabilities to Compromise Software

Note: This Secure by Design Alert is part of an ongoing series aimed at advancing industry-wide best practices to eliminate entire classes of vulnerabilities during the design and development phases of the product lifecycle. The Secure by Design initiative seeks to foster a cultural shift across the technology industry, normalizing the development of products that are secure to use out of the box. Visit cisa.gov to learn more about the principles of Secure by Design, take the Secure by Design Pledge, and stay informed on the latest Secure by Design Alerts.

The software development community has twenty years of extensive knowledge and effective solutions for buffer overflows—however, many software manufacturers continue to expose customers to products with these vulnerabilities.

Buffer overflow vulnerabilities are a prevalent type of memory safety software design defect that regularly lead to system compromise. The Cybersecurity and Infrastructure Security Agency (CISA) and Federal Bureau of Investigation (FBI) recognize that memory safety vulnerabilities encompass a wide range of issues—many of which require significant time and effort to properly resolve. While all types of memory safety vulnerabilities can be prevented by using memory safe languages during development, other mitigations may only address certain types of memory safety vulnerabilities. Regardless, buffer overflow vulnerabilities are a well understood subset of memory safety vulnerability and can be addressed by using memory safe languages and other proven techniques listed in this Alert. Despite the existence of well-documented, effective mitigations for buffer overflow vulnerabilities, many manufacturers continue to use unsafe software development practices that allow these vulnerabilities to persist. For these reasons—as well as the damage exploitation of these defects can cause—CISA, FBI, and others[1] designate buffer overflow vulnerabilities as unforgivable defects.

CISA and FBI maintain that the use of unsafe software development practices that allow the persistence of buffer overflow vulnerabilities—especially the use of memory-unsafe programming languages—poses unacceptable risk to our national and economic security.[2] As such, CISA and FBI urge manufacturers to use proven prevention methods and mitigations to eliminate this class of defect while urging software customers to demand secure products from manufacturers that include these preventions. This Alert outlines proven methods to prevent or mitigate buffer overflow vulnerabilities based on secure by design principles and software development best practices.

Buffer overflow vulnerabilities (CWE-119) arise when threat actors access or write information in the wrong part of a computer's memory (i.e., outside the memory buffer). These vulnerabilities can occur in two main memory regions in which buffers are managed: **stack-based overflows** (CWE-121) (allocated on a memory stack), and **heap-based overflows** (CWE-122) (allocated on a memory heap). Buffer overflow vulnerabilities pose serious security risks, as they may lead to data corruption, sensitive data exposure, program crashes, and unauthorized code execution.[3] Threat actors frequently exploit these vulnerabilities to gain initial access to an organization's network and then move laterally to the wider network.[3] Examples of buffer overflow vulnerabilities include:

- CVE-2025-21333
- CVE-2025-0282
- CVE-2024-49138
- CVE-2024-38812

- CVE-2023-6549
- CVE-2022-0185

The authoring agencies urge manufacturers to take immediate action to prevent these vulnerabilities from being introduced into their products. Manufacturers can prevent buffer overflow vulnerabilities by using the secure by design practices listed in this Alert. Software manufacturer senior executives and business leaders should ask their product and development teams to document past buffer overflow vulnerabilities and how they are working to eliminate this class of defect.

CISA and FBI also recommend that software customers ensure manufacturers demonstrate adherence to the safe software development practices contained in this Alert by requesting that manufacturers provide a Software Bill of Materials (SBOM) and a secure software development attestation.[4]

To prevent buffer overflow vulnerabilities, technical leaders should implement the following secure by design practices. (**Note:** This is not a comprehensive list of methodologies for mitigating and preventing buffer overflow vulnerabilities but contains those CISA identified as the most effective and feasible.)

