Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: GaiaStarter Protocol
Platform: Astar Network
Language: Solidity
Date: May 11th, 2022

Table of contents

Introduction 4
Project Background 4
Audit Scope 5
Claimed Smart Contract Features 6
Audit Summary 7
Technical Quick Stats 8
Code Quality 9
Documentation 9
Use of Dependencies 9
AS-IS overview 10
Severity Definitions 15
Audit Findings 16
Conclusion 24
Our Methodology 25
Disclaimers 27
Appendix

- Code Flow Diagram 28
- Slither Results Log 34
- Solidity static analysis 39
- Solhint Linter 46

Email: audit@EtherAuthority.io

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

Email: audit@EtherAuthority.io

Introduction

EtherAuthority was contracted by the GaiaStarter team to perform the Security audit of the GaiaStarter Protocol smart contracts code. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on May 11th, 2022.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

The GaiaStarter Contracts have functions like Stake, UnStake, Claim, WithDraw, fund, poolLength, add, set, deposited, pending, totalPending, deposit, burn, mint, AirDrop, snapshot, release, pending, deposited, etc. The GaiaStarter Contracts inherit the ERC20, IERC721, Ownable, SafeERC20, SafeMath, ERC20Burnable, ERC20Snapshot, AccessControl, Pausable, ReentrancyGuard standard smart contracts from the OpenZeppelin library. These OpenZeppelin contracts are considered community-audited and time-tested, and hence are not part of the audit scope.

Audit scope

Name	Code Review and Security Analysis Report for GaiaStarter Protocol Smart Contracts
Platform	**Astar / Solidity**
File 1	NFTStaking.sol
File 1 MD5 Hash	651D758CE14200112A55C1D6BD92EDEF
File 2	MasterChef.sol
File 2 MD5 Hash	7629993B187DD656965F000778BD9400
File 3	Token.sol
File 3 MD5 Hash	CE1686F76467E6D408A18ECBE187447D
File 4	IFO.sol
File 4 MD5 Hash	7BB85D39FACFD583834FE568DF20F05A
File 5	IFOMasterChef.sol
File 5 MD5 Hash	C094A68D2B63748691F29DEFFB759EB6
File 6	Crowdsale.sol
File 6 MD5 Hash	1A7CF0D247CC045BF0D937CC6CA04A51
Audit Date	May 11th, 2022

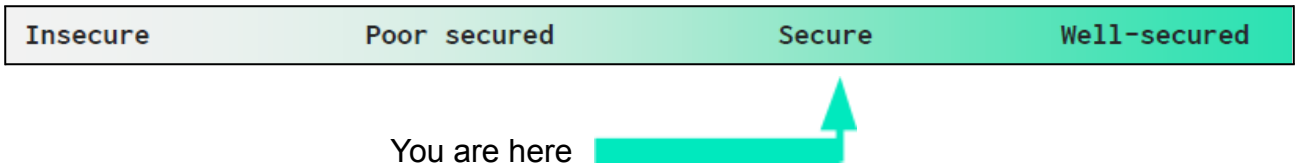
Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
File 1 NFTStaking.sol • Deposit Amount: 5 Sextillion. • Expectation Stake Token CNT: 5000 • Expectation Emission Token CNT: 10,000.	**YES, This is valid.**
File 2 MasterChef.sol • MasterChef has functions like: add, set, fund, poolLength, totalPending, etc.	**YES, This is valid.**
File 3 Token.sol • Token has functions like: snapshot, mint, etc.	**YES, This is valid.**
File 4 IFO.sol • IFO has functions like: setOfferingAmount, setStartTimestamp, deposit,etc.	**YES, This is valid.**
File 5 IFOMasterChef.sol • IFOMasterChef has functions like: fund, poolLength, deposited, etc. • Symbol:	**YES, This is valid.**
File 6 Crowdsale.sol • Crowdsale has functions like: buyTokens, release, releaseAmount, setRate, etc.	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **"Secured"**. This token contract does contain owner control, which does not make it fully decentralized.



We used various tools like Slither, Solhint and Remix IDE. At the same time this finding is based on critical analysis of the manual audit.
All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit overview section. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 1 medium and 3 low and very low level issues.

Investors Advice: Technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Moderated
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Moderated
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Moderated
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Moderated
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Moderated
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Code Quality

This audit scope has 6 smart contract files. Smart contracts contain Libraries, Smart contracts, inherits and Interfaces. This is a compact and well written smart contract.

The libraries in the GaiaStarter Protocol are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties / methods can be reused many times by other contracts in the GaiaStarter Protocol.

The GaiaStarter team has not provided unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are **well** commented on smart contracts.

Documentation

We were given a GaiaStarter Protocol smart contract code in the form of github weblink. The hash of that code is mentioned above in the table.

As mentioned above, code parts are **well** commented. So it is easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Use of Dependencies

As per our observation, the libraries are used in this smart contracts infrastructure that are based on well known industry standard open source projects.

Apart from libraries, its functions are used in external smart contract calls.

AS-IS overview

NFTStaking.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	owner	read	Passed	No Issue
3	onlyOwner	modifier	Passed	No Issue
4	renounceOwnership	write	access only Owner	No Issue
5	transferOwnership	write	access only Owner	No Issue
6	_transferOwnership	internal	Passed	No Issue
7	Stake	external	Passed	No Issue
8	UnStake	external	Passed	No Issue
9	GetClaimableRewardsAll	external	Passed	No Issue
10	GetClaimableRewardsUnit Address	external	Passed	No Issue
11	Claim	external	Passed	No Issue
12	GetStakingAddressList	read	Passed	No Issue
13	GetStakingTokenIdList	read	Passed	No Issue
14	SetdepositNFTToken	external	access only Owner	No Issue
15	SetdepositToken	external	access only Owner	No Issue
16	SetdepositAddress	external	access only Owner	No Issue
17	SetRewordToken	external	access only Owner	No Issue
18	DepositRewordToken	external	access only Owner	No Issue
19	SetStakingEndTime	external	access only Owner	No Issue
20	SetStakingDeadTime	external	access only Owner	No Issue
21	UnStakeByOwner	external	access only Owner	No Issue
22	WithDraw	external	access only Owner	No Issue
23	RewardPerERC20Token	read	Passed	No Issue
24	InsertStakingAddressList	write	Passed	No Issue
25	DeleteAllTokenList	write	Passed	No Issue

MasterChef.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Visibility for constructor is ignored	Refer Audit Findings
2	owner	read	Passed	No Issue
3	onlyOwner	modifier	Passed	No Issue
4	renounceOwnership	write	access only Owner	No Issue
5	transferOwnership	write	access only Owner	No Issue
6	_transferOwnership	internal	Passed	No Issue
7	poolLength	read	Passed	No Issue

8	fund	write	Function input parameters lack of check, Critical operation lacks event log	Refer Audit Findings
9	add	write	Same LPToken can be added more than once, Function input parameters lack of check,Critical operation lacks event log	Refer Audit Findings
10	set	write	Critical operation lacks event log	Refer Audit Findings
11	deposited	external	Passed	No Issue
12	pending	external	Passed	No Issue
13	totalPending	external	Passed	No Issue
14	massUpdatePools	write	Critical operation lacks event log,Infinite loop	Refer Audit Findings
15	updatePool	write	Critical operation lacks event log	Refer Audit Findings
16	deposit	write	Passed	No Issue
17	withdraw	write	Function input parameters lack of check	Refer Audit Findings
18	emergencyWithdraw	write	Passed	No Issue
19	erc20Transfer	internal	Function input parameters lack of check	Refer Audit Findings

Token.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	onlyRole	modifier	Passed	No Issue
3	supportsInterface	read	Passed	No Issue
4	hasRole	read	Passed	No Issue
5	_checkRole	internal	Passed	No Issue
6	getRoleAdmin	read	Passed	No Issue
7	grantRole	write	Passed	No Issue
8	revokeRole	write	Passed	No Issue
9	renounceRole	write	Passed	No Issue
10	_setupRole	internal	Passed	No Issue
11	_setRoleAdmin	internal	Passed	No Issue
12	_grantRole	internal	Passed	No Issue
13	_revokeRole	internal	Passed	No Issue
14	name	read	Passed	No Issue
15	symbol	read	Passed	No Issue

