Preliminary Comments

Security Assessment

January 24th, 2021

Preliminary Report

For :
Inverse Protocol

By :

Zach Zhou @ CertiK
jun.zhou@certik.org

Rudolph Wang @ CertiK
shaozong.wang@certik.org

Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product’s IT infrastructure and or source code.

Overview

Project Summary

Project Name	Inverse Protocol
Description	a counter-volatility DeFi protocol
Platform	Ethereum; Solidity
Codebase	GitHub Repository
Commit	dbd391ffdbedff9cb40ab40e251d88d94a7320e4

Audit Summary

Delivery Date	Jan. 24th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Jan. 22, 2021 - Jan. 24, 2021

Vulnerability Summary

Total Issues	14
Total Critical	0
Total Major	0
Total Minor	1
Total Informational	11
Total Discussion	2

Executive Summary

This report has been prepared for **Inverse Protocol** smart contract to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic Analysis, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The price when user buy or sell token is set by owner. So the system should only be used if the price is set rightly.

File in Scope

ID	Contract	SHA-256 Checksum
XV	XIV.sol	145a5a36fa9f344ae7381515de484321e33cafd4b4acfcc0c4a0790631dd2a69
XT	XIVStaking.sol	727fbe7d48bea43eca77e0365d65f34654f7794580ecaeb067a3bf5a031dcf91
OW	Ownable.sol	b26ced0e66afb4f896408275e375dccaa2092599bf3d6903f580b933a324444c
SM	SafeMath.sol	f2a8cd7df3406b3935899d99348be3b1502e47ed986f60b408ca599e1323751d

Findings

ID	Title	Type	Severity
XV-01	Old Compiler Version Declaration	Coding Style	Informational
XV-02	Redundant code	Optimization	informational
XV-03	Divisor cannot be zero	Mathermatical Operations	Informational
XV-04	Proper usage of "public" and "external" type	Gas Optimization	Informational
XV-05	Response prompt message in the require function	Optimization	Informational
XT-01	Abnormal initial price	Optimization	Informational
XT-02	State variables that could be declared constant	Gas Optimization	Informational
XT-03	Unused state variables	Optimization	Informational
XT-04	Empty 3rd contract address	Optimization	Discussion
XT-05	Improve the solution to set XIV price	Optimization	Discussion
XT-06	Emit events for some functions	Optimization	Informational
XT-07	Incorrect condition	Logical issue	Minor
XT-08	Error require message	Optimization	Informational
XT-09	Unfinished function	Optimization	Informational

XV-01: Old Compiler Version Declaration

Type	Severity	Location
Optimization	Informational	XIV.sol L1, XIVStaking.sol L1, Ownable.sol L1, SafeMath.sol L1

Description:

`solc` frequently releases new compiler versions. Using an old version prevents access to new Solidity security checks.

Some codes need to be upgraded after changing the solidity version.

Recommendation:

Deploy with any of the following Solidity versions:

- 0.5.11 - 0.5.13,
- 0.5.15 - 0.5.17,
- 0.6.8,
- 0.6.10 - 0.6.11.

Use a simple pragma version that allows any of these versions. Consider using the latest version of Solidity for testing.

XV-02: Redundant code

Type	Severity	Location
Optimization	Informational	XIV.sol L7 L48

Description:

The library `SafeMath` and `Ownable` are redundant.

Recommendation:

Import the `SafeMath.sol` and `Ownable.sol` files.

```
    import "./Ownable.sol";
    import "./SafeMath.sol";
```

XV-03: Divisor cannot be zero

Type	Severity	Location
Mathematical Operations	Informational	XIV.sol L25

Description:

The divisor cannot be zero.

Recommendation:

Call `require` validation in the `div` function.

```
    require(b > 0, errorMessage);
```

XV-04: Proper Usage of "public" and "external" type

Type	Severity	Location
Gas Optimization	Informational	XIV.sol L86 L93 L175 XIVStaking.sol L80 L107 L 125 L142 L148 L158 L177 L189 L201 L214

Description:

"public" functions that are never called by the contract could be declared "external" . When the inputs are arrays "external" functions are more efficient than "public" functions.

Examples

Functions like : `buyTokens` , `sellTokens` , `stakeTokens` , `unStakeTokens`

Recommendation:

Consider using the "external" attribute for functions never called from the contract.

XV-05: Response prompt message in the require function

Type	Severity	Location
Optimization	Informational	XIV.sol L134 L135

Description:

Response prompt message when verification error occurs.