- Where feasible, use **memory-safe languages** when developing software to shift the burden of memory management from the developer to the programming language's built-in safety features.[5]
  Note: It is possible to disable or override the memory safety guarantees of some memory-safe languages; developers must avoid doing so to prevent buffer overflow vulnerabilities. Also, using a memory-safe language for one part of a software package does not fix memory-unsafe code in other libraries.
  - **The authoring agencies recognize there is significant effort required to rewrite codebases in memory-safe languages.** Therefore, we recommend manufacturers develop and implement a phased transition plan for increasing memory-safe language usage. Ideally, this plan should include using memory-safe languages to develop new code and—over time and when feasible—transition their software's most highly privileged/exposed code to memory-safe languages.[6] While transitioning to memory-safe languages, manufacturers should also consider leveraging technologies to limit memory safety vulnerabilities in their existing code bases.
- **Enable compiler flags that implement compile time and runtime protections against buffer overflows**[7] (to the maximum extent possible given the application's performance requirements) and implement canaries that alert if an overflow occurs.[8]
- Regularly run **unit tests** with an **instrumented toolchain** such as AddressSanitizer and MemorySanitizer that exercises source code with runtime checks for buffer overflows and other memory safety issues. In a codebase with significant unit test coverage, such tools can detect many (but not all) memory safety related issues before they become a vulnerability.
- Conduct aggressive adversarial **product testing**, including static analysis, fuzzing, and manual reviews (as needed) to ensure the quality and security of code throughout the development lifecycle.[9]
- **Publish a memory-safety roadmap** outlining how the manufacturer plans to develop new products with memory-safe languages and migrate code to memory safe languages in a prioritized manner.
- **Conduct root cause analysis of past vulnerabilities, including buffer overflows**, to spot trends and patterns. Where possible, take actions to eliminate entire classes of vulnerabilities across products, rather than the superficial causes.

Secure by Design Lessons to Learn

Products that are secure by design reasonably protect against malicious cyber actors exploiting the most common and dangerous classes of product defect. Incorporating security at the outset of the software development lifecycle (beginning in the design phase and continuing through development, release, and updates) reduces the burden on customers and risk to the public. Despite this finding, buffer overflow vulnerabilities remain a prevalent class of defect.

Where feasible, manufacturers should work to eliminate buffer overflow vulnerabilities by developing new software using memory-safe languages and the best practices described in this Alert. Furthermore, eliminating buffer overflow vulnerabilities can help reduce the prevalence of other memory safety issues, such as format string, off-by-one, and use-after-free vulnerabilities. For more information on memory safety and transitioning to using memory-safe languages, review the following resources:

- CISA and FBI's The Case for Memory Safe Roadmaps, Exploring Memory Safety in Critical Open Source Projects, and Product Security Bad Practices.
- White House Office of the National Cyber Director's (ONCD) Back to the Building Blocks report.
- National Security Agency's (NSA) Software Memory Safety.
- Google's Perspective on Memory Safety and Eliminating Memory Safety Vulnerabilities at the Source, Microsoft's We Need a Safer Systems Programming Language, Amazon Web Service's (AWS) Sustainability with Rust, and Mozilla's Implications of Rewriting a Browser Component in Rust.

Secure by Design Principles to Follow

Manufacturers should consider pledging to follow the three Secure by Design principles, jointly developed by 17 global cybersecurity agencies, outlined below.

Principle 1: Take Ownership of Customer Security Outcomes

Software manufacturers must prioritize customer security by eliminating buffer overflow vulnerabilities. Key investments include providing secure building blocks for developers to prevent errors that could compromise reliability and user data. Relying on post-detection fixes is unsustainable; instead, manufacturers should implement best practices during the software development lifecycle. Automated safeguards should prevent unsafe functions, while static analysis tools and rigorous code reviews can help identify flaws before deployment. By addressing vulnerabilities at the source, manufacturers enhance security without relying on customers for fixes.

Principle 2: Embrace Radical Transparency and Accountability

Manufacturers must prioritize transparency in disclosing product vulnerabilities. They should track vulnerabilities in products, including for software-as-a-service (SaaS) products, and report them through the Common Vulnerabilities and Exposures (CVE) program. CVE records must be complete, accurate, and timely, and be complemented by an appropriate Common Weakness Enumeration (CWE) to facilitate tracking of software defect classes.

