Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	LunaFi VLFI Token
Website:	https://lunafi.io
Platform:	Polygon
Language:	Solidity
Date:	June 4th, 2023

Table of contents

Introduction 4
Project Background 4
Audit Scope 4
Claimed Smart Contract Features 5
Audit Summary 6
Technical Quick Stats 7
Code Quality 8
Documentation 8
Use of Dependencies 8
AS-IS overview 9
Severity Definitions 12
Audit Findings 13
Conclusion 17
Our Methodology 18
Disclaimers 20
Appendix
- Code Flow Diagram 21
- Slither Results Log 22
- Solidity static analysis 24
- Solhint Linter 26

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 LunaFi team to perform the Security audit of the VLFI Token smart contract 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 June 4th, 2023.

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

- LunaFi is a gaming smart contract which has functions stake, unStake, updateFarm, updateBets, storeBets, ceil, etc
- LunaFi contract inherits the ERC20Upgradeable, AccessControlUpgradeable, SignatureChecker, draft-EIP712Upgradeable, draft-ERC20PermitUpgradeable, ERC20VotesUpgradeable, SafeERC20 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 LunaFi Token Smart Contract**
Platform	**Polygon / Solidity**
File	VLFI.sol
File MD5 Hash	41ADA3D302F5D886BB1767CECD5DE28D
Revised File MD5 Hash	1A44CD3DEFA59CD5EAC000E7904E361C
Audit Date	June 5th, 2023
Revision Date	June 8th, 2023

Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Token Name: VLFI • Token Symbol: vLFI • Decimals: 18	**YES, This is valid.**
Manager has control over following functions: • Set the poolName. • Set the pool decimals. • Set the initial pool token price. • Set treasury address. • Set the cooldownActive state. • Withdraw the funds to the treasury.	**YES, This is valid.**
Staking_Manager has control over following functions: • Set the unstake window time. • Set cooldown seconds. • Sets the rewards per second.	**YES, This is valid.**
Data Provider Oracle has control over following functions: • Set the VOI and signature of the authorized user.	**YES, This is valid.**

House Pool Data Provider has control over following functions: ● Store the bets placed. ● update the bet information. ● Settle bets.	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity based 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 1 critical, 3 high, 0 medium and 1 low and some very low level issues. These issues are fixed / acknowledged in the revised smart contract code.

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.

Email: audit@EtherAuthority.io

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	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	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	Passed
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	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Email: audit@EtherAuthority.io

Code Quality

This audit scope has 1 smart contract. Smart contract contains Libraries, Smart contracts, inherits and Interfaces. This is a compact and well written smart contract.

The libraries in the LunaFi Token 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 LunaFi Token.

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

Code parts are well commented on in the smart contracts. Ethereum's NatSpec commenting style is used, which is a good thing.

Documentation

We were given a LunaFi Staking Token smart contract code in the form of a file. The hash of that code is mentioned above in the table.

As mentioned above, code parts are well commented on. 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.

Another source of information was its website https://lunafi.io which provided rich information about the project architecture.

Use of Dependencies

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

Apart from libraries, its functions are not used in external smart contract calls.

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	onlyValid	modifier	Passed	No Issue
3	initialize	external	initializer	No Issue
4	setPoolName	external	access only Role	No Issue
5	setDecimals	external	access only Role	No Issue
6	initLpTokenPrice	external	Passed	No Issue
7	setTreasuryAddress	external	access only Role	No Issue
8	getTreasuryAddress	external	Passed	No Issue
9	getUserCooldown	external	Passed	No Issue
10	getUserRewardDebt	external	Passed	No Issue
11	getCooldownSeconds	external	Passed	No Issue
12	getUnstakeWindowTime	external	Passed	No Issue
13	getAccRewardPerShare	external	Passed	No Issue
14	getLastRewardTime	external	Passed	No Issue
15	getRewards	external	Passed	No Issue
16	getRewardPerSecond	external	Passed	No Issue
17	getSportsBookContract	external	Passed	No Issue
18	setSportsBookContract	external	access only Role	No Issue
19	setUnstakeWindowTime	external	access only Role	No Issue
20	setCooldownSeconds	external	access only Role	No Issue
21	updateFarm	write	Passed	No Issue
22	setRewardPerSecond	write	access only Role	No Issue
23	permitAndStake	external	Passed	No Issue
24	unstakeMax	write	Passed	No Issue
25	setCoolDownActiveState	external	access only Role	No Issue
26	getCoolDownActiveState	external	Passed	No Issue
27	activateCooldown	external	Passed	No Issue
28	claimRewards	external	Passed	No Issue
29	getuserEVTrackerForTheUser	external	Passed	No Issue
30	getUserBets	external	Passed	No Issue
31	getBetInfoByID	external	Passed	No Issue
32	getEV	external	Passed	No Issue
33	getMaxExposure	external	Passed	No Issue
34	getMyLiquidity	external	Passed	No Issue
35	setVOI	external	access only Role	No Issue
36	storeBets	external	Passed	No Issue
37	updateBets	external	Passed	No Issue
38	settleBets	external	Passed	No Issue
39	calculateNewEVValue	read	Passed	No Issue
40	stake	write	Passed	No Issue
41	unStake	write	Passed	No Issue
42	withdrawToTreasury	external	access only Role	No Issue

