DYP STAKING AUDIT

January 2021

BLOCKCHAIN CONSILIUM

Contents

Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED "AS IS", WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND, AND BLOCKCHAIN CONSILIUM DISCLAIMS ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT. COPYRIGHT OF THIS REPORT REMAINS WITH BLOCKCHAIN CONSILIUM.

Purpose of the report

The Audits and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of Solidity code and only the Solidity code we note as being within the scope of our review within this report. The Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the Solidity programming language that could present security risks. Cryptographic tokens and smart contracts are emergent technologies and carry with them high levels of technical risk and uncertainty.

The Audits are not an endorsement or indictment of any particular project or team, and the Audits do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Audits in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. This Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. There is no owed duty to any Third-Party by virtue of publishing these Audits.

Introduction

We first thank dyp.finance for giving us the opportunity to audit their smart contract. This document outlines our methodology, audit details, and results.

dyp.finance asked us to review their DYP staking smart contract (GitHub Commit Hash: `7a52225fa5beb2ef4c3258416c1756864e92940d`). Blockchain Consilium reviewed the system from a technical perspective looking for bugs, issues and vulnerabilities in their code base. The Audit is valid for constant-return-staking.sol at `7a52225fa5beb2ef4c3258416c1756864e92940d` GitHub commit hash only. The audit is not valid for any other versions of the smart contract. Read more below.

Audit Summary

This code is clean, thoughtfully written and in general well architected. The code conforms closely to the documentation and specification.

Overall, the code is clear on what it is supposed to do for each function. The visibility and state mutability of all the functions are clearly specified, and there are no confusions.

https://github.com/dypfinance/DYP-staking-governance-dapp/blob/7a52225fa5beb2ef4c3258416c1756864e92940d/constant-return-staking.sol

Audit Result	**PASSED AS PER SPECIFICATIONS, 1 INFORMATIONAL OBSERVATION**
High Severity Issues	None
Moderate Severity Issues	None
Low Severity Issues	None
Informational Observations	*1 informational observation*

Overview

The project has one Solidity file for the DYP Staking Smart Contract, the constant-return-staking.sol file that contains about 794 lines of Solidity code. We manually reviewed each line of code in the smart contract.

Methodology:

Blockchain Consilium manually reviewed the smart contract line-by-line, keeping in mind industry best practices and known attacks, looking for any potential issues and vulnerabilities, and areas where improvements are possible.

We also used automated tools like slither for analysis and reviewing the smart contract. The raw output of these tools is included in the Appendix. These tools often give false-positives, and any issues reported by them but not included in the issue list can be considered not valid.

Classification / Issue Types Definition:

1. **High Severity:** which presents a significant security vulnerability or failure of the contract across a range of scenarios, or which may result in loss of funds.
2. **Moderate Severity:** which affects the desired outcome of the contract execution or introduces a weakness that can be exploited. It may not result in loss of funds but breaks the functionality or produces unexpected behaviour.
3. **Low Severity:** which does not have a material impact on the contract execution and is likely to be subjective.

The smart contract is considered to pass the audit, as of the audit date, if no high severity or moderate severity issues are found.

Attacks & Issues considered while auditing

In order to check for the security of the contract, we reviewed each line of code in the smart contract considering several known Smart Contract Attacks & known issues.

- **Overflows and underflows:**
  An overflow happens when the limit of the type variable uint256 , 2 ** 256, is exceeded. What happens is that the value resets to zero instead of incrementing more.

  For instance, if we want to assign a value to a uint bigger than 2 ** 256 it will simple go to 0—this is dangerous.

  On the other hand, an underflow happens when you try to subtract 0 minus a number bigger than 0.For example, if you subtract 0 - 1 the result will be = 2 ** 256 instead of -1.

This is quite dangerous. This contract **DOES** check for overflows and underflows, using **OpenZeppelin's** *SafeMath* for overflow and underflow protection.