16	decimals	read	Passed	No Issue
17	totalSupply	read	Passed	No Issue
18	balanceOf	read	Passed	No Issue
19	transferallowance	write	Passed	No Issue
20	allowance	read	Passed	No Issue
21	approve	write	Passed	No Issue
22	transferFrom	write	Passed	No Issue
23	increaseAllowance	write	Passed	No Issue
24	decreaseAllowance	write	Passed	No Issue
25	_transfer	internal	Passed	No Issue
26	_mint	internal	Passed	No Issue
27	_burn	internal	Passed	No Issue
28	_approve	internal	Passed	No Issue
29	_spendAllowance	internal	Passed	No Issue
30	_beforeTokenTransfer	internal	Passed	No Issue
31	_afterTokenTransfer	internal	Passed	No Issue
32	burn	write	Passed	No Issue
33	burnFrom	write	Passed	No Issue
34	_snapshot	internal	Passed	No Issue
35	_getCurrentSnapshotId	internal	Passed	No Issue
36	balanceOfAt	read	Passed	No Issue
37	totalSupplyAt	read	Passed	No Issue
38	_beforeTokenTransfer	internal	Passed	No Issue
39	_valueAt	read	Passed	No Issue
40	_updateAccountSnapshot	write	Passed	No Issue
41	_updateTotalSupplySnapshot	write	Passed	No Issue
42	_updateSnapshot	write	Passed	No Issue
43	_lastSnapshotId	read	Passed	No Issue
44	snapshot	write	access only Role	No Issue
45	mint	write	Unlimited minting	Refer Audit Findings
46	AirDrop	write	access only Role	No Issue
47	_beforeTokenTransfer	internal	Passed	No Issue

IFO.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Compiler warnings	Refer Audit Findings
2	owner	read	Passed	No Issue
3	onlyOwner	modifier	Passed	No Issue
4	renounceOwnership	write	access only Owner	No Issue
5	transferOwnership	write	access only Owner	No Issue
6	_transferOwnership	internal	Passed	No Issue

7	setOfferingAmount	write	Function input parameters lack of check	Refer Audit Findings
8	setStartTimestamp	write	access only Owner	No Issue
9	setEndTimestamp	write	access only Owner	No Issue
10	setRasingToken	write	access only Owner	No Issue
11	setOfferingToken	write	access only Owner	No Issue
12	setCrowdsale	write	access only Owner	No Issue
13	setPool	write	access only Owner	No Issue
14	setWhiteList	write	Function input parameters lack of check	Refer Audit Findings
15	deposit	write	Function input parameters lack of check	Refer Audit Findings
16	withdraw	write	Function input parameters lack of check	Refer Audit Findings
17	harvestAndVesting	write	Function input parameters lack of check	Refer Audit Findings
18	hasHarvested	external	Passed	No Issue
19	hasCollateral	external	Passed	No Issue
20	getUserAllocation	read	Passed	No Issue
21	getOfferingAmount	read	Passed	No Issue
22	getTokenAmount	internal	Passed	No Issue
23	getAddressListLength	external	Passed	No Issue
24	finalOfferingTokenWithdraw	write	Function input parameters lack of check	Refer Audit Findings
25	release	write	Passed	No Issue
26	releaseAmount	read	Passed	No Issue
27	setAvailableReleaseFlg	write	access only Owner	No Issue

IFOMasterChef.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	owner	read	Passed	No Issue
3	onlyOwner	modifier	Passed	No Issue
4	renounceOwnership	write	access only Owner	No Issue
5	transferOwnership	write	access only Owner	No Issue
6	_transferOwnership	internal	Passed	No Issue
7	poolLength	external	Passed	No Issue
8	fund	write	Function input parameters lack of	Refer Audit Findings

			check, Critical operation lacks event log	
9	add	write	Function input parameters lack of check, Critical operation lacks event log	Refer Audit Findings
10	set	write	Critical operation lacks event log	Refer Audit Findings
11	deposited	external	Passed	No Issue
12	pending	external	Passed	No Issue
13	totalPending	external	Passed	No Issue
14	massUpdatePools	write	Critical operation lacks event log, Infinite loop	Refer Audit Findings
15	updatePool	write	Critical operation lacks event log	Refer Audit Findings
16	deposit	write	Passed	No Issue
17	claim	write	Passed	No Issue
18	withdraw	write	Function input parameters lack of check	Refer Audit Findings
19	withdrawAll	write	Passed	No Issue
20	erc20Transfer	internal	Function input parameters lack of check	Refer Audit Findings
21	setAvailableClaimFlg	write	access only Owner	No Issue

Crowdsale.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Compiler warnings	Refer Audit Findings
2	owner	read	Passed	No Issue
3	onlyOwner	modifier	Passed	No Issue
4	renounceOwnership	write	access only Owner	No Issue
5	transferOwnership	write	access only Owner	No Issue
6	_transferOwnership	internal	Passed	No Issue
7	fallback	external	Passed	No Issue
8	buyTokens	write	Passed	No Issue
9	release	write	Passed	No Issue
10	releaseAmount	read	Passed	No Issue
11	_preValidatePurchase	internal	Compiler warnings	Refer Audit Findings
12	_postValidatePurchase	internal	Passed	No Issue
13	_deliverTokens	internal	Passed	No Issue
14	_processPurchase	internal	Passed	No Issue
15	_updatePurchasingState	internal	Passed	No Issue

16	_getTokenAmount	internal	Compiler warnings	Refer Audit Findings
17	forwardFunds	internal	Passed	No Issue
18	setRate	external	access only Owner	No Issue
19	finalOfferingTokenWithdraw	write	access only Owner	No Issue
20	hasClosed	read	Passed	No Issue
21	addToWhitelist	external	Passed	No Issue
22	addManyToWhitelist	external	access only Owner	No Issue
23	removeFromWhitelist	external	access only Owner	No Issue
24	setAvailableReleaseFlg	write	access only Owner	No Issue

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

Email: audit@EtherAuthority.io

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

(1) Same LPToken can be added more than once: **MasterChef.sol**

The owner can add the same lptoken more than once.

Resolution: We suggest putting validation for duplicate lptoken while adding into the pool.

Low

(1) Function input parameters lack of check:

Variable validation is not performed in below functions:

IFO.sol

- setWhiteList = _whiteListaddress
- deposit = _amount
- withdraw = _amount
- harvestAndVesting = _amount
- finalOfferingTokenWithdraw = stageSecond
- setOfferingAmount = _offerAmount

MasterChef.sol

- add = _lpToken
- erc20Transfer = _to
- fund = _amount
- withdraw = _amount

IFOMaster.sol

- add = _lpToken
- erc20Transfer = _to
- fund = _amount

- withdraw = _amount

Resolution: We advise to put validation like integer type variables should be greater than 0 and address type variables should not be address(0).

(2) Critical operation lacks event log:
Missing event log for:

MasterChef.sol

- fund
- add
- set
- updatePool
- massUpdatePools

IFOMaster.sol

- fund
- add
- set
- updatePool
- massUpdatePools

Resolution: Write an event log for listed events.

(3) Infinite loop:
In below functions ,for loops do not have pid length limit , which costs more gas:

MasterChef.sol

- massUpdatePools.

IFOMaster.sol

- massUpdatePools.

Resolution: Upper limit should have a certain limit in for loops.

Very Low / Informational / Best practices:

(1) Unlimited minting: **Token.sol**
Minter can mint unlimited tokens.

Resolution: We suggest putting a minting limit.

(2) Immutable variables:

These variable values are set in the constructor & will be unchanged:

IFO.sol

stable, offeringToken, collateralToken, startTimestamp, endTimestamp, wallet, depositFeeBP

MasterChef.sol

rewardPerSecond, erc20 , startTimestamp, endTimestamp, feeAddress

IFOMaster.sol

rewardPerSecond, erc20 , startTimestamp, endTimestamp, feeAddres.

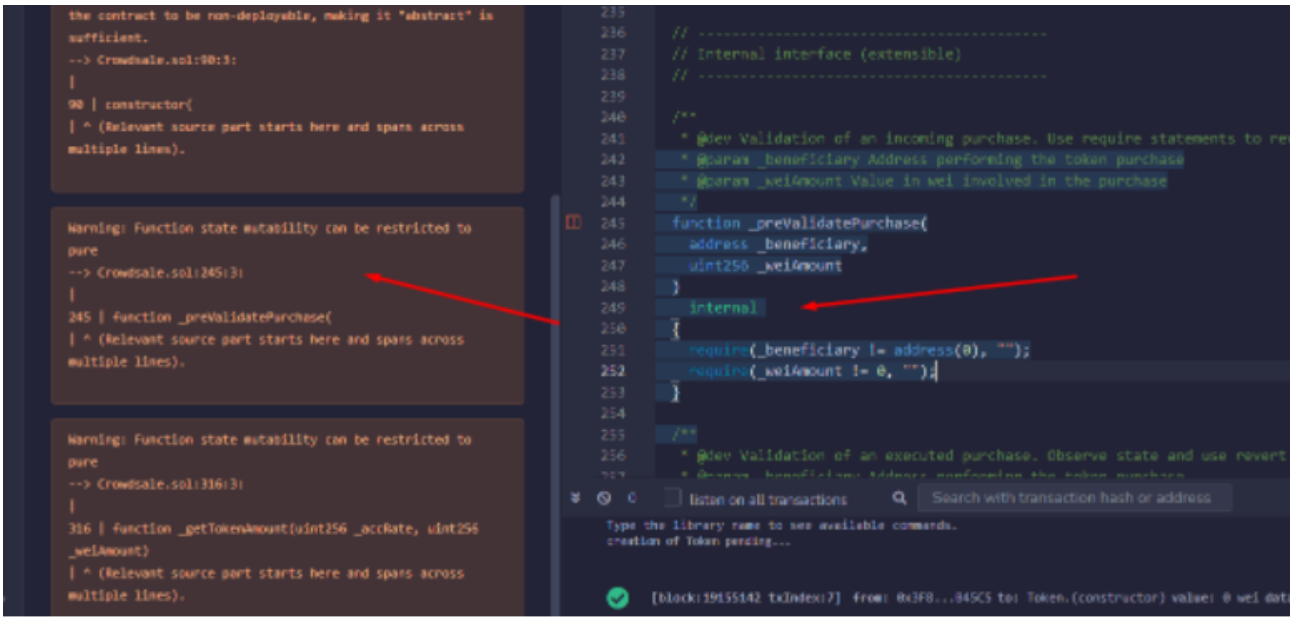
NFTStaking.sol

rewardRate, stakingStartTime, expectationStakeTokenCnt, expectationEmissionTokenCnt

Resolution: We suggest setting all variables as immutable.

