Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Audit-Report Threema Rust Crypto Libraries 02.2022

Cure53, Dr.-Ing. M. Heiderich, Dr. A. Pirker, Dipl.-Ing. D. Gstir, R. Weinberger

Index

Introduction

This report - entitled 3MA-02 - details the scope, results, and conclusory summaries of a cryptography review and code audit against two community-maintained libraries written in Rust, namely the XSalsa20Poly1305 (aka *crypto_secretbox*) and the Rust re-implementation of the NaCl *crypto_box*. The work was requested by the Threema GmbH in November 2021 and initiated by Cure53 in February 2022, namely in CW06. A total of three days were invested to reach the coverage expected for this project.

The testing conducted for 3MA-02 was divided into two separate work packages (WPs) for execution efficiency, as follows:

- **WP1**: Cryptography Review against XSalsa20Poly1305 aka *crypto_secretbox*
- **WP2**: Cryptography Review against re-implementation of NaCl *crypto_box*

Cure53 was granted source-code access to both libraries and any other pertinent information required to complete the reviews. For these purposes, the methodology chosen was white-box; notably, all software in scope is open-source software.

A team of four senior testers was assigned to this project's preparation, execution, and finalization. All preparatory actions were completed in early February 2022, namely in CW05, to ensure that the testing phase could proceed without hindrance.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Communications were facilitated via the dedicated Threema channel originally deployed to combine the workspaces of Threema and Cure53 during a previous audit, thereby allowing an optimal collaborative working environment to flourish. All participatory personnel from both parties were invited to partake throughout the test preparations and discussions. One can denote that communications proceeded smoothly on the whole. The scope was well-prepared and clear, no noteworthy roadblocks were encountered throughout testing, and cross-team queries were kept to a minimum as a result. Threema delivered excellent test preparation and assisted the Cure53 team in every respect to procure maximum coverage and depth levels for this exercise.

Cure53 gave frequent status updates concerning the test and any related findings, whilst simultaneously offering prompt queries and receiving efficient, effective answers from the maintainers. Live reporting was not requested, which in hindsight proved a sufficient decision considering the relatively low severity levels of the findings detected. Regarding the findings in particular, the Cure53 team achieved comprehensive coverage over the WP1 and WP2 scope items, identifying a grand total of one sole finding. Positively, this was merely deemed a general weakness with lower exploitation potential.

Even though a small volume of findings was expected prior to the audit - owing to the highly-specific scope and limited timeframe within which to complete testing - nevertheless, that only one finding was unearthed remains a positive indication regarding the security posture of both cryptography libraries. Furthermore, the categorization of the miscellaneous issue with a mere *Info* severity rating corroborates the strong impression gained, and should be trivially easy to address and mitigate.

The report will now shed more light on the scope and testing setup as well as provide a comprehensive breakdown of the available materials. Subsequently, the report will outline the assigned testing methodology and then list all findings in chronological order; first the vulnerabilities, followed by the general weaknesses detected. Each finding will be accompanied by a technical description and Proof of Concepts (PoCs) where applicable, plus any relevant mitigatory or preventative advice to action.

In summation, the report will finalize with a conclusion in which the Cure53 team will elaborate on the impressions gained toward the general security posture of the two pertinent libraries written in Rust, giving high-level hardening advice where applicable.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Scope

- **Cryptography review and source code audits against two libraries written in Rust**
  - **WP1**: Cryptography review against XSalsa20Poly1305 aka *crypto_secretbox*
    - All sources have been shared with Cure53 and are available as OSS
      - https://github.com/RustCrypto/AEADs/tree/master/xsalsa20poly1305
    - Version in scope:
      - 0.8.0
    - Commit in scope:
      - 53a4d5736a7697ab489edae9e66701c72be5bc05
  - **WP2**: Cryptography review against re-implementation of NaCl *crypto_box*
    - All sources have been shared with Cure53 and are available as OSS
      - https://github.com/RustCrypto/nacl-compat/tree/master/crypto_box
    - Version in scope:
      - 0.7.1
    - Commit in scope:
      - 95506c5d8cce56c69a92d45a253139443131d1ac

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Test Methodology

This source code audit constituted a review of two repositories implemented in Rust, namely:

- The reimplementation of NaCl *crypto_box*[1]
- The XSalsa20Poly1305 aka *crypto_secretbox*[2]

The security assessment was conducted via a manual source-code inspection. Before commencing the review, the confirmation was made that the dependencies of the crates were out-of-scope for this audit. Therefore, only the provided source was subject to testing here.

