Tracking Warnings at Scale

Extending Clang Diagnostics to Support Issue Baselining and Backslide Prevention

Dave Bartolomeo (Apple)
Usama Hameed (Apple)

Motivation

- Improving memory safety for existing C and C++ code is important
- This is realized in Clang through a set of compiler options that produce warnings about unsafe code patterns

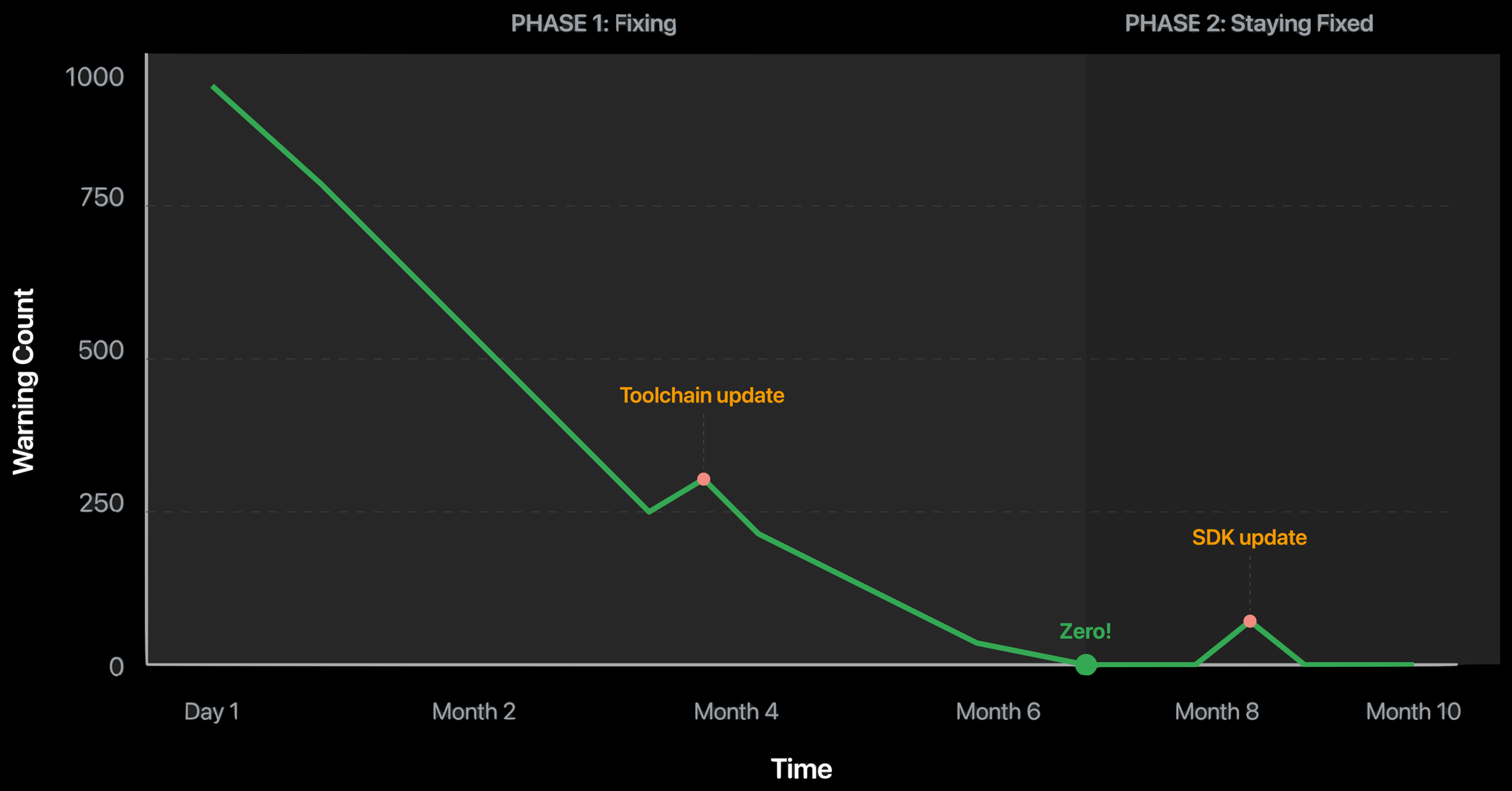
  `--Wunsafe-buffer-usage`

  `--fbounds-safety`
- When you enable these options, you can get hundreds or thousands of warnings

