C and C++: how safe can you get without a rewrite?

Before your next release, build with the OpenSSF hardening flags and a bounds-checked standard library, and run tests and fuzzers under ASan and UBSan in CI. Write new components in a memory-safe language, and judge each change by whether it checks the size, owner and lifetime of memory that outside data reaches.

Reviewed	Next review	Version	Written for
5 Nov 2026	**5 Dec 2026**	**1.0**	**Developers and tech leads maintaining production C and C++, and the security people who review it**

Latest version
sheets.einblick.dev/secure-cpp/

If you only do five things

Seven of the 2025 CWE Top 25 are memory-safety weaknesses[1]. Writing new code in memory-safe languages cut them from 76% of Android's vulnerabilities in 2019 to under 20% in 2025[2,3].

1. **Build every release with the OpenSSF hardening flags.** For GCC and Clang these include `-D_FORTIFY_SOURCE=3`, `-fstack-protector-strong`, `-fstack-clash-protection`, `-ftrivial-auto-var-init=zero`, PIE and full RELRO[4]; GCC's `-fhardened` sets them all on GNU/Linux[5]. On MSVC, add `/guard:cf`[6].
2. **Bounds-check the standard library in production.** Define `_GLIBCXX_ASSERTIONS`, set libc++'s mode to `_LIBCPP_HARDENING_MODE_FAST`[4,7] or define `_MSVC_STL_HARDENING=1`[8]. At Google, hardened libc++ cost 0.3% on average, found over 1,000 bugs and cut segfaults by 30%[9].
3. **Run every test under ASan and UBSan.** `-fsanitize=address` finds out-of-bounds access and use after free at about half speed[10], `-fsanitize=undefined` signed overflow and other undefined behavior[11]. Build those jobs with `-U_FORTIFY_SOURCE`, as distros often enable it[4].
4. **Fuzz every parser of outside data.** Give each file format, protocol and decoder a libFuzzer target, and run it on every pull request with ClusterFuzzLite under ASan, MSan and UBSan[12].
5. **Write new components in a memory-safe language.** New code holds most bugs: five-year-old code has 3.4 to 7.4 times lower vulnerability density[2]. For critical-infrastructure products, CISA and the FBI wanted a published memory-safety roadmap by the end of 2025[13].

Review a C or C++ pull request

1. **Does a length, count or offset come from the input?** Check it against the bytes left and the destination's size before any copy, index or allocation.
2. **Can a size calculation wrap?** Look for * and + on sizes, `int` lengths and signed-to-unsigned casts. Ask for `__builtin_mul_overflow`[14].
3. **Who owns each pointer, and can anything outlive it?** Check raw `new`, `delete` and `free`, stored `string_view` and `span` members, and callbacks that capture `this` or &.
4. **Is every error code and union tag checked?** Ask for `[[nodiscard]]` on functions that return errors and a tag check before each union read.
5. **Does it switch off a check?** Warning pragmas, `no_sanitize`, `_LIBCPP_HARDENING_MODE_NONE`, `[[indeterminate]]` and `/GS-` need a second reviewer.