The NaCl *crypto_box* pertains to a reimplementation of the Networking and Cryptography library (NaCl)[3] and primarily constitutes a thin wrapper layer for alternative crates such as *x25519_dalek*[4]. The NaCl *crypto_box* offers an implementation to support an elliptic-curve integrated encryption scheme, which utilizes an X25519 Diffie-Hellman key exchange to establish a shared secret. This consequently serves as input to a key-derivation step using either HSalsa20 or HChaCha20.

The implementation utilizes this shared key as the key parameter to create an instance of either XSalsa20Poly1305 or XChaCha20Poly1305 after key derivation. The crate was subject to deep-dive assessments regarding the creation of both secret and keys, which garnered a positive impression as their respective implementations were considered sound and correct following testing. Said implementations block attackers from altering key material after creation, and the secret is cleared after dropping. Furthermore, the Diffie-Hellman key exchange is sufficiently implemented through correct usage of the *x25519_dalek* crate.

The XSalsa20Poly1305 implementation corresponds to a wrapper crate for alternative existing crates such as salsa20[5] and poly1305[6] for example, which initiates by delegating the cryptographic computations to said crates. The verification was made that the implementation invokes the functions of alternate crates correctly.

[1] https://github.com/RustCrypto/AEADs/tree/master/xsalsa20poly1305
[2] https://github.com/RustCrypto/nacl-compat/tree/master/crypto_box
[3] https://nacl.cr.yp.to/index.html
[4] https://docs.rs/x25519-dalek/latest/x25519_dalek/
[5] https://lib.rs/crates/salsa20
[6] https://starry-network.github.io/starry_node/poly1305/index.html

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Both implementations were assessed for common attack scenarios against libraries, including:

- **Timing attacks** on authentication tag comparisons for the XSalsa20Poly1305 implementation. The comparison of the authentication tag is achieved using a time-constant approach, which does not provide any side-channel information to a would-be attacker.
- **Absent or insufficient input-data validation**, such as key-length validation and similar. A validation deficiency of this nature often leads to unintended usage of the library and therefore facilitates the proliferation of a multitude of vulnerabilities. Due to the correct usage of Rust and fixed size arrays, the testing team was able to confirm that the provided input data is sufficiently validated.
- **Sensitive information leakage** via incorrectly-implemented wiping processes. This should be considered an essential function for libraries, considering that secret keys are utilized for encryption. Regarding this, insufficient data removal may result in data leakage of any magnitude. Therefore, assessments were made to determine whether the implementations correctly clear sensitive information of any kind. Positively, one could confirm that all sensitive information is wiped from memory. However, depending on the target platform and the use case, some issues could potentially arise with the usage of the *zeroize* crate, as detailed in ticket 3MA-02-001.
- **Incorrect usage of cryptographic primitives**, which could have severe consequences. On some occasions, the API of the low-level library would not be concise enough to prevent misuse. Testing was initiated to determine whether the cryptographic primitives offered by the dependencies - namely X25519, XSalsa20, HSalsa20, and HChaCha Poly1305 - were integrated correctly. Positively, no issues were detected in this area.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Miscellaneous Issues

This section covers any and all noteworthy findings that did not lead to an exploit but might assist an attacker in successfully achieving malicious objectives in the future. Most of these results are vulnerable code snippets that did not provide an easy way to be called. Conclusively, while a vulnerability is present, an exploit might not always be possible.

3MA-02-001 WP1/2: Potentially insufficient protection of in-memory secrets *(Info)*

Clearing secret data from memory once deprecated is a standard practice for cryptographic hygiene. Both *crypto_box* and *xsalsa20poly1305* handle data clearance via the *zeroize*[7] crate. This crate ensures that any subsequent reads following a zeroization of a memory region will only return zeroed memory. Additionally, it guarantees that calls of this nature are not optimized away by the compiler. This can occur when the compiler infers that a certain memory buffer is unutilized after the zeroization.

