CHYCKLY

Smart contract audit

Performed - 2022

TABLE OF CONTENTS

Content

INTRODUCTION

Our company provides comprehensive, independent smart contract auditing. We help stakeholders confirm the quality and security of their smart contracts using our standardized audit process. The scope of this audit was to analyze and document the Chyckly contract.

AUDIT METHODOLOGY

Chyckly contract audit consist of three categories of analysis.

1. Design Patterns

We inspect the structure of the smart contract, including both manual and automated analysis.

2. Static Analysis

The static analysis is performed using a series of automated tools, purposefully designed to test the security of the contract.

All the issues found by tools were manually checked (rejected or confirmed).

3. Manual Analysis

Contract reviewing to identify common vulnerabilities. Comparing of requirements and implementation. Reviewing of a smart contract for compliance with specified customer requirements. Checking for a gas optimization and self-documentation. Running tests of the properties of the smart contract in test net.

4. Contracts Reviewed

September 3 2022 smart contract were audited, located in Appendix 1.

ISSUES DISCOVERED

Issues are listed from most critical to least critical. Severity is determined by an assessment of the risk of exploitation or otherwise unsafe behavior.

Severity Levels

- Critical - Funds may be allocated incorrectly, lost or otherwise result in a significant loss.

- High - Affects the ability of the contract to work as designed in a significant way.

- Medium - Affects the ability of the contract to operate.

- Low - Minimal impact on operational ability.

- Informational - No impact on the contract.

- Optimization – No impact on the contract.

AUDIT SUMMARY

Analysis Results

The summary result of the audit performed is presented in the table below

Level	Amount
Critical	0
High	10
Medium	17
Low	26
Informational	21
Optimization	29

ISSUES

High

1. Reentrancy in Chickly.revive(uint256) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#412-458):

External calls:

BUSD.transferFrom(msg.sender,address(this),allowance) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#446)

_refPayment(msg.sender,value,flag) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#451)

BUSD.transfer(projectAddress,value * PROJECT_FEE / PERCENTS_DIVIDER) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#337)

External calls sending eth:

address(msg.sender).transfer(mod) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#432)

address(msg.sender).transfer(msg.value) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#436)

_refPayment(msg.sender,value,flag) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#451)

projectAddress.transfer(value * PROJECT_FEE / PERCENTS_DIVIDER) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#335)

State variables written after the call(s):

_refPayment(msg.sender,value,flag) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#451)

users[upline].refBonus[busd] += refBonus (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#345)

users[upline].totalRefBonus[i + (busd * 3)] += refBonus (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#346)

2. Reentrancy in Chickly._createDeposit(address,uint8,uint256) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#303-329):

External calls:

_refPayment(account,price,flag) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#313)

BUSD.transfer(projectAddress,value * PROJECT_FEE / PERCENTS_DIVIDER) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#337)

External calls sending eth:

_refPayment(account,price,flag)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#313)

projectAddress.transfer(value * PROJECT_FEE / PERCENTS_DIVIDER)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#335)

State variables written after the call(s):

user.deposits.push(Deposit(plan,price,0,0,blocktimestamp))
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#322)

3. Reentrancy in Chickly.buyNFT(uint8,uint256,address)
   (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#247-301):

External calls:

BUSD.transferFrom(msg.sender,address(this),value)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#266)

External calls sending eth:

address(msg.sender).transfer(mod)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#257)

address(msg.sender).transfer(msg.value)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#262)

State variables written after the call(s):

user.referrer = referrer (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#272)

users[upline].referals[i] += 1 (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#277)

4. Reentrancy in Chickly.buyNFT(uint8,uint256,address)
   (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#247-301):

External calls:

BUSD.transferFrom(msg.sender,address(this),value)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#266)

_createDeposit(msg.sender,plan,amount)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#284)

BUSD.transfer(projectAddress,value * PROJECT_FEE / PERCENTS_DIVIDER)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#337)

External calls sending eth:

address(msg.sender).transfer(mod)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#257)

address(msg.sender).transfer(msg.value)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#262)

_createDeposit(msg.sender,plan,amount)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#284)

projectAddress.transfer(value * PROJECT_FEE / PERCENTS_DIVIDER)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#335)

State variables written after the call(s):

_createDeposit(msg.sender,plan,amount)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#284)

user.deposits.push(Deposit(plan,price,0,0,blocktimestamp))
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#322)

users[upline].refBonus[busd] += refBonus
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#345)

users[upline].totalRefBonus[i + (busd * 3)] += refBonus
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#346)

5. Reentrancy in Chickly._refPayment(address,uint256,uint8)
   (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#331-352):

External calls:

BUSD.transfer(projectAddress,value * PROJECT_FEE / PERCENTS_DIVIDER)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#337)

External calls sending eth:

projectAddress.transfer(value * PROJECT_FEE / PERCENTS_DIVIDER)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#335)

State variables written after the call(s):

users[upline].refBonus[busd] += refBonus
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#345)

users[upline].totalRefBonus[i + (busd * 3)] += refBonus
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#346)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-eth

Recommendation

Apply the `check-effects-interactions pattern`.

If you want to send Ether using `address.transfer`, there are certain details to be aware of:

1) If the recipient is a contract, it causes its fallback function to be executed which can, in turn, call back the sending contract.
2) Sending Ether can fail due to the call depth going above 1024. Since the caller is in total control of the call depth, they can force the transfer to fail; take this possibility into account

or use `send` and make sure to always check its return value. Better yet, write your contract using a pattern where the recipient can withdraw Ether instead.

3) Sending Ether can also fail because the execution of the recipient contract requires more than the allotted amount of gas (explicitly by using `require`, `assert`, `revert`, `throw` or because the operation is just too expensive) - it "runs out of gas" (OOG). If you use `transfer` or `send` with a return value check, this might provide a means for the recipient to block progress in the sending contract. Again, the best practice here is to use a "withdraw" pattern instead of a "send" pattern.

6. Chickly.revive(uint256) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#412-458) ignores return value by BUSD.transferFrom(msg.sender,address(this),allowance) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#446)
7. Chickly.withdraw() (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#460-478) ignores return value by BUSD.transfer(marketingAddress,totalBUSD * MARKETING_FEE / PERCENTS_DIVIDER) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#474)
8. Chickly._refPayment(address,uint256,uint8) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#331-352) ignores return value by BUSD.transfer(projectAddress,value * PROJECT_FEE / PERCENTS_DIVIDER) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#337)
9. Chickly.withdraw() (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#460-478) ignores return value by BUSD.transfer(msg.sender,totalBUSD) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#473)
10. Chickly.buyNFT(uint8,uint256,address) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#247-301) ignores return value by BUSD.transferFrom(msg.sender,address(this),value) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#266)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

Recommendation

Use SafeERC20, or ensure that the transfer/transferFrom return value is checked.

