Blaize.Security

December 15th, 2021 / V. 1.0

PEAKDEFI SMART CONTRACT AUDIT

TABLE OF CONTENTS

AUDIT RATING

The PEAKDEFI smart contract's source code was taken from one repository provided by the PEAKDEFI team.

SCORE **9.5**/10

The scope of the project is **PEAKDEFI** set of contracts**:**

1/ ERC20.sol

2/ GovernorAlpha.sol

3/ PeakToken_V2.sol

4/ Timelock.sol

5/ TokenProxy.sol

6/ Migration.sol

The scope of the audit is the code at the main branch with commit:

- https://github.com/PeakDeFi/peakdefi-governance 21b312592069841b5b9dd29833596f48c01e29a9

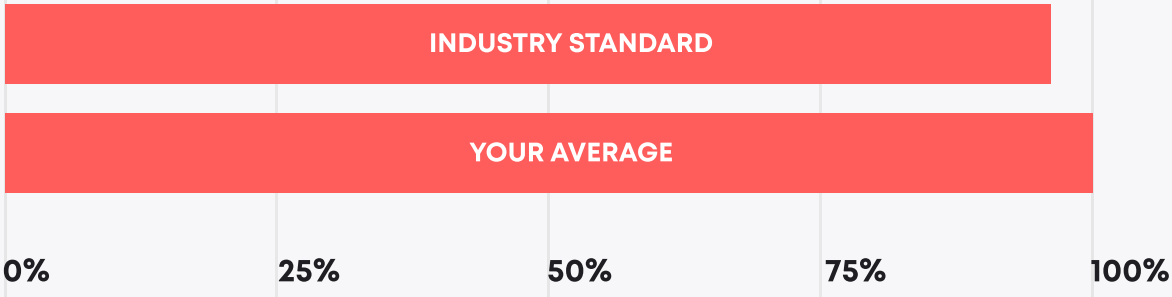
Post-audit scope for validation includes the code at the main branch with commit:

- https://github.com/PeakDeFi/peakdefi-governance/tree/audit-update 99345921b6bb6fb769ae19bdc86369447996e837

TECHNICAL SUMMARY

In this report, we consider the security of the contracts for PEAKDEFI protocol. Our task is to find and describe security issues in the smart contracts of the platform. This report presents the findings of the security audit of **PEAKDEFI** smart contracts conducted between **November 29th, 2021 - December 14th, 2021.**

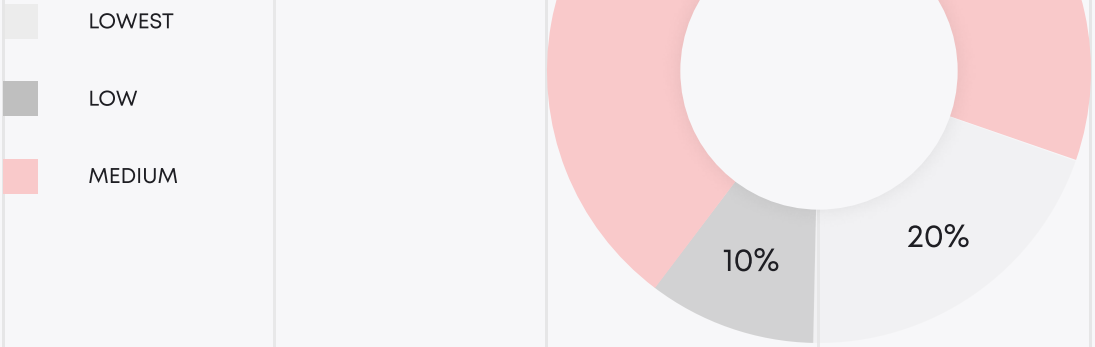
Testable code



The testable code is 100%, which is above the industry standard of 95%.

The scope of the audit includes the unit test coverage, that bases on the smart contracts code, documentation and requirements presented by the PEAKDEFI team. Coverage is calculated based on the set of Truffle framework tests and scripts from additional testing strategies. Though, in order to ensure a security of the contract Blaize.Security team recommends the PEAKDEFI team put in place a bug bounty program to encourage further and active analysis of the smart contracts.

THE GRAPH OF VULNERABILITIES DISTRIBUTION:



The table below shows the number of found issues and their severity. A total of 6 problems were found. All issues that influence security were fixed or verified by the PEAKDEFI team.

	FOUND	FIXED/VERIFIED
Medium	3	3
Low	1	1
Lowest	2	1

SEVERITY DEFINITION

Critical

A system contains several issues ranked as very serious and dangerous for users and the secure work of the system. Needs immediate improvements and further checking.

High

A system contains a couple of serious issues, which lead to unreliable work of the system and migh cause a huge information or financial leak. Needs immediate improvements and further checking.

Medium

A system contains issues which may lead to mediumfinancial loss or users' private information leak. Needs immediate improvements and further checking.