However, a few key facets remain uncovered by *zeroize* and can therefore instigate risk depending on the threat model. Additional information is provided by the *zeroize* documentation, which is briefly summarized as follows:

- Certain Rust operations can retain unintentional copies of secrets in memory. These include *move* and *copy* operations; heap reallocations when using *Vec* and *String*; and the instance whereby reference borrowers make copies of data.
- Memory contents can be swapped to disk by the OS or included in core dumps.
- Leakage via CPU registers.
- Protection against microarchitectural attacks such as Meltdown[8] and Spectre[9].

In the vast majority of typical scenarios, the usage of *zeroize* to achieve a basic level of cryptographic hygiene provides sufficient protection. However, certain use cases such as embedded devices or shared systems whereby an attacker may have greater access privileges - and thus greater attack opportunities - require additional measures to achieve comprehensive prevention.

Therefore, it is recommended to analyze the use case of Threema regarding attack vectors of this nature. Should they be considered negligible, no mitigatory action would be needed here.

[7] https://docs.rs/zeroize/latest/zeroize/
[8] https://meltdownattack.com/
[9] https://spectreattack.com/

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Conclusions

The impressions gained during this report - which details and extrapolates on all findings identified during the CW06 testing against two specific libraries written in Rust by the Cure53 team - will now be discussed at length. To summarize, the confirmation can be made that the scope items under scrutiny have garnered a positive impression.

This audit assessed two Rust libraries (crates) entitled *xsalsa20poly1305* (version 0.8.0) and *crypto_box* (version 0.7.1). Both libraries are components of the RustCrypto GitHub group, which aims to implement cryptography in pure Rust. Both crates are intended as replacements for the *crypto_secretbox** and *crypto_box** functions of the NaCl C library. The primary objective of this audit was to assess the cryptographic accuracy of these libraries and their functional compatibility to NaCl. Compositionally, the libraries in scope are essentially wrapper crates for other cryptographic libraries. Dependencies that implement the basic cryptographic primitives - such as Salsa20, X25519, and Poly1305 - were not in scope for this audit.

The Cure53 testing team was in constant communication with the customer throughout the audit, frequently relaying status updates and raising any queries or concerns when needed. The usage of the dedicated Threema channel proved efficient and effective, with assistance provided whenever requested. Positively, no vulnerabilities or other severe issues could be identified during the allotted time frame of this audit. Both libraries appear constitutionally sound and carefully written to prevent any common issues that blight cryptographic libraries in general.

The auditors inspected the provided source code for typical vulnerabilities relating to cryptographic implementations such as timing attacks, improper input sanitization and validation, and sensitive data leakage to memory. In all of these areas, *xsalsa20poly1305* and *crypto_box* withheld well to scrutiny, deterring any kind of vulnerability or weakness. All sensitive data for buffers allocated by the code is sufficiently wiped after use, or delegates for wiping when de-initialized by the Rust runtime using the *zeroize* crate. Furthermore, the generation of cryptographically-sensitive information such as secret keys for the X25519 curve and nonces, for example, appear sound and utilize robust random-number generators. Finally, secret-key usage - even though ‘unclamped’ upon generation (similarly to other libraries such as *x25519_dalek*) - is implemented correctly. This owes to the clamping operation performed on the *x25519_dalek* crate’s x25519 function.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

While the code integrates common patterns to avoid sensitive information leakage, one must note that Rust is considered a programming language still in relative infancy that undergoes heavy and recurrent development. Any alterations made to the Rust compiler - particularly pertaining to the optimization logic - could therefore undermine these mitigations.

In summary, excellent coverage over both work packages has been achieved, with the positive outcome of one sole miscellaneous issue unearthed. Moving forward, from a cryptography and security viewpoint it is highly recommended to perform deep-dive code assessments against those referred dependencies, as the core functionality of both crates lies within them.

Cure53 would like to thank Silvan Engeler, Manuel Kasper and Danilo Bargen from the Threema team for their excellent project coordination, support and assistance, both before and during this assignment.