Blaize.Security

February 2d, 2022 / V. 1.0

PEAKDEFI LAUNCHPAD SMART CONTRACT AUDIT

TABLE OF CONTENTS

AUDIT RATING

The PeakDeFi Launchpad contract's source code was taken from the repository provided by the PeakDeFi team.

SCORE **9.7** /10

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

1/ SaleFactory

2/ Staking

3/ PeakDefiSale

Repository:
https://github.com/PeakDeFi/peakdefi-launchpad

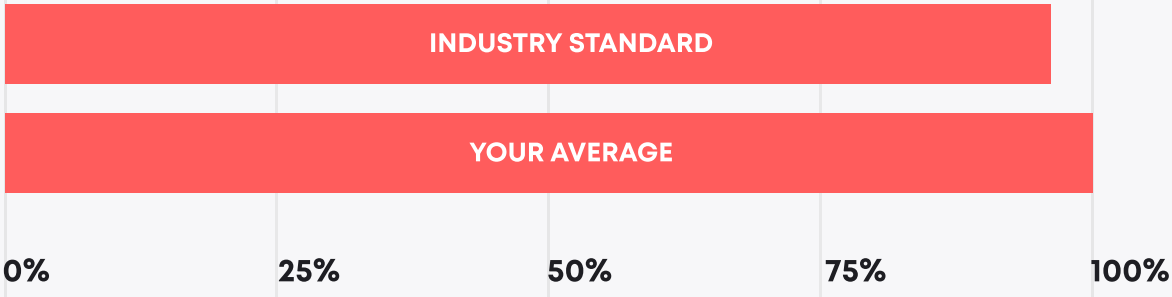
The scope of the audit is the code at the main branch with commit:
678fbc8e0785d567c2248027dc3be93d76f87b8e

Post-audit scope for validation includes the code at the main branch with commit:
e94cb77a3db211cec5c772c31c216d879ef4ff4a

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 **January 17th, 2022 - February 2d, 2022.**

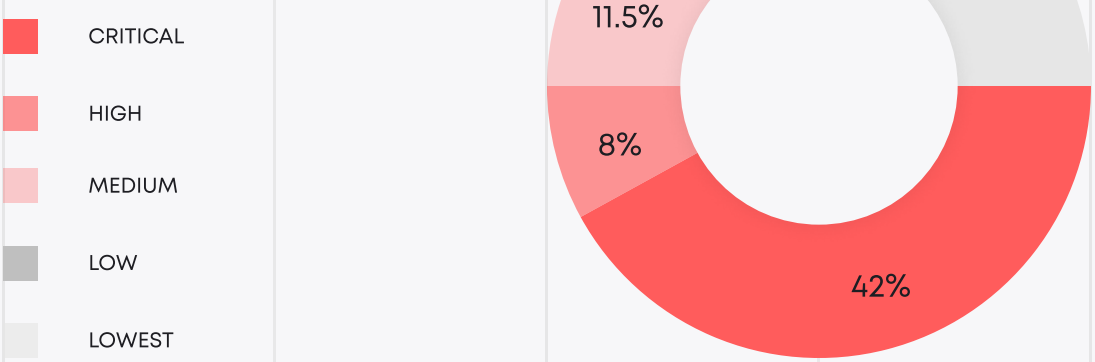
Testable code



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 26 problems were found. 23 issues were fixed or verified by the PeakDeFi team.

	FOUND	FIXED/VERIFIED
Critical	11	11
High	2	2
Medium	3	3
Low	2	2
Lowest	8	5

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

The contract contained several critical issue connected to the incorrect funds flow and incorrect storage usage. There were no restrictions for admin functions and there were several scenarios which could cause lock of the funds on the contract. Though, the team has fixed pointed issues and they are also covered with appropriate cases..

All other issues were connected to missed checks, which could cause particular disfunction of the contract, and code quality. Nevertheless, all security risk issues were fixed by the team.

The overall security is high enough though the code lacks of documentation and the overal quality may be increased. Nevertheless, it performs all desired actions and has solid functionality.

	RATING
Security	9.8
Gas usage and logic optimization	9.8
Code quality	9.2
Test coverage*	10
Total	9.7

* There was no initial test coverage presented by PeakDeFi team, the whole unit tests system was written by Blaize.Security engineers.

COMPLETE ANALYSIS

Common

LOWEST	✔ Resolved

Unnecessary usage of SafeMath library.

Starting from Solidity version 0.8 usage of SafeMath library is unnecessary since Solidity has built-in checks for over/underflow. SafeMath only increases gas spending during function calls.

Recommendation:

Replace all SafeMath functions with arithmetic operators.

SaleFactory.sol

CRITICAL	✔ Resolved

Contract code size exceeds 24576 bytes.