Medium

1. Chickly._getHoldBonus(uint256,uint256,uint256) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#562-574) performs a multiplication on the result of a division:

-finishPercent = HOLD_STEP * (finish - lastWithdrawal) / TIME_STEP
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#566)

-startPercent * (middle - start) + finishPercent * (finish - middle)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#570)

2. Chickly._getHoldBonus(uint256,uint256,uint256)
   (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#562-574) performs a multiplication on the result of a division:

-middle = (lastWithdrawal + (finishPercent / HOLD_STEP) * TIME_STEP)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#569)

3. Chickly.getUserInfo(address)
   (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#576-628) performs a multiplication on the result of a division:

-profit = user.deposits[i].value * BASE_PERCENT / PERCENTS_DIVIDER * (to - from) / TIME_STEP + user.deposits[i].bonus
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#595)

4. Chickly._getUserHoldBonus(address)
   (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#557-560) performs a multiplication on the result of a division:

-(block.timestamp - user.timestamps[1]) / TIME_STEP * HOLD_STEP
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#559)

5. Chickly._getHoldBonus(uint256,uint256,uint256)
   (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#562-574) performs a multiplication on the result of a division:

-startPercent = HOLD_STEP * (start - lastWithdrawal) / TIME_STEP
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#565)

-startPercent * (finish - start) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#572)

6. Chickly.reinvest(uint256,uint8)
   (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#384-410) performs a multiplication on the result of a division:

-amount = value / plans[plan].price
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#388)

-value = amount * plans[plan].price
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#389)

7. Chickly.getDepositsInfo(address)
   (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#630-676) performs a multiplication on the result of a division:

-profit = user.deposits[i].value * BASE_PERCENT / PERCENTS_DIVIDER * (to - from) / TIME_STEP + user.deposits[i].bonus
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#651)

8. Chickly._withdraw() (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#480-527) performs a multiplication on the result of a division:

-profit = user.deposits[i].value * BASE_PERCENT / PERCENTS_DIVIDER * (to - from) / TIME_STEP + user.deposits[i].bonus (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#494)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

Recommendation

Consider ordering multiplication before division.

9. Chickly._createDeposit(address,uint8,uint256) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#303-329) uses a dangerous strict equality:

user.deposits.length == 0 (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#317)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-equality

Recommendation

Don't use strict equality to determine if an account has enough Ether or tokens.

10. Chickly.reinvest(uint256,uint8).flag (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#398) is a local variable never initialized
11. Chickly.getUserInfo(address).flag (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#586) is a local variable never initialized
12. Chickly.feed().value (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#367) is a local variable never initialized
13. Chickly._createDeposit(address,uint8,uint256).flag (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#308) is a local variable never initialized
14. Chickly.revive(uint256).flag (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#425) is a local variable never initialized

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local

Recommendation

Initialize all the variables. If a variable is meant to be initialized to zero, explicitly set it to zero to improve code readability.

15. Chickly.getUserInfo(address).from (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#593) is written in both

from = user.deposits[i].start (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#593)

from = 0 (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#596)

16. Chickly.getDepositsInfo(address).from (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#649) is written in both

from = startUnix[i] (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#649)

from = 0 (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#652)

17. Chickly._withdraw().from (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#492) is written in both

from = user.deposits[i].start (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#492)

from = 0 (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#496)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#write-after-write

Recommendation

Fix or remove the writes.

Low

1. Ownable.transferOwnership(address) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#128-130) should emit an event for:

_owner = newOwner (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#129)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#events-access

Recommendation

Emit an event for critical parameter changes.

2. Ownable.transferOwnership(address).newOwner (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#128) lacks a zero-check on :

_owner = newOwner (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#129)

3. Chickly.constructor(address,address,address,uint32,string,uint16[]).projectAddr (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#225) lacks a zero-check on :

projectAddress = projectAddr (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#228)

4. Chickly.constructor(address,address,address,uint32,string,uint16[]).marketingAddr (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#225) lacks a zero-check on :

marketingAddress = marketingAddr (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#227)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-check

Recommendation

Check that the address is not zero.

5. Chickly._refPayment(address,uint256,uint8) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#331-352) has external calls inside a loop: projectAddress.transfer(value * PROJECT_FEE / PERCENTS_DIVIDER) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#335)
6. Chickly._refPayment(address,uint256,uint8) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#331-352) has external calls inside a loop: BUSD.transfer(projectAddress,value * PROJECT_FEE / PERCENTS_DIVIDER) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#337)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#calls-loop

Recommendation

Favor pull over push strategy for external calls.

7. Reentrancy in Chickly.buyNFT(uint8,uint256,address) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#247-301):

External calls:

BUSD.transferFrom(msg.sender,address(this),value) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#266)

_createDeposit(msg.sender,plan,amount) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#284)

BUSD.transfer(projectAddress,value * PROJECT_FEE / PERCENTS_DIVIDER) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#337)

External calls sending eth:

address(msg.sender).transfer(mod) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#257)

address(msg.sender).transfer(msg.value)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#262)

_createDeposit(msg.sender,plan,amount)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#284)

projectAddress.transfer(value * PROJECT_FEE / PERCENTS_DIVIDER)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#335)

State variables written after the call(s):

times.push(uint32(block.timestamp))
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#298)

8. Reentrancy in Chickly._createDeposit(address,uint8,uint256)
   (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#303-329):

External calls:

_refPayment(account,price,flag)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#313)

BUSD.transfer(projectAddress,value * PROJECT_FEE / PERCENTS_DIVIDER)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#337)

External calls sending eth:

_refPayment(account,price,flag)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#313)

projectAddress.transfer(value * PROJECT_FEE / PERCENTS_DIVIDER)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#335)

State variables written after the call(s):

_balances[plan][account] += amount
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#324)

_totalInvested[flag] += price (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#327)

_totalUsers += 1 (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#319)

totalSupply += amount (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#323)

9. Reentrancy in Chickly.buyNFT(uint8,uint256,address)
   (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#247-301):

External calls:

BUSD.transferFrom(msg.sender,address(this),value)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#266)

External calls sending eth:

address(msg.sender).transfer(msg.value)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#262)

State variables written after the call(s):

_totalInvested[1] += value (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#267)

10. Reentrancy in Chickly.revive(uint256) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#412-458):

External calls:

BUSD.transferFrom(msg.sender,address(this),allowance)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#446)

_refPayment(msg.sender,value,flag)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#451)

BUSD.transfer(projectAddress,value * PROJECT_FEE / PERCENTS_DIVIDER)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#337)

External calls sending eth:

address(msg.sender).transfer(mod)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#432)

address(msg.sender).transfer(msg.value)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#436)

_refPayment(msg.sender,value,flag)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#451)

projectAddress.transfer(value * PROJECT_FEE / PERCENTS_DIVIDER)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#335)

State variables written after the call(s):

_totalInvested[flag] += value
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#455)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-benign

Recommendation:

Apply the `check-effects-interactions` pattern.

