QuillAudits

Audit Report
January 2023

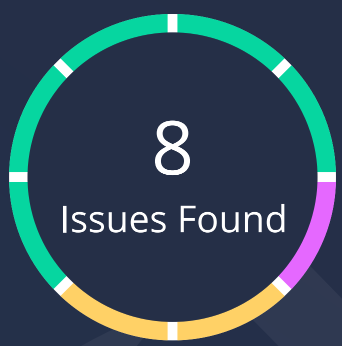
For

GASH

Table of Content

Executive Summary

Project Name	Gashpoint Staking
Timeline	20th Dec, 2022 to 6th Jan, 2023
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyze and document the Gashpoint Staking smart contract codebase for quality, security, and correctness.
Code Base	*https://bitbucket.org/bugcontract/stakecontract/src/master/f0f3d9040b68f82fc9ed8a84710395ce850b0c55* Initial Commit hash: ce14ec659eb91172be27162178cb353909c9dc4b
Fixed In	*https://bitbucket.org/bugcontract/stakecontract/commits/41bdc1f6b9bc151d8df380de87942ef8841a8443*



High
Medium
Low
Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	1	3	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	1	2	1

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

High Severity Issues

No issues found

Medium Severity Issues

1. Centralization Related Risks

Description

Any compromise to the owner's privileged accounts may allow a hacker to take advantage of this authority and manipulate the Gashpoint Staking contract.
Most of the function is controlled by the owner and it's given the right to mint any amount of token at any time.

Remediation

The risk describes the current project design and potentially makes iterations to improve the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the GashPoint Team to carefully manage the privileged account's private key to avoid any potential risks of being hacked.

In general, we strongly recommend centralized privileges or roles in the GashPoint staking contract be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Status

Acknowledged

2. DDOS attack in stakeNFT and Withdraw and some getters

Description

When the for loop is run for a bigger range of values in the stakeNFT, withdraw and some getters then it raises a scenario of DDOS to the system.
When the gas prices are high then the lower range/realistic range of minting and transfer will fail.

Remediation

Compute the current gas price when the stakeNFT and withdraw calls are made and the range of for loop should be handled accordingly.
This could be done with the GASPRICE opcode proposed in EIP-1559. Until EIP-1559 is implemented, it is not straightforward to compute the current gas price without an external oracle such as ETH Gas Station. However, such oracles could be DDoSed as we have seen on Feb 23rd, 2021.

Status

Resolved

Low Severity Issues

3. SafeTransfer Methods missing

Description

It is good to add a require() statement that checks the return value of token transfers or to use something like a safeTransfer/safeTransferFrom unless one is sure the given token reverts in case of a failure. Failure to do so will cause silent failures of transfers and affect token accounting in the contract.

Remediation

Replace transferFrom() with safeTransferFrom() since rewardToken can be any ERC20 token implementation. If transferFrom() does not return a value (e.g., USDT), the transaction reverts because of a decoding error. Revert without error.

Status

Resolved

4. Inherited OwnableUpgradeable uses single-step ownership transfer

Description

During the code review, It has been noticed that the BuggedStake contracts use single-step ownership transfer on the OwnableUpgradeable contract.

```
import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
...
import "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";
...
import "@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol";
...
import "@openzeppelin/contracts-upgradeable/utils/math/SafeMathUpgradeable.sol";
...
import "./bsc-library/contracts/IBEP20.sol";
...
import "./bsc-library/contracts/SafeBEP20.sol";
...
import "@openzeppelin/contracts-upgradeable/token/ERC721/IERC721Upgradeable.sol";
...
import "@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol";
        MilkLai, 4 weeks ago • initial
UnitTest stub | dependencies | uml | draw.io | ...
contract BuggedSTAKE is
    Initializable,
```

Remediation

Consider using the *Ownable2StepUpgradable* contract in the implementation.

Status

Acknowledged

5. Avoid leaving a contract uninitialized

Description

In the OpenZeppelin contracts, an uninitialized contract can be taken over by an attacker. To prevent the implementation contract from being used, the constructor can invoke the _disableInitializers function to automatically lock it when it is deployed.

Remediation

Consider using *_disableInitializers* function in the constructor.

