Report

v. 1.0

Customer

Enso

Smart Contract Audit

Shortcut Router

12th November 2023

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	12.11.23	A. Zveryanskaya	Initial Draft
0.2	12.11.23	A. Zveryanskaya	Minor revision
1.0	12.11.23	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.

Enso is financial infrastructure for developers that wish to develop, embed, or utilize DeFi interactions through their application. Whether this may be a wallet, smart wallet, exchange, defi interface aggregator, router, or any other application wishing to interact with the decentralized finance ecosystem. Enso provides the tools to execute and fetch all relevant metadata of DeFi protocols enabling developers to build the next generation of financial applications.

3 Project scope

We were asked to review:

- Original Code
- Code with Fixes

Files:

```
/
        EnsoShortcutRouter.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.

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 3 major, and a few less important issues. All identified Major issues have been fixed.

	Info	Fixed
Major	0	3
Moderate	2	1
Minor	3	6

Fixed 10 out of 15 issues

6 Major Issues

CVF-3. FIXED

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

Description This allows "_ETH" to appear inside the "tokensIn" array several times, but the "amountsIn" values are not summed in such a case, which is inconsistent with how ERC20 tokens are handled.

Recommendation Consider either forbidding "_ETH" to appear more than once, or summing the amounts from such appearances and checking "msg.value" against the sum.

84
```
if (tokenIn == _ETH) {
    ethFlag = true;
    if (msg.value != amountIn) revert WrongValue();
```

CVF-4. FIXED

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

Description It is not checked that all these arrays have the same length.

Recommendation Consider doing that or, even better, pass an array of structs instead of multiple arrays.

Client Comment *While all array lengths are being correctly checked (tokensIn and tokensOut are not expected to match), we have updated the contract to use structs to reduce confusion and save gas.*

133
```
address[] memory tokensIn,
address[] memory tokensOut,
uint256[] memory amountsIn,
uint256[] memory minAmountsOut,
```

138
```
bytes32[] calldata commands,
bytes[] calldata state
```

CVF-6. FIXED

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

Description This error should include the token whose amount is insufficient or the index of such token.

162
```
if (amountOut < minAmountsOut[i]) revert AmountTooLow();
```

7 Moderate Issues

CVF-5. INFO

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

Description The function allows duplicate values in this array, but the corresponding amounts are not summed. This could lead to weird behavior and hide problems.

Recommendation Consider either forbidding duplicate values, e.g. by requiring the array to be sorted, or summing the minimum amounts corresponding to duplicates.

Client Comment *If there are duplicate tokensOut this function may check their balance multiple times and that will cost the user more gas for their mistake. But checking the uniqueness of each token will cost all users more gas on every transaction to protect against those few cases of user error.*

```
134 address[] memory tokensOut,
```

CVF-7. FIXED

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

Description The returned value is ignored.

Recommendation Consider properly checking the returned value or using the ”safeTransferFrom”

```
59 IERC20(tokenIn).transferFrom(msg.sender, address(enso), amountIn);
```

CVF-8. INFO

- **Category** Suboptimal
- **Source** EnsoShortcutRouter.sol

Description Using the receiver's balance before and after in order to determine how much tokens the receiver actually received is unreliable, as some receivers may have callbacks executed on incoming transfers, and these callbacks could forward received tokens or modify the receiver's balance is some other way.

Recommendation A reliable way would be to either count all the transfers performed towards the receiver, or accumulating the output tokens at the router's balance, counting them there, and only then sending to the receiver.

Client Comment *It will not be possible to reliably check balances in either EnsoShortcuts (which runs the weiroll script that transfers funds away) or EnsoShortcutRouter (which should never receive funds in the first place). I don't think it is unreasonable to expect the user to know the ultimate destination of their funds and pass the correct receiver address to this function. Any complex transactions that have funds being distributed to multiple receivers are beyond the scope of this function and any relevant safety checks should be implemented directly in the weiroll script.*

```
113 uint256 balance = tokenOut == _ETH ? receiver.balance : IERC20(
        ↪ tokenOut).balanceOf(receiver);

117     amountOut = receiver.balance - balance;

119     amountOut = IERC20(tokenOut).balanceOf(receiver) - balance;

149     balances[i] = tokenOut == _ETH ? receiver.balance : IERC20(
            ↪ tokenOut).balanceOf(receiver);

158         amountOut = receiver.balance - balances[i];

160         amountOut = IERC20(tokenOut).balanceOf(receiver) - balances[
                ↪ i];
```

8 Minor Issues

CVF-9. FIXED

- **Category** Procedural
- **Source** EnsoShortcutRouter.sol

Recommendation Consider specifying as ”^0.8.0” unless there is something special about this particular version.

```
2 pragma solidity ^0.8.17;
```

CVF-10. INFO

- **Category** Procedural
- **Source** EnsoShortcutRouter.sol

Description We didn’t review this file.

```
6 import { MinimalWallet } from "shortcuts-contracts/wallet/
    ↪ MinimalWallet.sol";
  import { AccessController } from "shortcuts-contracts/access/
    ↪ AccessController.sol";
```

CVF-11. FIXED

- **Category** Procedural
- **Source** EnsoShortcutRouter.sol

Recommendation This contract should be defined in a separate file named ”EnsoShortcuts.sol”.

```
9 contract EnsoShortcuts is VM, MinimalWallet, AccessController {
```

CVF-12. INFO

- **Category** Procedural
- **Source** EnsoShortcutRouter.sol

Description We didn’t review the function called here and its exact semantics is unclear.

```
13 _setPermission(OWNER_ROLE, owner_, true);
```

CVF-13. FIXED

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

Recommendation The type of this constant should be ”IERC20”.

```
32 address private constant _ETH = address(0
     ↪ xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE);
```

CVF-14. FIXED

- **Category** Suboptimal
- **Source** EnsoShortcutRouter.sol

Recommendation These errors could be made more useful by adding certain parameters into them.

```
36 error WrongValue();
   error AmountTooLow();
   error ArrayMismatch();
```

CVF-15. FIXED

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

Recommendation The type of this argument should be ”IERC20”.

```
50 address tokenIn,
```

CVF-16. INFO

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

Description This allows the same ”tokenIn” to be transferred several times. Probably not an issue, but could hide problems. A single way to forbid this, is to require the ”tokensIn” values to be sorted.

Client Comment *It is understood that restricting duplicate tokens may prevent a token from being transferred several times. However, there will always be the risk of a user transferring incorrect tokens or amounts for the weiroll script being run. This is a risk that is understood and accepted as it is assumed that since users are trusting Enso to correctly build weiroll commands and state, they will also trust us to pass the correct tokens in and amounts. We prefer to rely upon simulations to validate the correct execution of user intents over relying upon gas-intensive smart contracts to validate the correctness of complex transactions.*

```
88 IERC20(tokenIn).transferFrom(msg.sender, address(enso), amountIn);
```

CVF-17. FIXED

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

Recommendation The type for these arguments should be ”IERC20”.

```
105 address tokenIn,
    address tokenOut,
```

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