QuillAudits

Audit Report

September, 2020

Contents

Introduction

This Audit Report highlights the overall security of the Upswing Smart Contract. With this report, we have tried to ensure the reliability of their smart contract by complete assessment of their system's architecture and the smart contract codebase.

Auditing Approach and Methodologies applied

The Quillhash team has performed thorough testing of the project starting with analysing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third party smart contracts and libraries.

Our team then performed a formal line by line inspection of the Smart Contract to find any potential issue like race conditions, transaction-ordering dependence, timestamp dependence, and denial of service attacks.

In the Unit testing Phase, we coded/conducted Custom unit tests written for each function in the contract to verify that each function works as expected. In Automated Testing, We tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was tested in collaboration of our multiple team members and this included -

- Testing the functionality of the Smart Contract to determine proper logic has been followed throughout the process.
- Analysing the complexity of the code by thorough, manual review of the code, line-by-line.
- Deploying the code on testnet using multiple clients to run live tests
- Analysing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
- Checking whether all the libraries used in the code are on the latest version.
- Analysing the security of the on-chain data.

Audit Details

Project Name: Upswing
Website/Etherscan Code: Etherscan
Languages: Solidity (Smart contract), Javascript (Unit Testing)
Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Slither, Surya

Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient and working according to its specifications. The audit activities can be grouped in the following three categories:

Security
Identifying security related issues within each contract and the system of contracts.

Sound Architecture
Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.

Code Correctness and Quality
A full review of the contract source code. The primary areas of focus include:

- Correctness
- Sections of code with high complexity
- Readability
- Quantity and quality of test coverage

Summary of Upswing Smart Contract

QuillAudits conducted a security audit of a smart contract of Upswing. Upswing contract is used to create the ERC20 token, which is an *Upswing Token*, Smart contract contains basic functionalities of an ERC20 token.

Name: Upswing
Symbol: UPS

And some advanced features other than essential functions.

- Leverage
- Pressure
- Steam

Security

Every issue in this report was assigned a severity level from the following:

High severity issues
They will bring problems and should be fixed.

Medium severity issues
They could potentially bring problems add should eventually be fixed.

Low severity issues
They are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Number of issues per severity

	Low	Medium	High
Open	0	0	0
Closed	4	3	0

Manual Audit

For this section the code was tested/read line by line by our developers. We also used Remix IDE's JavaScript VM and Kovan networks to test the contract functionality.

Low Level Severity Issues

1. steamCont, never used in contracts
   Please remove the unused variables and functions from smart contracts

   Status: Fixed by Developer

2. Steam.sellPressure (Steam.sol#22) is never used in Steam (Steam.sol#7-74)
   Please remove the unused variables and functions from smart contracts

   Status: Fixed by Developer

3. Steam.burnerTokensToMint (Steam.sol#23) is never used in Steam (Steam.sol#7-74)
   Please remove the unused variables and functions from smart contracts

   Status: Fixed by Developer

4. myPressure function is a public function that calls an internal function amount pressure, you can change the internal function to public and use that instead of calling a function internally and there is no need to have mypressure() function.

   Please remove the commented code and unnecessary comments. Overall, code is working fine as it intended to; be based on business logic. But code is not optimized and does not follow the best practices and also has linting errors.

   Status: Not considered by Developers

Medium Severity Issues

1. UpSwing.onlyAllowed() (UpsSwing.sol#89-92) compares to a boolean constant:
   -require(bool,string)(allowed[_msgSender()] == true,onlyAllowed) (UpsSwing.sol#90)

```
contract A {
    function f(bool x) public {
        // ...
        if (x == true) { // bad!
           // ...
        }
        // ...
    }
}
```

Boolean constants can be used directly and do not need to be compared to true or false.

Recommendation

Remove the equality to the boolean constant.

Status: Fixed

2. Low level call in Address.sendValue(address,uint256) (UpsSwing.sol#66-72):
   - (success) = recipient.call.value(amount)() (UpsSwing.sol#70)

The use of low-level calls is error-prone. Low-level calls do not check for code existence or call success.

Recommendation

Avoid low-level calls. Check the call success. If the call is meant for a contract, check for code existence.