Status

Resolved

6. Divide before multiple is followed

Description

At various places in the BuggedStake contract division before multiple is followed. By this there can be loss in data and the limit of smaller type can make it dangerous.

Remediation

In general, it's usually a good idea to re-arrange arithmetic to perform multiplication before division. So we recommend the team follow the same. *Reference*.

Status

Acknowledged

7. Used locked pragma version

Description

The pragma versions used in the contract are not locked. Consider
using the latest versions among 0.8.11 for deploying the contracts and libraries as it does not compile for any other version and can be confusing for a developer. Solidity source files indicate the versions of the compiler they can be compiled with.

pragma solidity ^0.8.0; // bad: compiles between 0.8.0 and 0.8.11
pragma solidity 0.8.0; // good: compiles w 0.8.0 only but not the latest version
pragma solidity 0.8.11; // best: compiles w 0.8.11

Remediation

Use best compiles and locked pragma in the contracts.

Status

Resolved

Informational Issues

8. Recommendations and Gas optimizations

1. For test stake use *smock* Library.

2. Gas optimization

- safeMath wrapper should be removed as it’s been inbuilt from solidity version 0.8 onwards.
- The pre-increment operation is cheaper (about 5 GAS per iteration) so use ++i instead of i++ or i+= 1 in for loop. We recommend using pre-increment in all the for loops.
- In for loop the default value initialization to 0 should not be there remove from all the for loop
- != 0 costs 6 less GAS compared to > 0 for unsigned integers in require statements with the optimizer enabled. We recommend using !=0 instead of > 0 in all the contracts.
- In the EVM, there is no opcode for non-strict inequalities (>=, <=) and two operations are performed (> + =.) Consider replacing >= with the strict counterpart >. Recommend following the inequality with a strict one.
- All the public functions which are not used internally need to be converted to external.

3. When the state gets updated the event should always get fired.
   For eg: setAddress, setRewardOwner should emit events. We recommend emitting the events for all the state changes.

4. In Function isAdmin should directly return the or logic of hasRole(DEFAULT_ADMIN_ROLE, account) and hasRole(MULTI_SIG_ROLE, account)

