SMART CONTRACT AUDIT

ZOKYO.

October 26th, 2021 | v. 1.0

PASS

Zokyo's Security Team has concluded that these smart contracts pass security qualifications and are fully production-ready

TECHNICAL SUMMARY

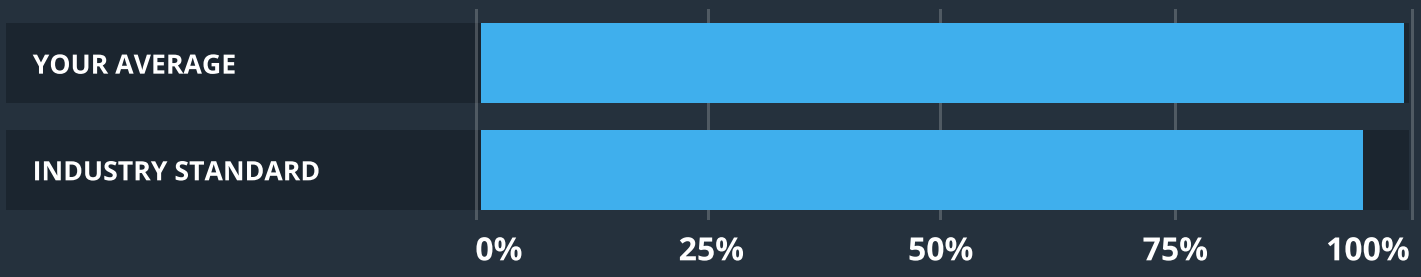
This document outlines the overall security of the Gamestation smart contracts, evaluated by Zokyo's Blockchain Security team.

The scope of this audit was to analyze and document the Gamestation smart contract codebase for quality, security, and correctness.

Contract Status

There were no critical issues found during the audit.

Testable Code



The testable code is 99.47%, which is above the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the Ethereum network's fast-paced and rapidly changing environment, we at Zokyo recommend that the Gamestation team put in place a bug bounty program to encourage further and active analysis of the smart contract.

TABLE OF CONTENTS

AUDITING STRATEGY AND TECHNIQUES APPLIED

The Smart contract's source code was taken from the Gamestation repository.

Repository:
https://bitbucket.org/applicature/tokengear.contracts/src/gamestation-audit-20-08-2021/

Last commit:
353dc41ff8d40c096babea8e36e5a2312053b3aa

Contracts under the scope:

- VestingFactory;
- Vesting.

Throughout the review process, care was taken to ensure that the token contract:

- Implements and adheres to existing Token standards appropriately and effectively;
- Documentation and code comments match logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices in efficient use of gas, without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the latest vulnerabilities;
- Whether the code meets best practices in code readability, etc.

Zokyo's Security Team has followed best practices and industry-standard techniques to verify the implementation of smart contracts. To do so, the code is reviewed line-by-line by our smart contract developers, documenting any issues as they are discovered. Part of this work includes writing a unit test suite using the Truffle testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

1	Due diligence in assessing the overall code quality of the codebase.
2	Cross-comparison with other, similar smart contracts by industry leaders.
3	Testing contract logic against common and uncommon attack vectors.
4	Thorough, manual review of the codebase, line-by-line.

SUMMARY

Zokyo team has conducted a security audit of the given codebase. The contracts provided for an audit are well written and structured. All the findings within the auditing process are presented in this document.

There were no critical issues found during the auditing process. Zokyo security team has found 1 informational issue and 2 issues with low severity levels. Most of them were successfully resolved by the Gamestation team.

All the mentioned findings may have an effect only in the case of specific conditions performed by the contract owner. Contracts bear no operational or security risk to the contract owner or end user.

Based on the results of the audit, Zokyo security team can give a score of 99 to the provided codebase.

STRUCTURE AND ORGANIZATION OF DOCUMENT

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the ability of the contract to compile or operate in a significant way.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

LOW | UNRESOLVED

Smart contract Vesting is not completely covered by NatSpec annotations.
https://docs.soliditylang.org/en/v0.7.4/style-guide.html

Recommendation:
Add comments in the Vesting contract.

LOW | RESOLVED

Use of multiple for loops in both 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.

