Stampsdaq®

SMART CONTRACT AUDIT

ZOKYO.

Dec 14th, 2021 | v. 1.0

PASS

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

SCORE

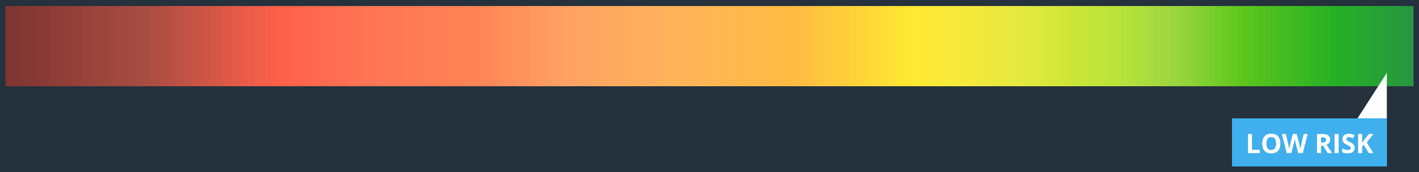
99

TECHNICAL SUMMARY

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

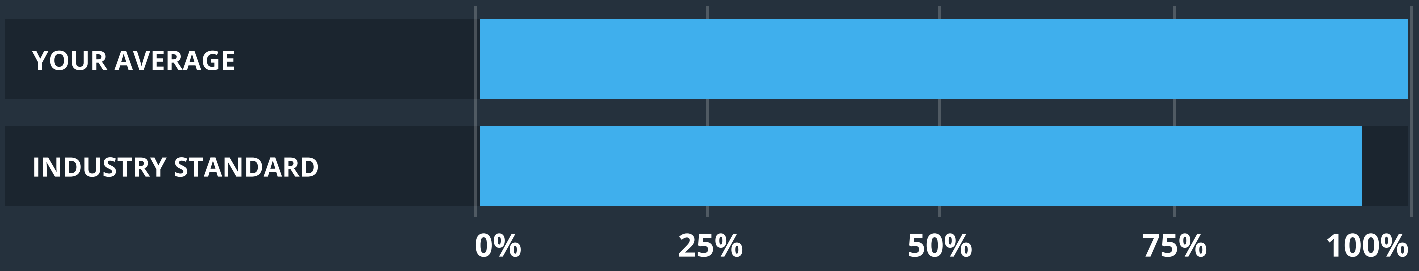
The scope of this audit was to analyze and document the StampsDaq smart contract codebase for quality, security, and correctness.

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 StampsDaq 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 StampsDaq repository.

Repository:
https://github.com/STAMPSDAQ-LLC/audit/commit/
ecbbe07fa63327080413ef1aa6a2487a8f65997e

Last commit:
e9d6cd59915e71ecacf00e7ecc9ed91071f9008c

Contracts under the scope:

- ComissionManager;
- DestructibleInterface;
- OwnableInterface;
- Postdollar;
- RewardPool;
- Sell;
- StampToken.

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 StampsDaq 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 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.

During the auditing process, our auditor's team found 1 issue with a high severity level, 2 issues with a low severity level, and a couple of informational issues which were successfully resolved by the StampsDaq team.

Based on the results of the audit, we can give a score of 99 and state that the audited contracts are 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

HIGH | RESOLVED

The contracts that are implementing the DesctructibleInterface give the owner too much control, the owner can call the destroyAndSend function and transfer all the ether from the contracts to an arbitrary address this is a risk especially if the private key of the owner gets compromised, this kind of functionalities should be used with a multi-sig.

Recommendation:
Implement a multi-signature functionality to be able to call the functions from the Destructible contract.

LOW | RESOLVED

An malicious entity can forcefully send ether to the RewardPool contract by creating another contract, sending ether to it, and then calling the selfdestruct function with the parameter being the RewardPool address, and then call the fill function with value 0, this way making the _fill variable true, but the value of the other variables like _collectorPool will be 0, this holds no security risk at the moment from what I can observe because funds can be unlocked by adding more ether through the fill function or by calling self-destruct with the parameter being the owner address, but it’s clearly not intended behavior because from the design of the RewardPool contract we can observe it is built to not accept ether through the fallback or receive functions.

Recommendation:
To mitigate this exact scenario where _fill variable will be true and the other variables will have the value 0, add a sanity check for the msg.value variable at the beginning of the function.

LOW | RESOLVED

In the RewardPool contract, the functions payReward and fill should be made external to optimize the gas cost, the external functions are cheaper to call than the public function, and there are no cases when you call these functions internally, so it will make more sense to be external.

Recommendation:
Change the accessibility of the functions payReward and fill from public to external to optimize gas costs.

INFORMATIONAL | RESOLVED

Some require functions in the project (for example RewardPool, lines 49, 65, 66, 68, 72,...), does not contain a message, to be fully compatible with best standards, all required messages should contain a message, to be able to tell where the error happened in case of revert.

Recommendation:
Be sure that all the required functions from the project contain a descriptive error message.

INFORMATIONAL | RESOLVED

For gas optimization and reusability the function _compareStrings from RewardPool, _concat, and _uintToString from StampToken should be moved in a library and imported from there.

