kasta

SMART CONTRACT AUDIT

ZOKYO.

October 28th, 2021 | v. 1.0

PASS

Zokyo's Security Team has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges

TECHNICAL SUMMARY

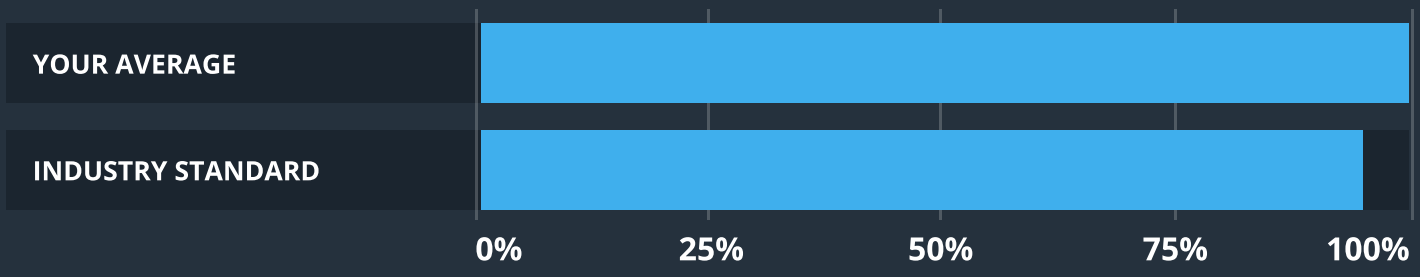
This document outlines the overall security of the Kasta smart contracts, evaluated by Zokyo's Blockchain Security team.

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

Contract Status

There were 2 critical issues found during the audit.

Testable Code



The testable code is 100%, 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 Kasta 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 Kasta repository.

Repository:
https://github.com/kasta-io/kasta-token/commit/36ef75da55000cafbbb6fe4a5ffe6fbe5970a23a

Last commit:
e1d2f71c75d04e8ae4f64c179741ab5a6934e0a4

Contracts:

- KastaToken;
- KastaTokenV1.

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 Kasta 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

The contracts provided for an audit are well written and structured. All the findings spotted within the auditing process are presented in this document.

Zokyo auditing team has found 2 critical, 3 issues with a low severity level, and a few informational issues. It is worth mentioning that both critical issues were resolved.

Based on the fixes provided by the Kasta team and on the quality and security of the codebase, Zokyo Security team can give a score of 98 to the audited smart contracts.

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

Risk transferring tokens that are staked

CRITICAL | RESOLVED

KastaToken/KastaTokenV1: Staked tokens are not transferred to any address (owner, contract, pool contract, .. etc). This shall allow the stakers to have control over their staked tokens and transfer the tokens wherever they like while at the same time their staked balance in KastaToken persists.

Consider this test scenario which explains the finding:

```
it.skip('CRITICAL: Holder is permitted to transfer staked tokens', async function () {
    const stakeAmount = BigInt(this.decimalsMultiplier * AVAILABLE_BALANCE).toString();
    await this.kastaToken.stake(stakeAmount, { from: accounts[2] });
    let tx = await this.kastaToken.transfer( accounts[3], stakeAmount, { from: accounts[2] });

    expectEvent(tx, 'Transfer');
    let balance3 = await this.kastaToken.balanceOf(accounts[3]);
    expect(balance3.toString()).to.be.equal(stakeAmount);
});
```

Here account3 normally receives the staked amount from account2 while the staked balance of account2 persists in KastaToken.

Recommendation:
Transfer the staked balance away from the staker's control.

Comment:
A working solution has been presented by the partner implemented in KastaTokenV1 but the solution is not applied to KastaToken yet.

Risk allocating tokens that are staked

CRITICAL | RESOLVED

KastaTokenV1: allocate(uint256, address, string) does not check the amount of tokens staked by the owner risking transferring an amount of token that is staked by the contract.

Recommendation:
Either check the amount staked by owner before allocation or disallow owner from staking his/her tokens. Or better off, it is solved accordingly by solving the first issue already.

Lock pragma to a specific version

LOW | RESOLVED

Lock the pragma to a specific version, since not all the EVM compiler versions support all the features, especially the latest one's which are kind of beta versions, So the intended behavior written in code might not be executed as expected.

Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, the latest compiler which may have higher risks of undiscovered bugs.

