Report

v. 1.0

Customer

Enso

Smart Contract Audit

Enso-Weiroll

9th December 2022

Report prepared by

ABDK Consulting

Contents

1 Changelog **3**

2 Introduction **4**

3 Project scope **5**

4 Methodology **6**

5 Our findings **7**

6 Major Issues **8**
CVF-4. FIXED 8

7 Moderate Issues **9**
CVF-9. INFO 9

8 Minor Issues **10**
CVF-1. INFO 10
CVF-2. FIXED 10
CVF-3. FIXED 11
CVF-5. FIXED 12
CVF-6. FIXED 12
CVF-7. FIXED 13
CVF-8. FIXED 13

1 Changelog

#	Date	Author	Description
0.1	9.12.22	A. Zveryanskaya	Initial Draft
0.2	9.12.22	A. Zveryanskaya	Minor revision
1.0	9.12.22	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.

Enso is the investment app where friends invest and build strategies for each other.

3 Project scope

We were asked to review:

- Original Repository
- Fix Repository

Files:

contracts/	
VM.sol	CommandBuilder.sol

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.

5 Our findings

We found 1 major, and a few less important issues. All identified Major issues have been fixed.

	Active	Fixed
Major	0	1
Moderate	1	0
Minor	1	6

Fixed 7 out of 9 issues

6 Major Issues

CVF-4. FIXED

- **Category** Suboptimal
- **Source** CommandBuilder.sol

Description The ”count” variable is increased by 32 in all execution branches.

Recommendation Consider increasing in one place after the conditional statement.

```
98  +    count += 32;
107 +    count += 32;
116 +    count += 32;
137 +    count += 32;
149 +count += 32;
```

7 Moderate Issues

CVF-9. INFO

- **Category** Unclear behavior
- **Source** CommandBuilder.sol

Description There is no check to ensure that the length encoded here matches the actual number of array elements.

Recommendation Consider either adding such check, or exclude the array length from the state and just derive is as the number of elements between array start and array end.

Client Comment *This actually is checked during the initial loop to determine the length of the encoded value. It is checked on line 180 inside 'setupDynamicArray()'. I feel like subsequent checks would be unnecessary and increase gas consumption. However, I can make an additional comment in the code indicating such to reduce confusion*

```
271 +mstore(add(add(ret, 36), free), mload(add(stateVar, 32)))
```

8 Minor Issues

CVF-1. INFO

- **Category** Bad datatype
- **Source** CommandBuilder.sol

Description The type of this variable should be: uint256[10].

Client Comment *The suggested data type increases gas consumption. Unless there is another reason for changing it, we'll stick with the cheaper datatype*

```
25 +uint256[] memory offsets = new uint256[](10); // Optionally store
      ↪ the length of all dynamic types (a command cannot fit more
      ↪ than 10 dynamic types)
```

CVF-2. FIXED

- **Category** Readability
- **Source** CommandBuilder.sol

Description The name is very confusing. This array names "offsets" actually stores the lengths of dynamic values.

Client Comment *Name changed to 'dynamicLengths'*

```
25 +uint256[] memory offsets = new uint256[](10); // Optionally store
      ↪ the length of all dynamic types (a command cannot fit more
      ↪ than 10 dynamic types)
```

CVF-3. FIXED

- **Category** Suboptimal
- **Source** CommandBuilder.sol

Description The expression ret + 36 + count is calculated for every field.

Recommendation Consider initializing a pointer before the loop pointing to ret + 36 and increasing this pointer by 32 on every loop iteration.

Client Comment *We have reduced redundant operation for pointers inside the 'buildInputs()' and 'encodeDynamicTuple() functions. However, since the 'encodeDynamicArray()' function only uses the operation once, it doesn't seem necessary to assign it to a variable*

```
92      mstore(add(add(ret, 36), count), free)
```

```
103 +    mstore(add(add(ret, 36), count), free)
```

```
112 +    mstore(add(add(ret, 36), count), free)
```

```
125     mstore(add(add(ret, 36), count), free)
```

```
146  mstore(add(add(ret, 36), count), mload(add(stateVar, 32)))
```

CVF-5. FIXED

- **Category** Bad naming
- **Source** CommandBuilder.sol

Description The semantics of the returned values is unclear.

Recommendation Consider giving descriptive names to the returned values and/or adding documentation comments.

```
162 +) internal pure returns (uint256) {
```

```
177 +) internal pure returns (uint256) {
```

```
197 +) internal view returns (uint256[] memory, uint256, uint256,
    ↪ uint256) {
```

```
218 +) internal view returns (uint256[] memory, uint256, uint256,
    ↪ uint256) {
```

```
262 +) internal view returns (uint256, uint256, uint256, uint256) {
```

```
292 +) internal view returns (uint256, uint256, uint256, uint256) {
```

CVF-6. FIXED

- **Category** Unclear behavior
- **Source** CommandBuilder.sol

Description Modifying an argument is a bad practice.

Recommendation Consider rewriting as: unchecked { return count + 32; }

```
167 +unchecked {
    +    count += 32;
    +}
170 +return count;
```

CVF-7. FIXED

- **Category** Unclear behavior
- **Source** CommandBuilder.sol

Description Modifying an argument is a bad practice.

Recommendation Consider rewriting as: unchecked { return count + argLen + 32; }

```
184 +unchecked {
    +    count += argLen + 32;
    +}
    +return count;
```

CVF-8. FIXED

- **Category** Suboptimal
- **Source** CommandBuilder.sol

Description This code duplicates code of the "buildInputs" function.

Recommendation Consider refactoring to avoid code duplication.

Client Comment *The code inside 'if (idx == IDX_DYNAMIC_END) {...}' is not duplicate code, so for a fix I de-duplicated the code that follows that 'if' statement and put it into a function called 'setupDynamicType()'*

```
230 +if (idx & IDX_VARIABLE_LENGTH != 0) {
    +    if (idx == IDX_DYNAMIC_END) {
    +        offsets[offsetIdx] = offset;
    +        // Return
    +        return (offsets, nextOffsetIdx, count, i);
    +    } else if (idx == IDX_ARRAY_START) {
    +        (offsets, nextOffsetIdx, count, i) = setupDynamicArray(
       ↪ state, indices, offsets, nextOffsetIdx, count, i);
    +    } else if (idx == IDX_TUPLE_START) {
    +        (offsets, nextOffsetIdx, count, i) = setupDynamicTuple(
       ↪ state, indices, offsets, nextOffsetIdx, count, i);
    +    } else {
240 +        count = setupDynamicVariable(state, count, idx);
    +    }
    +} else {
    +    count = setupStaticVariable(state, count, idx);
     }
```

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