- **Reentrancy Attack:**
  One of the major dangers of calling external contracts is that they can take over the control flow, and make changes to your data that the calling function wasn't expecting. This class of bug can take many forms, and both of the major bugs that led to the DAO's collapse were bugs of this sort.

  This smart contract does make state changes after external calls, however the tokenContract and external calls are trusted and thus *is not found vulnerable* to re-entrancy attack.

- **Replay attack:**
  The replay attack consists of making a transaction on one blockchain like the original Ethereum's blockchain and then repeating it on another blockchain like the Ethereum's classic blockchain. The ether is transferred like a normal transaction from a blockchain to another. Though it's no longer a problem because since the version 1.5.3 of *Geth* and 1.4.4 of *Parity* both implement the attack protection EIP 155 by Vitalik Buterin.
  So the people that will use the contract depend on their own ability to be updated with those programs to keep themselves secure.

- **Short address attack:**
  This attack affects ERC20 tokens, was discovered by the Golem team and consists of the following:

  A user creates an Ethereum wallet with a trailing 0, which is not hard because it's only a digit. For instance: `0xiofa8d97756as7df5sd8f75g8675ds8gsdg0`
  Then he buys tokens by removing the last zero:
  Buy 1000 tokens from account `0xiofa8d97756as7df5sd8f75g8675ds8gsdg`. If the contract has enough amount of tokens and the buy function doesn't check the length of the address of the sender, the Ethereum's virtual machine will just add zeroes to the transaction until the address is complete.

  The virtual machine will return 256000 for each 1000 tokens bought. This is abug of the virtual machine.

  Here is a **fix for short address attacks**

```
modifier onlyPayloadSize(uint size) {
    assert(msg.data.length >= size + 4);
    _;
}
function transfer(address _to, uint256 _value) onlyPayloadSize(2 * 32) {
    // do stuff
}
```

Whether or not it is appropriate for token contracts to mitigate the short-address attack is a contentious issue among smart-contract developers. Many, including those behind the OpenZeppelin project, have explicitly chosen not to do so. Blockchain Consilium doesn't consider short address attack an issue of the smart contract at the smart contract level.

This contract is not an ERC20 Token thus it is not found vulnerable to short address attacks.

You can read more about the attack here: ERC20 Short Address Attacks.

- **Approval Double-spend**

  ERC20 Standard allows users to approve other users to manage their tokens, or spend tokens from their account till a certain amount, by setting the user's allowance with the standard `approve` function, then the allowed user may use `transferFrom` to spend the allowed tokens.

  Hypothetically, given a situation where Alice approves Bob to spend 100 Tokens from her account, and if Alice needs to adjust the allowance to allow Bob to spend 20 more tokens, normally – she'd check Bob's allowance (100 currently) and start a new `approve` transaction allowing Bob to spend a total of 120 Tokens instead of 100 Tokens.

  Now, if Bob is monitoring the Transaction pool, and as soon as he observes new transaction from Alice approving more amount, he may send a `transferFrom` transaction spending 100 Tokens from Alice's account with higher gas price and do all the required effort to get his spend transaction mined before Alice's new approve transaction.

  Now Bob has already spent 100 Tokens, and given Alice's approve transaction is mined, Bob's allowance is set to 120 Tokens, this would allow Bob to spend a total of 100 + 120 = 220 Tokens from Alice's account instead of the allowed 120 Tokens. This exploit situation is known as Approval Double-Spend Attack.

  A potential solution to minimize these instances would be to set the non-zero allowance to 0 before setting it to any other amount.

  It's possible for `approve` to enforce this behaviour without interface changes in the ERC20 specification:

  ```
  if ((_value != 0) && (approved[msg.sender][_spender] != 0)) return false;
  ```

However, this is just an attempt to modify user behaviour. If the user does attempt to change from one non-zero value to another, the double spend might still happen, since the attacker may set the value to zero by already spending all the previously allowed value before the user's new approval transaction.

