Secure Code Review

Findings Report Presented to:

TrustWallet

September 15, 2023
Version: 2.0

Presented by:

Kudelski Security, Inc.
Route de Genève 22-24
1033 Cheseaux-sur-Lausanne
Switzerland

FOR PUBLIC RELEASE

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

Overview

TrustWallet engaged Kudelski Security to perform a Cryptographic review of selected crates of the external public library `starknet-rs`[1], in particular of the crates `starknet-crypto`, `starknet.curve` and `starknet-ff`, plus their usage by TrustWallet in their own source code.

The assessment was conducted remotely by the Kudelski Security Team. Testing took place between July 20, 2023 and August 7, 2023, and focused on the following objectives:

- Provide the customer with an assessment of their overall security posture and any risks with regards to general code practices, as well as cryptography primitives and technical specification matching.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the result of our security analysis.
- To perform additional fuzzing to test the robustness of the code.

This report contains an overview of the findings discovered during the engagement.

An additional review took place on September 15, 2023, of the pull request https://github.com/trustwallet/wallet-core/pull/3381.

Key Findings

The following are the major themes and issues identified during the testing period. These, along with other items, within the findings section, should be prioritized for remediation to reduce to the risk they pose.

- Inconsistency between formal definitions of the cryptographic algorithms and their implementation in the Starknet library.
- Parameters of the cryptographic algorithms are hardcoded.
- Inadequate security against the side channel attacks.

During the code review, the following positive observations were noted regarding the scope of the engagement:

- The code was clean, concise, and commented throughout.
- Finally, at every stage of the audit, the team remained accessible and promptly addressed any questions or concerns.

[1] https://github.com/xJonathanLEI/starknet-rs

Scope and Rules of Engagement

The Kudelski Security team performed a Secure Code Review for TrustWallet. The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

The source code was supplied with the commit hashes in private repositories at:

- https://github.com/xJonathanLEI/starknet-rs/tree/starknet-curve/v0.3.0/
  - Commit Hash: `9190156676d7c84987b7c711d53336c640e1fe85`
  - Subfolder `starknet-curve (Version 0.3.0)`
- https://github.com/xJonathanLEI/starknet-rs/tree/starknet-crypto/v0.5.1
  - Commit Hash: `91425ad9c20cceaf87f5ae179341f7e1f7958507`
  - Subfolder `starknet-crypto (Version 0.5.1)`
- https://github.com/xJonathanLEI/starknet-rs/tree/starknet-ff/v0.3.3
  - Commit Hash: `a04f09addc9a8cbe85f8e9d080ce80ce7fa90936`
  - Subfolder `starknet-ff (Version 0.3.3)`
- https://github.com/trustwallet/wallet-core/commit/502878aadcac340b98c8619dd8e141dec94b1ad5
  - `Key_pairs.rs`

In-Scope Crates		
Starknet-crypto	Starknet-cuve	Starknet-ff
src ├── ecdsa.rs ├── error.rs ├── fe_utils.rs ├── lib.rs ├── pedersen_hash.rs ├── pedersen_points.rs ├── poseidon_hash.rs ├── rfc6979.rs └── test_utils.rs	src ├── curve_params.rs ├── ec_point.rs └── lib.rs	src ├── fr.rs └── lib.rs

Table 1: Scope

A Subsequent review was conducted on September 17 to assess the remediation efforts carried out by the Client

Remarks

During the initial Kick-off phase of the project, the Kudelski Security team assessed which components of the scope were of most concern for the Client. Following this discussion, Kudelski Security team focused on the correct implementation of cryptographic primitives and common security pitfalls in Rust. Side-channel attacks and time-constant implementation of cryptography were not of primary concern. Nevertheless, the Kudelski Security team do present issues with cryptographic implementations vulnerable to side-channel attacks as informational notes.

TECHNICAL ANALYSIS & FINDINGS

During the Secure Code Review, the Kudelski security team discovered 6 findings that had a low severity rating.

The following chart displays the findings by severity.