Low

A system contains several risks ranked as relatively small with the low impact on the users' information and financial security. Needs improvements.

Lowest

A system does not contain any issue critical to the secure work of the system, yet is relevant for best

AUDITING STRATEGYAND TECHNIQUES APPLIED \ PROCEDURE

We have scanned this smart contract for commonly known and more specific vulnerabilities:

- Unsafe type inference;
- Timestamp Dependence;
- Reentrancy;
- Implicit visibility level;
- Gas Limit and Loops;
- Transaction-Ordering Dependence;
- Unchecked external call - Unchecked math;
- DoS with Block Gas Limit;
- DoS with (unexpected) Throw;
- Byte array vulnerabilities;
- Malicious libraries;
- Style guide violation;
- ERC20 API violation;
- Uninitialized state/storage/ local variables;
- Compile version not fixed.

Procedure

In our report we checked the contract with the following parameters:

- Whether the contract is secure;
- Whether the contract corresponds to the documentation;
- Whether the contract meets best practices in efficient use of gas, code readability;

Automated analysis:

Scanning contract by several public available automated analysis tools such as Mythril, Solhint, Slither and Smartdec. Manual verification of all the issues found with tools.

Manual audit:

Manual analysis of smart contracts for security vulnerabilities. Checking smart contract logic and comparing it with the one described in the documentation.

EXECUTIVE SUMMARY

According to the assessment, the Customer's smart contracts have no critical security problems. All unclear or suspicious functionality was verified with PEAKDEFI team and with addtional tests.

The scope includes standard reused governance contracts (GovernorAlpha and Timelock) forked initially from the Compound protocol governance.

Also, the token implementation is unstandard, though it fully implements IERC20 interface with additional delegation feature. Nevertheless, the protocol uses upgradeable token contract, which is not recommended in general, since it allows to change the functionality after token deployment.

We described issues and added Customer’s comments according to the conclusion of these documents. Please read the whole document to estimate the risks well.

	RATING
Security	8.8
Gas usage and logic optimization	9.5
Code quality	9.5
Test coverage	10
Total	9.5

COMPLETE ANALYSIS

MEDIUM

✔ Verified

Delegation can be blocked

There is no ability to delegate concrete amount of tokens, only the full balance. So there is an ability to block delegation.
Scenario:
User1 delegates votes to User2
User1 increases token balance by achieving more StackTokens
User1 re-delegates votes to User3
_moveDelegates() reverts the transaction, because of an attempt to subtract more tokens than delegated to User2.

Recommendation:

provide the functionality to delegate a certain amount of tokens instead of full balance and/or re-check the calculations in _moveDelegates() for sources of the votes.

Post-audit:

By the client, the system designed to delegate full amount, so this issue is acceptable. Thus it is marked as verified. Though, the issue is still possible with manual operations.

MEDIUM

✔ Verified

Non-standard ERC20 token

There are several issues with token
Non-standard (non zeppelin) implementation of ERC20 is used. In general it is recommended to import OpenZeppelin contract and inherit it;
The ERC20 token in the PeakDeFi contracts implements a non-standard function - access control and minting role. In general it is recommended to implement additional functionality in the “leaf” contract, which implements token itself.

Recommendation:

Use import of standard ERC20 and move non-standard functionality to the PeakToken_v2

Post-audit:

By the client, the current implementation is acceptable as it meets the ERC20 standards and provide the necessary interfaces.

MEDIUM

✔ Verified

Not an upgradeable contract

PeakToken_v2.sol implements initialize() function but does not utilize an upgradeable version of the OpenZeppelin library.

Recommendation:

Use an upgradeable version of the OpenZeppelin library for the upgradeable contract. Also, usually it is not recommended to implement token as an upgradeable contract, since it gives too much power to the owner/deployer of the contract.

Post-audit:

By the client, PeakToken_V2 is upgradable contract replacing the V1 token with additional delegation feature.

LOW

✔ Resolved

Mis-use of the functionality

PeakToken_v2.sol, _transfer() - the function overrides standard functionality with moving delegates, instead of using the _afterTokenTransfer() function.

Recommendation:

Consider optimisation of the functionality by using _afterTokenTransfer()

LOWEST

✔ Verified

Missing revert for delegating 0 or to zero address

For now the contract allows to perform the transaction for delegating 0 tokens or for 0 address. It creates misleading transactions with no effect.

Recommendation:

provide require() statements to prohibit delegating 0 tokens or for 0 address to save gas and prevent misleading transactions.

Post-audit:

Verified after the converstaion with the client

LOWEST

Unresolved

Update Solidity version

Solidity 0.6.12 used in the contracts is acceptable, though it is recommended to use the latest stable version of Solidity (which is 0.8.10 in the current moment) in order to get the latest optimisations and the latest security patches in the compiler.

