Wise Audit

September 2020

By CoinFabrik

Introduction

CoinFabrik was asked to audit the contracts for the WISE project. First we will provide a summary of our discoveries and then we will show the details of our findings.

The *WISE* project is a collection of smart contracts written in Solidity enabling the deployment of an *ERC-20* compliant Token in the Ethereum blockchain. The token has staking and referral capabilities, with Uniswap-provided liquidity.

Summary

The contracts audited are from the WISE repository. The audit is based on the commit *e077145e6d179eadbd2e18d03d6ad24f7bcfa4a0*. They are updated to reflect changes that were performed in later commits.

Contracts

The audited contracts are:

- *LiquidityTransformer.sol*: Main contract implementing investment and payout functionality.
- *Helper.sol*: Helper functions.
- *Declaration.sol*: Declaration of various constants and struct types.
- *Global.sol*: Handling of global variables that are used throughout the project.
- *Snapshot.sol*: Snapshot mechanism implementation for handling inflation.
- *WiseToken.sol*: WISE ERC20 Token.
- *StakingToken.sol*: Handles staking logic.
- *ReferralToken.sol*: Handles referral logic.
- *Timing.sol*: Functions dealing with time logic.
- *LiquidityToken.sol*: Handles liquidity stakes.

Analyses

The following analyses were performed:

- Misuse of the different call methods

- Integer overflow errors
- Division by zero errors
- Outdated version of Solidity compiler
- Front running attacks
- Reentrancy attacks
- Misuse of block timestamps
- Softlock denial of service attacks
- Functions with excessive gas cost
- Missing or misused function qualifiers
- Needlessly complex code and contract interactions
- Poor or nonexistent error handling
- Failure to use a withdrawal pattern
- Insufficient validation of the input parameters
- Incorrect handling of cryptographic signatures

Detailed findings

Severity Classification

Security risks are classified as follows:

- **Critical:** These are issues that we manage to exploit. They compromise the system seriously. They must be fixed **immediately**.
- **Medium:** These are potentially exploitable issues. Even though we did not manage to exploit them or their impact is not clear, they might represent a security risk in the near future. We suggest fixing them **as soon as possible**.
- **Minor:** These issues represent problems that are relatively small or difficult to take advantage of but can be exploited in combination with other issues. These kinds of issues do not block deployments in production environments. They should be taken into account and be fixed **when possible**.
- **Enhancement:** These kinds of findings do not represent a security risk. They are best practices that we suggest to implement.

This classification is summarized in the following table:

SEVERITY	EXPLOITABLE	ROADBLOCK	TO BE FIXED
Critical	Yes	Yes	Immediately
Medium	In the near future	Yes	As soon as possible
Minor	Unlikely	No	Eventually
Enhancement	No	No	Eventually

Issues Found by Severity

Critical severity

No issues of critical severity were found.

Medium severity

No issues of medium severity were found.

Minor severity

No issues of minor severity were found.

Non-security Issues

Wrong minimum WISE supply for August 5th

According to the documentation on the WISE website (https://wisetoken.net/docs#sec-2-2-2), the minimum value for the token supply corresponding to August the 5th is of 3.5M WISE tokens.

In the contract LiquidityTransformer.sol, the daily minimum supply of tokens for each day is defined in the dailyMinSupply mapping, however at the index value of 25

which would correspond to August the 5th the specified value is of 3000000 (3M WISE) instead of 3500000 (3.5M WISE):

```
dailyMinSupply[25] = 3000000;
```

We suggest either changing the value in the contract or modifying the documentation if 3M WISE is the actual intended value for this day.

This issue was fixed in commit *8df791c1..974a*

Enhancements

Reduce gas usage by ordering of fields in struct

The project can slightly reduce its gas usage by changing the ordering of the fields of the *ReferrerLink* struct in *Declaration.sol*

According to the data reported by *eth-gas-reporter* when running test-wise, it can be observed a gas reduction when deploying the contract of 13446 gas, and a further gas reduction of 1154 gas on average in each call to *createStake*.

The code for the new struct is as follows:

```
struct ReferrerLink {
    uint256 rewardAmount;
    uint256 processedDays;
    address staker;
    bytes16 stakeID;
    bool isActive;
}
```

This is a very simple change that has no possible side-effects and allows the Solidity compiler to pack the struct in a more efficient way by using less storage slots.

Unnecessary assembly usage in Helper.sol

In *Helper.sol* the two following functions can be seen:

```
function toBytes(uint256 x) internal pure returns (bytes memory b) {
        b = new bytes(16);
        assembly { mstore(add(b, 16), x) }
}
```

```
function toBytes16(uint256 x) internal pure returns (bytes16 b) {
bytes memory bytesArr = toBytes(x);
assembly {
                b := mload(add(bytesArr, 16))
        }
}
```

They can be joined into a single function without the assembly usage, making it less error-prone and more gas-efficient, the following code could be used instead:

```
function toBytes16(uint256 x) internal pure returns (bytes16 b) {
    return bytes16(bytes32(x));
}
```

Note that we no longer need *toBytes* since it was only used in *toBytes16*.

Observations

Not allowing contracts to be used as referrals

Currently the project checks that a certain address to be used as a referral is not a contract by using the *notContract* function which checks for a codesize of 0.

Anyway, this is not a good enough method for determining whether the address belongs to a contract or not, and can be bypassed.

One solution we propose is to request the referrer for a one-time registration as a valid user by calling an *approveReferrer* function. We can ensure that this is called by a valid person and not by a contract checking that *msg.sender* and *tx.origin* have the same value.

The status of each address, whether it is considered a valid referrer or not, can then be stored in a *mapping* and requested when needed.

Other Analyses

Additionally to those mentioned in the Summary, the following, more specific analyses, were also performed:

Gas Usage

- We attempted to reduce gas consumption by changing certain multiplications and divisions by bit shifts, however since this only helps when working with

powers of 2. This resulted in no gas reduction and in several cases it slightly increased it.
- We attempted to change the ordering in other structs but no improvement was noticed, many of the fields used by the project are 256 bits in size and since that's the maximum size of a slot no further optimization can be made.
- We looked for places where we could pack several variables into one but none that decreased the gas usage was found.

Overflows

- We analyzed certain places where *SafeMath* was not used for possible overflows, but none were found, all of the computations are considered to be safe or to have a negligible probability of overflowing.
- We looked into functions which take array parameters (like *reserveWise*) to see if we could exploit them by sending arrays that were too large, but no possible disruptions were found.

Front-Running

- We checked for the possibility of front-running the Oracle call, but this was not feasible.

Snapshot Mechanism

- We analyzed whether or not a user can take advantage of the snapshot mechanism in order to profit from a revalue in the Wise token price.
- For example: using FlashLoans an user could move the ratio at the moment of taking the snapshot making an unexpected variation of the ratio at that particular block returning to its normal state at the end of the block.

Conclusion

We found that although the project shows a certain complexity, the code well written and security has been taken into account. The documentation provided was also very helpful and relates correctly to what is implemented in the contracts.

No security issues were found and the only non-security issue was quickly fixed by the team.

Disclaimer: This audit report is not a security warranty, investment advice, or an approval of the WISE project since CoinFabrik has not reviewed its platform. Moreover, it does not provide a smart contract code faultlessness guarantee.