Recommendation:

Response prompt message when verification error occurs.

```
    require(_to != address(0), "error message");
```

XT-01: Abnormal initial price

Type	Severity	Location
Optimization	Informational	XIVStaking.sol L34 L35

Description:

Abnormal initial price.

```
    uint256 XIVPrice=1000000000000000000; //in wei, 18 decimals
    uint256 XIVPriceInUSDT=100000000; // in tokens with decimals
```

Recommendation:

Set the price in the `constructor` function.

XT-02: State variables that could be declared constant

Type	Severity	Location
Gas Optimization	Informational	XIVStaking.sol L36 L37 L39 L40

Description:

Constant state variables should be declared constant to save gas.

Recommendation:

Consider changeing it as following example:

```
address constant XIVTokenContractAddress = 0xe667ee780908cAf92De34d7a749d9ACC1FB702C6;
```

XT-03: Unused state variables

Type	Severity	Location
Optimization	Informational	XIVStaking.sol L39 L40

Description:

The state variables are not used.

```
    address public ethOracleAddress=0x8A753747A1Fa494EC906cE90E9f37563A8AF630e;
    address oracleWrapperContractAddress = 0x8EAb4Ee4C4c4619Fe48C87248369E44380B9E2BF;
```

Recommendation:

Remove the state variables if they are unuseful.

XT-04: Ethereum testnet contract address

Type	Severity	Location
Optimization	Discussion	XIVStaking.sol L36 L37

Description:

We found both of the `XIVTokenContractAddress` and `USDTContractAddress` are ethereum testnet address. Please use the ethereum main net addresses before deployment.

```
address XIVTokenContractAddress = 0xe667ee780908cAf92De34d7a749d9ACC1FB702C6;
address USDTContractAddress = 0xBbf126a88DE8c993BFe67c46Bb333a2eC71bC3fF;
```

XT-05: Improve the way to set XIV price

Type	Severity	Location
Optimization	Discussion	XIVStaking.sol L80

Description:

How does this contract get the prices of every token? And how to set the price with the `setXIVPrice` function? Please describe the solution.

Recommendation:

It's better to get price by UniswapV2 Oracle or other price Oracle.

Example

```
    IUniswapV2Pair constant ethPair =
    IUniswapV2Pair(0xA478c2975Ab1Ea89e8196811F51A7B7Ade33eB11);
     (uint112 reserves0, uint112 reserves1, ) = ethPairDAI.getReserves();
    price = reserves0/reserves1;
```

XT-06: Emit events for some functions

Type	Severity	Location
Optimization	Informational	XIVStaking.sol L106 L125 L142 L148

Description:

`buyTokens`, `sellTokens`, `stakeTokens`, `unStakeTokens` will change users' balance and tokens. These operations are sensitive.

Recommendation:

Emit events for these sensitive operations.

XT-07: Incorrect condition

Type	Severity	Location
Logical issue	Minor	XIVStaking.sol L118

Description:

The customers may don't have enought tokens before they buy tokens.

```
    require(tokenObj.balanceOf(msg.sender) >= amountOfXIV, "You don't have enough XIV balance");
```

Recommendation:

Create a new validation function instead of `checkTokens(amount)` .

XT-08: Error require message

Type	Severity	Location
Optimization	Informational	XIVStaking.sol L151

Description:

```
require(tokenObj.balanceOf(address(this)) >= amount, "You don't have enough XIV balance");
```

This `require` expression verifies that the balance of XIV contract has sufficient balance. Therefore, the prompt information is not accurate. And when will the contract have insufficient balance?

Recommendation:

Change the prompt information to `"XIV contract doesn't have enough balance."`.

XT-09: Unfinished function

Type	Severity	Location
Optimization	Informational	XIVStaking.sol L156

Description:

The function looks like unfinished.

```
    function betForDefi(uint256 amountOfXIV, uint256 typeOfBet, address _betContractAddress)
    public{
            // 0-> defi Fixed, 1->defi flexible, 2-> index Fixed and 3-> index flexible
            checkTokens(amountOfXIV);
            if(typeOfBet==0){
                // defi fixed

            }else if(typeOfBet==1){
                //defi flexible

            }else if(typeOfBet==2){
                //index Fixed
                defiCoinsForFixedIndexMapping[_betContractAddress];

            }else if(typeOfBet==3){
                //index flexible

            }
        }
```

Recommendation:

Remove the redundant code if it's useless.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a `struct` assignment operation affecting an in-memory `struct` rather than an instorage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete` .

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.

Icons explanation

 : Issue resolved

 : Issue not resolved / Acknowledged. The team will be fixing the issues in the own timeframe.

 : Issue partially resolved. Not all instances of an issue was resolved.