43	getMaxWithdrawal	read	Passed	No Issue
44	cleanUserMapping	internal	Passed	No Issue
45	getNextCooldownTimestamp	read	Passed	No Issue
46	_transfer	internal	Passed	No Issue
47	_afterTokenTransfer	internal	Passed	No Issue
48	_mint	internal	Passed	No Issue
49	createFarm	internal	Passed	No Issue
50	_burn	internal	Passed	No Issue
51	updateAttributes	internal	Passed	No Issue
52	_updateTVL	internal	Passed	No Issue
53	_setVoi	internal	Passed	No Issue
54	_setME	internal	Passed	No Issue
55	_setEV	internal	Passed	No Issue
56	ceil	internal	Passed	No Issue
57	farmUtil	internal	Passed	No Issue
58	TreasuryAmountWithdrawal	internal	Passed	No Issue
59	__ERC20_init	internal	access only Initializing	No Issue
60	__ERC20_init_unchained	internal	access only Initializing	No Issue
61	name	read	Passed	No Issue
62	symbol	read	Passed	No Issue
63	decimals	read	Passed	No Issue
64	totalSupply	read	Passed	No Issue
65	balanceOf	read	Passed	No Issue
66	transfer	write	Passed	No Issue
67	allowance	read	Passed	No Issue
68	approve	write	Passed	No Issue
69	transferFrom	write	Passed	No Issue
70	increaseAllowance	write	Passed	No Issue
71	decreaseAllowance	write	Passed	No Issue
72	_transfer	internal	Passed	No Issue
73	_update	internal	Passed	No Issue
74	_mint	internal	Passed	No Issue
75	_burn	internal	Passed	No Issue
76	_approve	internal	Passed	No Issue
77	_spendAllowance	internal	Passed	No Issue
78	__AccessControl_init	internal	access only Initializing	No Issue
79	__AccessControl_init_unchained	internal	access only Initializing	No Issue
80	onlyRole	modifier	Passed	No Issue
81	supportsInterface	read	Passed	No Issue
82	hasRole	read	Passed	No Issue
83	_checkRole	internal	Passed	No Issue
84	_checkRole	internal	Passed	No Issue
85	getRoleAdmin	read	Passed	No Issue
86	grantRole	write	access only Role	No Issue

87	revokeRole	write	access only Role	No Issue
88	renounceRole	write	Passed	No Issue
89	_setRoleAdmin	internal	Passed	No Issue
90	_grantRole	internal	Passed	No Issue
91	_revokeRole	internal	Passed	No Issue
92	__ERC20Permit_init	internal	access only Initializing	No Issue
93	__ERC20Permit_init_unchained	internal	access only Initializing	No Issue
94	permit	write	Passed	No Issue
95	nonces	read	Passed	No Issue
96	DOMAIN_SEPARATOR	external	Passed	No Issue
97	__ERC20Votes_init	internal	access only Initializing	No Issue
98	__ERC20Votes_init_unchained	internal	access only Initializing	No Issue
99	_maxSupply	internal	Passed	No Issue
100	_update	internal	Passed	No Issue
101	_getVotingUnits	internal	Passed	No Issue
102	numCheckpoints	read	Passed	No Issue
103	checkpoints	read	Passed	No Issue
104	__EIP712_init	internal	access only Initializing	No Issue
105	__EIP712_init_unchained	internal	access only Initializing	No Issue
106	_domainSeparatorV4	internal	Passed	No Issue
107	_buildDomainSeparator	read	Passed	No Issue
108	_hashTypedDataV4	internal	Passed	No Issue
109	eip712Domain	read	Passed	No Issue
110	_EIP712Name	internal	Passed	No Issue
111	_EIP712Version	internal	Passed	No Issue
112	_EIP712NameHash	internal	Passed	No Issue
113	_EIP712VersionHash	internal	Passed	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

(1) Logical vulnerability:

Users can claim rewards for his balance although he has not staked. This was the reason for the exploit on 2023-05-22.

Resolution: We suggest rewards should be given only for the staked amount of the user. By this we can avoid the hack that happened recently.