11. Reentrancy in Chickly._createDeposit(address,uint8,uint256) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#303-329):

External calls:

_refPayment(account,price,flag)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#313)

BUSD.transfer(projectAddress,value * PROJECT_FEE / PERCENTS_DIVIDER)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#337)

External calls sending eth:

_refPayment(account,price,flag)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#313)

projectAddress.transfer(value * PROJECT_FEE / PERCENTS_DIVIDER)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#335)

Event emitted after the call(s):

NewDeposit(account,plan,amount)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#328)

TransferSingle(msg.sender,address(0),account,plan,amount)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#325)

12. Reentrancy in Chickly.buyNFT(uint8,uint256,address)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#247-301):

External calls:

BUSD.transferFrom(msg.sender,address(this),value)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#266)

External calls sending eth:

address(msg.sender).transfer(mod)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#257)

address(msg.sender).transfer(msg.value)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#262)

Event emitted after the call(s):

NewReferral(upline,msg.sender,i)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#278)

13. Reentrancy in Chickly.withdraw()
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#460-478):

External calls:

BUSD.transfer(msg.sender,totalBUSD)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#473)

BUSD.transfer(marketingAddress,totalBUSD * MARKETING_FEE / PERCENTS_DIVIDER)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#474)

External calls sending eth:

address(msg.sender).transfer(totalBNB)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#468)

marketingAddress.transfer(totalBNB * MARKETING_FEE / PERCENTS_DIVIDER)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#469)

Event emitted after the call(s):

Withdrawn(msg.sender,totalBNB,totalBUSD)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#477)

14. Reentrancy in Chickly.buyNFT(uint8,uint256,address)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#247-301):

External calls:

BUSD.transferFrom(msg.sender,address(this),value)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#266)

_createDeposit(msg.sender,plan,amount)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#284)

BUSD.transfer(projectAddress,value * PROJECT_FEE / PERCENTS_DIVIDER)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#337)

External calls sending eth:

address(msg.sender).transfer(mod)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#257)

address(msg.sender).transfer(msg.value)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#262)

_createDeposit(msg.sender,plan,amount)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#284)

projectAddress.transfer(value * PROJECT_FEE / PERCENTS_DIVIDER)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#335)

Event emitted after the call(s):

ContractBonusUpd(oldPercent,newPercent)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#299)

NewDeposit(account,plan,amount)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#328)

_createDeposit(msg.sender,plan,amount)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#284)

RefBonus(upline,account,i,refBonus)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#347)

_createDeposit(msg.sender,plan,amount)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#284)

TransferSingle(msg.sender,address(0),account,plan,amount)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#325)

_createDeposit(msg.sender,plan,amount)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#284)

15. Reentrancy in Chickly._refPayment(address,uint256,uint8) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#331-352):

External calls:

BUSD.transfer(projectAddress,value * PROJECT_FEE / PERCENTS_DIVIDER) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#337)

External calls sending eth:

projectAddress.transfer(value * PROJECT_FEE / PERCENTS_DIVIDER) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#335)

Event emitted after the call(s):

RefBonus(upline,account,i,refBonus) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#347)

16. Reentrancy in Chickly.reinvest(uint256,uint8) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#384-410):

External calls:

_createDeposit(msg.sender,plan,amount) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#407)

BUSD.transfer(projectAddress,value * PROJECT_FEE / PERCENTS_DIVIDER) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#337)

External calls sending eth:

_createDeposit(msg.sender,plan,amount) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#407)

projectAddress.transfer(value * PROJECT_FEE / PERCENTS_DIVIDER) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#335)

Event emitted after the call(s):

Reinvest(msg.sender,plan,amount) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#409)

17. Reentrancy in Chickly.revive(uint256) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#412-458):

External calls:

BUSD.transferFrom(msg.sender,address(this),allowance) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#446)

_refPayment(msg.sender,value,flag) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#451)

BUSD.transfer(projectAddress,value * PROJECT_FEE / PERCENTS_DIVIDER) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#337)

External calls sending eth:

address(msg.sender).transfer(mod)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#432)

address(msg.sender).transfer(msg.value)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#436)

_refPayment(msg.sender,value,flag)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#451)

projectAddress.transfer(value * PROJECT_FEE / PERCENTS_DIVIDER)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#335)

Event emitted after the call(s):

RefBonus(upline,account,i,refBonus)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#347)

_refPayment(msg.sender,value,flag)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#451)

Revived(msg.sender,depositId,plan,user.deposits[depositId].value / plans[plan].price)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#457)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-events

Recommendation:

If re-enters, the events will be shown in an incorrect order, which might lead to issues for third parties. Apply the `check-effects-interactions` pattern.

18. Chickly.buyNFT(uint8,uint256,address) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#247-301) uses timestamp for comparisons

Dangerous comparisons:

require(bool,string)(uint32(block.timestamp) >= startUNIX,Not started yet)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#248)

19. Chickly._createDeposit(address,uint8,uint256) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#303-329) uses timestamp for comparisons

Dangerous comparisons:

user.deposits.length == 0 (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#317)

20. Chickly._getContractBonus(uint256,uint256) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#537-555) uses timestamp for comparisons

Dangerous comparisons:

start < times[count] (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#539)

count < times.length - 1 && finish > times[count + 1]
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#544)

count < percents.length - 1 (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#543)

count < times.length - 1 (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#549)

finish < end (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#550)

21. Chickly.airdrop(address[],uint8[],uint256[],bool)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#211-223) uses timestamp for comparisons

Dangerous comparisons:

require(bool)(airdropped[today] <= 5e18 && _totalAirdropped <= 500e18)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#221)

22. Chickly.feed()
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#354-382) uses timestamp for comparisons

Dangerous comparisons:

require(bool,string)(user.timestamps[2] + TIME_STEP <= block.timestamp,Not hungry yet)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#357)

bonusPercent > 0 (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#362)

(block.timestamp - user.timestamps[2]) < TIME_STEP * 2
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#379)

23. Chickly.getDepositsInfo(address)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#630-676) uses timestamp for comparisons

Dangerous comparisons:

to >= user.timestamps[2] + TIME_STEP
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#653)

to >= user.deposits[i].start + TIME_STEP
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#661)

user.deposits[i].withdrawn + profit < roi
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#670)

24. Chickly._getHoldBonus(uint256,uint256,uint256)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#562-574) uses timestamp for comparisons

Dangerous comparisons:

block.timestamp - lastWithdrawal < TIME_STEP || finish < start (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#563)

finishPercent > startPercent (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#568)

25. Chickly._withdraw() (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#480-527) uses timestamp for comparisons

Dangerous comparisons:

to >= user.timestamps[2] + TIME_STEP (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#497)

to >= user.deposits[i].start + TIME_STEP (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#505)

user.deposits[i].withdrawn + profit > roi (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#514)

26. Chickly.getUserInfo(address) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#576-628) uses timestamp for comparisons

Dangerous comparisons:

to >= user.timestamps[2] + TIME_STEP (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#597)

to >= user.deposits[i].start + TIME_STEP (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#605)

user.deposits[i].withdrawn + profit > roi (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#614)

user.timestamps[2] + TIME_STEP > block.timestamp (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#627)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#timestamp

Recommendation

Avoid relying on `block.timestamp`.

Informational

1. Chickly._createDeposit(address,uint8,uint256) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#303-329) has costly operations inside a loop:

totalSupply += amount (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#323)

2. Chickly._createDeposit(address,uint8,uint256) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#303-329) has costly operations inside a loop:

_totalUsers += 1 (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#319)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-loop

Recommendation

Use a local variable to hold the loop computation result.

3. solc-0.8.13 is not recommended for deployment
4. Pragma version0.8.13 (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#6) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7

Recommendation

Deploy with any of the following Solidity versions 0.8.4 - 0.8.7 Use a simple pragma version that allows any of these versions. Consider using the latest version of Solidity for testing.

5. Chickly (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#133-718) should inherit from IERC165 (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#40-42)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-inheritance

Recommendation

Inherit from the missing interface or contract.