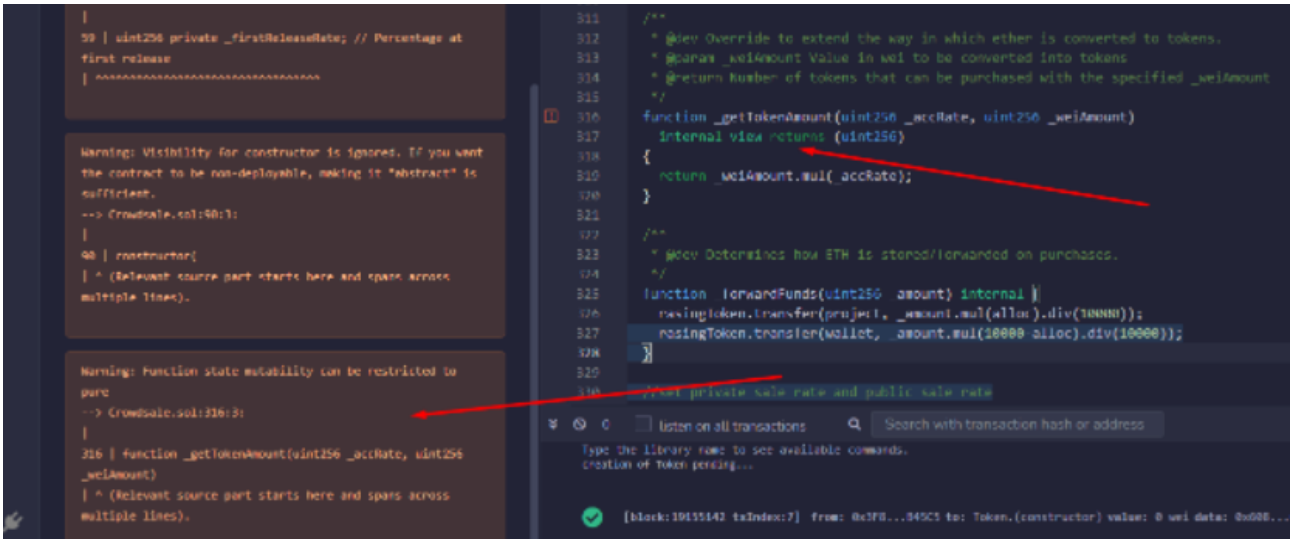
(3) Compiler warnings:

CrowdSale.sol

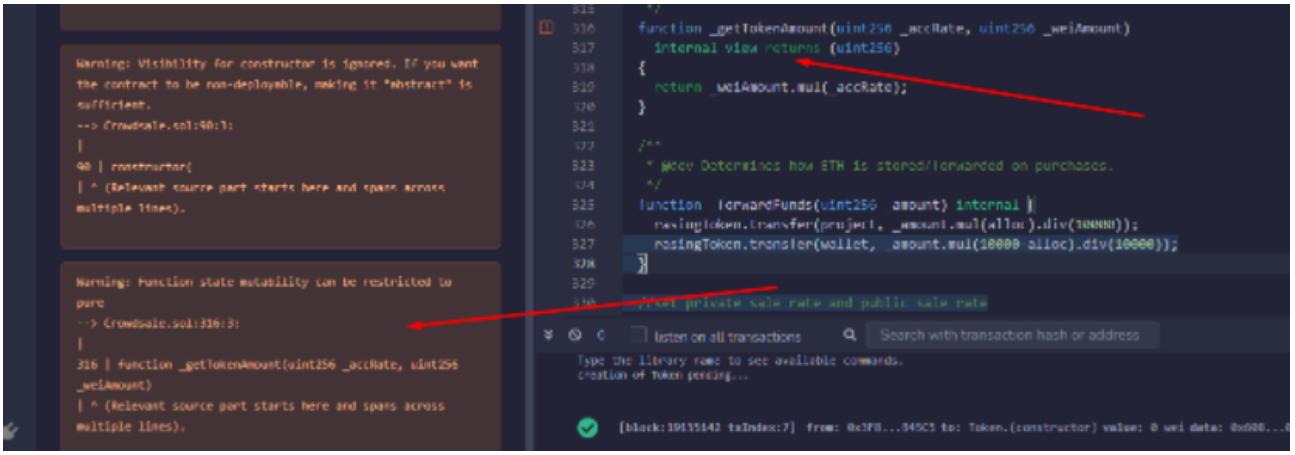


Function state mutability can be restricted to pure.

Resolution: We advise adding "pure" keywords.



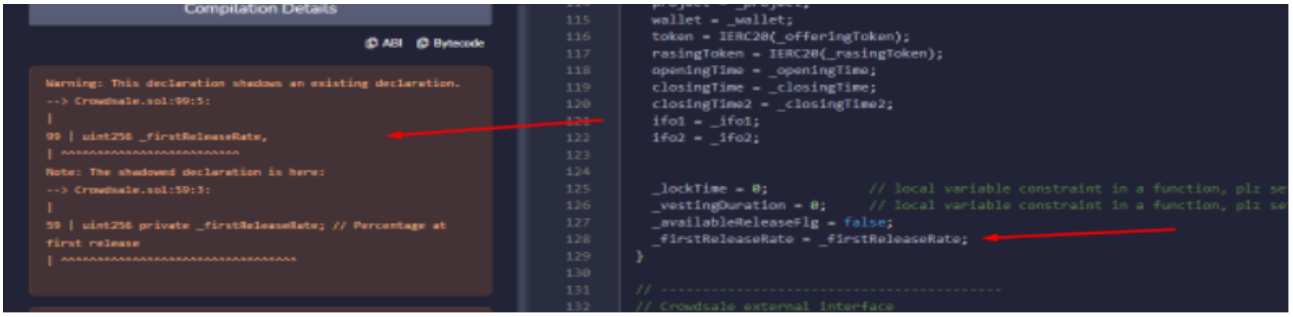
Resolution: We advise to remove the "view" keyword and add "pure" keyword.



This declaration shadows an existing declaration.

Resolution: We advise to change the variable name from _firstReleaseRate to _firstReleaseRate1.

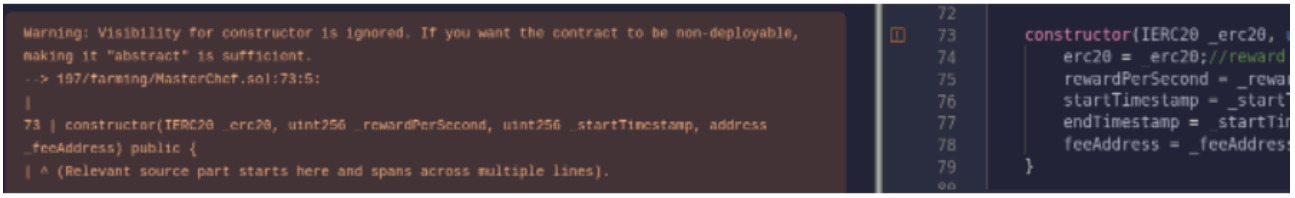
IFO.sol



This declaration shadows an existing declaration.

Resolution: We advise to change the variable name from _firstReleaseRate to _firstReleaseRate1.