Status: Fixed

3. Library Address in upswing smart contract is unused, Please remove it or use as an address in upswing contract.

Status: Fixed

High severity issues

No high severity issues

Unit Testing

Test Suite

Contract: Upswing

- Should correctly initialise constructor of UpSwing token Contract (129ms)
- Should check the name of a token
- Should contain a symbol of a token
- Should check a decimal of a token
- Should check a balance of a token contract
- Should check a balance of an owner contract
- Should check a total supply of a contract
- Should check the leverage of a smart contract
- Should check a univ2 address
- Should check a univ2 liquidity
- Should check a univ2 ups burned

- Should check a univ2 total supply
- Should check a mySteam of accounts 0
- Should check a mySteam accounts 0
- Should Not be able to set univ2 address by not allowed only (61ms)
- Should be able to set univ2 address by allowed only (60ms)
- Should Not be able to set treasury address by not allowed only (55ms)
- Should be able to set treasury address by allowed only (52ms)
- Should Not be able to set leverage address by not allowed only (74ms)
- Should be able to set leverage address by allowed only (103ms)
- Should check approval by accounts 4 to accounts 2
- Should Approve accounts[4] to spend specific tokens of accounts[2]
- Should increase Approve accounts[4] to spend specific tokens of accounts[2]
- Should decrease Approve accounts[4] to spend specific tokens of accounts[2]
- Should Approve accounts[4] to spend specific tokens of accounts[2] (116ms)
- Should be able to transfer tokens by owner (49ms)
- Should check a balance of a token Receiver
- Should be able to transfer tokens by owner (56ms)
- Should check a balance of a token Receiver
- Should be able to transfer from accounts[4] to accounts[1]

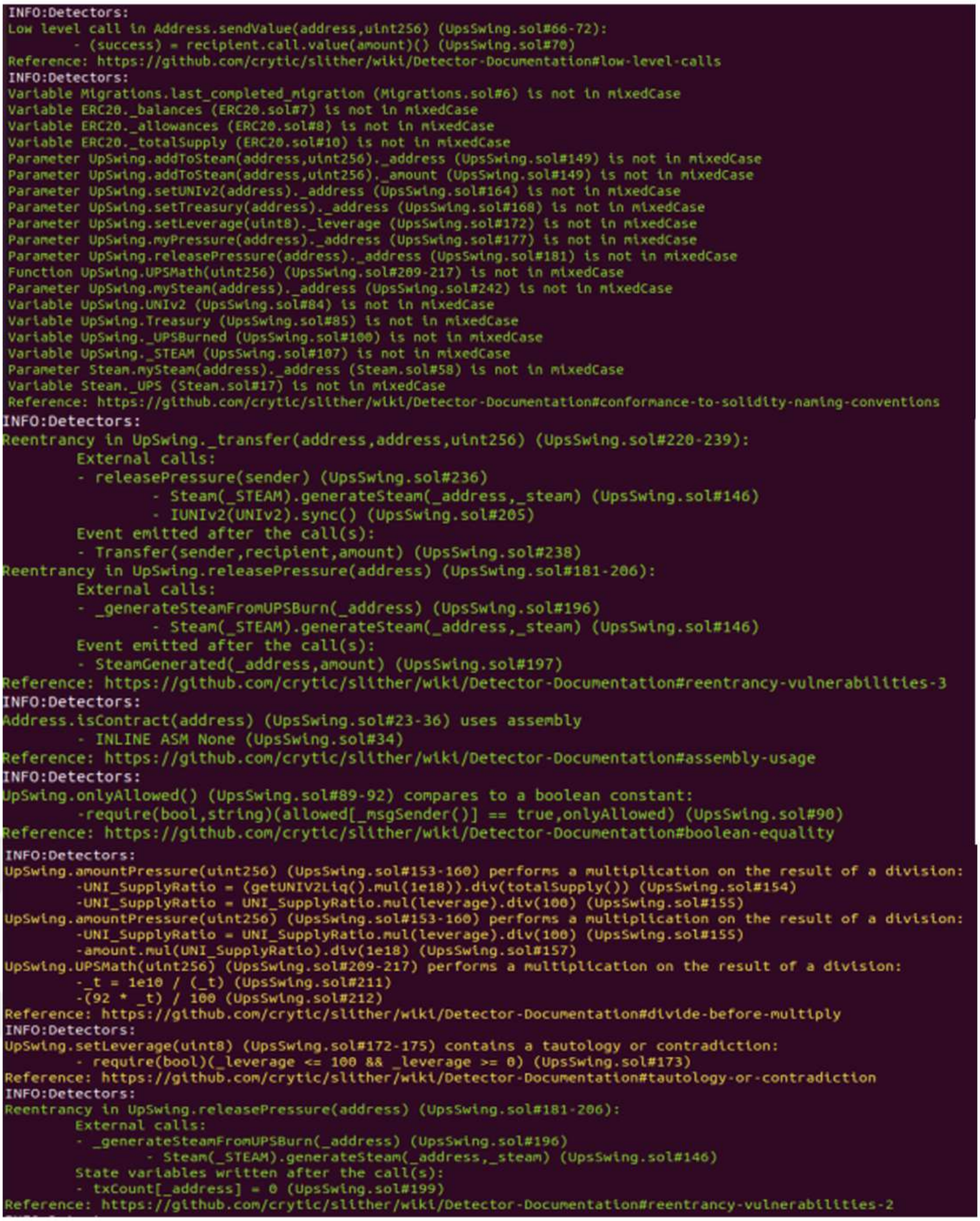
- Should check a balance of a univ2
- Should be able to transfer tokens by uniswap address (50ms)
- Should check a balance of a token Receiver
- Should check a my pressure univ2
- Should check a mySteam of accounts 0
- Should check a mySteam of univ2
- Should be able to transfer tokens by uniswap address