Recommendation:
Create libraries to improve accessibility, modularity, and gas costs.

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)

	ComissionManager	DestructibleInterface
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

	OwnableInterface	Postdollar
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

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

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security team

As part of our work assisting StampsDaq 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 StampsDaq 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
ComissionManager.sol	100.00	88.99	100.00	100.00	
DestructibleInterface.sol	100.00	100.00	100.00	100.00	
Helpers.sol	100.00	100.00	100.00	100.00	
OwnableInterface.sol	100.00	100.00	100.00	100.00	
Postdollar.sol	100.00	100.00	100.00	100.00	
RewardPool.sol	100.00	85.71	100.00	100.00	
Sell.sol	100.00	87.59	100.00	100.00	
StampToken.sol	100.00	100.00	100.00	100.00	
All files	**100.00**	**91.00**	**100.00**	**100.00**	

Test Results

```
Contract: Postdollar
  constructor
    ✓ should deploy with correct token name (158ms)
    ✓ should deploy with correct token symbol (146ms)
    ✓ should mint initial supply to owner correctly (125ms)
  Functions:
    transfer
      ✓ should transfer tokens correctly (442ms)
      ✓ cannot transfer with zero sender address (529ms)
      ✓ cannot transfer with zero recipient address (256ms)
      ✓ cannot transfer more tokens than sender balance (208ms)
    transferFrom
      ✓ should transfer tokens correctly (603ms)
      ✓ cannot transfer with zero sender address (246ms)
      ✓ cannot transfer with zero recipient address (260ms)
      ✓ cannot transfer more tokens than sender balance (274ms)
    approve
      ✓ should approve tokens correctly (291ms)
      ✓ cannot approve with zero spender address (221ms)
    allowance
      ✓ should return allowance correctly (97ms)
      ✓ should increase allowance correctly (277ms)
      ✓ should decrease allowance correctly (353ms)
      ✓ cannot decrease allowance if subtracted amount more than current allowance (284ms)

Contract: ComissionManager
  Functions:
    updateGlobalComission
      ✓ should revert if co-owners are not set (1587ms)
      ✓ only owner or co-owners can update the global commission (310ms)
      ✓ should update global commission correctly (374ms)
      ✓ should revert if new global commission is greater than 333 or is equal to 0 (545ms)
    setExclusiveComission
      ✓ should set special commission for one address correctly (330ms)
      ✓ should revert if exclusive commission owner is a zero address or this contract (479ms)
      ✓ should revert if special commission is greater than 333 or is equal to 0 (288ms)
```

```
        getComission
          ✓ should return commission value for specified address correctly (97ms)
          ✓ should return the global commission if the special commission for one address is zero (87ms)
          ✓ should revert if the payer is a zero address or this contract (193ms)
        getComissionedPrice
          ✓ should return commissioned price for specified address correctly (546ms)
          ✓ should revert if the payer is a zero address or this contract (233ms)
        receive
          ✓ should revert (156ms)
        fallback
          ✓ should revert (179ms)

  Contract: RewardPool
      Functions:
        fill
          ✓ should revert if co-owners are not setted (1061ms)
          ✓ should receive and distribute rewards correctly (310ms)
          ✓ should catch event (41ms)
        payReward
          ✓ should revert if the pool is not filled (237ms)
          ✓ should send rewards correctly (1386ms)
          ✓ should catch event
          ✓ should revert if the payee is a zero address or this contract (248ms)
          ✓ should revert if the amount exceeds pool balance (1348ms)
        receive
          ✓ should revert (184ms)
        fallback
          ✓ should revert (105ms)
        destroyAndSend/destroy
          ✓ should revert if the recipient is a zero address (317ms)
          ✓ should terminate contract and send balance to the recipient (1322ms)

  Contract: StampToken
      constructor
        ✓ should deploy with correct token name (106ms)
        ✓ should deploy with correct token symbol (103ms)
      Functions:
        singleEmission
          ✓ should mint token sequential correctly (930ms)
        ✓ should set uri correctly for the created token (127ms)
      multipleEmission
        ✓ should mint multiple sequential tokens correctly (1491ms)
        ✓ should set uri correctly for the created tokens (794ms)

  Contract: Sell
    constructor
      ✓ cannot deploy if the STAMPSDAQ NFT contract is a zero address or this contract (643ms)
      ✓ cannot deploy with zero price (709ms)
      ✓ cannot deploy if the deployer does not own the token (1220ms)
    Functions:
      complete
        ✓ should revert if amount is less than endPrice (747ms)
        ✓ should revert if token is not approved (867ms)
        ✓ should revert if _tokenOwner address does not own the token (1461ms)
        ✓ should buy token correctly (5426ms)
        ✓ should catch event
        ✓ should revert if the offer is already complete (329ms)
      getPayablePrice
        ✓ should return commissioned price for payer address correctly (346ms)
      receive
        ✓ should revert (430ms)
      fallback
        ✓ should revert (147ms)

  61 passing (50s)
```

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

ZOKYO.