XDeFi

Security Assessment

February 10th, 2021

For :
XDeFi

By :

Zach Zhou @ CertiK
jun.zhou@certik.org

Rudolph Wang @ CertiK
shaozhong.wang@certik.org

Dan She @ CertiK
dan.she@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	xdefi-base
Description	Decentralized exchange
Platform	Ethereum; Solidity
Codebase	GitHub Repository
Commit	e9e6c8735f2ec050eb6806d2315dac49e7f4e3d5

Audit Summary

Delivery Date	Feb. 10th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Jan. 27, 2021 - Feb. 4, 2021

Vulnerability Summary

Total Issues	15
Total Critical	1
Total Major	0
Total Minor	2
Total Informational	12

Executive Summary

This report has been prepared for **XDeFi** 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.

Reviews on the correctness of calculations are based on constants set in `XConst.sol` , `XNum.sol` and `XMath.sol` .

File in Scope

ID	Contract	SHA-256 Checksum
IERC	IERC20.sol	20f07b47212ef769e60247eb2118cd4eba7190d8445f735586f88ee9e896bed6
IXC	IXConfig.sol	ebfda3ecd53104924cc8b5d93ea892ec0a74ea7290f76034f41ab2a09a431e8a
IXF	IXFactory.sol	63cd001585f3b2b757d395cdbb115bd109735e3e401d9dcfb652fc78bd5c353e
IXP	IXPool.sol	29ef6b99537de06436e9008eb967626ef5933fc2b18d07b47e92cd178b09cc08
XD	XDEX.sol	e5c4d2de1b0ca5cf7c8017e627f3dcaf30e8b678957b19d67278a8d0bccd8830
AD	Address.sol	620759cc0a8e031499c8d57734a2b16e24e0c7efb9fde8838932c08ce8c1bade
RG	ReentrancyGuard.sol	07fb6f5d455a72c39081d43e8a7860fcc8bfcd569f04d7a7df6cba73dac12ef5
SE	SafeERC20.sol	1ac3d7be166fed68b6f2225508cc7e8bfcd10aa3aba87674520e0b2a10237699
XM	XMath.sol	131c3cd5659b0b98c6fa02f205c5f43bdbda5ae77534380fb9c28872c3fc5072
XN	XNum.sol	384e3e6af54d3c06e0c0aad3cc2e360f980b82e53c8f9d4ee173bf4096c7eb5c
MG	Migrations.sol	9084bb37044827b32e5e48b30c530d32a157e47a6e163b4bcce8882cc2880477
XCG	XConfig.sol	15ee3684b8e5273200ed66d3e9ee358a1733bd3a0e44412c5b0919904ad168bf
XCT	XConst.sol	8d66faf862c25fda5589757bafa805625464b818fabb9dccc8b012a3b6c3b5ad
XF	XFactory.sol	7ddd986be65346c8aba7f0c06041413ff108eb5f9818112fb5e81a3afd6e9b51
XP	XPool.sol	32cd82b6f87ae604df5f233abfa2531f26dd3e9fbea74037917aa0ac725b2c3e
XPC	XPoolCreator.sol	5bd6b9f705e2c64b140b11e87aad423ec11f27e654693183836fb0a99ead69a4
XPT	XPToken.sol	5d475733330fc8e3b9d002a78b55e073cbf788eecce8c3157f51550c8fcbddee
XSP	XSwapProxyV1.sol	885acef08c78d344f3519cea541a6c858c7082d8ea4cb55a51d45e9b33d7dfac
XV	XVersion.sol	ffa213b7f759398359e74dea4d3dd0493a0e0a147f2defc9769e0c763389ea27

Documentation

The sources of truth regarding the operation of the contracts in scope were lackluster and are something we advise to be enriched to aid in the legibility of the codebase as well as project. To help aid our understanding of each contract's functionality we referred to in-line comments and naming conventions.

These were considered the specification, and when discrepancies arose with the actual code behaviour, we consulted with the **XDeFi** team or reported an issue.

Review Notes

Certain optimization steps that we pinpointed in the source code mostly referred to coding standards and inefficiencies, however 1 critical vulnerability and 2 Minor vulnerabilities were identified during our audit that solely concerns the specification.

Certain discrepancies between the expected specification and the implementation of it were identified and were relayed to the team, however they pose no type of vulnerability and concern an optional code path that was unaccounted for.

The project has adequate doumentation and specification outside of the source files, and the code comment coverage is good.

