Security advisory for the standard library (CVE-2022-21658)

Jan. 20, 2022 · The Rust Security Response WG

> *This is a cross-post of the official security advisory. The official advisory contains a signed version with our PGP key, as well.*

The Rust Security Response WG was notified that the `std::fs::remove_dir_all` standard library function is vulnerable to a race condition enabling symlink following (CWE-363). An attacker could use this security issue to trick a privileged program into deleting files and directories the attacker couldn't otherwise access or delete.

This issue has been assigned CVE-2022-21658.

Overview

Let's suppose an attacker obtained unprivileged access to a system and needed to delete a system directory called `sensitive/`, but they didn't have the permissions to do so. If `std::fs::remove_dir_all` followed symbolic links, they could find a privileged program that removes a directory they have access to (called `temp/`), create a symlink from `temp/foo` to `sensitive/`, and wait for the privileged program to delete `foo/`. The privileged program would follow the symlink from `temp/foo` to `sensitive/` while recursively deleting, resulting in `sensitive/` being deleted.

To prevent such attacks, `std::fs::remove_dir_all` already includes protection to avoid recursively deleting symlinks, as described in its documentation:

deleting symlinks, as described in its documentation:

> *This function does **not** follow symbolic links and it will simply remove the symbolic link itself.*

Unfortunately that check was implemented incorrectly in the standard library, resulting in a TOCTOU (Time-of-check Time-of-use) race condition. Instead of telling the system not to follow symlinks, the standard library first checked whether the thing it was about to delete was a symlink, and otherwise it would proceed to recursively delete the directory.

This exposed a race condition: an attacker could create a directory and replace it with a symlink between the check and the actual deletion. While this attack likely won't work the first time it's attempted, in our experimentation we were able to reliably perform it within a couple of seconds.

Affected Versions

Rust 1.0.0 through Rust 1.58.0 is affected by this vulnerability. We're going to release Rust 1.58.1 later today, which will include mitigations for this vulnerability. Patches to the Rust standard library are also available for custom-built Rust toolchains here.

Note that the following targets don't have usable APIs to properly mitigate the attack, and are thus still vulnerable even with a patched toolchain:

- macOS before version 10.10 (Yosemite)
- REDOX

Mitigations

We recommend everyone to update to Rust 1.58.1 as soon as possible, especially people developing programs expected to run in privileged contexts (including system daemons and setuid binaries), as those have the highest risk of being affected by this.

Note that adding checks in your codebase before calling `remove_dir_all` will **not** mitigate the vulnerability, as they would also be vulnerable to race conditions like `remove_dir_all` itself. The existing mitigation is working as intended outside of race conditions.

Acknowledgments

We want to thank Hans Kratz for independently discovering and disclosing this issue to us according to

the Rust security policy, for developing the fix for UNIX-like targets and for reviewing fixes for other platforms.

We also want to thank Florian Weimer for reviewing the UNIX-like fix and for reporting the same issue back in 2018, even though the Security Response WG didn't realize the severity of the issue at the time.

Finally we want to thank Pietro Albini for coordinating the security response and writing this advisory, Chris Denton for writing the Windows fix, Alex Crichton for writing the WASI fix, and Mara Bos for reviewing the patches.