DEFIYIELD

TECHNICAL ANALYSIS FOR BISHARES.FINANCE

CONTENT

1. ABOUT DEFIYIELD

DeFiYield is one of the leading smart contract auditing providers focused on checking security of yield farming projects and the world's only DeFi cross-chain asset management protocol based on machine learning techniques.

Our first audits were conducted back in July 2020, shortly after the yield farming industry boomed, bringing impressive return opportunities for users. At the same time, scams happened every day and users were not protected against them in any way. No one performed yield-farming-focused audits at the time and a lot of projects were launching without even doing proper internal audits. This is why DeFiYield took the lead and has been developing and pushing security standards in the community since then.

2. PROJECT SUMMARY

Project Name	Bishares.finance
Blockchain	Binance Smart Chain
Language	Solidity
Scope	BiShares is a DeFi platform for investing in index funds developed on Binance Smart Chain. Index funds allow users to diversify their investments by investing in multiple projects only buying one asset (Index token).

3. EXECUTIVE SUMMARY

Six smart contracts were analysed to check the availability of code vulnerabilities:

Solidity

- IndexPool.sol
- MarketCapSqrtController.sol
- PoolFactory.sol
- PoolFactoryAccessControl.sol
- PoolInitializer.sol
- UnboundTokenSeller.sol
- BisharesUniswapV2Oracle.sol

Total Issues Found: 14

Severity	Count	Title
Critical	0	-
High	0	-
Medium	3	- Unchecked Return Value
Low	9	- Incorrect Solidity version - Uninitialized local variables
Info	2	- Public function that should be declared external

4. METHODOLOGY

4.1 Smart Contract Code Analysis

4.1.1 Manual Check

DefiYield's system for manual smart contract code auditing is based on experience from analyzing hundreds of malicious and vulnerable smart contracts. The system allows the DefiYield Safe auditors to consistently go through all smart contract elements and their combinations most frequently used to steal user funds.

Issues covered:

Unverified contract

Unlimited minting to a malicious destination

Infinite token supply

Dangerous token migration

Pausing token transfers anytime for unlimited period

Pausing token transfer for limited period (defined in the contract)

Pausing funds withdrawals (Centralized pausing for any funds withdrawals)

Pausing funds withdrawals with emergency withdrawal available

Proxy patterns

Funds lock with centralized control

Unfair token distribution: high % of team rewards

Suspicious functions

Insufficient timelock for important contract changes

Overprivileged EOA-contract-owner

a. The owner can call a function that allows to withdraw all staked in the contract funds to a needed address;
b. The owner can change address of token reward distribution;
c. The owner can change location of staked user funds)
Unrestricted fee setting
a. withdrawal fee can be set up to 100%;
b. user reward fee can be decreased;
c. Team reward increased without any limitations in centralized way;
d. Other protocol fees with unexpected security consequences)

Using a singular exchange as a price source.

4.1.2 Automated Check

Safe by DeFiYield

Safe is a machine learning code scanner designed by DeFiYield for automated smart contract security checks. It focuses on detection of DeFi-specific smart contract vulnerabilities and malicious functions that are the most frequent reasons of rug pulls and hacker attacks.

Technique applied: syntax tree code representation checked against bug patterns.

Issues covered:

1. Unverified contracts unlimited minting to a malicious destination
2. dangerous token migration
3. pausing token transfers anytime for unlimited period

4. pausing token transfer for limited period (defined in the contract)
5. pausing funds withdrawals (centralized pausing for any funds withdrawals)
6. pausing funds withdrawals with emergency withdrawal available
7. proxy patterns
8. funds lock with centralized control.

MythX

Technique applied: Symbolic execution.

Issues covered:

1. Assert Violation
2. Integer Overflow and Underflow
3. Arbitrary Jump with Function Type Variable
4. Write to Arbitrary Storage Location
5. Uninitialized Storage Pointer
6. Outdated Compiler Version
7. Floating Pragma, Unchecked Call Return Value
8. Unprotected Ether Withdrawal
9. Unprotected SELFDESTRUCT Instruction
10. Reentrancy, State Variable Default Visibility
11. Uninitialized Storage Pointer
12. Use of Deprecated Solidity Functions
13. Delegatecall to Untrusted Callee
14. DoS with Failed Call
15. Authorization through tx.origin
16. Block values as a proxy for time
17. Incorrect Constructor Name
18. Shadowing State Variables
19. Weak Sources of Randomness from Chain Attributes
20. Requirement Violation
21. Write to Arbitrary Storage Location
22. DoS With Block Gas Limit
23. Typographical Error

24. Right-To-Left-Override control character (U+202E

Slither

Technique applied: Symbolic execution.