Recommendations

Overall, the codebase of the contracts should be refactored to assimilate the findings of this report, enforce linters and / or coding styles as well as correct any spelling errors and mistakes that appear throughout the code to achieve a high standard of code quality and security.

Findings

ID	Title	Type	Severity	Resolved
IXC-01	Incorrect Naming Convention Utilization	Coding Style	Informational	
XCG-01	Value of `XDEX` Is Not Correct	Optimization	Informational	
XCG-02	SAFU's Governance Plan	Optimization	Informational	
XM-01	Incorrect Simplification of Division Operation	Mathematical Operations	Minor	
XN-01	Numerical Method in `bpowApprox`	Mathematical Operations	Informational	
XP-01	Simplifying Existing Code	Optimization	Informational	
XP-02	Risk of Flash Loan Attacks	Optimization	Critical	
XP-03	Missing Check for Parameters	Gas Optimization	Informational	
XP-04	Problem of `MIN_POOL_AMOUNT`	Optimization	Informational	
XPT-01	State Variables That Could Be Declared Constant	Gas Optimization	Informational	
XSP-01	Proper Usage of “public” And “external” Type	Gas Optimization	Informational	
XSP-02	Missing Emit Events	Optimization	Informational	
XSP-03	Missing Some Important Checks	Logical Issue	Informational	
XSP-04	Approval Wasn't Withdrawn	Optimization	Minor	
XSP-05	State Variable Never Initialized Before Usage	Optimization	Informational	

IXC-01: Incorrect Naming Convention Utilization

Type	Severity	Location
Coding Style	Informational	IXConfig.sol,XFactory.sol,XPool.sol

Description:

Solidity defines a naming convention that should be followed. In general, parameters should use mixedCase, refer to: https://solidity.readthedocs.io/en/v0.6.12/style-guide.html#naming-conventions.

Variables shoud use mixedCase.
Examples:
Variables like: `SWAP_FEES` , `SAFU` ...etc.

Functions other than constructors should use mixedCase.
Examples:
Functions like: `XDEXAddress()` ...etc.

Modifiers should use mixedCase.
Examples:
Modifiers like: `_viewlock_()` ...etc.

Event should use CapWords.
Examples:
Events like: `XConfigINIT_SAFU` , `XConfigSET_CORE` , `XConfigSET_SAFU` , `XConfigSET_SAFU_FEE` , `XConfigADD_POOL_SIG` , `XConfigRM_POOL_SIG` , `XConfigADD_FARM_POOL` , `XConfigRM_FARM_POOL` ...etc.

Recommendation:

The recommendations outlined here are intended to improve the readability, and thus they are not rules, but rather guidelines to try and help convey the most information through the names of things.

Alleviation:

No alleviation.

XCG-01: Value of `XDEX` Is Not Correct

Type	Severity	Location
Optimization	Informational	XConfig.sol L24

Description:

Value of `XDEX` is its address in kovan testnet but not mainnet.

```
    // XDEX Token!
      address public constant XDEX =
          address(0xaDBc525ace6ed9c5195071f29036e7ecCd1DC158); // kovan
```

Recommendation:

Consider changing it to mainnet address.

Alleviation:

The team had removed this test address. Code change was applied in commit b7c02052ea32842c6324c5dc1d44f16d2c85dd9c.

XCG-02: SAFU's Governance Plan

Type	Severity	Location
Optimization	Informational	XConfig.sol

Description:

We noticed that there are certain amount of tokens will be sent to SAFU during swap operations. Core have permission to set its address. Function `collect()` could be used to transfer token‘s balance from XConfig contract to new safu .But if core change safu to a new address again, how are these funds transferred to the new fund pool?

Additionally, what is your SAFU's governance plan? Will governance of SAFU be transferred to community in the furture?

Alleviation:

No alleviation.

(**XDeFi Finance - Response**)

1. SAFU is a multi-sig account.
2. SAFU is the core of `XConfig.sol` contract instance.
3. DEV firstly deploys `XConfig.sol` contract, then setups the `xconfig.core` and `xconfig.safu` with `setSAFU()` and `setCore()` .
4. `setCore()` , `collect()` , `updateSafu()` , `updateFarm()` , `setSwapProxy` , `setSAFU()` of `XConfig` will be transferred to the `Timelock` contract after the Dex is online and runs stably for a period of time. Each operation will be locked for 24 hours.

