DEFIYIELD

polygon

SECURITY AUDIT FOR IRON.FINANCE

JUNE 17, 2021

www.defiyield.app

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	Iron.finance (on Polygon)
Blockchain	Polygon
Language	Solidity
Scope	Iron.finance is a DeFi multi-chain ecosystem developing on both Binance Smart Chain and Polygon. On Polygon, the project created a partially collateralized algorithmic stablecoin IRON and TITAN (IRON Titanium Token)- the governance, reward and collateral token protocol, which also serves as a collateral for IRON.

3. EXECUTIVE SUMMARY

Twenty smart contracts were analyzed for security vulnerabilities:

Solidity

- Share.sol
- unverified
- Timelock.sol
- TreasuryFund.sol
- Timelock.sol
- Treasury.sol
- ConsolidatedFund.sol
- Dollar.sol
- TreasuryPolicy.sol
- CollateralReserve.sol
- TreasuryVaultAave.sol
- VaultController.sol
- collateralRatioPolicy.sol
- Masterchef.sol (0)
- MasterChef.sol (1)
- MasterChef.sol (2)
- MasterChefFund.sol
- Pool.sol

- **ZapPool.sol**
- **VaultIronLP.sol**

Major EOA holders	None
Minting to malicious destination	None
Migration	Not available
Upgradability through proxy patterns	None
Funds lock period	None
Contract pause	Available for minting and redemption
Suspicious functions	Not found
Unverified contracts	Found

Total Issues Found: 47

Severity	Count	Title
Critical	16	- Unchecked token transfer - Insufficient timelock for important contract changes - Unverified contract - Contracts that lock ether and ERC tokens (x2) - Community rewards are minted to an EOA - TITAN allocation defined in contract variables exceeds planned mint amount of TITAN - Overprivileged role (x7) - Pausing funds withdrawal (x2)
High	2	- Community rewards are minted to an EOA - TITAN allocation defined in contract variables exceeds planned mint amount of TITAN
Medium	1	- Overprivileged role
Low	28	- Floating Pragma (x17) - State variables that should be declared constant (x7) - Presence of unused variables (x2) - Uninitialized local variables (x3)

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)
25. Presence of unused variables

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. They don't cause any issues with SC running. The necessity of handling these issues depends on decision of the dev team.

4.2 Non-Code Security Analysis

Metrics	Features
Documentation	Whitepaper or another tech doc describing the project.
	Completeness and usefulness of code commenting.
	Coverage of SC functions.
	Code requirement traceability.
Team and Development History	Team anonymity / openness.
	Closed / open software repository.
	SCs can be easily found in the project's sources.

	Development history: How old is the repo? How many devs are commiting to the repo and with what frequency?
On-Chain Activity	Use of mixers for money laundering.
	Are the SCs used actively?
Social Media Analysis	Team commits: posts, commenting.
	Community activity: likes, retweets, replies.
Quality Assurance	Testing: 1) Test reports available;
	2) Test scripts and instructions are given.
	Audits done by 3d parties prior to the contract deployment.
Token Distribution	Major EOA holders (>15% tokens per 1 private holder). Automated check.
Governance	Do users participate in governance?
	Governance type.

4.2.1 Documentation

This audit part is focused on quality of smart protocol documentation and code commenting.

The key aspects accessed are:

- Availability of a whitepaper or another technical document describing functionality of the project and the system of it's

smart contracts in detail;

- Completeness and usefulness of code commenting;
- Coverage of SC functions in the documentation;
- Code requirement traceability.

4.2.2 Team and Development History

The project team's positioning, behavior with the community and development history of the project can be very informative points for understanding if the project is open, community-oriented, fast-reacting on external warnings and therefore safe. *In order to answer these questions DeFiYield checks:*

- Team anonymity / openness
- Is the project's software repository closed or open?
- The SCs can be found either on the project's website, or on its GitBook, or in the README of its software repository
- Development history: How old is the repo? How many devs are commiting to the repo and with what frequency?

4.2.3 On-Chain Activity

In this part we check:

- Use of mixers such as Tornado cash for money laundering purposes, which is a common practice of scam projects. They often use mixers to pay for deployment of smart contracts, covering traces in that way.

• Are the SCs used actively? (Number of transactions executed with the SCs in the last 30 days.) If the project's smart contracts have not been used in the last 30 days, the project can be considered abandoned.

4.2.4 Social Media Analysis

Activity of the project team in social media and the way it interacts with users there can say a lot about real intentions and goals of the project. If the team members don't react to questions, requests and warnings of the community, and don't make any content commitments, users should be worried.

When analysing social media of the project we consider:

• *Team commits:* frequency and quality of posts, commenting;

• *Community reactions:* likes, retweets, replies.

4.2.5 Quality Assurance

In this section it's important to check if the project has a full suite of tests, scripts to run them and test reports.

Also, we check if there are audits done by 3d parties prior to deployment of the contracts.

4.2.6 Token Distribution

If there are EOAs holding large shares of the token (one private owner has more than 15% of the token total supply), there is a risk they will sell off their tokens dumping the price.