Issues covered:

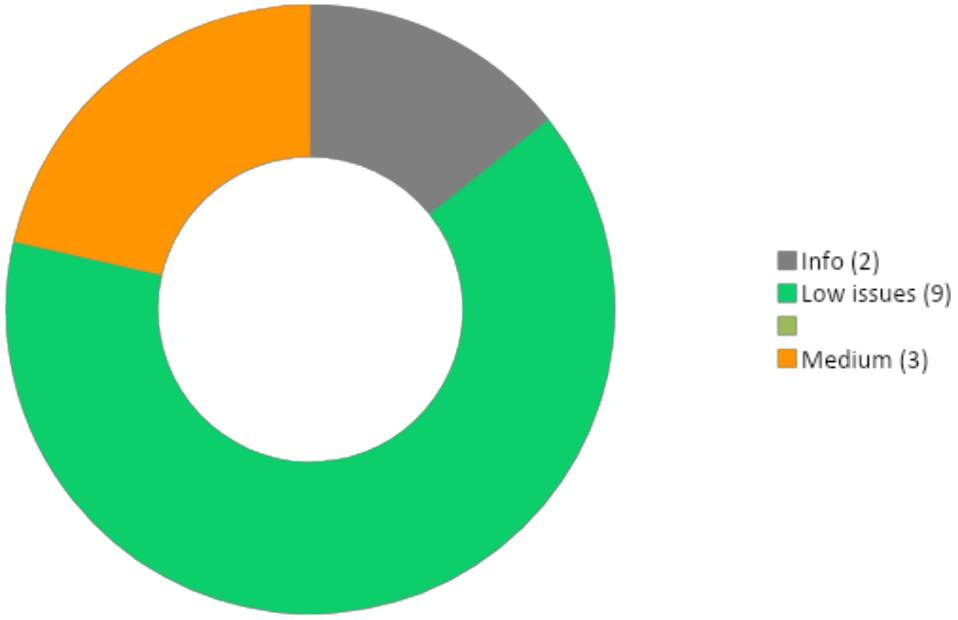
1. Modifying storage array by value
2. The order of parameters in a shift instruction is incorrect
3. Multiple constructor schemes
4. Contract's name reused
5. Public mappings with nested variables
6. Right-To-Left-Override control character is used
7. State variables shadowing
8. Functions allowing anyone to destruct the contract
9. Uninitialized state variables
10. Uninitialized storage variables
11. Unprotected upgradeable contract
12. Functions that send Ether to arbitrary destination
13. Tainted array length assignment
14. Controlled delegatecall destination
15. Reentrancy vulnerabilities (theft of ethers)
16. Signed storage integer array compiler bug
17. Unchecked tokens transfer
18. Weak PRNG
19. Detect dangerous enum conversion
20. Incorrect ERC20 interfaces
21. Incorrect ERC721 interfaces
22. Dangerous strict equalities
23. Contracts that lock ether
24. Deletion on mapping containing a structure
25. State variables shadowing from abstract contracts
26. Tautology or contradiction
27. Unused write

28. Misuse of Boolean constant
29. Constant functions using assembly code
30. Constant functions changing the state
31. Imprecise arithmetic operations order
32. Reentrancy vulnerabilities (no theft of ethers)
33. Reused base constructor
34. Dangerous usage of tx.origin
35. Unchecked low-level calls
36. Unchecked send
37. Uninitialized local variables
38. Unused return values
39. Modifiers that can return the default value
40. Built-in symbol shadowing; Local variables shadowing
41. Uninitialized function pointer calls in constructors
42. Local variables used prior their declaration
43. Constructor called not implemented
44. Multiple calls in a loop
45. Missing Events Access Control
46. Missing Events Arithmetic
47. Dangerous unary expressions
48. Missing Zero Address Validation
49. Benign reentrancy vulnerabilities
50. Reentrancy vulnerabilities leading to out-of-order Events
51. Dangerous usage of block.timestamp
52. Assembly usage
53. Assert state change
54. Comparison to boolean constant
55. Deprecated Solidity Standards
56. Un-indexed ERC20 event parameters
57. Function initializing state variables
58. Low level calls
59. Missing inheritance
60. Conformity to Solidity naming conventions
61. If different pragma directives are used
62. Redundant statements
63. Incorrect Solidity version
64. Unimplemented functions
65. Unused state variables
66. Costly operations in a loop
67. Functions that are not used

