DYP FUNDRAISER SMART CONTRACT AUDIT

October 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 fundraiser smart contract (GitHub Commit Hash: `4435b148f9526fd130ff563afa4d5d64b87379fd`). Blockchain Consilium reviewed the system from a technical perspective looking for bugs, issues and vulnerabilities in their code base. The Audit is valid for fundraiser.sol at the mentioned 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/launchpad/blob/4435b148f9526fd130ff563afa4d5d64b87379fd/Contracts/fundraiser.sol

Audit Scope & Info	
Platform	Ethereum
Language	Solidity
Audit Method	Whitebox
Repository	https://github.com/dypfinance/launchpad/blob/4435b148f9526fd130ff563afa4d5d64b87379fd/Contracts/fundraiser.sol
SHA256	`> shasum -a 256 fundraiser.sol` `555dd793952d3ebbcb4f21621010c9c43e2c2feca5f299b14720628ab76ca38d`

Audit Results & Findings	
High Severity Issues	None
Moderate Severity Issues	None
Low Severity Issues	None
Informational Observations	2

Overview

The project has one Solidity file for the DYP Fundraiser Smart Contract, the fundraiser.sol file that contains about 1,023 lines of Solidity code. We manually reviewed each line of code in the smart contract. The smart contract integrates Uniswap (an external AMM DEX) and use imports from OpenZeppelin smart contract libraries, code review of OpenZeppelin libraries or external services is outside the scope of this audit.

The smart contract allows whitelisted users to participate in fundraisers on DYP launchpad using any of the allowed tokens at a pre-decided rate, the contributed tokens are forwarded to DYP admins, the whitelist check is performed by signature of verify address that comes from the backend – it is recommended to securely store and manage private keys of the verify address and change the verify address from time to time.

After purchase, there's an unlock time for the sold tokens, after unlock time, DYP admin must supply the required amount of tokens to the contract so that users can claim their purchased tokens. The DYP admin is always able to transfer any amount of tokens from this smart contract, this feature can be useful in emergency but can also be misused if a malicious actor gets control of this feature.

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 & surya 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 token contract 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, 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 (invalid address for discussion purposes only)`
  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 smart contract, normally there would be no way to take them out, and this has been solved by implementing the "claimAnyToken" function to allow owner to transfer out any ERC20 compliant tokens.