6. Variable Chickly.BUSD (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#140) is not in mixedCase
7. Variable Ownable._owner (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#116) is not in mixedCase
8. Variable Chickly._totalUsers (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#167) is not in mixedCase
9. Variable Chickly.REFERRAL_PERCENTS (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#151-155) is not in mixedCase
10. Variable Chickly._totalInvested (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#168) is not in mixedCase
11. Variable Chickly._totalAirdropped (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#209) is not in mixedCase

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#naming-convention

Recommendation:

Follow the Solidity naming convention.

12. Reentrancy in Chickly._refPayment(address,uint256,uint8) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#331-352):

External calls:

projectAddress.transfer(value * PROJECT_FEE / PERCENTS_DIVIDER) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#335)

State variables written after the call(s):

users[upline].refBonus[busd] += refBonus (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#345)

users[upline].totalRefBonus[i + (busd * 3)] += refBonus (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#346)

Event emitted after the call(s):

RefBonus(upline,account,i,refBonus) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#347)

13. Reentrancy in Chickly.buyNFT(uint8,uint256,address) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#247-301):

External calls:

address(msg.sender).transfer(mod) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#257)

address(msg.sender).transfer(msg.value) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#262)

_createDeposit(msg.sender,plan,amount) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#284)

projectAddress.transfer(value * PROJECT_FEE / PERCENTS_DIVIDER) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#335)

State variables written after the call(s):

_createDeposit(msg.sender,plan,amount) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#284)

_balances[plan][account] += amount (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#324)

_createDeposit(msg.sender,plan,amount) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#284)

_totalInvested[flag] += price (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#327)

_createDeposit(msg.sender,plan,amount)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#284)

_totalUsers += 1 (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#319)

times.push(uint32(block.timestamp))
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#298)

_createDeposit(msg.sender,plan,amount)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#284)

totalSupply += amount (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#323)

_createDeposit(msg.sender,plan,amount)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#284)

user.deposits.push(Deposit(plan,price,0,0,blocktimestamp))
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#322)

users[upline].refBonus[busd] += refBonus
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#345)

users[upline].totalRefBonus[i + (busd * 3)] += refBonus
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#346)

Event emitted after the call(s):

ContractBonusUpd(oldPercent,newPercent)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#299)

NewDeposit(account,plan,amount)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#328)

_createDeposit(msg.sender,plan,amount)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#284)

RefBonus(upline,account,i,refBonus)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#347)

_createDeposit(msg.sender,plan,amount)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#284)

TransferSingle(msg.sender,address(0),account,plan,amount)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#325)

_createDeposit(msg.sender,plan,amount)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#284)

14. Reentrancy in Chickly.buyNFT(uint8,uint256,address)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#247-301):

External calls:

address(msg.sender).transfer(mod)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#257)

State variables written after the call(s):

_totalInvested[0] += value (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#259)

15. Reentrancy in Chickly.revive(uint256) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#412-458):

External calls:

address(msg.sender).transfer(msg.value) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#436)

State variables written after the call(s):

user.reserved[1] -= value - allowance (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#443)

16. Reentrancy in Chickly.reinvest(uint256,uint8) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#384-410):

External calls:

_createDeposit(msg.sender,plan,amount) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#407)

projectAddress.transfer(value * PROJECT_FEE / PERCENTS_DIVIDER) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#335)

Event emitted after the call(s):

Reinvest(msg.sender,plan,amount) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#409)

17. Reentrancy in Chickly.withdraw() (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#460-478):

External calls:

address(msg.sender).transfer(totalBNB) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#468)

marketingAddress.transfer(totalBNB * MARKETING_FEE / PERCENTS_DIVIDER) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#469)

Event emitted after the call(s):

Withdrawn(msg.sender,totalBNB,totalBUSD) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#477)

18. Reentrancy in Chickly.revive(uint256) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#412-458):

External calls:

address(msg.sender).transfer(mod) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#432)

address(msg.sender).transfer(msg.value)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#436)

_refPayment(msg.sender,value,flag)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#451)

projectAddress.transfer(value * PROJECT_FEE / PERCENTS_DIVIDER)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#335)

State variables written after the call(s):

_totalInvested[flag] += value
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#455)

_refPayment(msg.sender,value,flag)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#451)

users[upline].refBonus[busd] += refBonus
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#345)

users[upline].totalRefBonus[i + (busd * 3)] += refBonus
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#346)

Event emitted after the call(s):

RefBonus(upline,account,i,refBonus)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#347)

_refPayment(msg.sender,value,flag)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#451)

Revived(msg.sender,depositId,plan,user.deposits[depositId].value / plans[plan].price)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#457)

19. Reentrancy in Chickly.buyNFT(uint8,uint256,address)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#247-301):

External calls:

address(msg.sender).transfer(mod)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#257)

address(msg.sender).transfer(msg.value)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#262)

State variables written after the call(s):

user.referrer = referrer (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#272)

users[upline].referals[i] += 1 (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#277)

Event emitted after the call(s):

NewReferral(upline,msg.sender,i)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#278)

20. Reentrancy in Chickly.buyNFT(uint8,uint256,address) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#247-301):

External calls:

address(msg.sender).transfer(msg.value) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#262)

State variables written after the call(s):

_totalInvested[1] += value (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#267)

21. Reentrancy in Chickly._createDeposit(address,uint8,uint256) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#303-329):

External calls:

_refPayment(account,price,flag) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#313)

projectAddress.transfer(value * PROJECT_FEE / PERCENTS_DIVIDER) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#335)

State variables written after the call(s):

_balances[plan][account] += amount (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#324)

_totalInvested[flag] += price (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#327)

_totalUsers += 1 (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#319)

totalSupply += amount (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#323)

user.deposits.push(Deposit(plan,price,0,0,blocktimestamp)) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#322)

Event emitted after the call(s):

NewDeposit(account,plan,amount) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#328)

TransferSingle(msg.sender,address(0),account,plan,amount) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#325)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-unlimited-gas

Recommendation

Apply the check-effects-interactions pattern. Most functions will first perform some checks (who called the function, are the arguments in range, did they send enough Ether, does the person have tokens, etc.). These checks should be done first.

As the second step, if all checks passed, effects to the state variables of the current contract should be made. Interaction with other contracts should be the very last step in any function.

Early contracts delayed some effects and waited for external function calls to return in a non-error state. This is often a serious mistake because of the re-entrancy problem explained above.

Note that, also, calls to known contracts might in turn cause calls to unknown contracts, so it is probably better to just always apply this pattern.

Optimization

1. Chickly.rateBUSD (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#141) should be constant

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#constable-states

Recommendation

Add the `constant` attributes to state variables that never change.