Status: This issue is fixed in the revised contract code.

High Severity

(1) Rewards is not given before unstake:

After stake, the user claims the rewards and then unstake. At this point, the user can see his pending rewards and dept rewards but he cannot withdraw it. On restake all the pending and dept rewards reset to 0 and no rewards transferred to the user.

Resolution: We suggest giving rewards before the whole unstake.

Status: This issue is fixed in the revised contract code.

(2) Bets can be overwritten:

In the storeBets function is used to store the bets, but there is no check for the existing bet id. Hence the bets can be overwritten if the same Id used for it.

Resolution: We suggest validating for the bet id if it exists or not.

Status: This issue is fixed in the revised contract code.

(3) A bet can be settled more than once:

In the settleBets, bet can be set more than once. No check for an already settled bet.

Resolution: We suggest validating the bet before settling.

Status: This issue is fixed in the revised contract code.

Medium

No medium severity vulnerabilities were found.

Low

(1) Infinite loops possibility:

As array elements will increase, then it will cost more and more gas. And eventually, it will stop all the functionality. After several hundreds of transactions, all those functions depending on it will stop.
Below functions are having issues:

- storeBets
- updateBets
- settleBets

Resolution: We suggest validating the array length before executing for loop to avoid out of gas issue.

Status: This issue is fixed in the revised contract code.

Very Low / Informational / Best practices:

(1) If totalsupply reaches to 0 no one can stake:

If initIpTokenPrice is not set before the totalSupply reaches to zero, then no one can stake.

Resolution: We suggest setting initIpTokenPrice after initializing the staking contract.

Status: This issue is acknowledged in the revised contract code.

(2) Unused variable:

Variable: INT_ACC_REWARD_PRECISION

```
bytes32 public constant STAKING_MANAGER = keccak256("STAKING_MANAGER");
int256 private constant INT_ACC_REWARD_PRECISION = 1e18;
struct ValuesOfInterest {
    int256 expectedValue;
    int256 maxExposure;
    uint256 deadline;
    address signer;
}
```

Function: farmUtil()

```
///@param _amount amount
function farmUtil(uint256 _amount) internal {
    // Farm Related Logic
    cleanUserMapping();
    FarmInfo memory farm = updateFarm();
    UserInfo storage user = userInfo[msg.sender];
    if (balanceOf(msg.sender) > 0) {
        uint256 pending = uint256(
```

The INT_ACC_REWARD_PRECISION variable has been defined as a constant but is not used anywhere. In the farmUtil function, the input parameter _amount is not used.

Resolution: We suggest removing unused variables / parameters.

Status: This issue is fixed in the revised contract code.

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:

VLFI.sol

- setPoolName: The poolName can be set by the manager.
- setDecimals: The pool decimals are set by the manager.
- initLpTokenPrice: The initial pool token price can be set by the manager.
- setTreasuryAddress: Treasury address can be set by the manager.
- setSportsBookContract: Sports Book Contract address can be set by the manager.
- setCoolDownActiveState: Cooldown active state can be set by the manager.
- setUnstakeWindowTime: Unstake window time can be set by the staking manager.
- setCooldownSeconds: Cooldown seconds can be set by the staking manager.
- setRewardPerSecond: Rewards per second can be set by the staking manager.
- setVOI: VOI can be set by the data provider oracle manager.
- storeBets: Store the bets placed by the house pool data provider.
- updateBets: Update the bets information by the house pool data provider.
- settleBets: Settle bets by the house pool data provider.
- withdrawToTreasury: Manager can withdraw the funds to the treasury.

AccessControlUpgradeable.sol

- grantRole: Grants `role` to `account` can be set by the owner.
- revokeRole: Revokes `role` from `account` by the owner.
- renounceRole: Renounce Role from `account` by the owner.

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 a file and we have used all possible tests based on given objects as files. We have observed 1 critical severity issue, 3 high severity issues, 1 low severity issue and 2 informational severity issues in smart contracts. These issues are fixed / acknowledged in the revised smart contract code. **So, the smart contracts are ready for mainnet deployment.**

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.

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

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

Email: audit@EtherAuthority.io

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).

