deficliq

SMART CONTRACT AUDIT

ZOKYO.

Feb 2, 2021 | v. 2.0

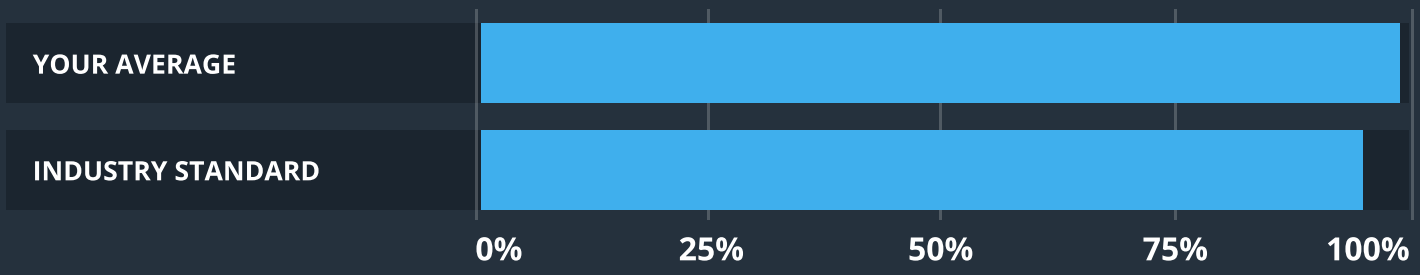
TECHNICAL SUMMARY

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

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

There were 2 critical and 1 high issue found during the manual audit which were successfully resolved.

Testable Code



Testable code is 99.07% 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 Cliq 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 4

Summary 6

Structure and Organization of Document 7

Complete Analysis 8

Code Coverage and Test Results for all files 13

- Tests written by Cliq team (original test coverage) 13
- Tests Written by Zokyo 13

AUDITING STRATEGY AND TECHNIQUES APPLIED

The Smart contract's source code was taken from the v1:
https://github.com/quary1993/deficliq-staking/commit/3e210711f51f64dbd94366a63873cd0d39a05090

v2:
https://github.com/quary1993/deficliq-staking/commit/bb11e4fcb93ae42c02ef0f111aafb567ff8b6b4a

v3:
https://github.com/quary1993/deficliq-staking/commit/cfb8c5a74c622253c2e272e4da0612adffcff92f

Requirements:
https://docs.google.com/document/d/1mJ8N5ThyPr1-wQBQK4927ZVfndZrL7Yv/edit
(file checksum SHA-1 e198a15453854428e794327e89534cc73a04bd7d)

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;
- Meets best practices in code readability, etc.

Zokyo's Security Team has followed best practices and industry-standard techniques to verify the implementation of Cliq Staking 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

There were 2 critical and 1 high issue found during the manual audit. All of them were resolved except 1 medium and 1 informational issue. The issue marked with a medium level has no effect on how smart contract work but may make code more clear, readable, and save gas during deployment. The mentioned informational ranking finding may have an effect only in case of specific conditions and may not have an effect on regular contract performance.

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

The days in packages not in seconds

CRITICAL | RESOLVED

In the constructor, you define 3 packages (lines 85, 86, 87) in which you set parameter _daysLocked in days (eg. 30). That parameter used to calculate reward. In functions checkStakeReward and checkStakeCliqReward in line 175 and 222 calculate yieldPeriods according to the next formula:

yieldPeriods = time passed from stake (in seconds) / daysLocked defined in packages (in days).

In case when a user stake 10 tokens for 1 day and then unstake, yieldPeriods will be 86400 / 30 = 2880, which means his reward will be absurdly huge.

Recommendation:
At initialization of packages use a lock period in seconds.

Infinite unstake

CRITICAL | RESOLVED

The changes to unstake function make it possible to continuously unstake the once staked amount and get the reward continuously instead of making it once.

Recommendation:
Add line

```
stakes[msg.sender][stakeIndex]._withdrawnTimestamp = block.timestamp;
```

before line 259. Move lines 268-273 to 286.

