05/05

Arcane

SMART CONTRACT AUDIT REPORT

Summary

Our team has conducted a smart contract audit for the given codebase.

The contracts are in good condition. Based on the fixes provided by the Arcane team and on the quality and security of the codebase provided, our team can give a score of 95 to the audited smart contracts.

During the auditing process, the team has found a couple of informational issues, 1 issues with a low level of severity and 2 issues with a informational level of severity.

The severity of the issue	Total issues found	Resolved issues	Unresolved issues
Informational	2	2	0
Low	1	0	1
Medium	0	0	0
High	0	0	0
Critical	0	0	0
Total	3	2	1

The testable code is 98% which is above the industry standard of 95%.

The test results and the coverage can be found in the accompanying section of this audit report.

Please mind that this audit does not certify the definite reliability and security level of the contract. This document describes all vulnerabilities, typos, performance issues, and security issues found by the auditing team. If the code is under development, we recommend running one more audit once the code is finalized.

Auditing Strategy and Techniques Applied

Arcane 

They apply systems thinking to dynamically map out the influencing factors for each project. They establish relationships with partners who create added value and bring together the public and private sectors. They create cross-industry links for knowledge flows. They apply values that reinforce each other: socio-cultural value, experimental value, technical value, and economical value.

Within the scope of this audit, two independent auditors deeply investigated the given codebase and analyzed the overall security and performance of smart contracts.

The debrief took place from April 25th to May 10th, 2022 and the final results are present in this document.

Auditing team has made a review of the following contracts:

- IERC20Mintable.sol
- IPool.sol
- IUniswapV2Factory.sol
- IUniswapV2Pair.sol
- IUniswapV2Router01.sol
- IUniswapV2Router02.sol
- ArcaneToken.sol
- TokenPreset.sol
- Pool.sol

The source code was taken from the following **source** - arcane-contracts

Initial commit submitted for the audit - arcane-contracts
Last commit - fd18379

Workflow of the auditing process

During the manual phase of the audit, auditing team manually looks through the code in order to find any security issues, typos, or discrepancies with the logic of the contract.

Within the testing part, auditors run integration tests using the Hardhat testing framework. The test coverage and the tests themselves are inserted into this audit report.

Auditing team uses the most sophisticated and contemporary methods and techniques to ensure the contract does not have any vulnerabilities or security risks:

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

Structure and organization of the findings

For the convenience of reviewing the findings in this report, auditors classified them in accordance with the severity of the issues. (from most critical to least critical). The acceptance criteria are described below.

All issues are marked as "*Resolved*" or "*Unresolved*", depending on whether they have been fixed by Arcane or not. The latest commit, indicated in this audit report should include all the fixes made.

To ease the explanation, the team has provided a detailed description of the issues and recommendations on how to fix them.

Hence, according to the statements above, we classified all the findings in the following way:

- **Critical** (The issue bear a definite risk to the contract, so it may affect the ability to compile or operate)
- **High** (Major security or operational risk found, that may harm the end-user or the overall performance of the contract)
- **Medium** (The issue affects the contract to operate in a way that doesn't significantly hinder its performance)
- **Low** (The found issue has a slight impact on the performance of the contract or its security)
- **Informational** (The issue does not affect the performance or security of the contract/recommendations on the improvements)

Manual Report

Resolved, Informational: Lack of NatSpec annotations

Interfaces IERC20Mintable, IUniswapV2Factory, IUniswapV2Pair, IUniswapV2Router01 and IUniswapV2Router02, also contract TokenPreset are not covered by NatSpec annotations.
Recommendation: Consider to cover by NatSpec all contract`s methods.

Resolved, Informational: Order of Layout

The layout contract elements in the Pool contract are not logically grouped.

The contract elements should be grouped and ordered in the following way:

- Pragma statements;
- Import statements;
- Interfaces;
- Libraries;
- Contract.

Inside each contract, library or interface, use the following order:

- Library declarations (using statements);
- Constant variables;
- Type declarations;
- State variables;
- Events;
- Modifiers;
- Functions.

Ordering helps readers to navigate the code and find the elements more quickly.

Recommendation: Consider changing the order of layout according to solidity documentation: Order of Layout.

Unresolved, Low: Lack of gas is possible

Use of multiple for loops in Pool and ArcaneToken contracts, this has the danger of running into 'out of gas' errors if they are not kept under control.

Recommendation: This can be avoided by adding a 'gasleft() < 20000' type of condition that if it returns true. It will break the execution so the 'out of gas' error message will be avoided.

Test results

To verify the contract security and performance a bunch of integration tests were made using the Hardhat testing framework.

Tests were based on the functionality of the code, business logic, and requirements and for the purpose of finding the vulnerabilities in the contacts.

