Report

v. 1.0

Customer

StarkWare

Cairo code audit

Perpetual. V 3.2

7th November 2023

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	07.11.23	A. Zveryanskaya	Initial Draft
0.2	07.11.23	A. Zveryanskaya	Minor revision
1.0	07.11.23	A. Zveryanskaya	Release

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting (Mikhail Vladimirov and Dmitry Khovratovich) at the customer request. The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations.

StarkWare Industries is an Israeli software company that specializes in cryptography. It develops zero-knowledge proof technology that compresses information to address the scalability problem of the blockchain, and works on the Ethereum platform.

3 Project scope

We have audited the version 3.1 of Perpetual project, concretely the diff between commit eaa2846 and commit e6189aa in the starkware-libs/stark-perpetual repository. The fixes were provided via commit 3eb3a26.

Files:

exchange/definitions/			
	constants.cairo	signature_ message_hashes.cairo	
perpetual/definitions/			
	constants.cairo	general_config_hash.cairo	general_config.cairo
perpetual/order/			
	order.cairo		
perpetual/output/			
	data_availability.cairo	forced.cairo	
perpetual/position/			
	status.cairo		
perpetual/transactions/			
	withdrawal.cairo		
perpetual/e			
	xecute_batch_utils.cairo	execute_batch.cairo	

4 Methodology

The methodology is not a strict formal procedure, but rather a selection of methods and tactics combined differently and tuned for each particular project, depending on the project structure and technologies used, as well as on client expectations from the audit.

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows best code practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places as well as their visibility scopes and access levels are relevant. At this phase, we understand overall system architecture and how different parts of the code are related to each other.
- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and done properly. At this phase, we understand user roles and permissions, as well as what assets the system ought to protect.
- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check if code actually does what it is supposed to do, if that algorithms are optimal and correct, and if proper data types are used. We also make sure that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

We classify issues by the following severity levels:

- **Critical issue** directly affects the smart contract functionality and may cause a significant loss.
- **Major issue** is either a solid performance problem or a sign of misuse: a slight code modification or environment change may lead to loss of funds or data. Sometimes it is an abuse of unclear code behaviour which should be double checked.
- **Moderate issue** is not an immediate problem, but rather suboptimal performance in edge cases, an obviously bad code practice, or a situation where the code is correct only in certain business flows.
- **Minor issues** contain code style, best practices and other recommendations.

5 Our findings

We found 2 major, and a few less important issues. All identified issues have been fixed.

	Info	Fixed
Major	0	2
Moderate	0	4
Minor	0	2

Fixed 8 out of 8 issues

6 Major Issues

CVF-2. FIXED

- **Category** Unclear behavior
- **Source** execute_batch_utils.cairo

Description This function cannot handle the case when n_risk_factor_segments is zero.

Recommendation Consider explicitly forbidding such a case via an assert.

```
78 +func validate_risk_factor_function{range_check_ptr}(
79 +    risk_factor_segments: RiskFactorSegment*,
   ↪ n_risk_factor_segments
```

CVF-3. FIXED

- **Category** Unclear behavior
- **Source** signature_message_hashes.cairo

Description It is not clear how the lengths of arrays to be hashed are related to the meta-data fields.

Recommendation Consider writing this explicitly so that it becomes clear why collisions are not possible.

```
330 +// packed_metadata = order_type 0x6 (10b) || nonce (32b) ||
   ↪ expiration_timestamp (32b)
331 +//      || n_give (12b) || n_receive (12b) || n_third_party_receive
   ↪  (12b) || n_conditions (12b)
332 +//      || system_id (126b) || padding (3)
```

```
360 +// w1 = h(h(h(condition[0], condition[1]), ...), condition[N])
361 +// w2 = h(h(h(w1, assets[0]), ...), assets[N])
362 +// w3 = h(h(h(w2, third_party_keys[0]), ...), third_party_keys[N])
363 +// w4 = h(h(h(w3, packed_vaults_and_amounts[0]), ...),
   ↪ packed_vaults_and_amounts[N])
364 +// w5 = h(h(h(w4, packed_third_party_indices[0]), ...),
   ↪ packed_third_party_indices[N])
365 +// w6 = h(w5, packed_metadata)
```

7 Moderate Issues

CVF-4. FIXED

- **Category** Overflow/Underflow
- **Source** general_config.cairo

Description Overflow is possible here.

Recommendation Consider either implementing some sort of overflow protection or clearly explaining why overflow won't happen here.

```
60 +amount = abs_balance * price;
```

CVF-5. FIXED

- **Category** Flaw
- **Source** signature_message_hashes.cairo

Description There is no range check to the "input" argument.

Recommendation Consider adding such a check or explaining why an "input" may never exceed 12 bits.

Client Comment *The update for CVF 3 adds the relevant assumption to 'multi_asset_order_hash'.*

```
205 +func pack_third_party_indices_inner(src: felt*, len, input) -> (
    ↪ packed_message: felt) {
```

CVF-6. FIXED

- **Category** Overflow/Underflow
- **Source** signature_message_hashes.cairo

Description Overflow is possible here.

```
210 +let msg = input * MULTI_ASSET_ORDER_LIST_FIELD_SIZE_UPPER_BOUND +
    ↪ src[0];
```

CVF-7. FIXED

- **Category** Unclear behavior
- **Source** signature_message_hashes.cairo

Description There is no check for the actual number of indices.

Recommendation Consider adding such a check.

```
215 +// Used to pack a single felt with up to 20 indices.
```

8 Minor Issues

CVF-9. FIXED

- **Category** Suboptimal
- **Source** general_config.cairo

Recommendation It would be easier to calculate this as: segment - RiskFactorSegment.SIZE

```
74 +let previous_segment = risk_factor_segments + (segment_idx - 1) *
     ↪ RiskFactorSegment.SIZE;
```

CVF-10. FIXED

- **Category** Bad datatype
- **Source** signature_message_hashes.cairo

Recommendation The value “2 ** 251” should be a named constant.

```
77 +const PADDING = 2 ** 251 / (
```

```
158 +const PADDING = 2 ** 251 / (
```

```
432 +const PADDING = 2 ** 251 / (
```

ABDK
Consulting

About us

Established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function.

The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

Contact

Email

dmitry@abdkconsulting.com

Website

abdk.consulting

Twitter

twitter.com/ABDKconsulting

LinkedIn

linkedin.com/company/abdk-consulting