Crowny

SMART CONTRACT AUDIT

ZOKYO.

May 28, 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

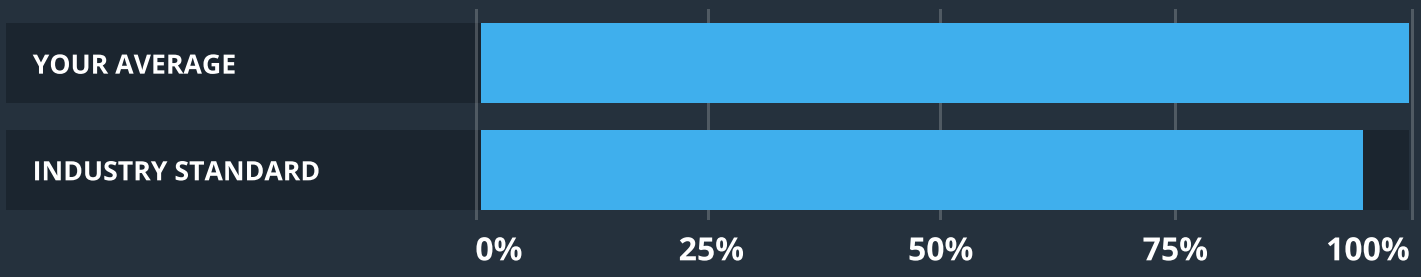
This document outlines the overall security of the Crowny smart contracts, evaluated by Zokyo's Blockchain Security team.

The scope of this audit was to analyze and document the Crowny 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 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 Crowny 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 Crowny repository – https://github.com/Crowny-io/zokyo-audit/tree/.
Initial commit – f2b982a54b9de27e28f57dff1ab90d02c5ca7bb8.
Last commit – 2b682587a0e861f19f21a9e73a954b0e4c776046.

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 Crowny 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.	3	Testing contract logic against common and uncommon attack vectors.
2	Cross-comparison with other, similar smart contracts by industry leaders.	4	Thorough, manual review of the codebase, line-by-line.

EXECUTIVE SUMMARY

In this version of the audit report only staking contract was taken into account. All the findings and conclusions are related to the aforementioned contract.

There were no critical issues found during the audit. All the mentioned findings may have an effect only in the case of specific conditions performed by the contract owner.

Contracts are well written and structured. The findings during the audit have no impact on contract performance, so it is fully production-ready.

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

CrownyStaking earlyReward calculation logic may be wrong

HIGH | RESOLVED

There are two factors that can make earlyReward lower than expected in some business logics(which can be intentional, but it's better to double-check):

1. Actual reward per withdrawn token is interpolated between a minimum of (withdrawBegins-stakingEnd) / (fullMaturity -stakingEnd)*(earlyWithdrawReward / totalStaked) at withdrawBegins to earlyWithdrawReward / totalStakedclose to fullMaturity. At the moment of maturity reward per token jumps from earlyWithdrawReward / totalStaked to totalReward / totalStaked. To avoid this reward jump at a moment of maturity reward per token can be interpolated from earlyWithdrawReward / totalStaked at a withdrawBegins to a totalReward / totalStaked at a fullMaturity.
2. Each early withdrawal makes withdrawals after a maturity more profitable(by making the proportion of rewardBalance / stakedBalance bigger. But this doesn't affect early withdrawals that will be made later, because they are based on an unchanged proportion of earlyWithdrawReward / totalStaked.

Recommendation:
Review earlyReward calculations on compliance with required business logic.

CrownyStaking stakingStart value doesn't actually restrict staking

HIGH | RESOLVED

In a CrownyStaking contract, the stakingStart state variable is set but isn't used anywhere. It's possible that this variable should actually restrict stake function via isAfter modifier.

Recommendation:
Add stakingStart restriction logic or remove this variable if it doesn't actually need to be used.

CrownyStaking setStakingReward earlyWithdrawReward parameter restriction can be relaxed

MEDIUM | RESOLVED

setStakingReward doesn't set a total reward token amount, but actually increases it. So the validation check on _earlyWithdrawReward parameter can be changed to be less than totalReward after it's increased by _fullMaturityReward instead of checking that it's less than _fullMaturityReward.

Recommendation:
Rework setStakingReward _earlyWithdrawReward amount validation check.