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

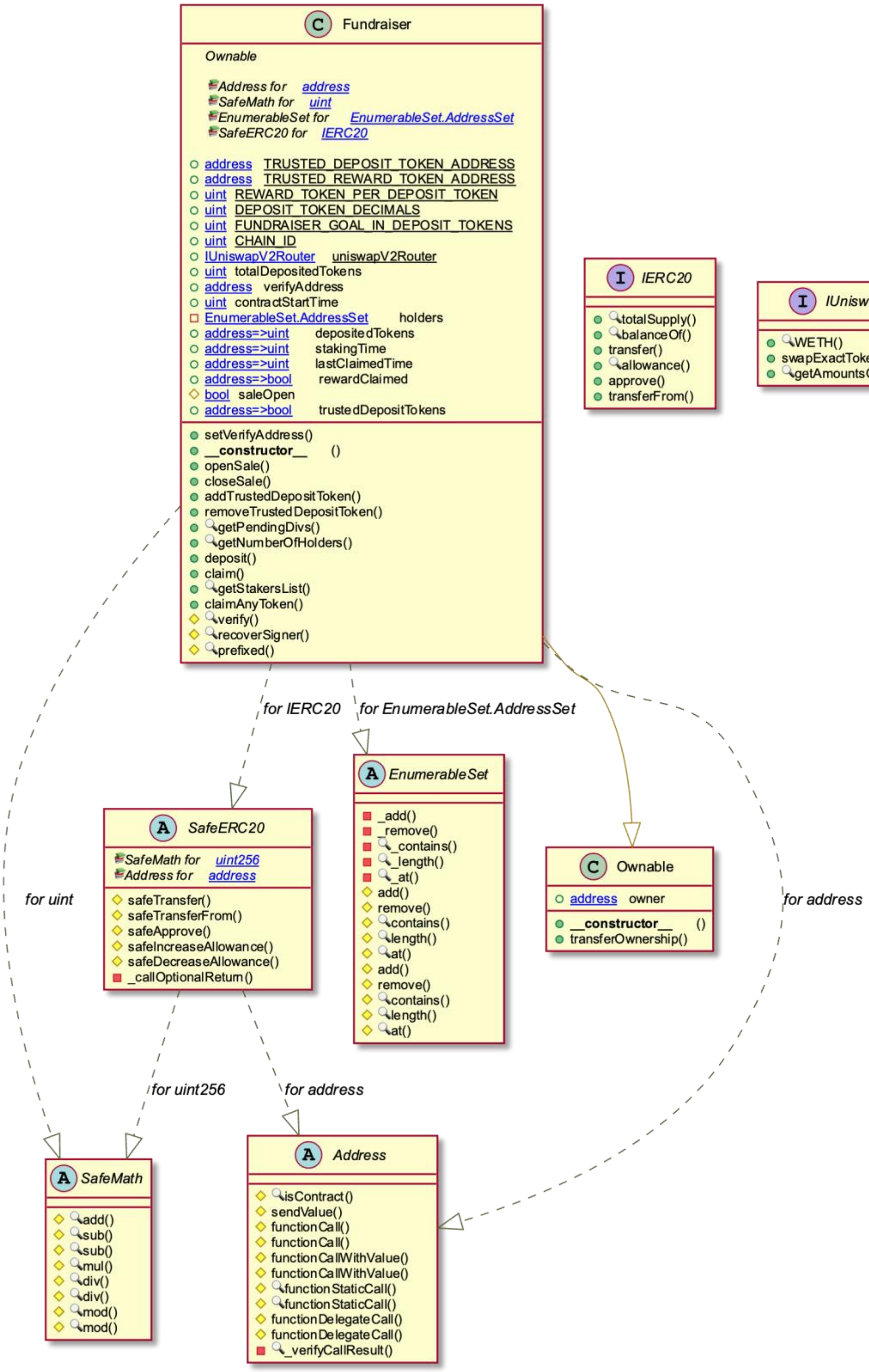
No low severity issues were found in the smart contract.

Informational Observations

- The smart contract integrates Uniswap, it might be vulnerable to sandwiching exploits or Uniswap Price manipulation, it is recommended to use decentralized oracles or Uniswap TWAPs to counter this issue. However the \`noContractsAllowed\` modifier and the external variable \`_amountOutMin\` mitigate this issue to some extent, but there's still need for research into effective ways for elimination of this issue.

- The smart contract allows admin to transfer any amount of tokens from the contract anytime, and also admin must supply enough tokens to the contract for the fundraiser, in this sense, the smart contract includes centralized features.

Inheritance Graph & UML Diagram



Appendix

Smart Contract Summary

- Contract SafeMath (Most derived contract)
    - From SafeMath
        - add(uint256,uint256) (internal)
        - div(uint256,uint256) (internal)
        - div(uint256,uint256,string) (internal)
        - mod(uint256,uint256) (internal)
        - mod(uint256,uint256,string) (internal)
        - mul(uint256,uint256) (internal)
        - sub(uint256,uint256) (internal)
        - sub(uint256,uint256,string) (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 IERC20 (Most derived contract)
    - From IERC20
        - allowance(address,address) (external)
        - approve(address,uint256) (external)
        - balanceOf(address) (external)
        - totalSupply() (external)
        - transfer(address,uint256) (external)
        - transferFrom(address,address,uint256) (external)

- Contract SafeERC20 (Most derived contract)
    - From SafeERC20
        - _callOptionalReturn(IERC20,bytes) (private)
        - safeApprove(IERC20,address,uint256) (internal)
        - safeDecreaseAllowance(IERC20,address,uint256) (internal)
        - safeIncreaseAllowance(IERC20,address,uint256) (internal)
        - safeTransfer(IERC20,address,uint256) (internal)
        - safeTransferFrom(IERC20,address,address,uint256) (internal)

