QuillAudits

Audit Report
June, 2023

For

Stronghands Protocol

Table of Content

Executive Summary

Project Name	StrongHands 3D
Overview	The design architecture of the contract allows for the creation of 3D tokens when there is a deposit made into the contract and reduces the supply of the token when there is a withdrawal, in accordance with the bonding curve algorithm. Every deposit and withdrawal action incurs a 10% dividend fee that gets shared among StrongHolds 3D token holders. Token holders can compound their earnings to yield more 3D tokens to themselves and could likewise harvest them, withdrawing from the protocol. There is a reward system also for referrals.
Timeline	12 June 2023 - 16 June 2023
Method	Manual Review, Functional Testing, Automated Testing etc.
Language	Solidity
Blockchain	PulseChain
Scope of Audit	The scope of this audit was to analyze the StrongHands 3D codebase for quality, security, and correctness. *https://scan.pulsechain.com/address/0x100d6623274B091e51DAb7cBe54521A2F6e7DB1c/contracts#address-tabs* Branch: Main
Contracts in Scope	Hourglass.sol

High
Medium
Low
Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	3
Resolved Issues	0	0	0	0

Types of Severities

High

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low

Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Manual Testing

A. Contract - Hourglass.sol

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

A.1 Some Non-changing Variables Could be Declared as Constant

```
string public name = "Stronghands FTM3D";
string public symbol = "FTM3D";
```

Description

The name and tinker of the 3D token are variables that have no setter function to update them later in the function. Adding the constant attribute to these variables would prevent the compiler from reserving storage slots for the variables.

Remediation

It is recommended to add the constant attribute to the variables.

Status

Acknowledged

Reference

A.2 Non-conformity to Solidity Naming Convention

Description

Solidity naming convention aids to improve the readability and conveying of information in contracts. This details how variable names should be written in order to foster easy comprehension. In the contract, there were some variables which are constant, but the variable names are not capitalized. There are exceptions to the naming convention rules.

Remediation

Redesign the variable names to conform to solidity naming convention.

Status

Acknowledged

Reference 1 | *Reference 2*

A.3 Some Public Functions could be Declared with External Visibility to Save Gas

Description

Whenever a function is not called internally in a contract, it is recommended to declare this function with external visibility instead of public in order to save gas. This small optimization trick could save some gas when some functions like view functions are called.

Remediation

Consider changing the public visibility of functions not called within the contract to external.

Status

Acknowledged

B. General Recommendation

Although the contract uses an older solidity compiler version, it employs the use of Safemath library to aid in most of its mathematical computation in the contract to avoid arithmetic errors.

Functional Tests

Some of the tests performed are mentioned below

- Should get the token name and symbols and other view functions
- Should successfully deposit any amount for different users with 10% dividend fee
- Should successfully earn referral bonus when referring new depositors
- Should reverts when users without StrongHands 3D tokens attempt to transfer tokens
- Should successfully transfer tokens to other addresses and the recipient amount of tokens increases
- Should revert when users with no accumulated dividends calls the compound function
- Should successfully compound users dividends and their referral bonus while also updating the users and contracts statistics
- Should revert when non token holders attempts to call the withdraw function to liquify tokens to base
- Should successfully call the withdraw function to liquify tokens to base and reduce the total supply
- Should revert when users who do not have enough dividend calls the harvest function
- Should successfully allow users to harvest accumulated earnings from the protocol
- Should allow users to exit the protocol | sell and harvest accumulated earnings
- Should get the statistics of all deposits, withdrawals, compound, and harvest associated to users and statistics of all the totals.