Problem: Fixing warnings on a large scale is hard

- Can take person-months or person-years to fix all the warnings
- Everything is moving underneath you
  - Source code, SDKs, the compiler itself
- It is easy to go backwards
- Developers can suppress warnings or accidentally disable them entirely

Warning Adoption at Scale



Solution: Support for tracking warnings
Developer Workflow for PRs

- Enable a warning
- Record existing instances of the warning as a “baseline”
  - Baselined warnings are not flagged during PRs
- Flag new instances of the warning when they are introduced in PRs
  - Fix, suppress, or defer fixing
  - Reviewers can scrutinize suppressions
- Ratchet up the baseline as warnings are fixed
- Measure progress!

How to implement this in the compiler and toolchain?

Which warnings changed? → Track warnings using structured diagnostics output (SARIF)

Stable diagnostic IDs

Stable instance fingerprints

Baselining of warnings → Warning tracking file (JSON in repo)

Tracking suppressed warnings → Emit suppression records in SARIF

Structured Diagnostics Output: SARIF

- Open standard JSON format for diagnostics: Static Analysis Results Interchange Format
  - https://docs.oasis-open.org/sarif/sarif/v2.1.0/sarif-v2.1.0.html
  - Used by other static analysis tools to manage warnings
  - Emitted by other C++ compilers
- Already in Clang, supported by the Clang Static Analyzer
- New: Added as output from Clang compiler, enabled with `-fdiagnostics-add-output=sarif:file=`*`path`*

Stable Diagnostic IDs

- Each Clang diagnostic now has a stable string identifier
  - Defaults to the internal Clang identifier (e.g., “warn_unsafe_buffer_operation”)
  - Can be explicitly specified in TableGen file
  - Emitted into SARIF file
- If a diagnostic is renamed or refactored, old IDs are preserved in LegacyStableIds for backwards compatibility
- New: Added to mainline Clang

Tracking Warning Instances over Time: Fingerprints

- How do you tell if two warning instances from two different builds are “the same”?
  - File and line number will not work, because line can move around
- Solution: Fingerprint - A string that uniquely identifies a warning while being resilient to unrelated changes
- Clang Static Analyzer already has one of these: Hash of checker name, function declaration, column number, line text, and warning message
  - Resilient against unrelated code being inserted or removed
  - Other algorithms we will experiment with

Reporting Suppressed Warnings

- New `-Waudit=`*`name`* option
- Emits warnings to SARIF even if they were suppressed or disabled
  - `#pragma clang diagnostic ignored "-W`*`name`*`"`
  - `--Wno-`*`name`*
  - Suppression mapping file
- SARIF output contains details about how that warning was suppressed
- Only enable for “interesting” warnings

What's in Clang Today

```
$ clang -Waudit=unsafe-buffer-usage -fdiagnostics-add-output=sarif:network.sarif net.c
```

```
{
  "runs": [{
    "tool": { "driver": { "rules": [
      {
        "id": “warn_unsafe_buffer_operation”,      // Stable type ID
        "deprecatedIds": […],                     // Legacy ID tracking
        "defaultConfiguration": {
          "level": "warning"
        }
      }
    ]}},
    "results": [{
      "ruleId": “warn_unsafe_buffer_operation”,
      "locations": […],
      "partialFingerprints": {                   // Instance signature
        "issueHash": "abc123..."
      },
      "suppressions": […]                        // Suppression info
     …
    }]
  }]
}
```