This feature is checked automatically with the DeFiYield smart contract safety checker.

4.2.7 Governance

A selected governance type reflects the role of the community in the project's decision making processes and the decentralization degree:

- In **off-chain** governance, after a decision is approved on the community level through voting, it must be integrated into the protocol's code by the dev team.

- In **on-chain** systems, governance voting rules are hardcoded into the protocol, meaning any protocol changes being voted for by the community are bridged with the code and performed automatically.

Along with the governance type, we analyze:

- Functions under the governance;
- If the community controls the project's treasury;
- If the governance contract controls other contracts;
- Timelock implementation;
- Timelock length (For user security it must be >24h);
- If the dev team can veto over community decisions;
- History of the governance process: availability of cases when the team ignored the will of the community.

5. FINDINGS

5.1 Smart Contract Security Analysis

5.1.1 The Share.sol contract

Address	0xaAa5B9e6c589642f98a1cDA99B9D024B8407285A
Purpose	The contract of TITAN (IRON Titanium Token) -the protocol's share token. It allows to mint the token for community rewards and treasury fund
Owner	The Timelock contract
Features	*The owner can call the following functions:* **claimCommunityRewards** - mints TITAN to the communityRewardController address - an EOA **setTreasuryAddress** - changes the treasury address **setCommunityRewardController** - changes the communityRewardController address - an EOA, which is supposed to hold TITAN tokens to distribute them to liquidity mining pools. Other protocol's contracts interact with the contract of TITAN through calling: **setTreasuryFund** - is called by the treasury fund contract to change the treasury fund address **claimTreasuryFundRewards** - is called by the treasury fund contract to mint new TITAN in accordance with the treasury fund emissionton rate and frequency.

Pool contracts can interact with the Share contract through calling the **poolMint** - function to mint new Share when users deposit funds and

poolBurnFrom - when they exit the pools.

Issues Found 4

Community rewards are minted to an EOA

Severity: High

SCW ID: -

Description: User rewards are minted to the EOA and get under a centralized control The EOA community controller address sends received minted tokens to an unverified contract

Location: claimCommunityRewards()

Recommendations: create a smart contract to mint the community rewards to, eliminating the centralized control over the funds.

Insufficient timelock for important contract changes

Severity: Critical

SCW ID: 197

Description: The timelock is insufficient for a proper user reaction to changed conditions (in case the new conditions are unsatisfactory for users in terms of investments goals and funds safety)

Location: setTreasuryAddress(), setTreasuryFund()

Recommendations: Extending the timelock delay to a minimum of 24h

TITAN allocation defined in contract variables exceeds planned mint amount of TITAN

Severity: High

SCW ID: -

Description: Iron Finance declares in its documentation that the total planned emission of TITAN is 1B tokens. The actual planned mint amount according to the Share.sol smart contract is 1.3B tokens.

Location:

uint256 public constant COMMUNITY_REWARD_ALLOCATION

uint256 public constant TREASURY_FUND_ALLOCATION

Recommendations: Giving users the correct information about the planned total supply of the TITAN token in the project's documentation and on the website. Deploying a new smart contract with corrected values of the COMMUNITY_REWARD_ALLOCATION and TREASURY_FUND_ALLOCATION variables.

Floating Pragma

Severity: Low

SCW ID: 103

Description: Contracts should be deployed with the same compiler version and flags that they have been tested with. If contracts are accidentally deployed, for example, with an outdated compiler version, bugs can occur, negatively affecting the contract system.

Location: Inherited contracts.

Recommendations: Locking the pragma version and considering known bugs for the compiler version that is chosen.

5.1.2 An unverified contract

Address	0x1F743919dd5EA3d90F985A1D5C7b1719e8d2F59d
Purpose	Unknown
Owner	Unknown
Features	*The contract receives funds from the Community Reward Controller and directs them to other contracts of the protocol (Masterchef, Masterchef) based on unknown logic. It also interacts with other protocol's constructs, including ConsolidatedFund.sol.*

Issues Found 1

Unverified contract

Severity: Critical

SCW ID: 185

Description: Code verification reflects whether the contract bytecode matches with that on the blockchain.

Recommendations: The project must publish the relevant code of its smart contracts and pass code verification.

5.1.3 The Timelock.sol contract

Address	0xb348c6Aa6a6429C6d04eABA739F8a9dC7C50b4De
Purpose	Delaying execution of the Share.sol contract's and the Treasury.sol contract's functions
Owner	EOA
Features	*The contract allows the contract admin to call functions from the Share.sol with the 12-hour delay.*

Issues Found

None

5.1.4 The TreasuryFund.sol contract

Address	0xEA93F3b84df65f73f37fEd8297C0A3C2D6BBb53F
Purpose	The contract calls and receives minting of TITAN dev rewards during the vesting period of 36 months
Owner	The Timelock contract
Features	*The owner can call the following functions:* **claim** - claimTreasuryFundReward minting TITAN to this contracts and burns the excess ratio of unclaimed treasury fund tokens **transfer** - transfers any amount of TITAN located in the contract to a selected address **transferDevFundOwnership** - changes address of the treasury fund for TITAN **setShareAddress** - sets new address of the share token

rescueFund() - transfers any amount of IERC20 tokens located in this contract to the owner (the Timelock contract)

Issues Found 3

Contracts that lock ether and ERC tokens

Severity: Critical

SCW ID: 150

Description: The function sends ERC20 tokens located in the contract to the Timelock contract that doesn't have withdrawal capacities.