If desired, a non-standard function can be added to minimize hassle for users. The issue can be fixed with minimal inconvenience by taking a change value rather than a replacement value:

```
function increaseAllowance (address _spender, uint256 _addedValue)
returns (bool success) {
  uint oldValue = approved[msg.sender][_spender];
  approved[msg.sender][_spender] = safeAdd(oldValue, _addedValue);
  return true;
}
```

Even if this function is added, it's important to keep the original for compatibility with the ERC20 specification.

Likely impact of this bug is low for most situations. This contract is not an ERC20 Token*, thus it is not found vulnerable to approval double-spend attack.*

For more, see this discussion on GitHub:
https://github.com/ethereum/EIPs/issues/20#issuecomment263524729

- **Accidental Token Loss**
  - When other ERC20 Tokens are transferred to the DYP staking smart contract, there would be no way to take them out, and this has been solved by implementing the "Any Token Transfer" function to allow owner to transfer out any ERC20 compliant token other than DYP token.

    Admin cannot transfer out DYP from this smart contract before the staking duration is over (supposed to be around 365 days approx. with a 30 days buffer before admin can claim).

Issues Found & Informational Observations

High Severity Issues

No high severity issues were found in the smart contract.

Moderate Severity Issues

No moderate severity issues were found in the smart contract.

Low Severity Issues

No low severity issues were found in the smart contract.

Informational Observations

DYP Team is to supply DYP rewards to the staking smart contract. As long as enough reward tokens are available in the smart contract this staking is supposed to work fine as per specifications, however when the staking rewards are running out and any user's pending earnings become more than contract's total DYP balance, the user will be unable to claim their earnings because the contract will have run out of reward tokens after a particular period of time.

It is recommended that the community must be informed about a set time limit when the rewards are supposed to run out and they must claim their pending earnings or unstake before the rewards have run out from the staking smart contract.

There's an `emergencyUnstake` function which would unstake user's deposit without claiming any earnings, setting pending earnings to 0, this can be directly accessed from smart contract and may be useful in such a situation if any.

Line by line comments

- Line 2:
  The compiler version is specified as 0.6.11, this means the code can be compiled with solidity compilers with 0.6.11 only, the latest compiler version at the time of auditing is 0.8.0.

- Lines 4 to 32:
  SafeMath library is included to check for underflow and overflows.

- Lines 34 to 218:
  Address library is included, used in the smart contract to help check whether a given address is a deployed smart contract or not.

- Lines 220 to 458:
  EnumerableSet library is included to implement address sets in the smart contract for keeping track of stakers list.

- Lines 460 to 499:
  Ownable contract is implemented to provide basic access control for transferring out other tokens except DYP from this smart contract

- Lines 501 to 508:
  Token interface & Legacy Token Interface is included to interact with ERC20 tokens and Legacy ERC20 Tokens.

- Lines 510 to 794:
  ConstantReturnStaking contract is implemented inheriting from Ownable contract. This contract defines the staking fee rate and unstaking fee rate, lockup period and various contract variables. It uses EnumerableSet and SafeMath for keeping track of stakers and implementing protection for underflow overflow.

  It implements stake, unstake, reinvest, and claim divs functions and one level referral system transferring a set percent (5%) rewards to referrer if any. An `emergencyUnstake` function to be used in a situation where the user would want to withdraw their deposit without withdrawing their pending rewards Staking and unstaking automatically claims pending earnings. The staking APY set in the smart contract is supposed to work fine till the staking rewards are available in ample amounts.

  After a period of time, if any user's pending earnings become more than the contract's total token balance, the user will be unable to claim their pending earnings, this makes the contract high risk and thus the community should be asked to unstake after a set period of time ends by when the staking rewards are calculated to run out.