MasterChef.sol



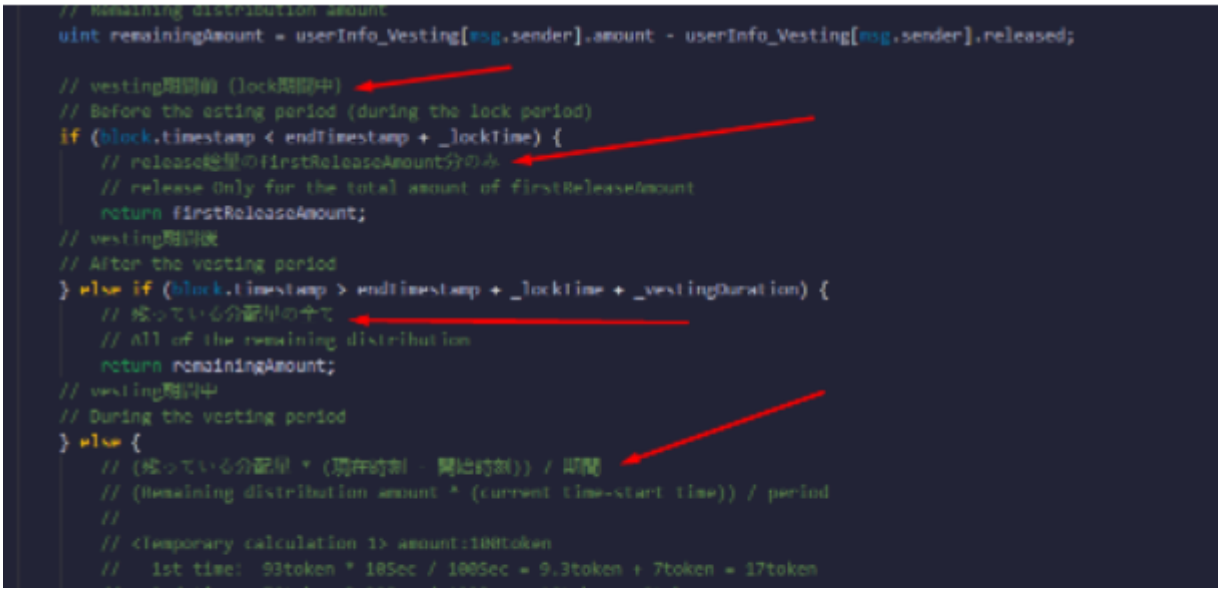
Resolution: Warning: Visibility for constructor is ignored. If you want the contract to be non-deployable, making it "abstract" is sufficient.

(4) SafeMath Library: **MasterChef.sol**

SafeMath Library is used in this contract code, but the compiler version is greater than or equal to 0.8.0, Then it will be not required to use, solidity automatically handles overflow / underflow.

Resolution: Remove the SafeMath library and use normal math operators, It will improve code size, and less gas consumption.

(5) Language Consistency: **IFO.sol, NFTStaking.sol**



Comment in Chinese and English Language.

Resolution: Consistency, Use the same language for all comments.

(6) Variable set by hardcoded values: **NFTStaking.sol**

```
stakingStartTime = 1646092800;
stakingEndTime = 9999999999;
```

stakingStartTime has been set by hardcoded values.

Resolution: Consider to set it by proper value while deploying.

(7) Integer variable initialized by zero: **NFTStaking.sol**

```
stakingCount = 0;    // 変更不可 Cannot be changed
```

The stakingCount is initialized by 0.

Resolution: We suggest removing this line from the constructor as the integer variable's default value is 0. So no need to initialize by 0.

Centralization

This smart contract has some functions which can be executed by the Admin (Owner) only. If the admin wallet private key would be compromised, then it would create trouble. Following are Admin functions:

- SetdepositNFTToken: NFTStaking owner can change the NFT contract address.
- SetdepositToken: NFTStaking owner can change deposit fee contract address and value.
- SetdepositAddress: NFTStaking owner can change the address to send the deposit fee.
- SetRewordToken: NFTStaking owner can reward change of contact address.
- DepositRewordToken: NFTStaking owner can deposit the reward in the contract.
- SetStakingEndTime: NFTStaking owner can change the staking end time.
- SetStakingDeadTime: NFTStaking owner can change the staking dead time.
- UnStakeByOwner: NFTStaking owner can force unStake by owner account.
- WithDraw: NFTStaking owner can withdraw reward from contract.
- add: MasterChef owners can add the same LP token more than once.

- set: MasterChef owners can update the given pool's ERC20 allocation point.
- setOfferingAmount: IFO owner can set offering amount.
- setStartTimestamp: IFO owner can set start timestamp.
- setEndTimestamp: IFO owner can set end timestamp.
- setRasingToken: IFO owner can set rasing token.
- setOfferingToken: IFO owner can set offering token.
- setCrowdsale: IFO owner can set crowd sale value.
- setPool: IFO owner can set ifo MasterChef address and specific pool for call some function value.
- setWhiteList: IFO owner can set whiteList address and rate.
- finalOfferingTokenWithdraw: IFO owner can withdraw after the end of all sale, withdraw all offering tokens.
- setAvailableReleaseFlg: IFO owner can set available release flg status.
- add: IFOMasterChef owner can add a new lp to the pool.
- set: IFOMasterChef owner can update the given pool's ERC20 allocation point.
- setAvailableClaimFlg: IFOMasterChef owner can set available release flg status.
- setRate: Crowdsale owner can set private sale rate and public sale rate.
- finalOfferingTokenWithdraw: Crowdsale owner can withdraw after the end of all sale, withdraw all offering tokens.
- addManyToWhitelist: Crowdsale owner can add list of addresses to whitelist.
- removeFromWhitelist: Crowdsale owner can remove single address from whitelist.
- setAvailableReleaseFlg: Crowdsale owner can set available release flg status.

To make the smart contract 100% decentralized, we suggest renouncing ownership in the smart contract once its function is completed.

Conclusion

We were given a contract code in the form of files. And we have used all possible tests based on given objects as files. We have observed some issues in the smart contracts. **So, the smart contracts are ready for the mainnet deployment after fixing those issues**.

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security state of the reviewed contract, based on standard audit procedure scope, is **"Secured"**.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

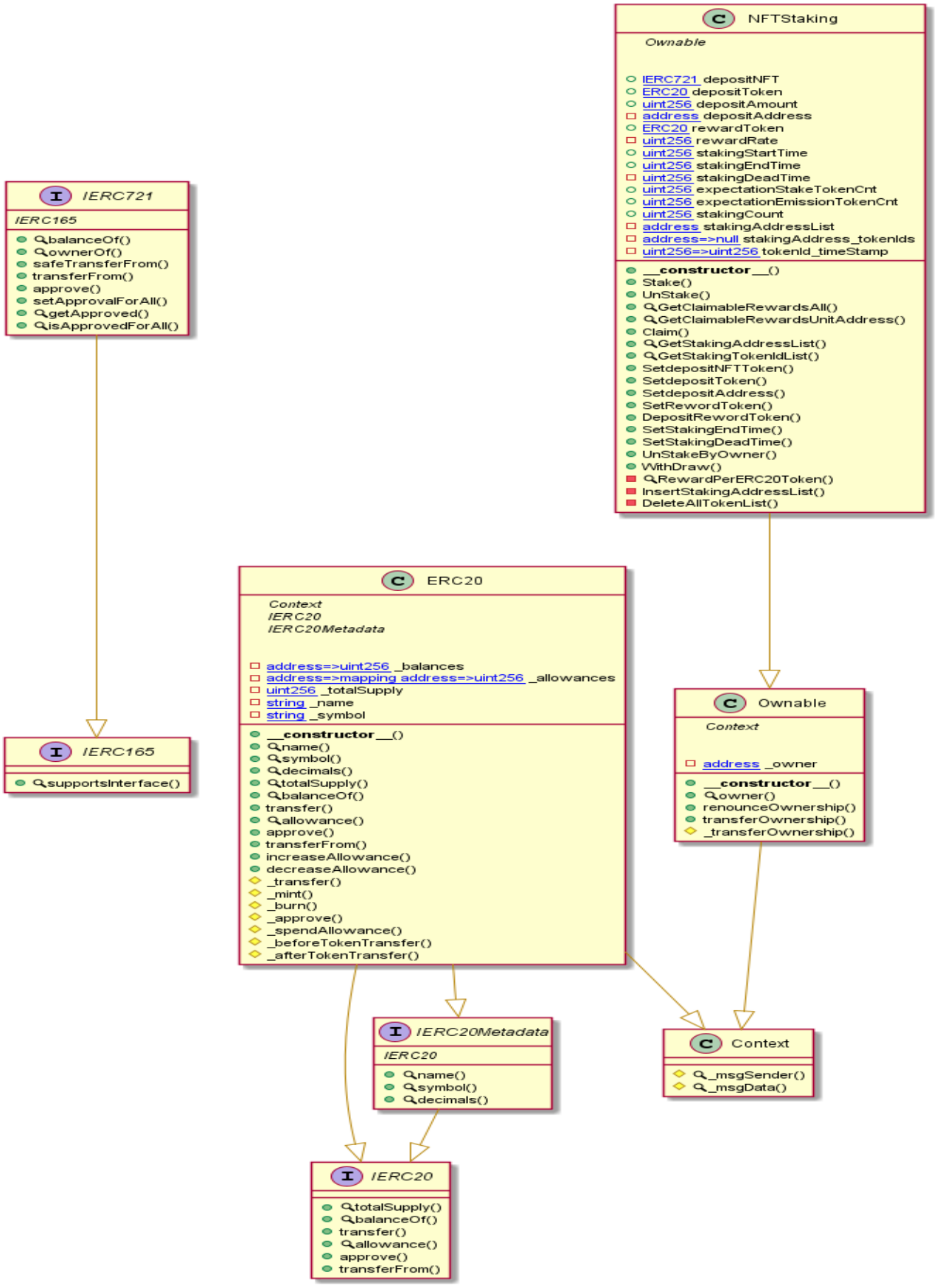
Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can’t guarantee explicit security of the audited smart contracts.