Location: rescueFund()

Recommendations: Introduction of the withdraw function

Unchecked token transfer

Severity: Critical

SCW ID: 158

Description: The return value of an external transfer/transferFrom call is not checked

Location: transfer()

Recommendations: Using SafeERC20 or ensure that the transfer/transferFrom return value is checked.

Floating Pragma

Severity: Low

SCW ID: 103

Description: Contracts should be deployed with the same compiler version and flags that they have been tested with. If contracts are accidentally deployed, for example, with an outdated compiler version, bugs can occur, negatively affecting the contract system.

Location: Inherited contracts.

Recommendation: Locking the pragma version and considering known bugs for the compiler version that is chosen

5.1.5 The Timelock.sol contract

Address	0xA6966FDcA5Ed836A27CFBc03D63Aa1eD5fc1119c
Purpose	Delaying execution of the TreasuryFund.sol contract's functions by 48 hours
Owner	EOA
Features	*The contract allows the contract admin to put into the execution queue, execute or cancel execution of functions from the Share.sol with the 48-hour delay.*

Issues Found

None

5.1.6 The Treasury.sol contract

Address	0x376b9e0Abbde0cA068DeFCD8919CA73369124825
Purpose	The contract is an intermediary contract for managing funds between vault, Collateral reserve, Profit Sharing Fund
Owner	The Timelock contract
Features	*The controller - an EOA - can call the following functions:* **recallFromVault** - sends the collateral token from the vault to Collateral reserve **enterVault** - allows to move collateral to the vault **rebalanceVault** - rebalances the vault **rebalanceIfUnderThreshold** - rebalances a selected vault when the collateral reserve is under the threshold (It must be not less than 15%) **extractProfit** - sends profit to the Profit Sharing Fund (the ConsolidatedFund contract) **setVault** - changes the vault address *The contract owner can call the following functions:* **addPool** - adds a pool to the contract **removePool** - removes a pool from the contract **setTreasuryPolicy** - changes the treasury policy **setCollateralRatioPolicy** - changes collateral ratio policy **setController** - changes the controller address **setOracleDollar** - changes the IRON oracle **setOracleShare** - changes the TITAN oracle

setOracleCollateral - changes the collateral token oracle

setCollateralAddress - changes the collateral token

setCollateralReserve - changes the Collateral Reserve address

setProfitSharingFund - changes address of the profit sharing fund

renounceOwnership - removes ownership of this contract

transferOwnership - changes the contract owner.

Issues Found 3

Insufficient timelock for important contract changes

Severity: Critical

SCW ID: 197

Description: The timelock is insufficient for a proper user reaction to changed conditions (in case the new conditions are unsatisfactory for users in terms of investments goals and funds safety)

Location: addPool(), setTreasuryPolicy(), setCollateralRatioPolicy(), setCollateralReserve(), setProfitSharingFund(), transferOwnership()

Recommendations: Extending the timelock delay to a minimum of 24h

Overprivileged role

Severity: Critical

SCW ID: 198-b

Description: The EOA can set any desired address as a vault, which will receive the flow of the collateral token

Location: setVault()

Recommendations: Setting a decentralized contract for the controller Role/ Set the function under a time lock with a minimum 24h delay.

Floating Pragma

Severity: Low

SCW ID: 103

Description: Contracts should be deployed with the same compiler version and flags that they have been tested with. If contracts are accidentally deployed, for example, with an outdated compiler version, bugs can occur, negatively affecting the contract system.

Recommendation: Locking the pragma version and considering known bugs for the compiler version that is chosen.

5.1.7 The ConsolidatedFund.sol contract

Address	0xE07f9242A58f59DC585EEf0620CA88940aA86205
Purpose	The contract received USDC and IRON from the dev wallet and transfers funds when LP rewards are harvested and when funds are withdrawn from pools.
Owner	**The Timelock contract**

Features	**addPool** - adds a new pool to this contract **removePool** - removes a pool from the list of pool addresses **rescueFund** - sends the contract balance to the contract owner **renounceOwnership()** - removes ownership of this contract **transferOwnership()** - changes the contract owner *Pools can call the **transferTo** function to receive any amount of a IERC20 token located on the contract's balance*

Issues Found 3

Contracts that lock ether and ERC tokens

Severity: Critical

SCW ID: 150

Description: The function sends ERC20 tokens located in the contract to the Timelock contract that doesn't have withdrawal capacities.