Appendix

Smart Contract Summary

- Contract SafeMath (Most derived contract)
    - From SafeMath
        - add(uint256,uint256) (internal)
        - div(uint256,uint256) (internal)
        - mul(uint256,uint256) (internal)
        - sub(uint256,uint256) (internal)
- Contract Address (Most derived contract)
    - From Address
        - _verifyCallResult(bool,bytes,string) (private)
        - functionCall(address,bytes) (internal)
        - functionCall(address,bytes,string) (internal)
        - functionCallWithValue(address,bytes,uint256) (internal)
        - functionCallWithValue(address,bytes,uint256,string) (internal)
        - functionDelegateCall(address,bytes) (internal)
        - functionDelegateCall(address,bytes,string) (internal)
        - functionStaticCall(address,bytes) (internal)
        - functionStaticCall(address,bytes,string) (internal)
        - isContract(address) (internal)
        - sendValue(address,uint256) (internal)
- Contract EnumerableSet (Most derived contract)
    - From EnumerableSet
        - _add(EnumerableSet.Set,bytes32) (private)
        - _at(EnumerableSet.Set,uint256) (private)
        - _contains(EnumerableSet.Set,bytes32) (private)
        - _length(EnumerableSet.Set) (private)
        - _remove(EnumerableSet.Set,bytes32) (private)
        - add(EnumerableSet.AddressSet,address) (internal)
        - add(EnumerableSet.UintSet,uint256) (internal)
        - at(EnumerableSet.AddressSet,uint256) (internal)
        - at(EnumerableSet.UintSet,uint256) (internal)

        - contains(EnumerableSet.AddressSet,address) (internal)
        - contains(EnumerableSet.UintSet,uint256) (internal)
        - length(EnumerableSet.AddressSet) (internal)
        - length(EnumerableSet.UintSet) (internal)
        - remove(EnumerableSet.AddressSet,address) (internal)
        - remove(EnumerableSet.UintSet,uint256) (internal)

- Contract Ownable
    - From Ownable
        - constructor() (public)
        - transferOwnership(address) (public)
- Contract Token (Most derived contract)
    - From Token
        - transfer(address,uint256) (external)
        - transferFrom(address,address,uint256) (external)
- Contract LegacyToken (Most derived contract)
    - From LegacyToken
        - transfer(address,uint256) (external)
- Contract ConstantReturnStaking (Most derived contract)
    - From Ownable
        - transferOwnership(address) (public)
    - From ConstantReturnStaking
        - claim() (external)
        - constructor() (public)
        - emergencyUnstake(uint256) (external)
        - getActiveReferredStaker(address,uint256) (external)
        - getNumberOfHolders() (external)
        - getNumberOfReferredStakers(address) (external)
        - getPendingDivs(address) (public)
        - getReferredStaker(address,uint256) (external)
        - getStakersList(uint256,uint256) (public)
        - getTotalPendingDivs(address) (external)
        - reInvest() (external)
        - stake(uint256,address) (external)

- transferAnyERC20Token(address,address,uint256) (external)
- transferAnyLegacyERC20Token(address,address,uint256) (external)
- transferReferralFeeIfPossible(address,uint256) (private)
- unstake(uint256) (external)
- updateAccount(address) (private)