Appendix

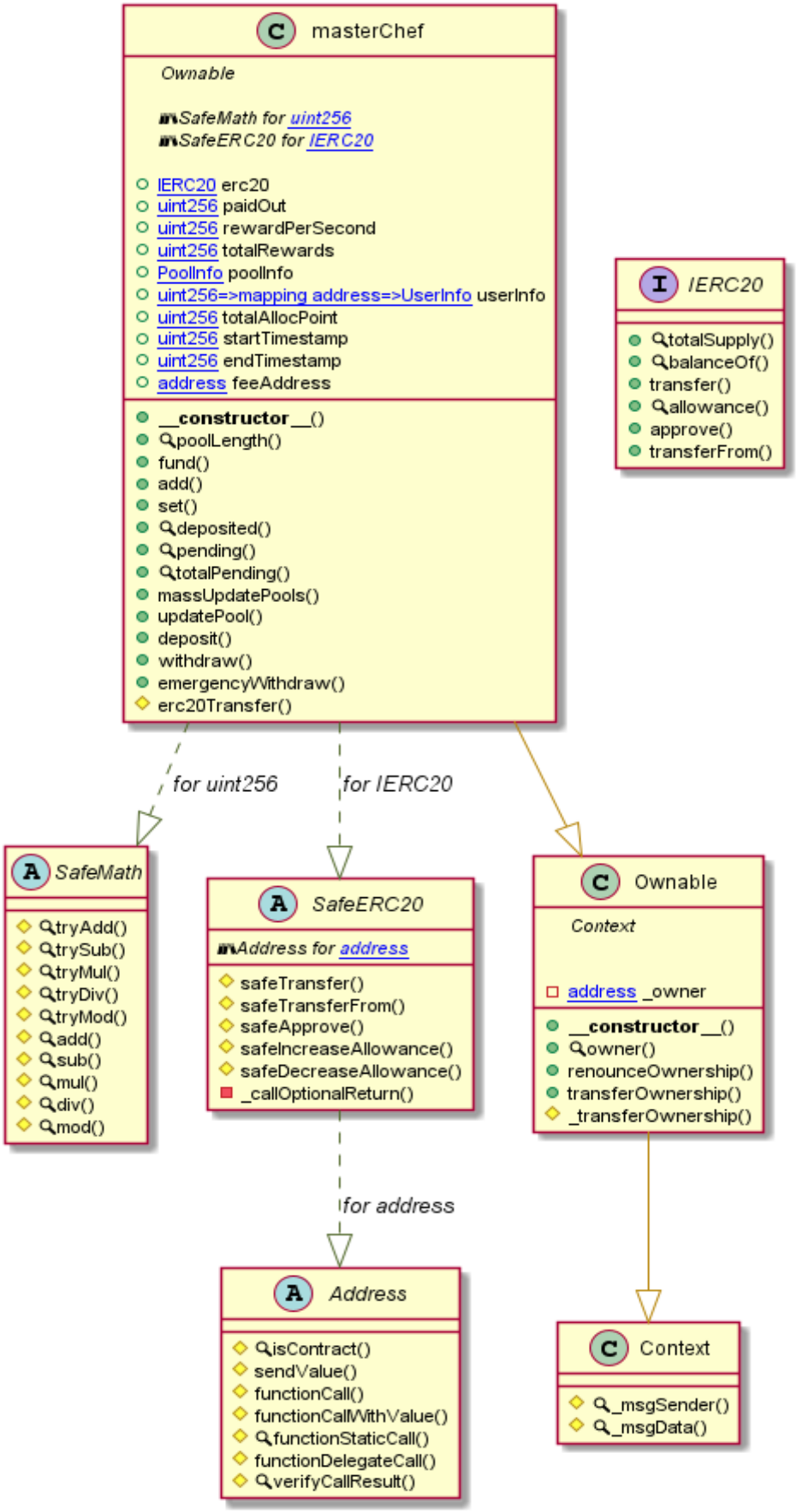
Code Flow Diagram - GaiaStarter Protocol