Final Result of Test

37 Passings (3s) Passed

0 Failed

Slither Tool Result



```
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (UpsSwing.sol#66-72):
        - (success) = recipient.call.value(amount)() (UpsSwing.sol#70)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Variable Migrations.last_completed_migration (Migrations.sol#6) is not in mixedCase
Variable ERC20._balances (ERC20.sol#7) is not in mixedCase
Variable ERC20._allowances (ERC20.sol#8) is not in mixedCase
Variable ERC20._totalSupply (ERC20.sol#10) is not in mixedCase
Parameter UpSwing.addToSteam(address,uint256)._address (UpsSwing.sol#149) is not in mixedCase
Parameter UpSwing.addToSteam(address,uint256)._amount (UpsSwing.sol#149) is not in mixedCase
Parameter UpSwing.setUNIv2(address)._address (UpsSwing.sol#164) is not in mixedCase
Parameter UpSwing.setTreasury(address)._address (UpsSwing.sol#168) is not in mixedCase
Parameter UpSwing.setLeverage(uint8)._leverage (UpsSwing.sol#172) is not in mixedCase
Parameter UpSwing.myPressure(address)._address (UpsSwing.sol#177) is not in mixedCase
Parameter UpSwing.releasePressure(address)._address (UpsSwing.sol#181) is not in mixedCase
Function UpSwing.UPSMath(uint256) (UpsSwing.sol#209-217) is not in mixedCase
Parameter UpSwing.mySteam(address)._address (UpsSwing.sol#242) is not in mixedCase
Variable UpSwing.UNIv2 (UpsSwing.sol#84) is not in mixedCase
Variable UpSwing.Treasury (UpsSwing.sol#85) is not in mixedCase
Variable UpSwing._UPSBurned (UpsSwing.sol#100) is not in mixedCase
Variable UpSwing._STEAM (UpsSwing.sol#107) is not in mixedCase
Parameter Steam.mySteam(address)._address (Steam.sol#58) is not in mixedCase
Variable Steam._UPS (Steam.sol#17) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Reentrancy in UpSwing._transfer(address,address,uint256) (UpsSwing.sol#220-239):
        External calls:
        - releasePressure(sender) (UpsSwing.sol#236)
                - Steam(_STEAM).generateSteam(_address,_steam) (UpsSwing.sol#146)
                - IUNIv2(UNIv2).sync() (UpsSwing.sol#205)
        Event emitted after the call(s):
        - Transfer(sender,recipient,amount) (UpsSwing.sol#238)
Reentrancy in UpSwing.releasePressure(address) (UpsSwing.sol#181-206):
        External calls:
        - _generateSteamFromUPSBurn(_address) (UpsSwing.sol#196)
                - Steam(_STEAM).generateSteam(_address,_steam) (UpsSwing.sol#146)
        Event emitted after the call(s):
        - SteamGenerated(_address,amount) (UpsSwing.sol#197)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Address.isContract(address) (UpsSwing.sol#23-36) uses assembly
        - INLINE ASM None (UpsSwing.sol#34)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
UpSwing.onlyAllowed() (UpsSwing.sol#89-92) compares to a boolean constant:
        -require(bool,string)(allowed[_msgSender()] == true,onlyAllowed) (UpsSwing.sol#90)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
INFO:Detectors:
UpSwing.amountPressure(uint256) (UpsSwing.sol#153-160) performs a multiplication on the result of a division:
        -UNI_SupplyRatio = (getUNIV2Liq().mul(1e18)).div(totalSupply()) (UpsSwing.sol#154)
        -UNI_SupplyRatio = UNI_SupplyRatio.mul(leverage).div(100) (UpsSwing.sol#155)
UpSwing.amountPressure(uint256) (UpsSwing.sol#153-160) performs a multiplication on the result of a division:
        -UNI_SupplyRatio = UNI_SupplyRatio.mul(leverage).div(100) (UpsSwing.sol#155)
        -amount.mul(UNI_SupplyRatio).div(1e18) (UpsSwing.sol#157)
UpSwing.UPSMath(uint256) (UpsSwing.sol#209-217) performs a multiplication on the result of a division:
        -_t = 1e10 / (_t) (UpsSwing.sol#211)
        -(92 * _t) / 100 (UpsSwing.sol#212)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
UpSwing.setLeverage(uint8) (UpsSwing.sol#172-175) contains a tautology or contradiction:
        - require(bool)(_leverage <= 100 && _leverage >= 0) (UpsSwing.sol#173)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#tautology-or-contradiction
INFO:Detectors:
Reentrancy in UpSwing.releasePressure(address) (UpsSwing.sol#181-206):
        External calls:
        - _generateSteamFromUPSBurn(_address) (UpsSwing.sol#196)
                - Steam(_STEAM).generateSteam(_address,_steam) (UpsSwing.sol#146)
        State variables written after the call(s):
        - txCount[_address] = 0 (UpsSwing.sol#199)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
```