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

```
INFO:Detectors:
Hourglass.withdraw(uint256) (src/Hourglass.sol#142-172) performs a multiplication on the result of a division:
        - _dividends = SafeMath.div(_base,dividendFee_) (src/Hourglass.sol#149)
        - profitPerShare_ = SafeMath.add(profitPerShare_,(_dividends * magnitude) / tokenSupply_) (src/Hourglass.sol#161)
Hourglass.purchaseTokens(uint256,address) (src/Hourglass.sol#393-449) performs a multiplication on the result of a division:
        - _fee = _fee - (_fee - (_amountOfTokens * (_dividends * magnitude / (tokenSupply_)))) (src/Hourglass.sol#429)
Hourglass.tokensToBase_(uint256) (src/Hourglass.sol#476-493) performs a multiplication on the result of a division:
        - _baseReceived = (SafeMath.sub((((tokenPriceInitial_ + (tokenPriceIncremental_ * (_tokenSupply / 1e18))) - tokenPriceIncremental_) * (tokens_ - 1e18)),(tokenPriceIncremental_ * ((token
s_ ** 2 - tokens_) / 1e18)) / 2) / 1e18) (src/Hourglass.sol#479-491)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
SafeMath.mul(uint256,uint256) (src/Hourglass.sol#5-10) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version0.6.12 (src/Hourglass.sol#2) allows old versions
solc-0.6.12 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Event HourglassonTokenPurchase(address,uint256,uint256,address) (src/Hourglass.sol#62) is not in CapWords
Event HourglassonTokenSell(address,uint256,uint256) (src/Hourglass.sol#63) is not in CapWords
Event HourglassonReinvestment(address,uint256,uint256) (src/Hourglass.sol#64) is not in CapWords
Event HourglassonWithdraw(address,uint256) (src/Hourglass.sol#65) is not in CapWords
Parameter Hourglass.deposit(address)._referredBy (src/Hourglass.sol#121) is not in mixedCase
Parameter Hourglass.withdraw(uint256)._amountOfTokens (src/Hourglass.sol#142) is not in mixedCase
Parameter Hourglass.transfer(address,uint256)._toAddress (src/Hourglass.sol#228) is not in mixedCase
Parameter Hourglass.transfer(address,uint256)._amountOfTokens (src/Hourglass.sol#228) is not in mixedCase
Parameter Hourglass.referralsOf(address)._user (src/Hourglass.sol#270) is not in mixedCase
Parameter Hourglass.totalRefEarningsOf(address)._user (src/Hourglass.sol#278) is not in mixedCase
Parameter Hourglass.getAmountStatsOf(address)._user (src/Hourglass.sol#286) is not in mixedCase
Parameter Hourglass.getRepeatStatsOf(address)._user (src/Hourglass.sol#295) is not in mixedCase
Parameter Hourglass.myDividends(bool)._includeReferralBonus (src/Hourglass.sol#322) is not in mixedCase
Parameter Hourglass.balanceOf(address)._customerAddress (src/Hourglass.sol#328) is not in mixedCase
Parameter Hourglass.dividendsOf(address,bool)._customerAddress (src/Hourglass.sol#333) is not in mixedCase
Parameter Hourglass.dividendsOf(address,bool)._includeReferralBonus (src/Hourglass.sol#333) is not in mixedCase
Parameter Hourglass.calculateTokensReceived(uint256)._baseToSpend (src/Hourglass.sol#370) is not in mixedCase
Parameter Hourglass.calculateBaseReceived(uint256)._tokensToSell (src/Hourglass.sol#379) is not in mixedCase
Parameter Hourglass.purchaseTokens(uint256,address)._incomingBase (src/Hourglass.sol#393) is not in mixedCase
Parameter Hourglass.purchaseTokens(uint256,address)._referredBy (src/Hourglass.sol#393) is not in mixedCase
Parameter Hourglass.baseToTokens_(uint256)._base (src/Hourglass.sol#452) is not in mixedCase
Parameter Hourglass.tokensToBase_(uint256)._tokens (src/Hourglass.sol#476) is not in mixedCase
Constant Hourglass.dividendFee_ (src/Hourglass.sol#78) is not in UPPER_CASE_WITH_UNDERSCORES
Constant Hourglass.tokenPriceInitial_ (src/Hourglass.sol#80) is not in UPPER_CASE_WITH_UNDERSCORES
Constant Hourglass.tokenPriceIncremental_ (src/Hourglass.sol#81) is not in UPPER_CASE_WITH_UNDERSCORES
Constant Hourglass.magnitude (src/Hourglass.sol#82) is not in UPPER_CASE_WITH_UNDERSCORES
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

