QuillAudits

Audit Report
July, 2022

For

STRONGHANDS
FINANCE

Table of Content

Executive Summary 01

Checked Vulnerabilities 03

Techniques and Methods 04

Manual Testing 05

A. Contract - NFTSales.sol 05

High Severity Issues 05

Medium Severity Issues 05

Low Severity Issues 05

A.1 | Lack of event emissions 05

Informational Issues 05

Functional Testing 06

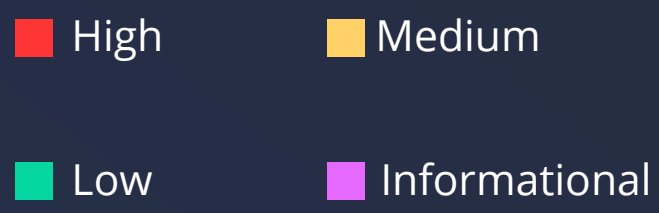
Automated Testing 07

Closing Summary 10

About QuillAudits 11

Executive Summary

Project Name	Stronghands
Timeline	18th July 2020 to 25 July 2022
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyse Stronghands codebase for quality, security, and correctness.
Git Repo link	*https://github.com/DeFi-Magic/stronghands_nft*
Git Branch	Main
Commit Hash	c34646a1d41ffad8e4c5b6907f873425c62e50c5



	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	1	0
Partially Resolved Issues	0	0	0	0
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
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility leve

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 - NFTSales.sol

High Severity Issues

No issues were found

Medium Severity Issues

No issues were found

Low Severity Issues

A.1 Possible to mint more than the set maximum supply of the tokens

Description

The following functions do not emit relevant events after executing sensitive actions:

- setMaxPerMint() changes the value of the maxPerMint variable
- setMaxPerWallet() changes the value of the maxPerWallet variable
- setPrice() changes the value of the price variable
- setSaleActive() changes the statue of the saleActive
- setTreasury() changes the address of the treasury
- updateConfig() updates the values of saleActive, maxSupply, price, maxPerMint, maxPerWallet and _tokenBaseURI

Remediation

In order to facilitate tracking and notify off-chain clients following the contracts' activity, we recommend emitting an event to log the update of the above variables for the above-mentioned functions.

Status

Acknowledged

Informational Issues

No issues were found

Functional Testing

Some of the tests performed are mentioned below

- updateConfig() should be called only by the owner (86ms)
- airdrop() should be called only by the owner (66ms)
- saleActive() should be FALSE as default
- saleActive() should be called only by the owner (39ms)
- reverts when sale is not active (53ms)
- reverts when an incorrect amount of ETH sent
- reverts when max supply reached (74ms)
- reverts when max mint per account reached (54ms)
- reverts when amount exceeds max per mint
- reverts when non owner sends ETH to the contract
- setMaxPerMint() should be called only by the owner
- setMaxPerWallet() should be called only by the owner
- setPrice() should be called only by the owner
- setSaleActive() should be called only by the owner
- setTreasury() should be called only by the owner
- withdrawRevenue() should be called only by the owner || treasury || proxyToApproved
- batchMint() should be called only by the proxy || treasury || proxyToApproved
- totalSupply() should return a correct value
- totalSupply() should be changed after buy or batchbuy

Automated Tests

```
INFO:Detectors:
NFTSales.withdrawRevenue() (NFTSales.sol#134-146) uses a dangerous strict equality:
        - amount == 0 (NFTSales.sol#142)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
ERC721Base._airdrop(address[],uint256[]).x (ERC721Base.sol#58) is a local variable never initialized
ERC721Base._batchMint(address,uint256).i (ERC721Base.sol#69) is a local variable never initialized
ERC721Min.tokenOfOwnerByIndex(address,uint256).i (ERC721Min.sol#541) is a local variable never initialized
ERC721Min.tokenOfOwnerByIndex(address,uint256).foundCount (ERC721Min.sol#540) is a local variable never initialized
ERC721Base._airdrop(address[],uint256[]).y (ERC721Base.sol#59) is a local variable never initialized
ERC721Base.isOwnerOf(address,uint32[]).i (ERC721Base.sol#124) is a local variable never initialized
ERC721Min.batchTransferFrom(address,address,uint256[]).i (ERC721Min.sol#266) is a local variable never initialized
ERC721Min.batchSafeTransferFrom(address,address,uint256[],bytes).i (ERC721Min.sol#277) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
ERC721Min._checkOnERC721Received(address,address,uint256,bytes) (ERC721Min.sol#487-517) ignores return value by IERC721Receiver(to).onERC721Received(_msgSe
nder(),from,tokenId,data) (ERC721Min.sol#494-513)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