INFORMATIONAL | RESOLVED

Specifying a pragma version with the caret symbol (^) upfront which tells the compiler to use any version of solidity bigger than specified considered not a good practice. Since there could be major changes between versions that would make your code unstable. The latest known versions with bugs are 0.8.3 and 0.8.4.

Recommendation:
Set the latest version without the caret. (The latest version that is also known as bug-free is 0.8.9).

	VestingFactory	Vesting
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	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

Tests written by Gamestation team

Code Coverage

The resulting code coverage (i.e., the ratio of tests-to-code) is as follows:

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	UNCOVERED LINES
contracts\	98.13	95.73	79.07	97.81	
FixedSwap.sol	96.26	91.46	97.63	95.61	91, 97
GameStationBridge.sol	100.00	100.00	100.00	100.00	
All files	**98.13**	**95.73**	**98.82**	**97.81**	

Test Results

```
Contract: Vesting Factory Contract
  Check constructor
    ✓ Should fail if address is incorrect (848ms)
    Should correct initialized
      ✓ Should correct setup owner (172ms)
      ✓ Should correct setup vesting implementation address (172ms)
      ✓ wner has a DEFAULT_ADMIN_ROLE (184ms)
  Functions
    add
      ✓ Should fail if user does not have DEFAULT_ADMIN_ROLE (502ms)
      ✓ Should add correct if all ok (504ms)
      ✓ Should emit correct event (352ms)
      ✓ Should fail if vesting already exists (386ms)
    list
      ✓ Should return correct list (173ms)
      ✓ Should return correct list after remove (532ms)
      ✓ Should return empty list when incorrect offset and limit (899ms)
```

owner
- ✓ Should return correct contract owner (202ms)

isRegistered
- ✓ Should return true if registered (155ms)
- ✓ Should return false if not registered (183ms)

remove
- ✓ Should fail if user does not have DEFAULT_ADMIN_ROLE (480ms)
- ✓ Should fail if vesting not found (331ms)
- ✓ Should correct remove (364ms)

setVestingImplementation
- ✓ Should fail if user does not have DEFAULT_ADMIN_ROLE (408ms)
- ✓ Should fail if wrong vesting implementation address (353ms)
- ✓ Should correct set vesting implementation address (367ms)

addCreator
- ✓ Should fail if user does not have DEFAULT_ADMIN_ROLE (408ms)
- ✓ Should correct add new CREATOR_ROLE (420ms)

createVesting
- ✓ Should fail if user does not have CREATOR_ROLE (448ms)
- ✓ Should correct transfer ownership (153ms)
- ✓ Should correct set ProxyAdmin (214ms)
- ✓ Should correct initialized (395ms)
- ✓ Should correct emit an event
- ✓ Should vesting work correctly (916ms)

Upgradeble case
- ✓ Should return new implementation address after create and new functionality (178ms)
- ✓ Should correct work with new functionality (457ms)
- ✓ Only Admin can call upgrade vesting (273ms)
- ✓ Should correct upgrade old vesting to new functionality (701ms)

removeCreator
- ✓ Should fail if user does not have DEFAULT_ADMIN_ROLE (347ms)
- ✓ Should correct remove CREATOR_ROLE (473ms)

Contract: Vesting
- ✓ GetInfo (146ms)

Deposit
- ✓ Should fail if incorrect sign (614ms)
- ✓ Should fail if nonce used before (254ms)
- ✓ if reward token decimals are less than staked token (4076ms)
- ✓ if reward token decimals and staked token decimals are equal (4218ms)