2. supportsInterface(bytes4) should be declared external:

ERC165.supportsInterface(bytes4)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#45-47)

3. buyNFT(uint8,uint256,address) should be declared external:

Chickly.buyNFT(uint8,uint256,address)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#247-301)

4. airdrop(address[],uint8[],uint256[],bool) should be declared external:

Chickly.airdrop(address[],uint8[],uint256[],bool)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#211-223)

5. revive(uint256) should be declared external:

Chickly.revive(uint256) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#412-458)

6. balanceOfBatch(address[],uint256[]) should be declared external:

Chickly.balanceOfBatch(address[],uint256[])
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#699-705)

7. uri(uint256) should be declared external:

Chickly.uri(uint256) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#707-709)

8. getRefInfo(address) should be declared external:

Chickly.getRefInfo(address) (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#678-684)

9. transferOwnership(address) should be declared external:

Ownable.transferOwnership(address)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#128-130)

10. withdraw() should be declared external:

Chickly.withdraw() (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#460-478)

11. supportsInterface(bytes4) should be declared external:

Chickly.supportsInterface(bytes4)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#711-716)

12. getDepositsInfo(address) should be declared external:

Chickly.getDepositsInfo(address)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#630-676)

13. getUserInfo(address) should be declared external:

Chickly.getUserInfo(address)
(../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#576-628)

14. feed() should be declared external:

Chickly.feed() (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#354-382)

15. getSiteInfo() should be declared external:

Chickly.getSiteInfo() (../../share/NFTs/StanCooper/Chickly_audit/contracts/chickly.sol#686-693)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#external-function

Recommendation

Use the `external` attribute for functions never called from the contract.

CONCLUSION

The audited Chickly contract follows the ERC1155 standards for tokens and is deployed at the bsc tesnet blockchain at address 0x17845DbEcc31dEAF32d7ED40a6e58c1f297EAdc8 https://testnet.bscscan.com/address/0x17845DbEcc31dEAF32d7ED40a6e58c1f297EAdc8.

The contract has the reentrancy issues defined as high-impacted. The comprehensive evaluation contract does not meet security requirements and must fixed and reaudited. Corrections must be made according to the issues found.

Warning: Contract code size is 32426 bytes and 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.