Slither Results

```
> slither constant-return-staking.sol

INFO:Detectors:
LegacyToken (constant-return-staking.sol#506-508) has incorrect ERC20 function
interface:LegacyToken.transfer(address,uint256) (constant-return-staking.sol#507)
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#incorrect-erc20-interface
INFO:Detectors:
ConstantReturnStaking.emergencyUnstake(uint256) (constant-return-staking.sol#708-
727) uses a dangerous strict equality:
        - holders.contains(msg.sender) && depositedTokens[msg.sender] == 0
(constant-return-staking.sol#724)
ConstantReturnStaking.getPendingDivs(address) (constant-return-staking.sol#608-
634) uses a dangerous strict equality:
        - depositedTokens[_holder] == 0 (constant-return-staking.sol#610)
ConstantReturnStaking.unstake(uint256) (constant-return-staking.sol#685-704) uses
a dangerous strict equality:
        - holders.contains(msg.sender) && depositedTokens[msg.sender] == 0
(constant-return-staking.sol#700)
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in ConstantReturnStaking.claim() (constant-return-staking.sol#729-737):
        External calls:
        - updateAccount(msg.sender) (constant-return-staking.sol#730)
                -
require(bool,string)(Token(TRUSTED_TOKEN_ADDRESS).transfer(account,amount),Could
not transfer referral fee!) (constant-return-staking.sol#600)
        State variables written after the call(s):
        - rewardsPendingClaim[msg.sender] = 0 (constant-return-staking.sol#733)
Reentrancy in ConstantReturnStaking.emergencyUnstake(uint256) (constant-return-
staking.sol#708-727):
        External calls:
        -
require(bool,string)(Token(TRUSTED_TOKEN_ADDRESS).transfer(owner,fee),Could not
transfer withdraw fee.) (constant-return-staking.sol#719)
        -
require(bool,string)(Token(TRUSTED_TOKEN_ADDRESS).transfer(msg.sender,amountAfterF
ee),Could not transfer tokens.) (constant-return-staking.sol#720)
        State variables written after the call(s):
        - depositedTokens[msg.sender] =
depositedTokens[msg.sender].sub(amountToWithdraw) (constant-return-
staking.sol#722)
Reentrancy in ConstantReturnStaking.reInvest() (constant-return-staking.sol#739-
751):
        External calls:
        - updateAccount(msg.sender) (constant-return-staking.sol#740)
                -
require(bool,string)(Token(TRUSTED_TOKEN_ADDRESS).transfer(account,amount),Could
not transfer referral fee!) (constant-return-staking.sol#600)
        State variables written after the call(s):
        - depositedTokens[msg.sender] = depositedTokens[msg.sender].add(amount)
(constant-return-staking.sol#746)
        - rewardsPendingClaim[msg.sender] = 0 (constant-return-staking.sol#743)
Reentrancy in ConstantReturnStaking.stake(uint256,address) (constant-return-
staking.sol#661-683):
        External calls:
        -
require(bool,string)(Token(TRUSTED_TOKEN_ADDRESS).transferFrom(msg.sender,address(
this),amountToStake),Insufficient Token Allowance) (constant-return-
staking.sol#663)
        - updateAccount(msg.sender) (constant-return-staking.sol#665)
                -
require(bool,string)(Token(TRUSTED_TOKEN_ADDRESS).transfer(account,amount),Could
not transfer referral fee!) (constant-return-staking.sol#600)
        -
require(bool,string)(Token(TRUSTED_TOKEN_ADDRESS).transfer(owner,fee),Could not
transfer deposit fee.) (constant-return-staking.sol#669)
        State variables written after the call(s):
        - depositedTokens[msg.sender] =
depositedTokens[msg.sender].add(amountAfterFee) (constant-return-staking.sol#671)
        - referrals[msg.sender] = referrer (constant-return-staking.sol#676)
Reentrancy in ConstantReturnStaking.unstake(uint256) (constant-return-
staking.sol#685-704):
        External calls:
        - updateAccount(msg.sender) (constant-return-staking.sol#690)
                -
require(bool,string)(Token(TRUSTED_TOKEN_ADDRESS).transfer(account,amount),Could
not transfer referral fee!) (constant-return-staking.sol#600)
        -
require(bool,string)(Token(TRUSTED_TOKEN_ADDRESS).transfer(owner,fee),Could not
transfer withdraw fee.) (constant-return-staking.sol#695)