NFTStaking Diagram

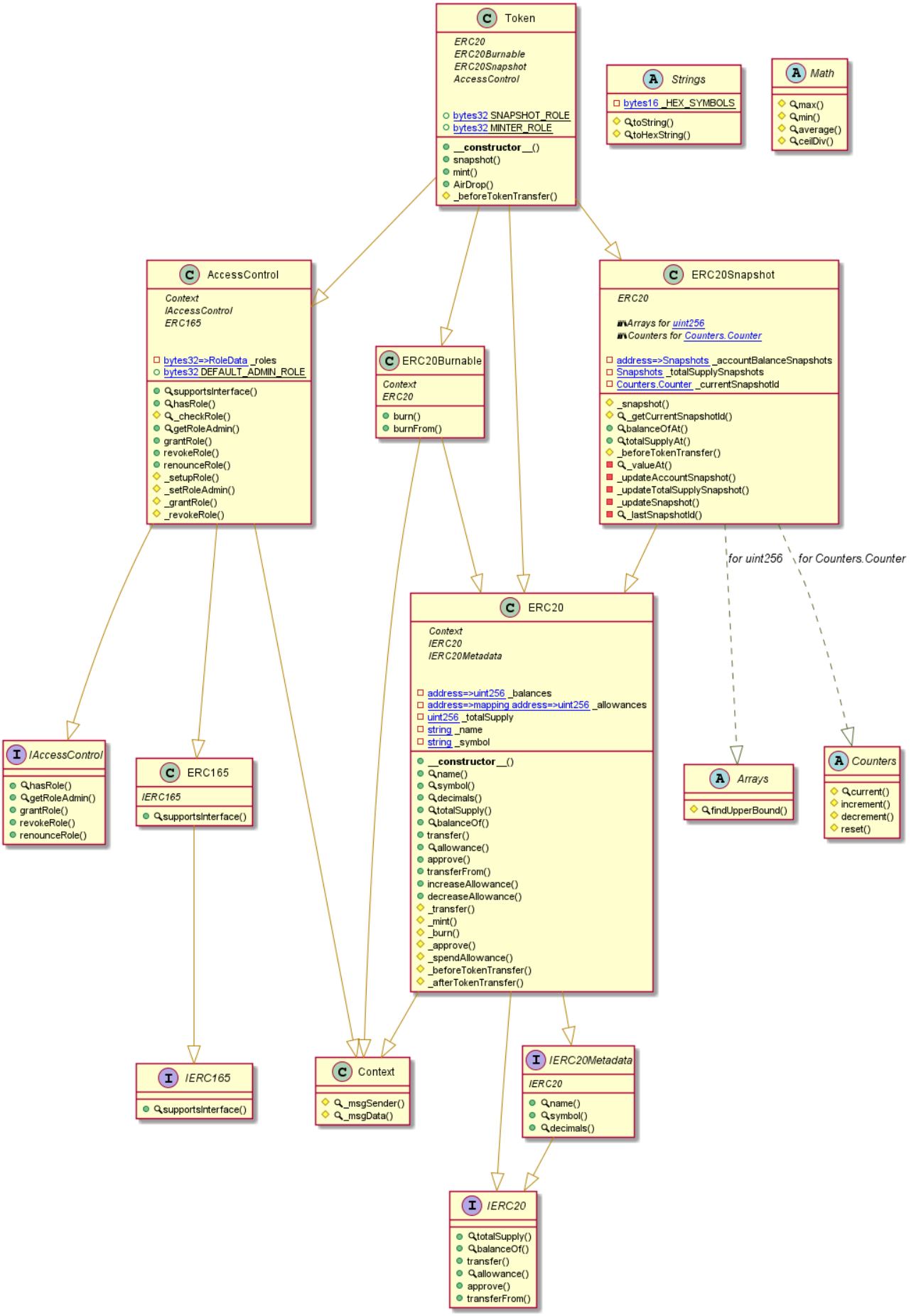


Email: audit@EtherAuthority.io

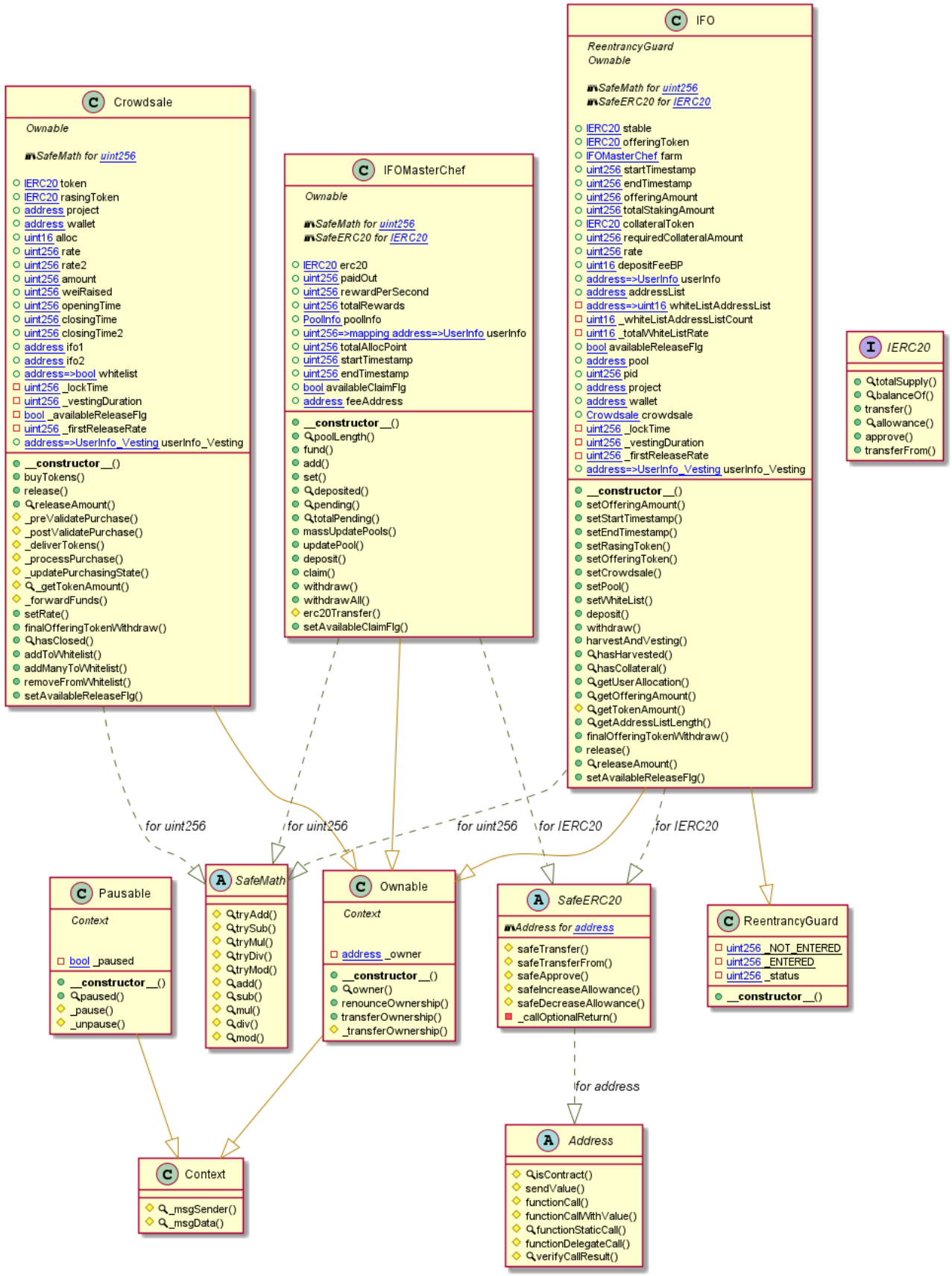
MasterChef Diagram



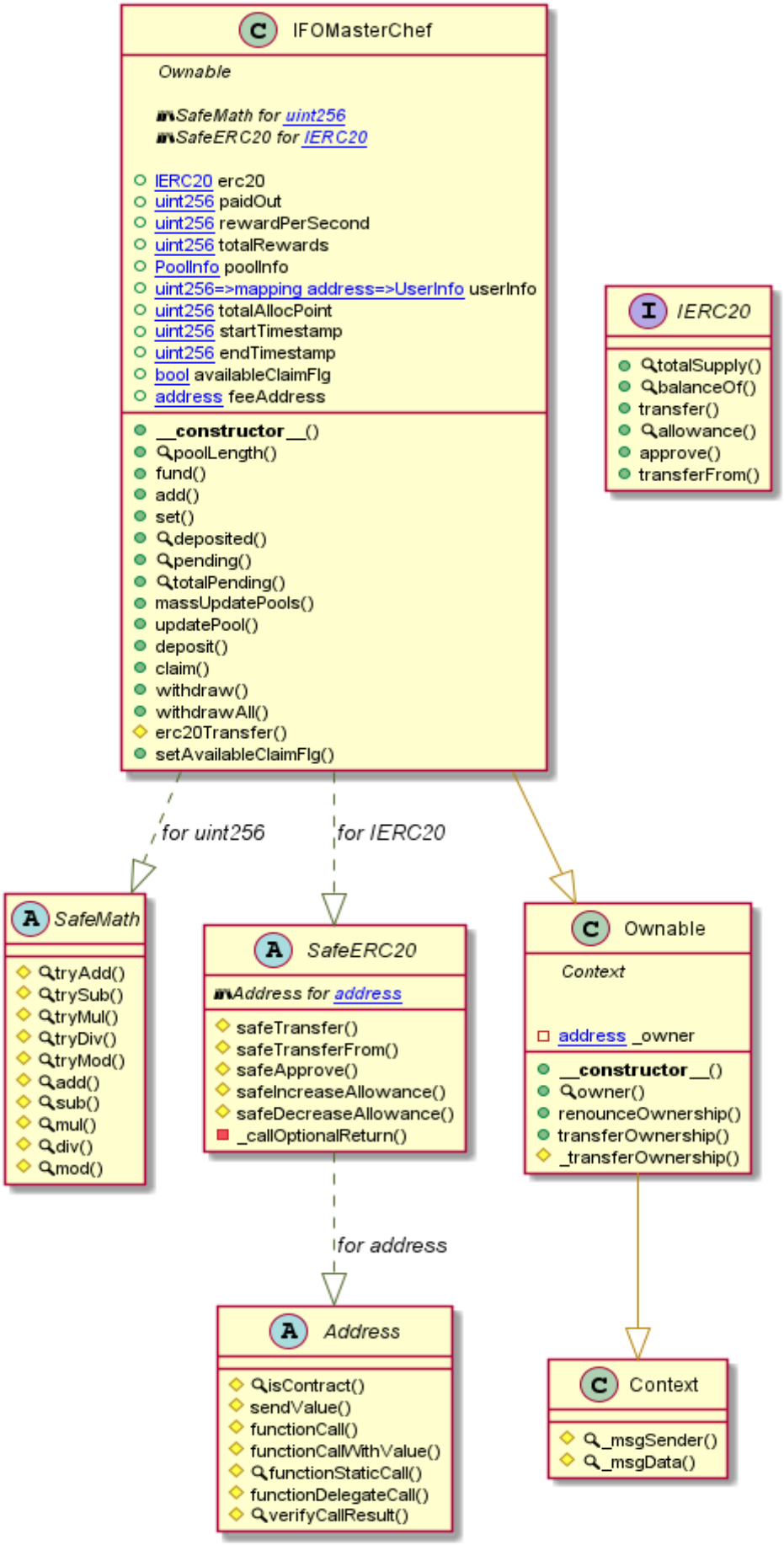
Token Diagram



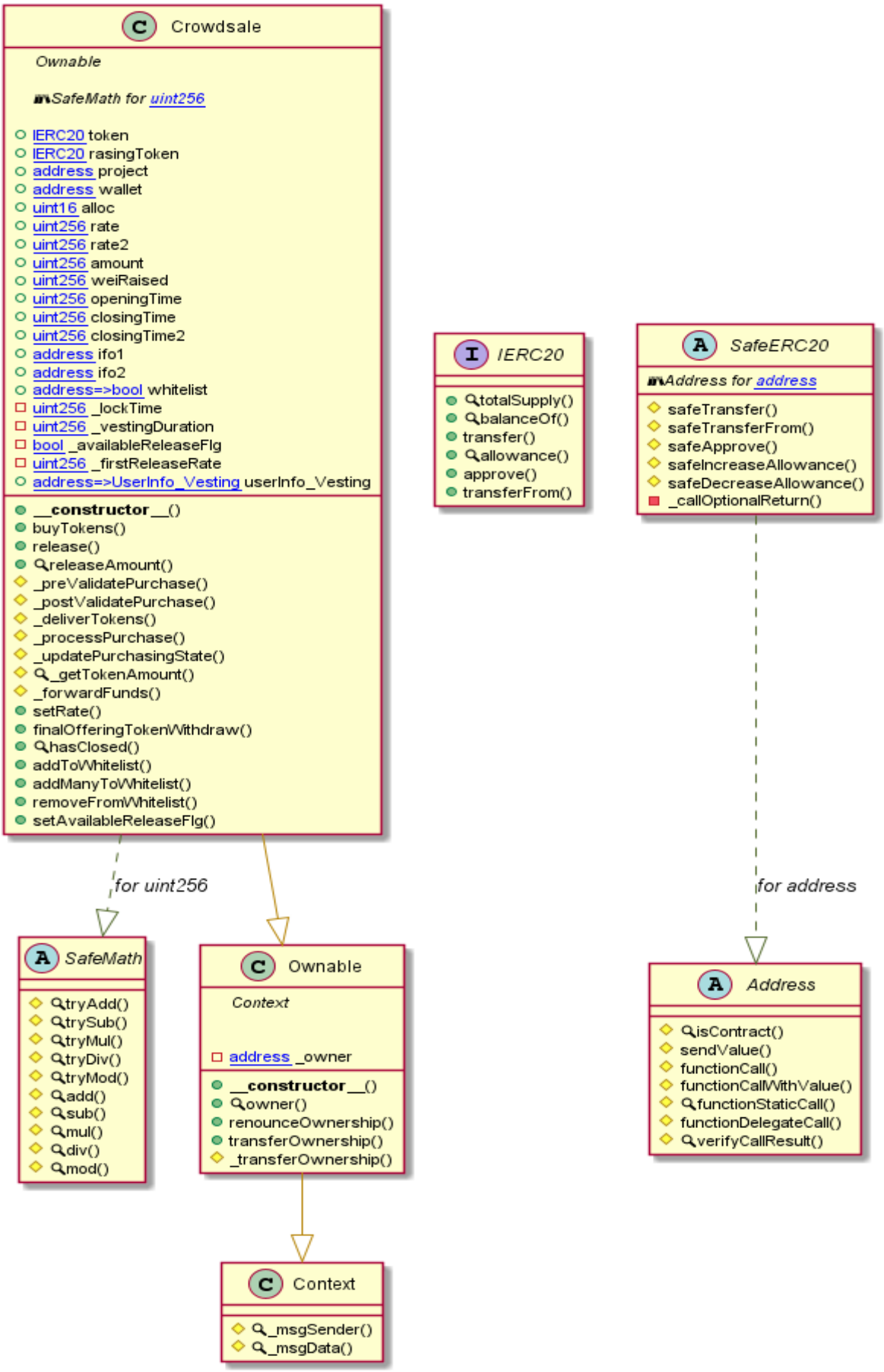
IFO Diagram



IFOMasterChef Diagram



Crowdsale Diagram



Slither Results Log

Slither log >> NFTStaking.sol

```
Variable NFTStaking.stakingAddress_tokenIds (NFTStaking.sol#692) is not in mixedCase
Variable NFTStaking.tokenId_timeStamp (NFTStaking.sol#697) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
name() should be declared external:
        - ERC20.name() (NFTStaking.sol#278-280)
symbol() should be declared external:
        - ERC20.symbol() (NFTStaking.sol#286-288)
decimals() should be declared external:
        - ERC20.decimals() (NFTStaking.sol#303-305)
totalSupply() should be declared external:
        - ERC20.totalSupply() (NFTStaking.sol#310-312)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (NFTStaking.sol#317-319)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (NFTStaking.sol#329-333)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (NFTStaking.sol#352-356)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (NFTStaking.sol#374-383)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (NFTStaking.sol#397-401)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (NFTStaking.sol#417-426)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (NFTStaking.sol#635-637)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (NFTStaking.sol#643-646)
GetStakingAddressList() should be declared external:
        - NFTStaking.GetStakingAddressList() (NFTStaking.sol#894-896)
GetStakingTokenIdList() should be declared external:
        - NFTStaking.GetStakingTokenIdList() (NFTStaking.sol#900-902)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:NFTStaking.sol analyzed (8 contracts with 75 detectors), 81 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Slither log >> MasterChef.sol

```
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (MasterChef.sol#631-633)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (MasterChef.sol#639-642)
fund(uint256) should be declared external:
        - masterChef.fund(uint256) (MasterChef.sol#729-734)
add(uint256,IERC20,uint16,bool) should be declared external:
        - masterChef.add(uint256,IERC20,uint16,bool) (MasterChef.sol#739-754)
set(uint256,uint256,uint16,bool) should be declared external:
        - masterChef.set(uint256,uint256,uint16,bool) (MasterChef.sol#757-765)
deposit(uint256,uint256) should be declared external:
        - masterChef.deposit(uint256,uint256) (MasterChef.sol#832-857)
withdraw(uint256,uint256) should be declared external:
        - masterChef.withdraw(uint256,uint256) (MasterChef.sol#860-875)
emergencyWithdraw(uint256) should be declared external:
        - masterChef.emergencyWithdraw(uint256) (MasterChef.sol#878-886)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:MasterChef.sol analyzed (7 contracts with 75 detectors), 78 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Slither log >> Token.sol

```
INFO:Detectors:
Pragma version^0.8.4 (Token.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Function Token.AirDrop(address[],uint256[]) (Token.sol#1109-1115) is not in mixedCase
Parameter Token.AirDrop(address[],uint256[])._targetAddresses (Token.sol#1109) is not in mixedCase
Parameter Token.AirDrop(address[],uint256[])._value (Token.sol#1109) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
grantRole(bytes32,address) should be declared external:
        - AccessControl.grantRole(bytes32,address) (Token.sol#464-466)
revokeRole(bytes32,address) should be declared external:
        - AccessControl.revokeRole(bytes32,address) (Token.sol#477-479)
renounceRole(bytes32,address) should be declared external:
        - AccessControl.renounceRole(bytes32,address) (Token.sol#495-499)
name() should be declared external:
        - ERC20.name() (Token.sol#586-588)
symbol() should be declared external:
        - ERC20.symbol() (Token.sol#594-596)
decimals() should be declared external:
        - ERC20.decimals() (Token.sol#611-613)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (Token.sol#637-641)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (Token.sol#660-664)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (Token.sol#682-691)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (Token.sol#705-709)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (Token.sol#725-734)
```

```
burn(uint256) should be declared external:
        - ERC20Burnable.burn(uint256) (Token.sol#915-917)
burnFrom(address,uint256) should be declared external:
        - ERC20Burnable.burnFrom(address,uint256) (Token.sol#930-933)
balanceOfAt(address,uint256) should be declared external:
        - ERC20Snapshot.balanceOfAt(address,uint256) (Token.sol#1000-1004)
totalSupplyAt(uint256) should be declared external:
        - ERC20Snapshot.totalSupplyAt(uint256) (Token.sol#1009-1013)
snapshot() should be declared external:
        - Token.snapshot() (Token.sol#1101-1103)
mint(address,uint256) should be declared external:
        - Token.mint(address,uint256) (Token.sol#1105-1107)
AirDrop(address[],uint256[]) should be declared external:
        - Token.AirDrop(address[],uint256[]) (Token.sol#1109-1115)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:Token.sol analyzed (15 contracts with 75 detectors), 41 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Slither log >> IFO.sol

```
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (IFO.sol#634-636)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (IFO.sol#642-645)
release() should be declared external:
        - Crowdsale.release() (IFO.sol#899-908)
finalOfferingTokenWithdraw(address) should be declared external:
        - Crowdsale.finalOfferingTokenWithdraw(address) (IFO.sol#1057-1060)
hasClosed() should be declared external:
        - Crowdsale.hasClosed() (IFO.sol#1065-1068)
setAvailableReleaseFlg(bool) should be declared external:
        - Crowdsale.setAvailableReleaseFlg(bool) (IFO.sol#1097-1099)
fund(uint256) should be declared external:
        - IFOMasterChef.fund(uint256) (IFO.sol#1175-1180)
add(uint256,IERC20,uint16,address,bool) should be declared external:
        - IFOMasterChef.add(uint256,IERC20,uint16,address,bool) (IFO.sol#1184-1200)
set(uint256,uint256,uint16,bool) should be declared external:
        - IFOMasterChef.set(uint256,uint256,uint16,bool) (IFO.sol#1203-1211)
deposit(address,uint256,uint256) should be declared external:
        - IFOMasterChef.deposit(address,uint256,uint256) (IFO.sol#1284-1307)
claim(uint256,address) should be declared external:
        - IFOMasterChef.claim(uint256,address) (IFO.sol#1310-1326)
withdraw(uint256,uint256,address) should be declared external:
        - IFOMasterChef.withdraw(uint256,uint256,address) (IFO.sol#1329-1345)
withdrawAll(uint256,address) should be declared external:
        - IFOMasterChef.withdrawAll(uint256,address) (IFO.sol#1348-1360)
setAvailableClaimFlg(bool) should be declared external:
        - IFOMasterChef.setAvailableClaimFlg(bool) (IFO.sol#1368-1370)
setOfferingAmount(uint256) should be declared external:
        - IFO.setOfferingAmount(uint256) (IFO.sol#1457-1460)
setStartTimestamp(uint256) should be declared external:
        - IFO.setStartTimestamp(uint256) (IFO.sol#1462-1465)
```