```
INFO:Detectors:
Variable ERC721Base._contractURI (ERC721Base.sol#11) is not in mixedCase
Variable ERC721Base._tokenBaseURI (ERC721Base.sol#12) is not in mixedCase
Variable ERC721Min._owners (ERC721Min.sol#29) is not in mixedCase
Variable ERC721Min._balances (ERC721Min.sol#32) is not in mixedCase
Parameter NFTSales.setTreasury(address)._treasury (NFTSales.sol#109) is not in mixedCase
Parameter NFTSales.updateConfig(bool,uint256,uint256,uint256,uint256,string)._saleActive (NFTSales.sol#118) is not in mixedCase
Parameter NFTSales.updateConfig(bool,uint256,uint256,uint256,uint256,string)._maxSupply (NFTSales.sol#119) is not in mixedCase
Parameter NFTSales.updateConfig(bool,uint256,uint256,uint256,uint256,string)._price (NFTSales.sol#120) is not in mixedCase
Parameter NFTSales.updateConfig(bool,uint256,uint256,uint256,uint256,string)._maxPerMint (NFTSales.sol#121) is not in mixedCase
Parameter NFTSales.updateConfig(bool,uint256,uint256,uint256,uint256,string)._maxPerWallet (NFTSales.sol#122) is not in mixedCase
Parameter NFTSales.isApprovedForAll(address,address)._owner (NFTSales.sol#148) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (@openzeppelin/contracts/access/Ownable.sol#61-63)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (@openzeppelin/contracts/access/Ownable.sol#69-72)
contractURI() should be declared external:
        - ERC721Base.contractURI() (ERC721Base.sol#115-117)
isOwnerOf(address,uint32[]) should be declared external:
        - ERC721Base.isOwnerOf(address,uint32[]) (ERC721Base.sol#119-128)
name() should be declared external:
        - ERC721Min.name() (ERC721Min.sol#138-140)
symbol() should be declared external:
        - ERC721Min.symbol() (ERC721Min.sol#145-147)
approve(address,uint256) should be declared external:
        - ERC721Min.approve(address,uint256) (ERC721Min.sol#152-164)
setApprovalForAll(address,bool) should be declared external:
        - ERC721Min.setApprovalForAll(address,bool) (ERC721Min.sol#189-198)
safeTransferFrom(address,address,uint256) should be declared external:
        - ERC721Min.safeTransferFrom(address,address,uint256) (ERC721Min.sol#234-240)
batchTransferFrom(address,address,uint256[]) should be declared external:
        - ERC721Min.batchTransferFrom(address,address,uint256[]) (ERC721Min.sol#261-269)
batchSafeTransferFrom(address,address,uint256[],bytes) should be declared external:
        - ERC721Min.batchSafeTransferFrom(address,address,uint256[],bytes) (ERC721Min.sol#271-280)
transfer(address,uint256) should be declared external:
        - ERC721Min.transfer(address,uint256) (ERC721Min.sol#426-433)
tokenOfOwnerByIndex(address,uint256) should be declared external:
        - ERC721Min.tokenOfOwnerByIndex(address,uint256) (ERC721Min.sol#539-549)
```