Location: rescueFund()

Recommendations: Introduction of the withdraw function

Insufficient timelock for important contract changes

Severity: Critical

SCW ID: 197

Description: The timelock is insufficient for a proper user reaction to changed conditions (in case the new conditions are unsatisfactory for users in terms of investments goals and funds safety)

Location: addPool(), transferOwnership()

Recommendations: Extending the timelock delay to a minimum of 24h

Floating Pragma

Severity: Low

SCW ID: 103

Description: Contracts should be deployed with the same compiler version and flags that they have been tested with. If contracts are accidentally deployed, for example, with an outdated compiler version, bugs can occur, negatively affecting the contract system.

Recommendation: Locking the pragma version and considering known bugs for the compiler version that is chosen.

5.1.8 The Dollar.sol contract

Address	0xD86b5923F3AD7b585eD81B448170ae026c65ae9a
Purpose	The contract of the IRON token
Owner	**The Timelock contract**
Features	*The owner of the contract can call the following functions with the 12-hour delay:* **setTreasuryAddress** - changes Treasury **claimCommunityRewards** - mints new TITAN to the community reward controller **setTreasuryFund** - changes Treasury Fund

transferOwnership -changes the contract owner

renounceOwnership - removes the contract ownership

Functions called by pools:

poolMint - mints new TITAN to the caller

poolBurnFrom - burn TITAN from the caller

The Treasury fund can call **claimTreasuryFundRewards** to trigger minting of TITAN for dev rewards

Issues Found 2

Insufficient timelock for important contract changes

Severity: Critical

SCW ID: 197

Description: The timelock is insufficient for a proper user reaction to changed conditions (in case the new conditions are unsatisfactory for users in terms of investments goals and funds safety)

Location: setTreasuryAddress(), transferOwnership()

Recommendations: Extending the timelock to a minimum of 24h. The suggested delay - 48h.

Floating Pragma

Severity: Low

SCW ID: 103

Description: Contracts should be deployed with the same compiler version and flags that they have been tested with. If contracts are accidentally deployed, for example, with an outdated compiler version, bugs can occur, negatively affecting the contract system.

Location: Inherited contracts.

Recommendation: Locking the pragma version and considering known bugs for the compiler version that is chosen

5.1.9 The TreasuryPolicy.sol contract

Address	0x4e370b2b787192D587889B8c7E8AB0d56446e40B
Purpose	The contract manages the main parameters for minting, redemption and collateralization of the protocol's stablecoin
Owner	**EOA**
Features	*The contract owner can call:* **setTreasury** - changes the Treasury address **setExcessCollateralSafetyMargin** - changes the excess collateral safety margin within the set requirements (The minimum is 15%) **setIdleCollateralUtilizationRatio** - changes the idle collateral utilization ratio within the set limitations - it can not exceed 80% **setReservedCollateralThreshold** - changes the reserved collateral threshold within the set requirements (The minimum is 15%)

setMintingFee - changes the minting fee within the set limitations

setRedemptionFee - changes the redemption fee within the set limitations

transferOwnership - changes the contract owner

renounceOwnership - removes ownership of the contract

Issues Found 4

Overprivileged role: the EOA contract owner can change protocol parameters without limitations

Severity: Medium

SCW ID: 198-d

Description: the EOA contract owner can change the protocol's parameters in the centralized way.

Recommendations: transferring the contract ownership to the Timelock contract with the minimum of 24h transaction delay.

Overprivileged role: a privileged EOA can replace addresses inside the smart contract

Severity: Critical

SCW ID: 198-c

Description: The EOA contract owner can replace the Treasury fund address with a malicious contract address.

Location: setTreasury().

Recommendations: renounce ownership of the smart contract/transferring the contract ownership to the Timelock contract with 48h delay.

Floating Pragma

Severity: Low

SCW ID: 103

Description: Contracts should be deployed with the same compiler version and flags that they have been tested with. If contracts are accidentally deployed, for example, with an outdated compiler version, bugs can occur, negatively affecting the contract system.

Location: Inherited contracts.

Recommendation: Locking the pragma version and considering known bugs for the compiler version that is chosen

Presence of unused variables

Severity: Low

SCW ID: 131

Description: Unused variables are allowed in Solidity and they do not pose a direct security issue, but it is can lead to unnecessary gas consumption and generally indicates of poor code quality.

Location: PRICE_PRECISION, RATIO_PRECISION.

Recommendation: Remove unused variables.

5.1.10 The CollateralReserve.sol contract

Address	0xEc12B5d70a84895F819FE037dc4EABDbD24707f2
Purpose	The contract holds the collateral token
Owner	**The Timelock contract**
Features	*The contract owner can call:* **renounceOwnership -** removes the contract ownership **setTreasury -** changes the Treasury address **transferOwnership -** changes the contract owner The Treasury can call the **transferTo** function to transfer and receive funds from the collateral reserve**.**

Issues Found 2

Insufficient timelock for important contract changes

Severity: Critical

SCW ID: 197

Description: The timelock is insufficient for a proper user reaction to changed conditions (in case the new conditions are unsatisfactory for users in terms of investments goals and funds safety)

Location: setTreasuryAddress(), transferOwnership()

Recommendations: Extending the timelock to a minimum of 24h. The suggested delay - 48h.