C and C++ traps and what to use instead

- **`memcpy(buf, p, len)` with `len` from the input.** OpenSSL copied an AEAD IV into a fixed-size stack buffer without a length check, before any authentication (CVE-2025-15467, high)[15]. Check each length against the destination and the bytes left; `_FORTIFY_SOURCE=3` catches it only if it knows the size[4].
- **`n * size` and `len + 1` in size calculations.** Unsigned arithmetic wraps, and signed overflow is undefined, so the compiler may delete your overflow check[4]. Integer overflows in Onigmo, Ruby's regex engine, could let crafted patterns run code on GitLab (CVE-2026-97229, exploited)[16,17]. Use `__builtin_mul_overflow`[14].
- **Error codes that let a decoder keep going.** Brotli's decoder treated an invalid block switch as a need for more input and kept appending to an eight-byte buffer (CVE-2026-111497, exploited)[18,17]. Treat format errors as final, bound every append and mark error returns `[[nodiscard]]`.
- **Ownership an API leaves unclear.** OpenSSL's `PKCS7_verify()` freed a BIO its caller still owned, so the caller's `BIO_free()` became a use after free (CVE-2026-45447, high)[15]. Own with `std::unique_ptr`, don't keep a `std::string_view` past its owner, and in C document who frees what.
- **Reading a union member as the wrong type.** OpenSSL read an `ASN1_TYPE` union member without checking its type (CVE-2026-22796)[15], and CISA added type confusion bugs in V8 and Windows to its exploited list in 2026[17]. Check the tag first; in C++ use `std::variant`.
- **`printf(msg)` and your own logging wrappers.** OpenSSL passed a peer-supplied name as the format string to `ERR_raise_data()`, so `CN=%s%n` crashed CMP clients (CVE-2026-63073)[15]. Pass data through `"%s"`, and mark wrappers `format(printf, m, n)` to get printf's checks[19].

Tools that catch it

- **Sanitizers.** ASan (about 2x slower), UBSan, MSan for uninitialized reads (every library instrumented, and no `-ftrivial-auto-var-init`, which masks them) and TSan for data races, which can't run with ASan[10,20,4]. MSVC has `/fsanitize=address`[21].
- **Fuzzers.** libFuzzer now gets only bug fixes[22]; ClusterFuzzLite runs it on pull requests and in batch jobs[12].
- **Warnings.** `-Wall`, `-Wextra`, `-Wformat=2`, `-Wconversion` and `-Werror=format-security`[4]. In C++, Clang's `-Wunsafe-buffer-usage` flags buffer access through raw pointers[23]. On MSVC, `/sdl` makes uninitialized-variable warnings errors[24].

What you can skip

- **Waiting for C++26.** It isn't final yet[31], its library hardening standardizes checks the three major libraries already ship[30], and `-ftrivial-auto-var-init=zero` initializes locals today[4].
- **Rewriting stable code at once.** Vulnerabilities decay as code ages[2]: harden old code and rewrite the parsers that take outside data.
- **Valgrind in CI.** Sanitizers find similar bugs at a 2x to 4x slowdown, against up to 20x to 50x[4].

What changed in 2026

- **GCC 16 (30 April) zero-initializes locals under `-std=c++26`** and now defaults to `-std=gnu++20`[25,26].
- **Clang 23 added `-Wlifetime-safety`,** which detects use after free within a function. Add it to CI[27].
- **OpenSSL published 63 CVEs in 2026, against 6 in 2025;** all three High ones are memory-safety bugs[15]. 3.0 and 3.6 are now out of support; 3.5 is supported until April 2030, 4.0 until May 2027[28].
- **CISA added an exploited Brotli overflow on 27 October.** Update every copy, vendored ones too, to 1.0.10, 1.1.1 or 1.2.1[29,17].

Mistakes that cost the most

- **Checks only debug builds get.** GCC 15's libstdc++ enables its assertions by default only in unoptimized builds[32], and `_FORTIFY_SOURCE` needs `-O1` or higher[4]. Set both in release builds.
- **Shipping sanitizer runtimes.** Their environment-variable options can let attackers clobber root-owned files through setuid binaries. Only UBSan's minimal runtime or trap mode belongs in production[4,11].

What this looks like for your code

New project

- Ask first whether it can be Rust or another memory-safe language; see the Rust sheet.
- If it must be C++, use `std::span`, `std::unique_ptr` and `-Wunsafe-buffer-usage`[23].
- Start with the OpenSSF flags, a hardened library, sanitizer jobs and a fuzz target per parser.

Existing codebase

- Roll out hardening as Google did: tests first, then pre-production, then production over months[9].
- Replace `[0]` and `[1]` trailing arrays with `[]`, then add `-fstrict-flex-arrays=3`[4].
- Fuzz first the code CISA calls priority, network-facing and crypto[13]; publish a date after which new code is memory-safe[33].