        -
require(bool,string)(Token(TRUSTED_TOKEN_ADDRESS).transfer(msg.sender,amountAfterF
ee),Could not transfer tokens.) (constant-return-staking.sol#696)
        State variables written after the call(s):
        - depositedTokens[msg.sender] =
depositedTokens[msg.sender].sub(amountToWithdraw) (constant-return-
staking.sol#698)
Reentrancy in ConstantReturnStaking.updateAccount(address) (constant-return-
staking.sol#574-595):
        External calls:
        - success = transferReferralFeeIfPossible(referrals[account],referralFee)
(constant-return-staking.sol#580)
                -
require(bool,string)(Token(TRUSTED_TOKEN_ADDRESS).transfer(account,amount),Could
not transfer referral fee!) (constant-return-staking.sol#600)
        State variables written after the call(s):
        - lastClaimedTime[account] = now (constant-return-staking.sol#594)
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
ConstantReturnStaking.stake(uint256,address) (constant-return-staking.sol#661-683)
ignores return value by holders.add(msg.sender) (constant-return-staking.sol#673)
```

```
ConstantReturnStaking.stake(uint256,address) (constant-return-staking.sol#661-683)
ignores return value by
totalReferredAddressesOfUser[referrals[msg.sender]].add(msg.sender) (constant-
return-staking.sol#679)
ConstantReturnStaking.stake(uint256,address) (constant-return-staking.sol#661-683)
ignores return value by
activeReferredAddressesOfUser[referrals[msg.sender]].add(msg.sender) (constant-
return-staking.sol#680)
ConstantReturnStaking.unstake(uint256) (constant-return-staking.sol#685-704)
ignores return value by holders.remove(msg.sender) (constant-return-
staking.sol#701)
ConstantReturnStaking.unstake(uint256) (constant-return-staking.sol#685-704)
ignores return value by
activeReferredAddressesOfUser[referrals[msg.sender]].remove(msg.sender) (constant-
return-staking.sol#702)
ConstantReturnStaking.emergencyUnstake(uint256) (constant-return-staking.sol#708-
727) ignores return value by holders.remove(msg.sender) (constant-return-
staking.sol#725)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-
return
INFO:Detectors:
Reentrancy in ConstantReturnStaking.reInvest() (constant-return-staking.sol#739-
751):
        External calls:
        - updateAccount(msg.sender) (constant-return-staking.sol#740)
                -
require(bool,string)(Token(TRUSTED_TOKEN_ADDRESS).transfer(account,amount),Could
not transfer referral fee!) (constant-return-staking.sol#600)
        State variables written after the call(s):
        - stakingTime[msg.sender] = now (constant-return-staking.sol#748)
Reentrancy in ConstantReturnStaking.stake(uint256,address) (constant-return-
staking.sol#661-683):
        External calls:
        -
require(bool,string)(Token(TRUSTED_TOKEN_ADDRESS).transferFrom(msg.sender,address(
this),amountToStake),Insufficient Token Allowance) (constant-return-
staking.sol#663)
        - updateAccount(msg.sender) (constant-return-staking.sol#665)
                -
require(bool,string)(Token(TRUSTED_TOKEN_ADDRESS).transfer(account,amount),Could
not transfer referral fee!) (constant-return-staking.sol#600)
        -
require(bool,string)(Token(TRUSTED_TOKEN_ADDRESS).transfer(owner,fee),Could not
transfer deposit fee.) (constant-return-staking.sol#669)
        State variables written after the call(s):
        - stakingTime[msg.sender] = now (constant-return-staking.sol#682)
Reentrancy in ConstantReturnStaking.transferReferralFeeIfPossible(address,uint256)
(constant-return-staking.sol#597-606):
        External calls:
        -
require(bool,string)(Token(TRUSTED_TOKEN_ADDRESS).transfer(account,amount),Could
not transfer referral fee!) (constant-return-staking.sol#600)
        State variables written after the call(s):
        - totalClaimedReferralFee = totalClaimedReferralFee.add(amount)
(constant-return-staking.sol#601)
Reentrancy in ConstantReturnStaking.updateAccount(address) (constant-return-
staking.sol#574-595):
        External calls:
```

```
        - success = transferReferralFeeIfPossible(referrals[account],referralFee)