Checks Effects Interactions Pattern

HIGH | RESOLVED

Function unstake and forceWithdraw does not follow the pattern Checks Effects Interactions which makes them vulnerable to re-entrancy attack.

Recommendation:
Reduce the attack surface for malicious contracts trying to hijack control flow after an external call. For unstake: move line 351 before transfer stake amount (lines 332 and 342). For forceWithdraw: move line 376 before transfer stake amount (line 372).

Duplicate of code

MEDIUM | UNRESOLVED

Public functions checkStakeReward, checkStakeCliqReward of the cliqStaking contract, can be merged, they are almost the same. The calculations of reward can be switched in the same way as made in unstake functions.

Recommendation:
Consider merging of these functions - it brings about readability and allows you to save gas when deploying contracts.

Unused parameters in struct YieldType

MEDIUM | RESOLVED

The Parameter _earlyPenalty and _daysBlocked of struct YieldType are used nowhere.

In require on lines 267, 294, 336 must be used _daysBlocked instead of _daysLocked?

Recommendation:
Consider delete unused parameters.

Order of Layout

INFORMATIONAL | RESOLVED

Layout contract elements in cliqStaking 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 order of layout according to solidity documentation: Order of Layout.

Order of Functions

INFORMATIONAL | RESOLVED

The functions in cliqStaking contract are not grouped according to their visibility and order.

Functions should be grouped according to their visibility and ordered in the following way:

- constructor;
- fallback function (if exists);
- external;
- public;
- internal;
- private.

Ordering helps readers to identify which functions they can call and to find the constructor and fallback definitions easier.

Recommendation:
Consider changing functions order according to solidity documentation: Order of Functions.

Constant State Variables

INFORMATIONAL | RESOLVED

The declaration of the constant state variable of the CliqStaking contract missed keyword constant (90 line).

Moreover, according to the OpenZeppelin docs, the roles should be unique and the best way to achieve this is by using public constant hash digests:

```
bytes32 public constant MY_ROLE = keccak256("MY_ROLE");
```

Recommendation:
Make sure that all variables named as constant should have keyword constant.

Inconsistent coding style

INFORMATIONAL | RESOLVED

There are minor deviations from cliqStaking coding style. In particular private functions, should be prepended with an underscore to explicitly denote their visibility.

It brings about readability and allows you to adhere to a consistent coding style in smart contracts.

Recommendation:
Make sure that all functions with private visibility are named prepended with an underscore.

Useless require

INFORMATIONAL | RESOLVED

In functions, checkStakeReward on line 174 and checkStakeCliqReward on line 219 are useless requirements. The case when timeDiff will be < 0 possible only when stakingTime > currentTime but in these cases transaction will be reverted with the message "SafeMath: subtraction overflow" (SafeMath).

Recommendation:
Delete useless require lines 174, 219.

Naming of constants

INFORMATIONAL | RESOLVED

The public constant does not follow Naming Conventions in Solidity. Constants should be named with all capital letters with underscores separating words.

Recommendation:
Consider rename constant name to follow best practice of coding style.

Invalid condition of require, reverted with the wrong message

INFORMATIONAL | UNRESOLVED

In functions unstake and forceWithdraw invalid condition of require. If we provide a stake index for a not defined stake it will be reverted with a message "invalid opcode" instead of the message from require.

Recommendation:
Change condition of require, eg.:

```
stakeIndex < stakes[msg.sender].length.
```

Rewriting _withdrawnTimestamp

INFORMATIONAL | UNRESOLVED

In function unstake on line 281 rewriting _withdrawnTimestamp. This line can be deleted since we set it above (line 256).

Recommendation:
Delete unneeded line of code, it will save gas during each call of unstake.

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Cliq team (original test coverage)

The Cliq team did not write or provide any integration/e2e tests.