✓ Should fail if vesting is started (263ms)
✓ Should be can't transfer if is fiat (618ms)
✓ Check correct transfer (294ms)
✓ Check correct deposited (134ms)
✓ Check correct totalDeposited (544ms)
✓ Check set nonce (93ms)
Should fail if incorrect amount
✓ if more than max allocation (284ms)
✓ if more than min allocation (256ms)
✓ if more the remaining allocation (204ms)
✓ if more the remaining allocation (4429ms)
DOMAIN_SEPARATOR
✓ Should return _domainSeparatorV4() (85ms)
getAvailAmountToDeposit
✓ Should return 0, 0 if all allocation used (58ms)
✓ Should return 0, 0 if total currency is not bigger than total deposited (4948ms)
✓ Should get correct if remaining is less the max allocation (586ms)
✓ Should success (576ms)
setVesting
✓ Should revert if vesting type is linear but count period of vesting and period duration are zero (250ms)
✓ Should revert if interval unlocking part is invalid (2164ms)
✓ Should set vesting correctly (2748ms)
✓ Should revert if last interval unlocking part is invalid (2745ms)
✓ Should revert if interval starting timestamp is invalid (2685ms)
addDepositeAmount
✓ Should revert if caller, not owner (281ms)
✓ Should revert if arr length incorrect (238ms)
✓ Should revert if vesting is started (752ms)
✓ Should fail if not have enough allocation (407ms)
✓ Should correct set amount (661ms)
✓ Should correct change totalDeposited (514ms)
Deployed
✓ signer (103ms)
✓ Should revert initializing token if was initialized before (191ms)
✓ Should revert if total supply incorrect (2423ms)
✓ Should revert if rewardToken incorrect (332ms)
✓ Should revert if deposit token incorrect (279ms)

✓ Should revert if signer incorrect (304ms)
✓ Should revert if token price incorrect (2596ms)
✓ Should revert if allocation incorrect (705ms)
✓ Should revert if initial percentage incorrect (254ms)

getBalanceInfo

✓ Should return zero if zero deposited (113ms)
✓ Should return all balance how locked, if vesting doesn't start (106ms)
✓ Should return all unlocked balance if vesting started (602ms)
✓ Should return correct after harvest and by month linear (1683ms)

increaseTotalSupply

✓ Should increase total supply correctly (571ms)

setTimePoint

✓ Should revert setting if start date will be later than end date (239ms)

setSigner

✓ Should set signer correctly (434ms)
✓ Should revert if arrays have different sizes (250ms)

setSpecificAllocation

✓ Should set specific allocation correctly (253ms)
✓ Check specific allocation for user (778ms)
✓ Should revert if arrays have different sizes (229ms)

setSpecificVesting

✓ Should set specific vesting correctly (2993ms)
✓ Should revert if it was initialized before (3254ms)

harvest

✓ Should revert if vesting can't be started (287ms)
✓ Should correct transfer amount (924ms)

completeVesting

✓ Should revert if caller, not owner (277ms)
✓ Should revert if vesting can't be started (196ms)
✓ Should revert if Withdraw funds were called before (1335ms)
✓ No need to transfer tokens when everything is sold (6908ms)
✓ Should correct transfer amount (2071ms)

harvestFor

✓ Should correct transfer amount (1095ms)

Test case

✓ When VESTING_TYPE is SWAP (4901ms)
✓ When VESTING_TYPE is INTERVAL (10004ms)
✓ When one user both all tokens (8645ms)

```
    ✓ More users with specific cases (6358ms)
    ✓ Sales after increasing total supply (8314ms)

Contract: Vesting
  Upgradeable test
    ✓ Canot seconds call initialize methods (467ms)
    ✓ Correct deploy when deploying not in VestingFactory (1196ms)
  The check has an issue with interval pools
    ✓ When 10 intervals with and claim all in last action (2044ms)
    ✓ When 20 intervals with and claim all in last action (4363ms)

105 passing (10m)
```

Tests are written by Zokyo Security team

As part of our work assisting Gamestation in verifying the correctness of their contract code, our team was responsible for writing integration tests using the Truffle testing framework.

Tests were based on the functionality of the code, as well as a review of the Gamestation contract requirements for details about issuance amounts and how the system handles these.

Code Coverage

The resulting code coverage (i.e., the ratio of tests-to-code) is as follows:

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	UNCOVERED LINES
contracts\	99.47	99.79	100.00	97.40	
FixedSwap.sol	99.39	94.32	100.00	97.02	... 320, 324, 471
GameStationBridge.sol	100.00	100.00	100.00	100.00	
All files	**99.47**	**99.79**	**100.00**	**97.40**	

Test Results

```
Contract: Vesting
  Vesting Initializing Phase Test Cases
    ✓ should initialize vesting correctly (1720ms)
    ✓ shouldn't initialize vesting if address of reward token and deposit token is zero (578ms)
    ✓ shouldn't initialize vesting if max allocation is zero and less than min allocation (345ms)
    ✓ shouldn't initialize vesting if initialUnlockPercentage more than const (308ms)
    ✓ shouldn't initialize vesting if the signer is zero's address (300ms)
  Vesting Functions Phase Test Cases
    ✓ should initialize token correctly (1911ms)
    ✓ shouldn't initialize token if token was initialized (291ms)
    ✓ shouldn't initialize token if totalSypply_ or tokenPrice_ is zero (1650ms)
    ✓ should increase total supply correctly (848ms)
    ✓ shouldn't increase total supply if vesting is completed (1031ms)
```