Status

Resolved

Automated Tests

```
ethsec@68ba21596f56:/code/contracts$ slither Staking_Flatten.sol
Compilation warnings/errors on Staking_Flatten.sol:
Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a comment containing "SPDX-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifier: UNL
ICENSED" for non-open-source code. Please see https://spdx.org for more information.
--> Staking_Flatten.sol



AccessControlUpgradeable.__gap (Staking_Flatten.sol#1402) shadows:
        - ERC165Upgradeable.__gap (Staking_Flatten.sol#771)
        - ContextUpgradeable.__gap (Staking_Flatten.sol#899)
OwnableUpgradeable.__gap (Staking_Flatten.sol#1656) shadows:
        - ContextUpgradeable.__gap (Staking_Flatten.sol#899)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variable-shadowing

BuggedSTAKE.pendingReward(address) (Staking_Flatten.sol#1828-1847) performs a multiplication on the result of a division:
        -dayPassed = ((block.timestamp.sub(user.depositTimestamp)).div(86400)).mul(10 ** 18).div(10 ** 18) (Staking_Flatten.sol#1833-1835)
BuggedSTAKE.pendingReward(address) (Staking_Flatten.sol#1828-1847) performs a multiplication on the result of a division:
        -dayPassed = ((block.timestamp.sub(user.depositTimestamp)).div(86400)).mul(10 ** 18).div(10 ** 18) (Staking_Flatten.sol#1833-1835)
        -baseProfit = dayPassed.mul(depositTokenCount).mul(APR).mul(baseNFTPrice).mul(1e6) (Staking_Flatten.sol#1840-1844)
BuggedSTAKE.withdraw() (Staking_Flatten.sol#1880-1929) performs a multiplication on the result of a division:
        -dayPassed = ((block.timestamp.sub(user.depositTimestamp)).div(86400)).mul(10 ** 18).div(10 ** 18) (Staking_Flatten.sol#1887-1889)
BuggedSTAKE.withdraw() (Staking_Flatten.sol#1880-1929) performs a multiplication on the result of a division:
        -dayPassed = ((block.timestamp.sub(user.depositTimestamp)).div(86400)).mul(10 ** 18).div(10 ** 18) (Staking_Flatten.sol#1887-1889)
        -baseProfit = dayPassed.mul(withdrawTokenIds.length).mul(APR).mul(baseNFTPrice).mul(1e6) (Staking_Flatten.sol#1894-1898)
BuggedSTAKE.claim() (Staking_Flatten.sol#1932-1968) performs a multiplication on the result of a division:
        -dayPassed = ((block.timestamp.sub(user.depositTimestamp)).div(86400)).mul(10 ** 18).div(10 ** 18) (Staking_Flatten.sol#1940-1942)
BuggedSTAKE.claim() (Staking_Flatten.sol#1932-1968) performs a multiplication on the result of a division:
        -dayPassed = ((block.timestamp.sub(user.depositTimestamp)).div(86400)).mul(10 ** 18).div(10 ** 18) (Staking_Flatten.sol#1940-1942)
        -baseProfit = dayPassed.mul(withdrawTokenIds.length).mul(APR).mul(baseNFTPrice).mul(1e6) (Staking_Flatten.sol#1946-1950)
BuggedSTAKE.getRewardRate(uint256) (Staking_Flatten.sol#1975-1987) performs a multiplication on the result of a division:
        -dayPassed = ((block.timestamp.sub(fromBlockTS)).div(86400)).mul(10 ** 18).div(10 ** 18) (Staking_Flatten.sol#1979-1981)
BuggedSTAKE.getRewardRate(uint256) (Staking_Flatten.sol#1975-1987) performs a multiplication on the result of a division:
        -dayPassed = ((block.timestamp.sub(fromBlockTS)).div(86400)).mul(10 ** 18).div(10 ** 18) (Staking_Flatten.sol#1979-1981)
        -rewardRate = dayPassed.mul(3) (Staking_Flatten.sol#1984)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

BuggedSTAKE.withdraw() (Staking_Flatten.sol#1880-1929) uses a dangerous strict equality:
        - require(bool)(stakingToken.ownerOf(withdrawTokenIds[i]) == msg.sender) (Staking_Flatten.sol#1927)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

BuggedSTAKE.setBaseNFTPrice(uint256) (Staking_Flatten.sol#1789-1794) should emit an event for:
        - baseNFTPrice = _baseNFTPrice (Staking_Flatten.sol#1793)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

BuggedSTAKE.stakeNFT(uint256[]) (Staking_Flatten.sol#1851-1876) has external calls inside a loop: stakingToken.transferFrom(msg.sender,address(this),depositTokenIDs[i]) (Staking_Flatten.sol#1868-1872)
BuggedSTAKE.stakeNFT(uint256[]) (Staking_Flatten.sol#1851-1876) has external calls inside a loop: require(bool)(stakingToken.ownerOf(depositTokenIDs[i]) == address(this)) (Staking_Flatten.sol#1873)