Tests Written by Zokyo

Execution Report

```
[romario@legion deficliq-staking]$ truffle run coverage

> Using Truffle library from local node_modules.

> server:           http://127.0.0.1:8555
> truffle:          v5.1.59
> ganache-core:     v2.13.0
> solidity-coverage: v0.7.14

Network Info
============
> id:      *
> port:    8555
> network: soliditycoverage


Instrumenting for coverage...
=============================

> cliqStaking.sol
> mock/cliq.sol
> mock/ERC20Mock.sol

Coverage skipped for:
=============================
```

```
> Migrations.sol

Compiling your contracts...
===========================
✓ Fetching solc version list from solc-bin. Attempt #1
> Compiling ./.coverage_contracts/Migrations.sol
> Compiling ./.coverage_contracts/cliqStaking.sol
> Compiling ./.coverage_contracts/mock/ERC20Mock.sol
> Compiling ./.coverage_contracts/mock/cliq.sol
> Compiling @openzeppelin/contracts/GSN/Context.sol
> Compiling @openzeppelin/contracts/access/AccessControl.sol
> Compiling @openzeppelin/contracts/math/SafeMath.sol
> Compiling @openzeppelin/contracts/token/ERC20/ERC20.sol
> Compiling @openzeppelin/contracts/token/ERC20/ERC20Burnable.sol
> Compiling @openzeppelin/contracts/token/ERC20/ERC20Capped.sol
> Compiling @openzeppelin/contracts/token/ERC20/IERC20.sol
> Compiling @openzeppelin/contracts/utils/Address.sol
> Compiling @openzeppelin/contracts/utils/EnumerableSet.sol
✓ Fetching solc version list from solc-bin. Attempt #1
> Compilation warnings encountered:

    /home/romario/Work/Audit/Cliq/last_commit/deficliq-staking/.coverage_contracts/mock/cliq.sol:
Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a
comment containing "SPDX-License-Identifier: <SPDX-License>" to each source file. Use
"SPDX-License-Identifier: UNLICENSED" for non-open-source code. Please see https://spdx.org for more
information.
,@openzeppelin/contracts/token/ERC20/ERC20.sol:55:5: Warning: Visibility for constructor is ignored. If
you want the contract to be non-deployable, making it "abstract" is sufficient.
    constructor (string memory name_, string memory symbol_) public {
    ^ (Relevant source part starts here and spans across multiple lines).
,@openzeppelin/contracts/token/ERC20/ERC20Capped.sol:19:5: Warning: Visibility for constructor is
ignored. If you want the contract to be non-deployable, making it "abstract" is sufficient.
    constructor (uint256 cap_) internal {
    ^ (Relevant source part starts here and spans across multiple lines).
,/home/romario/Work/Audit/Cliq/last_commit/deficliq-staking/.coverage_contracts/mock/cliq.sol:13:5:
Warning: Visibility for constructor is ignored. If you want the contract to be non-deployable, making it
"abstract" is sufficient.
    constructor(
```

```
    ^ (Relevant source part starts here and spans across multiple lines).
,/home/romario/Work/Audit/Cliq/last_commit/deficliq-staking/.coverage_contracts/cliqStaking.sol:10:1:
Warning: Contract code size exceeds 24576 bytes (a limit introduced in Spurious Dragon). This contract
may not be deployable on mainnet. Consider enabling the optimizer (with a low "runs" value!), turning off
revert strings, or using libraries.
contract CliqStaking is AccessControl {
^ (Relevant source part starts here and spans across multiple lines).

> Artifacts written to
/home/romario/Work/Audit/Cliq/last_commit/deficliq-staking/.coverage_artifacts/contracts
> Compiled successfully using:
   - solc: 0.7.6+commit.7338295f.Emscripten.clang


Compiling your contracts...
===========================
✓ Fetching solc version list from solc-bin. Attempt #1
