Bumper Finance

Staking, Vesting, Airdrop

SMART CONTRACT AUDIT

21.12.2021

Made in Germany by Chainsulting.de

Table of contents

1. Disclaimer ... 3
2. About the Project and Company ... 4
   2.1 Project Overview ... 5
3. Vulnerability & Risk Level ... 6
4. Auditing Strategy and Techniques Applied ... 7
   4.1 Methodology ... 7
   4.2 Used Code from other Frameworks/Smart Contracts ... 8
   4.3 Tested Contract Files ... 10
   4.4 Metrics / CallGraph ... 11
   4.5 Metrics / Source Lines & Risk ... 12
   4.6 Metrics / Capabilities ... 13
   4.7 Metrics / Source Unites in Scope ... 14
5. Scope of Work ... 15
   5.1 Manual and Automated Vulnerability Test ... 16
   5.1.1 Locking and no rewards scenario ... 16
   5.1.2 Public functions should be declared external ... 17
   5.1.3 Floating compiler versions ... 18
   5.1.4 Using same require checks ... 19
   5.2 SWC Attacks ... 20
   5.3 Verify Claims ... 24
   5.3.1. Staking ... 24
   5.3.2. Vesting ... 25

5.3.3. Airdrop ... 26
6. Executive Summary ... 26
7. Deployed Smart Contract ... 27

1. Disclaimer

The audit makes no statements or warrantees about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of Bumper Finance. If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

Major Versions / Date	Description
0.1 (02.12.2021)	Layout
0.2 (03.12.2021)	Test Deployment
0.5 (03.12.2021)	Automated Security Testing Manual Security Testing
0.6 (03.12.2021)	Testing SWC Checks
0.7 (03.12.2021)	Verify Claims
0.9 (03.12.2021)	Summary and Recommendation
1.0 (03.12.2021)	Final document
1.1 (14.12.2021)	Re-check commit 22276afba19b9bad87150db7d62ff1455773881a
1.2 (21.12.2021)	Added deployed contract addresses

2. About the Project and Company

Website: https://www.bumper.fi

Twitter: https://twitter.com/bumperfinance

Medium: https://medium.com/bumper-finance

Telegram: https://t.me/bumperfinance

Discord: https://discord.gg/YyzRws4Ujd

Github: https://github.com/Bumper-Fi

2.1 Project Overview

Bumper is a DeFi price protection protocol built on Ethereum which protects the price of crypto assets. Bumper provides a decentralised software facility for ‘Takers’ of protection to operate diametrically to ‘Makers’ of liquidity. Protected positions incur a floating daily premium, nominally 3% p.a, that is used to incentivise stablecoin depositors into a risk-free liquidity Reserve.

The Bumper protocol is a pure, decentralised market for on-chain asset price risk, which is transferred from a stablecoin Reserve through to cascading redundancy modules. At any point in time the Reserve has a measurable aggregate liability representing all positions. Should the liability exceed parameterized safety levels, the protocol rebalances, firstly by utilising first order dynamics, such as Premium/ Yield curves/ BUMP distributions and then by opening up to arbitrageur bots and if necessary DEX’s. A separate risk pool, attracting a higher yield tranche, acts to backstop any realized losses caused by sharp volatility.

Conclusively, these redundancy measures make Bumper a highly productive tool to achieve efficient risk transfer pricing via liability pooling.

3. Vulnerability & Risk Level

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 – 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon as possible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

4. Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as there were discovered.

4.1 Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
   i. Review of the specifications, sources, and instructions provided to Chainsulting to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Chainsulting describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

4.2 Used Code from other Frameworks/Smart Contracts (direct imports)

Dependency / Import Path	Source
@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/v4.3.1/contracts/access/OwnableUpgradeable.sol
@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.3.1/contracts/proxy/utils/Initializable.sol
@openzeppelin/contracts-upgradeable/security/PausableUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.3.1/contracts/security/PausableUpgradeable.sol
@openzeppelin/contracts-upgradeable/security/ReentrancyGuardUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.3.1/contracts/security/ReentrancyGuardUpgradeable.sol
@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.3.1/contracts/token/ERC20/IERC20Upgradeable.sol
@openzeppelin/contracts-upgradeable/token/ERC20/extensions/ERC20PausableUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.3.1/contracts/token/ERC20/extensions/ERC20PausableUpgradeable.sol
@openzeppelin/contracts-upgradeable/token/ERC20/extensions/draft-IERC20PermitUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.3.1/contracts/token/ERC20/extensions/draft-IERC20PermitUpgradeable.sol
@openzeppelin/contracts-upgradeable/token/ERC20/utils/SafeERC20Upgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.3.1/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol
@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.3.1/contracts/utils/ContextUpgradeable.sol