```
INFO:Detectors:
Pragma version^0.8.0 (@openzeppelin/contracts/access/Ownable.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin/contracts/security/ReentrancyGuard.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.
12/0.7.6
Pragma version^0.8.0 (@openzeppelin/contracts/token/ERC1155/IERC1155.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12
/0.7.6
Pragma version^0.8.0 (@openzeppelin/contracts/token/ERC721/IERC721.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0
.7.6
Pragma version^0.8.0 (@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol#4) necessitates a version too recent to be trusted. Consider deploying with
0.6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol#4) necessitates a version too recent to be trusted. Consider depl
oying with 0.6.12/0.7.6
Pragma version^0.8.1 (@openzeppelin/contracts/utils/Address.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin/contracts/utils/Context.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin/contracts/utils/Strings.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin/contracts/utils/introspection/ERC165.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.
6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin/contracts/utils/introspection/IERC165.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0
.6.12/0.7.6
Pragma version0.8.1 (ERC721Base.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.1 (ERC721Min.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.1 (NFTSales.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.1 (Proxyable.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.1 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (@openzeppelin/contracts/utils/Address.sol#60-65):
        - (success) = recipient.call{value: amount}() (@openzeppelin/contracts/utils/Address.sol#63)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (@openzeppelin/contracts/utils/Address.sol#128-139):
        - (success,returndata) = target.call{value: value}(data) (@openzeppelin/contracts/utils/Address.sol#137)
Low level call in Address.functionStaticCall(address,bytes,string) (@openzeppelin/contracts/utils/Address.sol#157-166):
        - (success,returndata) = target.staticcall(data) (@openzeppelin/contracts/utils/Address.sol#164)
Low level call in Address.functionDelegateCall(address,bytes,string) (@openzeppelin/contracts/utils/Address.sol#184-193):
        - (success,returndata) = target.delegatecall(data) (@openzeppelin/contracts/utils/Address.sol#191)
Low level call in NFTSales.withdrawRevenue() (NFTSales.sol#134-146):
        - (success) = treasury.call{value: amount}() (NFTSales.sol#143)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

```
INFO:Detectors:
Address.functionCall(address,bytes) (@openzeppelin/contracts/utils/Address.sol#85-87) is never used and should be removed
Address.functionCall(address,bytes,string) (@openzeppelin/contracts/utils/Address.sol#95-101) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (@openzeppelin/contracts/utils/Address.sol#114-120) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (@openzeppelin/contracts/utils/Address.sol#128-139) is never used and should be removed
Address.functionDelegateCall(address,bytes) (@openzeppelin/contracts/utils/Address.sol#174-176) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (@openzeppelin/contracts/utils/Address.sol#184-193) is never used and should be removed
Address.functionStaticCall(address,bytes) (@openzeppelin/contracts/utils/Address.sol#147-149) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (@openzeppelin/contracts/utils/Address.sol#157-166) is never used and should be removed
Address.sendValue(address,uint256) (@openzeppelin/contracts/utils/Address.sol#60-65) is never used and should be removed
Address.verifyCallResult(bool,bytes,string) (@openzeppelin/contracts/utils/Address.sol#201-221) is never used and should be removed
Context._msgData() (@openzeppelin/contracts/utils/Context.sol#21-23) is never used and should be removed
ERC721Min._baseURI() (ERC721Min.sol#84-86) is never used and should be removed
ERC721Min._burn(uint256) (ERC721Min.sol#406-417) is never used and should be removed
ERC721Min._exists(uint256) (ERC721Min.sol#321-323) is never used and should be removed
ERC721Min._isApprovedOrOwner(address,uint256) (ERC721Min.sol#332-343) is never used and should be removed
ERC721Min._mintToSender() (ERC721Min.sol#389-394) is never used and should be removed
ERC721Min._safeMint(address) (ERC721Min.sol#354-356) is never used and should be removed
ERC721Min._safeMint(address,bytes) (ERC721Min.sol#362-368) is never used and should be removed
Strings.toHexString(address) (@openzeppelin/contracts/utils/Strings.sol#72-74) is never used and should be removed
Strings.toHexString(uint256) (@openzeppelin/contracts/utils/Strings.sol#41-52) is never used and should be removed
Strings.toHexString(uint256,uint256) (@openzeppelin/contracts/utils/Strings.sol#57-67) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

```
INFO:Detectors:
Reentrancy in NFTSales.withdrawRevenue() (NFTSales.sol#134-146):
        External calls:
        - (success) = treasury.call{value: amount}() (NFTSales.sol#143)
        Event emitted after the call(s):
        - WithdrawRevenue(_msgSender(),amount) (NFTSales.sol#145)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Address.verifyCallResult(bool,bytes,string) (@openzeppelin/contracts/utils/Address.sol#201-221) uses assembly
        - INLINE ASM (@openzeppelin/contracts/utils/Address.sol#213-216)
ERC721Min._checkOnERC721Received(address,address,uint256,bytes) (ERC721Min.sol#487-517) uses assembly
        - INLINE ASM (ERC721Min.sol#509-511)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Different versions of Solidity is used:
        - Version used: ['0.8.1', '^0.8.0', '^0.8.1']
        - ^0.8.0 (@openzeppelin/contracts/access/Ownable.sol#4)
        - ^0.8.0 (@openzeppelin/contracts/security/ReentrancyGuard.sol#4)
        - ^0.8.0 (@openzeppelin/contracts/token/ERC1155/IERC1155.sol#4)
        - ^0.8.0 (@openzeppelin/contracts/token/ERC721/IERC721.sol#4)
        - ^0.8.0 (@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol#4)
        - ^0.8.0 (@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol#4)
        - ^0.8.1 (@openzeppelin/contracts/utils/Address.sol#4)
        - ^0.8.0 (@openzeppelin/contracts/utils/Context.sol#4)
        - ^0.8.0 (@openzeppelin/contracts/utils/Strings.sol#4)
        - ^0.8.0 (@openzeppelin/contracts/utils/introspection/ERC165.sol#4)
        - ^0.8.0 (@openzeppelin/contracts/utils/introspection/IERC165.sol#4)
        - 0.8.1 (ERC721Base.sol#2)
        - 0.8.1 (ERC721Min.sol#2)
        - 0.8.1 (NFTSales.sol#2)
        - 0.8.1 (Proxyable.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
```