> Everything is up to date, there is nothing to compile.

Contract: CliqStaking
    check basic init
      ✓ has a name
      ✓ has a totalStakedFunds
      ✓ has a REWARD_PROVIDER
      check constructor
        ✓ should set a staked token
        ✓ should set a Cliq token
        ✓ should set a role
        should define packages
          ✓ should set 3 packages
          Silver Package
            ✓ has a name
            ✓ has a days period
            ✓ has a days blocked
            ✓ has a percentage interest
            ✓ has amount of Cliq for each 1mln tokens staked
          Gold Package
            ✓ has a name
```

```
        ✓ has a days period
        ✓ has a days blocked
        ✓ has a percentage interest
        ✓ has amount of Cliq for each 1mln tokens staked
      Platinum Package
        ✓ has a name
        ✓ has a days period
        ✓ has a days blocked
        ✓ has a percentage interest
        ✓ has amount of Cliq for each 1mln tokens staked
  Functions
    stakesLength
      ✓ should return correct stakes length (2219ms)
    packageLength
      ✓ should be a 3 packages (64ms)
    stakeTokens
      ✓ should revert staking on pause (318ms)
      ✓ should revet if _amount !> 0 (57ms)
      ✓ should revert if no staking package (48ms)
      ✓ should revert if stake reward type not known (228ms)
      ✓ should add to totalStakedBalance (384ms)
      ✓ should add to stakes (306ms)
      ✓ should update hasStaked (166ms)
      ✓ should transfer token (219ms)
      ✓ should catch Transfer event (123ms)
      ✓ should catch StakeAdded event (165ms)
    checkStakeReward
      ✓ should revert if reward type not Native token (606ms)
      ✓ if it was unstaked, return reward for staked period (934ms)
      ✓ should calculate reward correctly (2076ms)
      ✓ if it was unstaked return staked period (602ms)
      ✓ should calculate timeDiff correctly (309ms)
    checkStakeCliqReward
      ✓ should revert if reward type not CLIQ token (425ms)
      ✓ if it was unstaked return reward for staked period (413ms)
      ✓ should calculate reward correctly (1069ms)
      ✓ if it was unstaked, return staked period (348ms)
```

```
      ✓ should calculate timeDiff correctly (379ms)
    unstake
      ✓ should revert if stake not defined (674ms)
      ✓ should revert if stake already withdrawn (1006ms)
      ✓ should decrease total balance (1842ms)
      ✓ should decrease user total staked balance (2047ms)
      ✓ should close the staking package(set _withdrawnTimestamp) (2003ms)
      reward type Native token
        ✓ should revert if not enough liquidity (350ms)
        ✓ should revert if try to unstake sooner than the blocked time (399ms)
        ✓ should decrease reward pool (581ms)
        ✓ should transfer staked amount + reward (612ms)
        ✓ should catch Unstaked event (389ms)
      reward type CLIQ token
        ✓ should revert if not enough liquidity (341ms)
        ✓ should revert if try to unstake sooner than the blocked time (598ms)
        ✓ should decrease reward pool (465ms)
        ✓ should transfer staked amount + reward (581ms)
        ✓ should catch Unstaked event (286ms)
    forceWithdraw
      1) should revert if stake not defined

Events emitted during test:
---------------------------

IERC20.Transfer(
  from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
  to: <indexed> 0xEF8DBe8eEfcce1b47944708592e6e70e8C9C21C8 (type: address),
  value: 100000000000000000000000 (type: uint256)
)

IERC20.Transfer(
  from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