Contract exceeds 24576 bytes thus, it might be non-deployable to mainnet.

Recommendation:

Reduce code size.

Answer from client.

For compiling we use remix and code optimization. Cant make it smaller than now.

Post-audit.

Compiling in Remix with optimization runs 200 created a bytecode which didn't exceed 24576 bytes.

MEDIUM ✔ Resolved

Function getAllSales() doesn't return all sales

Function doesn't work under any conditions.

Recommendation:

// require(endIndex > startIndex, "Bad input"); change to require(endIndex >= startIndex, "Bad input")
// address[] memory sales = new address[](endIndex - startIndex); change to
address[] memory sales = new address[](endIndex - startIndex + 1);
// for(uint i = startIndex; i < endIndex; i++) change to
for(uint i = startIndex; i <= endIndex; i++)

LOW ✔ Resolved

Variables lack validation for zero address.

Line 27 in constructor. Passed variables '_adminAddress' and '_allocationStaking'
Should be validated not to be zero address. This is especially crucial for variable '_adminAddress' since the storage variable 'admin' has no additional setters.

Recommendation:

Validate that passed address variables are not zero addresses.

LOWEST	✔ Resolved

Missing revert string in 'require'.

Line 34. 'Require' statements should provide error strings for better error clarification.

Recommendation:

Add revert strings to 'require' statements.

PeakDeFiSale.sol

CRITICAL	✔ Resolved

calculateTotalTokensSold() doesn't work properly. Funds can be stuck.

Condition if(tokensPerTier * sale.tokenPriceInBUST / 10**sale.token.decimals() < t.BUSTDeposited) cannot be satisfied because if all tokens in tier are sold t.BUSTDeposited is equal to tokensPerTier * sale.tokenPriceInBUST / 10**sale.token.decimals
In else block totalTokensSold has to be calculated as sum of all tokens sold in each tier

Recommendation:

Change the sign of comparison in the first condition to less or equal.
Accumulate number of sold tokens from all tiers in case if not all tokens were sold

CRITICAL ✔ Resolved

Admin function updateTokenPriceInBUSD() doesn't work

Function fails to pass requirement sale.saleStart > block.timestamp
Variable sale.saleStart is never assigned.

Recommendation:

Assign value to variable

CRITICAL ✔ Resolved

Vesting parameters cannot be set after sale is created

Logic error in Line 110

Recommendation:

Remove "!" sign form require statement.

CRITICAL ✔ Resolved

Funds might not be withdrawn correctly.

Lines 512, 528, 536. It is said that the Maintoken, in which the funds are collected is ERC20(BUSD, based on naming of variables 'sale.totalBUSDRaised'). The funds are collected like ERC20 token (line 402). However in the functions withdrawEarningsInternal(), withdrawLeftoverInternal(), withdrawRegistrationFees() funds are transferred as ETH(or any native coin of blockchain) with function safeTransferPEAK().

Recommendation:

Verify that funds withdrawal and collection is performed correctly.

Post-audit.

Functions withdrawLeftoverInternal() and withdrawRegistrationFees() were removed. Function withdrawEarningsInternal() was fixed.

CRITICAL ✔ Resolved

Anyone can call functions.

Lines 283, 296. Functions set essential for contract operation information, however they are not restricted and anyone can call them.

Recommendation:

Restrict functions from being called by anyone.

CRITICAL ✔ Resolved

Function call will revert if the user registered for tier 0.

Line 370. Tiers start with 0index, thus user is able to register in tier 0. However, the 'require' statement will revert if the user's tier is equal 0, though it might be a valid tier for the user.

Recommendation:

Choose another way to validate the user is in the whitelist, or start tiers with index 1.

CRITICAL ✔ Resolved

User's leftovers might get stuck.

During any first claim with function withdrawTokens() use also claims any leftovers if there are some and this is the only chance to claim leftovers.
However this is possible for the user to claim all his portions by calling function withdrawMultiplePortions() and not claim leftovers. After claiming all portions, it is impossible to call function withdrawTokens() and claim leftovers, thus the user won't be able to claim his funds back.

Recommendation:

Consider adding another external function for claiming leftovers.

HIGH

✔ Resolved

Potential loss of accuracy

Lines 400, 603. In case sale.tokenPriceInBUST has 18 decimals(as the BUSD token has), the contract might lose accuracy during calculations.
For example, 2 BUSD tokens were deposited which 2 * 10^18 and the price is set as 0.4 * 10^18. The result of division will be (2 * 10^18) / (0.4 * 10^18) = 5, where it should 5 * (10^sale.token.decimals()) counting that different tokens might have different decimals.
This should also be taken into account in line 443.