Correcting the required statements to be more precise

LOW | RESOLVED

KastaToken/KastaTokenV1: Since the transactions would fail if the required condition is not satisfied, So it is recommended to add a statement that justifies the reason for the failure of the transaction.

Recommendation:

1. In KastaToken.sol & KastaTokenV1.sol file, in stake(), replace

```
require(amount > 0, "Cannot stake 0");
```

with

```
require(amount > 0, "KastaToken: Cannot stake 0 tokens");
```

Similarly in unstake() replace

```
require(amount > 0, "Cannot unstake 0");
```

with

```
require(amount > 0, "KastaToken: Cannot unstake 0 tokens");
```

2. In KastaToken.sol & KastaTokenV1.sol files, in stake(), replace

```
require(!paused(), "KastaToken: token staking while paused");
```

with

```
require(!paused(), "KastaToken: Cannot stake tokens while paused");
```

Similarly in unstake() replace

```
"KastaToken: token unstaking while paused");
```

with

```
require(!paused(), "KastaToken: Cannot unstake tokens while paused");
```

3. In KastaToken.sol & KastaTokenV1.sol files, in unstake() replace

```
require(userbalanceStaked >= amount,  "Cannot unstake more than available" );
```

with

```
require(userbalanceStaked >= amount,  "KastaToken: Cannot stake more than staked balance" );
```

4. In KastaToken.sol & KastaTokenV1.sol files, in _beforeTokenTransfer(), replace

```
require(!paused(), "KastaToken: token transfer while paused");
```

with

```
require(!paused(), "KastaToken: Cannot transfer tokens while paused");
```

Tests not asserting the revert causes

LOW | RESOLVED

All tests are utilizing the statement catchRevert without asserting the reason of the revert. This might cause a false positive passing a test because it reverted for a reason while it was expected by the developer to revert for something else introducing undesired contract behavior.

Recommendation:
Use openzeppelin module test-helpers:

```
const { expectRevert} = require('@openzeppelin/test-helpers');
```

As follow:

```
await expectRevert( this.kastaToken.unpause ({ from: accounts[2] }), "KastaToken: must have pauser role to unpause" );
```

Transfer return boolean value not validated

INFORMATIONAL | RESOLVED

KastaTokenV1:

```
allocate(uint256, address, string):134:
```

does not validate the transfer going:

```
transfer(to, amount);
```

Recommendation:
Wrap the transfer call inside a required statement.

No tests for the implementation separately

INFORMATIONAL | RESOLVED

KastaToken: Contracts tested are the outputs of deployProxy and upgradeProxy while it is recommended to create separate tests for KastaToken separately according to the authors of the '@openzeppelin/truffle-upgrades' module themselves.

To test upgradeable contracts we should create unit tests for the implementation contract, along with creating higher-level tests for testing interaction via the proxy.

Potential testing bug

INFORMATIONAL | RESOLVED

stake_test.js:87 in this statement:

```
this.kastaToken.unstake(1, { from: accounts[2] })
```

Recommendation:
Missing await.

Unreachable code

INFORMATIONAL | RESOLVED

KastaToken/KastaTokenV1: require statement

```
require(!paused(), "KastaToken: token transfer while paused");
```

is unreachable because

```
super._beforeTokenTransfer(from, to, amount);
```

already reverts for the same reason if the contract is paused

```
require(!paused(), "ERC20Pausable: token transfer while paused");
```

Recommendation:
Might be better of removing that line as long as the base contract invokes the same require statement.

	KastToken	KastaTokenV1
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 are written by the Kasta 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\	100.00	87.50	100.00	100.00	
KastToken	100.00	85.00	100.00	100.00	
KastaTokenV1	100.00	89.29	100.00	100.00	
All files	**100.00**	**87.50**	**100.00**	**100.00**	

Test Results

Contract: KastaToken allocation amount
- ✓ cannot allocate 0 (790ms)
- ✓ can allocate less than available balance (99ms)
- ✓ can allocate exact available balance (93ms)
- ✓ cannot allocate more than the available balance (85ms)
- ✓ cannot allocate more than the available balance on different calls (192ms)

Contract: KastaToken allocation pausing
- ✓ cannot allocate while contract paused (103ms)
- ✓ allocation is allowed when the contract is unpaused (183ms)

Contract: KastaToken allocation role access
- ✓ cannot allocate without admin role (57ms)
- ✓ allocation is allowed when role is granted (93ms)
- ✓ cannot allocate with pauser role (207ms)