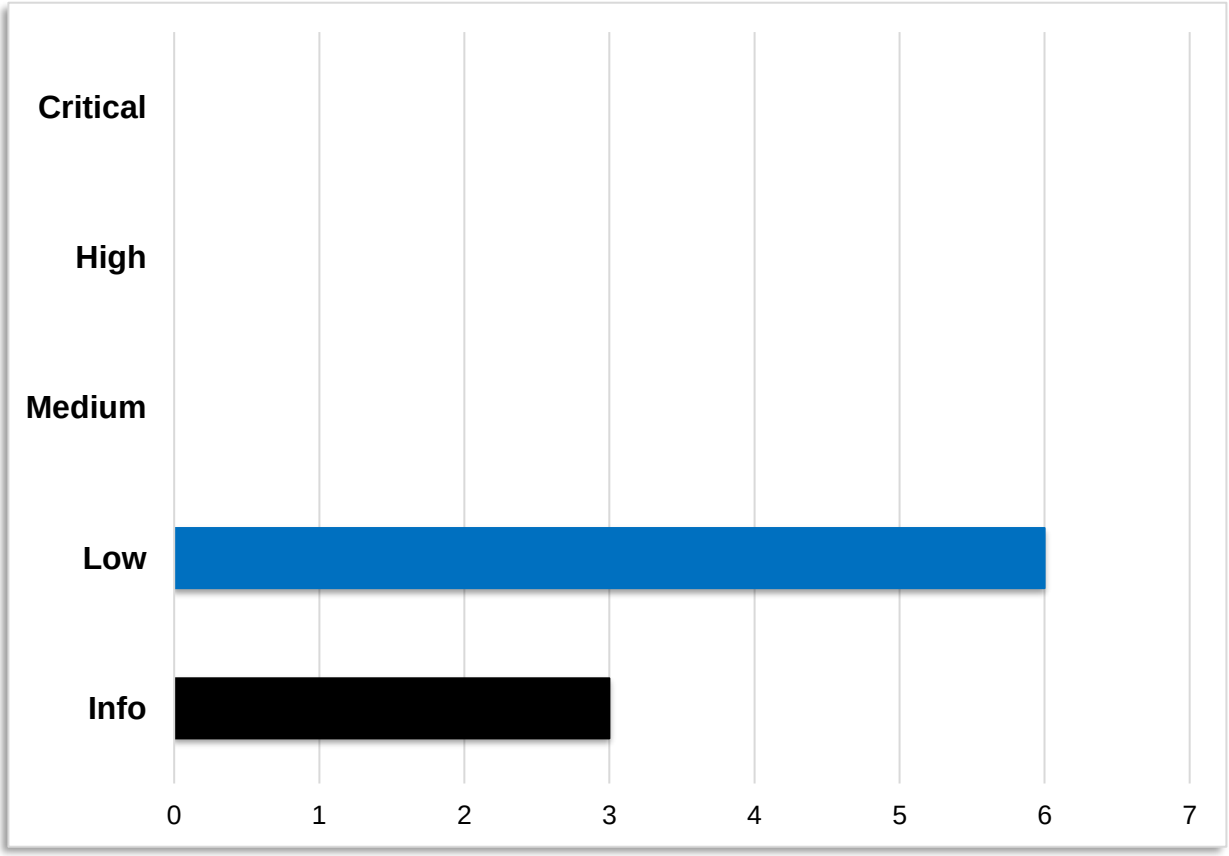

Figure 1: Findings by Severity

Findings

The *Findings* section provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

#	Severity	Description	Status
KS-TW-01	**LOW**	Poseidon MDS matrix diverges from paper's specification	Open
KS-TW-02	**LOW**	ECDSA verify function accepts two distinct points	Open
KS-TW-03	**LOW**	Elliptic curve point multiplication is not constant time	Open
KS-TW-04	**LOW**	Pedersen hash is not constant time	Open
KS-TW-05	**LOW**	Poseidon hash instances are hardcoded	Open
KS-TW-06	**LOW**	Infinite loop when calling *ecdsa_sign* in tw_starknet	Resolved
KS-TW-07	**INFORMATIONAL**	Incorrect error handling in library crate	Acknowledged
KS-TW-08	**INFORMATIONAL**	Remaining TODOs in the code	Acknowledged
KS-TW-09	**INFORMATIONAL**	Insufficient tests coverage in starknet-curve	Acknowledged

Table 2: Findings Overview

METHODOLOGY

During this source code review, the Kudelski Security Services team reviewed code within the project within an appropriate IDE. During every review, the team spends considerable time working with the Client to determine correct and expected functionality, business logic, and content to ensure that findings incorporate this business logic into each description and impact. Following this discovery phase the team works through the following categories:

- Authentication
- Authorization and Access Control
- Auditing and Logging
- Injection and Tampering
- Configuration Issues
- Logic Flaws
- Cryptography

These categories incorporate common vulnerabilities such as the OWASP Top 10.

Tools

The following tools were used during this portion of the test.

- Visual Studio Code
- Semgrep
- Cargo-Fuzz

Vulnerability Scoring Systems

Kudelski Security utilizes a vulnerability scoring system based on impact of the vulnerability, likelihood of an attack against the vulnerability, and the difficulty of executing an attack against the vulnerability based on a three level rating system rating system.

Impact
The overall effect of the vulnerability against the system or organization based on the areas of concern or affected components discussed with the client during the scoping of the engagement.

> **High:**
> The vulnerability has a severe effect on the company and systems or has an affect within one of the primary areas of concern noted by the client.
>
> **Medium:**
> It is reasonable to assume that the vulnerability would have a measurable affect on the company and systems that may cause minor financial or reputational damage.
>
> **Low:**
> There is little to no affect from the vulnerability being compromised. These vulnerabilities could lead to complex attacks or create footholds used in more severe attacks.

Likelihood
The likelihood of an attacker discovering a vulnerability, exploiting it, and obtaining a foothold varies based on a variety of factors including compensating controls, location of the application, availability of commonly used exploits, and institutional knowledge.

> **High:**
> It is extremely likely that this vulnerability will be discovered and abused.
>
> **Medium:**
> It is likely that this vulnerability will be discovered and abused by a skilled attacker.
>
> **Low:**
> It is unlikely that this vulnerability will be discovered or abused when discovered.

Difficulty
Difficulty is measured according to the ease of exploit by an attacker based on availability of readily available exploits, knowledge of the system, and complexity of attack. It should be noted that a LOW difficulty results in a HIGHER severity.

> **Easy:**
> The vulnerability is easy to exploit or has readily available techniques for exploit.
>
> **Moderate:**
> The vulnerability is partially defended against, difficult to exploit, or requires a skilled attacker to exploit.
>
> **Difficult:**
> The vulnerability is difficult to exploit and requires advanced knowledge from a skilled attacker to write an exploit.

Severity
Severity is the overall score of the weakness or vulnerability as it is measured from Impact, Likelihood, and Difficulty.