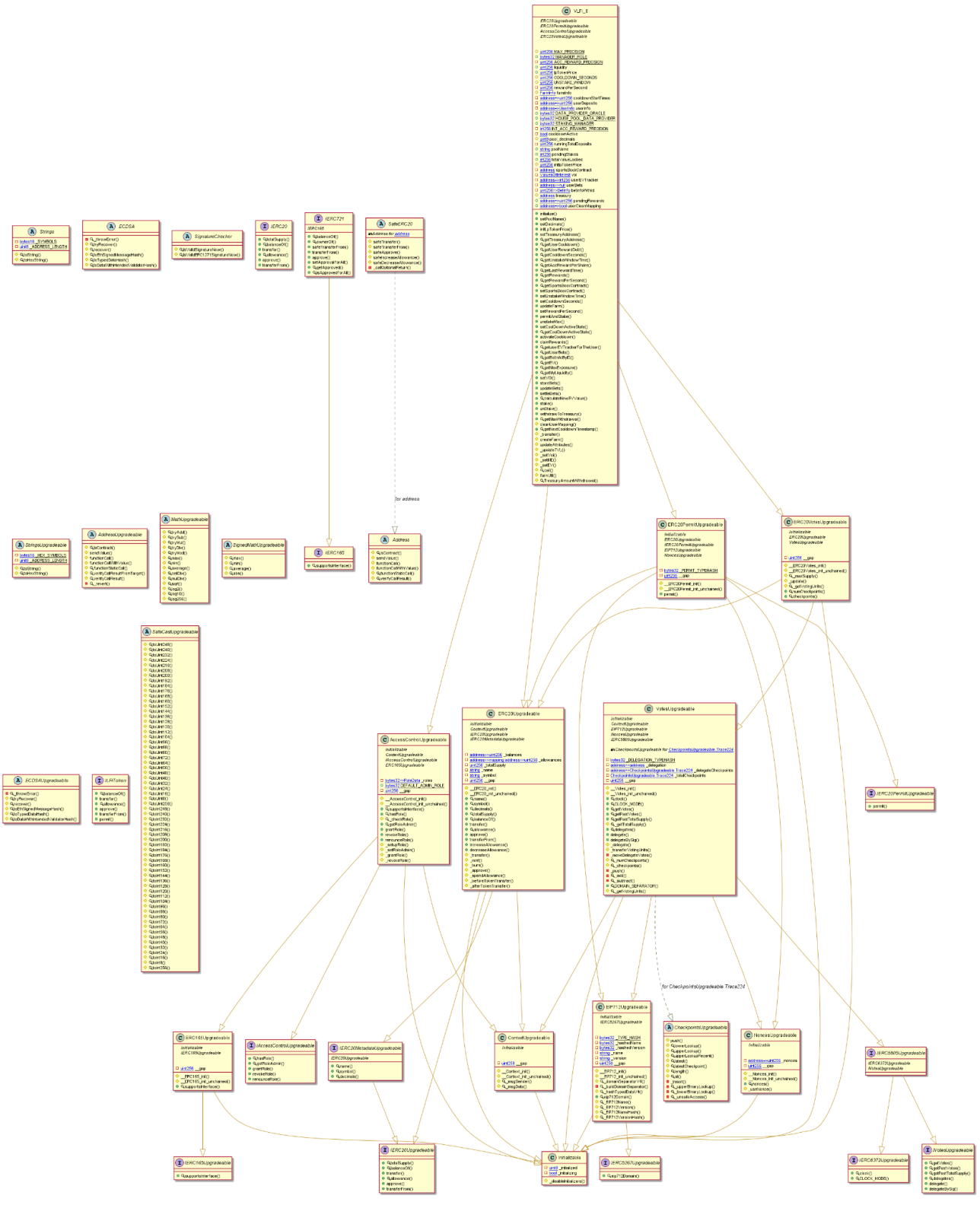
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we 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 - LunaFi Token