```
INFO:Detectors:
Reentrancy in Hourglass.harvest() (src/Hourglass.sol#202-225):
        External calls:
        - _customerAddress.transfer(_dividends) (src/Hourglass.sol#215)
        State variables written after the call(s):
        - totalHarvested_ += _dividends (src/Hourglass.sol#221)
        - userData_[msg.sender].harvested += _dividends (src/Hourglass.sol#218)
        - userData_[msg.sender].xHarvested += 1 (src/Hourglass.sol#219)
        Event emitted after the call(s):
        - onWithdraw(_customerAddress,_dividends) (src/Hourglass.sol#224)
Reentrancy in Hourglass.transfer(address,uint256) (src/Hourglass.sol#228-251):
        External calls:
        - harvest() (src/Hourglass.sol#238)
                - _customerAddress.transfer(_dividends) (src/Hourglass.sol#215)
        State variables written after the call(s):
        - payoutsTo_[_customerAddress] -= int256(profitPerShare_ * _amountOfTokens) (src/Hourglass.sol#245)
        - payoutsTo_[_toAddress] += int256(profitPerShare_ * _amountOfTokens) (src/Hourglass.sol#246)
        - tokenBalanceLedger_[_customerAddress] = SafeMath.sub(tokenBalanceLedger_[_customerAddress],_amountOfTokens) (src/Hourglass.sol#241)
        - tokenBalanceLedger_[_toAddress] = SafeMath.add(tokenBalanceLedger_[_toAddress],_amountOfTokens) (src/Hourglass.sol#242)
        Event emitted after the call(s):
        - Transfer(_customerAddress,_toAddress,_amountOfTokens) (src/Hourglass.sol#249)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4
INFO:Detectors:
Variable Hourglass.tokensToBase_(uint256)._tokenSupply (src/Hourglass.sol#478) is too similar to Hourglass.tokenSupply_ (src/Hourglass.sol#99)
Variable Hourglass.contractStats()._totalCompounded (src/Hourglass.sol#282) is too similar to Hourglass.totalCompounded_ (src/Hourglass.sol#104)
Variable Hourglass.contractStats()._totalHarvested (src/Hourglass.sol#282) is too similar to Hourglass.totalHarvested_ (src/Hourglass.sol#105)
Variable Hourglass.contractStats()._totalWithdrawn (src/Hourglass.sol#282) is too similar to Hourglass.totalWithdrawn_ (src/Hourglass.sol#103)
Variable Hourglass.baseToTokens_(uint256)._tokenPriceInitial (src/Hourglass.sol#453) is too similar to Hourglass.tokenPriceInitial_ (src/Hourglass.sol#80)
Variable Hourglass.contractStats()._totalDeposited (src/Hourglass.sol#282) is too similar to Hourglass.totalDeposited_ (src/Hourglass.sol#102)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar
INFO:Detectors:
Hourglass.slitherConstructorConstantVariables() (src/Hourglass.sol#29-505) uses literals with too many digits:
        - tokenPriceInitial_ = 100000000000 (src/Hourglass.sol#80)
Hourglass.slitherConstructorConstantVariables() (src/Hourglass.sol#29-505) uses literals with too many digits:
        - tokenPriceIncremental_ = 10000000000 (src/Hourglass.sol#81)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
Hourglass.name (src/Hourglass.sol#74) should be constant
Hourglass.symbol (src/Hourglass.sol#75) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
```

Summary

In this report, we have considered the security of StrongHands 3D. We performed our audit according to the procedure described above.

Some issues of informational severity were found. Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the StrongHands 3D Platform. This audit does not provide a security or correctness guarantee of the audited smart contracts.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the StrongHands 3D Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

850+
Audits Completed

$16B
Secured

800K
Lines of Code Audited

Follow Our Journey

Audit Report
June, 2023

For

Stronghands Protocol

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com