Floating Pragma

Severity: Low

SCW ID: 103

Description: Contracts should be deployed with the same compiler version and flags that they have been tested with. If contracts are accidentally deployed, for example, with an outdated compiler version, bugs can occur, negatively affecting the contract system.

Location: Inherited contracts.

Recommendation: Locking the pragma version and considering known bugs for the compiler version that is chosen

5.1.11 The TreasuryVaultAave.sol contract

Address	0x21401319caBA905010Ee77A36f87BD176Edc4b96
Purpose	The contract allows the Treasury depositing and withdrawing assets into AaveLendingPool
Owner	**The VaultController contract**
Features	*The contract owner can call:* **claimIncetiveRewards -** sends unclaimed rewards from AaveLendingPool to the contract owner **setTreasury -** changes the Treasury address

	setIncentiveController - changes address of the Incentive Controller **executeTransaction -** executes transaction in a target contract **renounceOwnership -** removes the contract owner **transferOwnerhip -** changes the contract owner

Issues Found 1

Floating Pragma

Severity: Low

SCW ID: 103

Description: Contracts should be deployed with the same compiler version and flags that they have been tested with. If contracts are accidentally deployed, for example, with an outdated compiler version, bugs can occur, negatively affecting the contract system.

Location: Inherited contracts.

Recommendation: Locking the pragma version and considering known bugs for the compiler version that is chosen

5.1.12 The VaultController.sol contract

Address	0xaDe32E79bDF422a83B091E39dd2A26BCE547Fc1b
Purpose	The contract manages transferring funds from TreasuryVaultAave to Collateral Reserve

Owner	**The Timelock contract**
Features	*The contract owner can call:* **claimIncentiveRewards function -** harvests TreasuryVaultAave, transferring funds to Collateral Reserve ***Only the contract owner can call:*** **setAdmin -** changes the admin **setSwapOption -** changes the router and swap path **setCollateralReserve -** changes the Collateral Reserve address **executeTransaction -** executes a transaction in a target contract **renounceOwnership -** removes ownership of the contract **transferOwnership -** changes the contract owner

Issues Found 3

Insufficient timelock for important contract changes

Severity: Critical

SCW ID: 197

Description: The timelock is insufficient for a proper user reaction to changed conditions (in case the new conditions are unsatisfactory for users in terms of investments goals and funds safety)

Location: setCollateralReserve(), transferOwnership()

Recommendations: Extending the timelock to a minimum of 24h. The suggested delay - 48h.

Floating Pragma

Severity: Low

SCW ID: 103

Description: Contracts should be deployed with the same compiler version and flags that they have been tested with. If contracts are accidentally deployed, for example, with an outdated compiler version, bugs can occur, negatively affecting the contract system.

Location: Inherited contracts.

Recommendation: Locking the pragma version and considering known bugs for the compiler version that is chosen

State variables that should be declared constant (x3)

Severity: Low

SCW ID: 183

Description: Constant state variables should be declared constant to save gas.

Location: slippage, usdc, wmatic.

Recommendation: Locking the pragma version and considering known bugs for the compiler version that is chosen

5.1.13 The CollateralRatioPolicy.sol contract

Address	0xDE3BaA1e28740e7fDbdBf65E78efcb3aA994b110
Purpose	The contract allow managing the collateral ratio for IRON
Owner	**EOA**
Features	*The contract owner can call:* **setRatioStep -** changes the ratio step - the amount to change the collateralization ratio by upon refreshCollateralRatio() **setPriceTarget -** changes the price target, which is used for the collateral ratio mechanism **setRefreshCooldown -** changes the refresh cooldown period, which is currently set to 1 hour (3600 seconds) at genesis. **setPriceBand -** changes the price band value **setTreasury -** changes the treasury address **setDollar -** changes address of the Dollar contract(the IRON token contract) **reset -** changes the target collateral ratio and the effective collateral ratio within the set limits **toggleCollateralRatio -** pause and unpause the use of the collateral ratio **toggleEffectiveCollateralRatio -** pause and unpause the use of the effective collateral ratio **setOracleDollar -** changes the oracle for IRON **renounceOwnership -** removes the contract owner **transferOwnership -** changes the the contract owner **executeTransaction -** executes a transaction in a target contract

renounceOwnership - removes ownership of the contract

transferOwnership - changes the contract owner

Issues Found 3

A privileged EOA can replace addresses inside the smart contract (x5)

Severity: Critical

SCW ID: 198-c

Description: The contract owner can replace the treasury, dollar and dollar oracle addresses in a centralized way, which can have negative impact on the protocol work

Location: setTreasury(), setDollar(), setOracleDollar(), setPriceBand(), setPriceTarget()

Recommendations: Transferring ownership of the smart contract to the Timelock contract with a 48-hour delay.

Floating Pragma

Severity: Low

SCW ID: 103

Description: Contracts should be deployed with the same compiler version and flags that they have been tested with. If contracts are accidentally deployed, for example, with an outdated compiler version, bugs can occur, negatively affecting the contract system.

Location: Inherited contracts.