(constant-return-staking.sol#580)
                -
require(bool,string)(Token(TRUSTED_TOKEN_ADDRESS).transfer(account,amount),Could
not transfer referral fee!) (constant-return-staking.sol#600)
        State variables written after the call(s):
        - rewardsPendingClaim[account] = rewardsPendingClaim[account].add(amount)
(constant-return-staking.sol#588)
        - totalClaimedRewards = totalClaimedRewards.add(amount) (constant-return-
staking.sol#591)
        - totalEarnedTokens[account] = totalEarnedTokens[account].add(amount)
(constant-return-staking.sol#589)
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in ConstantReturnStaking.claim() (constant-return-staking.sol#729-737):
        External calls:
        - updateAccount(msg.sender) (constant-return-staking.sol#730)
                -
require(bool,string)(Token(TRUSTED_TOKEN_ADDRESS).transfer(account,amount),Could
not transfer referral fee!) (constant-return-staking.sol#600)
        -
require(bool,string)(Token(TRUSTED_TOKEN_ADDRESS).transfer(msg.sender,amount),Coul
d not transfer earned tokens.) (constant-return-staking.sol#734)
        Event emitted after the call(s):
        - RewardsTransferred(msg.sender,amount) (constant-return-staking.sol#735)
Reentrancy in ConstantReturnStaking.reInvest() (constant-return-staking.sol#739-
751):
        External calls:
        - updateAccount(msg.sender) (constant-return-staking.sol#740)
                -
require(bool,string)(Token(TRUSTED_TOKEN_ADDRESS).transfer(account,amount),Could
not transfer referral fee!) (constant-return-staking.sol#600)
        Event emitted after the call(s):
        - Reinvest(msg.sender,amount) (constant-return-staking.sol#749)
Reentrancy in ConstantReturnStaking.stake(uint256,address) (constant-return-
staking.sol#661-683):
        External calls:
        -
require(bool,string)(Token(TRUSTED_TOKEN_ADDRESS).transferFrom(msg.sender,address(
this),amountToStake),Insufficient Token Allowance) (constant-return-
staking.sol#663)
        - updateAccount(msg.sender) (constant-return-staking.sol#665)
                -
require(bool,string)(Token(TRUSTED_TOKEN_ADDRESS).transfer(account,amount),Could
not transfer referral fee!) (constant-return-staking.sol#600)
        Event emitted after the call(s):
        - ReferralFeeTransferred(account,amount) (constant-return-
staking.sol#602)
                - updateAccount(msg.sender) (constant-return-staking.sol#665)
Reentrancy in ConstantReturnStaking.transferReferralFeeIfPossible(address,uint256)
(constant-return-staking.sol#597-606):
        External calls:
        -
require(bool,string)(Token(TRUSTED_TOKEN_ADDRESS).transfer(account,amount),Could
not transfer referral fee!) (constant-return-staking.sol#600)
        Event emitted after the call(s):
        - ReferralFeeTransferred(account,amount) (constant-return-
staking.sol#602)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
ConstantReturnStaking.updateAccount(address) (constant-return-staking.sol#574-595)
uses timestamp for comparisons
        Dangerous comparisons:
        - pendingDivs > 0 (constant-return-staking.sol#576)
ConstantReturnStaking.transferReferralFeeIfPossible(address,uint256) (constant-
return-staking.sol#597-606) uses timestamp for comparisons
        Dangerous comparisons:
        - account != address(0) && amount > 0 (constant-return-staking.sol#598)
        -
require(bool,string)(Token(TRUSTED_TOKEN_ADDRESS).transfer(account,amount),Could
not transfer referral fee!) (constant-return-staking.sol#600)
ConstantReturnStaking.getPendingDivs(address) (constant-return-staking.sol#608-
634) uses timestamp for comparisons
        Dangerous comparisons:
        - depositedTokens[_holder] == 0 (constant-return-staking.sol#610)
        - _now > stakingEndTime (constant-return-staking.sol#615)
        - lastClaimedTime[_holder] >= _now (constant-return-staking.sol#619)
ConstantReturnStaking.unstake(uint256) (constant-return-staking.sol#685-704) uses
timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(depositedTokens[msg.sender] >=
amountToWithdraw,Invalid amount to withdraw) (constant-return-staking.sol#686)