Recommendation:

Consider decimals in the formulas.
For formulas in lines 400, 603 the calculation should look like: amountOfTokensDeposited * (10**sale.token.decimals()) / sale.tokenPriceInBUST
For formula in line 443 the calculation should look like: tokensForUser * sale.tokenPriceInBUST / 10**sale.token.decimals().

HIGH	✔ Resolved

The Amount of tokens sold might exceed the amount of tokens to sell.

Line 400. It is not verified that sale.totalTokensSold doesn't exceed sale.amountOfTokensToSell.
Also, In case during sale or vesting, the price of tokens decreases(by setting a lower value in updateTokenPriceInBUSD()) users' bought amount could exceed sale.amountOfTokensToSell.
For example, sale.amountOfTokensToSell is 100 and the price of token is 1$. User1 provides 50 BUSD and buys 50 tokens as well as user2 does.
When vesting starts, the admin sets the price of the token to be 0.5$. User1 claims his tokens with a value of 50$ and the amount of tokens for user1 is now 50 / 0.5 = 100 tokens, which he claims. When user2 decides to claim his tokens, there are already not enough tokens on the contract's balance.

Recommendation:

Verify that sale.totalTokensSold cannot exceed sale.amountOfTokensToSell in all cases.

Post-audit.

Client added a restriction in function updateTokenPriceInBUSD(), validating that a price cannot be changed once the sale has started.
However there is no validation in function participate() that sale.totalTokensSold doesn't exceed sale.amountOfTokensToSell.

MEDIUM	✔ Resolved

Use SafeERC20 library.

Lines 338, 446, 522. ERC20 tokens should be transferred with ‘safeTransfer’ and ‘safeTransferFrom’. SafeERC20 performs all the checks that tokens were transferred, including ono-standard ERC20 implementation(like in USDT tokens).

Recommendation:

Use ‘safeTransfer’ and ‘safeTransferFrom’ instead.

MEDIUM	✔ Resolved

Use ‘portionVestingPrecision’ instead of 100.

Line 601. Variable ‘portionVestingPrecision’ indicates the summation of elements from array ‘vestingPrecentPerPortion’ and can be greater than 100. Thus, this variable should be used to calculate an accurate amount of tokens for portions.

Recommendation:

Use the variable ‘portionVestingPrecision’ in the formula.

Post-audit.

Variable ‘portionVestingPrecision’ was replaced by 100.

LOW	✔ Resolved

Variable lacks zero address check.

Line 191. Passed variable '_mainToken' is not validated not to be zero address.

Recommendation:

Validate that variable is not zero address before assigning it to the storage variable.

Post-audit.

Variable 'mainToken' was replaced by variable 'BUSDToken' and being initialized during contract deployment.

LOWEST	✔ Unresolved

Missing revert string in 'require'.

Lines 112, 113, 125, 129, 241, 459, 504, 507, 517, 518. 'Require' statements should provide error strings for better error clarification.

Recommendation:

Add revert strings to 'require' statements.

LOWEST	✔ Resolved

Remove the test function.

Line 276. Pre-production code should not contain test code.

Recommendation:

Make sure to remove all test code from contracts.

LOWEST	✔ Resolved

Use the Address library.

Line 480. ETHshould be sent with Address.sendValue. This function performs all the necessary security checks.

Recommendation:

Use the Address library instead.

Post-audit.

Transferring ETH was removed from contract.

Stacking.sol

CRITICAL	✔ Resolved

Wrong comparison sign.

Function withdraw(). Line 59. Currently the user is able to pass a greater value than he has staked and 'require' won't revert, since it is compared, that user.amount is less or equal than '_amount'. Also it prevents the user from withdrawing less funds than he has staked.

Recommendation:

Change a comparison sign from '<=' to '>='.

CRITICAL ✔ Resolved

The Withdrawal fee will always be 30%.

Function withdraw(). In Function withdraw(), before withdrawing funds, an internal function harvest() is called. This function changes user.staking to current block.timestamp. After that a withdrawal fee is calculated, based on the difference between user.staking and block.timestamp. This difference will always be equal to 0 causing the user to pay a maximum 30% of withdrawal fee, even though the actual difference between deposit timestamp and withdrawal timestamp could be much greater.

Recommendation:

Make sure that the difference between actual deposit timestamp and withdrawal timestamp is calculated correctly and users don't lose their funds.

CRITICAL ✔ Resolved

Anyone can call a function.

Line 44, function setStakingToken. Function sets essential information in contract, however anyone can call it.

Recommendation:

Restrict function from being called by anyone.

LOWEST	✔ Unresolved

Variable lack check for zero address.

Line 45. Parameter _erc20 should be checked not to be zero address.

Recommendation:

Add ‘require’ statement to validate that variable is not a zero address.