- Contract IUniswapV2Router (Most derived contract)
    - From IUniswapV2Router
        - WETH() (external)

        - getAmountsOut(uint256,address[]) (external)
        - swapExactTokensForTokens(uint256,uint256,address[],address,uint256) (external)

- Contract Fundraiser (Most derived contract)
    - From Ownable
        - transferOwnership(address) (public)
    - From Fundraiser
        - addTrustedDepositToken(address) (external)
        - claim() (external)
        - claimAnyToken(address,uint256) (external)
        - closeSale() (external)
        - constructor() (public)
        - deposit(uint256,address,uint256,uint256,uint256,bytes) (external)
        - getNumberOfHolders() (external)
        - getPendingDivs(address) (public)
        - getStakersList(uint256,uint256) (public)
        - openSale() (external)
        - prefixed(bytes32) (internal)
        - recoverSigner(bytes32,bytes) (internal)
        - removeTrustedDepositToken(address) (external)
        - setVerifyAddress(address) (external)
        - verify(uint256,bytes) (internal)

Slither Results

```
> slither fundraiser.sol

INFO:Detectors:
Fundraiser.getPendingDivs(address) (fundraiser.sol#877-885) uses a dangerous
strict equality:
        - depositedTokens[_holder] == 0 (fundraiser.sol#879)
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#dangerous-strict-equalities
INFO:Detectors:
Fundraiser.deposit(uint256,address,uint256,uint256,uint256,bytes)
(fundraiser.sol#891-946) ignores return value by
uniswapV2Router.swapExactTokensForTokens(amountToDeposit,_amountOutMin,path,addres
s(this),_deadline) (fundraiser.sol#922)
Fundraiser.deposit(uint256,address,uint256,uint256,uint256,bytes)
(fundraiser.sol#891-946) ignores return value by holders.add(msg.sender)
(fundraiser.sol#942)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-
return
INFO:Detectors:
Fundraiser.setVerifyAddress(address).newVerifyAddress (fundraiser.sol#820) lacks a
zero-check on :
                - verifyAddress = newVerifyAddress (fundraiser.sol#821)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-
zero-address-validation
INFO:Detectors:
Reentrancy in Fundraiser.claim() (fundraiser.sol#949-960):
        External calls:
        - IERC20(TRUSTED_REWARD_TOKEN_ADDRESS).safeTransfer(account,pendingDivs)
(fundraiser.sol#956)
        State variables written after the call(s):
        - lastClaimedTime[account] = now (fundraiser.sol#959)
Reentrancy in Fundraiser.deposit(uint256,address,uint256,uint256,uint256,bytes)
(fundraiser.sol#891-946):
        External calls:
        -
IERC20(depositToken).safeTransferFrom(msg.sender,address(this),amountToDeposit)
(fundraiser.sol#898)
        - IERC20(depositToken).safeApprove(address(uniswapV2Router),0)
(fundraiser.sol#903)
        -
IERC20(depositToken).safeApprove(address(uniswapV2Router),amountToDeposit)
(fundraiser.sol#904)
        -
uniswapV2Router.swapExactTokensForTokens(amountToDeposit,_amountOutMin,path,addres
s(this),_deadline) (fundraiser.sol#922)
        State variables written after the call(s):
        - depositedTokens[msg.sender] =
depositedTokens[msg.sender].add(amountToStake) (fundraiser.sol#933)
        - totalDepositedTokens = totalDepositedTokens.add(amountToStake)
(fundraiser.sol#934)
Reentrancy in Fundraiser.deposit(uint256,address,uint256,uint256,uint256,bytes)
(fundraiser.sol#891-946):
        External calls:
        -
IERC20(depositToken).safeTransferFrom(msg.sender,address(this),amountToDeposit)
(fundraiser.sol#898)
        - IERC20(depositToken).safeApprove(address(uniswapV2Router),0)
(fundraiser.sol#903)
        -
IERC20(depositToken).safeApprove(address(uniswapV2Router),amountToDeposit)
(fundraiser.sol#904)
        -
uniswapV2Router.swapExactTokensForTokens(amountToDeposit,_amountOutMin,path,addres