        - require(bool,string)(now.sub(stakingTime[msg.sender]) > LOCKUP_TIME,You
recently staked, please wait before withdrawing.) (constant-return-
staking.sol#688)
        - holders.contains(msg.sender) && depositedTokens[msg.sender] == 0
(constant-return-staking.sol#700)
ConstantReturnStaking.emergencyUnstake(uint256) (constant-return-staking.sol#708-
727) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(depositedTokens[msg.sender] >=
amountToWithdraw,Invalid amount to withdraw) (constant-return-staking.sol#709)
        - require(bool,string)(now.sub(stakingTime[msg.sender]) > LOCKUP_TIME,You
recently staked, please wait before withdrawing.) (constant-return-
staking.sol#711)
        - holders.contains(msg.sender) && depositedTokens[msg.sender] == 0
(constant-return-staking.sol#724)
ConstantReturnStaking.claim() (constant-return-staking.sol#729-737) uses timestamp
for comparisons
        Dangerous comparisons:
        - amount > 0 (constant-return-staking.sol#732)
        -
require(bool,string)(Token(TRUSTED_TOKEN_ADDRESS).transfer(msg.sender,amount),Coul
d not transfer earned tokens.) (constant-return-staking.sol#734)
ConstantReturnStaking.reInvest() (constant-return-staking.sol#739-751) uses
timestamp for comparisons
        Dangerous comparisons:
        - amount > 0 (constant-return-staking.sol#742)
ConstantReturnStaking.transferAnyERC20Token(address,address,uint256) (constant-
return-staking.sol#783-786) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(tokenAddress != TRUSTED_TOKEN_ADDRESS || now >
contractStartTime.add(ADMIN_CAN_CLAIM_AFTER),Cannot Transfer Out main tokens!)
(constant-return-staking.sol#784)
ConstantReturnStaking.transferAnyLegacyERC20Token(address,address,uint256)
(constant-return-staking.sol#790-793) uses timestamp for comparisons
```

```
        Dangerous comparisons:
        - require(bool,string)(tokenAddress != TRUSTED_TOKEN_ADDRESS || now >
contractStartTime.add(ADMIN_CAN_CLAIM_AFTER),Cannot Transfer Out main tokens!)
(constant-return-staking.sol#791)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-
timestamp
INFO:Detectors:
Address.isContract(address) (constant-return-staking.sol#55-64) uses assembly
        - INLINE ASM (constant-return-staking.sol#62)
Address._verifyCallResult(bool,bytes,string) (constant-return-staking.sol#200-217)
uses assembly
        - INLINE ASM (constant-return-staking.sol#209-212)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-
usage
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (constant-return-
staking.sol#82-88):
        - (success) = recipient.call{value: amount}() (constant-return-
staking.sol#86)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string)
(constant-return-staking.sol#143-150):
        - (success,returndata) = target.call{value: value}(data) (constant-
return-staking.sol#148)
Low level call in Address.functionStaticCall(address,bytes,string) (constant-
return-staking.sol#168-174):
        - (success,returndata) = target.staticcall(data) (constant-return-
staking.sol#172)
Low level call in Address.functionDelegateCall(address,bytes,string) (constant-
return-staking.sol#192-198):
        - (success,returndata) = target.delegatecall(data) (constant-return-
staking.sol#196)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-
level-calls
INFO:Detectors:
Parameter ConstantReturnStaking.getPendingDivs(address)._holder (constant-return-
staking.sol#608) is not in mixedCase
Parameter ConstantReturnStaking.getTotalPendingDivs(address)._holder (constant-
return-staking.sol#636) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (constant-return-staking.sol#494-
498)
getStakersList(uint256,uint256) should be declared external:
        - ConstantReturnStaking.getStakersList(uint256,uint256) (constant-return-
staking.sol#753-778)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-
function-that-could-be-declared-external
INFO:Slither:constant-return-staking.sol analyzed (7 contracts with 46 detectors),
43 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and
Github integration
```