High-assurance or regulated

- Follow Chromium's Rule of 2: of untrusted input, unsafe language and high privilege, pick two at most[34].
- Use libc++ extensive mode where benchmarks allow[7], and MSan with every library instrumented[20].
- On Arm cores with MTE, tag stack memory with GCC 16's `-fsanitize=memtag`[25].

Version history

5 Nov 2026, version 1.0 Published after a copy edit and a source-by-source fact check, which limited Google's overhead figure to hardened libc++, warned that zero-initialization masks MemorySanitizer findings, added the open C++26 ballot and updated OpenSSL's supported versions.

Sources

1. 2025 CWE Top 25. MITRE, updated 15 Dec 2025. cwe.mitre.org
2. Eliminating Memory Safety Vulnerabilities at the Source. Google, 25 Sep 2024. security.googleblog.com
3. Rust in Android: move fast and fix things. Google, 13 Nov 2025. blog.google
4. Compiler Options Hardening Guide for C and C++. OpenSSF, 20 Aug 2026. best.openssf.org
5. Instrumentation Options. GCC manual, checked 5 Nov 2026. gcc.gnu.org
6. /guard (Enable Control Flow Guard). Microsoft Learn, checked 5 Nov 2026. learn.microsoft.com
7. libc++ Hardening Modes. LLVM, checked 5 Nov 2026. libcxx.llvm.org
8. STL Hardening. Microsoft STL wiki, 4 Mar 2026. github.com
9. Retrofitting spatial safety to hundreds of millions of lines of C++. Google, 15 Nov 2024. security.googleblog.com
10. AddressSanitizer. Clang docs, checked 5 Nov 2026. clang.llvm.org
11. UndefinedBehaviorSanitizer. Clang docs, checked 5 Nov 2026. clang.llvm.org
12. ClusterFuzzLite. Google, checked 5 Nov 2026. google.github.io
13. Product Security Bad Practices, version 2. CISA and FBI, Jan 2025. ic3.gov
14. Integer Overflow Builtins. GCC manual, checked 5 Nov 2026. gcc.gnu.org
15. Vulnerabilities. OpenSSL, checked 5 Nov 2026. openssl-library.org
16. Regexp compilation fixes related to GitLab's CVEs. Ruby, 23 Oct 2026. ruby-lang.org
17. Known Exploited Vulnerabilities Catalog. CISA, data feed, 4 Nov 2026. github.com
18. October 21, 2026 Security Releases. Node.js, 21 Oct 2026. nodejs.org
19. Common Attributes. GCC manual, checked 5 Nov 2026. gcc.gnu.org
20. MemorySanitizer. Clang docs, checked 5 Nov 2026. clang.llvm.org
21. AddressSanitizer. Microsoft Learn, checked 5 Nov 2026. learn.microsoft.com
22. libFuzzer. LLVM, checked 5 Nov 2026. llvm.org
23. C++ Safe Buffers. Clang docs, checked 5 Nov 2026. clang.llvm.org
24. /sdl (Enable Additional Security Checks). Microsoft Learn, checked 5 Nov 2026. learn.microsoft.com
25. GCC 16 Release Series: Changes. GCC, 16.1, 30 Apr 2026. gcc.gnu.org
26. Optimize Options. GCC manual, checked 5 Nov 2026. gcc.gnu.org
27. Clang 23.1.0 Release Notes. LLVM, checked 5 Nov 2026. releases.llvm.org
28. Release strategy. OpenSSL, updated 7 May 2026. openssl-library.org
29. CVE-2026-111497 (Brotli decoder). Google, 22 Oct 2026. cve.org
30. P3471R4: Standard library hardening. WG21, 14 Feb 2025. open-std.org
31. P4433R0: Do not remove Contracts from C++26. WG21, 23 Oct 2026. open-std.org
32. GCC 15 Release Series: Changes. GCC, 15.1, 25 Apr 2025. gcc.gnu.org
33. The Case for Memory Safe Roadmaps. CISA, NSA and partners, Dec 2023. ic3.gov
34. The Rule Of 2. Chromium, checked 5 Nov 2026. chromium.googlesource.com