Contract: KastaToken individual tests – transfer
- ✓ cannot transfer tokens while contract is paused (154ms)
- ✓ can transfer tokens while contract is unpaused (400ms)

Contract: KastaToken grant roles
- ✓ deployer account should have admin role
- ✓ deployer account should have pauser role (38ms)
- ✓ deployer can grant admin role (103ms)
- ✓ deployer can grant pauser role (188ms)

Contract: KastaToken can't grant roles
- ✓ non deployer account shouldn't have admin role
- ✓ non deployer account shouldn't have pauser role
- ✓ non deployer can't grant admin role (282ms)
- ✓ non deployer can't grant pauser role (259ms)

Contract: KastaToken individual tests – staking
- ✓ cannot stake 0 (125ms)
- ✓ cannot stake while contract paused (147ms)
- ✓ stake is allowed when contract is unpaused (670ms)
- ✓ can stake less than balance (195ms)
- ✓ can stake exact available balance (262ms)
- ✓ cannot stake more than balance (313ms)
- ✓ cannot stake more than balance on different calls (622ms)
- ✓ admin role cannot stake tokens (105ms)

Contract: KastaToken individual tests – staking and balance
- ✓ staked tokens are transferred to contract address (392ms)
- ✓ cannot transfer staked tokens (580ms)

Contract: KastaToken individual tests – unstaking
- ✓ cannot unstake 0 (474ms)
- ✓ cannot unstake while contract paused (401ms)
- ✓ unstake is allowed when contract unpaused (1045ms)
- ✓ cannot unstake more than staked (683ms)
- ✓ can unstake exact available balance (444ms)
- ✓ cannot unstake more than balance on different calls (986ms)

Contract: KastaToken individual tests – unstaking and balance
- ✓ unstaked tokens are transferred back to the user (1020ms)

Contract: KastaToken (proxy)
- ✓ returns a value previously initialized (159ms)

Contract: KastaToken upgradeability
- ✓ balances remains after contract upgrade (57ms)
- ✓ user can unstake after contract upgrade (464ms)

Contract: KastaToken grant roles
- ✓ deployer account should have admin role (38ms)
- ✓ deployer account should have pauser role (52ms)
- ✓ deployer can grant admin role (766ms)
- ✓ deployer can grant pauser role (1276ms)

Contract: KastaToken can't grant roles
- ✓ non deployer account shouldn't have admin role (38ms)
- ✓ non deployer account shouldn't have pauser role (202ms)
- ✓ non deployer can't grant admin role (946ms)
- ✓ non deployer can't grant pauser role (1480ms)

Contract: KastaToken staking
- ✓ cannot unstake 0 (750ms)
- ✓ cannot stake while contract paused (523ms)
- ✓ stake is allowed when contract unpaused (9444ms)
- ✓ can stake less than balance (3330ms)
- ✓ can stake exact available balance (3465ms)
- ✓ cannot stake more than balance (72ms)
- ✓ cannot stake more than balance on different calls (320ms)
- ✓ admin role cannot stake tokens (130ms)

Contract: KastaToken staking and balance
- ✓ staked tokens are transferred to contract address (548ms)
- ✓ cannot transfer staked tokens (222ms)
- ✓ non deployer can't grant admin role (946ms)
- ✓ non deployer can't grant pauser role (1480ms)

Contract: KastaToken transfer

- ✓ cannot transfer tokens while contract is paused (234ms)
- ✓ can transfer tokens while contract is unpaused (305ms)

Contract: KastaToken unstaking

- ✓ cannot unstake 0 (107ms)
- ✓ cannot unstake while contract paused (826ms)
- ✓ unstake is allowed when contract unpaused (1123ms)
- ✓ cannot unstake more than staked (869ms)
- ✓ can unstake exact available balance (926ms)
- ✓ cannot unstake more than balance on different calls (1419ms)

Contract: KastaToken unstaking and balance

- ✓ unstaked tokens are transferred back to the user (779ms)
- ✓ cannot stake while contract paused (116ms)
- ✓ stake is allowed when contract is unpaused (178ms)

67 passing (5m)

Tests written by Zokyo Security team

As part of our work assisting Kasta 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 Kasta 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\	100.00	93.75	100.00	100.00	
KastToken	100.00	90.00	100.00	100.00	
KastaTokenV1	100.00	96.43	100.00	100.00	
All files	**100.00**	**93.75**	**100.00**	**100.00**	

