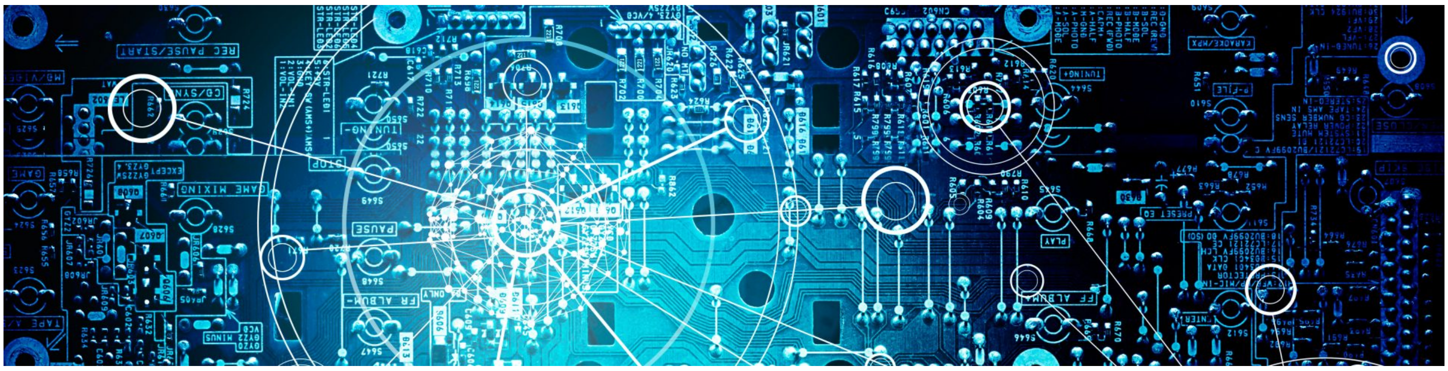
KUDELSKI SECURITY | RESEARCH

The Latest News from Research at Kudelski Security

HOME CATEGORIES ⌄ HOME CATEGORIES ⌄



AUDITING RUST CRYPTO: THE FIRST HOURS

February 7, 2019 JP Aumasson Crypto Leave a comment

> *We are besieged by simple problems....*
> *Checklists can provide protection*

SEARCH

Se

CATEGORIES

Select

ARCHIVES

Select

> — Atul Gawande, The Checklist Manifesto: How to Get Things Right

Checklists are a simple yet effective component of security and safety procedures in various fields, from flight safety and surgery to network security, and of course cryptography.

Two years ago we wrote Auditing code for crypto flaws: the first 30 minutes, which lists basic sanity checks that an auditor can do when looking at crypto software. These checks are about the type of crypto components used, rather than their implementation, and therefore apply to any programming language.

Today we provide a new list of sanity checks, but this time less specific to crypto, and more specific to a programming language, namely Rust. This list is based on our experience auditing crypto software in Rust—from Wire to Zcash—and comes from internal notes that our team uses when starting a new audit. We hope that this list will benefit our readers, be it fellow auditors or software engineers.

These checks can not only help you find logic bugs or bad crypto choices, but also software bugs, or bugs caused by an unsafe use of the Rust language—although Rust is designed to prevent developers from shooting themselves in the foot, it provides enough leeway for them to do so and set the house on fire with a single line of code.

So here's a couple of things you want to check when starting the audit of a crypto software written in Rust:

1. Review clippy warnings; most of the time these are benign or irrelevant, but they can help spotting red flags.
2. Build and run all the unit tests, assess the code coverage and keep note of the un(der)tested component.
3. Review the dependencies listed in Cargo.toml and Cargo.lock: Will the latest version be used? (preferable but not always the right choice) Are these established, trustworthy packages? You may use the subcommand cargo-audit (thanks @dues__ for the pointer).
4. Look for `unsafe` code blocks, and evaluate the risk (can an attacker control the input used in these blocks? etc.)
5. Look for risky uses of `unwrap()`, which can cause panics, as opposed to pattern-matched errors.
6. Look for potential integer overflows (these will cause a panic when run in debug mode, but will silently wrap around in release mode).
7. Look for "public-in-private" types, whose behavior may not be as intended and could be used to bypass privacy checks.
8. Look for any recursive function calls and see if they could be abused to overflow the stack.
9. If FFI is used, find the foreign code called, and if relevant add its audit as a subproject.
10. Identify APIs that could benefit from being fuzzed, keep note of these.
11. Find which crypto primitives are used, what third-party libraries if any, and keep note of any new implementations of crypto components.

12. Find what RNG is used for crypto and security purposes? `rand::thread_rng` should be fine most of the time, but may fall back to a weak RNG is the OS' fails.
13. Are sensitive values set to zero after being used? In Rust this may be done using the `Drop` trait rather than explicitly.

Some disclaimers:

- As the post's title suggest, these are only basic checks that you may do when starting an audit, but an audit should not only consist of these. Most of the time will be spent understanding the logic of the code, looking for non-trivial bugs in its subcomponents and interactions thereof.
- This list is by no means meant to be exhaustive, and we'll happily add missing items that our readers will report (by leaving a comment of emailing the author). For less trivial information on Rust bugs, we recommend The Rustonomicon and Bugs You'll Probably Only Have in Rust.

Update (Feb. 8): The HN thread about this post includes insightful comments by people who know Rust much better than I do.

BLOCKCHAIN CODE CODING CRYPTO CRYPTOCURRENCY CRYPTOGRAPHY FEATURED IMAGE PROGRAMMING RUST WIRE ZCASH

« Audit of Zcash's Sapling update

Audit of KZen's Curv library »

LEAVE A REPLY

Enter your comment here...