BuggedSTAKE.withdraw() (Staking_Flatten.sol#1880-1929) has external calls inside a loop: stakingToken.transferFrom(address(this),msg.sender,withdrawTokenIds[i]) (Staking_Flatten.sol#1922-1926)
BuggedSTAKE.withdraw() (Staking_Flatten.sol#1880-1929) has external calls inside a loop: require(bool)(stakingToken.ownerOf(withdrawTokenIds[i]) == msg.sender) (Staking_Flatten.sol#1927)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

BuggedSTAKE.pendingReward(address) (Staking_Flatten.sol#1828-1847) uses timestamp for comparisons
        Dangerous comparisons:
        - dayPassed > maxRewardDay (Staking_Flatten.sol#1837)
BuggedSTAKE.stakeNFT(uint256[]) (Staking_Flatten.sol#1851-1876) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp >= stakingStart,Not open yet) (Staking_Flatten.sol#1853)
        - require(bool,string)(user.depositTimestamp == 0,Staking) (Staking_Flatten.sol#1858)
        - require(bool,string)(getStakingTokenIds(msg.sender).length < stakingTokenLimit,exceed the top) (Staking_Flatten.sol#1860-1863)
BuggedSTAKE.withdraw() (Staking_Flatten.sol#1880-1929) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(withdrawTokenIds.length > 0,Non stake) (Staking_Flatten.sol#1884)
        - dayPassed > maxRewardDay (Staking_Flatten.sol#1891)
```

```
BuggedSTAKE.stakeNFT(uint256[]) (Staking_Flatten.sol#1851-1876) uses timestamp for comparisons
	Dangerous comparisons:
	- require(bool,string)(block.timestamp >= stakingStart,Not open yet) (Staking_Flatten.sol#1853)
	- require(bool,string)(user.depositTimestamp == 0,Staking) (Staking_Flatten.sol#1858)
	- require(bool,string)(getStakingTokenIds(msg.sender).length < stakingTokenLimit,exceed the top) (Staking_Flatten.sol#1860-1863)
BuggedSTAKE.withdraw() (Staking_Flatten.sol#1880-1929) uses timestamp for comparisons
	Dangerous comparisons:
	- require(bool,string)(withdrawTokenIds.length > 0,Non stake) (Staking_Flatten.sol#1884)
	- dayPassed > maxRewardDay (Staking_Flatten.sol#1891)
	- reward > 0 (Staking_Flatten.sol#1908)
	- require(bool,string)(rewardTokenTransferFromCheck,RewardToken TransferFrom error) (Staking_Flatten.sol#1914-1917)
	- i < withdrawTokenIds.length (Staking_Flatten.sol#1921)
	- require(bool)(stakingToken.ownerOf(withdrawTokenIds[i]) == msg.sender) (Staking_Flatten.sol#1927)
BuggedSTAKE.claim() (Staking_Flatten.sol#1932-1968) uses timestamp for comparisons
	Dangerous comparisons:
	- require(bool,string)(withdrawTokenIds.length > 0,Non stake) (Staking_Flatten.sol#1936)
	- dayPassed > maxRewardDay (Staking_Flatten.sol#1943)
	- reward > 0 (Staking_Flatten.sol#1957)
	- require(bool,string)(rewardTokenTransferFromCheck,RewardToken TransferFrom error) (Staking_Flatten.sol#1963-1966)
BuggedSTAKE.getRewardRate(uint256) (Staking_Flatten.sol#1975-1987) uses timestamp for comparisons
	Dangerous comparisons:
	- block.timestamp < stakingStart (Staking_Flatten.sol#1976)
	- dayPassed < 33 (Staking_Flatten.sol#1983)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

AddressUpgradeable.verifyCallResult(bool,bytes,string) (Staking_Flatten.sol#414-434) uses assembly
	- INLINE ASM (Staking_Flatten.sol#426-429)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Different versions of Solidity is used:
	- Version used: ['>=0.4.0', '^0.8.0', '^0.8.1', '^0.8.2']
	- ^0.8.0 (Staking_Flatten.sol#9)
	- ^0.8.1 (Staking_Flatten.sol#244)
	- >=0.4.0 (Staking_Flatten.sol#446)
	- ^0.8.2 (Staking_Flatten.sol#555)
	- ^0.8.0 (Staking_Flatten.sol#701)
	- ^0.8.0 (Staking_Flatten.sol#734)
	- ^0.8.0 (Staking_Flatten.sol#784)
	- ^0.8.0 (Staking_Flatten.sol#867)
	- ^0.8.0 (Staking_Flatten.sol#912)
	- ^0.8.0 (Staking_Flatten.sol#1006)
	- ^0.8.0 (Staking_Flatten.sol#1147)
	- ^0.8.0 (Staking_Flatten.sol#1415)
	- ^0.8.0 (Staking_Flatten.sol#1566)
	- ^0.8.0 (Staking_Flatten.sol#1665)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

AccessControlUpgradeable.__AccessControl_init() (Staking_Flatten.sol#1194-1195) is never used and should be removed
AccessControlUpgradeable.__AccessControl_init_unchained() (Staking_Flatten.sol#1197-1198) is never used and should be removed