68. Reentrancy vulnerabilities through send and transfer
69. Variable names are too similar
70. Conformance to numeric notation best practices
71. State variables that could be declared constant
72. Public function that could be declared externally.

4.1.3 Issue Classification by Severity

Critical	Issues that can directly cause a loss of underlying funds with high probability. These issues must be removed ASAP.
High	There is a possibility of negative impacts on funds managed by the smart contract when certain conditions come into action.
Medium	Issues that affect contract functionality without causing financial losses, must be addressed by the developers.
Low	The issues must be addressed to follow the best SC coding practice.
Info	The issues refer to the best SC coding practice and don't cause any problems with using SCs. Their handling depends on the decision of the dev team.



Total Issues: 14

Related Smart Contract	ID	Issue name	Category	Severity	Status
IndexPool.sol	184	Public function that should be declared external (x2)	Solidity Coding Best Practices	Info	✘ Unresolved
	177	Incorrect Solidity version	Solidity Coding Best Practices	Low	✘ Unresolved
	160	Uninitialized local variables (x2)	Solidity Coding Best Practices	Low	✘ Unresolved
MarketCapSqrtController.sol	104-b	Unchecked Return Value (x2)	Solidity Coding Best Practices	Medium	✘ Unresolved
	177	Incorrect Solidity version	Solidity Coding Best Practices	Low	✘ Unresolved
PoolFactory.sol	177	Incorrect Solidity version	Solidity Coding Best Practices	Low	✘ Unresolved
PoolFactoryAccessControl.sol	177	Incorrect Solidity version	Solidity Coding Best Practices	Low	✘ Unresolved
PoolInitializer.sol	104-b	Unchecked Return Value	Solidity Coding Best Practices	Medium	✘ Unresolved
	177	Incorrect Solidity version	Solidity Coding Best Practices	Low	✘ Unresolved

UnboundTokenSeller.sol	177	Incorrect Solidity version	Solidity Coding Best Practices	Low	✘ Unresolved
BisharesUniswapV2Oracle.sol	177	Incorrect Solidity version	Solidity Coding Best Practices	Low	✘ Unresolved

*Acknowledged — the issue is described to the project team, however the team stated that the function was implemented for any urgent cases to save users' funds.

5. FINDINGS

5.1 Smart Contract Security Analysis

5.1.1 The IndexPool.sol contract

Features	The controller can call the following functions: setMaxPoolTokens setSwapFee delegateCompLikeToken reweighTokens reindexTokens setMinimumBalance

Issues Found

Public function that should be declared external (x2)

Severity: Info

SCW ID: 184

Description: To save gas, all functions that are not used by other functions of the contract should be declared as external.

Location: getUniswapV2oracle(), getUniswapRouter().

Recommendations: The functions listed above should be declared as external.

Incorrect Solidity version

Severity: Low

SCW ID: 177

Description: Avoid using complex pragma statements and do not use old solidity versions.

Location: Version used in inherited contracts: ' ^0.6.0'

Recommendations: Using the latest stable pragma compiler.

Uninitialized local variables (x2)

Severity: Low

SCW ID: 160

Description: Uninitialized local variables.

Location: extrapolatePoolValueFromToken(): extrapolatedValue, token.

Recommendations: Initialize all the variables. If a variable is meant to be initialized to zero, explicitly set it to zero to improve code readability.

5.1.2 The MarketCapSqrtController.sol Contract

Features	The owner can call the following functions: renounceOwnership() transferOwnership() createCategory() addToken() addTokens() removeToken() prepareIndexPool() setDefaultSellerPremium() updateSellerPremium() setMaxPoolTokens() setSwapFee() delegateCompLikeTokenFromPool()

Issues Found

Unchecked Return Value (x2)

Severity: Medium

SCW ID: 104-b

Description: If the return value is not checked, Ignoring its return value might cause unexpected exceptions.