Recommendation: Locking the pragma version and considering known bugs for the compiler version that is chosen

Presence of unused variables

Severity: Low

SCW ID: 131

Description: Unused variables are allowed in Solidity and they do not pose a direct security issue, but it is can lead to unnecessary gas consumption and generally indicates of poor code quality.

Location: RATIO_PRECISION.

Recommendation: Remove unused variables.

5.1.14 The MasterChef.sol contract (0)

Address	0x65430393358e55A658BcdE6FF69AB28cF1CbB77a
Purpose	A farming contract
Owner	**The Timelock contract**
Features	*Users interact with the contract through calling:* **deposit -** deposits LP tokens into the pool **withdraw -** withdraws LP tokens and rewards out of the pool **emergencyWithdraw -** allows to withdraw funds from the pool in emergency cases without getting rewards. ***The contract owner can call:***

setRewardPerBlock - changes the amount of rewards per block

renounceOwnership - removes the contract owner

transferOwnership - changes the contract owner

add - adds a new LP token to the pool

set - changes the given pool's reward allocation point

setMasterChiefFund - changes the MasterchiefFund address

Issues Found 3

Insufficient timelock for important contract changes

Severity: Critical

SCW ID: 197

Description: With the 12h delay, the MasterChiefFund address, currently holding 94% of the TITAN token, can be replaced with a malicious contract.

If the contract ownership will be transferred to an EOA, the latter would be able to change MasterChiefFund any time without any limitations

Location: transferOwnership(), setMasterChiefFund(), setRewardPerBlock()

Recommendations: Extending the timelock to 48h.

Floating Pragma

Severity: Low

SCW ID: 103

Description: Contracts should be deployed with the same compiler version and flags that they have been tested with. If contracts are accidentally deployed, for example, with an outdated compiler version, bugs can occur, negatively affecting the contract system.

Location: Inherited contracts.

Recommendation: Locking the pragma version and considering known bugs for the compiler version that is chosen

State variables that should be declared constant

Severity: Low

SCW ID: 183

Description: Constant state variables should be declared constant to save gas.

Location: BONUS_MULTIPLIER.

5.1.15 The MasterChef.sol contract (1)

Address	0xb444d596273C66Ac269C33c30Fbb245F4ba8A79d
Purpose	A farming contract
Owner	**The Timelock contract**
Features	*Users interact with the contract through calling:* **deposit -** deposits LP tokens into the pool

withdraw - withdraws LP tokens and rewards out of the pool

emergencyWithdraw - allows to withdraw funds from the pool in emergency cases without getting rewards.

The contract owner can call:

setRewardPerBlock - changes the amount of rewards per block

renounceOwnership - removes the contract owner

transferOwnership - changes the contract owner

add - adds a new LP to the pool

set - changes the given pool's reward allocation point

setFund - changes the ConsolidatedFund address

Issues Found 3

Insufficient timelock for important contract changes

Severity: Critical

SCW ID: 197

Description: With the 12h delay, the MasterChiefFund address, currently holding 94% of the TITAN token, can be replaced with a malicious contract.

If the contract ownership will be transferred to an EOA, the latter would be able to change MasterChiefFund any time without any limitations

Location: transferOwnership(), setFund(), setRewardPerBlock()

Recommendations: Extending the timelock to 48h.

Floating Pragma

Severity: Low

SCW ID: 103

Description: Contracts should be deployed with the same compiler version and flags that they have been tested with. If contracts are accidentally deployed, for example, with an outdated compiler version, bugs can occur, negatively affecting the contract system.

Location: Inherited contracts.

Recommendation: Locking the pragma version and considering known bugs for the compiler version that is chosen

State variables that should be declared constant

Severity: Low

SCW ID: 183

Description: Constant state variables should be declared constant to save gas.

Location: BONUS_MULTIPLIER.

5.1.16 The MasterChef.sol contract (2)

Address	0xa37DD1f62661EB18c338f18Cf797cff8b5102d8e
Purpose	A farming contract

Owner	**The Timelock contract**
Features	*Users interact with the contract through calling:* **deposit -** deposits LP tokens into the pool **withdraw -** withdraws LP tokens and rewards out of the pool **emergencyWithdraw -** allows to withdraw funds from the pool in emergency cases without getting rewards. ***The contract owner can call:*** **setRewardPerBlock -** changes the amount of rewards per block **renounceOwnership -** removes the contract owner **transferOwnership -** changes the contract owner **add -** adds a new LP to the pool **set -** changes the given pool's reward allocation point **setFund -** changes the ConsolidatedFund address

Issues Found 3

Insufficient timelock for important contract changes

Severity: Critical

SCW ID: 197

Description: With the 12h delay, the MasterChiefFund address, currently holding 94% of the TITAN token, can be replaced with a malicious contract.

If the contract ownership will be transferred to an EOA, the latter would be able to change MasterChiefFund any time without any limitations

Location: transferOwnership(), setFund(), setRewardPerBlock()

Recommendations: Extending the timelock to 48h.