  to: <indexed> 0xEF8DBe8eEfcce1b47944708592e6e70e8C9C21C8 (type: address),
  value: 100000000000000000000000 (type: uint256)
)

AccessControl.RoleGranted(
```

```
    role: <indexed> 0x0000000000000000000000000000000000000000000000000000000000000000
(type: bytes32),
    account: <indexed> 0xEF8DBe8eEfcce1b47944708592e6e70e8C9C21C8 (type: address),
    sender: <indexed> 0xEF8DBe8eEfcce1b47944708592e6e70e8C9C21C8 (type: address)
  )

  IERC20.Transfer(
    from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    to: <indexed> 0xE087EF304ea874B72A22DfB793ef4f6c5fb92ec9 (type: address),
    value: 200000000000000000000000 (type: uint256)
  )

  IERC20.Transfer(
    from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    to: <indexed> 0x1863C090EdAdf762594e91ac1Fc484282c8A8802 (type: address),
    value: 200000000000000000000000 (type: uint256)
  )

  IERC20.Transfer(
    from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    to: <indexed> 0x5B199f40E7a909155968FB7f3739E23D74F6168b (type: address),
    value: 200000000000000000000000 (type: uint256)
  )

  IERC20.Transfer(
    from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    to: <indexed> 0x0211dA3915F50cEc8F77b73B4992320ea5453b52 (type: address),
    value: 200000000000000000000000 (type: uint256)
  )

  AccessControl.RoleGranted(
    role: <indexed> 0xe56e19a7d558c827e5fe3dd17249e3d1e70e83b18ea8e11167d41e0a83cdd168
(type: bytes32),
    account: <indexed> 0x0211dA3915F50cEc8F77b73B4992320ea5453b52 (type: address),
    sender: <indexed> 0xEF8DBe8eEfcce1b47944708592e6e70e8C9C21C8 (type: address)
  )

  IERC20.Approval(
```

```
  owner: <indexed> 0xE087EF304ea874B72A22DfB793ef4f6c5fb92ec9 (type: address),
  spender: <indexed> 0x09A729FC675e52eB631f16071c0Ce94cB8fFDbbe (type: address),
  value: 2000000000000000000000 (type: uint256)
)

IERC20.Approval(
  owner: <indexed> 0x1863C090EdAdf762594e91ac1Fc484282c8A8802 (type: address),
  spender: <indexed> 0x09A729FC675e52eB631f16071c0Ce94cB8fFDbbe (type: address),
  value: 2000000000000000000000 (type: uint256)
)

IERC20.Approval(
  owner: <indexed> 0x0211dA3915F50cEc8F77b73B4992320ea5453b52 (type: address),
  spender: <indexed> 0x09A729FC675e52eB631f16071c0Ce94cB8fFDbbe (type: address),
  value: 20000000000000000000000000 (type: uint256)
)

IERC20.Transfer(
  from: <indexed> 0x0211dA3915F50cEc8F77b73B4992320ea5453b52 (type: address),
  to: <indexed> 0x09A729FC675e52eB631f16071c0Ce94cB8fFDbbe (type: address),
  value: 100000000000000000000 (type: uint256)
)

IERC20.Approval(
  owner: <indexed> 0x0211dA3915F50cEc8F77b73B4992320ea5453b52 (type: address),
  spender: <indexed> 0x09A729FC675e52eB631f16071c0Ce94cB8fFDbbe (type: address),
  value: 199990000000000000000000 (type: uint256)
)

CliqStaking.NativeTokenRewardAdded(
  _from: <indexed> 0x0211dA3915F50cEc8F77b73B4992320ea5453b52 (type: address),
  _val: 100000000000000000000 (type: uint256)
)

IERC20.Transfer(
  from: <indexed> 0xEF8DBe8eEfcce1b47944708592e6e70e8C9C21C8 (type: address),
  to: <indexed> 0x09A729FC675e52eB631f16071c0Ce94cB8fFDbbe (type: address),
  value: 200000000000000000000 (type: uint256)
```

```
)

IERC20.Transfer(
  from: <indexed> 0xE087EF304ea874B72A22DfB793ef4f6c5fb92ec9 (type: address),
  to: <indexed> 0x09A729FC675e52eB631f16071c0Ce94cB8fFDbbe (type: address),
  value: 50000000000000000000 (type: uint256)
)

IERC20.Approval(
  owner: <indexed> 0xE087EF304ea874B72A22DfB793ef4f6c5fb92ec9 (type: address),
  spender: <indexed> 0x09A729FC675e52eB631f16071c0Ce94cB8fFDbbe (type: address),