Location: addToken(), addTokens(): updatePrice(token) returns Boolean.

Recommendations: Check return value of the functions before continuing processing.

Incorrect Solidity version

Severity: Low

SCW ID: 177

Description: Avoid using complex pragma statements and do not use old solidity versions.

Location: Version used in inherited contracts: '>=0.6.0<0.8.0', '^0.6.0'.

Recommendations: Using the latest stable pragma compiler.

5.1.3 The PoolFactory.sol Contract

Features	The owner can call the following functions: renounceOwnership() transferOwnership() approvePoolController() disapprovePoolController()

Issues Found

Incorrect Solidity version

Severity: Low

SCW ID: 177

Description: Avoid using complex pragma statements and do not use old solidity versions.

Location: Version used in inherited contracts: '>=0.6.0<0.8.0', '^0.6.0'.

Recommendations: Using the latest stable pragma compiler.

5.1.4 The PoolFactoryAccessControl.sol Contract

Features	The owner can call the following functions: renounceOwnership() transferOwnership() transferPoolFactoryOwnership() grantAdminAccess() revokeAdminAccess() disapprovePoolController() Admin or owner can call the following functions: approvePoolController()

Issues Found

Incorrect Solidity version

Severity: Low

SCW ID: 177

Description: Avoid using complex pragma statements and do not use old solidity versions.

Location: Version used in inherited contracts: '>=0.6.0<0.8.0', '^0.6.0'.

Recommendations: Using the latest stable pragma compiler.

5.1.5 The PoolInitializer.sol Contract

Features	The controller can call the following functions: initialize()

Issues Found

Unchecked Return Value

Severity: Medium

SCW ID: 167-b

Description: If the return value is not checked, Ignoring its return value might cause unexpected exceptions.

Location: updatePrices(): updatePrices(tokens) returns Boolean.

Recommendations: Check return value of the functions before continuing processing.

Incorrect Solidity version

Severity: Low

SCW ID: 177

Description: Avoid using complex pragma statements and do not use old solidity versions.

Location: Version used in inherited contracts: '>=0.6.0<0.8.0', '>=0.6.2<0.8.0', '^0.6.0'.

Recommendations: Using the latest stable pragma compiler.

5.1.6 The UnboundTokenSeller.sol Contract

Features	The controller can call the following functions: setPremiumPercent() initialize()

Issues Found

Incorrect Solidity version

Severity: Low

SCW ID: 177

Description: Avoid using complex pragma statements and do not use old solidity versions.

Location: Version used in inherited contracts: '>=0.6.0<0.8.0', '>=0.6.2', '>=0.6.2<0.8.0', '^0.6.0'

Recommendations: Using the latest stable pragma compiler.

5.1.7 The BisharesUniswapV2Oracle.sol Contract

Features	Realization of UniswapV2 oracle.

Issues Found

Incorrect Solidity version

Severity: Low

SCW ID: 177

Description: Avoid using complex pragma statements and do not use old solidity versions.

Location: Version used in inherited contracts: '>=0.5.0', '^0.6.0'

Recommendations: Using the latest stable pragma compiler.

6. SECURITY SCORING

Metrics	Metric Score	Metric Weight
Static Analysis	84	1

Total Score: 84%

7. CONCLUSION

The audited subset of the project's contracts involves the core contracts of the BiShares.Finance ecosystem, which provides ability to invest in the index token and make user investments more diversified. It's worth mentioning that not all core contracts of BiShares are objects of this audit (the Bison contract was not provided for the review).

The code of the contracts audited is well commented and every function has its own description.

The contracts are well secured and all important functions are protected from reentrancy with _lock_ modifier.

The only issue to point out is that PoolInitializer.sol and MarketCapSqrtController.sol contain functions that are missing the return value check. This could cause an unexpected execution scenario if the functions won't run correctly.

Recommendations:

Checking return value of the functions before continuing processing in PoolInitializer.sol and MarketCapSqrtController.sol.

Using solidity 0.7.6 + OpenZeppelin 3.4-solc-0.7 (stable) OR solidity 0.8.6 + OpenZeppelin 4.2 (latest).

www.defiyield.app