Test Results

Contract: KastaTokenV0 – Fix-2

- ✓ can stake less than balance (251ms)
- ✓ can stake exact available balance (251ms)
- ✓ can unstake less than staked balance (219ms)
- ✓ can unstake exact staked balance (308ms)
- ✓ multiple unstaking with balance checking till failure (1045ms)

Contract: KastaToken V0 – logic only

- ✓ pause() - paused successfully by PAUSER_ROLE holder (126ms)
- ✓ pause() - pause fail by caller not a PAUSER_ROLE (84ms)
- ✓ unpause() - unpaused successfully by PAUSER_ROLE holder (194ms)
- ✓ unpause() - unpause fail by caller not a PAUSER_ROLE (174ms)
- ✓ _beforeTokenTransfer - ensure token is transferrable (132ms)

✓ _beforeTokenTransfer - token is not transferrable if paused (181ms)
✓ cannot stake 0 (81ms)
✓ cannot stake while contract paused (199ms)
✓ stake is allowed when contract unpaused (284ms)
✓ can stake less than balance (134ms)
✓ can stake exact available balance (110ms)
✓ cannot stake more than balance (79ms)
✓ cannot stake more than balance on different calls (432ms)
✓ cannot unstake 0 (75ms)
✓ cannot unstake while contract paused (326ms)
✓ unstake is allowed when contract unpaused (598ms)
✓ cannot unstake more than staked (189ms)
✓ can unstake exact available balance (399ms)
✓ cannot unstake more than balance on different calls (494ms)
✓ balanceStaked - shows staked balance (271ms)

Contract: KastaTokenV1
✓ can stake less than balance (196ms)
✓ can stake exact available balance (273ms)
✓ stake() - Token preserve its token balance across updates - address kept (710ms)
✓ can unstake less than staked balance (562ms)
✓ can unstake exact staked balance (516ms)
✓ multiple unstaking with balance checking till failure (689ms)
✓ owner cannot allocate staked tokens of his/her (123ms)

Contract: KastaTokenV1
✓ returns a value previously initialized
✓ pause() - paused successfully by PAUSER_ROLE holder (97ms)
✓ pause() - pause fail by caller not a PAUSER_ROLE (86ms)
✓ unpause() - unpaused successfully by PAUSER_ROLE holder (343ms)
✓ unpause() - unpause fail by caller not a PAUSER_ROLE (204ms)
✓ _beforeTokenTransfer - ensure token is transferrable (134ms)
✓ _beforeTokenTransfer - token is not transferrable if paused (367ms)
✓ cannot stake 0 (135ms)
✓ cannot stake while contract paused (336ms)
✓ stake is allowed when contract unpaused (407ms)
✓ can stake less than balance (104ms)

- ✓ can stake exact available balance (272ms)
- ✓ cannot stake more than balance (148ms)
- ✓ cannot stake more than balance on different calls (580ms)
- ✓ cannot unstake 0 (177ms)
- ✓ cannot unstake while contract paused (412ms)
- ✓ unstake is allowed when contract is unpaused (809ms)
- ✓ cannot unstake more than staked (288ms)
- ✓ can unstake exact available balance (335ms)
- ✓ cannot unstake more than balance on different calls (841ms)
- ✓ balanceStaked - shows staked balance (417ms)
- ✓ cannot allocate 0 (89ms)
- ✓ can allocate less than available balance (220ms)
- ✓ can allocate exact available balance (266ms)
- ✓ cannot allocate more than available balance (125ms)
- ✓ cannot allocate more than available balance on different calls (510ms)
- ✓ cannot allocate while contract paused (279ms)
- ✓ allocation is allowed when contract is unpaused (379ms)
- ✓ cannot allocate without admin role (115ms)
- ✓ allocation is allowed when role is granted (131ms)
- ✓ cannot allocate with pauser role (262ms)

Contract: KastaToken grant roles

- ✓ deployer account should have the admin role (41ms)
- ✓ deployer account should have pauser role
- ✓ deployer can grant admin role (175ms)
- ✓ deployer can grant pauser role (141ms)

Contract: KastaToken can't grant roles

- ✓ non deployer account shouldn't have admin role
- ✓ non deployer account shouldn't have pauser role
- ✓ non deployer can't grant admin role (285ms)
- ✓ non deployer can't grant pauser role (272ms)

71 passing (2m)

We are grateful to have been given the opportunity to work with the Kasta 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 Kasta team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

ZOKYO.