- ✓ should set time point correctly (1666ms)
- ✓ shouldn't set time point if the date is incorrect (1088ms)
- ✓ should complete vesting correctly if soldToken less than _totalSupply (652ms)
- ✓ should complete vesting correctly if soldToken more than _totalSupply (2267ms)
- ✓ shouldn't complete vesting if vesting was not started yet (353ms)
- ✓ shouldn't complete vesting if vesting already completed (852ms)
- ✓ should set signer correctly (567ms)
- ✓ shouldn't set signer if the signer is zero's address (367ms)
- ✓ should set specific allocation correctly (970ms)
- ✓ shouldn't set specific allocation if diff array size (295ms)
- ✓ should set specific vesting correctly (592ms)
- ✓ shouldn't set specific vesting if specific vesting already set (723ms)
- ✓ shouldn't set specific vesting if linear vesting setup incorrect (422ms)
- ✓ should set vesting correctly (1993ms)
- ✓ shouldn't set vesting if interval unlocking part is incorrect (1452ms)
- ✓ shouldn't set vesting if lastUnlockingPart hasn't value of MAX_INITIAL_PERCENTAGE (1297ms)
- ✓ shouldn't set vesting if interval starting timestamp is invalid (1337ms)
- ✓ should add deposit amount correctly (1747ms)
- ✓ shouldn't add deposit amount if diff array's size (334ms)
- ✓ shouldn't add deposit amount if sale is closed (487ms)
- ✓ shouldn't add deposit amount if not enough allocation (630ms)
- ✓ should do deposit correctly if _fiat is false (1537ms)
- ✓ should do deposit correctly if _fiat is true (3696ms)
- ✓ shouldn't do deposit if nonce already used (1745ms)
- ✓ shouldn't do deposit if signer is incorrect (759ms)
- ✓ shouldn't do deposit if sale is closed (934ms)
- ✓ shouldn't do deposit if amount is incorrect (1248ms)
- ✓ should get domain separator correctly (315ms)
- ✓ should do harvest for caller correctly (2036ms)
- ✓ should do harvest for other account correctly (2393ms)
- ✓ shouldn't do harvest if vesting isn't started (1928ms)
- ✓ shouldn't do harvest if amount is zero (3355ms)
- ✓ should get calculate unlock correctly (3853ms)
- ✓ should get calculate unlock correctly if type of vesting is swap (3590ms)
- ✓ should get calculate unlock correctly if timestamp of interval more than current timestamp (3924ms)
- ✓ should do harvest with interval correctly (3740ms)

```
    ✓ shouldn't do harvest with interval if interval index is incorrect (3328ms)
    ✓ should convert to decimals correctly if _fromDecimals more than _toDecimals (5402ms)
    ✓ should convert to decimals correctly if _fromDecimals less than _toDecimals (5717ms)

Contract: VestingFactory
  VestingFactory Initializing Phase Test Cases
    ✓ should initialize vesting implementation correctly (216ms)
    ✓ shouldn't initialize vesting implementation if address is zero (417ms)
    ✓ should setup owner correctly (124ms)
    ✓ should set role of owner correctly (188ms)
  VestingFactory Functions Phase Test Cases
    ✓ should set vesting implementation correctly (413ms)
    ✓ shouldn't set vesting implementation if address is zero (404ms)
    ✓ should add vesting correctly (826ms)
    ✓ shouldn't add vesting if vesting already exists (1850ms)
    ✓ should remove exists vesting correctly (1414ms)
    ✓ shouldn't remove vesting if vesting isn't found (532ms)
    ✓ should add creator correctly (944ms)
    ✓ should remove creator correctly (1439ms)
    ✓ should create vesting correctly (2481ms)

62 passing (7m)
```

We are grateful to have been given the opportunity to work with the Gamestation team.

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.

Zokyo's Security Team recommends that the Gamestation team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

ZOKYO.