s(this),_deadline) (fundraiser.sol#922)
        -
IERC20(TRUSTED_DEPOSIT_TOKEN_ADDRESS).safeTransfer(owner,contractBalance)
(fundraiser.sol#940)
        State variables written after the call(s):
        - stakingTime[msg.sender] = now (fundraiser.sol#944)
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in Fundraiser.claim() (fundraiser.sol#949-960):
        External calls:
```

```
        - IERC20(TRUSTED_REWARD_TOKEN_ADDRESS).safeTransfer(account,pendingDivs)
(fundraiser.sol#956)
        Event emitted after the call(s):
        - RewardsTransferred(account,pendingDivs) (fundraiser.sol#957)
Reentrancy in Fundraiser.deposit(uint256,address,uint256,uint256,uint256,bytes)
(fundraiser.sol#891-946):
        External calls:
        -
IERC20(depositToken).safeTransferFrom(msg.sender,address(this),amountToDeposit)
(fundraiser.sol#898)
        - IERC20(depositToken).safeApprove(address(uniswapV2Router),0)
(fundraiser.sol#903)
        -
IERC20(depositToken).safeApprove(address(uniswapV2Router),amountToDeposit)
(fundraiser.sol#904)
        -
uniswapV2Router.swapExactTokensForTokens(amountToDeposit,_amountOutMin,path,addres
s(this),_deadline) (fundraiser.sol#922)
        -
IERC20(TRUSTED_DEPOSIT_TOKEN_ADDRESS).safeTransfer(owner,contractBalance)
(fundraiser.sol#940)
        Event emitted after the call(s):
        - Deposit(msg.sender,amountToStake) (fundraiser.sol#945)
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Address.isContract(address) (fundraiser.sol#181-190) uses assembly
        - INLINE ASM (fundraiser.sol#188)
Address._verifyCallResult(bool,bytes,string) (fundraiser.sol#326-343) uses
assembly
        - INLINE ASM (fundraiser.sol#335-338)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-
usage
INFO:Detectors:
solc-0.6.12 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (fundraiser.sol#208-214):
        - (success) = recipient.call{value: amount}() (fundraiser.sol#212)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string)
(fundraiser.sol#269-276):
        - (success,returndata) = target.call{value: value}(data)
(fundraiser.sol#274)
Low level call in Address.functionStaticCall(address,bytes,string)
(fundraiser.sol#294-300):
        - (success,returndata) = target.staticcall(data) (fundraiser.sol#298)
Low level call in Address.functionDelegateCall(address,bytes,string)
(fundraiser.sol#318-324):
        - (success,returndata) = target.delegatecall(data) (fundraiser.sol#322)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-
level-calls
INFO:Detectors:
Function IUniswapV2Router.WETH() (fundraiser.sol#770) is not in mixedCase
Parameter Fundraiser.addTrustedDepositToken(address)._tokenAddress
(fundraiser.sol#865) is not in mixedCase
Parameter Fundraiser.removeTrustedDepositToken(address)._tokenAddress
(fundraiser.sol#871) is not in mixedCase
```

```
Parameter Fundraiser.getPendingDivs(address)._holder (fundraiser.sol#877) is not
in mixedCase
Parameter
Fundraiser.deposit(uint256,address,uint256,uint256,uint256,bytes)._amountOutMin
(fundraiser.sol#891) is not in mixedCase
Parameter
Fundraiser.deposit(uint256,address,uint256,uint256,uint256,bytes)._deadline
(fundraiser.sol#891) is not in mixedCase
Constant Fundraiser.uniswapV2Router (fundraiser.sol#812) is not in
UPPER_CASE_WITH_UNDERSCORES
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) (fundraiser.sol#620-624)
getStakersList(uint256,uint256) should be declared external:
        - Fundraiser.getStakersList(uint256,uint256) (fundraiser.sol#962-987)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-
function-that-could-be-declared-external
INFO:Slither:fundraiser.sol analyzed (8 contracts with 72 detectors), 25 result(s)
found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and
Github integration
```