@openzeppelin/contracts-upgradeable/utils/cryptography/MerkleProofUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.3.1/contracts/utils/cryptography/MerkleProofUpgradeable.sol
@openzeppelin/contracts/access/Ownable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.2.0/contracts/access/Ownable.sol
@openzeppelin/contracts/security/Pausable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.2.0/contracts/security/Pausable.sol
@openzeppelin/contracts/token/ERC20/ERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.2.0/contracts/token/ERC20/ERC20.sol
@openzeppelin/contracts/token/ERC20/IERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.2.0/contracts/token/ERC20/IERC20.sol
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.2.0/contracts/token/ERC20/utils/SafeERC20.sol
@openzeppelin/contracts/utils/cryptography/MerkleProof.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.2.0/contracts/utils/cryptography/MerkleProof.sol

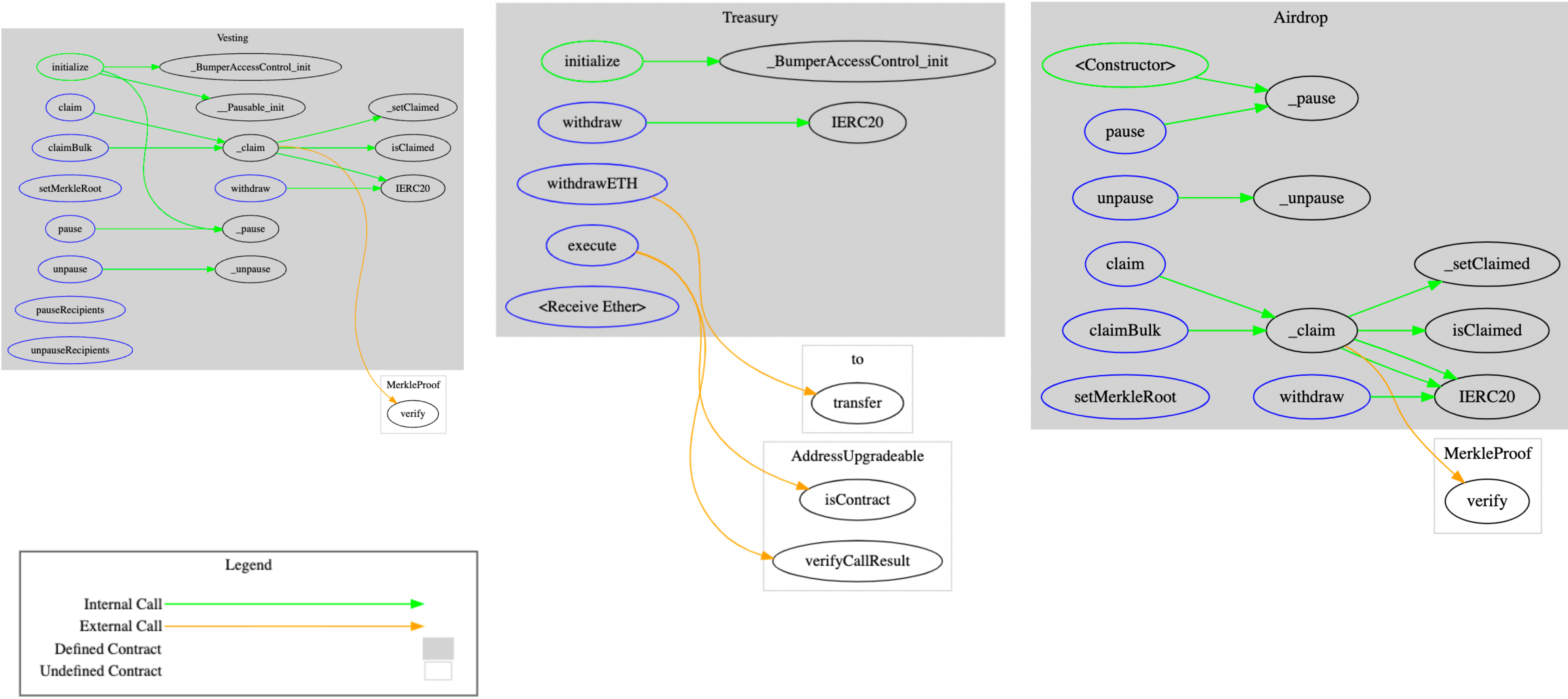
4.3 Tested Contract Files