Slither Results Log

Slither Log >> VLFI.sol

```
ERC20PermitUpgradeable.__ERC20Permit_init(string).name (VLFI.sol#1368) shadows:
        - ERC20Upgradeable.name() (VLFI.sol#1207-1209) (function)
        - IERC20MetadataUpgradeable.name() (VLFI.sol#1059) (function)
VLFI_8.initialize(string,string,uint256,uint256,uint256).name (VLFI.sol#2547) shadows:
        - ERC20Upgradeable.name() (VLFI.sol#1207-1209) (function)
        - IERC20MetadataUpgradeable.name() (VLFI.sol#1059) (function)
VLFI_8.initialize(string,string,uint256,uint256,uint256).symbol (VLFI.sol#2548) shadows:
        - ERC20Upgradeable.symbol() (VLFI.sol#1211-1213) (function)
        - IERC20MetadataUpgradeable.symbol() (VLFI.sol#1061) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

VLFI_8.initialize(string,string,uint256,uint256,uint256) (VLFI.sol#2546-2565) should emit an event for:
        - COOLDOWN_SECONDS = cooldownSeconds (VLFI.sol#2557)
        - UNSTAKE_WINDOW = unstakeWindow (VLFI.sol#2558)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

VLFI_8.setTreasuryAddress(address)._treasuryAddress (VLFI.sol#2584) lacks a zero-check on :
                - treasury = _treasuryAddress (VLFI.sol#2586)
VLFI_8.setSportsBookContract(address)._sportsAddress (VLFI.sol#2654) lacks a zero-check on :
                - sportsBookContract = _sportsAddress (VLFI.sol#2656)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Variable 'VotesUpgradeable._moveDelegateVotes(address,address,uint256).oldValue (VLFI.sol#2212)' in VotesUpgradeable._moveDele
gateVotes(address,address,uint256) (VLFI.sol#2209-2228) potentially used before declaration: (oldValue,newValue) = _push(_dele
gateCheckpoints[to],_add,SafeCastUpgradeable.toUint224(amount)) (VLFI.sol#2220-2224)
Variable 'VotesUpgradeable._moveDelegateVotes(address,address,uint256).newValue (VLFI.sol#2212)' in VotesUpgradeable._moveDele
gateVotes(address,address,uint256) (VLFI.sol#2209-2228) potentially used before declaration: (oldValue,newValue) = _push(_dele
gateCheckpoints[to],_add,SafeCastUpgradeable.toUint224(amount)) (VLFI.sol#2220-2224)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables
```

```
ERC20PermitUpgradeable.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (VLFI.sol#1374-1393) uses timestamp for c
omparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= deadline,ERC20Permit: expired deadline) (VLFI.sol#1383)
VotesUpgradeable.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (VLFI.sol#2172-2189) uses timestamp for comparis
ons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= expiry,Votes: signature expired) (VLFI.sol#2180)
VLFI_8.getRewards(address) (VLFI.sol#2620-2643) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp > farm.lastRewardTime && supply != 0 (VLFI.sol#2627)
VLFI_8.updateFarm() (VLFI.sol#2671-2685) uses timestamp for comparisons
        Dangerous comparisons:
        - farm.lastRewardTime < block.timestamp (VLFI.sol#2673)
VLFI_8.unstakeMax() (VLFI.sol#2713-2777) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)((block.timestamp) > (cooldownStartTimestamp + (COOLDOWN_SECONDS)),HOUSEPOOL:COOLDOWN_NOT_COMPLE
TE) (VLFI.sol#2718-2722)
        - require(bool,string)(block.timestamp - (cooldownStartTimestamp + (COOLDOWN_SECONDS)) <= UNSTAKE_WINDOW,HOUSEPOOL:UNS
TAKE_WINDOW_FINISHED) (VLFI.sol#2723-2728)
VLFI_8.unStake(address,uint256) (VLFI.sol#2940-3011) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)((block.timestamp) > (cooldownStartTimestamp + (COOLDOWN_SECONDS)),HOUSEPOOL:COOLDOWN_NOT_COMPLE
TE) (VLFI.sol#2944-2948)
        - require(bool,string)(block.timestamp - (cooldownStartTimestamp + (COOLDOWN_SECONDS)) <= UNSTAKE_WINDOW,HOUSEPOOL:UNS
TAKE_WINDOW_FINISHED) (VLFI.sol#2949-2954)
VLFI_8.getNextCooldownTimestamp(uint256,uint256,address,uint256) (VLFI.sol#3059-3092) uses timestamp for comparisons
        Dangerous comparisons:
        - toCooldownTimestamp > 36322041600 (VLFI.sol#3066)
        - toCooldownTimestamp == 0 (VLFI.sol#3069)
        - minimalValidCooldownTimestamp > toCooldownTimestamp (VLFI.sol#3074)
        - fromCooldownTimestamp < toCooldownTimestamp (VLFI.sol#3081)
        - (minimalValidCooldownTimestamp > userCooldownTimestamp) (VLFI.sol#3077-3080)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