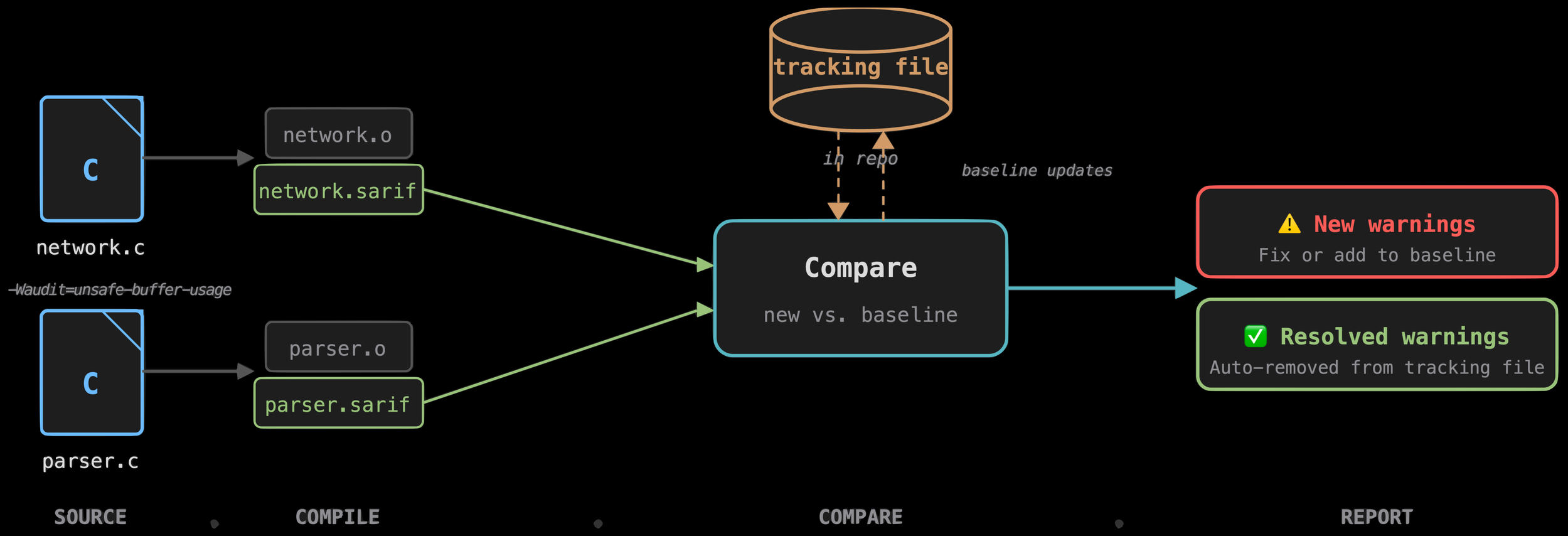
Warning Tracking File

- Suppressed and baselined warnings are tracked in a JSON file in the repo
- Every decision is auditable in git history
- Each record contains:
  - Stable warning ID
  - Warning instance fingerprints
  - Justification for suppression
- File is updated by the PR tooling when a warning is baselined, suppressed, or fixed

PR Analysis

- Compares SARIF output against the old and the new tracking files
  - Reports new warnings, resolved warnings, and suppressed warnings
  - New unsuppressed warnings are flagged
  - PR bot allows developers to suppress or baseline warnings, and updates the tracking file accordingly
- Planning to open source this tool

Putting It All Together



What Clang Users Will Be Able to Do

- Track warning progress over time
- Avoid backsliding
- Review suppressions in PRs
- No external database needed
- This will make it easier to improve the quality and security of C and C++ code

Next Steps

- Tooling for PR annotations and measuring progress
- Making sure warnings and security-related options aren’t disabled accidentally
- Experiment with other fingerprint algorithms and combinations
- Come talk to us if you have feedback or are interested in trying this out!

TM and © 2026 Apple Inc. All rights reserved.