AccessControlUpgradeable._setRoleAdmin(bytes32,bytes32) (Staking_Flatten.sol#1363-1367) is never used and should be removed
AddressUpgradeable.functionCall(address,bytes) (Staking_Flatten.sol#325-327) is never used and should be removed
AddressUpgradeable.functionCall(address,bytes,string) (Staking_Flatten.sol#335-341) is never used and should be removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256) (Staking_Flatten.sol#354-360) is never used and should be removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (Staking_Flatten.sol#368-379) is never used and should be removed
AddressUpgradeable.functionStaticCall(address,bytes) (Staking_Flatten.sol#387-389) is never used and should be removed
AddressUpgradeable.functionStaticCall(address,bytes,string) (Staking_Flatten.sol#397-406) is never used and should be removed
AddressUpgradeable.sendValue(address,uint256) (Staking_Flatten.sol#300-305) is never used and should be removed
AddressUpgradeable.verifyCallResult(bool,bytes,string) (Staking_Flatten.sol#414-434) is never used and should be removed
ContextUpgradeable.__Context_init() (Staking_Flatten.sol#881-882) is never used and should be removed
ContextUpgradeable.__Context_init_unchained() (Staking_Flatten.sol#884-885) is never used and should be removed
ContextUpgradeable._msgData() (Staking_Flatten.sol#890-892) is never used and should be removed
ERC165Upgradeable.__ERC165_init() (Staking_Flatten.sol#754-755) is never used and should be removed
```

```
ERC165Upgradeable.__ERC165_init_unchained() (Staking_Flatten.sol#757-758) is never used and should be removed
Initializable._disableInitializers() (Staking_Flatten.sol#682-688) is never used and should be removed
SafeBEP20._callOptionalReturn(IBEP20,bytes) (Staking_Flatten.sol#1117-1134) is never used and should be removed
SafeBEP20.safeApprove(IBEP20,address,uint256) (Staking_Flatten.sol#1055-1072) is never used and should be removed
SafeBEP20.safeDecreaseAllowance(IBEP20,address,uint256) (Staking_Flatten.sol#1092-1109) is never used and should be removed
SafeBEP20.safeIncreaseAllowance(IBEP20,address,uint256) (Staking_Flatten.sol#1074-1090) is never used and should be removed
SafeBEP20.safeTransfer(IBEP20,address,uint256) (Staking_Flatten.sol#1025-1034) is never used and should be removed
SafeBEP20.safeTransferFrom(IBEP20,address,address,uint256) (Staking_Flatten.sol#1036-1046) is never used and should be removed
SafeMathUpgradeable.add(uint256,uint256) (Staking_Flatten.sol#98-100) is never used and should be removed
SafeMathUpgradeable.div(uint256,uint256,string) (Staking_Flatten.sol#196-205) is never used and should be removed
SafeMathUpgradeable.mod(uint256,uint256) (Staking_Flatten.sol#156-158) is never used and should be removed
SafeMathUpgradeable.mod(uint256,uint256,string) (Staking_Flatten.sol#222-231) is never used and should be removed
SafeMathUpgradeable.sub(uint256,uint256,string) (Staking_Flatten.sol#173-182) is never used and should be removed
SafeMathUpgradeable.tryAdd(uint256,uint256) (Staking_Flatten.sol#27-33) is never used and should be removed
SafeMathUpgradeable.tryDiv(uint256,uint256) (Staking_Flatten.sol#69-74) is never used and should be removed
SafeMathUpgradeable.tryMod(uint256,uint256) (Staking_Flatten.sol#81-86) is never used and should be removed
SafeMathUpgradeable.tryMul(uint256,uint256) (Staking_Flatten.sol#52-62) is never used and should be removed
SafeMathUpgradeable.trySub(uint256,uint256) (Staking_Flatten.sol#40-45) is never used and should be removed
StringsUpgradeable.toHexString(address) (Staking_Flatten.sol#852-854) is never used and should be removed
StringsUpgradeable.toHexString(uint256) (Staking_Flatten.sol#821-832) is never used and should be removed
StringsUpgradeable.toString(uint256) (Staking_Flatten.sol#796-816) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (Staking_Flatten.sol#9) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.1 (Staking_Flatten.sol#244) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version>=0.4.0 (Staking_Flatten.sol#446) allows old versions