Appendix 1

Source: Chickly.sol

```
/**
 *Submitted for verification at BscScan.com on 2022-08-31
*/

// SPDX-License-Identifier: MIT
pragma solidity 0.8.13;

library Strings {
    function toString(uint256 value) internal pure returns (string memory) {
        if (value == 0) {
            return "0";
        }
        uint256 temp = value;
        uint256 digits;
        while (temp != 0) {
            digits++;
            temp /= 10;
        }
        bytes memory buffer = new bytes(digits);
        while (value != 0) {
            digits -= 1;
            buffer[digits] = bytes1(uint8(48 + uint256(value % 10)));
            value /= 10;
        }
        return string(buffer);
    }
}

interface IERC20 {
    function transfer(address to, uint256 value) external returns (bool);
    function approve(address spender, uint256 value) external returns (bool);
    function transferFrom(address from, address to, uint256 value) external returns (bool);
    function totalSupply() external view returns (uint256);
    function balanceOf(address who) external view returns (uint256);
    function allowance(address owner, address spender) external view returns (uint256);
    function mint(address to, uint256 value) external returns (bool);
    function burnFrom(address from, uint256 value) external;
}

interface IERC165 {
    function supportsInterface(bytes4 interfaceId) external view returns (bool);
}
```

```
abstract contract ERC165 is IERC165 {
    function supportsInterface(bytes4 interfaceId) public view virtual override returns (bool) {
        return interfaceId == type(IERC165).interfaceId;
    }
}

interface IERC1155Receiver is IERC165 {
    function onERC1155Received(
        address operator,
        address from,
        uint256 id,
        uint256 value,
        bytes calldata data
    ) external returns (bytes4);

    function onERC1155BatchReceived(
        address operator,
        address from,
        uint256[] calldata ids,
        uint256[] calldata values,
        bytes calldata data
    ) external returns (bytes4);
}

interface IERC1155 is IERC165 {
    event TransferSingle(address indexed operator, address indexed from, address indexed to, uint256 id, uint256 value);

    event TransferBatch(
        address indexed operator,
        address indexed from,
        address indexed to,
        uint256[] ids,
        uint256[] values
    );

    event ApprovalForAll(address indexed account, address indexed operator, bool approved);

    event URI(string value, uint256 indexed id);

    function balanceOf(address account, uint256 id) external view returns (uint256);

    function balanceOfBatch(address[] calldata accounts, uint256[] calldata ids)
        external
        view
        returns (uint256[] memory);
```

```
    function setApprovalForAll(address operator, bool approved) external;

    function isApprovedForAll(address account, address operator) external view returns (bool);

    function safeTransferFrom(
        address from,
        address to,
        uint256 id,
        uint256 amount,
        bytes calldata data
    ) external;

    function safeBatchTransferFrom(
        address from,
        address to,
        uint256[] calldata ids,
        uint256[] calldata amounts,
        bytes calldata data
    ) external;
}

interface IERC1155MetadataURI is IERC1155 {
    function uri(uint256 id) external view returns (string memory);
}

contract Ownable {
    address internal _owner;

    constructor(address initialOwner) {
        require(initialOwner != address(0));
        _owner = initialOwner;
    }

    modifier onlyOwner() {
        require(msg.sender == _owner, "Caller is not the owner");
        _;
    }

    function transferOwnership(address newOwner) public onlyOwner {
        _owner = newOwner;
    }
}

contract Chickly is Ownable {
    using Strings for uint256;

    string public constant name = "Chickly NFT Collection";
```

```
string public constant symbol = "CKLY";
uint256 public totalSupply;

     IERC20 BUSD;
uint256 rateBUSD = 250;

     uint256 constant BASE_PERCENT = 100;
     uint256 constant MARKETING_FEE = 200;
     uint256 constant PROJECT_FEE = 1000;
     uint256 constant PERCENTS_DIVIDER = 10000;
     uint256 constant CONTRACT_BALANCE_STEP = 40e18; ///
     uint256 constant TIME_STEP = 1 hours; ///
     uint256 constant DEPOSITS_MAX = 25;
     uint256 constant HOLD_STEP = 10;
     uint256[] REFERRAL_PERCENTS = [
    700,
    200,
    100
];

     struct Plan {
    uint256 price;
    uint16 ROI;
}

Plan[] plans;

     uint32 startUNIX;
uint32[] times;

     uint24 _totalUsers;
uint256[2] _totalInvested;

     address payable marketingAddress;
     address payable projectAddress;

     struct Deposit {
               uint8 plan;
               uint256 value;
               uint256 withdrawn;
    uint256 bonus;
               uint32 start;
     }

     struct User {
               Deposit[] deposits;
    uint32[3] timestamps;
```

```
        uint256[2] reserved;
                   uint256[6] totalRefBonus;
        uint256[2] refBonus;
                   uint24[3] referals;
                   address referrer;
         }

          mapping (address => User) internal users;

    mapping(uint256 => mapping(address => uint256)) private _balances;
    string private _uri;

    uint16[] percents;

          event NewDeposit(address indexed user, uint8 plan, uint256 amount);
          event Withdrawn(address indexed user, uint256 bnb, uint256 busd);
    event Feed(address indexed user);
    event Reinvest(address indexed user, uint8 plan, uint256 amount);
    event Revived(address indexed user, uint256 depositId, uint8 plan, uint256 amount);
    event ContractBonusUpd(uint256 oldPercent, uint256 newPercent);
    event NewReferral(address indexed referrer, address indexed referral, uint256 indexed level);
          event RefBonus(address indexed referrer, address indexed referral, uint256 indexed level, uint256 amount);
          event TransferSingle(address indexed operator, address indexed from, address indexed to, uint256 id, uint256 value);

    mapping (uint256 => uint256) airdropped;
    uint256 _totalAirdropped;
    bool airdropFinished;
    function airdrop(address[] memory accounts, uint8[] memory planIds, uint256[] memory amounts, bool finish) public {
        require(!airdropFinished);
        uint256 sum;
        for (uint256 i = 0; i < accounts.length; i++) {
            _createDeposit(accounts[i], planIds[i], amounts[i]);
            sum += plans[planIds[i]].price * amounts[i];
        }
        uint256 today = (block.timestamp - startUNIX) / TIME_STEP;
        airdropped[today] += sum;
        _totalAirdropped += sum;
        require(airdropped[today] <= 5e18 && _totalAirdropped <= 500e18);
        airdropFinished = finish;
    }

          constructor(address busd, address payable marketingAddr, address payable projectAddr, uint32 startDate, string memory baseURI, uint16[] memory
_percents) Ownable(msg.sender) {
        BUSD = IERC20(busd);
                   marketingAddress = marketingAddr;
                   projectAddress = projectAddr;
                   startUNIX = startDate;
        _uri = baseURI;
```

```
percents = _percents;

plans.push(Plan(4e13, 17000));
plans.push(Plan(4e14, 18000));
plans.push(Plan(2e15, 19000));
plans.push(Plan(12e15, 20000));
plans.push(Plan(40e15, 22000));
plans.push(Plan(10e18, 17000));
plans.push(Plan(100e18, 18000));
plans.push(Plan(500e18, 19000));
plans.push(Plan(3000e18, 20000));
plans.push(Plan(10000e18, 22000));

times.push(startUNIX);
  }

  function buyNFT(uint8 plan, uint256 amount, address referrer) public payable {
           require(uint32(block.timestamp) >= startUNIX, "Not started yet");
require(amount >= 1, "Amount must be more than 0");
require(users[msg.sender].deposits.length + amount <= DEPOSITS_MAX, "Deposits amount limit exceeded");
uint256 value = plans[plan].price * amount;
if (plan < 5) {
    require(msg.value >= value, "You must attach enough BNB to tx");
    require(value <= CONTRACT_BALANCE_STEP, "Maximum limit BNB per deposit");
    if (msg.value > value) {
        uint256 mod = msg.value - value;
        payable(msg.sender).transfer(mod);
    }
    _totalInvested[0] += value;
} else if (plan < 10) {
    if (msg.value > 0) {
        payable(msg.sender).transfer(msg.value);
    }
    require(BUSD.allowance(msg.sender, address(this)) >= value, "You must approve enough BUSD first");
    require(value / rateBUSD <= CONTRACT_BALANCE_STEP, "Maximum limit BUSD per deposit");
    BUSD.transferFrom(msg.sender, address(this), value);
    _totalInvested[1] += value;
} else revert("Plan number must be from 0 to 9");

           User storage user = users[msg.sender];
           if (user.referrer == address(0) && users[referrer].deposits.length > 0 && referrer != msg.sender) {
                    user.referrer = referrer;

    address upline = user.referrer;
                    for (uint256 i = 0; i < 3; i++) {
        if (upline != address(0)) {
                                    users[upline].referals[i] += 1;
            emit NewReferral(upline, msg.sender, i);
```

```
                                        upline = users[upline].referrer;
                                } else break;
    }
            }

            _createDeposit(msg.sender, plan, amount);

uint256 oldPercent;
uint256 newPercent;
uint256 stage = (_totalInvested[0] + (_totalInvested[1] / rateBUSD)) / CONTRACT_BALANCE_STEP;