XM-01: Incorrect Simplification of Division Operation

Type	Severity	Location
Mathematical Operations	Minor	XMath.sol L66~69, L108~111

Description:

Based on `XNum.bdiv`, if `tokenWeightIn == tokenWeightOut` then `weightRatio` (`tokenWeightIn.bdiv(tokenWeightOut)`) should be `1 * BONE` rather than `1`. Also, if `tokenWeightIn >> 1 == tokenWeightOut` (assuming weights are large enough so we can ignore the right most digit) then `weightRatio` should be `2 * BONE` rather than `2`.

However, since `weightRatio` is not directly used when it equals 1 or 2, the results are not changed.

Recommendation:

Consider changing line 66 to 69

```
        if (tokenWeightIn == tokenWeightOut) {
            weightRatio = 1;
        } else if (tokenWeightIn >> 1 == tokenWeightOut) {
            weightRatio = 2;
        } ...
```

to

```
        if (tokenWeightIn == tokenWeightOut) {
            weightRatio = 1 * BONE;
        } else if (tokenWeightIn >> 1 == tokenWeightOut) {
            weightRatio = 2 * BONE;
        } ...
```

Consider changing line 108 to 111

```
        if (tokenWeightOut == tokenWeightIn) {
            weightRatio = 1;
        } else if (tokenWeightOut >> 1 == tokenWeightIn) {
            weightRatio = 2;
        } ...
```

to

```
        if (tokenWeightOut == tokenWeightIn) {
            weightRatio = 1 * BONE;
        } else if (tokenWeightOut >> 1 == tokenWeightIn) {
            weightRatio = 2 * BONE;
        } ...
```

Alleviation:

The team heeded our advice and added precision to `weightRatio`.
The recommendations were applied in commit b5017abb60bcab8ac18700cdc3ee21f3d4b18462.

XN-01: Numerical Method in `bpowApprox`

Type	Severity	Location
Mathematical Operations	Informational	XNum.sol L94~127

Description:

`bpowApprox` calculates power by Taylor series:

$$(1+x)^a = \sum_k \frac{\prod_{i=1}^{k}(a-i+1)x^k}{k!}$$

The formula described in the comment

```
            //         = (product(a - i - 1, i=1-->k) * x^k) / (k!)
```

is misleading. It should be

```
            //         = (product(a - i + 1, i=1-->k) * x^k) / (k!)
```

Also, results are not guaranteed to be accurate at the given precision. The iteration is terminated when `term(i)` is smaller than `precision`. However, it is possible that the sum of remaining `term(i)` s is greater than `precision` and thus leads to small (about 10^{-8} when it is called in `bpow`) rounding errors at the given `precision`.

Alleviation:

The team had correct the comment which is applied in commit b5017abb60bcab8ac18700cdc3ee21f3d4b18462.

XP-01: Simplifying Existing Code

Type	Severity	Location
Optimization	Informational	XPool.sol

Description:

There are some identical checks like below. Extract them into modifiers looks more concise.

```
    require(msg.sender == controller, "ERR_NOT_CONTROLLER");
```

Recommendation:

Consider adding modifier like below. The same to other validations like `finalized` etc.

```
    modifier onlyController(){
        require(msg.sender == controller, "ERR_NOT_CONTROLLER");
        _;
    }
```

Alleviation:

No alleviation.

XP-02: Risk of Flash Loan Attacks

Type	Severity	Location
Optimization	Critical	XPool.sol L274, L397

Description:

Balancer was attacked 3 times in 2020. Hackers use flash loans to carry out attacks.

Use the first attack as an example:

Steps of attack STA pool.

1.Hacker borrow WETH from dYdX.

2.Hacker keep calling `swapExactAmountIn()` to minimize the number of STA, thereby pushing up the price of STA.

3.Swap WETH with STA(1e-18), since price of STA is very high, so a little STA can swap a lots WETH.

Since STA is a deflationary currency, certain amount of STA will be burned during `transfer()`. According to the contract of STA token, decimals will be round up. Therefore, when hacker transfer 1e-18 STA to Balancer, this token will be burned so that balance of STA in Balancer stays at 1e-18.

4.Then Hacker call function `gulp()` to reset value of `_record[STA]` with balance of STA in Balancer(1e-18). Therefore, STA can keep high price all the time.

```
    function gulp(address token) external _logs_ _lock_ {
            require(_records[token].bound, "ERR_NOT_BOUND");
            _records[token].balance = IERC20(token).balanceOf(address(this));
    }
```

5.Repeat step 3, 4. Then, hacker obtained a lots WETH.

6.Hacker repay flash loan.