In this section, we provide both tests written by Arcane and auditors.

It's important to note that auditors do not modify, edit or add tests to the existing tests provided in the Arcane repo. We write totally separate tests with code coverage of a minimum of 95%, to meet the industry standards.

Tests written by auditing team

Test Coverage

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts\	98.96	98.16	98.28	98.62	
ArcaneToken.sol	100	100	100	100	
Pool.sol	97.92	96.29	96.55	97.24	361,662,666
contracts\interfaces\	100	100	100	100	
IERC20Mintable.sol	100	100	100	100	
IPool.sol	100	100	100	100	
IUniswapV2Factory.sol	100	100	100	100	
IUniswapV2Pair.sol	100	100	100	100	
IUniswapV2Router01.sol	100	100	100	100	
IUniswapV2Router02.sol	100	100	100	100	
contracts\mock\	100	100	100	100	
TokenPreset.sol	100	100	100	100	
WrappedToken.sol	100	100	100	100	
All files	99.65	99.38	99.42	99.54	

Test Results

Pool
✔ GetInfo
Deposite
✔ if reward token decimals are less than staked token (357ms)
✔ if reward token decimals and staked token decimals are equal (295ms)
✔ Should be fail if pool is started (75ms)
✔ Check correct transfer
✔ Check correct deposited
✔ Check correct totalDeposited (55ms)
✔ Check set nonce
Should be fail if incorrect amount
✔ if more then max allocation
✔ if less then min allocation
✔ if more then remaining allocation
✔ if more then remaining allocation (216ms)
getAvailAmountToDeposit
✔ Should return 0, 0 if all allocation used
✔ Should return 0, 0 if total currency is not bigger than total deposited (300ms)
✔ Should get correct if remaning is less the max allocaiton (65ms)
✔ Should success (67ms)
setVesting
✔ Should revert if pool type is linear but count period of pool and period duration are zero
✔ Should revert if interval unlocking part is invalid (89ms)
✔ Should set pool correctly (95ms)
✔ Should revert if last interval unlocking part is invalid (93ms)
✔ Should revert if interval starting timestamp is invalid (119ms)
addDepositAmount
✔ Should revert if caller not owner
✔ Should revert if arr length incorrect
✔ Should revert if pool is started
✔ Should fail if not have enough allocation (114ms)
✔ Should correct set amount (66ms)
✔ Should correct change totalDeposited
✔ Should success when not enough gas with skip part fo array [skip-on-coverage] (96ms)
Deployed
✔ Should revert initializing token if was initialized before
✔ Should revert if total supply incorrect (89ms)
✔ Should revert if rewardToken incorrect
✔ Should revert if deposit token incorrect
✔ Should revert if token price incorrect (73ms)
✔ Should revert if allocation incorrect (41ms)
✔ Should revert if initial percentage incorrect
getBalanceInfo
✔ Should return zero if zero deposited

```
    ✔ Should return all balance how locked, if pool don't start
    ✔ Should return all unlocked balance if pool started (55ms)
    ✔ Should return correct after harvest and by month linear (193ms)
  increaseTotalSupply
    ✔ Should increase total supply correctly
  setTimePoint
    ✔ Should revert setting if start date will be later than end date
  setSpecificAllocation
    ✔ Should set specific allocation correctly
    ✔ Check specific allocation for user (66ms)
    ✔ Should revert if arrays have different size
    ✔ Should fail when not enough gas with skip part fo array [skip-on-coverage]
  setSpecificVesting
    ✔ Should set specific pool correctly (62ms)
    ✔ Should revert if it was initialized before (75ms)
  harvest
    ✔ Should revert if pool can't be started
    ✔ Should correct transfer amount (80ms)
  completeVesting
    ✔ Should revert if caller not owner
    ✔ Should revert if pool can't be started
    ✔ Should revert if Withdraw funds was called before (67ms)
    ✔ No need to transfer tokens when everything is sold (223ms)
    ✔ Should correct transfer amount (96ms)
  harvestFor
    ✔ Should correct transfer amount (48ms)
  Test case
    ✔ When VESTING_TYPE is SWAP (354ms)
    ✔ When VESTING_TYPE is INTERVAL (903ms)
    ✔ When one user bouth all tokens (769ms)
    ✔ More user with specific case (676ms)
    ✔ Sales after increasing total supply (815ms)
  With Cliff period, max allocation is zero, and specific allocation
    ✔ Should correct calculate harvest (511ms)
Contract: ArcaneToken (upgradeable)
  ArcaneToken Upgradeable Phase Test Cases
    ✔ shouldn't call initialize method again (615ms)
    ✔ should deploy correctly if deploy not in arcane
    ✔ should standard transfer amount of tokens correctly if satisfying the threshold
(13229ms)

Contract: ArcaneToken
  ArcaneToken Initializing Phase Test Cases
    ✔ should set router correctly (142ms)
    ✔ should get balance of owner correctly (209ms)
    ✔ should exclude from fee correctly (263ms)
```