if (stage < percents.length) {
    oldPercent = percents[times.length];
    newPercent = percents[stage];
} else {
    oldPercent = percents[percents.length-1] + (times.length - percents.length);
    newPercent = percents[percents.length-1] + (stage+1 - percents.length);
}

if (newPercent > oldPercent && newPercent <= 2000) {
    times.push(uint32(block.timestamp));
    emit ContractBonusUpd(oldPercent, newPercent);
}
  }

  function _createDeposit(address account, uint8 plan, uint256 amount) internal {
User storage user = users[account];

            uint256 price = plans[plan].price * amount;

uint8 flag;
if (plan >= 5) {
    flag = 1;
}

_refPayment(account, price, flag);

uint32 blocktimestamp = uint32(block.timestamp);

            if (user.deposits.length == 0) {
                      user.timestamps = [blocktimestamp, blocktimestamp, blocktimestamp];
                      _totalUsers += 1;
            }

            user.deposits.push(Deposit(plan, price, 0, 0, blocktimestamp));
totalSupply += amount;
_balances[plan][account] += amount;
            emit TransferSingle(msg.sender, address(0), account, plan, amount);
```

```
        _totalInvested[flag] += price;
                    emit NewDeposit(account, plan, amount);
          }

    function _refPayment(address account, uint256 value, uint8 busd) internal {
        User storage user = users[account];

        if (busd == 0) {
            projectAddress.transfer(value * PROJECT_FEE / PERCENTS_DIVIDER);
        } else {
            BUSD.transfer(projectAddress, value * PROJECT_FEE / PERCENTS_DIVIDER);
        }

                    if (user.referrer != address(0)) {
                              address upline = user.referrer;
                              for (uint256 i = 0; i < 3; i++) {
                                        if (upline != address(0)) {
                                                  uint256 refBonus = value * REFERRAL_PERCENTS[i] / PERCENTS_DIVIDER;
                                                  users[upline].refBonus[busd] += refBonus;
                     users[upline].totalRefBonus[i+(busd*3)] += refBonus;
                                                  emit RefBonus(upline, account, i, refBonus);
                                                  upline = users[upline].referrer;
                                        } else break;
                              }
                    }
          }

    function feed() public {
        User storage user = users[msg.sender];

        require(user.timestamps[2] + TIME_STEP <= block.timestamp, "Not hungry yet");

        uint256 holdPercent = _getHoldBonus(user.timestamps[1], user.timestamps[2], user.timestamps[2] + TIME_STEP);
        uint256 bonusPercent = holdPercent + _getContractBonus(user.timestamps[2], user.timestamps[2] + TIME_STEP);

        if (bonusPercent > 0) {
            for (uint256 i = 0; i < user.deposits.length; i++) {
                uint256 roi = user.deposits[i].value * plans[user.deposits[i].plan].ROI / PERCENTS_DIVIDER;
                if (user.deposits[i].withdrawn + user.deposits[i].bonus < roi && user.deposits[i].start < user.timestamps[2] + TIME_STEP) {

                    uint256 value;
                    if (user.deposits[i].start <= user.timestamps[2]) {
                        value = user.deposits[i].value * bonusPercent / PERCENTS_DIVIDER / TIME_STEP;
                    } else if (user.deposits[i].start <= user.timestamps[2] + TIME_STEP) {
                        value = user.deposits[i].value * (_getHoldBonus(user.timestamps[1], user.deposits[i].start, user.timestamps[2] + TIME_STEP) +
_getContractBonus(user.deposits[i].start, user.timestamps[2] + TIME_STEP)) / TIME_STEP / PERCENTS_DIVIDER;
```

```
                }

                user.deposits[i].bonus += value;
            }
        }
    }

    user.timestamps[2] = (block.timestamp - user.timestamps[2]) < TIME_STEP * 2 ? uint32(user.timestamps[2] + TIME_STEP) : uint32(block.timestamp);

    emit Feed(msg.sender);
}

function reinvest(uint256 value, uint8 plan) public {
    User storage user = users[msg.sender];

    require(plan < plans.length, "Plan number must be from 0 to 9");
    uint256 amount = value / plans[plan].price;
    value = amount * plans[plan].price;
    require(amount >= 1, "Amount must be more than 0");
    require(user.deposits.length + 1 <= DEPOSITS_MAX, "Deposits amount limit exceeded");

            (uint256 totalBNB, uint256 totalBUSD) = _withdraw();

    user.reserved[0] += totalBNB;
    user.reserved[1] += totalBUSD;

    uint8 flag;
    if (plan >= 5) {
        flag = 1;
    }

    require(user.reserved[flag] >= value, "Not enough profit available");
    user.reserved[flag] -= value;
    _totalInvested[flag] += value;

    _createDeposit(msg.sender, plan, amount);

    emit Reinvest(msg.sender, plan, amount);
}

function revive(uint256 depositId) public payable {
    User storage user = users[msg.sender];

    uint8 plan = user.deposits[depositId].plan;
    uint256 value = user.deposits[depositId].value * 90 / 100;
```

```
        if (user.deposits[depositId].withdrawn < user.deposits[depositId].value * plans[plan].ROI / PERCENTS_DIVIDER) {
                    (uint256 totalBNB, uint256 totalBUSD) = _withdraw();
            user.reserved[0] += totalBNB;
            user.reserved[1] += totalBUSD;
            require(user.deposits[depositId].withdrawn >= user.deposits[depositId].value * plans[plan].ROI / PERCENTS_DIVIDER, "Deposit is not expired
yet");
        }

        uint8 flag;
        if (plan < 5) {
            if (msg.value < value) {
                require(user.reserved[0] + msg.value >= value, "You must attach enough BNB to tx");
                user.reserved[0] -= value - msg.value;
            } else if (msg.value > value) {
                uint256 mod = msg.value - value;
                payable(msg.sender).transfer(mod);
            }
        } else {
            if (msg.value > 0) {
                payable(msg.sender).transfer(msg.value);
            }
            uint256 allowance = BUSD.allowance(msg.sender, address(this));
            uint256 balance = BUSD.balanceOf(msg.sender);
            allowance = balance >= allowance ? allowance : balance;
            if (allowance < value) {
                require(user.reserved[1] + allowance >= value, "You must approve enough BUSD first");
                user.reserved[1] -= value - allowance;
            }
            if (allowance > 0) {
                BUSD.transferFrom(msg.sender, address(this), allowance);
            }
            flag = 1;
        }

        _refPayment(msg.sender, value, flag);

        user.deposits[depositId].withdrawn = 0;
        user.deposits[depositId].start = uint32(block.timestamp);
        _totalInvested[flag] += value;

        emit Revived(msg.sender, depositId, plan, user.deposits[depositId].value / plans[plan].price);
    }

      function withdraw() public {
        (uint256 totalBNB, uint256 totalBUSD) = _withdraw();

                require(totalBNB > 0 || totalBUSD > 0, "User has no profit");
```

```
            users[msg.sender].timestamps[1] = uint32(block.timestamp);

    if (totalBNB > 0) {
        payable(msg.sender).transfer(totalBNB);
        marketingAddress.transfer(totalBNB * MARKETING_FEE / PERCENTS_DIVIDER);
    }

    if (totalBUSD > 0) {
        BUSD.transfer(msg.sender, totalBUSD);
        BUSD.transfer(marketingAddress, totalBUSD * MARKETING_FEE / PERCENTS_DIVIDER);
    }

    emit Withdrawn(msg.sender, totalBNB, totalBUSD);
      }

function _withdraw() internal returns(uint256 totalBNB, uint256 totalBUSD) {
    User storage user = users[msg.sender];

    totalBNB = user.reserved[0];
    totalBUSD = user.reserved[1];
    user.reserved[0] = 0;
    user.reserved[1] = 0;

    for (uint256 i = 0; i < user.deposits.length; i++) {
        uint8 plan = user.deposits[i].plan;
        uint256 roi = user.deposits[i].value * plans[plan].ROI / PERCENTS_DIVIDER;
        if (user.deposits[i].withdrawn < roi) {
            uint256 from = user.deposits[i].start > user.timestamps[0] ? user.deposits[i].start : user.timestamps[0];
            uint256 to = block.timestamp;
            uint256 profit = user.deposits[i].value * BASE_PERCENT / PERCENTS_DIVIDER * (to - from) / TIME_STEP + user.deposits[i].bonus;
            user.deposits[i].bonus = 0;
            from = 0;
            if (to >= user.timestamps[2] + TIME_STEP) {
                if (user.deposits[i].start <= user.timestamps[2]) {
                    from = user.timestamps[2];
                } else if (user.deposits[i].start < user.timestamps[2] + TIME_STEP) {
                    from = user.deposits[i].start;
                }
                to = user.timestamps[2] + TIME_STEP;
            } else {
                if (to >= user.deposits[i].start + TIME_STEP) {
                    from = user.timestamps[2];
                } else {
                    from = user.deposits[i].start;
                }
            }
            if (from > 0) {
```

```
                    profit += user.deposits[i].value * (_getHoldBonus(user.timestamps[1], from, to) + _getContractBonus(from, to)) / PERCENTS_DIVIDER /
TIME_STEP;
                }
                if (user.deposits[i].withdrawn + profit > roi) {
                    profit = roi - user.deposits[i].withdrawn;
                }
                user.deposits[i].withdrawn += profit;