  value: 199500000000000000000000 (type: uint256)
)

CliqStaking.StakeAdded(
  _usr: <indexed> 0xE087EF304ea874B72A22DfB793ef4f6c5fb92ec9 (type: address),
  _packageName: 0x53696c766572205061636b6167650000000000000000000000000000000000000000
(type: bytes32),
  _amount: 50000000000000000000 (type: uint256),
  _stakeRewardType: 0 (type: uint16),
  _stakeIndex: 0 (type: uint256)
)

IERC20.Transfer(
  from: <indexed> 0x1863C090EdAdf762594e91ac1Fc484282c8A8802 (type: address),
  to: <indexed> 0x09A729FC675e52eB631f16071c0Ce94cB8fFDbbe (type: address),
  value: 50000000000000000000 (type: uint256)
)

IERC20.Approval(
  owner: <indexed> 0x1863C090EdAdf762594e91ac1Fc484282c8A8802 (type: address),
  spender: <indexed> 0x09A729FC675e52eB631f16071c0Ce94cB8fFDbbe (type: address),
  value: 199500000000000000000000 (type: uint256)
)

CliqStaking.StakeAdded(
  _usr: <indexed> 0x1863C090EdAdf762594e91ac1Fc484282c8A8802 (type: address),
  _packageName:
```

```
0x53696c766572205061636b6167650000000000000000000000000000000000000000 (type: bytes32),
    _amount: 50000000000000000000 (type: uint256),
    _stakeRewardType: 1 (type: uint16),
    _stakeIndex: 0 (type: uint256)
  )

  ---------------------------

      ✓ should revert if stake already withdrawn (860ms)
      ✓ should close the staking package(set _withdrawnTimestamp) (2004ms)
      ✓ should decrease total balance (1126ms)
      ✓ should decrease user total staked balance (1582ms)
      ✓ should revert if try to forceWithdraw sooner than the blocked time (375ms)
      ✓ should transfer staked amount (292ms)
      ✓ should catch ForcefullyWithdrawn event (265ms)
    pauseStaking
      ✓ can call only owner (223ms)
      ✓ should pause staking (227ms)
      ✓ should cath Paused event (61ms)
    unpauseStaking
      ✓ can call only owner (227ms)
      ✓ should pause staking (537ms)
      ✓ should cath Unpaused event (172ms)
    addStakedTokenReward
      ✓ only Reward provider can call (232ms)
      ✓ should transfer tokens (258ms)
      ✓ should catch NativeTokenRewardAdded event (67ms)
    removeStakedTokenReward
      ✓ only Reward provider can call (236ms)
      ✓ should revert if reward pool > removing amount (295ms)
      ✓ should transfer tokens (227ms)
      ✓ should catch NativeTokenRewardRemoved event (78ms)

79 passing (3m)
1 failing
```

```
1) Contract: CliqStaking
     Functions
       forceWithdraw
         should revert if stake not defined:

     Wrong kind of exception received
     + expected - actual

     -invalid opcode
     +The stake you are searching for is not defined

     at expectException (node_modules/@openzeppelin/test-helpers/src/expectRevert.js:20:30)
     at process._tickCallback (internal/process/next_tick.js:68:7)
```

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	UNCOVERED LINES
contracts/	99.05	95.65	100.00	99.07	
cliqStaking.sol	99.05	95.65	100.00	99.07	292
contracts/mock/	75.00	100.00	75.00	75.00	
ERC20Mock.sol	66.67	100.00	66.67	66.67	21
cliq.sol	100.00	100.00	100.00	100.00	
All files	**98.17**	**95.65**	**94.74**	**98.02**	

```
> Istanbul reports written to ./coverage/ and ./coverage.json
> solidity-coverage cleaning up, shutting down ganache server
Error: ✗ 1 test(s) failed under coverage.
    at plugin
(/home/romario/.nvm/versions/node/v10.22.0/lib/node_modules/solidity-coverage/plugins/truffle.plugin.
js:121:27)
    at process._tickCallback (internal/process/next_tick.js:68:7)
Truffle v5.1.59 (core: 5.1.59)
Node v10.22.0
```

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

ZOKYO.