The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review

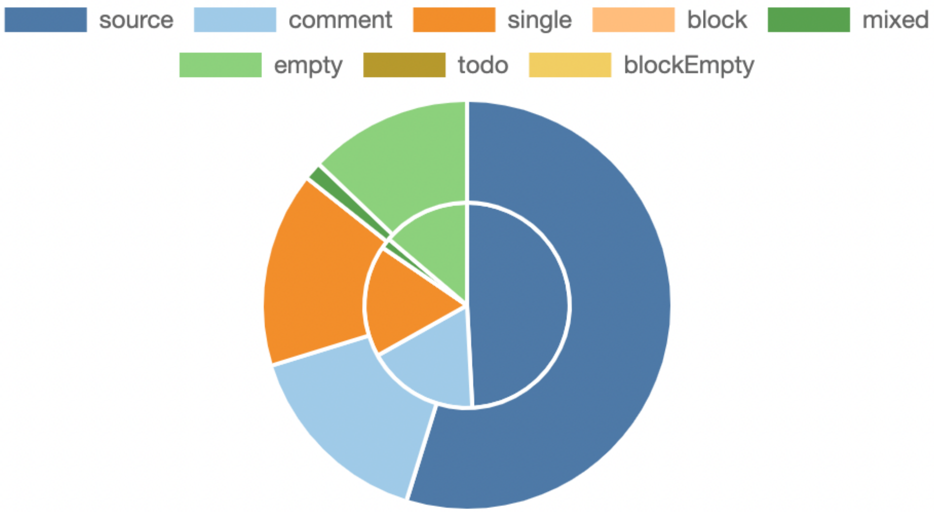
File	Fingerprint (MD5)
./contracts/interfaces/ITreasury.sol	8e3601f45dd9faf79d93acb62257060d
./contracts/interfaces/IVault.sol	cce9fc4325cb9edbcc50491ed2381892
./contracts/interfaces/IStakeChangedReceiver.sol	1c18f72d79db91c1998079896bd2cd76
./contracts/interfaces/IBumpToken.sol	17859a99b1de78a8cb96f9d026d45ae5
./contracts/treasury/Treasury.sol	c36094ffcdd8445781135fc65d23b3e7
./contracts/airdrop/Airdrop.sol	73d0e166a31f8c82248ae0f0384466e3
./contracts/staking/StakeRewards.sol	1f33771504fbad6b3bb313f5165ef807
./contracts/vesting/Vesting.sol	ba69ccd7bb43319511d4a1028efb5401
./contracts/TimeLockMechanism.sol	6c2f43a7a1a1f733346242651583bc97
./contracts/access/BumperAccessControl.sol	4319078266112e33c20573e2442c351f
./contracts/token/BUMPTokenV2.sol	571b3c3fc0a5967b51829626decac623
./contracts/token/BUMPTokenPre.sol	d9c70f11d6f77b5d3a5b821b247babe5
./contracts/access/Blacklistable.sol	6ab7abb83b3ae1fa8b483f984a6bc6d6