Floating Pragma

Severity: Low

SCW ID: 103

Description: Contracts should be deployed with the same compiler version and flags that they have been tested with. If contracts are accidentally deployed, for example, with an outdated compiler version, bugs can occur, negatively affecting the contract system.

Location: Inherited contracts.

Recommendation: Locking the pragma version and considering known bugs for the compiler version that is chosen

State variables that should be declared constant

Severity: Low

SCW ID: 183

Description: Constant state variables should be declared constant to save gas.

Location: BONUS_MULTIPLIER.

5.1.17 The MasterChefFund.sol contract

Address	0xf622A4e83ECbcfB7d8cb3007a3C6b03bCdA8666B
Purpose	Currently holding 94% of the TITAN total supply
Owner	**The Timelock contract**
Features	*The contract owner can call:* **addPool -** adds a pool to the contract **removePool -** removes a pool from the contract **renounceOwnership -** removes the contract owner **transferOwnership -** changes the contract owner Pools can call the **transferTo** function, allowing to transfer a needed amount of tokens located in the contract to any specified address.

Issues Found 2

Insufficient timelock for important contract changes

Severity: Critical

SCW ID: 197

Description: The timelock is insufficient for a proper user reaction to changed conditions (in case the new conditions are unsatisfactory for users in terms of investments goals and funds safety)

Location: addPool(), transferTo(), transferOwnership()

Recommendations: Extending the timelock to 48h.

Floating Pragma

Severity: Low

SCW ID: 103

Description: Contracts should be deployed with the same compiler version and flags that they have been tested with. If contracts are accidentally deployed, for example, with an outdated compiler version, bugs can occur, negatively affecting the contract system.

Location: Inherited contracts.

Recommendation: Locking the pragma version and considering known bugs for the compiler version that is chosen

5.1.18 The Pool.sol contract

Address	0xD078B62f8D9f5F69a6e6343e3e1eC9059770B830
Purpose	The contract allows users to mint and redeem IRON
Owner	**The Timelock contract**
Features	*The contract owner can call:* **toggleMinting -** pauses and unpauses minting **toggleRedeeming -** pauses and unpauses redemption **setOracle -** changes the USDC oracle **setRedemptionDelay -** changes number of blocks to wait before calling collectRedemption()

setTreasury - changes the Treasury address

renounceOwnership - removes the contract owner

transferOwnership - changes the contract owner

Issues Found 4

Insufficient timelock for important contract changes

Severity: Critical

SCW ID: 197

Description: The timelock is insufficient for a proper user reaction to changed conditions (in case the new conditions are unsatisfactory for users in terms of investments goals and funds safety)

Location: setTreasury(), transferOwnership()

Recommendations: Extending the timelock to 48h.

Floating Pragma

Severity: Low

SCW ID: 103

Description: Contracts should be deployed with the same compiler version and flags that they have been tested with. If contracts are accidentally deployed, for example, with an outdated compiler version, bugs can occur, negatively affecting the contract system.

Location: Inherited contracts.

Recommendation: Locking the pragma version and considering known bugs for the compiler version that is chosen

Pausing funds withdrawal: Centralized pausing for any funds withdrawals (x2)

Severity: **Critical**

SCW ID: 191

Description: Redemption of IRON can be delayed and paused to an unlimited period

Location: toggleRedeeming(), setRedemptionDelay().

Recommendation: Limiting the maximum redemption pause and delay

Uninitialized local variables (x2)

Severity: **Low**

SCW ID: 160

Description: Uninitialized state variables are a serious vulnerability when used in functions handling funds. For example, if the transfer function is called and the local variable defining the destination is uninitialized, the underlying to amount of crypto will be sent to the address 0x0 and lost.

Location: uint256 _share_amount, uint256 _collateral_amount.

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

5.1.19 The ZapPool.sol contract

Address	0xC7b1F244397e2157036a89CE0D58F3A467A7Ed2F
Purpose	The contract allows users to mint IRON StableCoin.
Owner	**The Timelock contract**
Features	*The contract owner can call:* **renounceOwnership -** removes the contract owner **transferOwnership -** changes the contract owner **toggleMinting() -** pauses and unpauses minting **setSlippage() -** changes slippage (within the set limitations) **setTreasury() -** changes the Treasury address **setOracle() -** changes the oracle address **setRouter(() -** sets the router

Issues Found 2

Insufficient timelock for important contract changes (x

Severity: Critical

SCW ID: 197

Description: The timelock is insufficient for a proper user reaction to changed conditions (in case the new conditions are

unsatisfactory for users in terms of investments goals and funds safety)

Location: setTreasury(), transferOwnership(), setOracle()

Recommendations: Extending the timelock delay to a minimum of 24h. The suggested delay is 48 hours.

Floating Pragma

Severity: Low

SCW ID: 103

Description: Contracts should be deployed with the same compiler version and flags that they have been tested with. If contracts are accidentally deployed, for example, with an outdated compiler version, bugs can occur, negatively affecting the contract system.

Location: Inherited contracts.