Recommendation:

Consider creating a whitelist and blacklist for tokens. Put any tokens knowingly exhibiting incorrect behavior into blacklist, and put tokens knowingly exhibiting correct behavior into whitelist. Warning any token that is neither whitelisted nor blacklisted since its smart contract has not been vetted for compatibility with XDEX.

Alleviation:

(**XDeFi Finance - Response**)

Whitelists and blacklists (mainly deflation tokens) are set up on the front-end. For tokens outside the whitelist, the front-end will display risk warnings; for the blacklisted tokens, the front-end will not process them.

XP-03: Missing Check for Parameters

Type	Severity	Location
Gas Optimization	Informational	XPool.sol

Description:

Some function arguments may be zero, which is meaningless. For example:

```
    function swapExactAmountIn(
            address tokenIn,
            uint256 tokenAmountIn,
            address tokenOut,
            uint256 minAmountOut,
            uint256 maxPrice
        ) external returns (uint256 tokenAmountOut, uint256 spotPriceAfter) {
          ......
        }
```

If `tokenAmountIn` is zero, execute this function will waste gas and the result is meaningless. Besides, `tokenIn` and `tokenOut` should not be the same.

Recommendation:

Consider adding below checks in the first line of related function. For example:

```
    function swapExactAmountIn(
            address tokenIn,
            uint256 tokenAmountIn,
            address tokenOut,
            uint256 minAmountOut,
            uint256 maxPrice
        ) external returns (uint256 tokenAmountOut, uint256 spotPriceAfter) {
          require(tokenAmountIn > 0, "zero!");
          require(tokenIn != tokenOut, "same token!");
          ......
        }
```

Alleviation:

No alleviation.

XP-04: Problem of `MIN_POOL_AMOUNT`

Type	Severity	Location
Disscussion	Informational	XPool.sol L352, L650

Description:

We noticed that when users obtained `XPT` by joining pool, there is no check for its amount (`poolAmountOut`). For example, liquidity provider can obtained 10**7(`BONE/(10**11)`) `XPT` which is less than `MIN_POOL_AMOUNT` . But when they exit pool, they cannot get tokens which they had put into the pool since `exitPool()` and `exitswapPoolAmountIn()` require `poolAmountIn` greater than `MIN_POOL_AMOUNT` . Is this case exist in your project? Is this meet your requirement?

Alleviation:

No alleviation.

(**XDeFi Finance - Response**)

For this case, frontend will prompt message to user.

XPT-01: State Variables That Could Be Declared Constant

Type	Severity	Location
Gas Optimization	Informational	XPToken.sol L46~48

Description:

Constant state variables should be declared constant to save gas.

Recommendation:

Consider changing it as following example:

```
    string private constant _name = "XDeFi Pool Token";
    string private constant _symbol = "XPT";
    uint8 private constant _decimals = 18;
```

Alleviation:

The team heeded our advice and declared variables which won't be changed in the contract as constant variables.
The recommendations were applied in commit 37d5b4c920889d145613ed52550435842ec5829c.

XSP-01: Proper Usage of "public" And "external" Type

Type	Severity	Location
Gas Optimization	Informational	XSwapProxyV1.sol, XMath.sol, XPToken.sol

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 `batchSwapExactIn()` , `batchSwapExactOut()` , `multihopBatchSwapExactIn()` , `multihopBatchSwapExactOut()` in contract `XSwapProxyV1.sol` .

`calcSpotPrice()` , `calcOutGivenIn()` , `calcInGivenOut()` , `calcPoolOutGivenSingleIn()` , `calcSingleOutGivenPoolIn()` in contract `XMath.sol` .

`name()` , `symbol()` , `decimals()` , `allowance()` , `balanceOf()` , `totalSupply()` , `approve()` , `transfer()` , `transferFrom()` in contract `XPToken.sol` .

Recommendation:

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

Alleviation:

The team heeded our advice and modified part of the code.The recommendations were applied in commit 31df48297aab46a050a5890beea9aac09549174e.

XSP-02: Missing Emit Events

Type	Severity	Location
Optimization	Informational	XSwapProxyV1.sol,XConfig.sol,XPool.sol,XFactory.sol

Description:

Several sensitive actions are defined without event declarations.

Examples:
Functions like : `setMaxExitFee()` , `setSwapProxy()` in contract `XConfig.sol` .

`constructor()` in contract `XFactory.sol` .

`setController()` , `updateFarm()` , `gulp()` in contract `XPool.sol` .