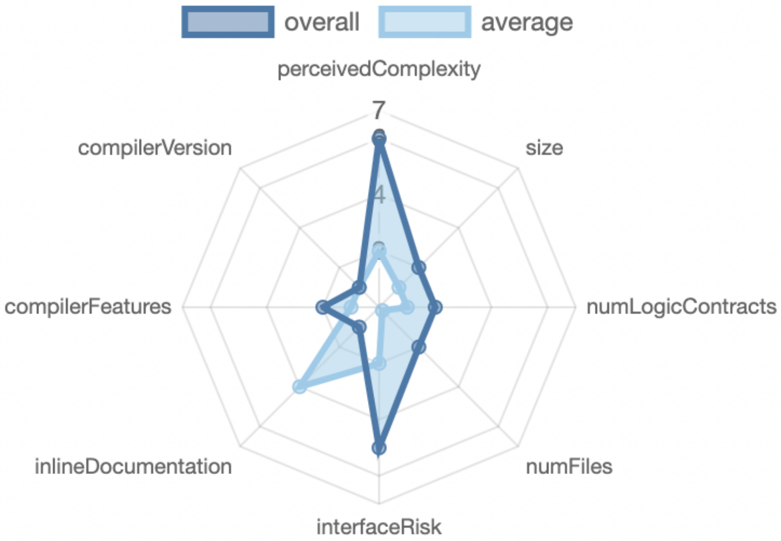
Commit: 22276afba19b9bad87150db7d62ff1455773881a

4.4 Metrics / CallGraph



4.5 Metrics / Source Lines & Risk





4.6 Metrics / Capabilities

Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥 Uses Assembly	💣 Has Destroyable Contracts

`^0.8.4`
`^0.8.0`
`^0.8.2`
`>=0.8.0 <0.9.0`
	`ABIEncoderV2`	`yes`	`yes`
			(1 asm blocks)	

📤 Transfers ETH	⚡ Low-Level Calls	👥 DelegateCall	🧮 Uses Hash Functions	🔖 ECRecover	🌀 New/Create/Create2
`yes`			`yes`		

Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

🌐Public	💰Payable
67	5

External	Internal	Private	Pure	View
53	59	1	2	19

StateVariables

Total	🌐Public
18	15

4.7 Metrics / Source Unites in Scope

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
🔍	contracts/interfaces/IStakeChangedReceiver.sol		1	6	5	3	1	3	
🔍	contracts/interfaces/IBumpToken.sol		1	12	6	3	1	13	
🔍	contracts/interfaces/IVault.sol		1	29	6	3	1	25	
🔍	contracts/interfaces/ITreasury.sol		1	12	6	3	1	13	💰
📝	contracts/airdrop/Airdrop.sol	1		168	133	76	38	62	🧮
📝	contracts/treasury/Treasury.sol	1		69	43	24	9	36	💰📤
📝	contracts/staking/StakeRewards.sol	1		442	410	272	79	215	
📝	contracts/vesting/Vesting.sol	1		186	160	87	48	79	🧮
📝	contracts/TimeLockMechanism.sol	1		31	28	20	3	6	
📝	contracts/access/BumperAccessControl.sol	1		60	57	39	9	30	
📝	contracts/token/BUMPTokenV2.sol	1		100	67	40	18	51	🧪📤
📝	contracts/token/BUMPTokenPre.sol	1		165	143	92	34	130	🖥🧪📤

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝🔍	Totals	8	4	1280	1064	662	242	663	🖥️🧪💰📤🧮

5. Scope of Work

The Bumper Finance Team provided us with the files that needs to be tested. The scope of the audit are the bumper staking, vesting and airdrop contracts.

The team put forward the following assumptions regarding the security, usage of the contracts:

- The smart contract is coded according to the newest standards and in a secure way

Staking:

- Contract owner is not able to withdraw stakes
- Contract owner is not able to freeze or pause stakes
- User stake rewards are correctly calculated
- Users are able to withdraw stakes after period ends

Vesting:

- Contract owner is not able to withdraw vestings
- Contract owner is not able to freeze or pause vestings
- Vesting periods are correctly calculated
- Users are able to withdraw vestings after period ends

Airdrop:

- Claiming of airdrops is working correctly

The main goal of this audit was to verify these claims. The auditors can provide additional feedback on the code upon the client's request.

5.1 Manual and Automated Vulnerability Test

CRITICAL ISSUES

During the audit, Chainsulting‘s experts found **no Critical issues** in the code of the smart contract.

HIGH ISSUES

During the audit, Chainsulting’s experts found **no High issues** in the code of the smart contract.