Pragma version^0.8.2 (Staking_Flatten.sol#555) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (Staking_Flatten.sol#701) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (Staking_Flatten.sol#734) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (Staking_Flatten.sol#784) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (Staking_Flatten.sol#867) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (Staking_Flatten.sol#912) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (Staking_Flatten.sol#1006) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (Staking_Flatten.sol#1147) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (Staking_Flatten.sol#1415) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (Staking_Flatten.sol#1566) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (Staking_Flatten.sol#1665) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in AddressUpgradeable.sendValue(address,uint256) (Staking_Flatten.sol#300-305):
	- (success) = recipient.call{value: amount}() (Staking_Flatten.sol#303)
Low level call in AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (Staking_Flatten.sol#368-379):
	- (success,returndata) = target.call{value: value}(data) (Staking_Flatten.sol#377)
Low level call in AddressUpgradeable.functionStaticCall(address,bytes,string) (Staking_Flatten.sol#397-406):
	- (success,returndata) = target.staticcall(data) (Staking_Flatten.sol#404)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Function ERC165Upgradeable.__ERC165_init() (Staking_Flatten.sol#754-755) is not in mixedCase
Function ERC165Upgradeable.__ERC165_init_unchained() (Staking_Flatten.sol#757-758) is not in mixedCase
Variable ERC165Upgradeable.__gap (Staking_Flatten.sol#771) is not in mixedCase
Function ContextUpgradeable.__Context_init() (Staking_Flatten.sol#881-882) is not in mixedCase
Function ContextUpgradeable.__Context_init_unchained() (Staking_Flatten.sol#884-885) is not in mixedCase
Variable ContextUpgradeable.__gap (Staking_Flatten.sol#899) is not in mixedCase
Function AccessControlUpgradeable.__AccessControl_init() (Staking_Flatten.sol#1194-1195) is not in mixedCase
Function AccessControlUpgradeable.__AccessControl_init_unchained() (Staking_Flatten.sol#1197-1198) is not in mixedCase
Variable AccessControlUpgradeable.__gap (Staking_Flatten.sol#1402) is not in mixedCase
Function OwnableUpgradeable.__Ownable_init() (Staking_Flatten.sol#1591-1593) is not in mixedCase
Function OwnableUpgradeable.__Ownable_init_unchained() (Staking_Flatten.sol#1595-1597) is not in mixedCase
Variable OwnableUpgradeable.__gap (Staking_Flatten.sol#1656) is not in mixedCase
Parameter BuggedSTAKE.setAddress(address,address)._stakingTokenAddress (Staking_Flatten.sol#1777) is not in mixedCase
Parameter BuggedSTAKE.setAddress(address,address)._rewardAddress (Staking_Flatten.sol#1777) is not in mixedCase
Parameter BuggedSTAKE.setBaseNFTPrice(uint256)._baseNFTPrice (Staking_Flatten.sol#1789) is not in mixedCase
```

```
Function OwnableUpgradeable.__Ownable_init_unchained() (Staking_Flatten.sol#1595-1597) is not in mixedCase
Variable OwnableUpgradeable.__gap (Staking_Flatten.sol#1656) is not in mixedCase
Parameter BuggedSTAKE.setAddress(address,address)._stakingTokenAddress (Staking_Flatten.sol#1777) is not in mixedCase
Parameter BuggedSTAKE.setAddress(address,address)._rewardAddress (Staking_Flatten.sol#1777) is not in mixedCase
Parameter BuggedSTAKE.setBaseNFTPrice(uint256)._baseNFTPrice (Staking_Flatten.sol#1789) is not in mixedCase
Parameter BuggedSTAKE.setOpen(bool)._isOpen (Staking_Flatten.sol#1797) is not in mixedCase
Parameter BuggedSTAKE.setStakePeriod(uint256,uint256)._start (Staking_Flatten.sol#1803) is not in mixedCase
Parameter BuggedSTAKE.setStakePeriod(uint256,uint256)._day (Staking_Flatten.sol#1803) is not in mixedCase
Parameter BuggedSTAKE.setAPR(uint256)._apr (Staking_Flatten.sol#1810) is not in mixedCase
Parameter BuggedSTAKE.setRewardOwner(address)._address (Staking_Flatten.sol#1816) is not in mixedCase
Parameter BuggedSTAKE.setStakeLimit(uint256)._amount (Staking_Flatten.sol#1822) is not in mixedCase
Parameter BuggedSTAKE.pendingReward(address)._user (Staking_Flatten.sol#1828) is not in mixedCase