```
Strings.toString(uint256) (VLFI.sol#9-25) uses assembly
        - INLINE ASM (VLFI.sol#17-19)
ECDSA.tryRecover(bytes32,bytes) (VLFI.sol#68-82) uses assembly
        - INLINE ASM (VLFI.sol#73-77)
ECDSA.toEthSignedMessageHash(bytes32) (VLFI.sol#121-127) uses assembly
        - INLINE ASM (VLFI.sol#122-126)
ECDSA.toTypedDataHash(bytes32,bytes32) (VLFI.sol#133-141) uses assembly
        - INLINE ASM (VLFI.sol#134-140)
Address.verifyCallResult(bool,bytes,string) (VLFI.sol#294-312) uses assembly
        - INLINE ASM (VLFI.sol#304-307)
AddressUpgradeable._revert(bytes,string) (VLFI.sol#519-528) uses assembly
        - INLINE ASM (VLFI.sol#521-524)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (VLFI.sol#694-743) uses assembly
        - INLINE ASM (VLFI.sol#698-702)
        - INLINE ASM (VLFI.sol#712-717)
        - INLINE ASM (VLFI.sol#721-727)
ECDSAUpgradeable.tryRecover(bytes32,bytes) (VLFI.sol#939-953) uses assembly
        - INLINE ASM (VLFI.sol#944-948)
ECDSAUpgradeable.toEthSignedMessageHash(bytes32) (VLFI.sol#992-998) uses assembly
        - INLINE ASM (VLFI.sol#993-997)
ECDSAUpgradeable.toTypedDataHash(bytes32,bytes32) (VLFI.sol#1004-1012) uses assembly
        - INLINE ASM (VLFI.sol#1005-1011)
CheckpointsUpgradeable._unsafeAccess(CheckpointsUpgradeable.Checkpoint224[],uint256) (VLFI.sol#1916-1924) uses assembly
        - INLINE ASM (VLFI.sol#1920-1923)
CheckpointsUpgradeable._unsafeAccess(CheckpointsUpgradeable.Checkpoint160[],uint256) (VLFI.sol#2048-2056) uses assembly
        - INLINE ASM (VLFI.sol#2052-2055)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```

```
VLFI_8.getRewards(address) (VLFI.sol#2620-2643) compares to a boolean constant:
        -userCleanMapping[_benefiter] != true (VLFI.sol#2637)
VLFI_8.cleanUserMapping() (VLFI.sol#3051-3057) compares to a boolean constant:
        -userCleanMapping[msg.sender] != true (VLFI.sol#3052)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
```

```
Pragma version0.8.17 (VLFI.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.16
solc-0.8.17 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (VLFI.sol#236-241):
        - (success) = recipient.call{value: amount}() (VLFI.sol#239)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (VLFI.sol#263-274):
        - (success,returndata) = target.call{value: value}(data) (VLFI.sol#272)
Low level call in Address.functionStaticCall(address,bytes,string) (VLFI.sol#280-289):
        - (success,returndata) = target.staticcall(data) (VLFI.sol#287)
Low level call in AddressUpgradeable.sendValue(address,uint256) (VLFI.sol#440-445):
        - (success) = recipient.call{value: amount}() (VLFI.sol#443)
Low level call in AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (VLFI.sol#467-476):
        - (success,returndata) = target.call{value: value}(data) (VLFI.sol#474)
Low level call in AddressUpgradeable.functionStaticCall(address,bytes,string) (VLFI.sol#482-489):
        - (success,returndata) = target.staticcall(data) (VLFI.sol#487)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Function ContextUpgradeable.__Context_init() (VLFI.sol#601-602) is not in mixedCase
Function ContextUpgradeable.__Context_init_unchained() (VLFI.sol#604-605) is not in mixedCase
Variable ContextUpgradeable.__gap (VLFI.sol#614) is not in mixedCase
Function NoncesUpgradeable.__Nonces_init() (VLFI.sol#1067-1068) is not in mixedCase
Function NoncesUpgradeable.__Nonces_init_unchained() (VLFI.sol#1070-1071) is not in mixedCase
Variable NoncesUpgradeable.__gap (VLFI.sol#1084) is not in mixedCase
Function EIP712Upgradeable.__EIP712_init(string,string) (VLFI.sol#1097-1099) is not in mixedCase
Function EIP712Upgradeable.__EIP712_init_unchained(string,string) (VLFI.sol#1101-1107) is not in mixedCase
Function EIP712Upgradeable._EIP712Name() (VLFI.sol#1149-1151) is not in mixedCase
Function EIP712Upgradeable._EIP712Version() (VLFI.sol#1153-1155) is not in mixedCase
Function EIP712Upgradeable._EIP712NameHash() (VLFI.sol#1157-1169) is not in mixedCase
Function EIP712Upgradeable._EIP712VersionHash() (VLFI.sol#1171-1183) is not in mixedCase
Variable EIP712Upgradeable.__gap (VLFI.sol#1185) is not in mixedCase
Function ERC20Upgradeable.__ERC20_init(string,string) (VLFI.sol#1198-1200) is not in mixedCase
Function ERC20Upgradeable.__ERC20_init_unchained(string,string) (VLFI.sol#1202-1205) is not in mixedCase
Variable ERC20Upgradeable.__gap (VLFI.sol#1362) is not in mixedCase
Function ERC20PermitUpgradeable.__ERC20Permit_init(string) (VLFI.sol#1368-1370) is not in mixedCase
```

```
Function ERC20PermitUpgradeable.__ERC20Permit_init_unchained(string) (VLFI.sol#1372) is not in mixedCase
Variable ERC20PermitUpgradeable.__gap (VLFI.sol#1403) is not in mixedCase
Function IERC6372Upgradeable.CLOCK_MODE() (VLFI.sol#2115) is not in mixedCase
