The Open Source Security Foundation (OpenSSF) seeks to make it easier to sustainably secure the development, maintenance, and consumption of the open source software (OSS) we all depend on. This includes fostering collaboration, establishing best practices, and developing innovative solutions.
The C and C++ Hardening Challenge

Addressing vulnerabilities in C and C++ on a large scale presents several significant challenges:

- Rewriting all existing C and C++ code to memory-safe languages is unbearably expensive
- Unsafe dependencies will slow down migration to memory-safe languages, such as Rust*

*) Recent data indicates that over 70% of Rust crates have dependencies on C or C++
Recent regulatory attention

- US Presidential Executive Order on Improving the Nation’s Cybersecurity
  - May 2021

- EU Cyber Resilience Act (Proposal)
  - September 2022

- NSA Guidance on Software Memory-safety Issues
  - November 2022

- CISA joint publication with 10 national cybersecurity authorities on Shifting the Balance of Cybersecurity Risk: Security-by-Design and -Default
  - December 2022

- Updated CISA publication with 18 national cybersecurity authorities: Principles and Approaches for Secure by Design
  - April 2023

- CISA joint publication with 8 national cybersecurity authorities on Memory Safe Roadmaps
  - October 2023

- Political agreement on EU Cyber Resilience Act
  - December 2023
Guide in *configuring programming tools* during development to *reduce attack surface of produced software*.

**C.f. Product Hardening**
Provides guidance in configuring a product’s operational parameters to secure defaults to reduce attack surface of deployed software.

**C and C++ Compilers**
Provide optional features that must be enabled to add protection against various security flaws to compiled binaries, both applications and shared libraries.

**Major Linux distributions**
Already package software with such protections enabled by default.

**Consuming OSS**
From source means you are responsible for ensuring that these protection features are enabled when building the software.
Challenges for deploying hardened compiler options

Possible deployment pitfalls

- Default enabled features depend on compiler, compiler version and where it is sourced from
- OSS projects that do not enable or support protection options in their build system or code
- Protection features that require tradeoffs in performance, memory, or increased binary size
- Protection features that are incompatible with certain language constructs or patterns

“[...] 85.3% of desktop binaries adopt Stack Canaries, but only 29.7% of embedded binaries do”

Building Embedded Systems Like It’s 1996

Ruotong Yu1 , Francesca Del Nin1 , Yiachen Zhang1 , Shan Huang1 , Pallavi Kaliyar3 , Sarah Zako5
1Sterns Institute of Technology 2University of Padua 3Norwegian University of Science and Technology
4Cyber Independent Testing Lab 5University of Utah 6Delft University of Technology

Abstract—Embedded devices are ubiquitous. However, preliminary evidence shows that attack mitigations protecting our desktop/server/phones are missing in embedded devices, posing a significant threat to embedded security. To this end, this paper presents an in-depth study on the adoption of common attack mitigations on embedded devices. Precisely, it measures the presence of standard mitigations against memory corruptions in over 10k Linux-based firmware of deployed embedded devices.

Network and Distributed Systems Security (NDSS) Symposium 2022

Compiler options hardening is not a silver bullet, but necessary in combination with memory-safe languages, secure coding standards, and security testing
What is covered by the guide?

1. **Recommneded Compiler Options**
   - Hardening options widely available in open-source compilers, currently GCC and Clang/LLVM
   - Includes both flags that will warn developers about flaws, as well as harden software
   - Most of these options are already enabled by the major Linux distributions today

2. **Discouraged Compiler Options**
   - Compiler options that, when used inappropriately, may result in potential defects with significant security implications in produced binaries.

3. **Sanitizers**
   - Compiler-based tools designed to detect and pinpoint memory-safety issues and other defects
   - Valuable diagnostics for debugging and testing
   - May be prohibitively expensive for release builds due to performance penalties & memory overhead

4. **Separating debug data from release builds**
   - Recommendation for managing debug information that aids in binary analysis and reverse engineering
   - However, decompilers can work without debug information, so security of a system must not depend on omitting such information
Roadmap and how to contribute

- New features, new compilers
- Separate guide for using GCC and Clang attribute annotations (work-in-progress)
- Contributions that improve readability, presentation, and accessibility also welcome
- Development happens in the Best Practices WG community on GitHub and on OpenSSF Slack.
- The Compiler Hardening sub-initiative has Zoom calls every other Wednesday at 13:00 UTC (see Public Calendar)
Ways to Participate

- Join a Working Group/Project
- Come to a Meeting (see Public Calendar)
- Collaborate on Slack
- Contribute on GitHub
- Become an Organizational Member
- Keep up to date by subscribing to the OpenSSF Mailing List
Compiler Options Hardening Guide for C and C++

Legal Notice

Copyright © Open Source Security Foundation®, The Linux Foundation®, & their contributors. The Linux Foundation has registered trademarks and uses trademarks. All other trademarks are those of their respective owners.

Per the OpenSSF Charter, this presentation is released under the Creative Commons Attribution 4.0 International License (CC-BY-4.0), available at https://creativecommons.org/licenses/by/4.0/. You are free to:

- Share — copy and redistribute the material in any medium or format for any purpose, even commercially.
- Adapt — remix, transform, and build upon the material for any purpose, even commercially.

The licensor cannot revoke these freedoms as long as you follow the license terms:

- Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
- No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.