Inconsistent order of modifiers in CrownyStaking contract

INFORMATIONAL | RESOLVED

CrownyStaking functions setStakingReward, stake and withdraw contain custom modifiers before visibility function modifier(external), which is inconsistent with other places in a code and Solidity Style Guide.

Recommendation:
Place custom modifiers after the visibility modifier in the functions above.

Re-audit:
The issue persists for setStakingReward function, custom modifier onlyOwner shouldn't precede visibility modifier onlyOwner.

SafeMath can be unnecessary

INFORMATIONAL | RESOLVED

Starting from Solidity 0.8.0 overflow revert checks are enabled by a compiler by default, so SafeMath library defaults to doing regular operations in many cases. Removing it can save a small amount of gas at the deployment.

Recommendation:
Review the necessity to use SafeMath through contracts.

CrownyStaking stakedBalance and rewardBalance doesn't change on withdrawals after the maturity is reached

INFORMATIONAL | RESOLVED

On withdrawals, after the maturity is reached, stakedBalance and rewardBalance public state variables aren't lowered accordingly. This doesn't affect contract functionality because the withdrawal proportion doesn't change and it saves gas on storage writes. But this can be confusing to outside users accessing variables through respective getters.

Recommendation:
Consider removing public modifiers from stakedBalance and rewardBalance state variables.

Gas can be saved by making some state variables immutable

INFORMATIONAL | RESOLVED

If a state variable is only set in a constructor and isn't changed afterward, gas can be saved by adding an immutable modifier to those variables. This will inline those values in deployed contract bytecode and avoid storage lookups when accessing those variables.

Recommendation:
Make state variables that are only modified on a constructor to immutable.

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Secured team

As part of our work assisting Crowny 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 Crowny contract requirements for details about issuance amounts and how the system handles these.

```
Contract: CrownyStaking
  Setup
    ✓ should deploy tokens (2555ms)
  On stake 1
    Setup
      ✓ should deploy contracts (769ms)
    On user
      ✓ should send reward tokens to staking contract (718ms)
      ✓ admin should set staking rewards (1157ms)
      ✓ should stake (2043ms)
      ✓ should call stakedToken (94ms)
      ✓ should call rewardToken (123ms)
      ✓ should call maturityPeriod (141ms)
      ✓ should call availableStakingAmount (139ms)
      ✓ should call stakedAmount (141ms)
      ✓ should call matureReward (93ms)
      ✓ should wait (5001ms)
      ✓ should withdraw rewards before maturity (609ms)
      ✓ should call timeElapsed (124ms)
      ✓ should wait (5006ms)
      ✓ should call claimUnrewardedTokens (174ms)
  On stake 2
    Setup
      ✓ should deploy contracts (826ms)
    On user
      ✓ should stake (1850ms)
```

```
    ✓ should wait (5010ms)
    ✓ should withdraw rewards after maturity (1561ms)
  On reverts
    Constructor
      ✓ _stakingStart < _stakingEnd (1373ms)
      ✓ _stakingEnd < _withdrawBegins (669ms)
      ✓ _withdrawBegins < _fullMaturity (637ms)
      ✓ _stakingCap > (754ms)
    On Admin
      ✓ should NOT call claimUnrewardedTokens [rewardBalance > 0] (839ms)
      ✓ should NOT call claimUnrewardedTokens [stakedBalance == 0] (8085ms)
      ✓ should NOT call setStakingReward [earlyWithdrawReward < totalReward] (2099ms)
    On Modifiers
      ✓ should trigger hasAllowance (1906ms)
      ✓ should trigger isBefore (1961ms)
      ✓ should trigger isAfter (1951ms)
    On User
      ✓ should NOT stake [amount is zero] (2182ms)
      On withdraw
        ✓ should deploy contracts (841ms)
        ✓ should wait (1010ms)
        ✓ should NOT withdraw [availableForWithdraw > 0] (376ms)
      On withdraw
        ✓ should deploy contracts (950ms)
        ✓ should stake (1816ms)
        ✓ should wait (5010ms)
        ✓ should NOT withdraw [availableForWithdraw > 0] (250ms)


  38 passing (1m)
```

FILE	% STMTS	% BRANCH	% FUNCS	% LINES
contracts/	100.00	96.43	100.00	100.00
CrownyStaking.sol	100.00	96.43	100.00	100.00
All files	**100.00**	**96.43**	**100.00**	**100.00**

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

ZOKYO.