Function VotesUpgradeable.__Votes_init() (VLFI.sol#2120-2121) is not in mixedCase
Function VotesUpgradeable.__Votes_init_unchained() (VLFI.sol#2123-2124) is not in mixedCase
Function VotesUpgradeable.CLOCK_MODE() (VLFI.sol#2140-2143) is not in mixedCase
Function VotesUpgradeable.DOMAIN_SEPARATOR() (VLFI.sol#2257-2259) is not in mixedCase
Variable VotesUpgradeable.__gap (VLFI.sol#2263) is not in mixedCase
Function ERC20VotesUpgradeable.__ERC20Votes_init() (VLFI.sol#2268-2269) is not in mixedCase
Function ERC20VotesUpgradeable.__ERC20Votes_init_unchained() (VLFI.sol#2271-2272) is not in mixedCase
Variable ERC20VotesUpgradeable.__gap (VLFI.sol#2297) is not in mixedCase
Function ERC165Upgradeable.__ERC165_init() (VLFI.sol#2301-2302) is not in mixedCase
Function ERC165Upgradeable.__ERC165_init_unchained() (VLFI.sol#2304-2305) is not in mixedCase
Variable ERC165Upgradeable.__gap (VLFI.sol#2310) is not in mixedCase
Function AccessControlUpgradeable.__AccessControl_init() (VLFI.sol#2314-2315) is not in mixedCase
Function AccessControlUpgradeable.__AccessControl_init_unchained() (VLFI.sol#2317-2318) is not in mixedCase
Variable AccessControlUpgradeable.__gap (VLFI.sol#2402) is not in mixedCase
Contract VLFI_8 (VLFI.sol#2407-3220) is not in CapWords
Parameter VLFI_8.setPoolName(string)._poolName (VLFI.sol#2568) is not in mixedCase
Parameter VLFI_8.setDecimals(uint8)._decimals (VLFI.sol#2573) is not in mixedCase
Parameter VLFI_8.initLpTokenPrice(uint256)._initLpTokenPrice (VLFI.sol#2578) is not in mixedCase
Parameter VLFI_8.setTreasuryAddress(address)._treasuryAddress (VLFI.sol#2584) is not in mixedCase
Parameter VLFI_8.getUserCooldown(address)._staker (VLFI.sol#2594) is not in mixedCase
Parameter VLFI_8.getUserRewardDebt(address)._benefiter (VLFI.sol#2599) is not in mixedCase
Parameter VLFI_8.getRewards(address)._benefiter (VLFI.sol#2621) is not in mixedCase
Parameter VLFI_8.setSportsBookContract(address)._sportsAddress (VLFI.sol#2654) is not in mixedCase
Parameter VLFI_8.setUnstakeWindowTime(uint256)._unstakeWindow (VLFI.sol#2660) is not in mixedCase
Parameter VLFI_8.setCooldownSeconds(uint256)._coolDownSeconds (VLFI.sol#2666) is not in mixedCase
Parameter VLFI_8.setRewardPerSecond(uint256)._rewardPerSecond (VLFI.sol#2688) is not in mixedCase
Parameter VLFI_8.permitAndStake(address,address,address,uint256).LFIamount (VLFI.sol#2699) is not in mixedCase
Parameter VLFI_8.setCoolDownActiveState(bool)._active (VLFI.sol#2780) is not in mixedCase
Parameter VLFI_8.getuserEVTrackerForTheUser(address)._user (VLFI.sol#2806) is not in mixedCase
Parameter VLFI_8.getUserBets(address)._user (VLFI.sol#2812) is not in mixedCase
Parameter VLFI_8.getBetInfoByID(uint256)._ID (VLFI.sol#2818) is not in mixedCase
Parameter VLFI_8.getMyLiquidity(address)._user (VLFI.sol#2843) is not in mixedCase
Parameter VLFI_8.updateBets(uint256[],uint256[])._Id (VLFI.sol#2881) is not in mixedCase
Parameter VLFI_8.updateBets(uint256[],uint256[])._payout (VLFI.sol#2882) is not in mixedCase
```

```
Parameter VLFI_8.calculateNewEVValue(uint256,int256,address)._newAmount (VLFI.sol#2904) is not in mixedCase
Parameter VLFI_8.calculateNewEVValue(uint256,int256,address)._newEV (VLFI.sol#2905) is not in mixedCase
Parameter VLFI_8.calculateNewEVValue(uint256,int256,address)._toUser (VLFI.sol#2906) is not in mixedCase
Parameter VLFI_8.withdrawToTreasury(uint256)._withdrawalAmount (VLFI.sol#3015) is not in mixedCase
Parameter VLFI_8.getMaxWithdrawal(address)._user (VLFI.sol#3026) is not in mixedCase
Function VLFI_8.TreasuryAmountWithdrawal() (VLFI.sol#3216-3218) is not in mixedCase
Variable VLFI_8.COOLDOWN_SECONDS (VLFI.sol#2420) is not in mixedCase
Variable VLFI_8.UNSTAKE_WINDOW (VLFI.sol#2421) is not in mixedCase
Variable VLFI_8.pool_decimals (VLFI.sol#2480) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Variable VLFI_8.COOLDOWN_SECONDS (VLFI.sol#2420) is too similar to VLFI_8.setCooldownSeconds(uint256)._coolDownSeconds (VLFI.s
ol#2666)
Variable VLFI_8.UNSTAKE_WINDOW (VLFI.sol#2421) is too similar to VLFI_8.setUnstakeWindowTime(uint256)._unstakeWindow (VLFI.sol
#2660)
Variable VLFI_8.setRewardPerSecond(uint256)._rewardPerSecond (VLFI.sol#2688) is too similar to VLFI_8.initialize(string,string
,uint256,uint256,uint256).rewardsPerSecond (VLFI.sol#2552)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar

VLFI_8.userDeposits (VLFI.sol#2437) is never used in VLFI_8 (VLFI.sol#2407-3220)
VLFI_8.INT_ACC_REWARD_PRECISION (VLFI.sol#2461) is never used in VLFI_8 (VLFI.sol#2407-3220)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable

VLFI_8.runningTotalDeposits (VLFI.sol#2481) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
VLFI.sol analyzed (36 contracts with 84 detectors), 290 result(s) found
```