`batchSwapExactInRefer()` , `batchSwapExactOutRefer()` , `multihopBatchSwapExactInRefer()` , `multihopBatchSwapExactOutRefer()` , `create()` , `joinPool()` , `joinswapExternAmountIn()` , `transferAll()` , `transferFromAllTo()` , `transferFromAllAndApprove()` in contract `XSwapProxyV1.sol` .

Recommendation:

Consider adding events for sensitive actions, and emit it in the function like below.

```
        event SET_CONTROLLER(address indexed manager);
        function setController(address manager) external _logs_ {
            require(msg.sender == controller, "ERR_NOT_CONTROLLER");
            controller = manager;
            emit SET_CONTROLLER(manager);
        }
```

Alleviation:

The team heeded our advice and modified part of the code.The recommendations were applied in commit f663b5888ba3aabe592fce42de27ef6160294644.

XSP-03: Missing Some Important Checks

Type	Severity	Location
Logical Issue	Informational	XSwapProxyV1.sol,Migrations.sol,XConfig.sol,XFactory.sol,XPool.sol,XPToken.sol, XDEX.sol

Description:

Functions `upgrade()` in contract `Migrations.sol` . `setCore()` in contract `XDEX.sol` . `constructor()` in contract `XFactory.sol` . `constructor()` , `setController()` , `bind()` in contract `XPool.sol` . `_move()` , `approve()` , `transferFrom()` in contract `XPToken.sol` . `setSAFU()` , `isFarmPool()` , `collect()` in contract `XConfig.sol` . `constructor()` , `batchSwapExactInRefer()` , `batchSwapExactOutRefer()` , `multihopBatchSwapExactInRefer()` , `multihopBatchSwapExactOutRefer()` , `create()` , `joinPool()` , `joinswapExternAmountIn()` , `getBalance()` , `transferAll()` , `transferFromAllTo()` , `transferFromAllAndApprove()` in contract `XSwapProxyV1.sol` .

All of them are missing address zero checks.

Recommendation:

Consider adding zero address check, for example:

```
    function setSAFU(address _safu) external onlyCore {
            require(_safu != address(0), "ERR_ZERO_ADDR");
            emit SET_SAFU(safu, _safu);
            safu = _safu;
    }
```

Alleviation:

The team heeded our advice and modified part of the code.The recommendations were applied in commit f663b5888ba3aabe592fce42de27ef6160294644.

XSP-04: Approval Wasn't Withdrawn

Type	Severity	Location
Optimization	Minor	XSwapProxyV1.sol L111, L174, L260, L329, L578

Description:

`swap.pool` obtained the approval to spend `MAX` amount of `XSwapProxyV1.sol`. But in the end this approval was not withdrawn. The same to `transferFromAllAndApprove()`.

```
    if (TI.allowance(address(this), swap.pool) < totalAmountIn) {
                    TI.safeApprove(swap.pool, 0);
                    TI.safeApprove(swap.pool, MAX);
    }
```

Recommendation:

Consider withdrawing approval in the end. For example:

```
        for (uint256 i = 0; i < swaps.length; i++) {
            Swap memory swap = swaps[i];
            IXPool pool = IXPool(swap.pool);

            if (TI.allowance(address(this), swap.pool) < totalAmountIn) {
                TI.safeApprove(swap.pool, 0);
                TI.safeApprove(swap.pool, MAX);
            }

           (uint256 tokenAmountOut, ) =
              pool.swapExactAmountInRefer(
              tokenIn,
              swap.tokenInParam,
              tokenOut,
              swap.tokenOutParam,
              swap.maxPrice,
              referrer
            );
            TI.safeApprove(swap.pool, 0);
            actualTotalIn = actualTotalIn.badd(swap.tokenInParam);
            totalAmountOut = tokenAmountOut.badd(totalAmountOut);
        }
```

Alleviation:

No alleviation.

(**XDeFi Finance - Response**)

`swap.pool` can't take the initiative to spend `XSwapProxyV1` 's money.

XSP-05: State Variable Never Initialized Before Usage

Type	Severity	Location
Optimization	Informational	XSwapProxyV1.sol

Description:

Variable `tokenAmountInFirstSwap` , `intermediateTokenAmount` does not initialize before usage.
It is used in `multihopBatchSwapExactOutRefer()` in contract `XSwapProxyV1.sol` .

Recommendation:

Consider initializing all the variables. If a variable is meant to be initialized to zero, explicitly set it to zero.

Alleviation:

The team heeded our advice and modified part of the code.The recommendations were applied in commit f663b5888ba3aabe592fce42de27ef6160294644.

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.