The software development community has outlined methods to avoid various buffer overflow vulnerabilities, such as those classified under CWE-119 and related CWEs. Manufacturers should identify the root causes of these vulnerabilities and aim to eliminate them and address broader memory safety issues. Additionally, they should maintain a robust vulnerability disclosure program (VDP) and a product security incident response team (PSIRT). CISA offers resources to help organizations establish and maintain a VDP.

Principle 3: Build Organizational Structure and Leadership to Achieve These Goals

Technology executives must treat software security resources as strategic investments, recognizing that secure by design practices deliver significant long-term savings. For example, Google's Android team transitioned to memory-safe programming languages for new code in 2019 after determining that the long-term costs of mitigating memory safety vulnerabilities would exceed the investment required to prevent them.[10] By addressing these vulnerabilities at their root, they ultimately reduced costs over time. Similarly, organizations such as Microsoft, AWS, and Mozilla have advocated for the adoption of memory-safe languages to enhance software security.[11]

Beyond financial considerations, leaders must recognize the broader implications of security on customers, the economy, and national security. This includes investing in initiatives and incentives that embed security as a core business priority. Organizations should proactively work to eliminate entire classes of vulnerabilities and perform regular reviews to identify and address recurring issues. Executives must actively monitor progress, ensure adequate resources are allocated, and commit to continuous improvement.

Action Items for Software Manufacturers and Customers

Software manufacturers should consider taking the Secure by Design Pledge to demonstrate their commitment to building their products to be secure by design. The pledge lays out seven key goals that signers commit to demonstrate measurable progress in making their products secure by design, including reducing systemic classes of vulnerability like buffer overflows.

Customers should demand that software is secure by design. Organizations looking to acquire software that is secure by design should refer to our Secure by Demand guidance and incorporate the following product security considerations into their procurement lifecycle:

- **Before procurement:** Ask questions to understand how each software manufacturer ensures product security.
- **During procurement:** Integrate the organization's product security requirements into contract language.
- **Following procurement:** Continually assess product security and security outcomes.

Disclaimer

The information in this report is being provided "as is" for informational purposes only. The authoring organizations do not endorse any commercial entity, product, company, or service, including any entities, products, or services linked within this document. Any reference to specific commercial entities, products, processes, or services by service mark, trademark, manufacturer, or otherwise, does not constitute or imply endorsement, recommendation, or favoring by the authoring organizations.

[1] See MITRE's Unforgivable Vulnerabilities and CWE Top 25 Most Dangerous Software Weaknesses.
[2] See CISA's Product Security Bad Practice Guidance.
[3] See OWASP Foundation's Buffer Overflow and Fortinet's What Is Buffer Overflow? Attacks, Types & Vulnerabilities.
[4] See NIST's Special Publication (SP) 800-218, Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities for manufacturer and vendor responsibilities for attesting to adhering to secure software development practices.
[5] See OWASP Foundation's Buffer Overflow.
[6] See CISA's Product Security Bad Practices guidance and Google's case study Eliminating Memory Safety Vulnerabilities at the Source for how Google's Android team greatly reduced the prevalence of memory safety vulnerabilities in Android by switching to using a memory-safe language.
[7] See examples for compiler flags that prevent buffer overflows listed in Red Flag Developer's Use Source-Level Annotations to Help GCC Detect Buffer Overflows and Use Compiler Flags for Stack Protection in GCC and Clang. Note that these flags are not necessarily implemented in all C compilers.
[8] See OWASP Foundation's CWE-119.
[9] See NIST's SSDF.
[10] See Google's Eliminating Memory Safety Vulnerabilities at the Source.
[11] See Microsoft's We Need a Safer Systems Programming Language, Mozilla's Implications of Rewriting a Browser Component in Rust, and AWS's Sustainability with Rust.