Recommendation:

Consider update of the Solidity used in the contracts

	ERC20.sol	GovernorAlpha.sol	PeakToken_V2.sol
✔ Re-entrancy	Pass	Pass	Pass
✔ Access Management Hierarchy	Pass	Pass	Pass
✔ Arithmetic Over/Under Flows	Pass	Pass	Pass
✔ Delegatecall Unexpected Ether	Pass	Pass	Pass
✔ Default Public Visibility	Pass	Pass	Pass
✔ Hidden Malicious Code	Pass	Pass	Pass
✔ Entropy Illusion (Lack of Randomness)	Pass	Pass	Pass
✔ External Contract Referencing	Pass	Pass	Pass
✔ Short Address/ Parameter Attack	Pass	Pass	Pass
✔ Unchecked CALL Return Values	Pass	Pass	Pass
✔ Race Conditions / Front Running	Pass	Pass	Pass
✔ General Denial Of Service (DOS)	Pass	Pass	Pass
✔ Uninitialized Storage Pointers	Pass	Pass	Pass
✔ Floating Points and Precision	Pass	Pass	Pass
✔ Tx.Origin Authentication	Pass	Pass	Pass
✔ Signatures Replay	Pass	Pass	Pass
✔ Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass	Pass

	Timelock.sol	TokenProxy.sol
✔ Re-entrancy	Pass	Pass
✔ Access Management Hierarchy	Pass	Pass
✔ Arithmetic Over/Under Flows	Pass	Pass
✔ Delegatecall Unexpected Ether	Pass	Pass
✔ Default Public Visibility	Pass	Pass
✔ Hidden Malicious Code	Pass	Pass
✔ Entropy Illusion (Lack of Randomness)	Pass	Pass
✔ External Contract Referencing	Pass	Pass
✔ Short Address/ Parameter Attack	Pass	Pass
✔ Unchecked CALL Return Values	Pass	Pass
✔ Race Conditions / Front Running	Pass	Pass
✔ General Denial Of Service (DOS)	Pass	Pass
✔ Uninitialized Storage Pointers	Pass	Pass
✔ Floating Points and Precision	Pass	Pass
✔ Tx.Origin Authentication	Pass	Pass
✔ Signatures Replay	Pass	Pass
✔ Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Contract: Peaktoken_V2

Initialization->

- ✓ Sets correct properties (108ms)

Methods->

- ✓ Should allow to mint only to user with minter role (765ms)
- ✓ Should transfer properly (237ms)
- ✓ Fails in case of incorrect transaction (95ms)
- ✓ Must delegate votes from sender to delegatee (334ms)
- ✓ Should set votes correctly while accessing votes twice in same block (345ms)
- ✓ Gets prior votes correctly (145ms)
- ✓ Must delegate votes from signature to delegatee (174ms)
- ✓ Signature is assigned only for delegatee
- ✓ Gets current votes properly (106ms)
- ✓ Gets prior votes properly (681ms)
- ✓ Gets prior votes properly from the middle checkpoint (480ms)
- ✓ Burns tokens (179ms)
- ✓ Doesn't allow to burn more tokens than the account has (132ms)

Transactions->

- ✓ Token holders has no votes until they are delegated (548ms)
- ✓ Votes are undelegated when delegator has sent his tokens (300ms)
- ✓ Amount of delegatee votes is increased when amount of tokens of deleaator is inceased (332ms)

- ✓ Amount of delegatee votes is decreased when amount of tokens of delegator is decreased (387ms)
- ✓ Amount of delegatee votes is decreased when some amount of tokens of delegator is burnt (342ms)
- ✓ Should allow two users to be delegators for each other (591ms)
- ✓ Should allow to be delegator with 0 tokens (319ms)

21 passing (23s)

TEST COVERAGE RESULTS

FILE	% STMTS	% BRANCH	% FUNCS
PeakToken_V2.sol (with ERC20.sol)	100.00	86.67	100.00

The scope includes standard reused governance contracts GovernorAlpha and Timelock, forked initially from the Compound protocol governance. These contracts were checked against the standard set of tests for such kind of governance and verified to be identical to the initial contracts.

DISCLAIMER

The information presented in this report is an intellectual property of the customer including all presented documentation, code databases, labels, titles, ways of usage as well as the information about potential vulnerabilities and methods of their exploitation. This audit report does not give any warranties on the absolute security of the code. Blaize.Security is not responsible for how you use this product and does not constitute any investment advice.

Blaize.Security does not provide any warranty that the working product will be compatible with any software, system, protocol or service and operate without interruption. We do not claim the investigated product is able to meet your or anyone else requirements and be fully secure, complete, accurate and free of any errors and code inconsistency.

We are not responsible for all subsequent changes, deletions and relocations of the code within the contracts that are the subjects of this report.

You should perceive Blaize.Security as a tool which helps to investigate and detect the weaknesses and vulnerable parts that may accelerate the technology improvements and faster error elimination.