MEDIUM ISSUES

5.1.1 Locking and no rewards scenario

Severity: MEDIUM
Status: ACKNOWLEDGED
File(s) affected: Vesting.sol, Airdrop.sol

Attack / Description	Code Snippet	Result/Recommendation

If there is no reward sent to the contract, the contract can run out of balance. There is no guarantee to claim rewards or be able to withdraw vested token, as there is a function to pause the contract and withdraw token for the Governance address.	Vesting.sol
	`function withdraw(`
	`        address _to,`
	`        address _toke,`
	`        uint256 _amount`
	`    ) external onlyGovernance {`
	`        IERC20(_toke).safeTransfer(_to, _amount);`
	`    }`
	Airdrop.sol
	`function withdraw(`
	`        address _to,`
	`        address _token,`
	`        uint256 _amount`
	`    ) external onlyOwner {`
		There are several options: 1. Only allow vesting, if the balance of the contract can guarantee the pay-out of rewards after locking period ends. 2. Use a multisig for the onlyGovernance functions, to guarantee no malicious usage of pause and withdraw functions.

	`IERC20(_token).safeTransfer(_to, _amount);` `}`	

LOW ISSUES

During the audit, Chainsulting's experts found **no Low issues** in the code of the smart contract.

INFORMATIONAL ISSUES

5.1.2 Public functions should be declared external

Severity: INFORMATIONAL
Status: ACKNOWLEDGED
Code: NA
File(s) affected: StakeRewards.sol, Vesting.sol

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation several functions are declared as public where they could be external. For public functions Solidity immediately copies array arguments to memory, while external functions can read directly from calldata. Because memory allocation is expensive, the gas consumption of public functions is higher.	StakeRewards.sol line 98 getUserStake() StakeRewards.sol line 107 getStakeOptions() Vesting.sol line 28 initialize()	We recommend declaring functions as external if they are not used internally. This leads to lower gas consumption and better code readability.

5.1.3 Floating compiler versions
Severity: INFORMATIONAL
Status: ACKNOWLEDGED
Code: SWC-103
File(s) affected: ALL

Attack / Description	Code Snippet	Result/Recommendation
The current pragma solidity directive is floating. It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.	pragma solidity ^0.8.0	It is recommended to follow the latter example, as future compiler versions may handle certain language constructions in a way the developer did not foresee. i.e. Pragma solidity 0.8.0 See SWC-103: https://swcregistry.io/docs/SWC-103

5.1.4 Using same require checks

Severity: INFORMATIONAL
Status: FIXED
Code: NA
File(s) affected: StakeRewardFixed.sol

Attack / Description	Code Snippet	Result/Recommendation

In the current implementation the same require checks are used multiple times at different places in the code.	StakeRewardFixed.sol line 132, 145, 168 & 226
	`require(amount > 0, "!amount");`
	
	StakeRewardFixed.sol line 131, 144 & 167
	`require(unlockTimestamp < uint64(block.timestamp), "!ended");`
		We recommend using modifiers for multiple used require checks to reduce code size and increase code readability. The two require checks can be combined into one modifier.

5.2 SWC Attacks

ID	Title	Relationships	Test Result
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	✅
SWC-130	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	✅
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator	✅
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	✅
SWC-127	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	✅
SWC-125	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	✅
SWC-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	✅
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller	✅

ID	Title	Relationships	Test Result
[SWC-122]	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	✅
SWC-121	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	✅
SWC-120	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	✅
SWC-119	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	✅
SWC-118	Incorrect Constructor Name	CWE-665: Improper Initialization	✅
SWC-117	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	✅
SWC-116	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✅
SWC-115	Authorization through tx.origin	CWE-477: Use of Obsolete Function	✅
SWC-114	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	✅

ID	Title	Relationships	Test Result
SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	✅
SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✅
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	✅
SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	✅
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	✅
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	✅
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	✅
SWC-106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	✅
SWC-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	✅
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	✅

ID	Title	Relationships	Test Result
[SWC-103]	Floating Pragma	[CWE-664: Improper Control of a Resource Through its Lifetime]	X
[SWC-102]	Outdated Compiler Version	[CWE-937: Using Components with Known Vulnerabilities]	✅
[SWC-101]	Integer Overflow and Underflow	[CWE-682: Incorrect Calculation]	✅
[SWC-100]	Function Default Visibility	[CWE-710: Improper Adherence to Coding Standards]	✅

5.3 Verify Claims

5.3.1. Staking

5.3.1.1. Contract owner is not able to withdraw stakes

In StakeRewardFixed.sol the withdrawing is strictly bound to the caller address and the owner is not able to withdraw users' funds. In StakeRewardFlexible.sol the governance can withdraw all funds from the staking contract. The *withdrawExtraTokens* function (line 354 – 359) requires an additional check for staking token address as it is done in StakeRewardFixed.sol line 379 – 387.