Parameter BuggedSTAKE.getStakingTokenIds(address)._address (Staking_Flatten.sol#1990) is not in mixedCase
Parameter BuggedSTAKE.getStakingTokenCount(address)._address (Staking_Flatten.sol#1999) is not in mixedCase
Parameter BuggedSTAKE.getUserInfo(address)._address (Staking_Flatten.sol#2008) is not in mixedCase
Parameter BuggedSTAKE.getBatchUserInfo(address[])._addressArray (Staking_Flatten.sol#2017) is not in mixedCase
Parameter BuggedSTAKE.getBatchPendingRewards(address[])._addressArray (Staking_Flatten.sol#2033) is not in mixedCase
Variable BuggedSTAKE.APR (Staking_Flatten.sol#1708) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

AccessControlUpgradeable.__gap (Staking_Flatten.sol#1402) is never used in BuggedSTAKE (Staking_Flatten.sol#1676-2089)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable

revokeRole(bytes32,address) should be declared external:
        - AccessControlUpgradeable.revokeRole(bytes32,address) (Staking_Flatten.sol#1308-1310)
renounceOwnership() should be declared external:
        - OwnableUpgradeable.renounceOwnership() (Staking_Flatten.sol#1628-1630)
transferOwnership(address) should be declared external:
        - BuggedSTAKE.transferOwnership(address) (Staking_Flatten.sol#2071-2088)
        - OwnableUpgradeable.transferOwnership(address) (Staking_Flatten.sol#1636-1639)
setAddress(address,address) should be declared external:
        - BuggedSTAKE.setAddress(address,address) (Staking_Flatten.sol#1777-1787)
setBaseNFTPrice(uint256) should be declared external:
        - BuggedSTAKE.setBaseNFTPrice(uint256) (Staking_Flatten.sol#1789-1794)
setOpen(bool) should be declared external:
        - BuggedSTAKE.setOpen(bool) (Staking_Flatten.sol#1797-1800)
setStakePeriod(uint256,uint256) should be declared external:
        - BuggedSTAKE.setStakePeriod(uint256,uint256) (Staking_Flatten.sol#1803-1807)
setAPR(uint256) should be declared external:
        - BuggedSTAKE.setAPR(uint256) (Staking_Flatten.sol#1810-1813)
setRewardOwner(address) should be declared external:
        - BuggedSTAKE.setRewardOwner(address) (Staking_Flatten.sol#1816-1819)
setStakeLimit(uint256) should be declared external:
        - BuggedSTAKE.setStakeLimit(uint256) (Staking_Flatten.sol#1822-1825)
stakeNFT(uint256[]) should be declared external:
        - BuggedSTAKE.stakeNFT(uint256[]) (Staking_Flatten.sol#1851-1876)
withdraw() should be declared external:
        - BuggedSTAKE.withdraw() (Staking_Flatten.sol#1880-1929)
claim() should be declared external:
        - BuggedSTAKE.claim() (Staking_Flatten.sol#1932-1968)
getUserInfo(address) should be declared external:
        - BuggedSTAKE.getUserInfo(address) (Staking_Flatten.sol#2008-2014)
getBatchUserInfo(address[]) should be declared external:
        - BuggedSTAKE.getBatchUserInfo(address[]) (Staking_Flatten.sol#2017-2030)
getBatchPendingRewards(address[]) should be declared external:
        - BuggedSTAKE.getBatchPendingRewards(address[]) (Staking_Flatten.sol#2033-2045)
addMultiSigMember(address) should be declared external:
        - BuggedSTAKE.addMultiSigMember(address) (Staking_Flatten.sol#2049-2051)
renounceMultiSigRole(address) should be declared external:
        - BuggedSTAKE.renounceMultiSigRole(address) (Staking_Flatten.sol#2054-2056)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
Staking_Flatten.sol analyzed (14 contracts with 75 detectors), 124 result(s) found
ethsec@68ba21596f56:/code/contracts$ 
```

Results

There were 75 results uncovered via Slither for the Gashpoint Staking contract, and we checked through all of them and found them to be false positives.

Closing Summary

In this report, we have considered the security of the Gashpoint Staking platform. We performed our audit according to the procedure described above.

2 medium, 5 low, and 1 informational issue are found in the Initial audit . In the end Gashpoint Resolved few issues and Acknowledged others.

Disclaimer

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

700+
Audits Completed

$16B
Secured

700K
Lines of Code Audited

Follow Our Journey

Audit Report
January, 2023

For

GASH

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com