```
INFO:Detectors:
ERC721Base.constructor(string,string).name (ERC721Base.sol#20) shadows:
        - ERC721Min.name() (ERC721Min.sol#138-140) (function)
        - IERC721Metadata.name() (@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol#16) (function)
ERC721Base.constructor(string,string).symbol (ERC721Base.sol#20) shadows:
        - ERC721Min.symbol() (ERC721Min.sol#145-147) (function)
        - IERC721Metadata.symbol() (@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol#21) (function)
NFTSales.constructor(string,string,uint256,uint256,uint256,uint256,string,address).name (NFTSales.sol#31) shadows:
        - ERC721Min.name() (ERC721Min.sol#138-140) (function)
        - IERC721Metadata.name() (@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol#16) (function)
NFTSales.constructor(string,string,uint256,uint256,uint256,uint256,string,address).symbol (NFTSales.sol#32) shadows:
        - ERC721Min.symbol() (ERC721Min.sol#145-147) (function)
        - IERC721Metadata.symbol() (@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol#21) (function)
NFTSales.isApprovedForAll(address,address)._owner (NFTSales.sol#148) shadows:
        - Ownable._owner (@openzeppelin/contracts/access/Ownable.sol#21) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
NFTSales.constructor(string,string,uint256,uint256,uint256,uint256,string,address)._treasury (NFTSales.sol#38) lacks a zero-check on :
                - treasury = _treasury (NFTSales.sol#44)
NFTSales.setTreasury(address)._treasury (NFTSales.sol#109) lacks a zero-check on :
                - treasury = _treasury (NFTSales.sol#114)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Modifier NFTSales.saleIsActive() (NFTSales.sol#50-53) does not always execute _; or revertReference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-modifier
INFO:Detectors:
Variable 'ERC721Min._checkOnERC721Received(address,address,uint256,bytes).retval (ERC721Min.sol#501)' in ERC721Min._checkOnERC721Received(address,address,uint256,bytes) (ERC721Min.sol#487-517) potentially used before declaration: retval == IERC721Receiver.onERC721Received.selector (ERC721Min.sol#502)
Variable 'ERC721Min._checkOnERC721Received(address,address,uint256,bytes).reason (ERC721Min.sol#503)' in ERC721Min._checkOnERC721Received(address,address,uint256,bytes) (ERC721Min.sol#487-517) potentially used before declaration: reason.length == 0 (ERC721Min.sol#504)
Variable 'ERC721Min._checkOnERC721Received(address,address,uint256,bytes).reason (ERC721Min.sol#503)' in ERC721Min._checkOnERC721Received(address,address,uint256,bytes) (ERC721Min.sol#487-517) potentially used before declaration: revert(uint256,uint256)(32 + reason,mload(uint256)(reason)) (ERC721Min.sol#510)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables
```

Solhint

```
======SOLHINT======


NFTSales.sol
    2:1   error    Compiler version 0.8.9 does not satisfy the ^0.5.8 semver requirement                compiler-version
   30:5   warning  Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)  func-visibility
   48:42  warning  Code contains empty blocks                                                           no-empty-blocks
  143:28  warning  Avoid to use low level calls                                                         avoid-low-level-calls

✖ 4 problems (1 error, 3 warnings)


======TYPOS Check======

1/1 ./NFTSales.sol 430.86ms X
/Users/enderphan/Quilhash/stronghands_nft-main/hardhat/contracts/NFTSales.sol:9:11 - Unknown word (Proxyable)
/Users/enderphan/Quilhash/stronghands_nft-main/hardhat/contracts/NFTSales.sol:11:31 - Unknown word (Reentrancy)
/Users/enderphan/Quilhash/stronghands_nft-main/hardhat/contracts/NFTSales.sol:11:48 - Unknown word (Proxyable)
CSpell: Files checked: 1, Issues found: 3 in 1 files

======uint Check======

No uint found

======require() Check======

0 require() without error message have been found
```

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.

Closing Summary

In this report, we have considered the security of the Stronghands smart contracts. We performed our audit according to the procedure described above.

A low severity issue was found, some suggestions and best practices are also provided in order to improve the code quality and security posture. In the End , Stronghands team acknowledged one low issue as it has no impact on security.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the Stronghands 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 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.

500+
Audits Completed

$15B
Secured

500K
Lines of Code Audited

Follow Our Journey

Audit Report
July, 2022

For

STRONGHANDS
FINANCE

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com