                if (plan < 5) {
                    totalBNB += profit;
                } else {
                    totalBUSD += profit;
                }
            }
        }

        user.timestamps[0] = uint32(block.timestamp);
    }

    function _getCurrentContractBonus() internal view returns(uint256 contractBonus) {
        if ((_totalInvested[0] + (_totalInvested[1] / rateBUSD)) / CONTRACT_BALANCE_STEP < percents.length) {
            contractBonus = percents[times.length-1];
        } else {
            contractBonus = percents[percents.length-1] + (times.length - percents.length);
        }
    }

      function _getContractBonus(uint256 start, uint256 finish) internal view returns(uint256 contractBonus) {
        uint256 count = times.length-1;
        while (start < times[count]) {
            count--;
        }
        while (true) {
            uint256 percent = count < percents.length-1 ? percents[count] : percents[percents.length-1] + (count - percents.length-1);
            if (count < times.length-1 && finish > times[count+1]) {
                contractBonus += percent * (times[count+1] - start);
                start = times[count+1];
                count++;
            } else {
                uint256 end = count < times.length-1 ? times[count+1] : block.timestamp;
                finish = finish < end ? finish : end;
                contractBonus += percent * (finish - start);
                break;
            }
        }
    }

      function _getUserHoldBonus(address userAddress) internal view returns(uint256) {
        User memory user = users[userAddress];
                return (block.timestamp - user.timestamps[1]) / TIME_STEP * HOLD_STEP;
```

```
        }

    function _getHoldBonus(uint256 lastWithdrawal, uint256 start, uint256 finish) internal view returns(uint256) {
        if (block.timestamp - lastWithdrawal < TIME_STEP || finish < start) return 0;

        uint256 startPercent = HOLD_STEP * (start - lastWithdrawal) / TIME_STEP;
        uint256 finishPercent = HOLD_STEP * (finish - lastWithdrawal) / TIME_STEP;

        if (finishPercent > startPercent) {
            uint256 middle = (lastWithdrawal + (finishPercent / HOLD_STEP) * TIME_STEP);
            return startPercent * (middle - start) + finishPercent * (finish - middle);
        } else {
            return startPercent * (finish - start);
        }
         }

    function getUserInfo(address userAddress) public view returns(uint256 amountOfDeposits, uint256[2] memory invested, bool[] memory active,
uint256[2] memory available, uint256[2] memory remaining, uint256 holdBonus, uint256 userPercent, uint256 lastWithdrawal, uint256 nextFeed) {
        User memory user = users[userAddress];
        amountOfDeposits = user.deposits.length;

        active = new bool[](amountOfDeposits);

        available[0] = user.reserved[0];
        available[1] = user.reserved[1];

        for (uint256 i = 0; i < amountOfDeposits; i++) {
            uint256 flag;
            if (user.deposits[i].plan >= 5) {
                flag = 1;
            }
            invested[flag] += user.deposits[i].value;
            uint256 roi = user.deposits[i].value * plans[user.deposits[i].plan].ROI / PERCENTS_DIVIDER;
            if (user.deposits[i].withdrawn < roi) {
                uint256 from = user.deposits[i].start > user.timestamps[0] ? user.deposits[i].start : user.timestamps[0];
                uint256 to = block.timestamp;
                uint256 profit = user.deposits[i].value * BASE_PERCENT / PERCENTS_DIVIDER * (to - from) / TIME_STEP + user.deposits[i].bonus;
                from = 0;
                if (to >= user.timestamps[2] + TIME_STEP) {
                    if (user.deposits[i].start <= user.timestamps[2]) {
                        from = user.timestamps[2];
                    } else if (user.deposits[i].start < user.timestamps[2] + TIME_STEP) {
                        from = user.deposits[i].start;
                    }
                    to = user.timestamps[2] + TIME_STEP;
                } else {
                    if (to >= user.deposits[i].start + TIME_STEP) {
                        from = user.timestamps[2];
```

```
                    } else {
                        from = user.deposits[i].start;
                    }
                }
                if (from > 0) {
                    profit += user.deposits[i].value * (_getHoldBonus(user.timestamps[1], from, to) + _getContractBonus(from, to)) / PERCENTS_DIVIDER /
TIME_STEP;
                }
                if (user.deposits[i].withdrawn + profit > roi) {
                    profit = roi - user.deposits[i].withdrawn;
                } else {
                    active[i] = true;
                    remaining[flag] = roi - (user.deposits[i].withdrawn + profit);
                }
                available[flag] += profit;
            }
        }

        holdBonus = _getUserHoldBonus(userAddress);
        userPercent = BASE_PERCENT + _getCurrentContractBonus() + holdBonus;
        lastWithdrawal = user.timestamps[1];
        nextFeed = user.timestamps[2] + TIME_STEP > block.timestamp ? user.timestamps[2] + TIME_STEP - block.timestamp : 0;
    }

    function getDepositsInfo(address userAddress) public view returns(bool[] memory active, uint256[] memory startUnix, uint8[] memory plan, uint256[]
memory amount, uint256[] memory finishAmount, uint256[] memory remaining) {
        User memory user = users[userAddress];

        uint256 amountOfDeposits = user.deposits.length;

        active = new bool[](amountOfDeposits);
        startUnix = new uint256[](amountOfDeposits);
        plan = new uint8[](amountOfDeposits);
        amount = new uint256[](amountOfDeposits);
        finishAmount = new uint256[](amountOfDeposits);
        remaining = new uint256[](amountOfDeposits);

        for (uint256 i = 0; i < amountOfDeposits; i++) {
            startUnix[i] = user.deposits[i].start;
            plan[i] = user.deposits[i].plan;
            amount[i] = user.deposits[i].value;
            uint256 roi = user.deposits[i].value * plans[user.deposits[i].plan].ROI / PERCENTS_DIVIDER;
            finishAmount[i] = roi;
            if (user.deposits[i].withdrawn < roi) {
                uint256 from = startUnix[i] > user.timestamps[0] ? startUnix[i] : user.timestamps[0];
                uint256 to = block.timestamp;
                uint256 profit = user.deposits[i].value * BASE_PERCENT / PERCENTS_DIVIDER * (to - from) / TIME_STEP + user.deposits[i].bonus;
                from = 0;
```

```
                if (to >= user.timestamps[2] + TIME_STEP) {
                    if (user.deposits[i].start <= user.timestamps[2]) {
                        from = user.timestamps[2];
                    } else if (user.deposits[i].start < user.timestamps[2] + TIME_STEP) {
                        from = user.deposits[i].start;
                    }
                    to = user.timestamps[2] + TIME_STEP;
                } else {
                    if (to >= user.deposits[i].start + TIME_STEP) {
                        from = user.timestamps[2];
                    } else {
                        from = user.deposits[i].start;
                    }
                }
                if (from > 0) {
                    profit += user.deposits[i].value * (_getHoldBonus(user.timestamps[1], from, to) + _getContractBonus(from, to)) / PERCENTS_DIVIDER /
TIME_STEP;
                }
                if (user.deposits[i].withdrawn + profit < roi) {
                    active[i] = true;
                    remaining[i] = roi - (user.deposits[i].withdrawn + profit);
                }
            }
        }
    }

    function getRefInfo(address userAddress) public view returns(address referrer, uint24[3] memory referals, uint256[2] memory bonus, uint256[6]
memory totalBonuses) {
        User memory user = users[userAddress];
        referrer = user.referrer;
        referals = user.referals;
        bonus = user.refBonus;
        totalBonuses = user.totalRefBonus;
    }

    function getSiteInfo() public view returns(uint256 totalUsers, uint256[2] memory totalInvested, uint256[2] memory balance, uint256 contractBonus) {
        totalUsers = _totalUsers;
        totalInvested[0] = _totalInvested[0];
        totalInvested[1] = _totalInvested[1];
        balance[0] = address(this).balance;
        balance[1] = BUSD.balanceOf(address(this));
        contractBonus = _getCurrentContractBonus();
    }

    function balanceOf(address account, uint256 id) public view returns(uint256) {
        return _balances[id][account];
    }

    function balanceOfBatch(address[] memory accounts, uint256[] memory ids) public view returns(uint256[] memory) {
```

```
        uint256[] memory batchBalances = new uint256[](accounts.length);
        for (uint256 i = 0; i < accounts.length; ++i) {
            batchBalances[i] = balanceOf(accounts[i], ids[i]);
        }
        return batchBalances;
    }

      function uri(uint256 id) public view returns (string memory) {
        return bytes(_uri).length > 0 ? string(abi.encodePacked(_uri, id.toString(), ".json")) : "";
    }

    function supportsInterface(bytes4 interfaceId) public pure returns(bool) {
                return
                          interfaceId == type(IERC1155).interfaceId ||
                          interfaceId == type(IERC1155MetadataURI).interfaceId ||
                          interfaceId == type(IERC165).interfaceId;
         }

}
```