Implementation Recommendations

- setCompleted(uint256) should be declared external:
  - Migrations.setCompleted(uint256) (Migrations.sol#16-18)
- transfer(address,uint256) should be declared external:
  - ERC20.transfer(address,uint256) (ERC20.sol#33-36)
- allowance(address,address) should be declared external:
  - ERC20.allowance(address,address) (ERC20.sol#41-43)
- approve(address,uint256) should be declared external:
  - ERC20.approve(address,uint256) (ERC20.sol#52-55)
- transferFrom(address,address,uint256) should be declared external:
  - ERC20.transferFrom(address,address,uint256) (ERC20.sol#69-73)
- increaseAllowance(address,uint256) should be declared external:
  - ERC20.increaseAllowance(address,uint256) (ERC20.sol#87-90)
- decreaseAllowance(address,uint256) should be declared external:
  - ERC20.decreaseAllowance(address,uint256) (ERC20.sol#106-109)
- setUNIv2(address) should be declared external:
  - UpSwing.setUNIv2(address) (UpsSwing.sol#164-166)
- setTreasury(address) should be declared external:
  - UpSwing.setTreasury(address) (UpsSwing.sol#168-170)
- setLeverage(uint8) should be declared external:
  - UpSwing.setLeverage(uint8) (UpsSwing.sol#172-175)
- mySteam(address) should be declared external:
  - UpSwing.mySteam(address) (UpsSwing.sol#242-244)
- getUNIV2Address() should be declared external:
  - UpSwing.getUNIV2Address() (UpsSwing.sol#246-248)
- getUPSTotalSupply() should be declared external:
  - UpSwing.getUPSTotalSupply() (UpsSwing.sol#254-256)

- getUPSBurned() should be declared external:
  - UpSwing.getUPSBurned() (UpsSwing.sol#258-260)

- name() should be declared external:
  - UpSwing.name() (UpsSwing.sol#262-264)

- symbol() should be declared external:
  - UpSwing.symbol() (UpsSwing.sol#266-268)

- decimals() should be declared external:
  - UpSwing.decimals() (UpsSwing.sol#270-272)

- name() should be declared external:
  - Steam.name() (Steam.sol#42-44)

- symbol() should be declared external:
  - Steam.symbol() (Steam.sol#46-48)

- decimals() should be declared external:
  - Steam.decimals() (Steam.sol#50-52)

- totalSupply() should be declared external:
  - Steam.totalSupply() (Steam.sol#54-56)

- mySteam(address) should be declared external:
  - Steam.mySteam(address) (Steam.sol#58-60)

- getSteamTotalSupply() should be declared external:
  - Steam.getSteamTotalSupply() (Steam.sol#62-64)

- getSteamMaxSupply() should be declared external:
  - Steam.getSteamMaxSupply() (Steam.sol#66-68)

- getSteamMinted() should be declared external:
  - Steam.getSteamMinted() (Steam.sol#70-72)

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the Upswing contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Summary

Use case of the smart contract is very well designed and Implemented. Overall, the code is written and demonstrates effective use of abstraction, separation of concerns, and modularity. The Upswing development team demonstrated high technical capabilities, both in the design of the architecture and in the implementation.
All the bugs, suggestions and recommendations has been considered by the Upswing team and some of the issues will be handled on their own as those issues or calls have to be handle by the owner.

QuillAudits

448-A EnKay Square, Opposite Cyber Hub,
Gurugram, Harayana, India - 122016

audits.quillhash.com

hello@quillhash.com