Recommendation: Locking the pragma version and considering known bugs for the compiler version that is chosen

5.1.20 The VaultIronLP.sol contract

Address	0xD93f72e8db90Be4b8A24062494c3a86bD0Ac9065 (An example contract. Each user has its own private vault)
Purpose	The private vault contract.
Owner	**User is the owner of their private vault**

Features	
	The available functions are: **deposit -** deposits funds into the vault **updateSlippage -** sets slippage value **compound -** compounds the rewards, can only be called by the harvester *The contract owner can call:* **setHarvestor -** defines the harvester address **rescueFund -** transfers tokens staked in the contract to the owner's address **withdrawAll -** withdraws all the contract balance to the owner's address **withdraw -** withdraws a needed amount of tokens to the contract owner **claimRewards -** transfers rewards earned to the owner's address **abandon -** abandons the vault **syncSwapRoutes**

Issues Found 3

Uninitialized local variables

Severity: Low

SCW ID: 160

Description: Uninitialized state variables are a serious vulnerability when used in functions handling funds. For example, if the transfer function is called and the local variable defining the destination is

uninitialized, the underlying to amount of crypto will be sent to the address 0x0 and lost.

Location: uint256 public swapTimeout

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

Floating Pragma

Severity: Low

SCW ID: 103

Description: Contracts should be deployed with the same compiler version and flags that they have been tested with. If contracts are accidentally deployed, for example, with an outdated compiler version, bugs can occur, negatively affecting the contract system.

Location: Inherited contracts.

Recommendation: Locking the pragma version and considering known bugs for the compiler version that is chosen

State variables that should be declared constant

Severity: Low

SCW ID: 183

Description: Constant state variables should be declared constant to save gas.

Location: uint256 public swapTimeout

5.2 Other Features of the Protocol

Total supply:

IRON: variable

TITAN: The total planned emission declared by the project is 1 billion; the actual possible: 1.3B tokens

Team reward:

30% of TITAN's total supply being vested to the dev team during the 36-month period.

Minting:

TITAN: The tokens are minted for dev rewards and community rewards: 300M will be minted for dev rewards, 1B will be minted for community rewards

IRON: the token is minted by users through collateralization with TITAN and USDC

Migration:

Not available

Proxy patterns:

No proxy patterns were used to enable upgradability of the smart contracts analyzed

Funds lock period:

None

Presale:

None

Pause:

Pausing available in Pool.sol (**toggleMinting**(), **toggleRedeeming** ()) and ZapPool.sol (**toggleMinting**())

Suspicious functions:

Not found

5.3 Non-Code Analysis

5.3.1 Documentation

The general functionality of the project is described in Iron.finance's Gitbook, but the complete presentation of the smart contract system is absent: not all deployed smart contracts are mentioned in the documentation, and there is no explanation to any of the contracts. No SC functions are covered.

Newly deployed contracts are not documented (Private vaults).

Code is poorly commented.

Code requirement traceability is absent.

Recommendations: Documenting all deployed smart contracts and describing their functionality.

5.3.2 Team and Development History

The team is anonymous.

The software repository is open, but doesn't contain all deployed contracts. Moreover, the code of some contracts in the Github repository does not match the code of the deployed contracts.

All contributions are done by one developer.

The roadmap is present but doesn't feature any clear development details.

Recommendations: Publishing all the protocol contracts to Github. Providing a clear Roadmap.

5.3.3 On-Chain Activity

No mixers were used by the project team by deployment of the contracts.

The contracts being analysed are actively used.

5.3.4 Social Media Analysis

The team is active in social media, posts news and updates regularly.

The community is active: likes and comments on twitter and active telegram community.

5.3.5 Quality Assurance

Project was not audited prior to deployment of its contracts

No unit tests and scripts on how to run them are provided.

Recommendations: Provide results and instructions for tests which are available in git repositories.

5.3.6 Token Distribution

There are no major token holders having a significant influence on the token price.

5.3.7 Governance

The project uses off-chain governance: the Snapshot voting.

The governance system is not documented.

Recommendations: Transferring to on-chain governance.

6. SECURITY SCORING

Metrics	Metric Score	Metric Weight
Static Analysis	25	0,5
Documentation	20	0,05
Team and Development History	25	0,05
On-Chain Activity	100	0,15
Social Media Analysis	100	0,05
Quality Assurance	0	0,05
Token Distribution	100	0,1
Governance	20	0,05

Total Score: 45.75%

7. CONCLUSION

The smart contracts analyzed contain a significant amount of smart contracts vulnerabilities, including critical ones, such as presence of an unverified contract and centralized control over underlying funds. The vulnerabilities must be removed as soon as possible.

It's worth pointing out that the whole smart contract system is unjustifiably tangled and must be optimized: in some cases, functionality that could be included into one contract is divided into a variety of separate contracts.

It's also noticeable that some contracts were designed to be under an EOA control: they contain functions that send tokens to the owner, which is currently set to the Timelock contract that doesn't have any withdrawal capacities.

The project's documentation is not informative and not complete: it doesn't cover all the smart contracts deployed. The code commenting covers less than 10% of the code. The Iron.finance's repository doesn't feature all the smart contracts used.

www.defiyield.app