```
setStartTimestamp(uint256) should be declared external:
        - IFO.setStartTimestamp(uint256) (IFO.sol#1462-1465)
setEndTimestamp(uint256) should be declared external:
        - IFO.setEndTimestamp(uint256) (IFO.sol#1467-1470)
setRasingToken(address) should be declared external:
        - IFO.setRasingToken(address) (IFO.sol#1472-1475)
setOfferingToken(address) should be declared external:
        - IFO.setOfferingToken(address) (IFO.sol#1477-1480)
setCrowdsale(address) should be declared external:
        - IFO.setCrowdsale(address) (IFO.sol#1482-1485)
setPool(address,uint256) should be declared external:
        - IFO.setPool(address,uint256) (IFO.sol#1488-1493)
setWhiteList(address,uint16) should be declared external:
        - IFO.setWhiteList(address,uint16) (IFO.sol#1496-1520)
deposit(uint256) should be declared external:
        - IFO.deposit(uint256) (IFO.sol#1525-1545)
withdraw(uint256) should be declared external:
        - IFO.withdraw(uint256) (IFO.sol#1548-1559)
harvestAndVesting(uint256) should be declared external:
        - IFO.harvestAndVesting(uint256) (IFO.sol#1562-1593)
finalOfferingTokenWithdraw(address) should be declared external:
        - IFO.finalOfferingTokenWithdraw(address) (IFO.sol#1632-1635)
release() should be declared external:
        - IFO.release() (IFO.sol#1639-1648)
setAvailableReleaseFlg(bool) should be declared external:
        - IFO.setAvailableReleaseFlg(bool) (IFO.sol#1694-1696)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:IFO.sol analyzed (11 contracts with 75 detectors), 221 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Slither log >> IFOMasterChef.sol

```
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (IFOMasterChef.sol#634-636)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (IFOMasterChef.sol#642-645)
fund(uint256) should be declared external:
        - IFOMasterChef.fund(uint256) (IFOMasterChef.sol#739-744)
add(uint256,IERC20,uint16,address,bool) should be declared external:
        - IFOMasterChef.add(uint256,IERC20,uint16,address,bool) (IFOMasterChef.sol#748-764)
set(uint256,uint256,uint16,bool) should be declared external:
        - IFOMasterChef.set(uint256,uint256,uint16,bool) (IFOMasterChef.sol#767-775)
deposit(address,uint256,uint256) should be declared external:
        - IFOMasterChef.deposit(address,uint256,uint256) (IFOMasterChef.sol#848-871)
claim(uint256,address) should be declared external:
        - IFOMasterChef.claim(uint256,address) (IFOMasterChef.sol#874-890)
withdraw(uint256,uint256,address) should be declared external:
        - IFOMasterChef.withdraw(uint256,uint256,address) (IFOMasterChef.sol#893-909)
withdrawAll(uint256,address) should be declared external:
        - IFOMasterChef.withdrawAll(uint256,address) (IFOMasterChef.sol#912-924)
setAvailableClaimFlg(bool) should be declared external:
        - IFOMasterChef.setAvailableClaimFlg(bool) (IFOMasterChef.sol#932-934)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:IFOMasterChef.sol analyzed (7 contracts with 75 detectors), 89 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Slither log >> Crowdsale.sol

```
Parameter Crowdsale.setAvailableReleaseFlg(bool)._flg (Crowdsale.sol#1014) is not in mixedCase
Variable Crowdsale.userInfo_Vesting (Crowdsale.sol#699) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Variable Crowdsale.constructor(uint16,address,address,address,address,uint256,uint256,uint256,uint256,address,address)._closing
Time (Crowdsale.sol#732) is too similar to Crowdsale.closingTime2 (Crowdsale.sol#683)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar
INFO:Detectors:
Crowdsale._firstReleaseRate (Crowdsale.sol#694) should be constant
Crowdsale.amount (Crowdsale.sol#676) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (Crowdsale.sol#632-634)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (Crowdsale.sol#640-643)
release() should be declared external:
        - Crowdsale.release() (Crowdsale.sol#816-825)
finalOfferingTokenWithdraw(address) should be declared external:
        - Crowdsale.finalOfferingTokenWithdraw(address) (Crowdsale.sol#974-977)
hasClosed() should be declared external:
        - Crowdsale.hasClosed() (Crowdsale.sol#982-985)
setAvailableReleaseFlg(bool) should be declared external:
        - Crowdsale.setAvailableReleaseFlg(bool) (Crowdsale.sol#1014-1016)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:Crowdsale.sol analyzed (7 contracts with 75 detectors), 72 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Solidity Static Analysis

NFTStaking.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in NFTStaking.Stake(uint256[]): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 747:52:

Gas & Economy

Gas costs:

Gas requirement of function NFTStaking.Claim is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 899:50:

Miscellaneous

Constant/View/Pure functions:

NFTStaking.InsertStakingAddressList(address) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 1092:893:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

more

Pos: 1006:78:

MasterChef.sol

Security

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

more

Pos: 783:76:

Email: audit@EtherAuthority.io

Gas & Economy

Gas costs:

Gas requirement of function masterChef.fund is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 731:11:

Miscellaneous

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 744:96:

Token.sol

Gas & Economy

Gas costs:

Gas requirement of function Token.AirDrop is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 1109:4:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.
more
Pos: 1059:8:

Miscellaneous

Constant/View/Pure functions:

Token._beforeTokenTransfer(address,address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 1119:4:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 1110:8:

IFO.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in IFO.harvestAndVesting(uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 1565:58:

Gas & Economy

Gas costs:

Gas requirement of function IFO.harvestAndVesting is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 1565:58:

Miscellaneous

Constant/View/Pure functions:

IFO.finalOfferingTokenWithdraw(address) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 1637:1:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 1568:26:

IFOMasterChef.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in IFOMasterChef.erc20Transfer(address,uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 927:34:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
more
Pos: 753:73:

Gas & Economy

Gas costs:

Gas requirement of function IFOMasterChef.fund is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 739:4:

Miscellaneous

Constant/View/Pure functions:

IFOMasterChef.updatePool(uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 826:34:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 749:8:

Crowdsale.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in Crowdsale.buyTokens(address,uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 781:2:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

more

Pos: 783:43:

Gas & Economy

Gas costs:

Fallback function of contract Crowdsale requires too much gas (infinite). If the fallback function requires more than 2300 gas, the contract cannot receive Ether.

Pos: 773:3:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.

more

Pos: 1008:34:

Miscellaneous

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

more

Pos: 788:8:

Solhint Linter

NFTStaking.sol

```
NFTStaking.sol:421:18: Error: Parse error: missing ';' at '{'
NFTStaking.sol:454:18: Error: Parse error: missing ';' at '{'
NFTStaking.sol:503:18: Error: Parse error: missing ';' at '{'
NFTStaking.sol:554:22: Error: Parse error: missing ';' at '{'
```

MasterChef.sol

```
MasterChef.sol:161:18: Error: Parse error: missing ';' at '{'
MasterChef.sol:184:18: Error: Parse error: missing ';' at '{'
MasterChef.sol:210:18: Error: Parse error: missing ';' at '{'
MasterChef.sol:561:18: Error: Parse error: missing ';' at '{'
```

Token.sol

```
Token.sol:811:18: Error: Parse error: missing ';' at '{'
Token.sol:862:22: Error: Parse error: missing ';' at '{'
```

IFO.sol

```
IFO.sol:185:18: Error: Parse error: missing ';' at '{'
IFO.sol:211:18: Error: Parse error: missing ';' at '{'
IFO.sol:564:18: Error: Parse error: missing ';' at '{'
```

IFOMasterChef.sol

```
IFOMasterChef.sol:161:18: Error: Parse error: missing ';' at '{'
IFOMasterChef.sol:184:18: Error: Parse error: missing ';' at '{'
IFOMasterChef.sol:210:18: Error: Parse error: missing ';' at '{'
IFOMasterChef.sol:563:18: Error: Parse error: missing ';' at '{'
```

Crowdsale.sol

```
Crowdsale.sol:184:18: Error: Parse error: missing ';' at '{'
```

```
Crowdsale.sol:210:18: Error: Parse error: missing ';' at '{'
Crowdsale.sol:561:18: Error: Parse error: missing ';' at '{'
```

Overall Software analysis result:

These software reported many false positive results and some are informational issues. So, those issues can be safely ignored.

Email: audit@EtherAuthority.io