LOWEST	✔ Resolved

Values should be used as storage constants.

Line 131. Use the storage variable ‘stakingPercent’ instead of value 7 (especially since ‘stakingPercent’ is not used anywhere in code.) Value 31556926 should also be used as a storage constant for better code understanding.

Recommendation:

Move values to storage constants.

LOWEST	✔ Unresolved

Use Solidity time units.

Lines 138-144. Use Solidity time units(days, weeks, etc) instead of calculating your own values. For better code understanding.

Recommendation:

Use Solidity time units instead.

	PeakDefiSale.sol	SaleFactory.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

Stacking.sol

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

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Contract: Sale Factory

- ✓ sales factory cannot be deployed with zero addresses (1355ms)
- ✓ getLastDeployedSale returns 0 when no sales is deployed (220ms)
- ✓ shows number of sales deployed (381ms)
- ✓ return all sales (688ms)
- ✓ sets allocation staking (386ms)

Contract: Peak DeFi Sale

- ✓ user deposit to buy all tokens (1 tier, 1 user) (2142ms)
- ✓ withdraw multiple portion at once (1347ms)
- ✓ unused amount of BUSDT is returned to user after sale (1389ms)
- ✓ allow user to register for sale if he staked enough tokens (1249ms)

methods requirements

setVestingParams()

- ✓ only admin can set vesting parameters (1210ms)
- ✓ doesn't set invalid vesting parameters (153ms)
- ✓ doesn't set vesting parameters if sum of percentages is not 100 (261ms)
- ✓ vesting parameters can be set only once (270ms)
- ✓ vesting parameterscannot be set before sale parameters (107ms)

setSaleParams()

- ✓ sale can be created only once (268ms)
- ✓ sale owner cannot be zero address (82ms)
- ✓ sale cann't be created with bad input (74ms)

setRegistrationTime()

- ✓ registration time cannot be set before sale is created (80ms)
- ✓ registration time start has to be greater than current time (176ms)
- ✓ registration time end has to be smaller than sale end time (321ms)

registerForSale()

- ✓ cannot register for sale with less staked token than minimal amount (1208ms)
- ✓ cannot register for sale twice from the same account (1353ms)
- ✓ can register only in registration time frame (1365ms)

addTiers()

- ✓ has to be at least 1 tier (81ms)
- ✓ tier weights and tier points has to be the same length (76ms)
- ✓ tier weights cannot be zero (71ms)

depositTokens()

- ✓ user cad deposit tokens only once (577ms)

participate()

- ✓ user cannot participate until sale is created (74ms)
- ✓ user can participate only in registration time frame (187ms)
- ✓ user can participate only once (1265ms)
- ✓ user can't participate with 0 tokens (1001ms)
- ✓ user has to be in whitelist to participate (1054ms)

withdrawTokens()

- ✓ user can't withdraw tokens before tokensUnlockTime (1304ms)
- ✓ user can't withdraw tokens if portion id is invalid (1846ms)
- ✓ user can't withdraw first portion if time hasn't come (1742ms)

withdrawMultiplePortions()

- ✓ user cant withdraw tokens before tokensUnlockTime (1732ms)
- ✓ user cant withdraw tokens if portion ids are invalid (1813ms)
- ✓ withdrawMultiplePortion is ignoring portions that were withdrawn (2164ms)

admin function

- ✓ admin can withdraw earnings (busdt) (2793ms)
- ✓ admin can withdraw not sold tokens (2576ms)
- ✓ admin can set token price (1265ms)

view methods

- ✓ shows sale parameters UnlockTime and PercentPerPortion (57ms)
- ✓ shows if user is whitelisted (121ms)
- ✓ shows maximum amount of tokens user can buy (45ms)

Contract: Staking

- ✓ only owner can set staking token (931ms)
- ✓ deposit tokens (494ms)
- ✓ pays percentages for 1st deposit to user if he deposits second time (1137ms)
- ✓ withdraw tokens after 13 days of staking with unstake fee 30% (1103ms)
- ✓ withdraw tokens after 27 days of staking with unstake fee 20% (1347ms)
- ✓ withdraw tokens after 55 days of staking with unstake fee 10% (906ms)
- ✓ withdraw tokens after 83 days of staking with unstake fee 5% (1026ms)
- ✓ withdraw tokens after 84 days of staking with unstake fee 0% (1034ms)
- ✓ does't allow user to withdraw more than user deposited (445ms)

53 passing (53s)

TEST COVERAGE RESULTS

FILE	% STMTS	% BRANCH	% FUNCS
SaleFactory.sol	100	100	100
Staking.sol	100	100	100
PeakDefiSale.sol	100	95.74	100
All files	**100**	**98.58**	**100**

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.