QuillAudits

Audit Report
March, 2022

For

STAKEALL

Contents

Contents

Overview

Stakeall Finance

Stakeall Finance is a fully decentralized non custodial DeFi platform focused on building DeFi strategies. DeFi strategies are basically a series of DeFi actions wrapped in a single transaction. These innovative DeFi strategies open a new gateway for users to participate in DeFi with better returns.

Scope of the Audit

The scope of this audit was to analyze Stakeall smart contract's codebase for quality, security, and correctness.

Stakeall Contract deployed at:
Codebase: c266b2e40ee100635175b935676ac64a96695622

FixedIn

Commit: 51ce0fd0273f1413622646de04268173cd9361c0

Deployed Contract Address:-
GraphProtocolStaking: https://etherscan.io/address/0x692f78A81A1BFb2A2FEa6c403Fd8776036fD874D

MaticProtocolStaking: https://etherscan.io/address/0xae008db8874246daC20B4853c0318AAa80ffe7D1

StakeAllImplementationM2:
https://etherscan.io/address/0x6F5D9B1cD26ae1200875d660ed457BDBca74aF5F

Basic Connector:
https://etherscan.io/address/0x81E79DfF0625DD2b945B09102875715814B46eD8

Connext:
https://polygonscan.com/address/0xB893a51fed5A65954440dc872D62692b6299d940

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- 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 level

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
Mythril, Slither, SmartCheck, Surya, Solhint.

Issue Categories

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

Risk-level	Description
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.

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	0	0	2	1
Closed	0	1	3	3

Issues Found – Code Review/Manual Testing

A. GraphProtocol Staking Contract

High severity issues

No issues found

Medium severity issues

1. **Missing zero check**

```
function delegateMultiple(
    address[] memory indexers,
    uint256 amount,
    uint256[] memory portions,
    uint256 getId
) external payable {
    require(
        portions.length == indexers.length,
        "Indexer and Portion length doesnt match"
    );
    uint256 delegationAmount = getUint(getId, amount);
    uint256 totalPortions = 0;
```

```
 */
function undelegateMultiple(
    address[] memory _indexers,
    uint256[] memory _shares
) external payable {
    require(
        _indexers.length == _shares.length,
        "Indexers & shares mismatch"
    );
    for (uint256 i = 0; i < _indexers.length; i++) {
        graphProxy.undelegate(_indexers[i], _shares[i]);
    }
```

```
function withdrawMultipleDelegate(
    address[] memory _indexers,
    address[] memory _delegateToIndexers
) external payable {
    for (uint256 i = 0; i < _indexers.length; i++) {
        graphProxy.withdrawDelegated(_indexers[i], _delegateToIndexers[i]);
    }
```

Line - 45, 94, 126
Function - delegateMultiple, undelegateMultiple, withdrawMultipleDelegate

Description
Missing zero address check for indexers, _indexers parameters that may lead to the delegation of amount to a zero address.

Remediation
Add a require check for the same.

Status: Fixed

Low severity issues

2. **Race Condition**

```
grtTokenAddress.approve(address(graphProxy), delegationAmount);
```

```
grtTokenAddress.approve(address(graphProxy), delegationAmount);
```

Line - 33, 69
Function - delegate, delegateMultiplener.

Description
ERC20 approve race condition in function delegate() and delegateMultiple().

Remediation
Use increaseAllowance instead. You would need to extend the interface in order to use it.

Status: Acknowledged

3. **Out of Gas issue**

```
function delegateMultiple(
    address[] memory indexers,
    uint256 amount,
    uint256[] memory portions,
    uint256 getId
) external payable {
    require(
        portions.length == indexers.length,
        "Indexer and Portion length doesnt match"
    );
    uint256 delegationAmount = getUint(getId, amount);
    uint256 totalPortions = 0;
```

```
function withdrawMultipleDelegate(
    address[] memory _indexers,
    address[] memory _delegateToIndexers
) external payable {
    for (uint256 i = 0; i < _indexers.length; i++) {
        graphProxy.withdrawDelegated(_indexers[i], _delegateToIndexers[i]);
    }
```

Line - 45, 126
Function - delegateMultiple, withdrawMultipleDelegate

Description
There is no check in place to limit the number of indexers and portions allowed to delegate. Thus, a large number of indexers and portions at the same time may lead the contract to run out of gas during loop execution.

Remediation
Limit the number of indexers allowed to delegate at the same time

Status: Fixed

Informational issues

No issues found

B. MaticProtocol Staking Contract

High severity issues

No issues found

Medium severity issues

No issues found

Low severity issues

1. **Race Condition**

```
    maticToken.approve(address(stakeManagerProxy), delegationAmount);
```

```
maticToken.approve(address(stakeManagerProxy), delegationAmount);
```

Line - 34, 74
Function - delegate, delegateMultiple.