Update (14.12): Additional check is implemented in StakeRewards.sol commit: 22276afba19b9bad87150db7d62ff1455773881a

Status: tested and verified ✅

5.3.1.2. Contract owner is not able to freeze or pause stakes

The contract owner can influence the functionality of the smart contract by setting the unlock timestamp. The unlock timestamp defines the point of time when the users are able to stake. This means staking is disabled but withdrawing's are still working. Subsequently stakes cannot be frozen.

Status: tested and verified ✅

5.3.1.3 User stake rewards are correctly calculated

Status: tested and verified ✅

5.3.1.4. Users are able to withdraw stakes after period ends

The users can withdraw the stakes and corresponding rewards after the staking period ends.

Status: tested and verified ✅

5.3.2. Vesting

5.3.2.1. Contract owner is not able to withdraw user vestings

The funds are sent to the addresses defined in the merkle tree.

Status: tested and verified ✅

5.3.2.2. Contract owner is not able to freeze or pause vestings

All user actions can be disabled(paused) and enabled(unpaused) by the governance. It is also possible to disable specific users by the governance by calling pauseRecipient.

Status: tested and verified ✅

5.3.2.3. Vesting periods are correctly calculated

There is no vesting period calculated in the contract. The withdraw timestamp is stored in a merkle tree and verified by a merkle proof.

Status: tested and verified ✅

5.3.2.4. Users are able to withdraw vestings after period ends

Users are able to withdraw vestings if the timestamp stored in the merkle tree is reached and the contract has sufficient balance. There is no balance added initially. Someone has to send sufficient balance to the contract.

Status: tested and verified ✅

5.3.3. Airdrop

5.3.3.1. Claiming of airdrops is working correctly

Users are able to withdraw airdrops if the timestamp stored in the merkle tree is reached, the contract is unpaused and the contract has sufficient balance. The contract is paused by default an there is no balance added initially. Someone has to send sufficient balance to the contract and the governance has to unpause the contract for claims.

Status: tested and verified ✅

6. Executive Summary

Two (2) independent Chainsulting experts performed an unbiased and isolated audit of the smart contract codebase. The final debriefs took place on the December 03, 2021.

The main goal of the audit was to verify the claims regarding the security of the smart contract. During the audit, no critical issues were found, after the manual and automated security testing and the claim have been successfully verified.

In the StakeRewardFlexible contract, the governance can drain out all tokens at any time.

```
function withdrawExtraTokens(address token, uint256 amount) external onlyGovernance {
    IERC20(token).safeTransfer(msg.sender, amount);
  }
```

The vesting and airdrop contract are relying on someone adding liquidity to the contract. The governance can pause or drain out tokens at any time. There is no guarantee for the user to get any funds.

```
function withdraw(
        address _to,
        address _toke,
        uint256 _amount
    ) external onlyGovernance {
```

```
        IERC20(_toke).safeTransfer(_to, _amount);
    }
```

Update (14.12): We have done a second check of the contracts and have been able to recognize a better staking and governance structure. Keep in mind that mapping can consume a lot of gas when lists increasing in size. It's referred to the new blacklisting function inside your token contracts, which is a strong tool and should be used wisely by your governance.

7. Deployed Smart Contract

VERIFIED

Staking = https://etherscan.io/address/0x652770a3152E29506e575269E94E059278088E57#code

Vesting = https://etherscan.io/address/0x967583939a2E660567345CFEe6BE66870075B3d1#code

Airdrop = https://etherscan.io/address/0xd26e4A8Ea2C3f558a4D0D4a96860134e8e33c893#code