✔ should set _maxTxAmount correctly (171ms)
✔ should set swap fee correctly (251ms)
✔ should set transfer fee correctly (240ms)
✔ should set swapAndLiquifyEnabled correctly (109ms)
✔ should set token's name correctly (148ms)
✔ should set token's symbol correctly (169ms)
✔ should set total supply correctly (142ms)
✔ should set decimals correctly (112ms)

ArcaneToken Get/Set Functions Phase Test Cases

✔ should set threshold correctly (502ms)
✔ should deliver correctly (817ms)
✔ shouldn't deliver if account isn't excluded (1470ms)
✔ should exclude account from reward without _rOwned correctly (1024ms)
✔ should exclude account from reward with _rOwned correctly (1416ms)
✔ shouldn't exclude account from reward if account isn't excluded (926ms)
✔ should include account in reward correctly (1991ms)
✔ shouldn't include account in reward if account is already excluded (364ms)
✔ should set transfer fee percent correctly (818ms)
✔ shouldn't set tax fee percent if value more than 100 (912ms)
✔ shouldn't set tax fee percent if caller isn't owner (413ms)
✔ should set swap fee percent correctly (691ms)
✔ shouldn't set liquidity fee percent if value more than 100 (640ms)
✔ should set max tx percent correctly (623ms)
✔ shouldn't set max tx percent if value more than 100 (471ms)
✔ should update router correctly (441ms)
✔ shouldn't update router if zero's address (347ms)
✔ should exclude account from fee correctly (426ms)
✔ should include account in fee correctly (566ms)
✔ should set enable for swap and liquify correctly (459ms)
✔ should lock correctly (1567ms)
✔ should unlock correctly (1104ms)
✔ shouldn't unlock if caller haven't permission (413ms)
✔ shouldn't unlock if _lockTime isn't exceeds (964ms)
✔ should return reflection from token correctly (337ms)
✔ shouldn't return reflection from token if amount more than supply (119ms)

ArcaneToken Transfer Functions Phase Test Cases

✔ shouldn't transfer amount of tokens if insufficient allowance (743ms)
✔ shouldn't transfer amount of tokens if zero's 'to' address (349ms)
✔ shouldn't transfer amount of tokens if amount is zero (423ms)
✔ shouldn't transfer amount of tokens if amount exceeds the maxTxAmount (516ms)
✔ should transfer amount of tokens from excluded account correctly (3578ms)
✔ should transfer amount of tokens from excluded account correctly if contract's address is excluded (4056ms)
✔ should transfer amount of tokens to excluded account correctly (3278ms)
✔ should transfer amount of tokens correctly if both accounts is excluded (4327ms)

✔ should standard transfer amount of tokens correctly if not satisfying the threshold (2699ms)
✔ should standard transfer amount of tokens correctly without fee (3613ms)
✔ should standard transfer amount of tokens correctly if not satisfying the threshold and with fee (2141ms)
✔ should standard transfer amount of tokens correctly if satisfying the threshold (8825ms)

ArcaneToken Withdraw Functions Phase Test Cases
✔ should withdraw leftovers correctly (9383ms)
✔ should withdraw alien tokens correctly (4632ms)
✔ shouldn't withdraw alien tokens if token's address is arcane and swap&liquify if enabled (553ms)
✔ should withdraw alien tokens by the owner if token's address is arcane and swap&liquify if disabled (4129ms)
✔ shouldn't withdraw alien tokens by not the owner if token's address is arcane and swap&liquify if disabled (2791ms)
✔ shouldn't withdraw alien tokens if amount is zero (1573ms)
✔ shouldn't withdraw alien tokens if amount is zero (1870ms)

ArcaneToken Fee Phase Test Cases
✔ should transfer tokens depending on transfer fee correctly taxFee = 0, liquidityFee = 2 (3478ms)
✔ should transfer tokens depending on fee correctly taxFee = 50, liquidityFee = 0 (4247ms)
✔ should transfer tokens depending on the buy fee correctly (8604ms)
✔ should transfer tokens correctly depending on the fee while add liquidity
✔ should transfer tokens depending on the sell fee correctly
✔ Balances after transfer all tokens when _taxFee = 0; _liquidityFee = 2

126 passing (3m40s)

We are delighted to have a chance to work together with Arcane team and contribute to their success by reviewing and certifying the security of the smart contracts.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.