Description
ERC20 approve race condition in function delegate() and delegateMultiple().

Remediation
Use increaseAllowance instead. You would need to extend the interface in order to use it.

Status: Acknowledged

2. **Out of Gas issue**

Function - delegateMultiple[#47], withdrawRewardsMultiple[#113], sellVoucherMultiple[#162]

Description
There is no check in place to limit the number of indexers and portions allowed to delegate. If the array length of validatorAddresses and portions parameter is too large, it could lead to out of gas issues during for loop execution.

Remediation
Consider adding a require check for max array length allowed for validatorAddresses.

Status: Fixed

3. **Missing array length check**

Function - delegateMultiple[#47], withdrawRewardsMultiple[#113], sellVoucherMultiple[#162], unstakeClaimedTokensMultiple[#249]

Description
Missing array length check in validatorContractAddresses, validatorAddresses, minShares, and unbondNonces parameters. This can lead to incorrect assignments and state changes while invoking buyVoucher on the lines: 264, 82.

Remediation
Consider adding a require check for max array length

Status: Fixed

Informational issues

No issues found

C. Basic/main.sol Contract

High severity issues

No issues found

Medium severity issues

No issues found

Low severity issues

No issues found

Informational issues

1. **Unused Parameters(Dead Code)**

   Function - depositByAllowance(#64)

   Description

   In basic/main.sol in the depositByAllowance() there is a parameter named _eventParam which is not used / declared twice and should be removed

   Remediation

   We recommend removing unused parameters.

   Status: Fixed

D. Connext/main.sol Contract

High severity issues

No issues found

Medium severity issues

No issues found

Low severity issues

No issues found

Informational issues

1. **Unchecked return value of external calls**

   Function - crosschainTransfer(#30)

   Description

   Whenever an external call is made from the contract, it is always recommended to check the return value of the call. In connext/main.sol, there is an external call to prepare() function however those values aren't checked by the contract.

   Remediation

   We recommend checking the return value of each and every low level and external function call.

   Status: Acknowledged

E. Common Issues for Basic/main.sol, GraphProtocol/main.sol, MaticProtocol/main.sol, connext/main.sol, Accounts/module2/Implementation_m2.sol

High severity issues

No issues found

Medium severity issues

No issues found

Low severity issues

No issues found

Informational issues

1. **Unlocked pragma (pragma solidity ^0.7.6 and ^0.7.0)**

   Description

   Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

   Remediation

   Here all the in-scope contracts have an unlocked pragma, it is recommended to lock the same. Moreover, we strongly suggest not to use experimental Solidity features (e.g., pragma experimental ABIEncoderV2) or third-party unaudited libraries. If necessary, refactor the current code base to only use stable features.

   Status: Fixed

2. **State variables that could be declared constant**

   Contract: Basic/main.sol (name #), Graphprotocol.main.sol (name #), Connext/main.sol (name #L10)

   Description
   While auditing the contracts we found out that there are multiple instances where a state variable is used and is not updated. The “name” parameter in both contracts is not updated and must be declared constant.

   Remediation
   We recommend declaring such variables as constant to save gas.

   Status: Fixed

Functional Testing Results

Some of the tests performed are mentioned below:

- ☑ Should delegate(), DelegateMultiple() and successfully call other functions.
- ☑ Should revert if the user tries to sellVoucher without transferring tokens.
- ☑ Should be able to be call sellVoucherMultiple deposit/withdraw.
- ☑ Delegation to the locked validator must fail.
- ☑ Verify owner signature and only the owner must be able to call the cast.
- ☑ In withdrawDelegated function(), if "_delegateToIndexer" is 0, withdraw must happen, else delegation must happen.
- ☑ uneven portion distribution must fail.
- ☑ Check deposit and withdraw function in basic/main.sol.
- ☑ Amount must be directly considered if value in getuint is 0.
- ☑ castWithSignature must verify input arrays.
- ☑ Should call CrossChainTransfer()
- ☑ Transactions should not go Out of gas due to unchecked array length. (The team has fixed this issue on the UI level and chances of users encountering this issue are very slim)

Issues Identified Via Automation Testing

Slither

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

Mythril

No issues were reported by Mythril.

Closing Summary

Some issues of Medium, Low and informational severity were found, which are now fixed by the developers. Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of Stakeall Finance. 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 Stakeall team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Audit Report
March, 2022

For

STAKEALL

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com