Solidity Static Analysis

VLFI.sol

Security

Inline assembly:

The Contract uses inline assembly, this is only advised in rare cases. Additionally static analysis modules do not parse inline Assembly, this can lead to wrong analysis results.

more

Pos: 1986:8:

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: 1383:16:

Low level calls:

Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return value is not handled properly. Please use Direct Calls via specifying the called contract's interface.

more

Pos: 474:50:

Gas & Economy

Gas costs:

Gas requirement of function VLFI_8.storeBets 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: 2788:4:

Email: audit@EtherAuthority.io

Gas costs:

Gas requirement of function VLFI_8.updateBets 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: 2814:4:

Gas costs:

Gas requirement of function VLFI_8.settleBets 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: 2826:4:

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: 2819:8:

Miscellaneous

Similar variable names:

VLFI_8._transfer(address,address,uint256) : Variables have very similar names "farm" and "from". Note: Modifiers are currently not considered by this static analysis.
Pos: 3068:24:

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: 2636:8:

Solhint Linter

VLFI.sol

```
VLFI.sol:10:18: Error: Parse error: missing ';' at '{'
VLFI.sol:362:18: Error: Parse error: missing ';' at '{'
VLFI.sol:641:18: Error: Parse error: missing ';' at '{'
VLFI.sol:649:18: Error: Parse error: missing ';' at '{'
VLFI.sol:656:18: Error: Parse error: missing ';' at '{'
VLFI.sol:665:18: Error: Parse error: missing ';' at '{'
VLFI.sol:672:18: Error: Parse error: missing ';' at '{'
VLFI.sol:695:18: Error: Parse error: missing ';' at '{'
VLFI.sol:760:18: Error: Parse error: missing ';' at '{'
VLFI.sol:773:18: Error: Parse error: missing ';' at '{'
VLFI.sol:781:18: Error: Parse error: missing ';' at '{'
VLFI.sol:818:18: Error: Parse error: missing ';' at '{'
VLFI.sol:826:18: Error: Parse error: missing ';' at '{'
VLFI.sol:859:18: Error: Parse error: missing ';' at '{'
VLFI.sol:867:18: Error: Parse error: missing ';' at '{'
VLFI.sol:892:18: Error: Parse error: missing ';' at '{'
VLFI.sol:913:18: Error: Parse error: missing ';' at '{'
VLFI.sol:1079:18: Error: Parse error: missing ';' at '{'
VLFI.sol:1264:18: Error: Parse error: missing ';' at '{'
VLFI.sol:1283:18: Error: Parse error: missing ';' at '{'
VLFI.sol:1299:18: Error: Parse error: missing ';' at '{'
VLFI.sol:1314:18: Error: Parse error: missing ';' at '{'
VLFI.sol:1344:22: Error: Parse error: missing ';' at '{'
```

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